# Master Slave Interface (MSI) Module

## HIGHLIGHTS

This section of the manual contains the following major topics:

| | |
|---|---|
| **Note:** | This family reference manual section is meant to serve as a complement to device data sheets. This document applies to all dsPIC33/PIC24 devices.<br><br>Please consult the note at the beginning of the **"Master Slave Interface (MSI)"** chapter in the current device data sheet to check whether this document supports the device you are using.<br><br>Device data sheets and family reference manual sections are available for download from the Microchip Worldwide Web site at: http://www.microchip.com. |

## 1.0 INTRODUCTION

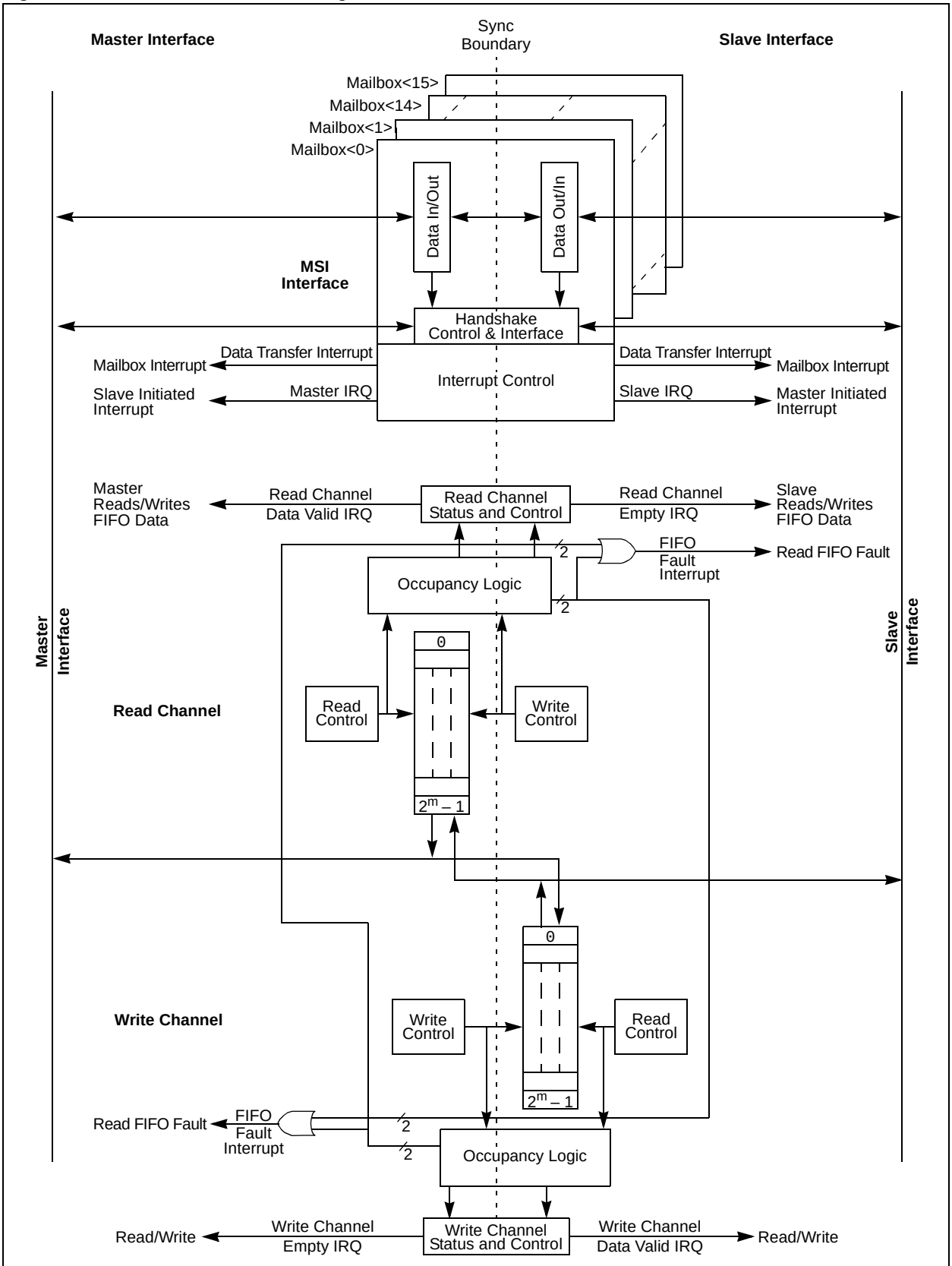
The Master Slave Interface (MSI) module is a bridge between the Master and a Slave processor system, each of which operate within independent clock domains.

The Master and Slave have their own registers to communicate between the MSI modules; the Master MSI registers are located in the Master SFR space and the Slave MSI registers are in the Slave SFR space.

The Master Slave Interface (MSI) includes these characteristics:

- 16 Unidirectional Data Mailbox Registers:
  - Direction of each Mailbox register is fuse-selectable
  - Byte and word-addressable
- 8 Mailbox Data Flow Control Protocol Blocks:
  - Individual fuse enables
  - Write port active; read port passive (i.e., no read data request required)
  - Automatic, interrupt driven (or polled), data flow control mechanism across MSI clock boundary
  - Fuse assignable to any of the Mailbox registers; supports any length data buffers (up to the number of available Mailbox registers)
  - DMA transfer compatible
- Master to Slave and Slave to Master Interrupt Request with Acknowledge Data Flow Control
- Optional 2-Channel FIFO Memory Structure
- FIFO Depth of 16-128 Words (verify with the data sheet for the actual implemented FIFO depth):
  - 1 read and 1 write channel
  - Circular operation with empty and full status and interrupts
  - Overflow/underflow detection with interrupts to Master and Slave
  - Interrupt-based, software polled or DMA transfer compatible
- Master and Slave Processor Cross-Boundary Control and Status:
  - Readable Operating mode status for both processors
  - Slave enable from Master (subject to satisfying a hardware write interlock sequencer)
  - Master interrupts when Slave is reset during code execution
  - Slave interrupts when Master is reset during code execution
- Optional (fuse) Decoupling of Master and Slave Resets, POR/BOR/$\overline{\text{MCLR}}$ always Resets Master and Slave; the Influence of the remaining Run-Time Resets on the Slave Enable is Fuse-Programmable

**Figure 1-1: MSI Module Block Diagram**




**Note:** Refer to the specific data sheet to determine the depth of the FIFO.

# 2.0 MASTER SLAVE CONFIGURATION REGISTERS

## 2.1 MSI Master Configuration Registers

The following registers are associated with the MSI Master module and are located in the Master SFR space:

- Register 2-1: MSI1CON
- Register 2-2: MSI1STAT
- Register 2-3: MSI1KEY
- Register 2-4: MSI1MBXS
- Register 2-5: MSI1MBXnD
- Register 2-6: MSI1FIFOCS
- Register 2-7: MRSWFDATA
- Register 2-8: MWSRFDATA

### 2.1.1 REGISTER MAP

Table 2-1 provides a brief summary of the related MSI Master module registers. The corresponding registers and their detailed descriptions appear after this summary.

