QSM

QUEUED SERIAL MODULE

Reference Manual

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Paragraph **Title** **Page**

SECTION 5
SCI SUBMODULE

TABLE OF CONTENTS
(Continued)

Paragraph **Title** **Page**

APPENDIX A
USING THE QSPI FOR ANALOG DATA AQUISITION

APPENDIX B
QSM MEMORY MAP AND REGISTERS

INDEX

TABLE OF CONTENTS
(Continued)

Paragraph **Title** **Page**

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

Figure	Title	Page

LIST OF TABLES

LIST OF TABLES
(Continued)

Table	Title	Page

SECTION 1
FUNCTIONAL OVERVIEW

The queued serial module (QSM) provides the microcontroller unit (MCU) with two serial communication interfaces divided into two submodules: the queued serial peripheral interface (QSPI) and the serial communications interface (SCI).

The QSPI is a full-duplex, synchronous serial interface for communicating with peripherals and other MCUs. It is enhanced by the addition of a queue for receive and transmit data.

The SCI is a full-duplex universal asynchronous receiver transmitter (UART) serial interface. These submodules operate independently.

This section provides a block diagram, memory map, pin description, and register descriptions of the QSM, with a breakdown of both the QSPI and SCI submodules. Operation of the QSPI submodule includes master mode and slave mode. For a detailed description refer to **4.4.1 Master Mode** and **4.4.2 Slave Mode**.

In addition, operation of the SCI submodule is divided into transmit and receive. A description of these operations is given in **5.3 Transmitter Operation** and **5.4 Receiver Operation**. To aid in grasping an understanding of the numerous bits and fields of the registers that appear throughout the text, a quick reference guide identifies all bit/field acronyms. (Refer to **Table 3-2**.)

1.1 Block Diagram

Figure 1-1 depicts the major components of the QSM, which consist of the global registers, logic control, and the QSPI and SCI submodules. Refer to **SECTION 4 QSPI SUBMODULE** and **SECTION 5 SCI SUBMODULE** for further definition of these components.

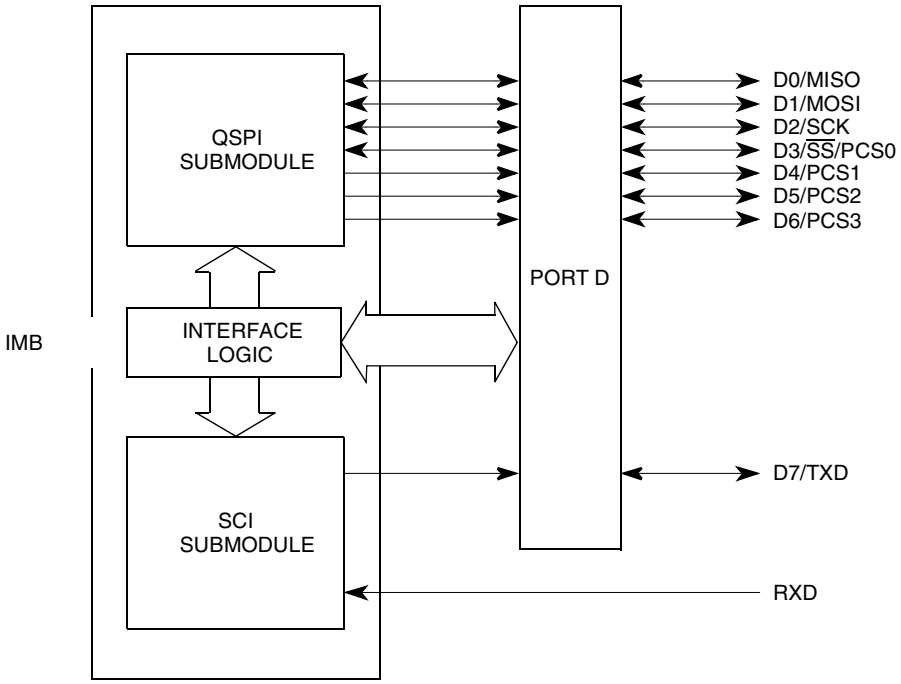


Figure 1-1 QSM Block Diagram

1.2 Memory Map

The QSM memory map is comprised of the global registers, the QSPI and SCI control and status registers, and the QSPI RAM as shown in **Figure 1-2**. For an accurate location of the QSM memory in the MCU memory map, refer to appropriate CPU manual. The QSM memory map may be divided into two segments: supervisor-only data space and assignable data space.

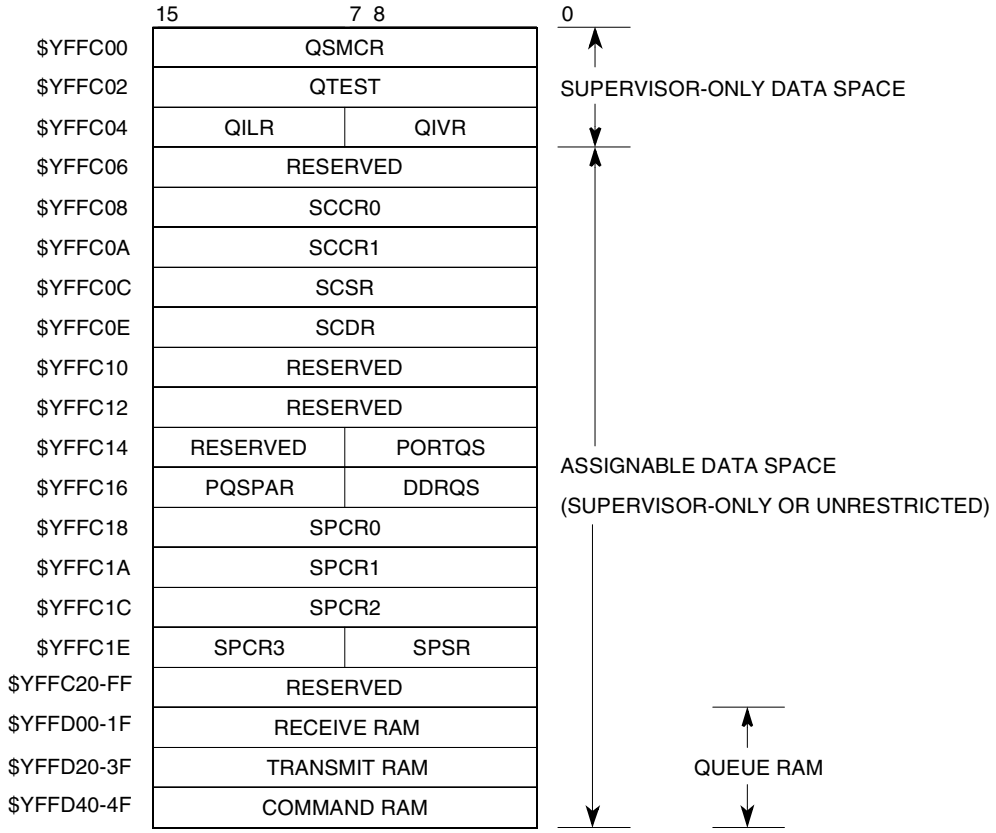


Figure 1-2 QSM Memory Map

The supervisor-only data space segment contains the QSM global registers. These registers define parameters needed by the QSM to integrate with the MCU. Access to these registers is permitted only when the CPU is operating in supervisor mode (CPU status register, S-bit = 1).

Assignable data space can be either restricted to supervisor-only access or unrestricted to both supervisor and user accesses. The supervisor (SUPV) bit in the QSM module configuration register (QSMCR) designates the assignable data space as either supervisor or unrestricted. If SUPV is set, then the space is designated as supervisor-only space. Access is then permitted only when the CPU is operating in supervisor mode. All attempts to read supervisor data spaces when not in supervisor mode (CPU status register, S-bit = 0) return a value of zero, and all attempts to write have no effect. If SUPV is clear, both user and supervisor accesses are permitted. To clear SUPV in the QSMCR, the CPU must be in supervisor mode (CPU status register, S-bit = 1). Refer to Processing States in the appropriate CPU manual for more information on supervisor mode.

The QSM assignable data space segment contains the submodules, QSPI and SCI, control/status registers, and the QSPI RAM. All registers and RAM may be accessed

on byte, word, and long-word boundaries. The 80 bytes of static RAM are distinct from the QSM register set. All bytes not used by the QSPI may be used as general-purpose RAM. When operating, the QSPI submodule uses three non-contiguous blocks of the 80-byte RAM for receive, transmit, and control data. More information on the QSPI RAM can be found in **4.3.6 QSPI RAM**.

The contents of most locations in the memory map may be rewritten with the identical value to that location, with one exception. (Refer to **4.3.3 QSPI Control Register 2 (SPCR2)**.) Writing a different value to certain control registers when a submodule using that register is enabled can cause unpredictable results. For predictable operation, if register bits are to be changed, the CPU should disable the submodule in an orderly fashion before altering the registers.

SECTION 2
SIGNAL DESCRIPTIONS

The QSM has nine external pins, as shown in **Figure 1-1**. Eight of the pins, if not in use for their submodule function, can be used as general-purpose I/O port pins. The ninth pin, RXD, is an input-only pin used exclusively by the SCI submodule.

The QSM pin control registers — DDRQS, QSM pin assignment register (PQSPAR, and QSM port data register (PORTQS) — affect pins being used as general-purpose I/O pins. The QSPI control register 0 (SPCR0) has one bit that affects seven pins employed as general-purpose output pins. Within this register the wired-OR mode (WOMQ) control bit determines whether MISO, MOSI, SCK, and PCS[3:0] function as open-drain output pins or as normal output pins, regardless of their use as general-purpose I/O pins or as QSPI output pins. Likewise, the SCI control register 1 (SCCR1) has one bit that affects the TXD pin when it is employed as a general-purpose output. In this register the wired-OR mode (WOMS) control bit determines whether TXD functions as an open-drain output pin or a normal output pin, regardless of this pin's use as a general-purpose output pin or as an SCI output pin. Refer to **3.3 QSM Pin Control Registers** for more information on these registers.

2.1 SCI Pins

There are two pins associated with the SCI, the RXD and TXD pins. The SCI pins and their functions are listed in **Table 2-1**.

2.1.1 RXD — Receive Data

This dedicated input signal furnishes serial data input to the SCI. The RXD pin cannot be used for general-purpose I/O.

2.1.2 TXD — Transmit Data

This signal is the serial data output from the SCI. TXD is available as a general-purpose I/O pin when the SCI transmitter is disabled. When used as general-purpose I/O, TXD may be configured either as input or output as determined by the TXD bit in the QSM register DDRQS. The state of the TXD bit is ignored while the SCI is enabled. The TXD pin is enabled for SCI use by the transmitter enable bit (TE) in the SCI Control Register 1 (SCCR1). Refer to **5.2.2 SCI Control Register 1 (SCCR1)** for more information.

Table 2-1 External Pin Inputs/Outputs to the SC

Pin Names	Mnemonics	Mode	Function
Receive Data	RXD	Receiver Disabled Receiver Enabled	Not Used Serial Data Input to SCI
Transmit Data	TXD	Transmitter Disabled Transmitter Enabled	General-Purpose I/O Serial Data Output from SCI

2.2 QSPI Pins

Seven pins are associated with the QSPI. When not needed for a QSPI application, they may be configured as general-purpose I/O pins. **Table 2-2** identifies the QSPI pins and their functions. QSM register DDRQS determines whether the pins are designated as input or output. The user must initialize DDRQS for the QSPI to function correctly.

2.2.1 PCS[3:0] — Peripheral Chip-Selects

These bidirectional signals provide QSPI peripheral chip-selects.

2.2.2 $\overline{SS}$ — Slave Select

Assertion of this bidirectional signal selects the QSPI when in slave mode. This is the same pin as PCS0.

2.2.3 SCK — QSPI Serial Clock

This bidirectional signal furnishes the clock from the QSPI in Master mode or furnishes the clock to the QSPI in slave mode.

2.2.4 MISO — Master In Slave Out

This bidirectional signal furnishes serial data input to the QSPI in master mode, and serial data output from the QSPI in slave mode.

2.2.5 MOSI — Master Out Slave In

This bidirectional signal furnishes serial data output from the QSPI in master mode, and serial data input to the QSPI in slave mode.

Table 2-2 External Pin Inputs/Outputs to the QSPI

Pin Names	Mnemonics	Mode	Function
Master In Slave Out	MISO	Master Slave	Serial Data Input to QSPI Serial Data Output from QSPI
Master Out Slave In	MOSI	Master Slave	Serial Data Output from QSPI Serial Data Input to QSPI
Serial Clock	SCK[1]	Master Slave	Clock Output from QSPI Clock Input to QSPI
Peripheral Chip-Selects	PCS[3:1]	Master	Outputs Select Peripheral(s)
Peripheral Chip-Select[2] Slave Select[3]	PCS0/ $\overline{SS}$	Master Slave	Output Selects Peripheral(s) Input Selects the QSPI
Slave Select[4]	$\overline{SS}$	Master	May Cause Mode Fault

NOTES:

1. All QSPI pins (except SCK) can be used as general-purpose I/O if they are not used by the QSPI while the QSPI is operating.
2. An output (PCS0) when the QSPI is in master mode.
3. An input ($\overline{SS}$) when the QSPI is in slave mode.
4. An input ($\overline{SS}$) when the QSPI is in master mode; useful in multimaster systems.

SECTION 3
CONFIGURATION AND CONTROL

Registers of the QSM are divided into four categories: QSM global registers, QSM pin control registers, QSPI submodule registers, and SCI submodule registers. The QSPI and SCI registers are defined in **4.3 QSPI Programmer's Model and Registers** and **5.2 SCI Programmer's Model and Registers**, respectively. Writes to unimplemented bits have no meaning or effect, and reads from unimplemented bits always return a logic zero value.

The modmap bit of the system integration module (SIM) module configuration register (MCR) defines the most significant bit (ADDR23) of the address, shown in each register figure as Y (Y = $7 or $F). This bit, concatenated with the rest of the address given, forms the absolute address of each register.

Table 3-1 is a summary of the registers, bits, and reset states for the full QSM module.

As previously mentioned, **Table 3-2** is a quick reference guide to all the bits/fields of the QSM module. Along with the function, the register and register location of each bit/field are identified.

Table 3-1 QSM Register Summary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QSMCR $YFFC00	STOP	FRZ1	FRZ0	0	0	0	0	0	SUPV	0	0	0	IARB			
RESET:	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
QTEST $YFFC02	0	0	0	0	0	0	0	0	0	0	0	0	TSBD	SYNC	TQSM	TMM
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
QILR/QIVR $YFFC04	0	0	ILQSPI			ILSCI			INTV							
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$YFFC06	RESERVED															
SCCR0 $YFFC08	0	0	0	SCBR												
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
SCCR1 $YFFC0A	0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SCSR $YFFC0C	0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF
RESET:	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
SCDR $YFFC0E	0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0
RESET:	0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U
$YFFC10	RESERVED															
$YFFC12	RESERVED															
PORTQS $YFFC14	0	0	0	0	0	0	0	0	DATA7 (TXD)	DATA6 (PCS3)	DATA5 (PCS2)	DATA4 (PCS1)	DATA3 (PCS0*)	DATA2 (SCK)	DATA1 (MOSI)	DATA0 (MISO)
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PQSPAR/ DDRQS $YFFC16	0	PCS3	PCS2	PCS1	PCS0*	0	MOSI	MISO	TXD	PCS3	PCS2	PCS1	PCS0*	SCK	MOSI	MISO
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPCR0 $YFFC18	MSTR	WOMQ	BITS				CPOL	CPHA	SPBR							
RESET:	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
SPCR1 $YFFC1A	SPE	DSCKL							DTL							
RESET:	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
SPCR2 $YFFC1C	SPIFIE	WREN	WRTO	0	ENDQP				0	0	0	0	NEWQP			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPCR3/ SPSR $YFFC1E	0	0	0	0	0	LOOPQ	HMIE	HALT	SPIF	MODF	HALTA	0	CPTQP			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$YFFC20– $YFFCFF	RESERVED															
RECEIVE RAM $YFFD00– $YFFD1F	QSPI RECEIVE DATA (16 WORDS)															
TRANSMIT RAM $YFFD20– $YFFD3F	QSPI TRANSMIT DATA (16 WORDS)															
COMMAND RAM $YFFD40– $YFFD4F	CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*	CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

Y = m111, where m is the modmap bit in the module configuration register for the SIM (Y = $7 or $F).

* The PCS0 bit listed above represents the dual-function PCS0/$\overline{SS}$.

Table 3-2 Bit/Field Quick Reference Guide (Sheet 1 of 2)

Bit/Field Mnemonic	Function	Register	Register Location
SPBR	Serial Clock Baud Rate	SPCR0	QSPI
BITS	Bits Per Transfer	SPCR0	QSPI
BITSE	Bits Per Transfer Enable	QSPI RAM	QSPI
SCBR	Baud Rate	SCCR0	SCI
CONT	Continue	QSPI RAM	QSPI
CPHA	Clock Phase	SPCR0	QSPI
CPOL	Clock Polarity	SPCR0	QSPI
CPTQP	Completed Queue Pointer	SPSR	QSPI
DSCK	Peripheral Select Chip (PSC) to Serial Clock (SCK) Delay	QSPI RAM	QSPI
DSCKL	Delay before Serial Clock (SCK)	SPCR1	QSPI
DT	Delay after Transfer	QSPI RAM	QSPI
DTL	Length of Delay after Transfer	SPCR1	QSPI
ENDQP	Ending Queue Pointer	SPCR2	QSPI
FE	Framing Error Flag	SCSR	SCI
FRZ[1:0]	Freeze1–0	QSMCR	QSM
HALT	Halt	SPCR3	QSPI
HALTA	Halt Acknowledge Flag	SPSR	QSPI
HMIE	Halt Acknowledge Flag (HALTA) and Mode Fault Flag (MODF) Interrupt Enable	SPCR3	QSPI
IARB	Interrupt Arbitration Identification Number	QSMCR	QSM
IDLE	Idle Line Detected Flag	SCSR	SCI
ILIE	Idle Line Interrupt Enable	SCCR1	SCI
ILQSPI	Interrupt Level for QSPI	QILR	QSM
ILSCI	Interrupt Level of SCI	QILR	QSM
ILT	Idle Line Detect Type	SCCR1	SCI
INTV	Interrupt Vector	QIVR	QSM
LOOPS	SCI Loop Mode	SCCR1	SCI
LOOPQ	QSPI Loop Mode	SPCR3	QSPI
M	Mode Select (8/9 Bit)	SCCR1	SCI
MISO	Master In Slave Out	PQSPAR/DDRQS/ PORTQS	QSM
MODF	Mode Fault Flag	SPSR	QSPI
MOSI	Master Out Slave In	PQSPAR/DDRQS/ PORTQS	QSM
MSTR	Master/Slave Mode Select	SPCR0	QSPI
NEWQP	New Queue Pointer Value	SPCR2	QSPI
NF	Noise Error Flag	SCSR	SCI
OR	Overrun Error Flag	SCSR	SCI
PCS0/$\overline{SS}$	Peripheral Chip-Select/Slave Select	PQSPAR/DDRQS/ PORTQS	QSM
PCS[3:1]	Peripheral Chip-Selects	PQSPAR/DDRQS/ PORTQS	QSM
PE	Parity Enable	SCCR1	SCI
PF	Parity Error Flag	SCSR	SCI
PT	Parity Type	SCCR1	SCI
R[8:0]	Receive 8–0	SCDR	SCI

Table 3-2 Bit/Field Quick Reference Guide (Sheet 2 of 2)

Bit/Field Mnemonic	Function	Register	Register Location
RAF	Receiver Active Flag	SCSR	SCI
RDRF	Receive Data Register Full Flag	SCSR	SCI
RE	Receiver Enable	SCCR1	SCI
RIE	Receiver Interrupt Enable	SCCR1	SCI
RWU	Receiver Wakeup	SCCR1	SCI
SBK	Send Break	SCCR1	SCI
SCK	Serial Clock	DDRQS/PORTQS	QSM
SPE	QSPI Enable	SPCR1	QSPI
SPIF	QSPI Finished Flag	SPSR	QSPI
SPIFIE	SPI Finished Interrupt Enable	SPCR2	QSPI
STOP	Stop	QSMCR	QSM
SUPV	Supervisor/Unrestricted	QSMCR	QSM
SYNC	SCI Baud Clock Sync Signal	QTEST	QSM
T[8:0]	Transmit 8–0	SCDR	SCI
TC	Transmit Complete Flag	SCSR	SCI
TCIE	Transmit Complete Interrupt Enable	SCCR1	SCI
TDRE	Transmit Data Register Empty Flag	SCSR	SCI
TE	Transmit Enable	SCCR1	SCI
TIE	Transmit Interrupt Enable	SCCR1	SCI
TMM	Test Memory Map	QTEST	QSM
TQSM	Test QSM Enable	QTEST	QSM
TSBD	SPI Test Scan Path Select	QTEST	QSM
TXD	Transmit Data	DDRQS/PORTQS	QSM
WAKE	Wakeup Type	SCCR1	SCI
WOMQ	Wired-OR Mode for QSPI Pins	SPCR0	QSPI
WOMS	Wired-OR Mode for SCI Pins	SCCR1	SCI
WREN	Wrap Enable	SPCR2	QSPI
WRTO	Wrap To Select	SPCR2	QSPI

3.1 Overall QSM Configuration Summary

After reset, the QSM remains in an idle state, requiring initialization of several registers before any serial operations may begin execution. The following registers, fields, and bits are fully described later in this section. A general sequence guide for initialization follows:

- QSMCR (refer to **3.2.1 QSM Configuration Register (QSMCR)**

This register must be initialized to properly configure:

- Interrupt arbitration identification number used by the entire QSM module
- Supervisor/unrestricted bit (SUPV)
- FREEZE and/or STOP configuration; which should remain cleared to zero for normal operation.
- QIVR and QILR (refer to **3.2.3 QSM Interrupt Level Register (QILR)** and **3.2.4 QSM Interrupt Vector Register (QIVR)**

These registers are written to choose the base vector number for the entire QSM module and individual interrupt levels for the QSPI and SCI submodules.

- PORTQS and DDRQS (refer to **3.3.1 QSM Port Data Register (PORTQS)** and **3.3.3 QSM Data Direction Register (DDRQS)**

The pin control registers should be initialized in the order PORTQS and then DDRQS, thus establishing the default state and direction of the QSM pins.

For configuration of the QSPI submodule, initialize as follows:

- RAM (refer to **4.3.6 QSPI RAM**)
- PQSPAR (refer to **3.3.2 QSM Pin Assignment Register (PQSPAR)**

Assignment of appropriate pins to the QSPI must be made with this register.

- SPCR0 (refer to **4.3.1 QSPI Control Register 0 (SPCR0)**

The system designer must choose a transfer rate (baud) for operation in master mode, an appropriate clock phase, clock polarity, and the number of bits to be transferred in a serial operation. Master/slave mode select (MSTR) must be set to configure the QSPI for master mode or cleared to configure operation in slave mode. WOMQ should be set to enable or cleared to disable wired-OR mode operation.

- SPCR1 (refer to **4.3.2 QSPI Control Register 1 (SPCR1)**
  - — SPE must be set to enable the QSPI; this register should be written last.
  - — DTL allows the user to program a delay after any serial transfer, which is invoked by the DT bit for any serial transfer.
  - — DSCKL allows the user to set a delay before SCK (after PCS valid), which is invoked by the DSCK bit for any transfer.
- SPCR2 (refer to **4.3.3 QSPI Control Register 2 (SPCR2)**
  - — NEWQP and ENDQP, respectively, determine the beginning of a queue and the number of serial transfers (up to 16) to be considered a complete queue.
  - — WREN is set to enable queue wraparound, and WRTO helps determine the address used in wraparound mode.
  - — SPIFIE is set to enable interrupts when SPIF is asserted.
- SPCR3 (refer to **4.3.4 QSPI Control Register 3 (SPCR3)**

HALT may be used for program debug, and HMIE is set to enable CPU interrupts when HALTA or MODF is asserted; LOOPQ is set only to enable a feedback loop that can be used for self-test mode.

For configuration of the SCI submodule, initialize as follows:

- SCCR0 (refer to **5.2.1 SCI Control Register 0 (SCCR0)**

The system designer must choose a transfer rate (baud) for serial transfer operation.

- SCCR1 (refer to **5.2.2 SCI Control Register 1 (SCCR1)**
  - — The type of serial frame (8- or 9-bit) and the use of parity must be determined by M, PE, and PT.
  - — For receive operation, the system designer must consider use and type of wakeup (WAKE, RWU, ILT, ILIE). The receiver must be enabled (RE) and, usually, RIE should be set.
  - — For transmit operation, the transmitter must be enabled (TE) and, usually, TIE should be set. The use of wired-OR mode (WOMS) must also be decided.

Once the transmitter is configured, data is not sent until TDRE and TC are cleared. To clear TDRE and TC, the SCSR read must be followed by a write to SCDR (either the lower byte or the entire word).

3.2 QSM Global Registers

The QSM global registers contain system parameters used by both the QSPI and the SCI submodules. These registers define parameters used by the QSM to interface with the CPU and other system modules. The four global registers are listed in **Table 3-3**.

Table 3-3 QSM Global Registers

Address	Name	Usage
$YFFC00	QSMCR	QSM Configuration Register
$YFFC02	QTEST	QSM Test Register
$YFFC04	QILR	QSM Interrupt Level Register
$YFFC05	QIVR	QSM Interrupt Vector Register

3.2.1 QSM Configuration Register (QSMCR)

QSMCR contains parameters for interfacing to the CPU and the intermodule bus (IMB). This register can be modified only when the CPU is in supervisor mode.

QSMCR — QSM Configuration Register **$YFFC00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	FRZ1	FRZ0	0	0	0	0	0	SUPV	0	0	0	IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

STOP — Stop Enable

1 = QSM clock operation stopped

0 = Normal QSM clock operation

STOP places the QSM into a low power state by disabling the system clock in most parts of the module. QSMCR is the only register guaranteed to be readable while STOP is asserted. The QSPI RAM is not readable; however, writes to RAM or any register are guaranteed valid while STOP is asserted. STOP may be negated by the CPU and by reset.

The system software must stop each submodule before asserting STOP to avoid complications at restart and to avoid data corruption. The SCI submodule receiver and transmitter should be disabled, and the operation should be verified for completion before asserting STOP. The QSPI submodule should be stopped by asserting the HALT bit in SPCR3 and by asserting STOP after the HALTA flag is set.

FRZ1 — Freeze1

1 = Halt the QSM (on a transfer boundary)

0 = Ignore the FREEZE signal on the IMB

FRZ1 determines what action is taken by the QSM when the FREEZE signal of the IMB is asserted. FREEZE is asserted whenever the CPU enters the background mode.

WARNING

Ignoring the FREEZE signal can cause unpredictable results in the background mode operation of the QSM, because the CPU is unable to service interrupt requests in this mode. If FRZ1 equals one when the FREEZE line is asserted, the QSM comes to an orderly halt on a transfer boundary as if HALT had been asserted. The output pins continue to drive their last state. Once the FREEZE signal is negated, the QSM module restarts automatically.

FRZ0 — Freeze0

Reserved for future enhancement.

Bits [12:8] — Not Implemented

SUPV — Supervisor/Unrestricted

1 = Supervisor access

All registers in the QSM are placed in supervisor-only space. For any access from within user mode, address acknowledge (AACK) is not returned and the bus cycle is transferred externally.

0 = User access

Because the QSM contains a mix of supervisor and user registers, AACK returns for accesses with either supervisor or user mode, and the bus cycle remains internal. If a supervisor-only register is accessed in user mode, the module responds as if an access had been made to an unimplemented register location.

SUPV defines the assignable QSM registers as either supervisor-only data space or unrestricted data space.

Bits [6:4] — Not Implemented

IARB — Interrupt Arbitration Identification Number

Each module that generates interrupts, including the QSM, must have an IARB field. The value in this field is used to arbitrate for the IMB when two or more modules generate simultaneous interrupts of the same priority level. No two modules can share the same IARB value. The reset value of the IARB field is $0, which prevents the QSM from arbitrating during an interrupt acknowledge cycle (IACK). The IARB field should be initialized by system software to a value between $F (highest priority) and $1 (lowest priority). Otherwise, any interrupts generated are identified by the CPU as spurious.

3.2.2 QSM Test Register (QTEST)

QTEST is used in testing the QSM. Accesses to QTEST must be made while the MCU is in test mode. Test mode is for manufacturing use only. Applications should not use this register or enter test mode.

QTEST — QSM Test Register **$YFFC02**

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	TSBD	SYNC	TQSM	TMM
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TSBD — SPI Test Scan Path Select
1 = Enable delay to SCK scan path
0 = Enable SPI baud clock scan path

SYNC — SCI Baud Clock Synchronization Signal
1 = Inhibit SCI source signal (QCSCI1)
0 = Activate SCI source signal

TQSM — QSM Test Enable
1 = Enable QSM to send test scan paths
0 = Disable scan path

TMM — Test Memory Map
1 = QSM responds to test memory addresses
0 = QSM responds to QSM memory addresses

3.2.3 QSM Interrupt Level Register (QILR)

The QILR determines the priority level of interrupts requested by the QSM and the vector used when acknowledging an interrupt. Separate fields exist to hold the interrupt levels for the QSPI and the SCI submodules. Priority is used to determine which interrupt is serviced first when two or more modules or external peripherals simultaneously request an interrupt. This register may be accessed only when the CPU is in supervisor mode.