**Table 2-1: MSI Master Register Map**

| Register Name | Bit Range | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MSI1CON | 15:0 | SLVEN | — | — | — | RFITSEL1 | RFITSEL0 | MTSIRQ | STMIACK | SRSTIE | r | r | r | r | r | r | r |
| MSI1STAT | 15:0 | SLVRST | SLVWDRST | SLVPWR1 | SLVPWR0 | VERFERR | SLVP2ACT | STMIRQ | MTSIACK | SLVDBG | — | — | — | — | — | — | — |
| MSI1KEY | 15:0 | — | — | — | — | — | — | — | — | MSI1KEY<7:0> | | | | | | | |
| MSI1MBXS | 15:0 | — | — | — | — | — | — | — | — | DTRDY<H:A> | | | | | | | |
| MSI1MBXnD | 15:0 | MSI1MBXnD<15:0> | | | | | | | | | | | | | | | |
| MSI1FIFOCS | 15:0 | WFEN | — | — | — | WFOF | WFUF | WFFULL | WFEMPTY | RFEN | — | — | — | RFOF | RFUF | RFFULL | RFEMPTY |
| MRSWFDATA | 15:0 | MRSWFDATA<15:0> | | | | | | | | | | | | | | | |
| MWSRFDATA | 15:0 | MWSRFDATA<15:0> | | | | | | | | | | | | | | | |

**Legend:** — = unimplemented, read as '0'; r = reserved bit.

**Register 2-1: MSI1CON: MSI1 Master Control Register**

| R/W-0 | U-0 | U-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| SLVEN | — | — | — | RFITSEL1 | RFITSEL0 | MTSIRQ | STMIACK |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |
|---|---|---|---|---|---|---|---|
| SRSTIE | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | r = Reserved bit | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15 **SLVEN:** Slave Enable bit
1 = Slave processor is enabled, Slave Reset is released and execution permitted
0 = Slave processor is disabled and held in Reset

bit 14-12 **Unimplemented:** Read as ‘0’

bit 11-10 **RFITSEL<1:0>:** Read FIFO Interrupt Threshold Select bits
11 = Triggers data valid interrupt when FIFO is full after Slave write
10 = Triggers data valid interrupt when FIFO is 75% full after Slave write
01 = Triggers data valid interrupt when FIFO is 50% full after Slave write
00 = Triggers data valid interrupt when 1st FIFO entry is written by Slave

bit 9 **MTSIRQ:** Master to Slave Interrupt Request bit
1 = Master has issued an interrupt request to the Slave
0 = Master has not issued a Slave interrupt request

bit 8 **STMIACK:** Interrupt Acknowledge bit (to Acknowledge the Slave interrupt)
1 = If STMIRQ = 1: Master Acknowledges Slave interrupt request, else protocol error
0 = If STMIRQ = 1: Master has not yet Acknowledged Slave interrupt request, else no Slave to Master interrupt request is pending

bit 7 **SRSTIE:** Slave Reset Event Interrupt Enable bit
1 = Master Slave Reset event interrupt occurs when Slave enters the Reset state
0 = Master Slave Reset event interrupt does not occur when Slave enters the Reset state

bit 6-0 **Reserved:** Maintain as ‘0’

**Register 2-2: MSI1STAT: MSI1 Master Status Register**

| R-0 | R/HS/C-0 | R-0 | R-0 | R/HS/C-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| SLVRST | SLVWDRST(1) | SLVPWR1 | SLVPWR0 | VERFERR | SLVP2ACT | STMIRQ | MTSIACK |
| bit 15 | | | | | | | bit 8 |

| R-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| SLVDBG | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | HS = Hardware Settable bit | C = Clearable bit | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15 **SLVRST:** Slave Reset Status bit
1 = Slave is in Reset
0 = Slave is not in Reset

bit 14 **SLVWDRST:** Slave Watchdog Timer (WDT) Reset Status bit(1)
1 = Slave has been reset by the Slave WDT
0 = Slave has not been reset by the WDT

bit 13-12 **SLVPWR<1:0>:** Slave Low-Power Operating Mode Status bits
11 = Slave is in Deep Sleep mode
10 = Slave is in Sleep mode
01 = Slave is in Idle mode
00 = Slave is not in a Low-Power mode

bit 11 **VERFERR:** PRAM Verify Error Status bit
1 = Error detected during execution of VFSLV (PRAM write verify) instruction
0 = No error detected during execution of VFSLV (PRAM write verify) instruction

bit 10 **SLVP2ACT:** Slave PRAM Panel 2 Active Status bit
This bit is a reflection of the Slave NVM Controller Status bit, P2ACTIV (NVMCON<10>), which is toggled after successful execution of a BOOTSWP instruction (during a Slave PRAM Live Update operation).
1 = Slave NVM Controller Status bit, P2ACTIV = 1
0 = Slave NVM Controller Status bit, P2ACTIV = 0

bit 9 **STMIRQ:** Slave to Master Interrupt Request Status bit
1 = Slave has issued an interrupt request to the Master
0 = Slave has not issued a Master interrupt request

bit 8 **MTSIACK:** Interrupt Acknowledge Status bit (Slave Acknowledged)
1 = If MTSIRQ = 1: Slave Acknowledges Master interrupt request, else protocol error
0 = If MTSIRQ = 1, Slave has not yet Acknowledged Master interrupt request, else no Master to Slave interrupt request is pending

bit 8 **SLVDBG:** Slave Debug Mode Status bit
1 = Slave is operating in Debug mode
0 = Slave is operating in Mission or Application mode

bit 6-0 **Unimplemented:** Read as '0'

**Note 1:** This bit is set by hardware and cleared by software or POR/BOR Reset. This bit is unaffected should the Slave be disabled (SLVEN (MSI1CON<15> = 0).

**Register 2-3: MSI1KEY: MSI1 Master Interlock Key Register(1)**

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |
|---|---|---|---|---|---|---|---|
| MSI1KEY<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15-8 **Unimplemented:** Read as '0'

bit 7-0 **MSI1KEY<7:0>:** MSI1 Key bits
The MSI1KEY<7:0> bits are monitored for specific write values.

**Note 1:** This is not a physical register; register reads always result in 00h.

**Register 2-4: MSI1MBXS: MSI1 Master Mailbox Data Transfer Status Register**

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| DTRDY<H:A> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-8 **Unimplemented:** Read as ‘0’

bit 7-0 **DTRDY<H:A>:** Data Ready Status bits

1 = Data transmitter has indicated that data is available to be read by data receiver in MSI1MBXnD (DTRDYx is automatically set by a data transmitter processor write to the assigned MSI1MBXnD). Meaning when configured as a:
- Transmitter: Data is written. Waiting for receiver to read.
- Receiver: New data is ready to read.

0 = No data is available to be read in receiver, MSI1MBXnD (or the handshake protocol logic block is disabled)

**Register 2-5: MSI1MBXnD: MSI1 Master Mailbox n Data Register (Master, n = 0 to 15)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| MSI1MBXnD<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| MSI1MBXnD<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-0 **MSI1MBXnD<15:0>:** MSI1 Master Mailbox n Data bits

When Configuration bit, FMBXMx = 1 (programmed):
Mailbox Data Direction: Master read, Slave writes Master; MSI1MBXnD<15:0> become R-0 (a Master write to MSI1MBXnD<15:0> will have no effect).

When Configuration bit, FMBXMx = 0 (programmed):
Mailbox Data Direction: Master write, Slave reads Master; MSI1MBXnD<15:0> becomes R/W-0.

**Register 2-6: MSI1FIFOCS: MSI1 Master FIFO Control/Status Register 1**

| R/W-0 | U-0 | U-0 | U-0 | R/C-0 | R-0 | R-0 | R-1 |
|---|---|---|---|---|---|---|---|
| WFEN(1) | — | — | — | WFOF(2) | WFUF(2) | WFFULL(2) | WFEMPTY(2) |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | U-0 | U-0 | U-0 | R-0 | R/C-0 | R-0 | R-1 |
|---|---|---|---|---|---|---|---|
| RFEN | — | — | — | RFOF | RFUF | RFFULL | RFEMPTY |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | C = Clearable bit | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15 **WFEN:** Write FIFO Enable bit(1)
1 = Enables (Master) Write FIFO
0 = Disables and initializes (Master) Write FIFO

bit 14-12 **Unimplemented:** Read as '0'

bit 11 **WFOF:** Write FIFO Overflow bit(2)
1 = Write FIFO overflow is detected
0 = No Write FIFO overflow is detected

bit 10 **WFUF:** Write FIFO Underflow bit(2)
1 = Write FIFO underflow is detected
0 = No Write FIFO underflow is detected

bit 9 **WFFULL:** Write FIFO Full Status bit(2)
1 = Write FIFO is full; last write by Master to Write FIFO (WFDATA) was into the last free location
0 = Write FIFO is not full

bit 8 **WFEMPTY:** Write FIFO Empty Status bit(2)
1 = Write FIFO is empty; last read by Slave from Write FIFO (WFDATA) emptied the FIFO of all valid data or FIFO is disabled (and initialized to the empty state)
0 = Write FIFO contains valid data not yet read by the Slave

bit 7 **RFEN:** Read FIFO Enable bit
1 = Enables (Master) Read FIFO
0 = Disables and initializes (Master) Read FIFO

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **RFOF:** Read FIFO Overflow bit
1 = Read FIFO overflow is detected
0 = No Read FIFO overflow is detected

bit 2 **RFUF:** Read FIFO Underflow bit
1 = Read FIFO underflow is detected
0 = No Read FIFO underflow is detected

bit 1 **RFFULL:** Read FIFO Full Status bit
1 = Read FIFO is full; last write by Slave to Read FIFO (RFDATA) was into the last free location
0 = Read FIFO is not full

bit 0 **RFEMPTY:** Read FIFO Empty Status bit
1 = Read FIFO is empty; last read by Master from Read FIFO (RFDATA) emptied the FIFO of all valid data or FIFO is disabled (and initialized to the empty state)
0 = Read FIFO contains valid data not yet read by the Master

**Note 1:** Clearing WFEN will also cause the WFEMPTY status bit to be set. After WFEN is subsequently set, WFEMPTY will remain set until the Master writes data into the Write FIFO.

**2:** Once set, these bits can be cleared by making WFEN = 0.

**Register 2-7: MRSWFDATA: Master Read (Slave Write) FIFO Data Register**

| R/W-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| MRSWFDATA<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| MRSWFDATA<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-0 **MRSWFDATA<15:0>:** Read FIFO Data Out Register bits

**Register 2-8: MWSRFDATA: Master Write (Slave Read) FIFO Data Register**

| R/W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |
|---|---|---|---|---|---|---|---|
| MWSRFDATA<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |
|---|---|---|---|---|---|---|---|
| MWSRFDATA<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-0 **MWSRFDATA<15:0>:** Write FIFO Data In Register bits

## 2.2 MSI Slave Configuration Registers

The following registers are associated with the Slave MSI module and are located in the Slave SFR space:

- Register 2-9: SI1CON
- Register 2-10: SI1STAT
- Register 2-11: SI1MBXS
- Register 2-12: SI1MBXnD
- Register 2-13: SI1FIFOCS
- Register 2-14: SWMRFDATA
- Register 2-15: SRMWFDATA

### 2.2.1 REGISTER MAP

Table 2-1 provides a brief summary of the related MSI Slave module registers. The corresponding registers and their detailed descriptions appear after this summary.

**Table 2-2: MSI Slave Register Map**

| Register Name | Bit Range | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SI1CON | 15:0 | — | — | — | — | RFITSEL1 | RFITSEL0 | STMIRQ | MTSIACK | MRSTIE | r | r | r | r | r | r | r |
| SI1STAT | 15:0 | MSTRST | — | MSTPWR1 | MSTPWR0 | — | — | MTSIRQ | STMIACK | — | — | — | — | — | — | — | — |
| SI1MBXS | 15:0 | — | — | — | — | — | — | — | — | DTRDY<H:A> | | | | | | | |
| SI1MBXnD | 15:0 | SI1MBXnD<15:0> | | | | | | | | | | | | | | | |
| SI1FIFOCS | 15:0 | SRFEN | — | — | — | SRFOF | SRFUF | SRFFULL | SRFEMPTY | SWFEN | — | — | — | SWFOF | SWFUF | SWFFULL | SWFEMPTY |
| SWMRFDATA | 15:0 | SWMRFDATA<15:0> | | | | | | | | | | | | | | | |
| SRMWFDATA | 15:0 | SRMWFDATA<15:0> | | | | | | | | | | | | | | | |

**Legend:** — = unimplemented, read as '0'; r = reserved bit.

**Register 2-9: SI1CON: MSI1 Slave Control Register**

| U-0 | U-0 | U-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | RFITSEL1 | RFITSEL0 | STMIRQ | MTSIACK |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 | r-0 |
|---|---|---|---|---|---|---|---|
| MRSTIE | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | r = Reserved bit | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-12 **Unimplemented:** Read as ‘0’

bit 11-10 **RFITSEL<1:0>:** Read FIFO Interrupt Threshold Select bits

11 = Triggers data valid interrupt when FIFO is full after Master write
10 = Triggers data valid interrupt when FIFO is 75% full after Master write
01 = Triggers data valid interrupt when FIFO is 50% full after Master write
00 = Triggers data valid interrupt when 1st FIFO entry is written by Master

bit 9 **STMIRQ:** Slave to Master Interrupt Request bit

1 = Slave issues an interrupt request to the Master
0 = Slave does not issue a Master interrupt request

bit 8 **MTSIACK:** Master to Slave Interrupt Acknowledge bit

1 = If MTSIRQ = 1: Slave Acknowledges a Master interrupt request, else protocol error
0 = If MTSIRQ = 1: Slave has not yet Acknowledged a Master interrupt request, else no Slave to Master interrupt request is pending

bit 7 **MRSTIE:** Master Reset Event Interrupt Enable bit

1 = Slave Master Reset event interrupt occurs when Master enters the Reset state
0 = Slave Master Reset event interrupt does not occur when Master enters the Reset state

bit 6-0 **Reserved:** Maintain as ‘0’