QILR — QSM Interrupt Level Register **$YFFC04**

15	14	13	12	11	10	9	8	7	0
0	0	ILQSPI			ILSCI			QIVR*	
RESET:									
0	0	0	0	0	0	0	0		

* QIVR — QSM Interrupt Vector Register

ILQSPI — Interrupt Level for QSPI
ILQSPI determines the priority level of all QSPI interrupts. This field should be programmed to a value between $0 (interrupts disabled) and $7 (highest priority). If both the QSPI and the SCI modules contain the same priority level (not equal to zero) and both modules simultaneously request interrupt servicing, the QSPI is given priority.

ILSCI — Interrupt Level of SCI
ILSCI determines the priority level of all SCI interrupts. This field should be programmed to a value between $0 (interrupts disabled) and $7 (highest priority).

3.2.4 QSM Interrupt Vector Register (QIVR)

At reset, QIVR is initialized to $0F, which corresponds to the uninitialized interrupt vector in the exception table. This vector is selected until QIVR is written. QIVR should be programmed to one of the user-defined vectors ($40-$FF) during initialization of the QSM.

After initialization, QIVR determines which two vectors in the exception vector table are to be used for QSM interrupts. The QSPI and SCI submodules have separate interrupt vectors adjacent to each other. Both submodules use the same interrupt vector with the least significant bit (LSB) determined by the submodule causing the interrupt. The value of INTV0 used during an IACK cycle is supplied by the bus interface unit (BIU). During an IACK, INTV[7:1] are driven on the DATA[7:1] lines. The INTV0 drives line DATA0 with a zero for an SCI interrupt and with a one for a QSPI interrupt. Writes to INTV0 have no meaning or effect. Reads of INTV0 return a value of one.

QIVR — QSM Interrupt Vector Register **$YFFC05**

15	8	7	6	5	4	3	2	1	0
QILR*		INTV[7:0]							
RESET:		0	0	0	0	1	1	1	1

INTV0 is set to a logic level one when the QSPI generates an interrupt and set to a logic level zero when the SCI generates an interrupt.

* QILR — QSM Interrupt Level Register

3.3 QSM Pin Control Registers

Table 3-3 identifies the three pin control registers of the QSM. The QSM determines the use of nine pins, eight of which form a parallel port on the MCU. Although these pins are used by the serial subsystems, any pin may alternately be assigned as general-purpose I/O on a pin by pin basis. For use of these pins as general-purpose I/O, they must not be assigned to the QSPI submodule in register PQSPAR. To avoid briefly driving incorrect data, the first byte to be output should be written before register DDRQS is configured for any output pins. DDRQS should then be written to determine the direction of data flow on the pins and to output the value contained in register PORTQS for all pins defined as outputs. Subsequent data for output is then written to PORTQS.

Table 3-4 QSM Pin Control Registers

Address	Name	Usage
$YFFC15	PORTQS	QSM Port Data Register
$YFFC16	PQSPAR	QSM Pin Assignment Register
$YFFC17	DDRQS	QSM Data Direction Register

3.3.1 QSM Port Data Register (PORTQS)

PORTQS determines the actual input or output value of a QSM port pin if the pin is defined in PQSPAR as general-purpose I/O. All QSM port pins may be used as general-purpose I/O. Writes to this register affect the pins defined as outputs; reads of this register return the actual value of the pins.

PORTQS — QSM Port Data Register **$YFFC15**

15 … 8	7	6	5	4	3	2	1	0
RESERVED	DATA7 (TXD)	DATA6 (PCS3)	DATA5 (PCS2)	DATA4 (PCS1)	DATA3 (PCS0/$\overline{SS}$)	DATA2 (SCK)	DATA1 (MOSI)	DATA0 (MISO)
RESET:	0	0	0	0	0	0	0	0

3.3.2 QSM Pin Assignment Register (PQSPAR)

PQSPAR determines which of the QSPI pins, with the exception of the SCK pin, are actually used by the QSPI submodule, and which pins are available for general-purpose I/O. Pins may be assigned to the QSPI or to function as general-purpose I/O on a pin-by-pin basis. QSPI pins designated by PQSPAR as general-purpose I/O are controlled only by DDRQS and PORTQS and the QSPI has no effect on these pins. PQSPAR does not affect the operation of the SCI submodule.

PQSPAR — QSM Pin Assignment Register **$YFFC16**

15	14	13	12	11	10	9	8	7 … 0
0	PCS3	PCS2	PCS1	PCSO/$\overline{SS}$	0	MOSI	MISO	DDRQS*
RESET:								
0	0	0	0	0	0	0	0	

Bit 15 — Not Implemented

TE in register SCCR1 determines whether the TXD pin is controlled by the SCI or functions as a general-purpose I/O pin.

PCS[3:1] — Peripheral Chip-Selects 3–1

PCS0/$\overline{SS}$ — Peripheral Chip-Select 0/Slave Select

These bits determine whether the associated QSM port pins function as general-purpose I/O pins or are assigned to the QSPI submodule.

Bit 10 — Not Implemented

(When the QSPI is enabled, the SCK pin is required.)

MOSI — Master Out Slave In

MISO — Master In Slave Out

These bits determine whether the associated QSM port pin functions as a general-purpose I/O pin or is assigned to the QSPI submodule.

3.3.3 QSM Data Direction Register (DDRQS)

DDRQS sets each I/O pin, except for TXD, as an input or an output regardless of whether the QSPI submodule is enabled or disabled. All QSM pins are configured during reset as general-purpose inputs. (The QSPI and SCI are disabled.) The RXD pin remains an input pin dedicated to the SCI submodule and does not function as a general-purpose I/O pin.

DDRQS — QSM Data Direction Register **$YFFC17**

15 8	7	6	5	4	3	2	1	0
PQSPAR*	TXD	PCS3	PCS2	PCS1	PCS0/$\overline{SS}$	SCK	MOSI	MISO
RESET:								
	0	0	0	0	0	0	0	0

* PQSPAR — QSM Pin Assignment Register

TXD — Transmit Data

This bit determines the direction of the TXD pin (input or output), only if the SCI transmitter is disabled. If the SCI transmitter is enabled, the TXD bit is ignored, and the TXD pin is forced to function as an output.

PCS[3:1] — Peripheral Chip-Selects 3–1

PCS0/$\overline{SS}$ — Peripheral Chip-Select 0/Slave Select

SCK — Serial Clock

MOSI — Master Out Slave In

MISO — Master In Slave Out

Refer to **4.4.2 Slave Mode** for additional details on this pin.

All of the above bits determine the QSPI port pin operation to be input or output.

1 = Output

0 = Input

SECTION 4
QSPI SUBMODULE

The QSPI submodule communicates with external peripherals and other MCUs via synchronous serial bus. The QSPI is fully compatible with the serial peripheral interface (SPI) systems found on other Motorola devices such as the M68HC11 and M68HC05 Families. It has all of the capabilities of the SPI system as well as several new features. The following paragraphs describe the features, block diagram, pin descriptions, programmer's model (memory map) inclusive of registers, and the master and slave operation of the QSPI.

4.1 Features

Standard SPI features are listed below, followed by a list of the additional features offered on the QSPI:

- Full-Duplex, Three-Wire Synchronous Transfers
- Half-Duplex, Two-Wire Synchronous Transfers
- Master or Slave Operation
- Programmable Master Bit Rates
- Programmable Clock Polarity and Phase
- End-of-Transmission Interrupt Flag
- Master-Master Mode Fault Flag
- Easily Interfaces to Simple Expansion Parts (A/D converters, EEPROMS, display drivers, etc.)

QSPI-Enhanced features are as follows:

- Programmable Queue — up to 16 preprogrammed transfers
- Programmable Peripheral Chip-Selects — four pins select up to 16 SPI chips
- Wraparound Transfer Mode — for autoscanning of serial A/D (or other) peripherals, with no CPU overhead
- Programmable Transfer Length — from 8–16 bits inclusive
- Programmable Transfer Delay — from 1 µs to 0.5 ms (at 16.78 MHz)
- Programmable Queue Pointer
- Continuous Transfer Mode — up to 256 bits

4.1.1 Programmable Queue

A programmable queue allows the QSPI to perform up to 16 serial transfers without CPU intervention. Each transfer corresponds to a queue entry containing all the information needed by the QSPI to independently complete one serial transfer. This unique feature greatly reduces CPU/QSPI interaction, resulting in increased CPU and system throughput.

4.1.2 Programmable Peripheral Chip-Selects

Four peripheral chip-select pins allow the QSPI to access up to 16 independent peripherals by decoding the four peripheral chip-select signals. Up to four independent peripherals can be selected by direct connection to a chip-select pin. The peripheral chip-selects simplify interfacing to two or more serial peripherals by providing dedicated peripheral chip-select signals, alleviating the need for CPU intervention.

4.1.3 Wraparound Transfer Mode

Wraparound transfer mode allows automatic, continuous re-execution of the preprogrammed queue entries. Newly transferred data replaces previously transferred data. Wraparound simplifies interfacing with A/D converters by automatically providing the CPU with the latest A/D conversions in the QSPI RAM. Consequently, serial peripherals appear as memory-mapped parallel devices to the CPU.

4.1.4 Programmable Transfer Length

The number of bits in a serial transfer is programmable from eight to 16 bits, inclusive. For example, ten bits could be used for communicating with an external 10-bit A/D converter. Likewise, a vacuum fluorescent display driver might require a 12-bit serial transfer. The programmable length simplifies interfacing to serial peripherals that require different data lengths.

4.1.5 Programmable Transfer Delay

An inter-transfer delay may be programmed from approximately 1 to 500 µs (using a 16.78-MHz system clock). For example, an A/D converter may require time between transfers to complete a new conversion. The default delay is 1 µs (17 clocks at 16.78-MHz). The programmable length of delay simplifies interfacing to serial peripherals that require delay time between data transfers.

4.1.6 Programmable Queue Pointer

The QSPI has a pointer that identifies the queue location containing the data for the next serial transfer. The CPU can switch from one task to another in the QSPI by writing to the queue pointer, changing the location in the queue that is to be transferred next. Otherwise, the pointer increments after each serial transfer. By segmenting the queue, multiple-task support can be provided by the QSPI.

4.1.7 Continuous Transfer Mode

The continuous transfer mode allows the user to send and receive an uninterrupted bit stream with a peripheral. A minimum of 8 bits and a maximum of 256 bits may be transferred in a single burst without CPU intervention. Longer transfers are possible; however, minimal CPU intervention is required to prevent loss of data. A 1 µs pause (using a 16.78-MHz system clock) is inserted between each queue entry transfer.

4.2 Block Diagram

Figure 4-1 provides a block diagram of the QSPI submodule components.

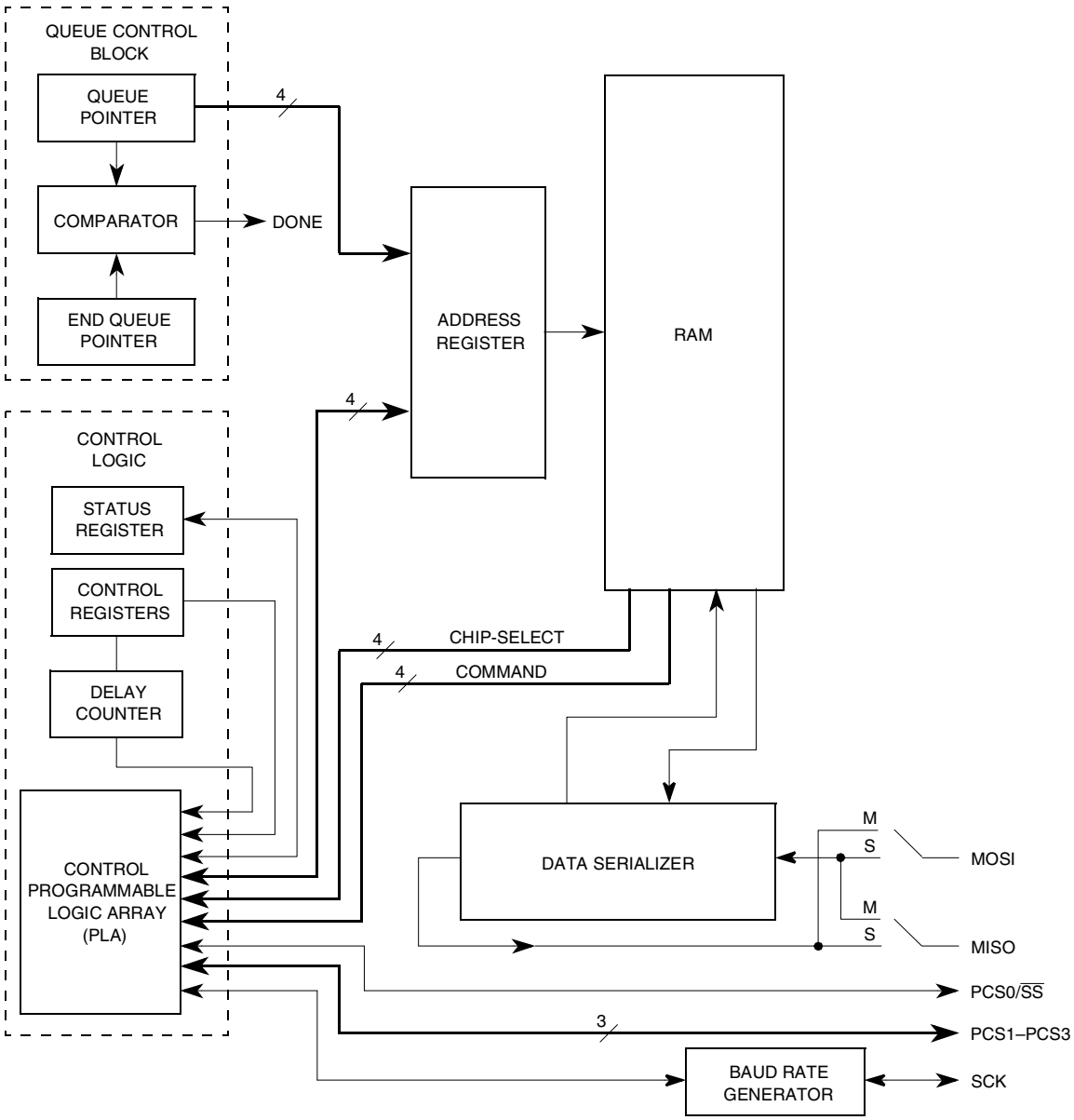


Figure 4-1 QSPI Submodule Diagram

4.3 QSPI Programmer's Model and Registers

The programmer's model (memory map) for the QSPI submodule consists of the QSM global and pin control registers (refer to **3.2 QSM Global Registers** and **3.3 QSM Pin Control Registers**), four QSPI control registers, one status register, and the 80-byte QSPI RAM. **Table 4-1** lists the registers and the QSPI RAM of the programmer's model. All of the registers and RAM can be read and written by the CPU. The four control

registers must be initialized in proper order before the QSPI is enabled to ensure defined operation. Only the control registers must adhere to the order of sequence prescribed in **3.1 Overall QSM Configuration Summary**. Write register SPCR1 last when setting up the QSPI, as this register contains the QSPI enable bit (SPE). Asserting this bit starts the QSPI. QSPI control registers are reset to a defined state and may then be changed by the CPU. Reset values are shown below each register.

Table 4-1 QSPI Registers

Address	Name	Usage
$YFFC18, 9	SPCR0	QSPI Control Register 0
$YFFC1A, B	SPCR1	QSPI Control Register 1
$YFFC1C, D	SPCR2	QSPI Control Register 2
$YFFC1E	SPCR3	QSPI Control Register 3
$YFFC1F	SPSR	QSPI Status Register
$YFFD00–1F	RAM	QSPI Receive Data (16 Words)
$YFFD20–3F	RAM	QSPI Transmit Data (16 Words)
$YFFD40–4F	RAM	QSPI Command Control (8 Words)

In general, rewriting the same value into a control register does not affect the QSPI operation with the exception of NEWQP (bits [3:0]) in SPCR2. Rewriting the same value to these bits causes the RAM queue pointer to restart execution at the designated location.

If control bits are to be changed, the CPU should halt the QSPI first. With the exception of SPCR2, writing a different value into a control register while the QSPI is enabled may disrupt operation. SPCR2 is buffered, preventing any disruption of the current serial transfer. After completion of the current serial transfer, the new SPCR2 values become effective.

4.3.1 QSPI Control Register 0 (SPCR0)

SPCR0 contains parameters for configuring the QSPI before it is enabled. Although the CPU can read and write this register, the QSM has read-only access.

SPCR0 — QSPI Control Register 0 **$YFFC18**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSTR	WOMQ	BITS				CPOL	CPHA	SPBR							
RESET:															
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

MSTR — Master/Slave Mode Select

1 = QSPI is system master and can initiate transmission to external SPI devices.

0 = QSPI is a slave device, and only responds to externally generated serial transfers.

MSTR configures the QSPI for either master or slave mode operation. This bit is cleared on reset and may only be written by the CPU, not the QSM.

WOMQ — Wired-OR Mode for QSPI Pins

1 = All QSPI port pins designated as output by DDRQS function as open drain outputs and can be wire-ORed to other external lines.

0 = Output pins have normal outputs instead of open-drain outputs.

WOMQ allows the QSPI pins to be wire-ORed, regardless of whether they are used as general-purpose outputs or as QSPI outputs. WOMQ affects the QSPI pins whether the QSPI is enabled or disabled. This bit does not affect the SCI submodule transmit (TXD) pin, which has its own WOMS bit in an SCI control register.

BITS — Bits Per Transfer

In master mode, BITS determines the number of data bits transferred for each serial transfer in the queue that has the command control bit (BITSE of the QSPI RAM) equal to one. If BITSE equals zero for a command, 8 bits are transferred for that command regardless of the value in BITS. Data transfers from 8 to 16 bits are supported. Illegal (reserved) values all default to 8 bits. BITSE is not used in slave mode. All transfers are of the length specified by BITS. **Table 4-2** shows the number of bits per transfer.

Table 4-2 Bits per Transfer if Command Control Bit BITSE = 1

Bits [13:10]	Bits per Transfer
0000	16
0001	Reserved
0010	Reserved
0011	Reserved
0100	Reserved
0101	Reserved
0110	Reserved
0111	Reserved
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

CPOL — Clock Polarity

1 = The inactive state value of SCK is high.

0 = The inactive state value of SCK is low.

CPOL is used to determine the inactive state value of the serial clock (SCK). CPOL is used in conjunction with CPHA to produce the desired clock-data relationship between master and slave device(s). QSPI clock/data timing relationships are specified in individual microcontroller user's manuals.

CPHA — Clock Phase

1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.

0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK.

CPHA determines which edge of SCK causes data to change and which edge of SCK causes data to be captured. CPHA is used in conjunction with CPOL to produce the desired clock-data relationship between master and slave device(s). Note that CPHA is set at reset.

SPBR — Serial Clock Baud Rate

The QSPI internally generates the baud rate for SCK, the frequency of which is programmable by the user. The clock signal is derived from the MCU system clock using a modulus counter. At reset, BAUD is initialized to a 2.1-MHz SCK frequency (16.78-MHz system clock).

The user programs a baud rate for SCK by writing a baud value from 2 to 255. The following equation determines the SCK baud rate:

$$\text{SCK Baud Rate} = \text{System Clock}/(2 * \text{SPBR}) \quad (4\text{-}1)$$

or

$$\text{SPBR} = \text{System Clock}/(2 * \text{SCK Baud Rate Desired}) \quad (4\text{-}2)$$

where SPBR equals 2, 3, 4,..., 255.

Programming SPBR with the values zero or one disables the QSPI baud rate generator. SCK is disabled and assumes its inactive state value. No serial transfers occur. SPBR has 254 active values. **Table 4-3** lists several possible baud values and the corresponding SCK frequency based on a 16.78-MHz system clock.

Table 4-3 Examples of SCK Frequencies

System Clock Frequency	Required Division Ratio	Value of SPBR	Actual SCK Frequency
16.78 MHz	4	2	4.19 MHz
	8	4	2.10 MHz
	16	8	1.05 MHz
	34	17	493 kHz
	168	84	100 kHz
	510	255	33 kHz

4.3.2 QSPI Control Register 1 (SPCR1)

SPCR1 contains parameters for configuring the QSPI before it is enabled. Although the CPU can read and write this register, the QSM has read access only, except for SPE. This bit is automatically cleared by the QSPI after completing all serial transfers or when a mode fault occurs.

SPCR1— QSPI Control Register 1 **$YFFC1A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPE	DSCKL							DTL							
RESET:															
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

SPE — QSPI Enable

1 = The QSPI is enabled and the pins allocated by QSM register PQSPAR are controlled by the QSPI.

0 = The QSPI is disabled, and the seven QSPI pins can be used as general-purpose I/O pins, regardless of the values in PQSPAR.

This bit enables or disables the QSPI submodule. Setting SPE causes the QSPI to begin operation. If the QSPI is a master, setting SPE causes the QSPI to begin initiating serial transfers. If the QSPI is a slave, the QSPI begins monitoring the PCS0/$\overline{SS}$ pin to respond to the external initiation of a serial transfer.

When the QSPI is disabled, the CPU may use the QSPI RAM. When the QSPI is enabled, both the QSPI and the CPU have access to the QSPI RAM. The CPU has both read and write access capability to all 80 bytes of the QSPI RAM. The QSPI can read only the transmit data segment and the command control segment, and can write only the receive data segment of the QSPI RAM.

The QSPI turns itself off automatically when it is finished by clearing SPE. An error condition called mode fault (MODF) also clears SPE. This error occurs when PCS0/$\overline{SS}$ is configured for input, the QSPI is a system master (MSTR = 1), and PCS0/$\overline{SS}$ is driven low externally.

To stop the QSPI, assert the HALT bit in SPCR3, then wait until the HALTA bit in SPSR is set. SPE may then be safely cleared to zero, providing an orderly method of quickly shutting down the QSPI after the current serial transfer is completed. The CPU can immediately disable the QSPI by just clearing SPE; however, loss of data from a current serial transfer may result and confuse an external SPI device.

DSCKL — Delay before SCK

This bit determines the length of time the QSPI delays from peripheral chip-select (PCS) valid to SCK transition for serial transfers in which the command control bit, DSCK of the QSPI RAM, equals one. PCS may be any of the four peripheral chip-select pins. The following equation determines the actual delay before SCK:

$$\text{PCS to SCK Delay} = [\text{DSCKL}/\text{System Clock Frequency}] \quad (4\text{-}3)$$

where DSCKL equals {1,2,3,... 127}.

NOTE

A zero value for DSCKL causes a delay of 128/system clocks, which equals 7.6 µs for a 16.78-MHz system clock. Because of design limits, a DSCKL value of one defaults to the same timing as a value of two.

If a queue entry's DSCK equals zero, then DSCKL is not used. Instead, the PCS valid-to-SCK transition is one-half SCK period.

DTL — Length of Delay after Transfer

These bits determine the length of time that the QSPI delays after each serial transfer in which the command control bit, DT of the QSPI RAM, equals one. The following equation is used to calculate the delay:

Delay after transfer = [(32 * DTL)/system clock frequency] (4-4)

where DTL equals {1, 2, 3,... 255}.

NOTE

A zero value for DTL causes a delay-after-transfer value of (32 * 256)/system clock, which equals 488.5 µs with a 16.78-MHz system clock.

If DT equals zero, a standard delay is inserted.

Standard Delay-after-Transfer = [17/System Clock] (4-5)

= 1 µs with a 16.78-MHz System Clock

Delay after transfer can be used to ensure that the deselect time requirement (for peripherals having such a requirement) is met. Some peripherals must be deselected for a minimum period of time between consecutive serial transfers. A delay after transfer can be inserted between consecutive transfers to a given peripheral to ensure that its minimum deselect time requirement is met or to allow serial A/D converters to complete conversion before the next transfer is made.

4.3.3 QSPI Control Register 2 (SPCR2)

SPCR2 contains parameters for configuring the QSPI. Although the CPU can read and write this register, the QSM has read access only. Writes to this register are buffered. A write to SPCR2 that changes any of the bit values (while the QSPI is operating) is ineffective on the current serial transfer, but becomes effective on the next serial transfer. Reads of SPCR2 return the actual current value of the register, not the buffer. Refer to **4.4 Operating Modes and Flowcharts** for a detailed description of this register.

SPCR2 — QSPI Control Register 2 **$YFFC1C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPIFIE	WREN	WRTO	0	ENDQP				0	0	0	0	NEWQP			
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SPIFIE — SPI Finished Interrupt Enable

1 = QSPI interrupts enabled

0 = QSPI interrupts disabled

SPIFIE enables the QSPI to generate a CPU interrupt upon assertion of the status flag SPIF. Because it is buffered, the value written to SPIFIE applies only upon completion of the queue (the transfer of the entry indicated by ENDQP). Thus, if a single sequence of queue entries is to be transferred (i.e., no WRAP), then SPIFIE should be set to the desired state before the first transfer.

If a subqueue (see bit NEWQP) is to be used, the same CPU write that causes a branch to the subqueue may enable or disable the SPIF interrupt for the subqueue. The primary queue retains its own selected interrupt mode, either enabled or disabled.

The SPIF interrupt must be cleared by clearing SPIF. Later interrupts may then be prevented by clearing SPIFIE to zero.

The QSPI has three possible interrupt sources, but only one interrupt vector. These sources are SPIF, MODF, and HALTA. When the CPU responds to a QSPI interrupt, the user must ascertain the exact interrupt cause by reading register SPSR. Any interrupt that was set may then be cleared by writing to SPSR with a zero in the bit position corresponding to the exact interrupt source. Clearing SPIFIE does not immediately clear an interrupt already caused by SPIF.

WREN — Wrap Enable

1 = Wraparound mode enabled
0 = Wraparound mode disabled

WREN enables or disables wraparound mode. If enabled, the QSPI executes commands in the queue through the command contained in ENDQP. Execution continues at either address $0 or at the address found in NEWQP, depending on the state of WRTO. The QSPI continues looping until either WREN is negated, HALT is asserted, or SPE is negated. Once WREN is negated, the QSPI finishes executing commands through the command at the address contained in ENDQP, sets the SPIF flag, and stops. When WREN is set, SPIF is set each time the QSPI transfers the entry indicated by ENDQP.

WRTO — Wrap To

When wraparound mode is enabled and after the end of queue has been reached, WRTO determines which address the QSPI executes next. End of queue is determined by an address match with ENDQP. Execution wraps to address $0 if WRTO is not set, or to the address found in NEWQP if WRTO is set.

Bit 12 — Not Implemented

ENDQP — Ending Queue Pointer

This field determines the last absolute address in the queue to be completed by the QSPI. After completing each command, the QSPI compares the queue pointer value of the just-completed command with the value of ENDQP. If the two values match, the QSPI assumes it has reached the end of the programmed queue and sets the SPIF flag to so indicate.

The QSPI RAM queue has 16 entries: $0–$F. The user may program the NEWQP to start executing commands, beginning at any of the 16 addresses. Similarly, the user may program the ENDQP to stop execution of commands at any of the 16 addresses.

The queue is a circular data structure. If ENDQP is set to a lower address than NEWQP, the QSPI executes commands through address $F, and then continues execution at address $0 and so on until it stops after executing the command at address ENDQP. A maximum of 16 commands are executed before stopping, unless wraparound mode is enabled or unless the user modifies NEWQP and/or ENDQP.

The user may write a NEWQP value at any time, changing the flow of execution. ENDQP may also be written at any time, changing the length of the queue. Wraparound mode may also be enabled, causing continuous execution until the mode is disabled or the QSPI is halted.

Bits [7:4] — Not Implemented

NEWQP — New Queue Pointer Value

NEWQP determines which queue entry the QSPI transfers first. NEWQP should be initialized before the QSPI is enabled with SPE. NEWQP may also be written while the QSPI is operating. When this happens, the QSPI completes transfer of the queue entry in progress and then immediately begins transferring queue entries starting with the entry indicated by the NEWQP.

In this way, NEWQP provides additional functionality to the QSPI by providing a mechanism for supporting multiple queues or subqueues within the QSPI RAM. By changing the value in NEWQP, the user can cause the QSPI to execute a sequence of QSPI commands beginning at any location in the queue. Therefore, the user is able to set up in advance separate subqueues for different tasks within the QSPI RAM. By writing to NEWQP, selection between the different subqueues within the QSPI RAM is accomplished.

If wraparound mode is enabled by setting WREN and WRTO in SPCR2, NEWQP assumes an additional function. When the end of the queue is reached, as determined by ENDQP, the address contained in NEWQP is used by the QSPI to wrap around to the first queue entry. The QSPI then re-executes the queued commands repeatedly until halted.

4.3.4 QSPI Control Register 3 (SPCR3)

SPCR3 contains parameters for configuring the QSPI. The CPU can read and write this register; the QSM has read-only access.

SPCR3 — QSPI Control Register **$YFFC1E**

15	14	13	12	11	10	9	8	7 … 0
0	0	0	0	0	LOOPQ	HMIE	HALT	SPSR*
RESET:								
0	0	0	0	0	0	0	0	

* SPSR — QSPI Status Register

Bits [15:11] — Not Implemented

LOOPQ — QSPI Loop Mode

1 = Feedback path enabled
0 = Feedback path disabled

LOOPQ enables or disables the feedback path on the data serializer for testing. If enabled, LOOPQ routes serial output data back into the data serializer, instead of received data. If disabled, LOOPQ allows regular received data into the data serializer. LOOPQ does not affect the QSPI output pins.