**Register 2-10: SI1STAT: MSI1 Slave Status Register**

| R-0 | U-0 | R-0 | R-0 | U-0 | U-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| MSTRST | — | MSTPWR1 | MSTPWR0 | — | — | MTSIRQ | STMIACK |
| bit 15 | | | | | | | bit 8 |

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15 **MSTRST:** Master Reset Status bit
1 = Master is in Reset
0 = Master is not in Reset

bit 14 **Unimplemented:** Read as ‘0’

bit 13-12 **MSTPWR<1:0>:** Master Low-Power Operating Mode Status bits
11 = Reserved
10 = Master is in Sleep mode
01 = Master is in Idle mode
00 = Master is not in a Low-Power mode

bit 11-10 **Unimplemented:** Read as ‘0’

bit 9 **MTSIRQ:** Master Interrupted Slave bit
1 = Master has issued an interrupt request to the Slave
0 = Master has not issued a Slave interrupt request

bit 8 **STMIACK:** Master Acknowledgment Status bit
1 = If STMIRQ = 1: Master Acknowledges Slave interrupt request, else protocol error
0 = If STMIRQ = 1: Master has not yet Acknowledged Slave interrupt request, else no Slave to Master interrupt request is pending

bit 7-0 **Unimplemented:** Read as ‘0’

**Register 2-11: SI1MBXS: MSI1 Slave Mailbox Data Transfer Status Register**

| U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |
| bit 15 | | | | | | | bit 8 |

| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| DTRDY<H:A> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | HS = Hardware Settable bit | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15-8 **Unimplemented:** Read as '0'

bit 7-0 **DTRDY<H:A>:** Data Ready Status bits

1 = Data transmitter has indicated that data is available to be read by data receiver in MSI1MBXnD (DTRDYx is automatically set by a data transmitter processor write to the assigned MSI1MBXnD). Meaning when configured as a:
- Transmitter: Data is written. Waiting for receiver to read.
- Receiver: New data is ready to read.

0 = No data is available to be read in receiver, MSI1MBXnD (or the handshake protocol logic block is disabled in the Configuration bits)

**Register 2-12: SI1MBXnD: MSI1 Slave Mailbox n Data Register (Slave, n = 0 to 15)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| SI1MBXnD<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| SI1MBXnD<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15-0 **SI1MBXnD<15:0>:** MSI1 Slave Mailbox n Data bits

When Configuration bit, FMBXMx = 1 (programmed):
Mailbox Data Direction: Master read, Slave writes Master; SI1MBXnD<15:0> become R-0 (a Master write to SI1MBXnD<15:0> will have no effect).

When Configuration bit, FMBXMx = 0 (programmed):
Mailbox Data Direction: Master write, Slave reads Master; SI1MBXnD<15:0> become R/W-0.

**Register 2-13: SI1FIFOCS: MSI1 Slave FIFO Status Register**

| R-0 | U-0 | U-0 | U-0 | R/C-0 | R/C-0 | R-0 | R-1 |
|---|---|---|---|---|---|---|---|
| SRFEN(1,2) | — | — | — | SRFOF(3) | SRFUF | SRFFULL(4) | SRFEMPTY |
| bit 15 | | | | | | | bit 8 |

| R-0 | U-0 | U-0 | U-0 | R-0 | R/C-0 | R-0 | R-1 |
|---|---|---|---|---|---|---|---|
| SWFEN | — | — | — | SWFOF | SWFUF | SWFFULL | SWFEMPTY |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | C = Clearable bit | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 15 **SRFEN:** Slave Read (Master Write) FIFO Enable bit(1,2)
1 = Enables Slave Read FIFO (Master Write)
0 = Disables Slave Read FIFO (Master Write)

bit 14-12 **Unimplemented:** Read as '0'

bit 11 **SRFOF:** Slave Read (Master Write) FIFO Overflow bit(3)
1 = Slave Read FIFO overflow is detected
0 = No Slave Read FIFO overflow is detected

bit 10 **SRFUF:** Slave Read (Master Write) FIFO Underflow bit
1 = Slave Read (Master Write) FIFO underflow is detected
0 = No Slave Read (Master Write) FIFO underflow is detected

bit 9 **SRFFULL:** Slave Read (Master Write) FIFO Full Status bit(4)
1 = Slave Read (Master Write) FIFO is full; last write by Master to Slave Read FIFO (SRMWFDATA) was into the last free location
0 = Slave Read (Master Write) FIFO is not full

bit 8 **SRFEMPTY:** Slave Read (Master Write) FIFO Empty Status bit
1 = Slave Read (Master Write) FIFO is empty; last read by Slave from Read FIFO (SRMWFDATA) emptied the FIFO of all valid data or FIFO is disabled (and initialized to the empty state)
0 = Slave Read (Master Write) FIFO contains valid data not yet read by the Slave

bit 7 **SWFEN:** Slave Write (Master Read) FIFO Enable bit
1 = Enables Slave Write (Master Read) FIFO
0 = Disables Slave Write (Master Read) FIFO

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **SWFOF:** Slave Write (Master Read) FIFO Overflow bit
1 = Slave Write (Master Read) FIFO overflow is detected
0 = No Slave Write (Master Read) FIFO overflow is detected

bit 2 **SWFUF:** Slave Write (Master Read) FIFO Underflow bit
1 = Slave Write (Master Read) FIFO underflow is detected
0 = No Slave Write (Master Read) FIFO underflow is detected

**Note 1:** SRFEN is a read-only bit that gets set when the Master enables its Write FIFO (WFEN (MSI1FIFOCS<15> = 1). The bit will be cleared only when the Master clears the WFEN bit.

**2:** SRFEN bit is set when the Master sets RFEN (MSIFIFOCS<7>) = 1.

**3:** Overflow bit is set when the buffer is full and the Master sends one more data without the Slave reading the SRMWFDATA register.

**4:** SRFFULL bit is set when the Slave read buffer is full. It can be cleared when the Slave reads the buffer (using the SRMWFDATA register).

**Register 2-13: SI1FIFOCS: MSI1 Slave FIFO Status Register (Continued)**

bit 1 **SWFFULL:** Slave Write FIFO (Master Read) Full Status bit
- 1 = Slave Write FIFO (Master Read) is full; last write by Slave to FIFO (SWMRFDATA) was into the last free location
- 0 = Slave Write (Master Read) FIFO is not full

bit 0 **SWFEMPTY:** Slave Write FIFO (Master Read) Empty Status bit
- 1 = Slave Write FIFO (Master Read) is empty; last read by Master from Read FIFO emptied the FIFO of all valid data or the FIFO is disabled (and initialized to the empty state)
- 0 = Slave Write FIFO (Master Read) contains valid data not yet read by the Master

**Note 1:** SRFEN is a read-only bit that gets set when the Master enables its Write FIFO (WFEN (MSI1FIFOCS<15> = 1). The bit will be cleared only when the Master clears the WFEN bit.

**2:** SRFEN bit is set when the Master sets RFEN (MSIFIFOCS<7>) = 1.

**3:** Overflow bit is set when the buffer is full and the Master sends one more data without the Slave reading the SRMWFDATA register.