HMIE — HALTA and MODF Interrupt Enable

1 = HALTA and MODF interrupts enabled
0 = HALTA and MODF interrupts disabled

HMIE enables or disables QSPI interrupts to the CPU caused when either the HALTA status flag or the MODF status flag in SPSR is asserted. When HMIE is set, the asser-

tion of either flag causes the QSPI to send a hardware interrupt to the CPU. When HMIE is clear, the asserted flag does not cause an interrupt.

HALT — Halt

1 = Halt enabled

0 = Halt not enabled

This bit is used by the CPU to stop the QSPI on a queue boundary. The QSPI halts in a known state from which it can later be restarted. When HALT is asserted by the CPU, the QSPI finishes executing the current serial transfer (up to 16 bits) and then halts. While halted, if the command control bit (CONT of the QSPI RAM) for the last command was asserted, the QSPI continues driving the peripheral chip-select pins with the value designated by the last command before the halt. If CONT was clear, the QSPI drives the peripheral chip-select pins to the value in QSM register PORTQS.

If HALT is asserted during the last command in the queue, the QSPI completes the last command, asserts both HALTA and SPIF, and clears SPE. If the last queue command has not been executed, asserting HALT does not set SPIF nor clear SPE. QSPI execution continues when the CPU clears HALT.

4.3.5 QSPI Status Register (SPSR)

SPSR contains QSPI status information. Only the QSPI can assert the bits in this register. The CPU reads this register to obtain status information and writes this register to clear status flags. CPU writes to CPTQP have no effect.

SPSR — QSPI Status Register **$YFFC1F**

15 … 8	7	6	5	4	3	2	1	0
SPCR3*	SPIF	MODF	HALTA	0	CPTQP			
RESET:	0	0	0	0	0	0	0	0

* SPCR3 — QSPI Control Register 3

SPIF — QSPI Finished Flag

1 = QSPI finished

0 = QSPI not finished

SPIF is set when the QSPI finishes executing the last command determined by the address contained in ENDQP in SPCR2. When the address of the command being executed matches the ENDQP, the SPIF flag is set after finishing the serial transfer.

If wraparound mode is enabled (WREN = 1), the SPIF is set, after completion of the command defined by ENDQP, each time the QSPI cycles through the queue. If SPIFIE in SPCR2 is set, an interrupt is generated when SPIF is asserted. Once SPIF is set, the CPU may clear it by reading SPSR followed by writing SPSR with a zero in SPIF.

MODF — Mode Fault Flag

1 = Another SPI node requested to become the network SPI master while the QSPI was enabled in master mode (MSTR = 1), or the PCS0/$\overline{SS}$ pin was incorrectly pulled low by external hardware.

0 = Normal operation

MODF is asserted by the QSPI when the QSPI is the serial master (MSTR = 1) and the slave select (PCS0/$\overline{SS}$) input pin is pulled low by an external driver. This is possible only if the PCS0/$\overline{SS}$ pin is configured as input by DDRQS. This low input to $\overline{SS}$ is not a normal operating condition. It indicates that a multimaster system conflict may exist, that another MCU is requesting to become the SPI network master, or simply that the hardware is incorrectly affecting PCS0/$\overline{SS}$. SPE in SPCR1 is cleared, disabling the QSPI. The QSPI pins revert to control by PORTQS. If MODF is set and HMIE in SPCR3 is asserted, the QSPI generates an interrupt to the CPU.

The CPU may clear MODF by reading SPSR with MODF asserted, followed by writing SPSR with a zero in MODF. After correcting the mode fault problem, the QSPI can be re-enabled by asserting SPE.

The PCS0/$\overline{SS}$ pin may be configured as a general-purpose output instead of input to the QSPI. This inhibits the mode fault checking function. In this case, MODF is not used by the QSPI.

HALTA — Halt Acknowledge Flag

1 = QSPI halted

0 = QSPI not halted

HALTA is asserted by the QSPI when it has come to an orderly halt at the request of the CPU, via the assertion of HALT. To prevent undefined operation, the user should not modify any QSPI control registers or RAM while the QSPI is halted.

If HMIE in SPCR3 is set, the QSPI sends interrupt requests to the CPU when HALTA is asserted. The CPU can only clear HALTA by reading SPSR with HALTA set and then writing SPSR with a zero in HALTA.

Bit 4 — Not Implemented

CPTQP — Completed Queue Pointer

CPTQP contains the queue pointer value of the last command in the queue that was completed. The value of CPTQP is not updated until the command has been completed entirely. While the first command in a queue is executing, CPTQP contains either the reset value ($0) or the pointer to the last command completed in the previous queue.

If the QSPI is halted, CPTQP may be used to determine which commands have not been executed. The CPTQP may also be used to determine which locations in the receive data segment of the QSPI RAM contain valid received data.

4.3.6 QSPI RAM

The QSPI uses an 80-byte block of dual-access static RAM, which can be accessed by both the QSPI and the CPU. Because of sharing, the length of time taken by the CPU to access the QSPI RAM when the QSPI is enabled, may be longer than when the QSPI is disabled. From one to four CPU wait states may be inserted by the QSPI in the process of reading or writing.

The size and type of access of the QSPI RAM by the CPU affects the QSPI access time. The QSPI is byte, word, and long-word addressable. Only word accesses of the

RAM by the CPU are coherent accesses because these accesses are an indivisible operation. If the CPU makes a coherent access of the QSPI RAM, the QSPI cannot access the QSPI RAM until the CPU is finished. However, a long-word or misaligned word access is not coherent because the CPU must break its access of the QSPI RAM into two parts, which allows the QSPI to access the QSPI RAM between the two accesses by the CPU.

The RAM is divided into three segments: receive data RAM, transmit data RAM, and command control RAM. Receive data is information received from a serial device external to the MCU. Transmit data is information stored by the CPU for transmission to an external peripheral chip. Command control contains all the information needed by the QSPI to perform the transfer. **Figure 4-2** illustrates the organization of the RAM.

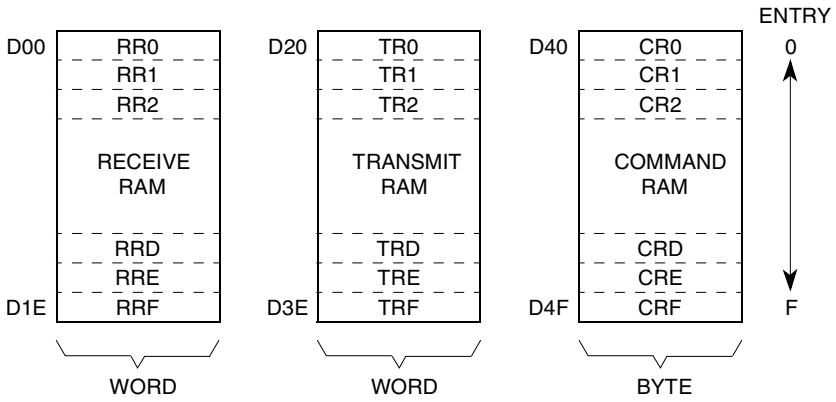


Figure 4-2 Organization of the QSPI RAM

Once the CPU has set up the queue of QSPI commands and enabled the QSPI, the QSPI operates independently of the CPU. The QSPI executes all of the commands in its queue, sets a flag indicating that it is finished, and then either interrupts the CPU or waits for CPU intervention.

4.3.6.1 Receive Data RAM

This segment of the RAM stores the data that is received by the QSPI from peripherals, SPI bus masters, or other MCUs. The CPU reads this segment of RAM to retrieve the data from the QSPI. Data stored in receive RAM is right-justified, i.e., the least significant bit is always in the right-most bit position within the word (bit 0) regardless of the serial transfer length. Unused bits in a receive queue entry are set to zero by the QSPI upon completion of the individual queue entry. The CPU can access the data using byte, word, or long-word addressing.

The CPTQP value in SPSR shows which queue entries have been executed. The CPU uses this information to determine which locations in receive RAM contain valid data before reading them.

4.3.6.2 Transmit Data RAM

This segment of the RAM stores the data that is to be transmitted by the QSPI to peripherals. The CPU normally writes one word of data into this segment for each queue command to be executed. If the corresponding peripheral, such as a serial input port, is used solely to input data, then this segment does not need to be initialized.

Information to be transmitted by the QSPI should be written by the CPU to the transmit data segment in a right-justified manner. The information in the transmit data segment of the RAM cannot be modified by the QSPI. The QSPI merely copies the information to its data serializer for transmission to a peripheral. Information in transmit RAM remains there until it is re-written by the CPU.

4.3.6.3 Command RAM

The command segment of the QSPI RAM is used only by the QSPI when it is in master mode. The CPU writes one byte of control information to this segment for each QSPI command to be executed. The information in the command RAM cannot be modified by the QSPI. It merely uses the information to perform the serial transfer.

Command RAM consists of 16 bytes. Each byte is divided into two fields. The first, the peripheral chip-select field, activates the correct serial peripheral during the transfer. The second, the command control field, provides transfer options specifically for that command/serial transfer. This feature gives the user more control over each transfer, providing the flexibility to interface to external SPI chips with different requirements.

A maximum of 16 commands can be in the queue command control bytes. These bytes are assigned an address from $0–$F. Queue execution by the QSPI proceeds from the address contained in NEWQP through the address contained in ENDQP. Both of these fields are contained in SPCR2.

COMMAND RAM — Command RAM **$YFFD40**

7	6	5	4	3	2	1	0
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*
—	—	—	—	—	—	—	—
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

$YFFD40

COMMAND CONTROL **PERIPHERAL CHIP-SELECT**

*The PCS0 bit represents the dual-function PCS0/$\overline{SS}$.

PCS[3:0]/$\overline{SS}$ — Peripheral Chip-Select

The four peripheral chip-select bits can be used directly to select one of four external chips for the serial transfer, or decoded by external hardware to select one of 16 chip-select patterns for the serial transfer. More than one peripheral chip-select may be activated at a time, which is useful for broadcast messages in a multinode SPI system. More than one peripheral chip may be connected to each PCS pin. Care must be taken by the system designer not to exceed the maximum drive capability of the pins. See the appropriate microcontroller user's manual for electrical specifications.

QSM register PORTQS determines the state of the PCS pins when the QSPI is disabled, and also determines the state of PCS pins that are not assigned to the QSPI when the QSPI is enabled. PORTQS determines the state of pins assigned to the QSPI between transfers as well.

To use a peripheral chip-select pin, the CPU assigns the pin to the QSPI in PQSPAR by writing a one to the appropriate bit. The default value of the PCS pin should be written to PORTQS. Next, the pin must be defined as an output in DDRQS by setting the appropriate bit, which causes the pin to start driving the default value.

The QSPI RAM may then be initialized for a serial transmission, with the peripheral chip-select bits of the command control byte appropriately configured to activate the desired PCS pin(s) during the serial transfer. When the command is executed, the PCS pin(s) are driven to the values contained in the appropriate control byte. After completing the serial transfer, the QSPI returns control of the peripheral chip-select signal(s) (if CONT = 0 in the command control byte) to register PORTQS.

CONT — Continue

1 = Keep peripheral chip-selects asserted after transfer is complete

0 = Return control of peripheral chip-selects to PORTQS after transfer is complete

Some peripheral chips must be deselected between every QSPI transfer. Other chips must remain selected between several sequential serial transfers. CONT is designed to provide the flexibility needed to handle both cases.

If CONT = 1 and the peripheral chip-select pattern for the next command is the same as that of the present command, the QSPI drives the PCS pins to the same value continuously during the two serial transfers. An unlimited number of serial transfers may be sent to the same peripheral(s) without deselecting it (them) by setting CONT = 1.

If CONT = 1 and the peripheral chip-select pattern for the next command is different from that of the present command, the QSPI drives the PCS pins to the new value for the second serial transfer. Although this case is similar to CONT = 0, a difference remains. When CONT = 1, the QSPI continues to drive the PCS pins using the pattern from the first transfer until it switches to using the pattern for the second transfer.

When CONT = 0, the QSPI drives the PCS pins to the values found in register PORTQS between serial transfers.

BITSE — Bits Per Transfer Enable

1 = Number of bits set in BITS field of SPCR0

0 = Eight bits

DT — Delay After Transfer

A/D converters require a known amount of time to perform a conversion. The conversion time for serial CMOS A/D converters may range from 1 – 100 µs.

To facilitate interfacing to peripherals with a latency requirement, the QSPI provides a programmable delay at the end of the serial transfer, with the DT field. The user may avoid using this delay option by executing transfers with other peripheral devices in between transfers with the peripheral that requires a delay. This interleaved operation improves the effective serial transfer rate.

The amount of the delay between transfers is programmable by the user via the DTL field in SPCR1. The range may be set from 1 – 489 µs at 16.78 MHz.

DSCK — PCS to SCK Delay

1 = DSCKL field in SPCR1 specifies value of delay from PCS valid to SCK

0 = PCS valid to SCK transition is 1/2 SCK

4.4 Operating Modes and Flowcharts

The QSPI utilizes an 80-byte block of dual-access static RAM accessible by both the QSPI and the CPU. Because of this dual access capability, up to two wait states may be inserted into CPU access times if the QSPI is in operation.

The RAM is divided into three segments: 16 command control bytes, 16 transmit data words of information to be transmitted, and 16 receive data words for data to be received. Once the CPU has a) set up a queue of QSPI commands, b) written the transmit data segment with information to be sent, and c) enabled the QSPI, the QSPI operates independently of the CPU. The QSPI executes all of the commands in its queue, sets a flag indicating completion, and then either interrupts the CPU or waits for CPU intervention.

The QSPI operates on a queue data structure contained in the QSPI RAM. Control of the queue is handled by three pointers: the new queue pointer (NEWQP), the completed queue pointer (CPTQP), and the end queue pointer (ENDQP). NEWQP, contained in SPCR2, points to the first command in the queue to be executed by the QSPI. CPTQP, contained in SPSR, points to the command last executed by the QSPI. ENDQP, also contained in SPCR2, points to the last command in the queue to be executed by the QSPI, unless wraparound mode is enabled (WREN = 1).

At reset, NEWQP is initialized to $0, causing QSPI execution to begin at queue address $0 when the QSPI is enabled (SPE = 1). CPTQP is set by the QSPI to the queue address ($0-$F) last executed, but is initialized to $0 at reset. ENDQP is also initialized to $0 at reset, but should be changed by the user to reflect the last queue entry to be transferred before enabling the QSPI. Leaving NEWQP and ENDQP set to $0 causes a single transfer to occur when the QSPI is enabled.

The organization of the QSPI RAM requires that one byte of command control data, one word of transmit data, and one word of receive data all correspond to one queue entry, $0-$F.

After executing the current command, ENDQP is checked against CPTQP for an end-of-queue condition. If a match occurs, the SPIF flag is set and the QSPI stops unless wraparound mode is enabled.

The QSPI operates in one of two modes: master or slave. Master mode is used when the MCU originates all data transfers. Slave mode is used when another MCU or a peripheral to the MCU initiates all serial transfers to the MCU via the QSPI. Switching between the two operating modes is achieved under software control by writing to the master (MSTR) bit in SPCR0.

In master mode, the QSPI executes the queue of commands as defined by the control bits in each entry. Chip-select pins are activated; data is transmitted, received, and placed in the QSPI RAM.

In slave mode, a similar operation occurs in response to the slave select ($\overline{SS}$) pin activated by an external SPI bus master. The primary differences are a) no peripheral chip-selects are generated, and b) the number of bits transferred is controlled in a different manner. When the QSPI is selected, it executes the next queue transfer to correctly exchange data with the external device.

The following flowcharts, **Figure 4-3**, **Figure 4-4**, and **Figure 4-5**, outline the operation of the QSPI for both master and slave modes. Note that the CPU must initialize the QSM global and pin registers and the QSPI control registers before enabling the QSPI for either master or slave operation. If using master mode, the necessary command control RAM should also be written before enabling the QSPI. Any data to be transmitted should also be written before the QSPI is enabled. When wrap mode is used, data for subsequent transmissions may be written at any time.

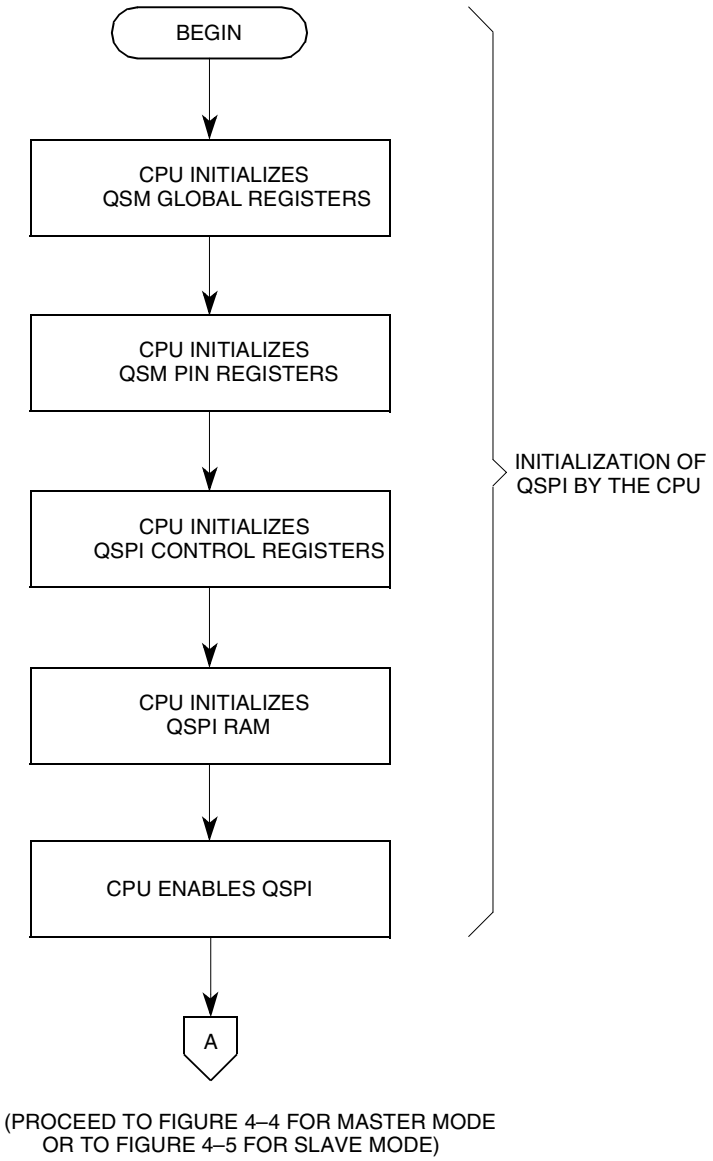


Figure 4-3 Flowchart of QSPI Initialization Operation

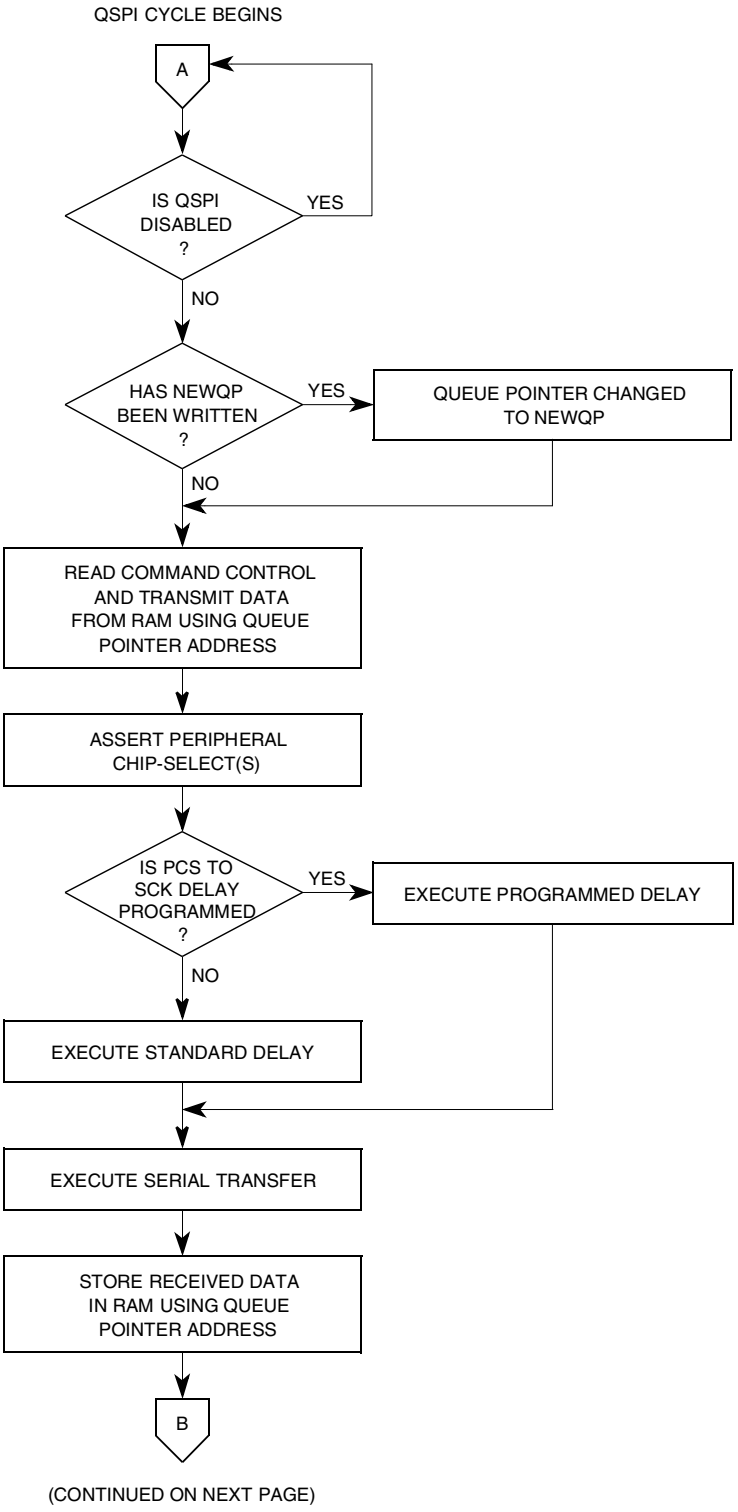


Figure 4-4 Flowchart of QSPI Master Operation (Part 1)

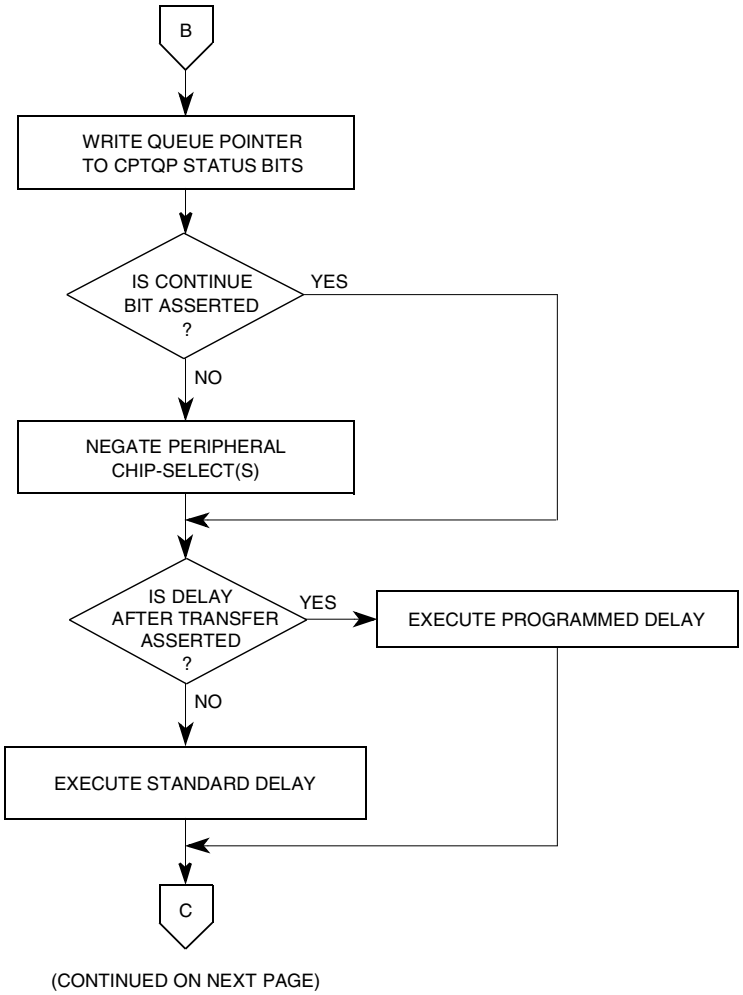


Figure 4-4 Flowchart of QSPI Master Operation (Part 2)

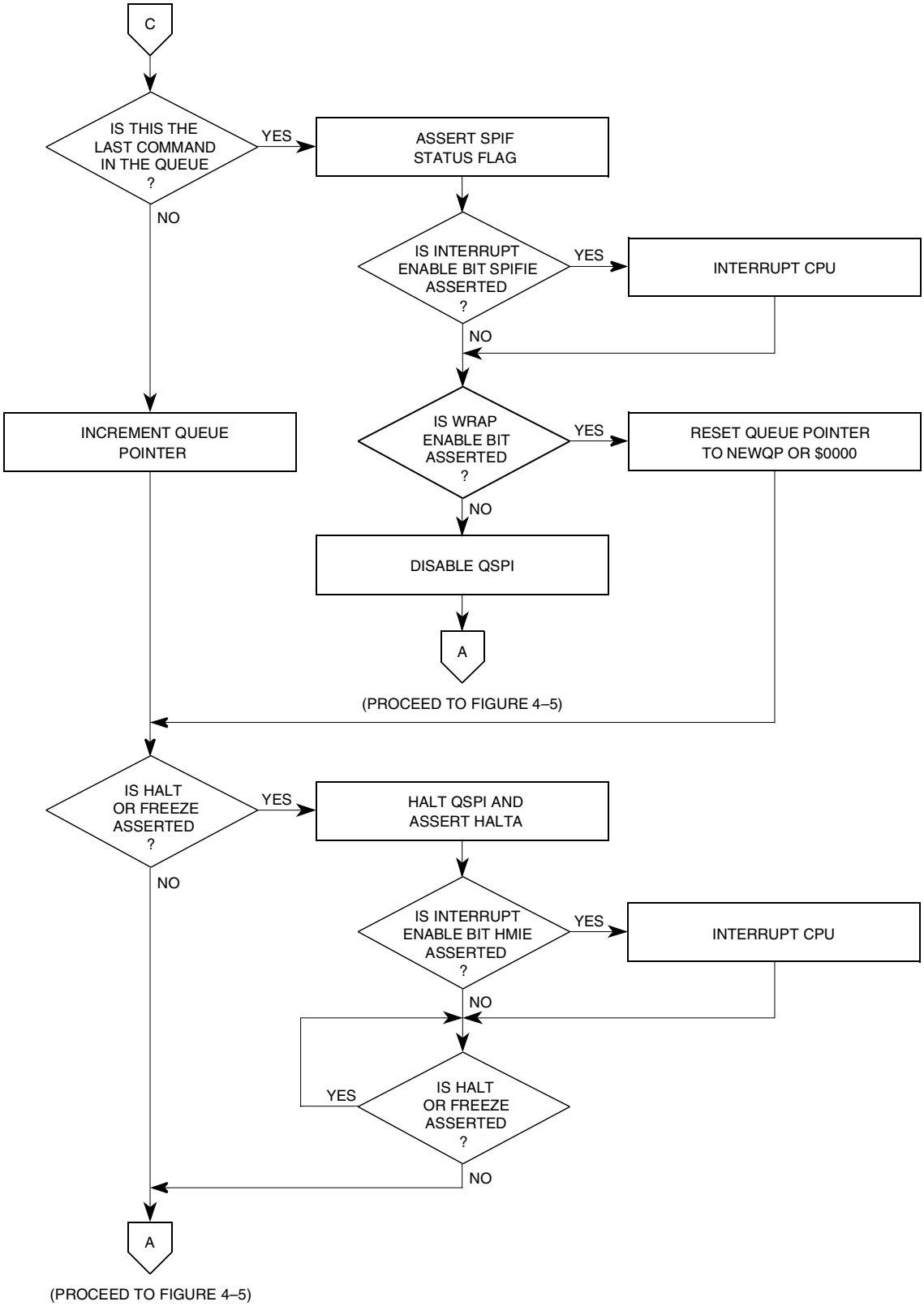


Figure 4-4 Flowchart of QSPI Master Operation (Part 3)