**4:** SRFFULL bit is set when the Slave read buffer is full. It can be cleared when the Slave reads the buffer (using the SRMWFDATA register).

**Register 2-14: SWMRFDATA: Slave Write (Master Read) FIFO Data Register**

| W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |
|---|---|---|---|---|---|---|---|
| SWMRFDATA<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 | W-0 |
|---|---|---|---|---|---|---|---|
| SWMRFDATA<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-0 **SWMRFDATA<15:0>:** Write FIFO Data Out Register bits

**Register 2-15: SRMWFDATA: Slave Read (Master Write) FIFO Data Register**

| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| SRMWFDATA<15:8> | | | | | | | |
| bit 15 | | | | | | | bit 8 |

| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |
|---|---|---|---|---|---|---|---|
| SRMWFDATA<7:0> | | | | | | | |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 15-0 **SRMWFDATA<15:0>:** Write FIFO Data Out Register bits

## 3.0 OVERVIEW

The Master Slave Interface (MSI) macro is the data gateway between the Master (main) processor and a Slave processor, and is primarily intended as a means to control the Slave processor and move data between the processors. The Master and Slave processors are expected to operate at significantly different clock speeds, so the MSI module also includes synchronization for data and signals that cross between clock domains. The macro consists of 16 independent, unidirectional mailbox-style data pipes. The data direction logic assignments are fuse-selectable.

Depending upon the mailbox direction, the data write and read processors could be either Master and Slave, or Slave and Master, respectively.

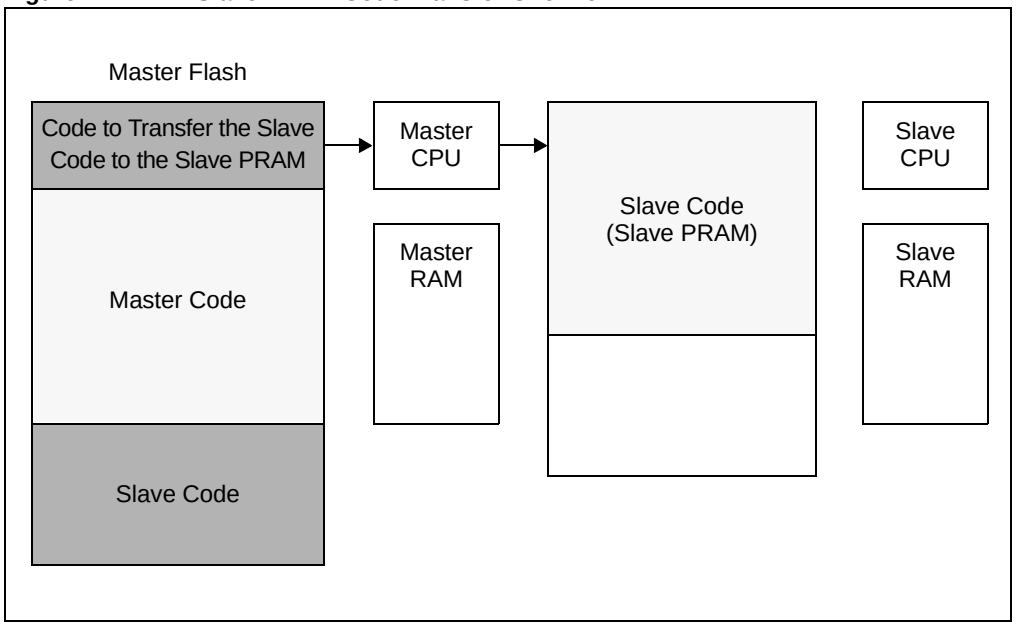
## 4.0 SLAVE PROCESSOR CONTROL

The MSI contains three control bits related to Slave processor control within the MSI1CON register.

### 4.1 Slave Enable (SLVEN) Control

When the device is powered for the first time, the Slave code is residing in the Master Flash. The user code in the Master will transfer the data from the Master to the Slave after the first power-up.

**Figure 4-1: Slave PRAM Code Transfer Overview**



The Slave has to be held in Reset when this transfer is happening; this is achieved by the MSI module. The SLVEN (MSI1CON<15>) control bit provides a means for the Master processor to enable or disable the Slave processor.

The Slave is disabled when SLVEN = 0. In this state:

- The Slave is held in the Reset state
- The Master has access to the Slave PRAM (to load it out of a device Reset)
- The Slave Reset Status bit, SLVRST (MSI1STAT<15>) = 1 (Master register)

The Slave is enabled when SLVEN = 1. In this state:

- The Slave Reset is released and it will start to execute code in whatever mode it is configured to operate
- The Master processor will no longer have access to the Slave PRAM (in Dual Panel mode, the Master will have access to the inactive PRAM)
- The Slave Reset Status bit, SLVRST (MSI1STAT<15>) = 0

The SLVRST bit status indicates when the Slave is in Reset. The associated interrupt only occurs when the Slave enters the Reset state after previously having not been in Reset. That is, no interrupt can be generated until the Slave is first enabled.

| **Note:** | The SLVEN bit may only be modified after satisfying the hardware write interlock, as described in Example 4-1. |
|---|---|

The SLVEN bit is protected from unexpected writes through a software unlocking sequence that is based on the MSI1KEY register. Given the critical nature of the MSI control interface, the MSI macro unlock mechanism is independent from that of the Flash controller for added robustness.

Completing a predefined data write sequence to the MSI1KEY register will open a window. The SLVEN bit should be written on the first instruction that follows the unlock sequence. No other bits within the MSI1CON register are affected by the interlock. The MSI1KEY register is not a physical register. A read of the MSI1KEY register will read all '0's.

When the SLVEN bit lock is enabled (i.e., the bits are locked and cannot be modified), the instruction sequence shown in Example 4-1 must be executed to open the lock. The unlock sequence is a prerequisite to both setting and clearing the target control bit.

**Example 4-1: MSI Enable Operation**

```
//Unlock Key to allow MSI Enable control

MOV.b    #0x55, W0
MOV.b    WREG, MSI1KEY
MOV.b    #0xAA, W0
MOV.b    WREG, MSI1KEY
// Enable MSI

BSET     MSI1CON, SLVEN
```

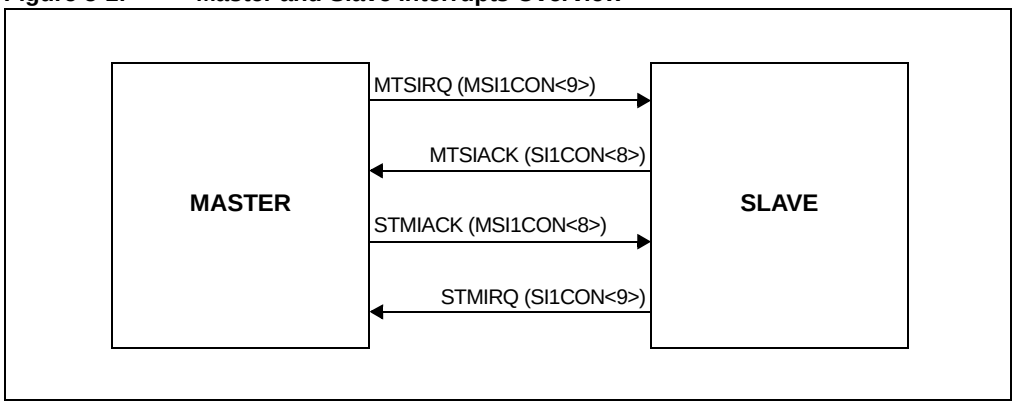
**Example 4-2: MSI Enable Operation in C Code**

```
#include <libpic30.h>
    _start_slave();
```