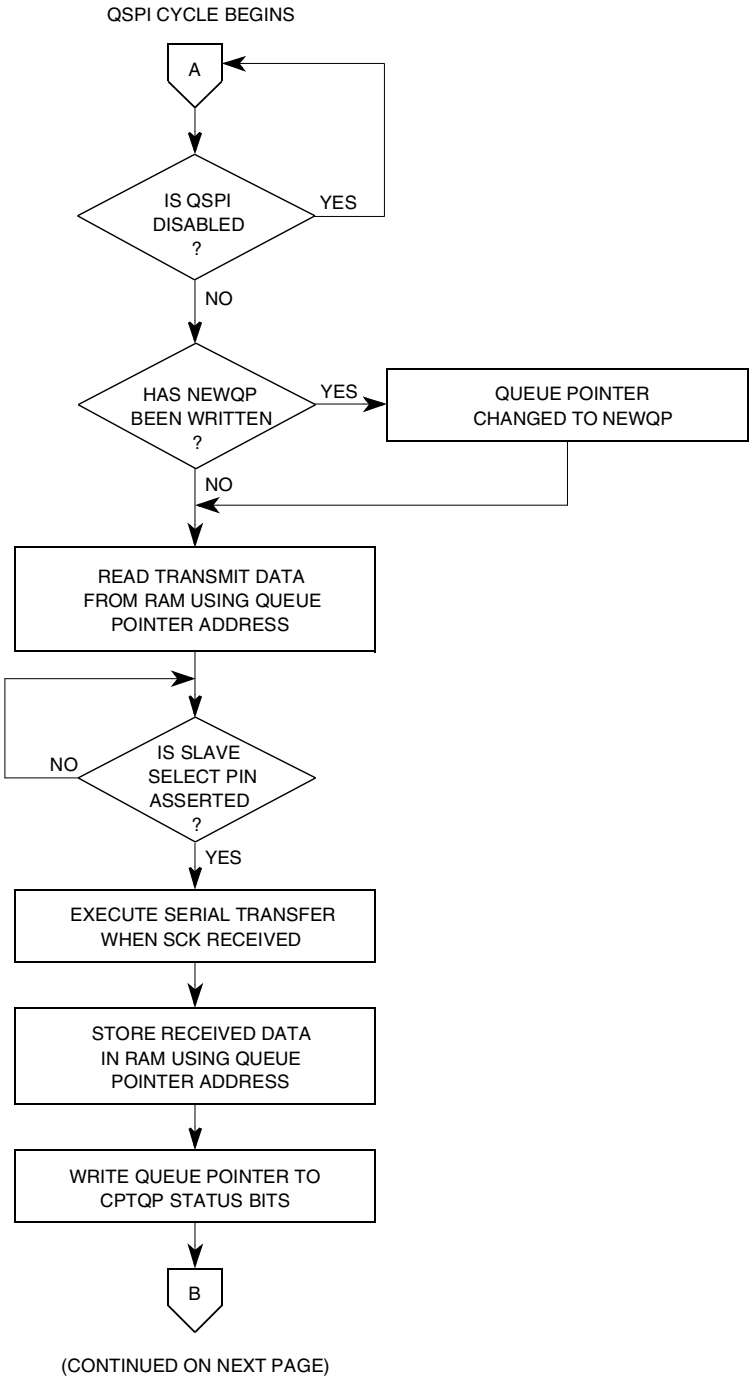


Figure 4-5 Flowchart of QSPI Slave Operation (Part 1)

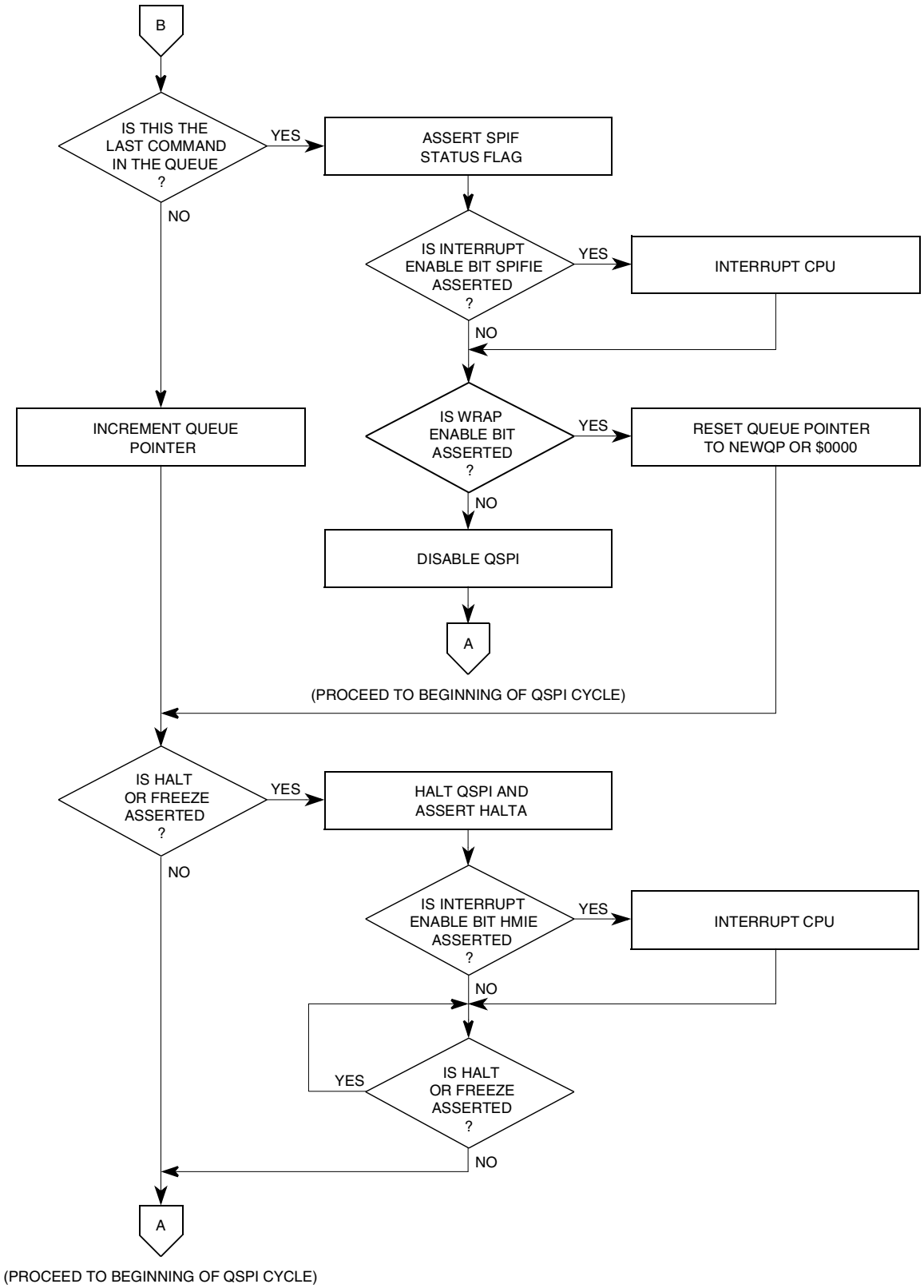


Figure 4-5 Flowchart of QSPI Slave Operation (Part 2)

Although the QSPI inherently supports multimaster operation, no special arbitration mechanism is provided. The user is given a mode fault flag (MODF) to indicate a request for SPI master arbitration; however, the system software must implement the arbitration. Note that unlike previous SPI systems, e.g., on the M68HC11 Family, MSTR is not cleared by a mode fault being set nor are the QSPI pin output drivers disabled; however, the QSPI is disabled when software clears SPE in QSPI register SPCR1.

Normally, the SPI bus performs simultaneous bidirectional synchronous transfers. The serial clock on the SPI bus master supplies the clock signal (SCK) to time the transfer of the bits. Four possible combinations of clock phase and polarity may be employed.

Data is transferred with the most significant bit first. The number of bits transferred per command defaults to eight, but may be programmed to a value from 8–16 bits, using the BITSE field.

Typically, outputs used for the SPI bus are not open-drain unless multiple SPI masters are in the system. If needed, WOMQ in SPCR0 may be set to provide open-drain outputs. An external pull-up resistor should be used on each output bus line. WOMQ affects all QSPI pins regardless of whether they are assigned to the QSPI or used as general-purpose I/O.

4.4.1 Master Mode

When operated in master mode, the QSPI may initiate serial transfers. The QSPI is unable to respond to any externally initiated serial transfers. QSM register DDRQS should be written to direct the data flow on the QSPI pins used. The SCK pin should be configured as an output. Pins MOSI and PCS3–PCS0/$\overline{SS}$ should be configured as outputs as necessary. MISO should be configured as an input if necessary.

QSM register PQSPAR should be written to assign the necessary bits to the QSPI. The pins necessary for master mode operation are MISO and/or MOSI, SCK, and one or more of the PCS pins, depending on the number of external peripheral chips to be selected. MISO is used as the data input pin in master mode, and MOSI is used as the data output pin in master mode. Either or both may be necessary, depending on the particular application. SCK is the serial clock output in master mode.

PCS[3:0]/$\overline{SS}$ are the select pins used to select external SPI peripheral chips for a serial transfer initiated by the QSPI. These pins operate as either active-high or active-low chip-selects. Other considerations for initialization are prescribed in **3.1 Overall QSM Configuration Summary**.

4.4.1.1 Master Mode Operation

After reset, the QSM registers and the QSPI control registers must be initialized as described above. In addition to the command control segment, the transmit data segment may, depending upon the application, need to be initialized. If meaningful data is to be sent out from the QSPI, the user should write the data to the transmit data segment before enabling the QSPI.

Shortly after SPE is set, the QSPI commences operation at the address indicated by NEWQP. The QSPI transmits the data found in the transmit data segment at the address indicated by NEWQP, and the QSPI stores received data in the receive data segment at the address indicated by NEWQP. Data is transferred synchronously with the internally generated SCK.

Transmit data is loaded into the data serializer (refer to Figure 4-1). The QSPI employs control bits, CPHA and CPOL, to determine which SCK edge the MISO pin uses to latch incoming data and which edge the MOSI pin uses to start driving the outgoing data. SPBR of SPCR0 determines the baud rate of SCK. DSCK DSCK and DSCKL determine any peripheral chip-selects valid to SCK start delay.

The number of bits transferred is determined by BITSE and BITS fields. Two options are available: the user may use the default value of 8 bits, or the user may program the length from 8 – 16 bits, inclusive.

Once the proper number of bits are transferred, the QSPI stores the received data in the receive data segment, stores the internal working queue pointer value in CPTQP, increments the internal working queue pointer, and loads the next data required for transfer from the queue. The internal working queue pointer address is the next command executed unless the CPU writes a new value first.

If CONT is set and the peripheral chip-select pattern does not change between the current and the pending transfer, the PCS pins are continuously driven in their designated state during and between both serial transfers. If the peripheral chip-select pattern changes, then the first pattern is driven out during execution of the first transfer, followed by the QSPI switching to the next pattern of the second transfer when execution of the second transfer begins. If CONT is clear, the deselected peripheral chip-select values (found in register PORTQS) are driven out between transfers.

DT causes a delay to occur after the specified serial transfer is completed. The length of the delay is determined by DTL. When DT is clear, the standard delay (1 µs at a 16.78-MHz system clock) occurs after the specified serial transfer is completed.

4.4.1.2 Master Wraparound Mode

When the QSPI reaches the end of the queue, it always sets the SPIF flag whether wraparound mode is enabled or disabled. An optional interrupt to the CPU is generated when SPIF is asserted. At this point, the QSPI clears SPE and stops unless wraparound mode is enabled. A description of SPIFIE may be found in **4.3.3 QSPI Control Register 2 (SPCR2)**.

In wraparound mode, the QSPI cycles through the queue continuously. Each time the end of the queue is reached, the SPIF flag is set. If the CPU fails to clear SPIF it remains set, and the QSPI continues to send interrupt requests to the CPU (assuming SPIFIE is set). The user may avoid causing CPU interrupts by clearing SPIFIE. As SPIFIE is buffered, clearing it after the SPIF flag is asserted does not immediately stop the CPU interrupts, but only prevents future interrupts from this source. To clear the current interrupt, the CPU must read QSPI register SPSR SPSR with SPIF asserted, followed by a write to SPSR with a zero in SPIF (clear SPIF).

Execution continues in wraparound mode, even while the QSPI is requesting interrupt service from the CPU. The internal working queue pointer increments to the next address, and the commands are executed again. SPE is not cleared by the QSPI. New receive data overwrites previously received data in the receive data segment.

Wraparound mode is properly exited in two ways: a) The CPU may disable wraparound mode by clearing WREN. The next time the end of the queue is reached, the QSPI sets SPIF, clears SPE, and stops; b) The CPU sets HALT. This second method halts the QSPI after the current transfer is completed, allowing the CPU to negate SPE. The CPU can immediately stop the QSPI by clearing SPE; however, this method is not recommended as it causes the QSPI to abort a serial transfer in process.

4.4.2 Slave Mode

When operating in slave mode, the QSPI may respond to externally initiated serial transfers. The QSPI is unable to initiate any serial transfers. Slave mode is typically used when multiple MCUs are in an SPI bus network, because only one device can be the SPI master (in master mode) at any given time.

QSM register DDRQS should be written to direct data flow on the QSPI pins used. The MISO and MOSI pins, if needed, should be configured as output and input, respectively. Pins SCK and PCS0/$\overline{SS}$ should be configured as inputs.

QSM register PQSPAR should be written to assign the necessary bits to the QSPI. The pins necessary for slave mode operation are MISO and/or MOSI, SCK, and PCS0/$\overline{SS}$. MISO is the data output pin in slave mode, and MOSI is the data input pin in slave mode. Either or both may be necessary depending on the particular application.

The serial clock (SCK) is the slave clock input in slave mode. PCS0/$\overline{SS}$ is the slave select pin used to select the QSPI for a serial transfer by the external SPI bus master when the QSPI is in slave mode. The external bus master selects the QSPI by driving PCS0/$\overline{SS}$ low.

When the MISO pin is configured for QSPI use (MISO bit in PQSPAR = 1) and the QSPI is set up for slave mode (MSTR bit in SPCR0 = 0) the MISO pin can be in a high-impedance state (three-stated). This occurs while the $\overline{SS}$ pin is at a logic level one. This overrides the MISO bit in the DDRQS if it is set to be an output. The MISO pin becomes active when $\overline{SS}$ is pulled low.

The command control segment is not implemented in slave mode; therefore, the CPU does not need to initialize it. This segment of the QSPI RAM and any other unused segments may be employed by the CPU as general-purpose RAM. Other considerations for initialization are prescribed in **3.1 Overall QSM Configuration Summary.**

4.4.2.1 Description of Slave Operation

After reset, the QSM registers and the QSPI control registers must be initialized as described above. Although the command control segment is not used, the transmit and receive data segments may, depending upon the application, need to be initialized. If

meaningful data is to be sent out from the QSPI, the user should write the data to the transmit data segment before enabling the QSPI.

If SPE is set and MSTR is not set, a low state on the slave select (PCS0/$\overline{SS}$) pin commences slave mode operation at the address indicated by NEWQP. The QSPI transmits the data found in the transmit data segment at the address indicated by NEWQP, and the QSPI stores received data in the receive data segment at the address indicated by NEWQP. Data is transferred in response to an external slave clock input at the SCK pin.

Because the command control segment is not used, the command control bits and peripheral chip-select codes have no effect in slave mode operation. The QSPI does not drive any of the four peripheral chip-selects as outputs. PCS0/$\overline{SS}$ is used as an input.

Although CONT cannot be used in slave mode, a provision is made to enable receipt of more than 16 data bits. While keeping the QSPI selected (PCS0/$\overline{SS}$ is held low), the QSPI stores the number of bits, designated by BITS, in the current receive data segment address, increments NEWQP, and continues storing the remaining bits (up to the BITS value) in the next receive data segment address.

As long as PCS0/$\overline{SS}$ remains low, the QSPI continues to store the incoming bit stream in sequential receive data segment addresses, until either the value in BITS is reached or the end-of-queue address is used with wraparound mode disabled. When the end of the queue is reached, the SPIF flag is asserted, optionally causing an interrupt. If wraparound mode is disabled, any additional incoming bits are ignored. If wraparound mode is enabled, storing continues at either address $0 or the address of NEWQP, depending on the WRTO value.

When using this capability to receive a long incoming data stream, the proper delay between transfers must be used. The QSPI requires time, approximately 1 µs at 16.78-MHz system clock, to prefetch the next transmit RAM entry for the next transfer. Therefore, the user may select a baud rate that provides at least a 1 µs delay between successive transfers to ensure no loss of incoming data. If the system clock is operating at a slower rate, the delay between transfers must be increased proportionately.

Because the BITSE option in the command control segment is no longer available, BITS sets the number of bits to be transferred for all transfers in the queue until the CPU changes the BITS value. As mentioned above, until PCS0/$\overline{SS}$ is negated (brought high), the QSPI continues to shift one bit for each pulse of SCK. If PCS0/$\overline{SS}$ is negated before the proper number of bits (according to BITS) is received, the QSPI, the next time it is selected, resumes storing bits in the same receive data segment address where it left off. If more than 16 bits are transferred before negating the PCS0/$\overline{SS}$, the QSPI stores the number of bits indicated by BITS in the current receive data segment address, then increments the address and continues storing as described above. Note that PCS0/$\overline{SS}$ does not necessarily have to be negated between transfers.

Once the proper number of bits (designated by BITS) are transferred, the QSPI stores the received data in the receive data segment, stores the internal working queue point-

er value in CPTQP, increments the internal working queue pointer, and loads the new transmit data from the transmit data segment into the data serializer. The internal working queue pointer address is used the next time PCS0/$\overline{SS}$ is asserted, unless the CPU writes to the NEWQP first.

The DT and DSCK command control bits are not used in slave mode. As a slave, the QSPI does not drive the clock line nor the chip-select lines and, therefore, does not generate a delay.

In slave mode, the QSPI shifts out the data in the transmit data segment. The transmit data is loaded into the data serializer (refer to **Figure 5-9**) for transmission. When the PCS0/$\overline{SS}$ pin is pulled low the MISO pin becomes active and the serializer then shifts the 16 bits of data out in sequence, most significant bit first, as clocked by the incoming SCK signal. The QSPI uses CPHA and CPOL to determine which incoming SCK edge the MOSI pin uses to latch incoming data, and which edge the MISO pin uses to drive the data out.

The QSPI transmits and receives data until reaching the end of the queue (defined as a match with the address in ENDQP), regardless of whether PCS0/$\overline{SS}$ remains selected or is toggled between serial transfers. Receiving the proper number of bits causes the received data to be stored. The QSPI always transmits as many bits as it receives at each queue address, until the BITS value is reached or PCS0/$\overline{SS}$ is negated.

4.4.2.2 Slave Wraparound Mode

When the QSPI reaches the end of the queue, it always sets the SPIF flag, whether wraparound mode is enabled or disabled. An optional interrupt to the CPU is generated when SPIF is asserted. At this point, the QSPI clears SPE and stops unless wraparound mode is enabled. A description of SPIFIE bit can be found in **4.3.3 QSPI Control Register 2 (SPCR2)**.

In wraparound mode, the QSPI cycles through the queue continuously. Each time the end of the queue is reached, the SPIF flag is set. If the CPU fails to clear SPIF, it remains set, and the QSPI continues to send interrupt requests to the CPU (assuming SPIFIE is set). The user may avoid causing CPU interrupts by clearing SPIFIE. As SPIFIE is buffered, clearing it after the SPIF flag is asserted does not immediately stop the CPU interrupts, but only prevents future interrupts from this source. To clear the current interrupt, the CPU must read QSPI register SPSR with SPIF asserted, followed by a write to SPSR with zero in SPIF (clear SPIF). Execution continues in wraparound mode even while the QSPI is requesting interrupt service from the CPU. The internal working queue pointer is incremented to the next address and the commands are executed again. SPE is not cleared by the QSPI. New receive data overwrites previously received data located in the receive data segment.

Wraparound mode is properly exited in two ways: a) The CPU may disable wraparound mode by clearing WREN. The next time end of the queue is reached, the QSPI sets SPIF, clears SPE, and stops; and, b) The CPU sets HALT. This second method halts the QSPI after the current transfer is completed, allowing the CPU to negate SPE. The CPU can immediately stop the QSPI by clearing SPE; however, this method is not recommended, as it causes the QSPI to abort a serial transfer in process.

SECTION 5
SCI SUBMODULE

The SCI submodule is used to communicate with external devices and other MCUs via an asynchronous serial bus. The SCI is fully compatible with the SCI systems found on other Motorola MCUs, such as the M68HC11 and M68HC05 Families. It has all of the capabilities of previous SCI systems as well as several significant new features. The following paragraphs describe the features, pins, programmer's model (memory map), registers, and the transmit and receive operations of the SCI.

5.1 Features

Standard SCI features are listed below, followed by a list of additional features offered.

Standard SCI Two-Wire System Features:

- Standard Nonreturn-to-Zero (NRZ) Mark/Space Format
- Advanced Error Detection Mechanism (detects noise duration up to 1/16 of a bit-time)
- Full-Duplex Operation
- Software Selectable Word Length (8- or 9-bit words)
- Separate Transmitter and Receiver Enable Bits
- May be Interrupt Driven
- Four Separate Interrupt Enable Bits

Standard SCI Receiver Features:

- Receiver Wakeup Function (idle or address mark bit)
- Idle-Line Detect
- Framing Error Detect
- Noise Detect
- Overrun Detect
- Receive Data Register Full Flag

Standard SCI Transmitter Features:

- Transmit Data Register Empty Flag
- Transmit Complete Flag
- Send Break

QSM-Enhanced SCI Two-Wire System Features:

- 13-Bit Programmable Baud-Rate Modulus Counter
  - — A baud rate modulus counter has been added to provide the user with more flexibility in choosing the crystal frequency for the system clock. The modulus counter allows the SCI baud rate generator to produce standard transmission frequencies for a wide range of system clocks. The user is no longer constrained to select crystal frequencies based on the desired serial baud rate. This counter provides baud rates from 64 baud to 524 kbaud with a 16.78-MHz system clock.

- Even/Odd Parity Generation and Detection
  — The user now has the choice either of seven or eight data bits plus one parity bit, or of eight or nine data bits with no parity bit. Even or odd parity is available. The transmitter automatically generates the parity bit for a transmitted byte. The receiver detects when a parity error has occurred on a received byte and sets a parity error flag.

QSM-Enhanced SCI Receiver Features:

- Two Idle-Line Detect Modes
  — Standard Motorola SCI systems detect an idle line when 10 or 11 consecutive bit-times are all ones. Used with the receiver wakeup mode, the receiver can be awakened prematurely if the message preceding the start of the idle line contained ones in advance of its stop bit. The new (second) idle-line detect mode starts counting idle time only after a valid stop bit is received, which ensures correct idle-line detection.
- Receiver Active Flag (RAF)
  — RAF indicates the status of the receiver. It is set when a possible start bit is detected and is cleared when an idle line is detected. RAF is also cleared if the start bit is determined to be line noise. This flag can be used to prevent collisions in systems with multiple masters.

5.2 SCI Programmer's Model and Registers

The programmer's model (memory map) for the SCI submodule consists of the QSM global and pin control registers (refer to **3.2 QSM Global Registers** and **3.3 QSM Pin Control Registers**) and the four SCI registers. The SCI registers are listed in **Table 5-1** and consist of two control registers, one status register, and one data register. All registers may be read or written at any time by the CPU. Rewriting the same value to any SCI register does not disrupt operation; however, writing a different value into an SCI register when the SCI is running may disrupt operation. To change register values, the receiver and transmitter should be disabled with the transmitter allowed to finish first. The status flags in register SCSR may be cleared at any time.

Table 5-1 SCI Register

Address	Name	Usage
$YFFC08	SCCR0	SCI Control Register 0
$YFFC0A	SCCR1	SCI Control Register 1
$YFFC0C	SCSR	SCI Status Register
$YFFC0E	SCDR	SCI Data Register Transmit Data Register (TDR)* Receive Data Register (RDR)*

*Reads access the RDR; writes access the TDR.

When initializing the SC, the SCCR1 has two bits that should be written last; the transmitter enable (TE) and receiver enable (RE) bits, which enable the SCI. Registers SCCR0 and SCCR1 should both be initialized at the same time or before TE and RE are asserted. A single word write to SCCR1 can be used to initialize the SCI and enable the transmitter and receiver.

Figure 5-1 and **Figure 5-2** show the block diagrams for the SCI receiver and transmitter.

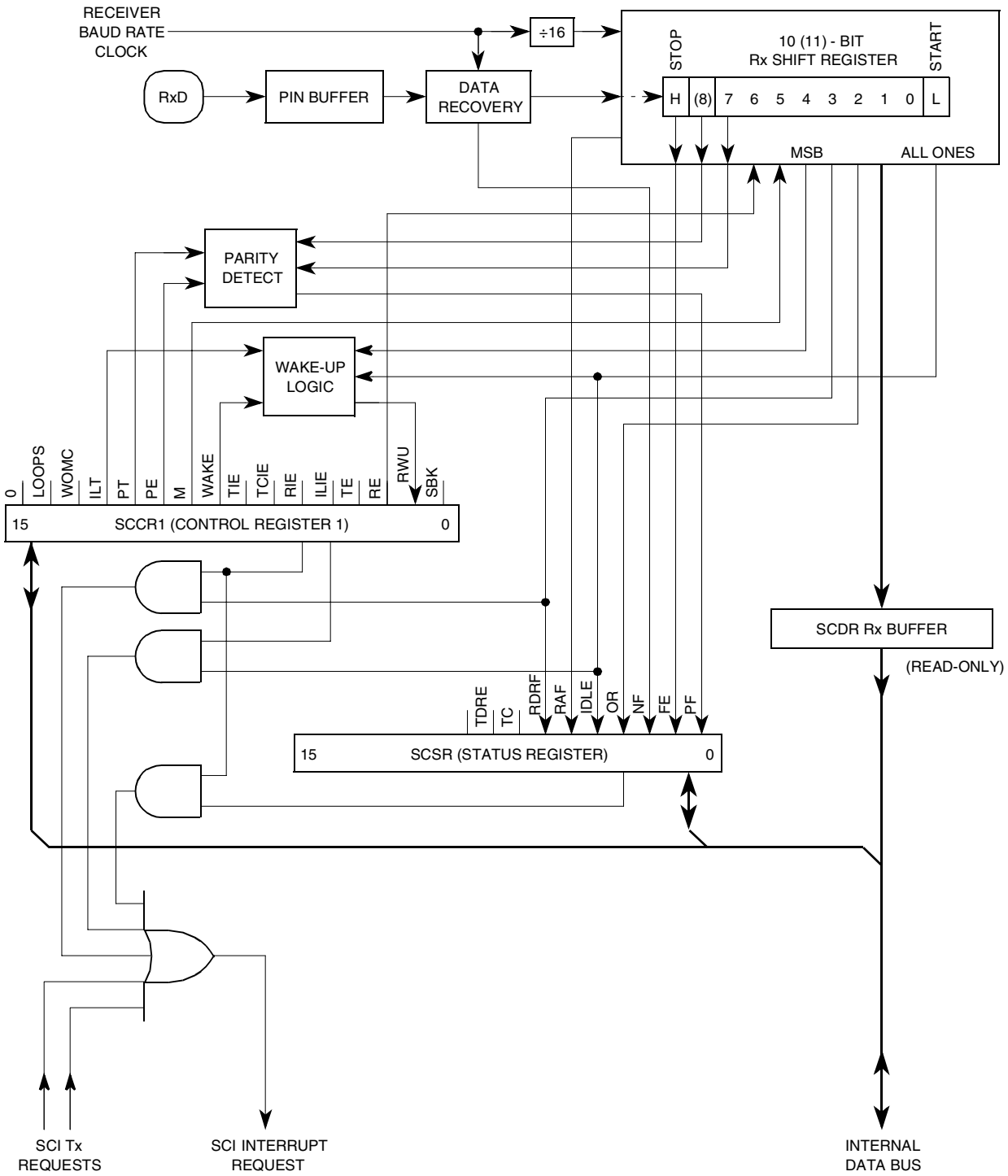


Figure 5-1 SCI Receiver Block Diagram

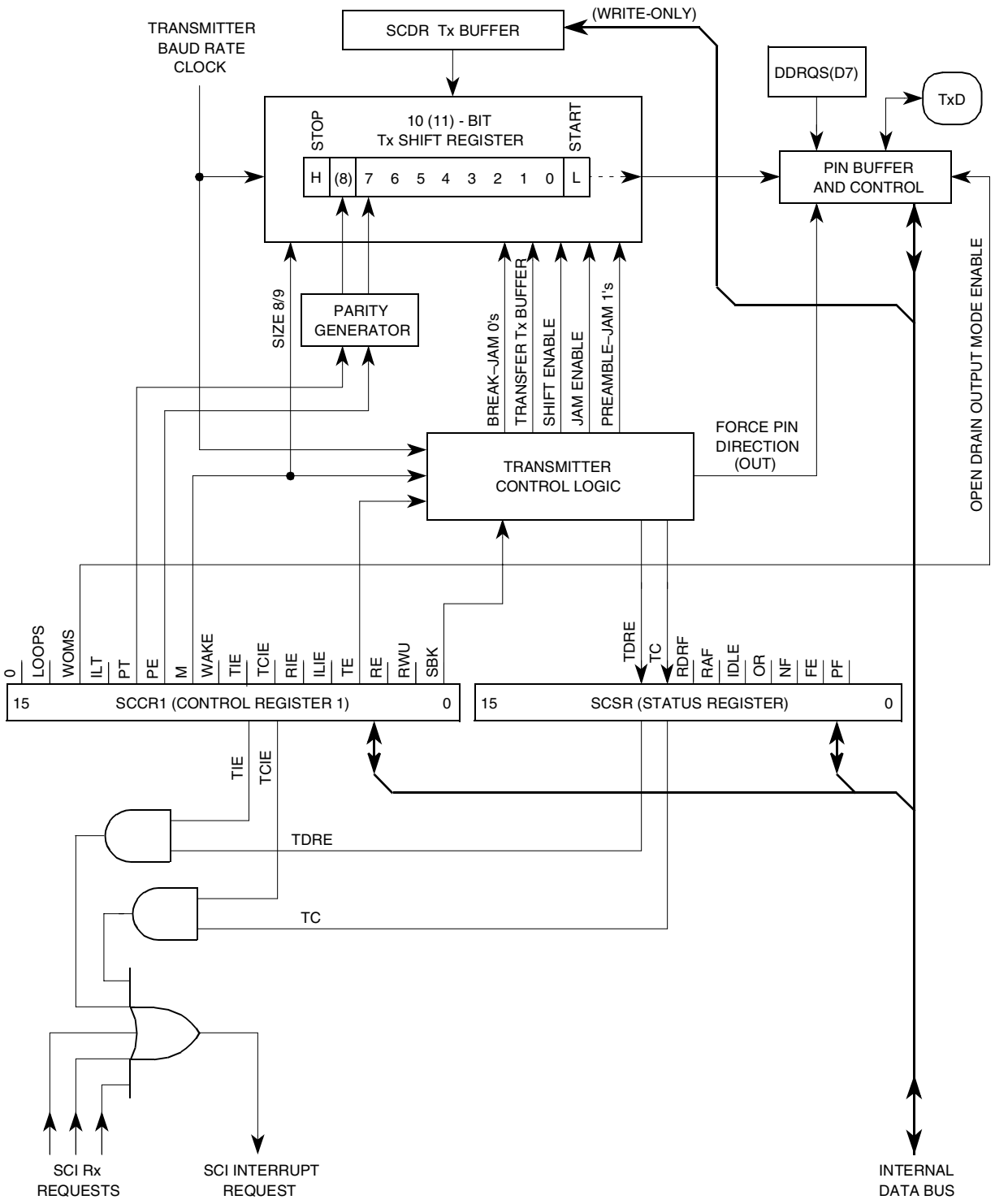


Figure 5-2 SCI Transmitter Block Diagram

5.2.1 SCI Control Register 0 (SCCR0)

SCCR0 contains the parameter for configuring the SCI baud rate. The baud rate should be set before the SCI is enabled. The CPU can read and write this register at any time.

SCCR0 — SCI Control Register 0 **$YFFC08**

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

Bits [15:13] — Not Implemented

SCBR — Baud Rate

The SCI baud rate is programmed by writing a 13-bit value to SCBR and is derived from the MCU system clock using a modulus counter.

The SCI receiver operates asynchronously. Therefore, the SCI requires an internal clock to synchronize itself to the incoming data stream. The SCI baud-rate generator produces a receiver sampling clock with a frequency 16 times that of the expected baud rate of the incoming data. From transitions within the received waveform, the SCI determines the most likely position of the bit boundaries and adjusts sampling points to the proper positions within the bit period. The receiver sampling rate is always 16 times the frequency of the SCI baud rate, which is calculated using the following equation:

$$\text{SCI Baud} = \text{System Clock}/(32 * \text{SCBR}) \quad (5\text{-}1)$$

where SCBR equals {1, 2, 3,... 8191}. Note that zero is a disallowed value for SCBR.

Writing a value of zero to SCBR disables the baud rate generator. There are 8191 different bauds available. The baud value depends on the value for SCBR and the system clock, as used in the above equation. **Table 5-2** shows possible baud rates for a 16.78-MHz system clock. The maximum baud rate with this system clock speed is 524 kbaud.

Table 5-2 Examples of SCI Baud Rates

Nominal Baud Rate	Actual Baud Rate	Percent Error	Value ofSCBR
500,000.00	524,288.00	4.86	1
38,400.00	37,449.14	–2.48	14
32,768.00	32,768.00	0.00	16
19,200.00	19,418.07	1.14	27
9,600.00	9,532.51	–0.70	55
4,800.00	4,809.98	0.21	109
2,400.00	2,404.99	0.21	218
1,200.00	1,199.74	–0.02	437
600.00	599.87	–0.02	874
300.00	299.94	–0.02	1,748
110.00	110.01	0.01	4,766
64.00	64.00	0.01	8,191

NOTE: These rates are based on a 16.78-MHz system clock.

More accurate baud rates can be obtained by varying the system clock frequency with the VCO synthesizer. Each VCO speed increment adjusts the baud rate up or down by 1/64; or 1.56%.

5.2.2 SCI Control Register 1 (SCCR1)

SCCR1 contains parameters for configuration of the SCI. The CPU can read and write this register at any time. The SCI may modify the RWU bit in some circumstances. In general, the interrupts enabled by these control bits are cleared by reading the status register SCSR, followed by reading (for receiver status bits) or by writing (for transmitter status bits) the data register SCDR. For further detail refer to **5.3 Transmitter Operation** and **5.4 Receiver Operation**, respectively.

SCCR1 — SCI Control Register 1 **$YFFC0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 15 — Not Implemented

LOOPS — LOOP Mode

1 = Test SCI operation, looping, feedback path enabled

0 = Normal SCI operation, no looping, feedback path disabled

LOOPS controls a feedback path on the data serial shifter. If enabled, the output of the SCI transmitter is fed back into the receive serial shifter as receiver input, and no data is driven out of the TXD pin nor is data received from the RXD pin. The TXD pin is driven high (idle line). Both the transmitter and receiver must be enabled for loop mode to function.

WOMS — Wired-OR Mode for SCI Pins

1 = If configured as an output, TXD is an open-drain output.

0 = If configured as an output, TXD is a normal CMOS output.

WOMS determines whether the TXD pin is an open-drain output or a normal CMOS output. This bit is used only when TXD is an output. If the TXD pin is being used as a general-purpose input pin, WOMS has no effect.

ILT — Idle-Line Detect Type

1 = Long idle-line detect (starts counting when the first one is received after a stop bit(s))

0 = Short idle-line detect (starts counting when the first one is received)

ILT determines which one of two types of idle-line detection is to be used by the SCI receiver. The short idle-line detection circuitry causes the SCI receiver to start counting ones at any point (even during the frame), which means that the stop bit and any contiguous one data bits at the end of the last byte are counted toward the 10 or 11 ones in an idle frame. Hence, the data content of the last byte transmitted may affect the timing of idle-line detection.

The long idle-line detection circuitry causes the SCI receiver to start counting ones right after a stop bit, which means that the stop bit and any contiguous one data bits

in a previous data byte are not counted toward the 10 or 11 ones in an idle line. Hence, the data content of the last byte transmitted does not affect the timing of idle-line detection.

PT — Parity Type

1 = Odd parity

If the data contains an even number of ones, then the parity bit equals one. If the data contains an odd number of ones, then the parity bit equals zero.

0 = Even parity

If the data contains an even number of ones, then the parity bit equals zero. If the data contains an odd number of ones, then the parity bit equals one.

When parity is enabled, PT determines whether parity is even or odd for both the receiver and the transmitter.

PE — Parity Enable

1 = SCI parity enabled; the transmitter generates the parity bit and the receiver checks incoming parity.

0 = SCI parity disabled

PE determines whether parity is enabled or disabled for both the receiver and the transmitter. If PE is set, the transmitter internally generates the parity bit and appends it to the data bits during transmission. The receiver checks the last bit before a stop bit to determine if the correct parity was received. If the received parity bit is not correct, the SCI sets the PF error flag in SCSR.

When PE is set, the most significant bit (MSB) of the data field is used for the parity function, which results in either seven or eight bits of user data, depending on the condition of M bit. **Table 5-3** lists the available choices.

Table 5-3 M and PE Bit Fields

M	PE	Result
0	0	8 Data Bits
0	1	7 Data Bits, 1 Parity Bit
1	0	9 Data Bits
1	1	8 Data Bits, 1 Parity Bit

M — Mode Select

1 = SCI frame: one start bit, nine data bits, one stop bit (eleven bits total)

0 = SCI frame: one start bit, eight data bits, one stop bit (ten bits total)

The M bit determines the SCI frame format. If M is clear (its reset value), the frame format is one start bit, eight data bits, one stop bit. If M is set, the frame format is one start bit, nine data bits, one stop bit.

The ninth data bit can be controlled by software to perform a function such as address mark. Frames with the ninth data bit set could be identified as an address mark. All receivers in a network could be placed in wakeup mode until an address mark is detected, at which time all receivers would wake up and read the address. All receivers being addressed could continue to receive the following message, while all receivers not being addressed could be put back into wakeup mode.

The ninth data bit could also serve as a second stop bit. By setting this bit permanently to one, communication with other SCIs requiring two stop bits could be accommodated.

Note that only 10 or 11 bits in a frame are allowed. If parity is to be enabled, the last data bit must be used for this purpose. The parity bit may be odd, even, mark, or space. Parity and address (control) bits are mutually exclusive. A choice must be made between one or the other, or neither. Every frame must have one start bit and at least one stop bit. The possible combinations are given in the bit description of PE.

WAKE — Wakeup by Address Mark

1 = SCI receiver awakened by address mark (eighth or ninth (last) bit set)
0 = SCI receiver awakened by idle-line detection

WAKE determines which one of two conditions wakes up the SCI receiver when it is in wakeup mode. If WAKE is clear (its reset value), the detection of an idle line (10 or 11 contiguous ones) which clears RWU causes the SCI receiver to wake up. If WAKE is set, the detection of an address mark (the last data bit of a frame is set) which clears RWU causes the SCI receiver to wake up.

TIE — Transmit Interrupt Enable

1 = SCI TDRE interrupts enabled
0 = SCI TDRE interrupts inhibited

When set, TIE enables an SCI interrupt whenever the TDRE flag in SCSR is set. The interrupt is blocked by negating TIE.

TCIE — Transmit Complete Interrupt Enable

1 = SCI TC interrupts enabled
0 = SCI TC interrupts inhibited

When set, TCIE enables an SCI interrupt whenever the TC flag in SCSR is set. The interrupt may be cleared by reading SCSR when TC is set and then by writing the transmit data register (TDR) of SCDR. The interrupt is blocked by negating TCIE.

RIE — Receiver Interrupt Enable

1 = SCI RDRF interrupts enabled
0 = SCI RDRF interrupts inhibited

When set, RIE enables an SCI interrupt whenever the RDRF flag in SCSR is set. The interrupt is blocked by negating RIE.

ILIE — Idle-Line Interrupt Enable

1 = SCI IDLE interrupts enabled
0 = SCI IDLE interrupts inhibited

When set, ILIE enables an SCI interrupt whenever the IDLE flag in SCSR is set. The interrupt is blocked by negating ILIE.

TE — Transmitter Enable

1 = SCI transmitter enabled, TXD pin dedicated to the SCI transmitter
0 = SCI transmitter disabled, TXD pin may be used as general-purpose I/O

When set, TE enables the SCI transmitter and assigns it to the TXD pin. When TE is clear, the TXD pin may be used for general-purpose I/O. An idle frame, called a pre-

amble, consisting of ten (or eleven) contiguous ones, is automatically transmitted whenever TE is changed from zero to one. Refer to **5.3 Transmitter Operation** for a detailed description of TE and the SCI transmit operation.

RE — Receiver Enable

1 = SCI receiver enabled
0 = SCI receiver disabled

RE enables the SCI receiver when set. When disabled, the receiver status bits RDRF, IDLE, OR, NF, FE, and PF are inhibited and are not asserted by the SCI. Refer to **5.4 Receiver Operation** for a complete description of RE and the SCI receiver operation.

RWU — Receiver Wakeup

1 = Wakeup mode enabled, all received data ignored until awakened
0 = Normal receiver operation, all received data recognized

Setting RWU enables the wakeup function, which allows the SCI to ignore received data until awakened by either an idle line or address mark (as determined by WAKE). When in wakeup mode, the receiver status flags are not set, and interrupts are inhibited. This bit is cleared automatically (returned to normal mode) when the receiver is awakened.

SBK — Send Break

1 = Break frame(s) transmitted after completion of the current frame
0 = Normal operation

SBK provides the ability to transmit a break code (ten or eleven contiguous zeros) from the SCI. When SBK is set, the SCI completes the current frame transmission (if it is transmitting) and then begins transmitting continuous frames of ten (or eleven) zeros until SBK is cleared. If SBK is toggled by writing it first to a one and then immediately to a zero (in less than one serial frame interval), the transmitter sends only one or two break frames before reverting to mark (idle line) or before commencing to send data. SBK is normally used to broadcast the termination of a transmission.

5.2.3 SCI Status Register (SCSR)

SCSR contains flags that the SCI sets to inform the user of various operational conditions. These flags are automatically cleared either by hardware or by a special acknowledgment sequence consisting of an SCSR read (either the upper byte, the lower byte, or the entire word) with a flag bit set, followed by a read (or write in the case of flags TDRE and TC) of data register SCDR (either the lower byte, or the entire word). An upper byte access of SCDR is only meaningful for reads. Note that a long-word read can consecutively access both registers SCSR and SCDR. This action clears the receive status flag bits that were set at the time of the read, but does not clear the TDRE or TC flags. To clear TDRE or TC, the SCSR read must be followed by a write to register SCDR (either the lower byte or the entire word).

If an internal SCI signal for setting a status bit comes after the CPU has read the asserted status bits but before the CPU has written or read register SCDR, the newly set status bit is not inadvertently cleared. Instead, register SCSR must be read again with the status bit set, and register SCDR must be written or read before the status bit is cleared.

NOTE

None of the status bits are cleared by reading a status bit while it is asserted and then by writing zero to that same bit. The procedure outlined above must be followed. Emphasis is also given to note that reading either byte of register SCSR causes all 16 bits to be accessed, and any status bits already set in either byte are armed to clear on a subsequent read or write of register SCDR.

As mentioned, register SCSR co-functions with register SCDR. SCDR is a combination of two data registers: the TDR and the RDR. Each of these data registers has a serial shifter.

SCSR — SCI Status Register **$YFFC0C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF
RESET:															
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Bits [15:9] — Not Implemented

TDRE — Transmit Data Register Empty Flag

1 = A new character may now be written to register TDR

0 = Register TDR still contains data to be sent to the transmit serial shifter

TDRE is set when the byte in register TDR is transferred to the transmit serial shifter. If this bit is zero, the transfer is yet to occur and a write to TDR will overwrite the previous value. New data is not transmitted if TDR is written without first clearing TDRE, which is accomplished by reading register SCSR with TDRE set, followed by a write to TDR. Reset sets this bit.

TC — Transmit Complete Flag

1 = SCI transmitter is idle

0 = SCI transmitter is busy

TC is set when the transmitter finishes shifting out all data, queued preambles (mark/idle line), or queued breaks (logic zero). TC is cleared when SCSR is read with TC set, followed by a write to register TDR.

RDRF — Receive Data Register Full Flag

1 = Register RDR contains new data

0 = Register RDR is empty or contains previously read data

RDRF is set when the content of the receive serial shifter is transferred to register RDR. If one or more errors are detected in the received word, the appropriate receive-related flag(s) NF, FE, and/or PF are set within the same clock cycle. RDRF is cleared when register SCSR is read with RDRF set, followed by a read of register RDR.

RAF — Receiver Active Flag

1 = SCI receiver is busy

0 = SCI receiver is idle

RAF indicates whether the SCI receiver is busy. This flag is set when the SCI receiver detects a possible start bit and is cleared when the chosen type of idle line is detected. RAF can be used to reduce collisions in systems with multiple masters.

The SCI receiver samples each start bit 16 times (at a rate of 16 times the baud rate). The 16 sample times are called RT1–RT16. RAF is set initially at RT1. The SCI receiver samples RT3, RT5, and RT7. If the receiver line is high during two or three of the three receive time (RT) samples, the start bit is considered invalid, and RAF is subsequently cleared. A more detailed description is found in **5.4.1 Receiver Bit Processor**.

IDLE — Idle-Line Detected Flag

1 = SCI receiver detected an idle-line condition

0 = SCI receiver did not detect an idle-line condition

IDLE is set when the SCI receiver detects an idle-line condition (reception of a minimum of ten or eleven consecutive ones as specified by ILT in SCCR1). This bit is not set by the idle-line condition when RWU in SCCR1 is set. Once cleared, IDLE is not set again until after RDRF is set (after the line is active and becomes idle again). If a break is received, RDRF is set, allowing a subsequent idle line to be detected again. IDLE is cleared when SCSR is read with IDLE set, followed by a read of register RDR.

Under certain conditions, the IDLE flag may be set immediately following the negation of RE (SCCR1). System designs should ensure this causes no detrimental effects.

OR — Overrun Error Flag

1 = RDRF is not cleared before new data arrives

0 = RDRF is cleared before new data arrives

OR is set when a new byte is ready to be transferred from the receive serial shifter to register RDR, and RDR is already full (RDRF is still set). Data transfer is inhibited until OR is cleared. Previous data in RDR remains valid, but additional data received during an overrun condition (including the byte that set OR) is lost.

A difference exists between OR and the other receiver status flags, NF, FE, and PF all reflect the status of data already transferred to register RDR. OR reflects an operational condition that resulted in a loss of data to RDR. OR is cleared when SCSR is read with OR set, followed by a read of register RDR.

NF — Noise Error Flag

1 = Noise occurred on the received data

0 = No noise detected on the received data

NF is set when the SCI receiver detects noise on a valid start bit, on any of the data bits, or on the stop bit(s). It is not set by noise on the idle line or on invalid start bits. Each bit is sampled three times for noise. If the three samples are not at the same logic level, the majority value is used for the received data value, and NF is set. NF is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with NF, an interrupt may be generated with RDRF and NF checked in this manner. NF is cleared when SCSR is read with NF set, followed by a read of register RDR.

FE — Framing Error Flag

1 = Framing error or break occurred on the received data

0 = No framing error on the received data

FE is set when the SCI receiver detects a zero where a stop bit (one) was to occur. A framing error results when the frame boundaries in the received bit stream are not synchronized with the receiver bit counter. FE is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with FE, an interrupt may be generated with RDRF and FE checked in this manner. A break can also cause FE to be set. FE is cleared when SCSR is read with FE set, followed by a read of register RDR.

PF — Parity Error Flag

1 = Parity error occurred on the received data

0 = No parity error occurred on the received data

PF is set when the SCI receiver detects a parity error. PF is not set until the entire frame is received and RDRF is set. Although an interrupt is not explicitly associated with PF, an interrupt may be generated with RDRF and PF checked in this manner. PF is cleared when SCSR is read with PF set, followed by a read of the register RDR.

5.2.4 SCI Data Register (SCDR)

SCDR contains two data registers, both at the same address. The first register is the RDR, which is a read-only register. It contains data received over the SCI serial interface. Initially, data is received into the receive serial shifter and is transferred by the receiver into RDR. The second register is the SCI TDR, which is a write-only register. Data to be transmitted over the SCI serial interface is written to TDR. The transmitter transfers this data to the transmit serial shifter, adding on additional format bits before the data is sent out on the SCI serial interface.

SCDR — SCI Data Register **$YFFC0E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0
RESET:															
0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U

R8/T8 — Receive 8/Transmit 8

This bit is the ninth serial data bit received (R8) when the SCI system is configured for a 9-bit data operation (M = 1). When the SCI system is configured for an 8-bit data operation (M = 0), this bit has no meaning or effect.

This bit is the ninth serial data bit transmitted (T8) when the SCI system is configured for 9-bit data operation (M = 1). When the SCI system is configured for an 8-bit data operation (M = 0), this bit has no meaning or effect.

Accesses to the lower byte of SCDR triggers the mechanism for clearing the status bits or for initiating transmissions whether byte, word, or long-word accesses are used.

R[7:0]/T[7:0] — Receive 7–0/Transmit 7–0

The first eight bits (7:0) contain the first eight data bits to be received (R[7:0]) when SCDR is read, and also contain the first eight data bits to be transmitted (T[7:0]) when SCDR is written.

5.3 Transmitter Operation

The transmitter consists of a transmit serial shifter and a parallel transmit data register (TDR) located in SCDR (refer to **5.2.4 SCI Data Register (SCDR)**). A character may be loaded into the TDR while another character is being shifted out, a capability called double buffering. The transmit serial shifter cannot be directly accessed by the CPU. The output of the transmit serial shifter is connected to the TXD pin whenever the transmitter is operating (TE = 1, or TE = 0 and transmitter operation not yet complete).

The following definitions apply to the transmitter and receiver operation:

Bit-Time — The time required to serially transmit or receive one bit of data, which is equal to one cycle of the baud frequency.

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 and be preceded by at least three receive time (RT) samples of logic one.

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

Frame — A start bit, followed by a specified number of data or information bits, terminated by a stop bit. The number of data or information bits must agree between the transmitting and receiving devices. The most common frame format is one start bit followed by eight data bits (LSB first) terminated by one stop bit, for a total of 10 bit-times in the frame. The SCI optionally provides a 9-bit data format that results in an 11 bit-time frame.

The M bit in SCCR1 specifies the number of bit-times in the frame (ten or eleven). The most common format for nonreturn-to-zero (NRZ) serial interface is one start bit (logic zero or space), followed by eight data bits (terminated LSB first), by one stop bit (logic one or mark). In addition to this standard format, the SCI provides hardware support for a 9-bit data format. This format is one start bit, eight data bits (LSB first), a parity or address (control) bit, and one stop bit. Following are all the possible formats:

Start bit, seven data bits, two stop bits
Start bit, seven data bits, address bit, one stop bit
Start bit, seven data bits, address bit, two stop bits
Start bit, seven data bits, parity bit, one stop bit
Start bit, eight data bits, one stop bit
Start bit, eight data bits, two stop bits
Start bit, eight data bits, parity bit, one stop bit
Start bit, eight data bits, address bit, one stop bit

When the transmitter is enabled by writing a one to TE in SCCR1, a check is made to determine if the transmit serial shifter is empty. If empty (TC = 1), a preamble consisting of all ones (no start bits) is transmitted. If the transmit serial shifter is not empty (TC

= 0), then normal shifting continues until the word in progress with stop bit(s) is sent. The preamble (an all ones frame) is then transmitted.

When TE is cleared, the transmitter is disabled only after all pending information is transmitted, including any data in the transmit serial shifter (inclusive of the stop bit), any queued preamble (idle frame), or any queued break (logic zero frame). The TC flag is set, and the TXD pin reverts to control by PORTQS and DDRQS. This function allows the user to terminate a transmission sequence in the following manner. After loading the last byte into register TDR and receiving the interrupt from TDRE in SCSR, (indicating that the data has transferred into the transmit serial shifter), the user clears TE. The last frame is transmitted normally, and the TXD pin reverts to control by PORTQS and DDRQS.

To insert a delimiter between two messages and place the nonlistening receivers in wakeup mode or to signal a retransmission (by forcing an idle line), TE is set to zero and then to one before the word in the transmit serial shifter has completed transmission. The transmitter waits until that word is transmitted and then starts transmission of a preamble (ten or eleven contiguous ones). After the preamble is transmitted, and if TDRE is set (no new data to transmit), the line continues to mark (remain high). Otherwise, normal transmission of the next word begins.

Two SCI messages may be separated with minimum idle time by using a preamble of ten bit-times (eleven if a 9-bit data format is specified) of marks (logic ones). The entire process can occur using the following procedure:

A. Write the last byte of the first message to the TDR.
B. Wait for TDRE to go high, indicating that the last byte is transferred to the transmit serial shifter.
C. Clear TE and then set TE back to one. This queues the preamble to follow the stop bit of the current transmission immediately.
D. Write the first byte of the second message to register TDR.

In this sequence, if the first byte of the second message is not transferred to register TDR prior to the finish of the preamble transmission, then the transmit data line (TXD pin) simply marks idle (logic one) until TDR is finally written. Also, if the last byte of the first message finishes shifting out (including the stop bit) and TE is clear, TC will go high and transmission will be considered complete. The TXD pin reverts to being a general-purpose I/O line.

The CPU writes data to be transmitted to register TDR, which automatically loads the data into the transmit serial shifter. Before writing to TDR, the user should check TDRE in SCSR. If TDRE = 0, then data is still waiting to be sent to the transmit serial shifter. Writing to TDR with TDRE clear overwrites previous data to be transferred. If TDRE = 1, then register TDR is empty, and new data may be written to TDR clearing TDRE.

As soon as the data in the transmit serial shifter has shifted out and if a new byte of data is in TDR (TDRE = 0), then the new data is transferred from register TDR to the transmit serial shifter, and TDRE is automatically set. An interrupt may optionally be generated at this point.

The data in the transmit serial shifter is prefixed by a start bit (logic zero) and suffixed by the ninth data bit, if M = 1, and by one stop bit. The ninth data bit can be used as normal data or as an extra stop bit. A parity bit is substituted if PE = 1. This data stream is shifted out over the TXD pin. When the data is completely shifted out and no preamble or send break is requested, then TC is set to one and the TXD pin remains high (logic one or mark).

Parity generation is enabled by setting PE in SCCR1 to a one. The last data bit, bit eight (or bit nine of the data if M = 1), is used as the parity bit, which is inserted between the normal data bits and the stop bit(s).

When TE is cleared, the transmitter yields control of the TXD pin in the following manner. If no information is being shifted out (i.e., if the transmitter is in an idle state, TC = 1), then the TXD pin reverts to being a general-purpose I/O pin. If a transmission is still in progress (TC = 0), the characters in the transmit serial shifter continue to be shifted out normally, followed by any queued break. When finished, TXD reverts to being a general-purpose I/O pin. To avoid terminating the transmitter before all data is transferred, the software should always wait for TDRE to be set before clearing TE.

Transmissions may be purposely aborted by the send break function. By writing SBK in SCCR1 to a one, a nonzero integer multiple of ten bit-times (eleven if 9-bit data format is specified) of space (logic zero) is transmitted. If SBK is set while a transmission is in progress, the character in the transmit serial shifter finishes normally (including the stop bit) before the break function begins. Break frames are sent until either SBK or TE is cleared. To guarantee the minimum break time, SBK should be quickly toggled to one and then back to zero. After the break time, at least one bit-time of mark idle (logic one) is transmitted to ensure that a subsequent start bit can be recognized.

The TXD pin has several control options to provide flexible operation. WOMS in SCCR1 can select either open-drain output (for wired-OR operation) or normal CMOS output. WOMS controls the function of the TXD pin whether the pin is being used for SCI transmissions (TE = 1) or as a general-purpose I/O pin.

In an SCI system with multiple transmitters, the wired-OR mode should be selected for the TXD pin of all transmitters, allowing multiple output pins to be coupled together. In the wired-OR mode, an external pull-up resistor on the TXD pin is necessary.

In some systems, a mark (logic one) signal is desired on the TXD pin, even when the transmitter is disabled. This is accomplished by writing a one to PORTQS in the appropriate position and configuring the TXD pin as an output in DDRQS. When the transmitter releases control of the TXD pin, it reverts to driving a logic one output, which is the same as mark or idle.

5.4 Receiver Operation

The receiver can be divided into two segments. The first is the receiver bit processor logic that synchronizes to the asynchronous receive data and evaluates the logic sense of each bit in the serial stream. The second receiver segment controls the functional operation and the interface to the CPU including the conversion of the serial data stream to parallel access by the CPU.

5.4.1 Receiver Bit Processor

The receiver bit processor contains logic to synchronize the bit-time of the incoming data and to evaluate the logic sense of each bit. To accomplish this an RT clock, which is 16 times the baud rate, is used to sample each bit. Each bit-time can thus be divided into 16 time periods called RT1–RT16. The receiver looks for a possible start bit by watching for a high-to-low transition on the RXD pin and by assigning the RT time labels appropriately.

When the receiver is enabled by writing RE in SCCR1 to one, the receiver bit processor logic begins an asynchronous search for a start bit. The goal of this search is to gain synchronization with a frame. The bit-time synchronization is done at the beginning of each frame so that small differences in the baud rate of the receiver and transmitter are not cumulative. The SCI also synchronizes on all one-to-zero transitions in the serial data stream, which makes the SCI tolerant to small frequency variations in the received data stream.

The sequence of events used by the receiver to find a start bit is listed below.

A. Sample RXD input during each RT period and maintain these samples in a serial pipeline that is three RT periods deep.
B. If RXD is low during this RT period, go to step A.
C. If RXD is high during this RT period, store sample and proceed to step D.
D. If RXD is low during this RT period, but not high for the previous three RT periods (which is noise only), set an internal working noise flag and go to step A, since this transition was not a valid start bit transition.
E. If RXD is low during this RT period and has been high for the previous three RT periods, call this period RT1, set RAF, and proceed to step F.
F. Skip RT2 but place RT3 in the pipeline and proceed to step G.
G. Skip RT4 and sample RT5. If both RT3 and RT5 are high (RT1 was noise only), set an internal working noise flag. Go to step c and clear RAF. Otherwise, place RT5 in the pipeline and proceed to step H.
H. Skip RT6 and sample RT7. If any two of RT3, RT5, or RT7 is high (RT1 was noise only), set an internal working noise flag. Go to step c and clear RAF. Otherwise, place RT7 in the pipeline and proceed to step I.
I. A valid start bit is found and synchronization is achieved. From this point on until the end of the frame, the RT clock will increment starting over again with RT1 on each one-to-zero transition or each RT16. The beginning of a bit-time is thus defined as RT1 and the end of a bit-time as RT16.

Upon detection of a valid start bit, synchronization is established and is maintained through the reception of the last stop bit, after which the procedure starts all over again to search for a new valid start bit. During a frame's reception, the SCI resynchronizes the RT clock on any one-to-zero transitions.

Additional logic in the receiver bit processor determines the logic level of the received bit and implements an advanced noise-detection function. During each bit-time of a frame (including the start and stop bits), three logic-sense samples are taken at RT8, RT9, and RT10. The logic sense of the bit-time is decided by a majority vote of these three samples. This logic level is shifted into register RDR for every bit except the start and stop bits.