## 5.0 INTER-PROCESSOR INTERRUPT REQUEST AND ACKNOWLEDGE

The Master and Slave processors may interrupt each other directly. The Master may issue an interrupt request to the Slave by asserting the MTSIRQ (MSI1CON<9>) control bit. Similarly, the Slave may issue an interrupt request to the Master by asserting the STMIRQ (SI1CON<9>) control bit. The interrupts are Acknowledged through the use of the Interrupt Acknowledge bits, STMIACK (MSI1CON<8>) for the Master to Slave interrupt request, and MTSIACK (SI1CON<8>) for the Slave to Master interrupt request (Figure 5-1).

**Figure 5-1: Master and Slave Interrupts Overview**




## 6.0 TRANSFER MODE

Based on the method of data transfer between the Master and the Slave, the transfer can be classified into two major types:

1. Mailbox-based transfer.
2. FIFO-based transfer.

## 7.0 MAILBOX TRANSFER MODE

The mailbox consists of 16 independent, unidirectional Data registers. Up to eight of these registers may be selected to operate with independent data flow control logic that supports a hardware Ready/Acknowledge protocol, creating a mailbox-style data pipe. The eight protocol-based communication status bits are indicated in the register, MSI1MBXS. There are eight hardware protocols, which are named A through H.

| **Note:** | Both the data direction and data flow control logic assignments are selected by the FMBXM<15:0> Configuration bits and are assigned before the code execution. |
|---|---|

Because the direction of each Data register is programmable, referencing the Master or Slave is not meaningful when discussing data transfer protocol. The terms, “transmitter” and “receiver”, are therefore, subsequently used to represent the data write and read processors, respectively. Depending upon the Data register direction, the data write and read processors could be either Master and Slave or Slave and Master, respectively.

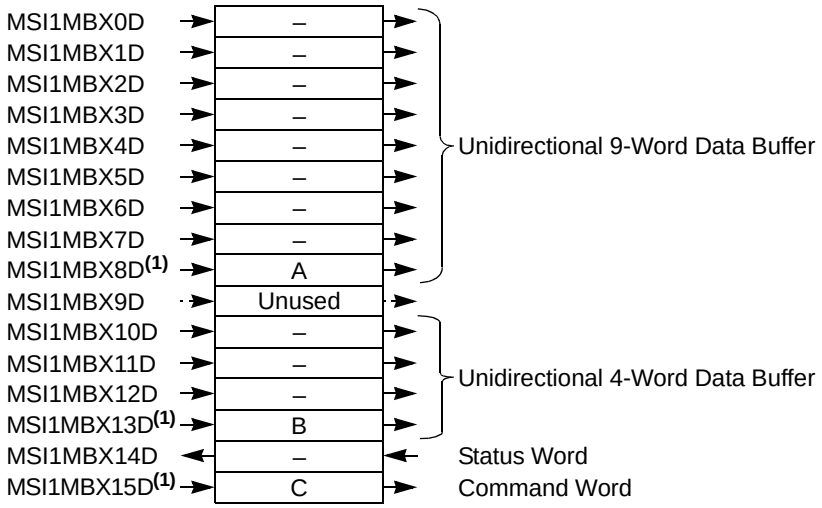
## 7.1 Mailbox Data Pipes

Access to the Data registers within the mailbox is controlled using the data flow control protocol. Consequently, access to a mailbox-based data pipe is mutually exclusive (i.e., it cannot be accessed simultaneously by both processors). Each processor must complete its access prior to handing access control to the other. For example, if the Master has configured the MSI1MBX0D as a transmitter, the Slave will have to wait for the interrupt (command) so that it can receive from the SI1MBX0D register.

Furthermore, for mailboxes that consist of more than one Data register, the direction of all the Data registers within the mailbox does not have to be the same. The data flow control protocol is used to transfer access control between the Master and Slave, but only the Data register assigned to the protocol hardware must comply with the required data direction rules.

Figure 7-1 shows an arrangement that supports 2 unidirectional buffers, a command word and a status word. A data flow control logic block is assigned to the last word accessed of each of the buffers (note that the buffer length is arbitrary within the limits of the number of mailboxes supported). A data flow control logic block is also assigned to the command word. However, access to the status word is controlled through software, so no data flow control logic block is required. The MSI1MBX9D mailbox is unused. For example, the interrupt for the specific protocol will not be generated until the protocol assigned register is read or written. As shown in Figure 7-1, after all the registers are written (MSI1MBX0D to MSIx7D), the Slave will not get a Protocol A interrupt until the Master writes the MSI1MBX8D register (Protocol A register).

**Figure 7-1: Mailbox Organization Example**




**Note 1:** MSI1MBX8D should be assigned to Protocol A by fuse, FMBXHS1. MSI1MBX13D and MSI1MBX15D should be assigned to their respective protocol using the FMBXHSx Configuration fuse.

**Note:** Mailboxes should not be used without handshaking. Care should be taken that the receiving core should not be read while the transmitting core is transmitting.

## 7.2 Mailbox Data Registers

Each of the 16 MSI Mailbox Data registers, MSI1MBXnD/SI1MBXnD (where $0 \le n \le 15$) is identical, other than their data direction. The MSI macro contains eight data flow control protocol hardware blocks, each of which may be assigned to any Data register to form a mailbox. The status of these mailboxes is updated in the DTRDYx bits (MSI1MBXS<7:0>), where x can be A-H, representing the eight data flow protocols.

## 7.3 Mailbox Register Accessibility

All MSI1MBXnD Mailbox Data registers are unidirectional, such that the register contents are never read/write from both the Master and Slave ports. Each MSI1MBXnD register is either read/write from the Master (as transmitter) and read-only from the Slave (as receiver), or read/write from the Slave (as transmitter) and read-only from the Master (as receiver), depending upon the selected channel data direction. This is achieved using the MBXM<15:0> Configuration bits FMBXM<15:0>:

**FMBXM MBXMn:** Mailbox Data Register n Channel Direction Fuse bits (n = 0 to 15)

1 = Mailbox Register #n is configured for Master data read (Slave to Master data transfer – Slave transmitter)

0 = Mailbox Register #n is configured for Master data write (Master to Slave data transfer – Master transmitter)

### 7.3.1 DATA HANDSHAKE

| | |
|---|---|
| **Note:** | The following text assumes that the referenced Data register (MSI1MBXnD/SI1MBXnD) is of the correct data direction to support the operation being described. |

An automated data flow control mechanism is supported to control the flow of data through the mailboxes. Each of the eight data flow handshake protocol hardware blocks controls 2 Data Ready Status bits (DTRDYx, where x is A, B, C, D, E, F, G or H), located in the MSI1MBXS and SI1MBXS registers. One flag is for the data transmitter and is located in the MSI1MBXS/SI1MBXS register on the transmit side of the interface. The other flag for the data receiver is located in the MSI1MBXS/SI1MBXS register on the receive side of the interface.