If RT8, RT9, and RT10 do not all agree, an internal working noise flag is set. Additionally for the start bit, if RT3, RT5, and RT7 do not all agree, the internal working noise flag is set. If this flag is set for any of the bit-times in a frame, the NF flag in SCSR is set concurrently with the RDRF flag in SCSR when the data is transferred to register RDR. The user must determine if the data received with NF set is valid. Noise on the RXD pin does not necessarily corrupt all data.

The operation of the receiver bit processor is shown in the following figures. These examples demonstrate the search for a valid start bit and the synchronization procedure as outlined above. The possibility of noise durations greater than one bit-time are not considered in these examples. **Figure 5-3** illustrates the ideal case with no noise present.

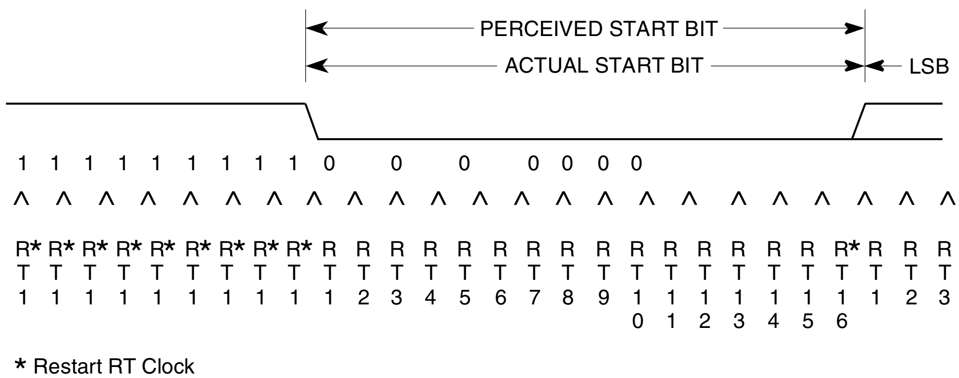


Figure 5-3 Start Search Example 1

Figure 5-4 shows the start bit search and resynchronization process being restarted because the first low detected was determined to be noise rather than the beginning of a start bit-time. Since the noise occurred before the start bit was found, it will not cause the internal working noise flag to be set.

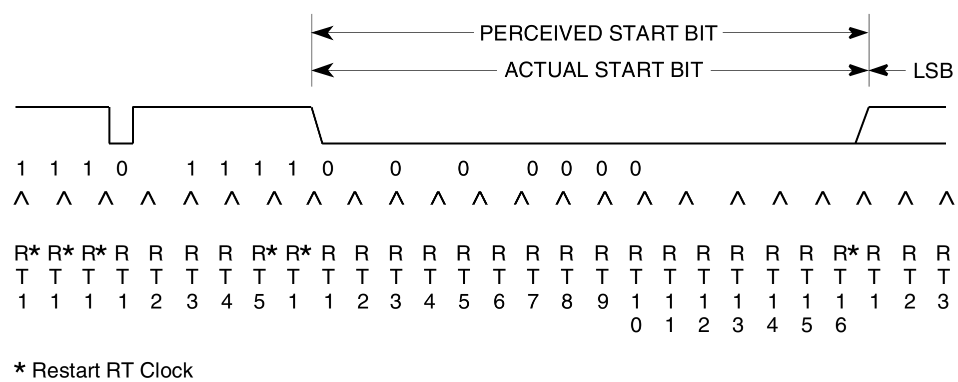


Figure 5-4 Start Search Example 2

Figure 5-5 shows that noise is perceived as the beginning of a start bit. Note that the high level sensed at RT3 causes the internal working noise flag to be set. Even though this figure shows improper alignment of the perceived bit-time boundaries to the actual bit-time boundaries, the logic sense samples taken at RT8, RT9, and RT10 fall well within the correct actual bit-time. The start bit and all other bits in the frame should be received correctly.

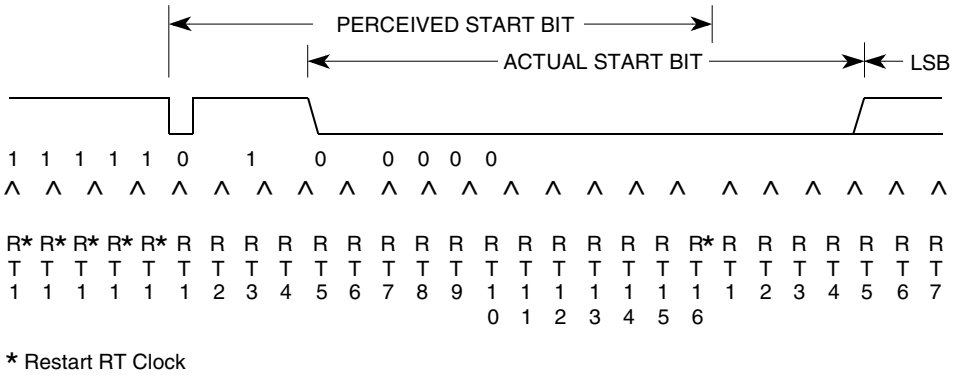


Figure 5-5 Start Search Example 3

Figure 5-6 shows how a large burst of noise is perceived as the beginning of a start bit. Note that RT5 is sensed logic high, setting the internal working noise flag. This figure also illustrates a worst-case alignment of the perceived bit-time boundaries to the actual bit-time boundaries; however, RT8, RT9, and RT10 all fall within the correct actual bit-time. The start bit is detected and the incoming data stream is correctly sensed.

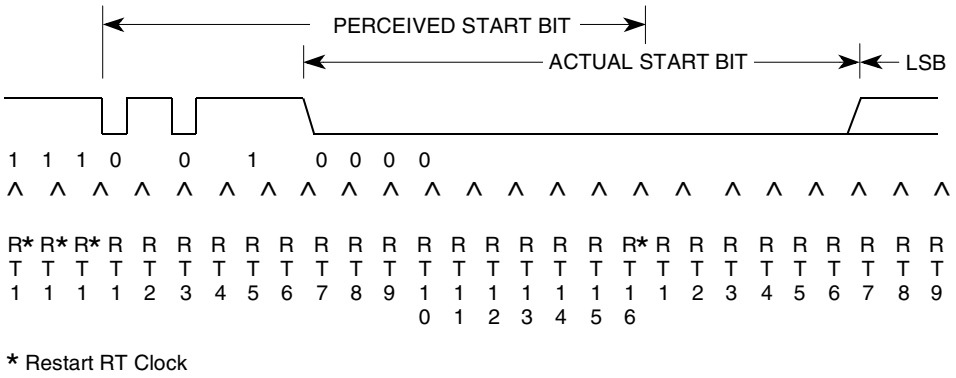
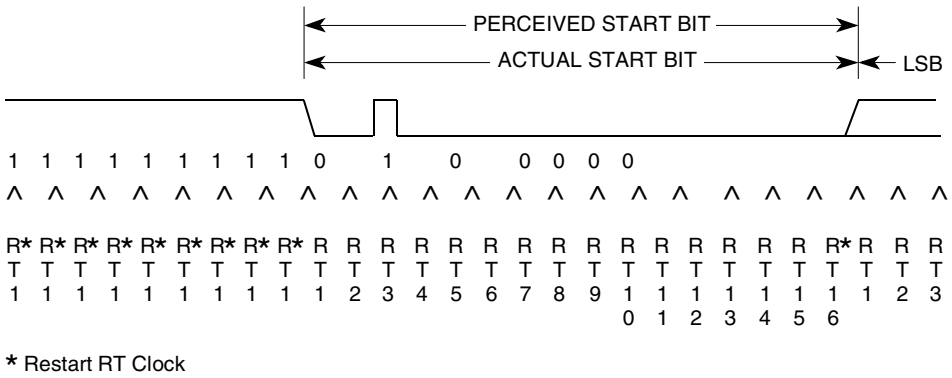


Figure 5-6 Start Search Example 4

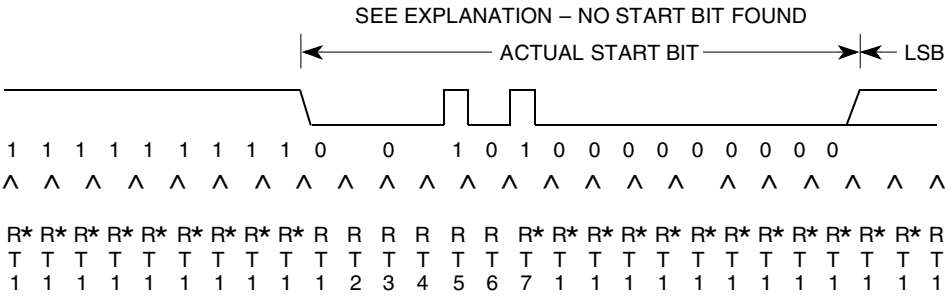
Figure 5-7 illustrates the effect of noise early within the start bit-time. Although this noise does not affect proper synchronization with the start bit-time, it does set the internal working noise flag.



* Restart RT Clock

Figure 5-7 Start Search Example 5

Figure 5-8 shows a large burst of noise near the beginning of the start bit that causes the start bit search to be restarted. During RT1 following RT7, a search for a new start bit could not be started as the previous three RT samples are not all high. The receiver bit processor misses this start bit. The frame might be partially received or missed entirely, depending on the data in the frame and when the start bit search logic synchronized upon what appeared to be a start bit. If a valid stop bit is not detected, an FE flag is set in SCSR.



* Restart RT Clock

Figure 5-8 Start Search Example 6

Figure 5-9 explores the case where the majority vote of RT8, RT9, and RT10 returns a logic-high level. However, the start bit is a special case that overrules the majority voting scheme. In review, at least three of the samples taken at RT1, RT3, RT5, and RT7 must be low. The start bit is detected and the RT clock is synchronized; because RT8–RT10 were not unanimous, the NF flag is set.

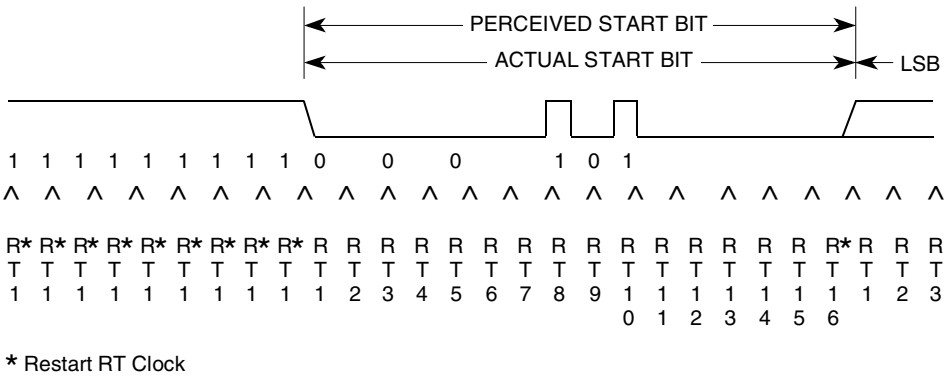


Figure 5-9 Start Search Example 7

5.4.2 Receiver Functional Operation

The receiver contains a receive serial shifter and a parallel RDR. While one character is in the process of being shifted in, another character may be held in RDR. This capability is called double buffering. The receive serial shifter cannot be accessed directly by the CPU. The input of the receive serial shifter is connected to the majority sampling logic of the receive bit processor.

The receiver is enabled when RE in SCCR1 is set to one. When RE is zero, the receiver is initialized and most of the receiver bit processor logic is disabled. The receiver bit processor logic drives a state machine (run by the RT clock) that determines the logic level for each bit-time. This state machine controls when the bit processor logic is to sample the RXD pin and also controls when data is to be passed to the receive serial shifter. Data is shifted into the receive serial shifter according to the most recent synchronization of the RT clock with the incoming data stream. From this point on, the data is moved synchronously with the MCU system clock.

The first bit shifted in is the start bit, which is always a logic zero. The next eight bits shifted in are the basic data byte (LSB first). The next bit shifted in depends on the mode selected by M in SCCR1. If M = 1, then the bit is the ninth data bit and is placed in R8 of SCDR, concurrent with the transfer of data from the receive serial shifter to register RDR.

The last bit shifted in for each frame is the stop bit, which is always a logic one. If a logic zero is sensed during this bit-time, the FE error flag in SCSR is set. A framing error is usually caused by mismatched baud rates between the receiver and transmitter or by a significant burst of noise. Note that a framing error is not always caught; the data in the expected stop bit-time may be a logic one regardless.

When the stop bit is received, the frame is considered to be complete, and the received character in the receive serial shifter is transferred in parallel to RDR. If M = 1, the ninth bit is transferred at the same time; however, if the RDRF flag in SCSR is set, transfers are inhibited. Instead, the OR error flag is set, indicating to the user that the CPU needs to service register RDR faster. The data in RDR is preserved, but the data in the receive serial shifter is lost.

All status flags associated with a serially received frame are set simultaneously and at a time that does not interfere with CPU access to the affected registers. When a completed frame is received, either the RDRF or OR flag is always set. If RIE in SCCR1 is set, an interrupt results whenever RDRF is set. The receiver status flags NF, FE, and PF are set simultaneously with RDRF, as appropriate. These receiver flags are never set with OR because the flags only apply to the data in the receive serial shifter. The receiver status flags do not have separate interrupt enables, since they are set simultaneously with RDRF and must be read by the user at the same time as RDRF.

All receiver status flags are cleared by the following sequence. Register SCSR is read first, followed by a read of register SCDR. Reading SCSR not only informs the CPU of the status of the received data, but also arms the clearing mechanism. Reading SCDR supplies the received data to the CPU and clears all of the status flags: RDRF, IDLE, OR, NF, FE, and PF.

5.4.2.1 Idle-Line Detect

The receiver hardware includes the ability to detect an idle line. This function can be used to indicate when a group of serial transmissions is finished. An idle line is defined as a minimum of ten bit-times (or eleven if a 9-bit data format is selected) of contiguous ones on the RXD pin. During a typical serial transmission, frames are transmitted isochronously, that is, no idle time occurs between frames. Even if all data bits in a frame are logic ones, the start bit ensures that at least one logic zero bit-time occurs for each frame.

Motorola MCUs from the M68HC11 and M68HC05 Families have SCIs with only one type of idle-line detect circuitry. On these MCUs, the receiver bit processor starts counting logic one bit-times at any point (even within a frame). This method allows the earliest recognition of an idle line because the stop bit and any contiguous ones preceding the stop bit are counted with the logic ones in the idle line following the stop bit. In some applications, the CPU overhead prevents the servicing of interrupts as soon as possible to ensure that no bit-time of an idle line occurs between frames. Although this idle line causes no deterioration of the message content, if one bit-time should occur after a data byte of all ones, the combination is seen as an idle line and causes sleeping SCIs to wake up.

The SCI on the QSM module contains this same idle-line detect logic called short idle-line detect as well as long idle-line detect. In long idle-line detect mode, the SCI begins counting logic ones after the stop bit is received. The data content of a byte, therefore, does not affect how quickly the idle line is detected. When RXD goes idle for the minimum required time, the IDLE flag in SCSR is set. ILT in SCCR1 is used to choose between short and long idle-line detection.

If ILIE in SCCR1 is set, a hardware interrupt request is generated when the IDLE flag is set. This flag is cleared by reading SCSR with IDLE set, followed by reading register RDR. The IDLE flag is not set again until after at least one frame has been received (RDRF = 1), which prevents an extended idle interval from causing more than one interrupt.

5.4.2.2 Receiver Wakeup

The SCI receiver hardware provides a receiver wakeup function to support multinode networks containing more than one receiver. This function allows the transmitting device to direct a message to an individual receiver or group of receivers by sending an address frame at the start of a message. All receivers not addressed for the current message invoke the receiver wakeup function, which effectively allows them to sleep through the rest of the message. Therefore, the CPU is alleviated from servicing register RDR, resulting in increased system performance.

The SCI receiver is placed in wakeup mode by writing a one to RWU in SCCR1. While RWU is set, all receiver status flag bits are inhibited from being set. Note that the IDLE flag cannot be used when RWU is set. Although the CPU can clear RWU by writing a zero to SCCR1, it is normally left alone by software and is cleared automatically by hardware in one of two methods: idle-line wakeup or address-mark wakeup.

WAKE in SCCR1 determines which method of wakeup is to be employed. If WAKE = 0, idle-line wakeup is selected. This method is compatible with the method originally used on the MC6801. If WAKE = 1, address-mark wakeup is selected, which uses a one in the MSB of data to denote an address frame and uses a zero to denote a normal data frame. Each method has its particular advantages and disadvantages.

Both wakeup methods require a software device addressing and recognition scheme and, therefore, can conform to all transmitters and receivers. The addressing information is usually the first frame(s) of the message. Receivers for which the message is not intended may set RWU and go back to sleep for the remainder of the message.

Idle-line wakeup allows a receiver to sleep until an idle line is detected, causing RWU to be cleared by the receiver and causing the receiver to wake up. The receiver waits through the idle times for the first frame of the next message. If the receiver is not the intended addressee, RWU may be set to put the receiver back to sleep. This method of receiver wakeup requires that a minimum of one frame of idle line be imposed between messages. As previously stated, no idle time is allowed between frames within a message.

Address-mark wakeup uses a special frame format to wake up the receiver. All frames consist of seven (or eight) data bits plus an MSB that indicates an address frame when set to a one. The first frame of each message should be an address frame. All receivers in the system must use a software scheme to determine which messages address them. If the message is not intended for a particular receiver, the CPU sets RWU so that the receiver goes back to sleep, thereby eliminating additional CPU overhead for servicing the rest of the message.

When the first frame of a new message is received with the MSB set, denoting an address frame, RWU is cleared. The byte is received normally, transferred to register RDR, and the RDRF flag is set. Address-mark wakeup allows messages to include idle times between frames and eliminates idle time between messages; however, an efficiency loss results from the extra bit-time (address bit) that is required on all frames.

APPENDIX A
USING THE QSPI FOR ANALOG DATA AQUISITION

A.1 Introduction

To effectively use digital microcontroller units (MCUs) in an analog world, analog information must be converted into digital form. In all applications, fast, accurate, and inexpensive conversion is desirable. Minimizing printed circuit board space and interconnections is also desirable.

NOTE

This application note can be applied to any MCU (i.e., MC68332, MC68HC16Z1, etc.) containing queued serial peripheral interface (QSPI) circuitry.

The MC68332 lacks any direct analog-to-digital (A/D) conversion capabilities. This deficiency is easily and inexpensively remedied by connecting the QSPI to an external serial A/D converter.

This application note presents hardware and software examples detailing use of the QSPI with multichannel 8- and 10-bit A/D converters, specifically the MC145040 and the MC145050. It describes design methodology for obtaining maximum A/D throughput, using one or more A/D converters. It also discusses how to simultaneously use other peripherals with the QSPI and how to determine overall system performance.

A.2 Operation of the MC145040 and MC145050 Family A/D Converters

The following paragraphs give a brief overview of the Motorola serial A/D converters. For a more thorough treatment of the subject, refer to **Reference 3.** and **Reference 4.**

The MC145040, MC145041, MC145050, and MC145051 are low-cost, ratiometric, 11-channel A/D converters. They are designed for connection to a microcomputer system with channel selection and conversion results being conveyed through a serial interface port. They require only 14 mW from a single 5-V power supply and yield ±1 LSB accuracy over the -40 to +125°C range. The reference voltage can be anywhere from +2.5 V to V_{DD}, and the analog input voltage may range from V_{SS} to V_{DD}.

The MC145050 and MC145051 are 10-bit converters; whereas, the MC145040 and MC145041 are 8-bit converters. The MC145040 and MC145050 use external clock sources to perform the conversion; the MC145041 and MC145051 use internal RC oscillators. The parts using external oscillators guarantee faster conversion rates because internal oscillator frequency must be limited to guarantee reasonable yield despite manufacturing tolerances. The remaining A/D converter description refers specifically to the MC145050 since it is the converter used in the examples presented.

Figure A-1 shows the pinout of the MC145050. It has 13 analog pins, consisting of 11

analog inputs, labeled AN0–AN11, and two voltage reference inputs, labeled V_{AG} (analog ground) and V_{REF} (positive reference voltage). Power is supplied through the V_{SS} and V_{DD} pins and is a nominal 5-V. The MC145050 requires an external clock to be supplied on the A/D CLK pin to regulate the data conversion.

Channel selection and conversion results are transferred through the digital serial communication pins. A serial transfer synchronizing clock must be fed into the SCLK input pin when the chip-select ($\overline{CS}$) pin is driven low. The address to be converted is serially transmitted into the DIN pin, and the conversion results are serially shifted out the DOUT pin.

The MC145050 is designed to be used in conjunction with multiple serial devices on a common bus; consequently, the DOUT pin is driven only when $\overline{CS}$ is asserted. The serial protocol employed is Motorola SPI, which is compatible with the National Semiconductor Microwire[a] system and the Texas Instrument TMS370 series SPI units. The Motorola queued serial module (QSM) also contains a QSPI that efficiently implements this protocol.

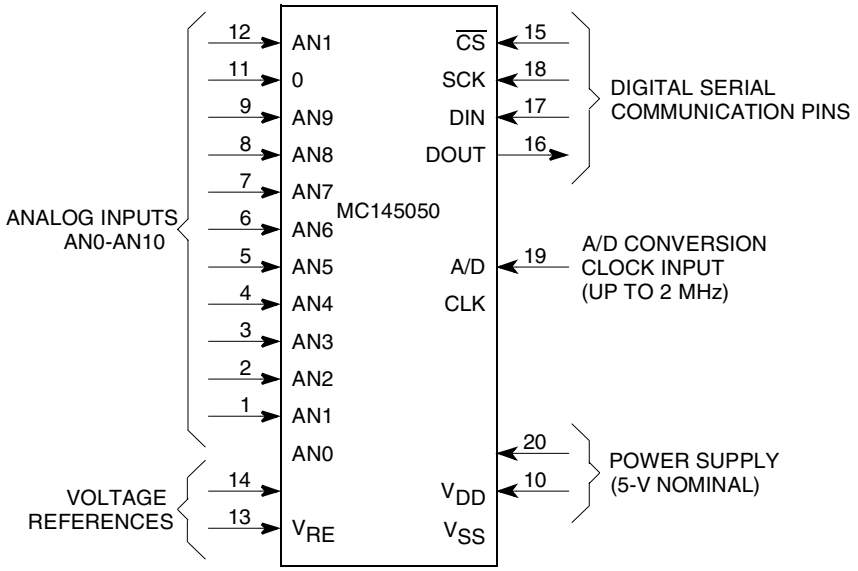


Figure A-1 MC145050 Pinout

A.3 Fundamentals of QSPI Operation

The following paragraphs give a brief overview of the QSPI as it applies to the examples that are presented. A more detailed description of the QSPI is contained in Section 5 of *MC68332 User's Manual* (see **Reference 2.**).

The QSPI is an intelligent, synchronous serial interface with a 16-entry, full-duplex queue. It can continuously scan up to 16 independent peripherals and maintain a queue of the most recently acquired information with no central processor unit (CPU) intervention. It features variable word lengths, programmable chip selects, and select-

able data/clock phase relationship. The baud rate and the delay between transfers are also programmable. The QSPI has a maximum transfer speed of one-fourth the MC68332 system clock speed.

Since the QSPI is capable of operation as a master or as a slave, all pins are bidirectional. **Figure A-2** shows a typical master mode configuration. The slave peripherals are selected via the peripheral chip-select pins, PCS[0:3], and the serial clock is provided by the SCK pin. QSPI output data is presented on the master out slave in (MOSI) pin, and input is taken from the master in slave out (MISO) pin.

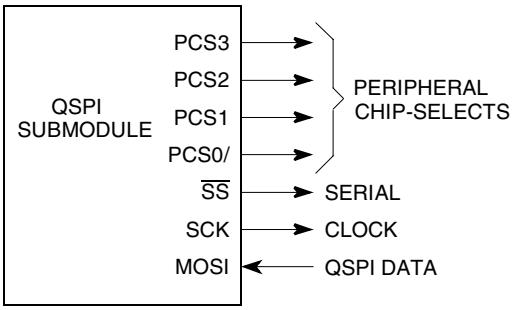


Figure A-2 Master Mode Representation of the QSPI

One of the most powerful elements of the QSPI is its queue. **Figure A-3** depicts the structure of the QSPI queue RAM. The queue may contain up to 16 entries, each consisting of a transmit word, a receive word, and a command control byte. The transmit and receive words are from 8 to 16 bits long and are LSB justified. For any given queue entry, the transmit and receive words are the same length.

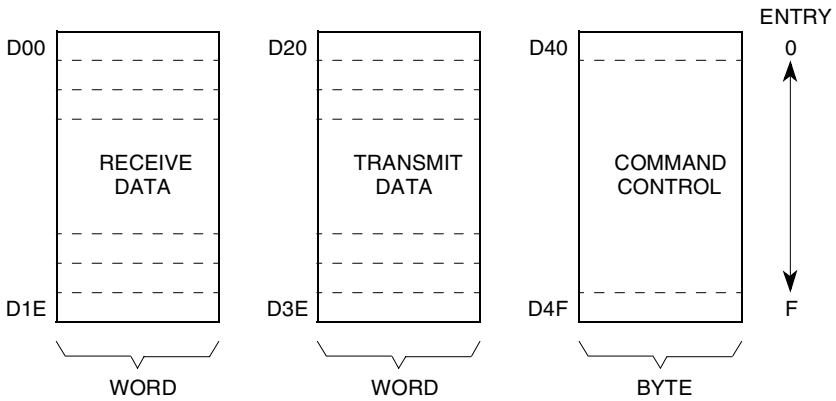


Figure A-3 Organization of the QSPI Ram

An important subset of the queue RAM is the command control RAM. **Figure A-4**

shows a breakdown of a single command control byte, and **Figure A-5** depicts a basic QSPI master mode timing diagram. The control byte allows the programmer to customize each serial transfer to the specific needs of the targeted peripheral. Chip-select patterns are stored in the PCS[0:3] bit fields of each applicable control byte and are driven onto the chip-select pins when the specified transfer begins. If set, the continue (CONT) bit allows the QSPI to continue driving the programmed chip-select value until the beginning of the next transfer. This procedure has the effect of concatenating multiple serial transfers to a single peripheral and allowing more than 16 bits per exchange. If the CONT bit is clear, a user-defined default value is driven onto the chip-select pins between serial transfers.

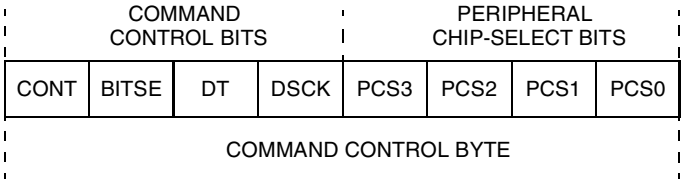


Figure A-4 Command Control Byte

The PCS to SCK delay (DSCK) and delay after transfer (DT) bits enable user-defined delays before and after the specified transfer. If DSCK is set, the first clock following the chip-select assertion is delayed by a user-specified amount of time. Otherwise, the first clock pulse is delayed one-half of an SCK period. This delay is necessary because some peripherals require a relatively long period of time to respond.

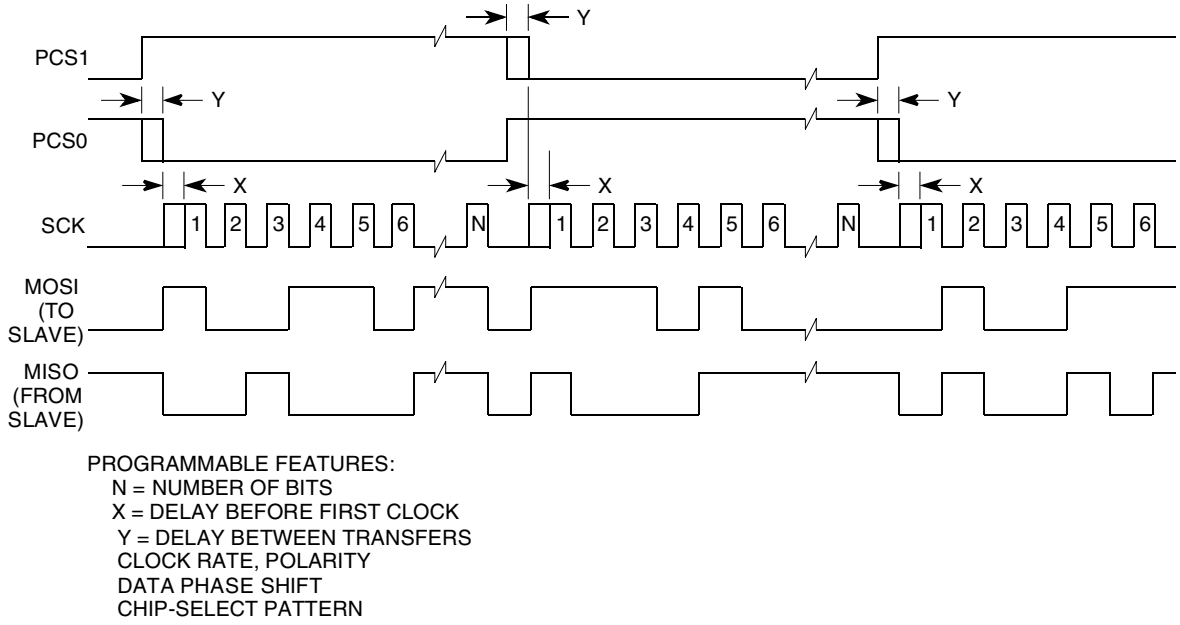


Figure A-5 Basic QSPI Master Mode Timing Diagram

If DT is set, a user-specified delay elapses before the next serial transfer is begun. Otherwise, the QSPI executes the next transfer as soon as possible (approximately 1 ms when the MC68332 operates at 16.778 MHz). This delay is useful if a peripheral needs time to perform a function that affects subsequent serial transfers. One example might be to wait for an A/D converter to perform a conversion.

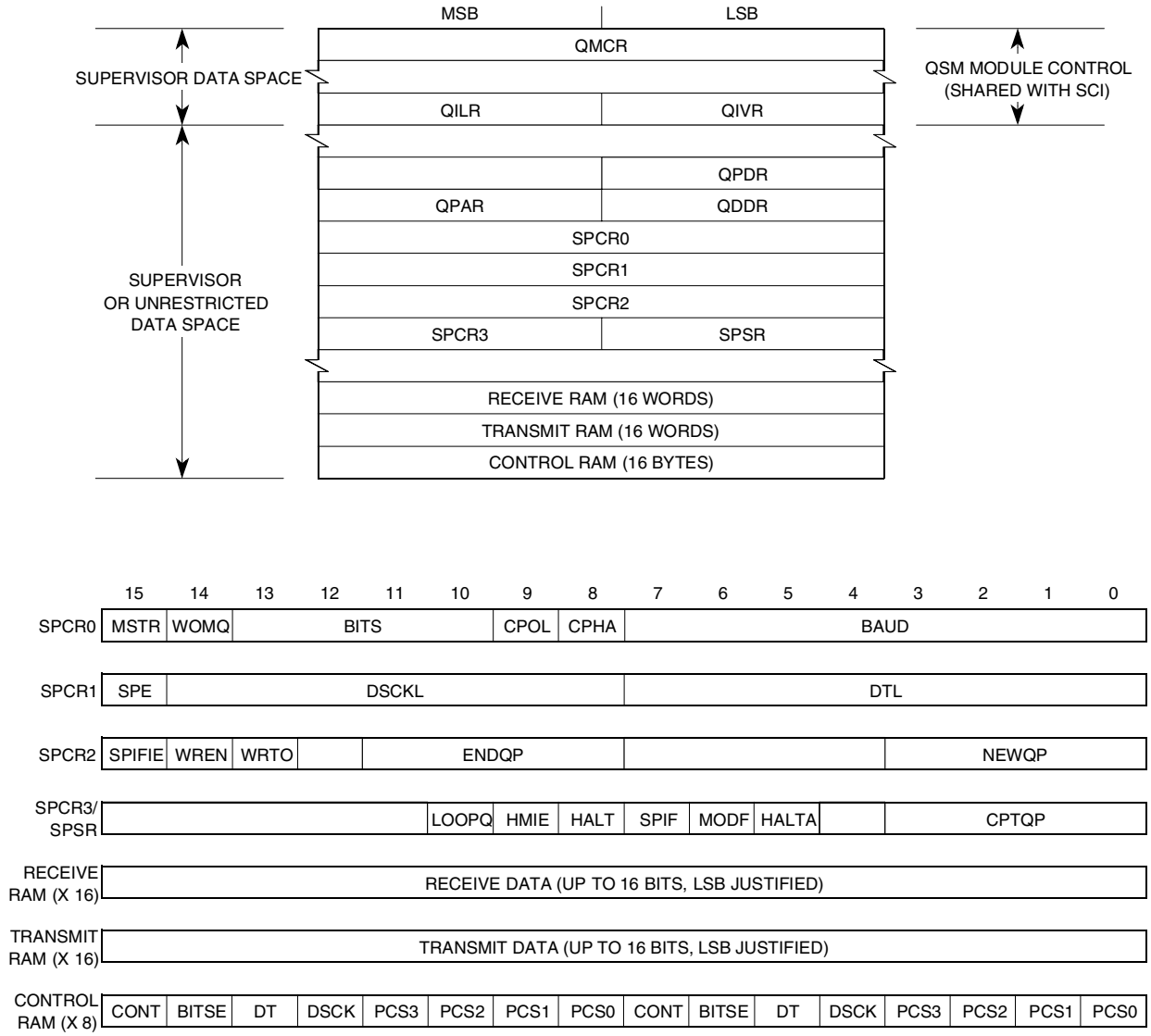
The remaining element in the control byte is the bits per transfer enable (BITSE) bit. If BITSE is set, the transfer length is a user-specified value, ranging from eight to 16 bits. If BITSE is cleared, the transfer length will default to eight bits.

Figure A-6 represents a programmer's model of the QSPI. The QSM data direction register (QDDR) determines whether a given QSPI pin is an input or an output. When read, the QSM port data register (QPDR) provides the logic level present on a QSM input pin or the data latched in an output pin. When written, the write data is latched into the output register. The QSM pin assignment register (QPAR) controls whether a pin is to be controlled by the QSPI or is to function as a general-purpose I/O pin.

Serial peripheral control register 0 (SPCR0) specifies six different functions. The master/slave mode select (MSTR) bit, if set, causes the QSPI to operate as the controller of the SPI transfer. The wired-OR mode for QSPI pins (WOMQ) bit, if set, causes all QSPI outputs to function in an open-drain mode, requiring external pull-up resistors. The bits per transfer (BITS) field allows the programmer to specify the number of bits in a non-default transfer (used if BITSE is set). The clock polarity (CPOL) bit determines the polarity of the SCK output, and the clock phase (CPHA) bit dictates the data's phase relationship to the SCK. The serial clock baud rate (BAUD) field determines the QSPI SCK frequency, from 33 kHz to 4.2 MHz (with the MC68332 system clock frequency at 16.778 MHz).

Serial peripheral control register 1 (SPCR1) specifies three different functions. Setting the QSPI enable (SPE) bit causes the QSPI to begin operation; clearing SPE causes operation to stop immediately. SPE is automatically cleared by the QSPI when it completes all specified transfers. The DSCKL field allows the programmer to set the non-default delay before SCK (used if DSCK is set). The DTL field controls the non-default delay after the transfer is completed (used if DT is set).

Serial peripheral control register 2 (SPCR2) specifies five queue control functions. The new queue pointer value (NEWQP) field determines which queue entry is to be transferred first. More queue entries are sequentially transferred until the entry specified by the ending queue pointer (ENDQP) field is completed. If the wrap enable (WREN) bit is set, transfers continue either at queue entry 0 or at the entry specified by the NEWQP field. The point the queue wraps to (entry 0 or NEWQP) is determined by the wrap to (WRTO) bit. The SPI finished interrupt enable (SPIFIE) bit is an interrupt enable. If set, an interrupt will be generated upon completion of the queue entry specified by the ENDQP field.



NOTE: Shading denotes not used area.

Figure A-6 QSPI Programmer's Model

Serial peripheral control register 3 (SPCR3) controls self-test and program debug functions, which will not be discussed in this application note. The serial peripheral status register (SPSR) contains two status fields of importance for this application. The completed queue pointer (CPTQP) field contains the queue entry number that was most recently completed. The QSPI finished flag (SPIF) bit is set when the CPTQP matches the ENDQP, which indicates that the specified queue has been completed and the QSPI has either shut down or wrapped to the designated point.

A.4 Basic System Implementation

The schematic diagram shown in **Figure A-7** depicts the basic minimal serial A/D data acquisition system. The only extraneous logic required for this system is the 2 MHz oscillator. The oscillator can be used to supply a number of other peripheral devices as well as additional A/D converters. Also, the oscillator can be eliminated entirely, and an MC145051 can be used in place of the MC145050; however, the speed of the conversions would be reduced.

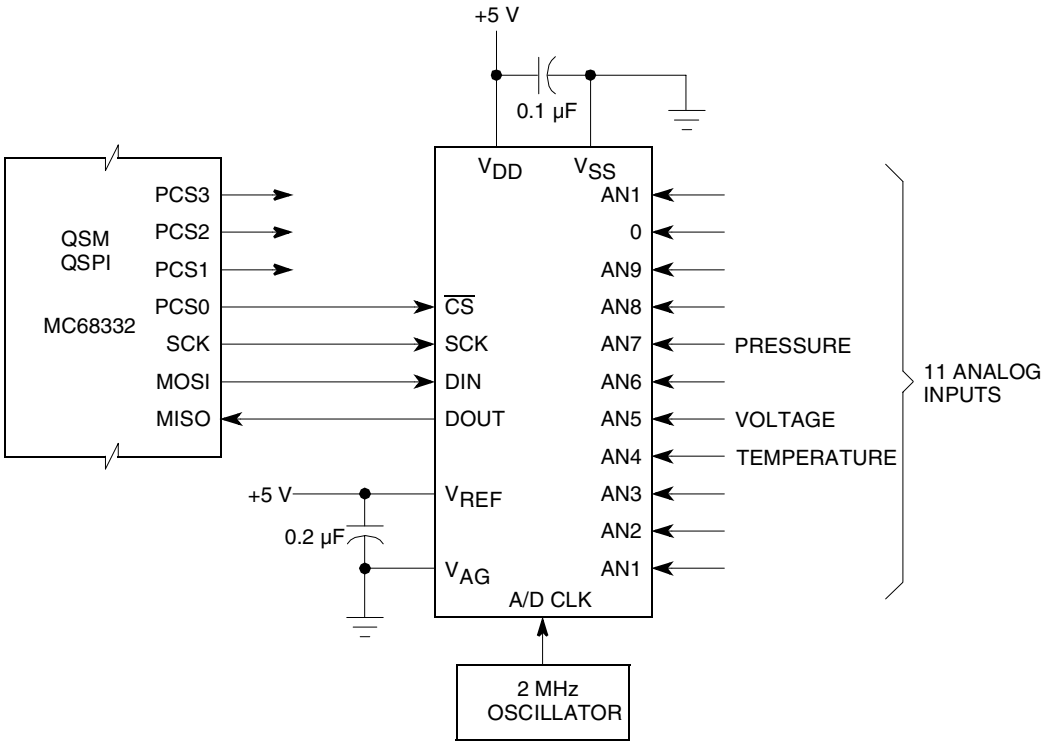


Figure A-7 Basic Serial A/D Data Acquisition System

The timing diagram (see **Figure A-8**) shows significant events on the pins of the MC145050. This timing sequence corresponds to the timing sequence illustrated in Figure 9 of **Reference 4.** Although not the fastest method for sampling the A/D converter, this timing sequence allows efficient use of the MC145050 on a bus in conjunction with other peripherals. During A/D conversion, the QSPI can select and exchange data with another device, maximizing overall serial bandwidth. The timing for 10-clock transfer not using $\overline{CS}$ may be slightly faster, but if it is used with other peripherals, the QSPI must wait for the conversion to be completed.

For successful operation, power supply decoupling and wiring should be carefully considered. The 0.1 mF decoupling capacitor should be placed as close as possible to the V_{DD} and V_{SS} pins. A nearby decoupling capacitor is also needed between the V_{REF} and V_{AG} pins. Separate lines should be run to the V_{REF} and V_{AG} inputs since any cur-

rent drain will cause IR voltage drop in the traces. If an active IC is being powered by the same trace, the switching current transients can cause enormous errors.

As the timing diagram shows, the MC145050 requires valid data on the DIN pin during the rising edge of SCK. The data is allowed to change on the falling edge of SCK. This determines the clock polarity and phase values that need to be programmed into the QSPI (CPOL = 0, CPHA = 0).

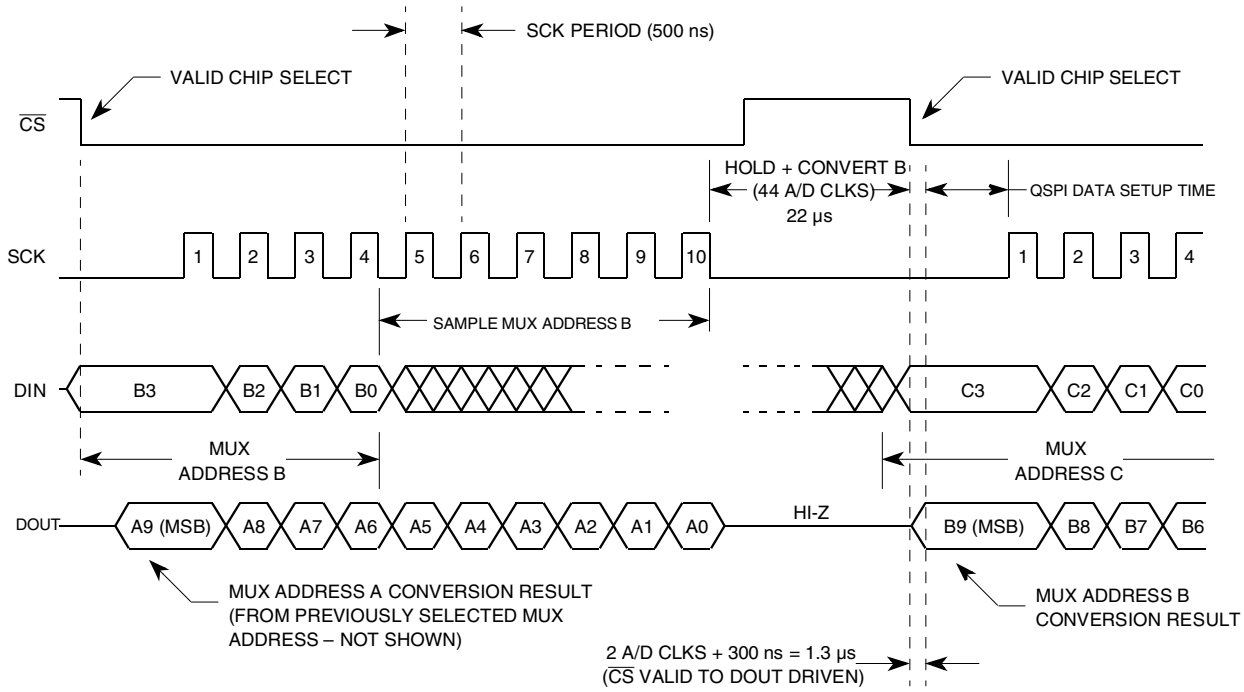


Figure A-8 MC14050 Conversion and Transfer Timing

A.5 Timing Considerations

One factor determining overall system speed is the source impedance of the signal being measured. The impedance limits the maximum SCK clock frequency because the SCK frequency is what determines the actual sample interval. For more information on source impedance effect on clock frequency, refer to **Reference 4.** A source impedance of less than 1000 ohms is assumed so that sample interval is not a constraint.

Calculate the maximum SCK frequency according to the following procedures. According to **Reference 4.**, the minimum SCLK pulse high and low widths (t_{wh}, t_{wl}) are both 190 ns, the maximum propagation delay from SCK to DOUT (t_{PHL}, t_{PLH}) is 240 ns, and the minimum setup time from DIN to SCK (t_{su}.A/D) is 100 ns.

Assuming a QSPI minimum data setup time (t_{su}.Q, MISO to SCK) of 10 ns, to meet QSPI input data timing requirements, the minimum clock pulse width is the greater of (t_{PLH} + t_{su}.Q) or (t_{PHL} + t_{su}.Q). This yields 250 ns.

Assuming a QSPI maximum data delay time (t_{dd}.Q, SCK to MOSI) of 10 ns, to meet MC145050 input data timing requirements, the minimum clock pulse width is the greater of t_{wh}, t_{wl}, or (t_{dd}.Q + t_{su}.A/D). This figure is 190 ns.

Data hold times on both the QSPI and the MC145050 are too minimal to present a problem, since data is not allowed to change until one-half SCK period after the latch is triggered. The minimum SCK period must be twice the largest minimum clock pulse width since the QSPI generates a symmetrical SCK waveform. This number is 500 ns, indicating a maximum SCK frequency of 2 MHz. The MC68332 will be clocked at a system clock frequency of 16 MHz, allowing an SCK frequency of exactly 2 MHz. The BAUD field value can be found from the following equation:

$$\text{BAUD} = \text{system clock frequency} / (2 * \text{desired SCK frequency})$$

Therefore, the BAUD field should be programmed to

$$\text{BAUD} = [16\ \text{MHz} / (2 * 2\ \text{MHz})] = 4$$

Another parameter that must be determined is the minimum time that must elapse between asserting the MC145050 CS pin and providing the first SCK pulse. According to **Reference 4.**, the maximum propagation delay from $\overline{CS}$ to DOUT driven (t_{PZL}, $t_{PZH)}$ is 2 A/D CLKs + 300 ns. Assuming a QSPI input data setup time of 10 ns and an A/D CLK frequency of 2 MHz, the total delay must be at least 10 + 300 + (2 * 500) = 1.31 ms. A minimum setup time from $\overline{CS}$ to SCK ($t_{su)}$ is 2 A/D CLKs + 425 ns. Since this value is 1.425 ms and is the larger value, the DSCKL field in QSPI SPCR1 must be programmed to provide at least this amount of delay. The MC68332 User's Manual (see **Reference 2.**) states the formula for DSCKL as follows:

$$\text{delay time} = \text{DSCKL} / \text{system clock frequency}$$

Solving for DSCKL gives

$$\text{DSCKL} = (1425\ \text{ns} / 62.5\ \text{ns}) = 22.8$$

Rounding up to the nearest whole delay, there are 23 DSCKL units for a total delay of 1.4375 ms. Also, the DSCK bit must be set in each command control byte that governs a transfer to the MC145050; otherwise, the standard delay of one-half SCK period will be used (in this case, 250 ns).

For a successful conversion to occur, a delay of 44 A/D CLKs must elapse from the last falling edge of SCK to the next assertion of $\overline{CS}$. The QSPI always provides a one-half SCK delay after the last SCK edge before the $\overline{CS}$ pins change state. The delay time before the next $\overline{CS}$ assertion must then be

$$(44 * 500\ \text{ns}) - 250\ \text{ns} = 21.75\ \text{ms}$$

The equation for delay between transfers is

$$\text{delay time} = (32 * \text{DTL}) / \text{system clock frequency}$$

thus, it follows that

$$DTL = (\text{system clock frequency} * \text{delay time}) / 32$$

therefore,

$$DTL = ((16 * 10^6) \text{ Hertz} * (21.75 * 10^{-6}) \text{ seconds}) / 32$$

$$DTL = 10.88 \text{ which rounds up to } 11$$

Plugging DTL = 11 into the original equation gives an actual delay of 22 ms.

A.6 QSPI Initialization and Operation

Since the fastest throughput is possible when using 10-bit transfers, the BITS field in SPCR0 must be set to ten. Additionally, the BITSE bit must be set in each command control byte associated with a transfer to the MC145050.

To simplify the example, assume conversions are only wanted from A/D channels 3, 4, and 6. Those channels will be sampled repeatedly, and each channel will have a separate fixed memory address where the most recently acquired result will always be available to the CPU. The WREN bit in SPCR2 and the first three queue entries will be used. The transmit RAM must contain the A/D multiplexer address to be converted, and the receive RAM will hold the conversion results.

Figure A-9 is an assembly language listing showing how the QSPI is configured to perform the stated functions.The first portion of the program is definitions, followed by initialization. The QSPI is then activated. The program waits until all conversions have been performed once before utilizing the results.

Figure A-10 shows the setup and operation of the queue RAM in this example. It is important to note that the conversion data requested by one queue entry is not shifted out until the next transfer; thus, the data is stored in the receive RAM corresponding to the latter transfer. Also, the very first transfer of output data from the A/D converter is invalid and should be ignored. This issue can be handled by simply waiting a known amount of time (until the first result has been updated).

Using a different approach, start the queue from entry F and then transfer and loop on entries 0, 1, and 2. Queue entry F executes once; whereas, entries 0-2 will repeat indefinitely, causing the invalid data word from the A/D converter to be stored in unused RAM (associated with queue entry F). After SPIF in the SPSR is set, all A/D result locations will contain valid data. From then on, the CPU merely reads the latest A/D results from their fixed locations, effectively making the serial A/D converter appear to the CPU as a parallel, memory-mapped peripheral. Having fixed locations for each channel's result allows the programmer to equate them with sensor names, making software easier to write and maintain (especially when compared to serial systems funneling all results through a single receive register).

The example in **Figure A-9** shows an interrupt service routine which will generate a warning if fuel pressure drops below a specific level. To cancel the warning, the pressure must increase above a second threshold. Similarly, a heating element is controlled to maintain an operator-specified temperature within a given range. Finally, an

unknown voltage is measured, scaled into millivolts, then displayed on an LED read-out. Again, note that the CPU just reads the latest conversion results.

The total time to complete the entire queue is calculated as follows:

time per entry = (no. of bits ∗ SCK period) + DSCKL period + DTL period
= (10 ∗ 500 ns) + 1.4375 ms + 22 ms
= 28.4375 ms
time per wrap = (no. of entries) ∗ time per entry
= 3 ∗ 28.4 ms
= 85.3 ms

The age of the oldest result is calculated as follows:

maximum age = [time per entry ∗ (no. of entries + 1)] + sample time
sample time = 6 ∗ SCK period = 6 ∗ 500 ns = 3 ms
maximum age = [28.4 ms ∗ (3 + 1)] + 3 ms = 116.75 ms

The maximum-age equation accounts for the fact that the analog level may change while sampling, conversion, and transfer occurs. If the sample time is not considered, the oldest data is simply the sum of the time per wrap and the time per entry because the A/D result data always emerges on the transfer following the transfer requesting the conversion.

A.7 Other Useful Concepts

If the QSPI is to be used to control another peripheral in addition to an A/D converter, it may be advisable to interleave the transfers to the two peripherals. Interleaving can improve the overall serial transfer rate (queue entries per second) by constructively utilizing the time ordinarily wasted waiting for a conversion.

If faster data acquisition is necessary, this concept can also apply to a second A/D converter. The conversion workload must be split between the two A/D converters so that one is sampling while the other is converting, reducing the average time between conversions from 28.4 ms to 14.2 ms. If three A/D converters are employed, the time drops to 9.5 ms. If a fourth A/D converter is used, the total acquisition time is reduced to the theoretical minimum value, 7.5 ms. The theoretical minimum is the sum of the transfer time (5 ms), the minimum DSCK time (1.4375 ms), and the minimum delay after transfer (1.0625 ms).

Another useful feature of the QSPI is the ability to support subqueues. Subqueues are formed when the normal queue execution sequence is altered to perform a special task. Often, the special task needs attention as soon as possible. Afterward, it is usually desirable to resume execution of the previously defined queue.

An example would be the continuous scanning of three A/D converter channels (as previously described), but upon detection of an interrupt, quickly setting an output port to a given value. After the output data is transferred, the QSPI should continue scanning the three A/D channels. This operation is easy due to the branching capability of

the QSPI. While the QSPI is operating, writing to the NEWQP field (lower byte of SPCR2) will cause the QSPI to complete the transfer already in progress, then execute the transfer specified by NEWQP. Normal operation (transferring queue entries in sequence) continues from the point indicated by NEWQP. If a new ENDQP value is also written, its value is used to determine the end of the queue. There is no implicit return mechanism, but if the queue is properly structured, the original operation will resume automatically.

Figure A-9 shows the queue structure and operation flow that demonstrates this capability. Assuming the QSPI is already in operation (scanning A/D channels 3, 4, and 6) when the interrupt arrives, the software merely sets up the QSPI RAM associated with the special event, then writes $0E to the lower byte of SPCR2. This procedure causes the QSPI to complete the present transfer, then transfer queue entries E and F. Since ENDQP is still two, the QSPI will then transfer entries 0, 1, and 2, then wrap back to entry 0. The software never has to modify any control registers or respond to QSPI interrupts because the original queue is resumed automatically. For minimum latency, the program should initialize the control RAM (and the transmit RAM, if possible) for the special operation before the operation is to occur to initiate the subqueue transfer.

A.8 References

The following are resources which contain further information on the topics discussed in this application note.

1. Harman, Thomas L. *The Motorola MC68020 and MC68030 Microprocessors: Assembly Language, Interfacing, and Design.* Englewood Cliffs, NJ: Prentice-Hall, 1989.
2. *MC68332 User's Manual* (MC68332 UM/AD). Motorola, Inc., 1990.
3. *8-Bit A/D Converters with Serial Interface* (MC145040/D). Motorola, Inc., 1990.
4. *10-Bit A/D Converters with Serial Interface* (MC145050/D). Motorola, Inc., 1990.

```
                        ********************************************************************************
                        ********************************************************************************
                        *          Example showing use of QSPI to control 3 A/D conversions
                        *
                        *          All timing numbers assume system clock frequency of 16.000 MHz
                        ********************************************************************************
                        ********************************EQUATES*****************************************
                        ********************************************************************************
                        *
                        ******     QSPI bit definitions (just what's needed for this example)
                        *
00000080                CONT       EQU      $80 control RAM structure
00000040                BITSE      EQU      $40
00000020                DT         EQU      $20
00000010                DSCK       EQU      $10
00000008                PCS3       EQu      $08
00000004                PCS2       EQU      $04
00000002                PCS1       EQU      $02
00000001                PCS0       EQU      $01
                        *
00000008                REGCSO     EQU      $08 QPDR, QPAR, QDDR
00000004                SCK        EQU      $04
00000002                MOSI       EQU      $02
00000001                MISO       EQU      $01
                        *
00008000                MSTR       EQU      $8000 SPCR0
00000400                BITS       EQU      $400
                        *
00008000                SPE        EQU      $8000 SPCR1
00000100                DSCKL      EQU      $100
                        *
00004000                WREN       EQU      $4000 SPCR2
00000100                ENDQ       EQU      $100
                        *
00000080                SPIF       EQU      $80 SPSR
                        *
                        ******     QSPI register addresses
                        *
fffffc14                QPDRW      EQU      $FFFFFC14 QPDR as aligned WORD
fffffc18                SPCR0      EQU      $FFFFFC18 control register 0
fffffc1c                SPCR2      EQU      $FFFFFC1C control register 2
fffffc1f                SPSR       EQU      $FFFFFC1F QSPI status register
                        *
                        ******     Control register initialization values
                        *
                        * QPDR, QPAR, QDDR
                        *
00000008                INQPDR     EQU      REGCS0 PCS0 default value 1
0000000F                INQPAR     EQU      REGCS0+SCK+MOSI+MISO pins assigned to QSPI
0000000E                INQDDR     EQU      REGCS0+SCK+NOSI QSPI output pins
00080f0E                INQPORT    EQU      INQPDR*$100+INQPAR*$100+INQDDR form into a LONG WORD
                        *
```

Figure A-9 Use of QSPI to Control A/D Conversions - 2 MHz A/D (Sheet 1 of 4)

```
                        * SPCR0, SPCR1
                        *
0000a804                INQS0      EQU      10*BITS+MSTR+4 master, 10 bits, CPOL,CPHA=0,0, baud=2MHz
0000970B                INQS1      EQU      23*DSCKL+SPE+11 start QSPI, DSCK=1.4375 uS, DTL=22 uS
a804970B                INQS01     EQU      INQSO*$10000+INQS1 form into long word
                        *
                        *
                        * SPCR2, SPCR3
                        *
0000420F                INQS2      EQU      2*ENDQ+WREN+$F wrap, endq = $2, newq = $F
00000000                INQS3      EQU      $0000 nothing special, same as RESET state
420f0000                INQS23     EQU      INQS2*$10000+INQS3 form into long word
                        *
                        ******     QSPI RAM addresses and initialization values
                        *
fffffD20                TXRAM0     EQU      $FFFFFD20 transmit RAM, entry 0
fffffD24                TXRAM2     EQU      $FFFFFD24 transmit RAM, entry 2
fffffD3E                TXRAMF     EQU      $FFFFFD3E transmit RAM, entry F E
                        *
fffffD40                CRAM0      EQU      $FFFFFD40 control RAM, entry 0
fffffD4F                CRAMF      EQU      $FFFFFD4F control RAM, entry F
                        *
                        ********************
                        *          QSPI RECEIVE RAM *
                        ********************
                        *          entry #
                        *          --------
fffffD00                FUELPS1    EQU      $FFFFFD00     0              QSPI location of A/D pressure result
fffffD02                TEMP       EQU      $FFFFFD02     1              QSPI location of A/D temperature result
fffffD04                V0LTAGE    EQU      $FFFFFD04     2              QSPI location of A/D voltage result
                        *
                        *
                        ********************************************
                        *          QSPI TRANSMIT RAM INITIALIZATION CONSTANTS *
                        ********************************************
                        *
                        *
                                                          TXQ entry      sensor
                                                          --------       ------
000000C0  TXR0          EQU        3*64     A/D channel 3 address 0      temperature
00000100  TXR1          EQU        4*64     A/D channel 4 address 1      voltage
00000180  TXR2          EQU        6*64     A/D channel 6 address 2      pressure
                        *
00000180  TXRF          EQU        6*64     A/D channel 6 address F      pressure
                        *
00000100                TXR01      EQU      TXR0*$10000+TXR1form into a LONG WORD
                        *
                        *          multiply A/D address by 64 to put the LSB into bit 6 of the 10-bit transfer
                        *          (MSB of the 4-bit A/D address will be MSB of 10-bit transfer)
                        *
                        *          NOTE: transmit queue entry 0 requests a conversion on A/D channel 3,
                        *          the temperature sensor. This result will be returned into receive
                        *          RAM in queue entry 1. The A/D result always gets transmitted
                        *          on the A/D transfer following its request.
                        *
```