The data transmitter is always assumed to be the transfer initiator, so a hardware data request from the data receiver is not required. Should the application require a data request to initiate a transfer, it must be handled through software. The receiving processor software will have to indicate to the transmitting processor that data is required. This may be achieved, either through an interrupt, or through a mailbox-based software command protocol.

#### 7.3.1.1 Enabling the Handshake Protocol Hardware Blocks

Each of the handshake protocol hardware blocks has a fuse enable associated with it. The fuse must be programmed in order to enable the corresponding handshake protocol hardware block. The FMBXHS1<3:0> Configuration bits correspond to Handshake Protocol Hardware Block A, the FMBXHS1<7:4> Configuration bits correspond to Handshake Protocol Hardware Block B, etc. (consult the device data sheet for fuse details).

**FMBXHS1 MBXHSA<3:0>:** Mailbox Handshake Protocol Block D Register Assignment bits

1111 = MSI1MBXD15 is assigned to Mailbox Handshake Protocol Block A

**. . .**

0001 = MSI1MBXD1 is assigned to Mailbox Handshake Protocol Block A

0000 = MSI1MBXD0 is assigned to Mailbox Handshake Protocol Block A

**FMBXHS1 MBXHSB<3:0>:** Mailbox Handshake Protocol Block B Register Assignment bits

1111 = MSI1MBXD15 is assigned to Mailbox Handshake Protocol Block B

**. . .**

0001 = MSI1MBXD1 is assigned to Mailbox Handshake Protocol Block B

0000 = MSI1MBXD0 is assigned to Mailbox Handshake Protocol Block B

**FMBXHSx MBXHSn<3:0>**, where n = A to H
(check the device data sheet for the correct equivalent configuration settings).

### 7.3.2 ASSIGNING THE HANDSHAKE PROTOCOL HARDWARE BLOCKS

Each of the eight protocol blocks is assigned to a specific MSI Mailbox Data register by eight, 4-bit fields within the FMBXHSx register (MBXHSn<3:0>):

1. The selected MSI Mailbox register is referred to as the Mailbox Protocol Data register.
2. Unassigned Mailbox registers are referred to as Mailbox Data registers.

A Protocol Data register may be a single mailbox, or one Mailbox register within a set of Mailbox registers, defined as a buffer through software. When mailboxes are defined as buffers, the last buffer access must be to the Protocol Data register. Similarly, when the receiving processor sees that data is ready and accesses the mailbox, the last buffer access must also be to the Protocol Data register. The user software (Master, as well Slave) has to decide how many mailboxes should be associated with each Protocol Data register. If MSI1MBX0D and MSI1MBX1D are selected as Mailbox Data registers associated with Protocol A (with MSI1MBX4D Mailbox Protocol Data Register A), the transmitter should make sure that the Mailbox Protocol Data register (MSI1MBX4D) is written last, after the operation on the Mailbox Data registers is done (MSI1MBX0D, MSI1MBX1D).

Similarly, when the receiver is receiving the data, the receiver should make sure that the Protocol Data register (MSI1MBX4D) is read last, after the operation on the Mailbox Data registers is complete (MSI1MBX0D, MSI1MBX1D).

## 7.4 Mailbox Data Transfer Using Interrupts

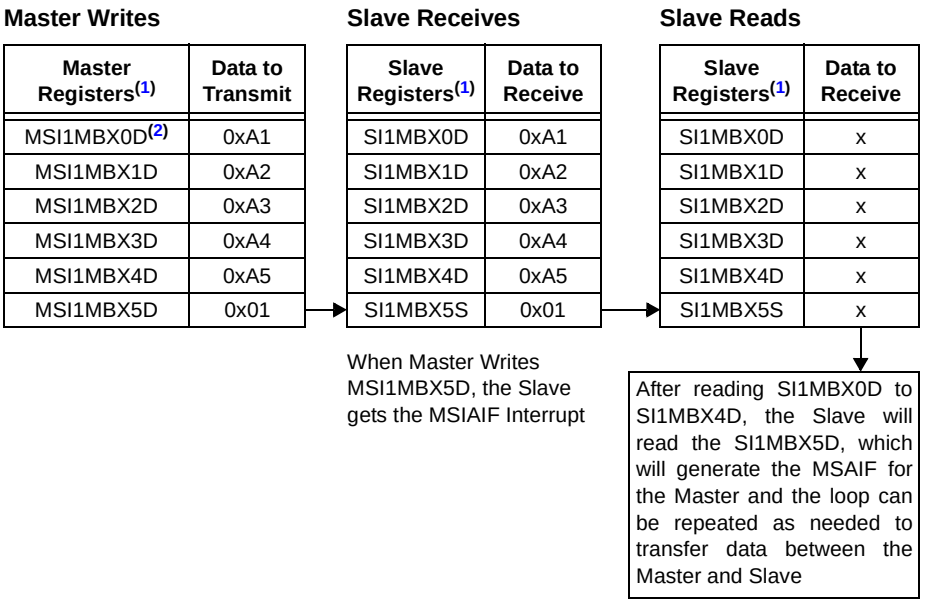
When neither processor is accessing the mailbox, the data flow control hardware is in the Idle state (DTRDYx (MSI1MBXS<7:0> = 0). The transmitting processor may now access the mailbox to start the data transfer data flow control.

The data flow control operates as described below, where the MSI1MBX0D-MSI1MBX4D registers are assigned to be the Mailbox Data registers and the MSI1MBX5D is the Mailbox Protocol Data register (A):

1. Transmitting processor:
   a) Write all but the last data word.
   b) DIN → MSI1MBX5D (last data write) 1 → DTRDYA Send a Ready to Read interrupt to receiver (automatic).
2. Receiving processor:
   Receive Ready to Read Interrupt 1
   a) Read/transmit all but the last data word (if a buffer), MSI1MBX0D-MSI1MBX4D.
   b) MSI1MBX5D → DOUT (last data read) 0 → DTRDYA (Automatic 2) Send a Ready to Write interrupt to transmitter (automatic).
3. Transmitting processor:
   Loopback to 1

In Figure 7-2, the MSI1MBX0D to MSI1MBX4D registers are configured for Master to Slave transmit using the MBXMn bits in the FMBXM Configuration register and MSI1MBX5D is configured for Protocol A using FMBXHS1<3:0> = 0101. Example 7-1 and Example 7-2 show code to allow a mailbox transfer between the Master and Slave.

**Figure 7-2: Mailbox Data Transfer Flow**




**Note 1:** Mailbox Data registers can be transmitters or receivers as assigned by the FMBXM Configuration bits. This example shows all the Mailbox Data registers as Transmit registers.

**2:** Mailbox Protocol A Data register (as assigned by FMBXHS1<3:0> = 0101).

After Step 3a, the data flow control is complete and the transmitting processor may exit or proceed to send more data (i.e., loopback to Step 1).