Figure A-9 Use of QSPI to Control A/D Conversions - 2 MHz A/D (Sheet 2 of 4)

```
                        ******************************************
                        *QSPI CONTROL RAM INITIALIZATION CONSTANTS*
                        ******************************************
                        *
00000070                CRXB       EQU      BITSE+DSCK+DT 10-bits, both delays - same for all transfers
                        *
00007070                CRXW       EQU      CRXB*$100+CRXBform into a WORD
70707070                CRXL       EQU      CRXW*$10000+CRXWform into a LONG WORD
                        *
                        *
                        ******     Misc.
                        *
00001388                VREF       EQU      5000          VREF is 5000 millivolts
00004000                SETPT      EQU      $4000         address of temperature setpoint variable
VARIABLE
                        *
                        ************************************************************************
                        ******************QSPI initialization and startup
                        ************************************************************************
                        *
00005000                           ORG      $5000
                        *
                        * Initialize QSPI TRANSMIT RAM *
00005000  21FC 00 C0 0100 STARTMOVE.L -TXR01,TXRAMOentries 0, 1
          FD20
00005008  31FC 0180 FD24           MOVE.W   -TXR2,TXRAM2  entry 2
0000500e  31FC 0180 FD E           MOVE.W   -TXRF,TXRAMF  entry F
                        *
                        * Initialize QSPI CONTROL RAM
                        *
00005014  21FC 70 70 70 70         MOVE.L   -CRXL,CRAMO   entries 0, 1, 2, 3 (3 is superfluous)
          FD40
0000501c  11FC 00 70 FD 4F         MOVE.B   -CRXB,CRAMF   entry F
                        *
                        * Initialize QSPI control registers, START transfers
                        *
00005022  21FC 00 08 0F 0E         MOVE.L   #INQPORT,QPDRWsetup QPDR, QPAR, QDDR
          FC14
0000502a  21FC 420F       0000             MOVE.L       #INQS23,SPCR2  setup SPCR2, SPCR3
          FC1C
00005032  21FC A804       970B     MOVE.L   #INQS01,SPCR0 setup SPCR0, SPCR1, start QSPI.
          FC18
                        *
                        *
0000503a  0838 0007 FC1F WAIT      BTST.B   #7,SPSR       wait until a valid conversion result
00005040  67f8                     BEQ.B    WAIT          is available for all channels
                        *
                        *          All data available, continue on to main program.
                        *
                        ************************************************************************
                        **********************CPU data acquisition**************************
                        ************************************************************************
                        *
                        *          The following code could be periodically executed in response
                        *          to a real-time interrupt. The interrupt could even be generated
                        *          by the QSPI, upon completion of each queue.
                        *
                        *
00005042  303c 0117       INTSRV   MOVE.W   #279,D0       load constant for minimum fuel pressure
00005046  B078 FD00                CMP. W   FUELPSI,D0    test if A/D pressure result is below minimum
0000504a  6504                     BCS.B    CHKRCV
```

Figure A-9 Use of QSPI to Control A/D Conversions 2 MHz A/D (Sheet 3 of 4)

```
0000504C  6146                      BSR.B    LOPRESS      generate fuel pressure warning
0000504C  600C                      BRA.B    CHKTEMP      speeds up interrupt service routine
                          *
00005050  303C 0145       CHKRCV    MOVE.W   #325,D0      constant for recovered fuel pressure
00005054  B078 FD00                 CMP.W    FUELPSI,D0   test if A/D pressure result is above minimum
00005058  6202                      BHI.B    CHKTEMP
                          *
0000505A  6138                      BSR.B    PRESSOK      cancel fuel pressure warning
                          *
                          *
                          *         The following code segment will control
                          *         a temperature using a 5 count deadband.
                          *
                          *
0000505c  3038 4000       CHKTEMP   MOVE.W   SETPT,D0     get temperature setpoint
00005060  5B40                      SUBQ.W   #5,D0         compute lower threshold
00005062  B078 FD02                 CMP.W    TEMP,D0      compare with A/D result
00005066  6508                      BCS.B    OK1          branch if actual temp. is above threshold
                          *
                          *
00005068  6100 002A                 BSR      HEATON       activate heater
0000506c  6000 001                  BRA      DOVOLTS      speeds up interrupt service routine
00005070  3038 4000       OK1       MOVE.W   SETPT,D0     get temperature setpoint
00005074  5A40                      ADDQ.W   #5,D0         compute upper threshold
00005076  B078 FD02                 CMP.W    EMP,D0       compare with A/D result
0000507a  6204                      BHI.B    DOVOLTS      branch if actual temp. is below threshold
                          *
0000507c  6100 0016                 BSR      HEATON       activate heater
                          *
                          *
                          *         The following code segment will measure voltage on
                          *         A/D channel 4 and scale the result into millivolts.
                          *
                          *
00005080  303C 1388       DOVOLTS   MOVE.W   #VREF,D0     load scale numerator (VREF = 5000 mV)
00005084  C0f8 FD04                 MULU.W   VOLTAGE,D0   multiply by A/D channel 4 conversion result
00005088  E088                      LSR.L    #8,D0        divide by 256
0000508a  E488                      LSR.L    #2,D0        divide by 4 (total of divide by 1024)
0000508c  4241                      CLR.W    D1
0000508e  D141                      ADDX.W   D1,D0        round for maximum accuracy, result in D0
00005090  6102                      BSR.B    DISPV        display voltage on a digital readout
                          *
                          *
00005092  4E73                      RTE                   return from interrupt service routine
                          *
                          *
00005094                  LOPRESS   EQU *                 dummy subroutines
00005094                  PRESSOK   EQU *
00005094                  HEATON    EQU *
00005094                  HEATOFF   EQU *
00005094                  DISPV     EQU *
00005094  4E75                      RTS
                          *
                          *
=======                   0 Error(s)
=======                   0 Warning(s)
```

Figure A-9 Use of QSPI to Control A/D Conversions 2 MHz A/D (Sheet 4 of 4)

QUEUE ENTRY NUMBER		TRANSMIT RAM (ADDR)CONTENTS	CONTROL RAM (ADDR) CONTENTS	RECEIVE RAM (ADDR)CONTENTS
	0	(FFFD20,1) A/D MUX. ADDR. 3	(FFFD40) 1 0 BIT, DSCK, DT ENABLES, PCS0 = 0	(FFFD00,1) A/D CHANNEL 6 RESULT
	1	(FFFD22,3) A/D MUX. ADDR. 4	(FFFD41) 1 0 BIT, DSCK, DT ENABLES, PCS0 = 0	(FFFD02,3) A/D CHANNEL 3 RESULT
ENDQP →	2	(FFFD24,5) A/D MUX. ADDR. 6	(FFFD42) 1 0 SIT, DSCK, DT ENABLES, PCS0 = 0	(FFFD04,5) A/D CHANNEL 4 RESULT
	3	(X) X	(X) X	(X) X
	4	(X) X	(X) X	(X) X
	•	•	•	•
	•	•	•	•
	•	•	•	•
	•	•	•	•
	•	•	•	•
	E	(X) X	(X) X	(X) X
NEWQP →	F	(FFFD3E,F) A/D MUX. ADDR. 6	(FFFD4F) 10 BIT, DSCK, DT ENABLES, PCSO 0	(FFFDI E,F) A/D INVALID DATA

X = DON'T CARE, UNUSED

ENTRY NUMBER			QSPI OPERATION FLOW	NOTE:
				WRTO = 0
START NEWQP →	F		REQUEST A/D CHANNEL 6, GET UNDEFINED DATA	WREN = 1
	0		REQUEST A/D CHANNEL 3, GET CHANNEL 6 RESULT	NEWQP = F
	1		REQUEST A/D CHANNEL 4, GET CHANNEL 3 RESULT	ENDQP = Z
ENDQP →	2		REQUEST A/D CHANNEL 6, GET CHANNEL 4 RESULT	
		←	SET SPIF AFTER COMPLETION OF ENTRY #2	
	0		REQUEST A/D CHANNEL 3, GET CHANNEL 6 RESULT	
	1		REQUEST A/D CHANNEL 4, GET CHANNEL 3 RESULT	
	2		REQUEST A/D CHANNEL 6, GET CHANNEL 4 RESULT	
	0		REQUEST A/D CHANNEL 3, GET CHANNEL 6 RESULT	
	1		REQUEST A/D CHANNEL 4, GET CHANNEL 3 RESULT	
	2		REQUEST A/D CHANNEL 6, GET CHANNEL 4 RESULT	

Figure A-10 Example Queue Structure and Operation Flow

QUEUE ENTRY NUMBER	TRANSMIT RAM (ADDR)CONTENTS	CONTROL RAM (ADDR) CONTENTS	RECEIVE RAM (ADDR)CONTENTS
0	(FFFD20,1) A/D MUX. ADDR. 3	(FFFD40) 10 BIT, DSCK, DT ENABLES, PCSO = 0	(FFFD00,1) A/D CHANNEL 6 RESULT
1	(FFFD22,3) A/D MUX. ADDR. 4	(FFFD41) 10 BIT, DSCK, DT ENABLES, PCSO = 0	(FFFD02,3) A/D CHANNEL 3 RESULT
ENDQP → 2	(FFFD24,5) A/D MUX. ADDR. 6	(FFFD42) 10 BIT DSCK, DT ENABLES, PCSO = 0	(FFFD04,5) A/D CHANNEL 4 RESULT
3	(X) X	(X) X	(X) X
4	(X) X	(X) X	(X) X
•	•	•	•
•	•	•	•
•	•	•	•
	•	•	•
D	(X) X	(X) X	(X) X
E	(FFFD3C,D) OUTPUT PORT DATA	(FFFD4E) 8 BIT, NO DELAYS, PCS1 = 0	(FFFDIC,D) PORT INPUT DATA
F	(FFFD3E,F) A/D MUX. ADDR. 6	(FFFD4F) 10 BIT DSCK, DT ENABLES, PCS0=0	0 (FFFD1E,F) LAST A/D CHANNEL DATA

X = DON'T CARE, UNUSED

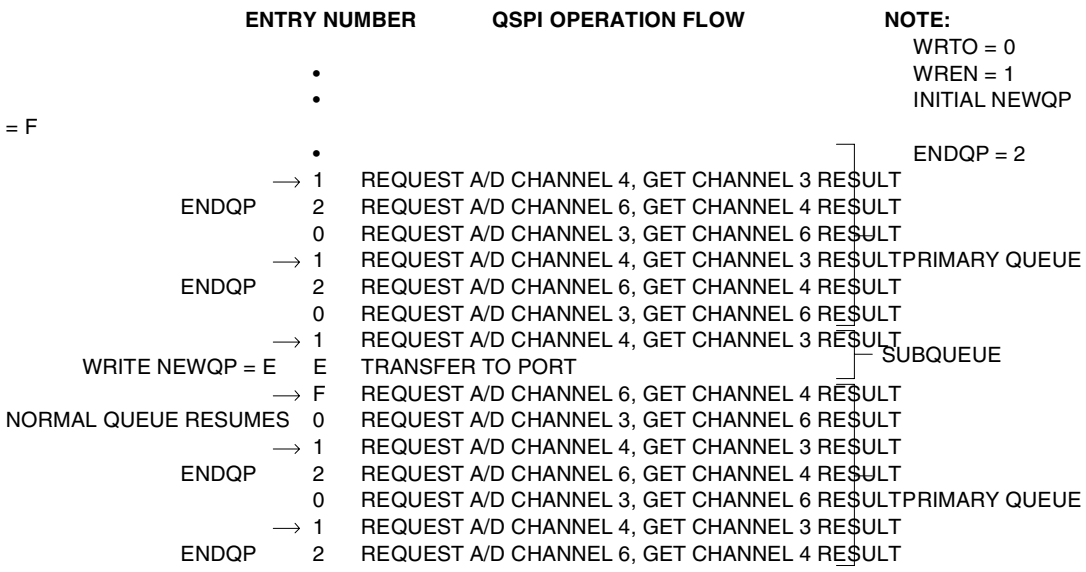
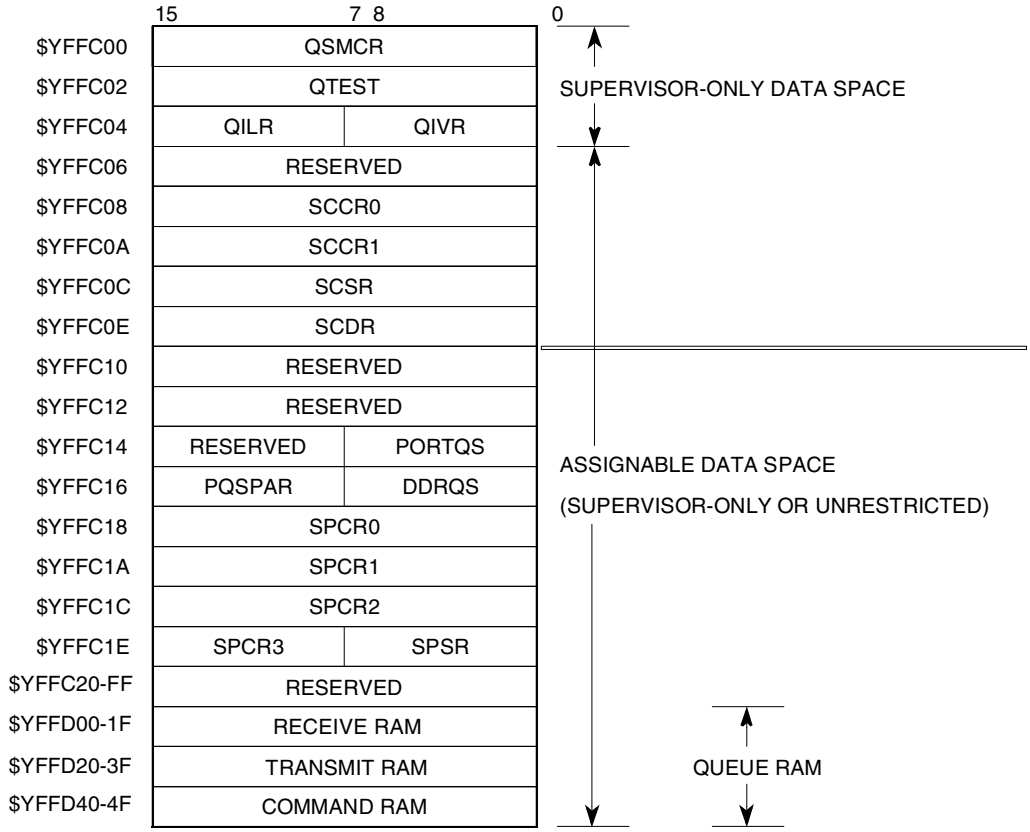


Figure A-11 Example Subqueue Structure and Operation Flow

APPENDIX B
QSM MEMORY MAP AND REGISTERS

B.1 QSM Memory Map



Y = m111 where m is the modmap bit in the SIM MCR (Y = $7 or $F).

Figure B-1 QSM Memory Map

B.2 QSM Registers

QSMCR — QSM Configuration Register **$YFFC00**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP	FRZ1	FRZ0	0	0	0	0	0	SUPV	0	0	0	IARB			
RESET:															
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

STOP — Stop Enable

1 = QSM clock operation stopped

0 = Normal QSM clock operation

FRZ1 — Freeze 1

1 = Halt the QSM (on a transfer boundary)

0 = Ignore the FREEZE signal on the IMB

FRZ0 — Freeze 0

Reserved for future enhancement.

Bits [12:8] — Not Implemented

SUPV — Supervisor/Unrestricted

1 = Supervisor access

0 = User access

Bits [6:4] — Not Implemented

IARB — Interrupt Arbitration Identification Number

System software should initialize the IARB field to a value between $F (top priority) and $1 (lowest priority). Otherwise, any interrupts generated are identified by the CPU as spurious.

QTEST — QSM Test Register **$YFFC02**

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	TSBD	SYNC	TQSM	TMM
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TSBD — SPI Test Scan Path Select

1 = Enable delay to SCK scan path

0 = Enable SPI baud clock scan path

SYNC — SCI Baud Clock Synchronization Signal

1 = Inhibit SCI source signal (QCSCI1)

0 = Activate SCI source signal

TQSM — QSM Test Enable

1 = Enable QSM to send test scan paths

0 = Disable scan path

TMM — Test Memory Map

1 = QSM responds to test memory addresses.

0 = QSM responds to QSM memory addresses.

QILR — QSM Interrupt Level Register **$YFFC04**

15	14	13	12	11	10	9	8	7	0
0	0	ILQSPI			ILSCI			QIVR*	
RESET:									
0	0	0	0	0	0	0	0		

* QIVR — QSM Interrupt Vector Register

ILQSPI — Interrupt Level for QSPI

ILQSPI determines the priority level of all QSPI interrupts. Program this field to a value

between $0 (interrupts disabled) and $7 (highest priority).

ILSCI — Interrupt Level of SCI

LSCI determines the priority level of all SCI interrupts. Program this field to a value between $0 (interrupts disabled) and $7 (highest priority).

QIVR — QSM Interrupt Vector Register **$YFFC05**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QILR*								INTV							
RESET:								0	0	0	0	1	1	1	1

* QILR — QSM Interrupt Level Register

INTV — Interrupt Vector

SCCR0 — SCI Control Register 0 **$YFFC08**

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

Bits [15:13] — Not Implemented

SCBR — Baud Rate

The SCI baud rate is programmed by writing a 13-bit value to SCBR.

SCI Baud = System Clock/(32 * SCBR)

where SCBR equals {1, 2, 3,... 8191}.

SCCR1 — SCI Control Register 1 **$YFFC0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LOOPS	WOMS	ILT	PT	PE	M	WAKE	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 15 — Not Implemented

LOOPS — LOOP Mode

LOOPS controls a feedback path on the data serial shifter.

1 = Test SCI operation, looping, feedback path enabled
0 = Normal SCI operation, no looping, feedback path disabled

WOMS — Wired-OR Mode for SCI Pins

1 = If configured as an output, TXD is an open-drain output.
0 = If configured as an output, TXD is a normal CMOS output.

ILT — Idle-Line Detect Type

1 = Long idle-line detect (starts counting when the first one is received after a stop bit(s))
0 = Short idle-line detect (starts counting when the first one is received)

PT — Parity Type
1 = Odd parity
0 = Even parity

PE — Parity Enable
1 = SCI parity enabled
0 = SCI parity disabled

M	PE	Result
0	0	8 Data Bits
0	1	7 Data Bits, 1 Parity Bit
1	0	9 Data Bits
1	1	8 Data Bits, 1 Parity Bit

M — Mode Select
1 = SCI frame: one start bit, nine data bits, one stop bit (eleven bits total)
0 = SCI frame: one start bit, eight data bits, one stop bit (ten bits total)

WAKE — Wakeup by Address Mark
1 = SCI receiver awakened by address mark (eighth or ninth (last) bit set)
0 = SCI receiver awakened by idle-line detection

TIE — Transmit Interrupt Enable
1 = SCI TDRE interrupts enabled
0 = SCI TDRE interrupts inhibited

TCIE — Transmit Complete Interrupt Enable
1 = SCI TC interrupts enabled
0 = SCI TC interrupts inhibited

RIE — Receiver Interrupt Enable
1 = SCI RDRF interrupts enabled
0 = SCI RDRF interrupts inhibited

ILIE — Idle-Line Interrupt Enable
1 = SCI IDLE interrupts enabled
0 = SCI IDLE interrupts inhibited

TE — Transmitter Enable
1 = SCI transmitter enabled; TXD pin dedicated to the SCI transmitter
0 = SCI transmitter disabled; TXD pin can be used as general-purpose I/O

RE — Receiver Enable
1 = SCI receiver enabled
0 = SCI receiver disabled

RWU — Receiver Wakeup
1 = Wakeup mode enabled, all received data ignored until awakened
0 = Normal receiver operation, all received data recognized

SBK — Send Break

1 = Break frame(s) are transmitted after completion of the current frame

0 = Normal operation

SCSR — SCI Status Register **$YFFC0C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	TDRE	TC	RDRF	RAF	IDLE	OR	NF	FE	PF
RESET:															
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Bits [15:9] — Not Implemented

TDRE — Transmit Data Register Empty Flag

1 = A new character can now be written to register TDR.

0 = Register TDR still contains data to be sent to the transmit serial shifter.

TC — Transmit Complete Flag

1 = SCI transmitter is idle

0 = SCI transmitter is busy

RDRF — Receive Data Register Full Flag

1 = Register RDR contains new data

0 = Register RDR is empty or contains previously read data

RAF — Receiver Active Flag

1 = SCI receiver is busy

0 = SCI receiver is idle

IDLE — Idle-Line Detected Flag

1 = SCI receiver detected an idle-line condition

0 = SCI receiver did not detect an idle-line condition

OR — Overrun Error Flag

1 = RDRF is not cleared before new data arrives

0 = RDRF is cleared before new data arrives

NF — Noise Error Flag

1 = Noise occurred on the received data

0 = No noise detected on the received data

FE — Framing Error Flag

1 = Framing error or break occurred on the received data

0 = No framing error on the received data

PF — Parity Error Flag

1 = Parity error occurred on the received data

0 = No parity error on the received data

SCDR — SCI Data Register **$YFFC0E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	R8/T8	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0
RESET:															
0	0	0	0	0	0	0	U	U	U	U	U	U	U	U	U

R8/T8 — Receive 8/Transmit 8

R[7:0]/T[7:0] — Receive 7–0/Transmit 7–0

QPDR — QSM Port Data Register **$YFFC15**

15	7	6	5	4	3	2	1	0
RESERVED	DATA 7 (TXD)	DATA6 (PCS3)	DATA5 (PCS2)	DATA4 (PCS1)	DATA3 (PCS0/$\overline{SS}$)	DATA2 (SCK)	DATA1 (MOSI)	DATA0 (MISO)
RESET:								
	0	0	0	0	0	0	0	0

DATA[7:0] — Pin Data

TXD/MISO — Pin Function

PQSPAR — QSM Pin Assignment Register **$YFFC16**

15	14	13	12	11	10	9	8	7	0
0	PCS3	PCS2	PCS1	PCSO/$\overline{SS}$	0	MOSI	MISO	DDRQS*	
RESET:									
0	0	0	0	0	0	0	0		

* DDRQS — QSM Port Data Direction Register
0 = General-purpose I/O
1 = QSPI module

Bit 15 — Not Implemented

PCS[3:1] — Peripheral Chip-Selects 3–1

PCS0/$\overline{SS}$ — Peripheral Chip-Select 0/Slave Select

Bit 10 — Not Implemented

MOSI — Master Out Slave In

MISO — Master In Slave Out

These bits determine whether the associated QSM port pin functions as a general-purpose I/O pin or is assigned to the QSPI submodule.

DDRQS — QSM Data Direction Register **$YFFC17**

15 … 8	7	6	5	4	3	2	1	0
PQSPAR*	TXD	PCS3	PCS2	PCS1	PCS0/$\overline{SS}$	SCK	MOSI	MISO
RESET:								
	0	0	0	0	0	0	0	0

* PQSPAR — QSM Pin Assignment Register
0 = Input
1 = Output

TXD — Transmit Data

PCS[3:1] — Peripheral Chip-Selects 3–1

PSC0/$\overline{SS}$ — Peripheral Chip-Select 0/Slave Select

SCK — Serial Clock

MOSI — Master Out Slave In

SPCR0 — QSPI Control Register 0 **$YFFC18**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSTR	WOMQ	BITS				CPOL	CPHA	SPBR							
RESET:															
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

MSTR — Master/Slave Mode Select
1 = QSPI is system master and can initiate transmission to external SPI devices.
0 = QSPI is a slave device, and only responds to externally generated serial MSTR.

WOMQ — Wired-OR Mode for QSPI Pins
1 = All QSPI port pins designated as output by DDRQS function as open-drain outputs.
0 = Output pins have normal outputs instead of open-drain outputs.

BITS — Bits Per Transfer
In master mode, BITS determines the number of data bits transferred for each serial transfer in the queue.

Bit 13	Bit 12	Bit 11	Bit 10	Bits per Transfer
0	0	0	0	16
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	Reserved
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Reserved
0	1	1	1	Reserved
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

CPOL — Clock Polarity

1 = The inactive state value of SCK is high.

0 = The inactive state value of SCK is low.

CPHA — Clock Phase

1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.

0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK.

SPBR — Serial Clock Baud Rate

The QSPI internally generates the baud rate for SCK, the frequency of which is programmable by the user. The following equation determines the SCK baud rate:

SCK Baud Rate = System Clock/(2 * SPBR)

or

SPBR = System Clock/(2 * SCK Baud Rate Desired)

where SPBR equals {2, 3, 4,..., 255}.

SPCR1 — QSPI Control Register 1 **$YFFC1A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPE	DSCKL							DTL							
RESET:															
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

SPE — QSPI Enable

1 = The QSPI is enabled and the pins allocated by QSM register PQSPAR are controlled by the QSPI.

0 = The QSPI is disabled and the seven QSPI pins can be used as general-purpose I/O pins, regardless of the values in PQSPAR.

DSCKL — Delay before SCK

This bit determines the length of time the QSPI delays from peripheral chip-select (PCS) valid to SCK transition for serial transfers in which the command control bit, DSCK of the QSPI RAM, equals one.

PCS to SCK Delay = [DSCKL/System Clock Frequency]

where DSCKL equals {1,2,3,... 127}.

DTL — Length of Delay after Transfer

These bits determine the length of time that the QSPI delays after each serial transfer in which the command control bit, DT of the QSPI RAM, equals one.

Delay after Transfer = [(32 * DTL)/System Clock Frequency]

where DTL equals {1,2,3,... 255}.

SPCR2 — QSPI Control Register 2 **$YFFC1C**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPIFIE	WREN	WRTO	0	ENDQP				0	0	0	0	NEWQP			
RESET:															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SPIFIE — SPI Finished Interrupt Enable

1 = QSPI interrupts enabled
0 = QSPI interrupts disabled

WREN — Wrap Enable

1 = Wraparound mode enabled
0 = Wraparound mode disabled

WRTO — Wrap To

1 = Wrap to address found in NEWQP
0 = Wrap to address $0

Bit 12 — Not Implemented

ENDQP — Ending Queue Pointer

This field determines the last absolute address in the queue to be completed by the QSPI.

Bits [7:4] — Not Implemented

NEWQP — New Queue Pointer Value

NEWQP determines which queue entry the QSPI transfers first.

SPCR3 — QSPI Control Register 3 **$YFFC1E**

15	14	13	12	11	10	9	8	7	0
0	0	0	0	0	LOOPQ	HMIE	HALT	SPSR*	
RESET:									
0	0	0	0	0	0	0	0		

* SPSR — QSPI Status Register

Bits [15:11] — Not Implemented

LOOPQ — QSPI Loop Mode
1 = Feedback path enabled
0 = Feedback path disabled

HMIE — HALTA and MODF Interrupt Enable
1 = HALTA and MODF interrupts enabled
0 = HALTA and MODF interrupts disabled

HALT — Halt
1 = Halt enabled
0 = Halt not enabled
This bit is used by the CPU to stop the QSPI on a queue boundary.

SPSR — QSPI Status Register **$YFFC1F**

15 8	7	6	5	4	3	2	1	0
SPCR3*	SPIF	MODF	HALTA	0	CPTQP			
RESET:	0	0	0	0	0	0	0	0

* SPCR3 — QSPI Control Register 3

SPIF — QSPI Finished Flag
1 = QSPI finished
0 = QSPI not finished
SPIF is set when the QSPI finishes executing the last command determined by the address contained in ENDQP in SPCR2.

MODF — Mode Fault Flag
1 = Another SPI node requested to become the network SPI master while the QSPI was enabled in master mode (MSTR = 1), or the PCS0/$\overline{SS}$ pin was incorrectly pulled low by external hardware.
0 = Normal operation

HALTA — Halt Acknowledge Flag
1 = QSPI halted
0 = QSPI not halted
HALTA is asserted by the QSPI when it has come to an orderly halt at the request of the CPU, through the assertion of HALT.

Bit 4 — Not Implemented

CPTQP — Completed Queue Pointer
CPTQP contains the queue pointer value of the last command in the queue that was completed.

COMMAND RAM **$YFFD40**

7	6	5	4	3	2	1	0
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*
—	—	—	—	—	—	—	—
CONT	BITSE	DT	DSCK	PCS3	PCS2	PCS1	PCS0*

$YFFD4F

COMMAND CONTROL PERIPHERAL CHIP-SELECT

*The PCS0 bit represents the dual-function PCS0/$\overline{SS}$.

CONT — Continue

1 = Keeps peripheral chip-selects asserted after transfer is complete.

0 = Returns control of peripheral chip-selects to QPDR after transfer is complete.

BITSE — Bits Enable

1 = Number of bits to transfer defined in BITS field of SPCR0.

0 = Eight bits to transfer

DT — Delay After Transfer

1 = Delay

0 = No delay

DSCK — PCS to SCK Delay

1 = DSCKL field in SPCR1 specifies value of delay from PCS valid to SCK

0 = PCS valid to SCK transition is 1/2 SCK

PCS[3:0]/$\overline{SS}$ — Peripheral Chip-Select

The four peripheral chip-select bits can be used directly to select one of four external chips for the serial transfer, or decoded by external hardware to select one of 16 chip-select patterns for a serial transfer.

INDEX

–N–

–O–

–P–

–Q–

–R–

–S–

–T–

–V–

–W–