As noted above, a write to the MSI1MBX5D register by the Transmitter register will result in a receiver data flow control protocol interrupt (Ready to Read), from the corresponding mailbox of the receiver, by setting the MSIAIF interrupt and DTRDYA = 1.

Similarly, when the receiver reads the MSI1MBX5D register (after reading the Mailbox Data registers), the MSIAIF, as well the DTRDYA bit of the transmitter, will get set.

**Note:** Interrupts associated with unused protocol hardware blocks should be disabled by the user in the interrupt controller.

**Example 7-1: Mailbox Transfer Using Protocol A Interrupt (To and From Data Transfer Between Master and Slave)**

```
////////////////////////////////////////////////////////////////////////////////////////////
MASTER PROJECT
////////////////////////////////////////////////////////////////////////////////////////////
#include "p33CH128RA508.h"
#pragma  config MBXM0 = M2S    //(Master to Slave data transfer)
#pragma  config MBXM1 = M2S    //(Master to Slave data transfer)
#pragma  config MBXM2 = M2S    //(Master to Slave data transfer)
#pragma  config MBXM3 = M2S    //(Master to Slave data transfer)
#pragma  config MBXM4 = M2S    //(Master to Slave data transfer)
#pragma  config MBXM5 = M2S    //(Master to Slave data transfer) Protocol A assigned to mailbox 5
#pragma  config MBXM6 = S2M    //(Slave to Master data transfer)
#pragma  config MBXM7 = S2M    //(Slave to Master data transfer)
#pragma  config MBXM8 = S2M    //(Slave to Master data transfer)
#pragma  config MBXM9 = S2M    //(Slave to Master data transfer)
#pragma  config MBXM10 = S2M   //(Slave to Master data transfer)
#pragma  config MBXM11 = S2M   //(Slave to Master data transfer)
#pragma  config MBXM12 = S2M   //(Slave to Master data transfer)
#pragma  config MBXM13 = S2M   //(Slave to Master data transfer)
#pragma  config MBXM14 = S2M   //(Slave to Master data transfer)
#pragma  config MBXM15 = S2M   //(Slave to Master data transfer)

// FMBXHS1
#pragma  config MBXHSA = MBX5  //(MSIxMBXD5 assigned to mailbox handshake protocol block A)
#pragma  config MBXHSB = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block B)
#pragma  config MBXHSC = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block C)
#pragma  config MBXHSD = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block D)
// FMBXHS2
#pragma  config MBXHSE = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block E)
#pragma  config MBXHSF = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block F)
#pragma  config MBXHSG = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block G)
#pragma  config MBXHSH = MBX15 //(MSIxMBXD15 assigned to mailbox handshake protocol block H)
// FMBXHSEN
#pragma  config HSAEN = ON     //(Mailbox data flow control handshake protocol block enabled.)
#pragma  config HSBEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSCEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSDEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSEEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSFEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSGEN = OFF    //(Mailbox data flow control handshake protocol block disabled.)
#pragma  config HSHEN = OFF
```

**Example 7-1: Mailbox Transfer Using Protocol A Interrupt (To and From Data Transfer Between Master and Slave) (Continued)**

```
unsigned int temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9,temp10,Flag;
int     main()
{
        IEC8bits.MSIAIE=1;     //enable interrupt for protocol A
        Flag=0;
        MSI1MBX0D=0xA0;
        MSI1MBX1D=0xA1;
        MSI1MBX2D=0xA2;
        MSI1MBX3D=0xA3;
        MSI1MBX4D=0xA4;
        Switch();              //waiting to start the to and fro transfer (writing to MSI1MBX5D will
                               //initiate the transfer to the slave and set MSIAIF of slave)

        MSI1MBX5D=0xA5;        //this will initiate transfer
while(1)
{
        while(Flag==0);        //Wait till the slave responds by reading MSI1MBX5D(which will generate
                               //the master MSIAIF interrupt)

           Flag=0;             //this flag is set in the MSAIF interrupt vector
        MSI1MBX0D=0xA0;
        MSI1MBX1D=0xA1;
        MSI1MBX2D=0xA2;
        MSI1MBX3D=0xA3;
        MSI1MBX4D=0xA4;
        MSI1MBX5D=0xA5;        //writing MSI1MBX5D will initiate transfer setting MSIAIF of the slave
}
/////////////////interrupt vector for protocol A/////////////////////////////////////
void __attribute__ ((interrupt, no_auto_psv)) _MSIAInterrupt(void)
{
        IFS8bits.MSIAIF=0;
        //Read the data from slave
        temp1=MSIMBX6D;
        temp2=MSIMBX7D;
        temp3=MSIMBX8D;
        temp4=MSIMBX9D;
        temp5=MSIMBX10D;
        temp6=MSIMBX11D;
        temp7=MSIMBX12D;
        temp8=MSIMBX13D;
        temp9=MSIMBX14D;
        temp10=MSIMBX15D;
        // set the flag for the next round of data transfer
        IFS8bits.MSIAIF=0;
        Flag=1;
}
```

**Example 7-2: Mailbox Transfer Using Protocol A Interrupt (To and From Data Transfer Between Slave and Master)**

```
////////////////////////////////////////////////////////////////////////////////////////////
SLAVE PROJECT (Slave side of the project)
////////////////////////////////////////////////////////////////////////////////////////////

unsigned int temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9,temp10,Flag;
int     main(void)
{

               IEC8bits.MSIAIE=1;
               Flag=0;
    while (1)
{
        while(Flag==0);    //Wait till master initiates the transfer by writing data in MSI1MBX5D
        Flag=0;            //This flag is set in the MSIAIF interrupt
            SI1MBX6D=0x03;  //load all the data that need to be transfered to Master SI1MBX6D to SI1MBX15D
            SI1MBX7D=0x04;
            SI1MBX8D=0x05;
            SI1MBX9D=0x06;
            SI1MBX10D=0x07;
            SI1MBX11D=0x08;
            SI1MBX12D=0x08;
            SI1MBX13D=0x0A;
            SI1MBX14D=0x0B;
            SI1MBX15D=0x0C;
        temp5=SI1MBX5D;    // Reading the SI1MBX5D will generate MSIAIF interrupt for Master


    }
}

// MSAIF interrupt//////////////////////////////////////////////////////////////////////////
void __attribute__ ((interrupt, no_auto_psv)) _MSIAInterrupt(void)
{
            temp0=MSI1MBX0D;
            temp1=MSI1MBX1D;
            temp2=MSI1MBX2D;
            temp3=MSI1MBX3D;
            temp4=MSI1MBX4D;
            // need to wait to read MSIMBX5D unless Master needs to be interrupted

            IFS8bits.MSIAIF=0;
            Flag=1;

}
```

## 7.5 Mailbox Data Transfer Using Software Polling

Although using interrupts is intended to be the primary method of managing the mailbox data flow control protocol, it is possible to also poll the status bits with software. In applications where the data sent through a mailbox is to be used within a periodic control process, software polling of the mailbox data flow control status flag could be the preferred approach. The transmitting and receiving processor polling software should test its respective DTRDYx flag in order to determine the state of the data flow control.

Transmitting processor:

- DTRDYx = 1: Not ready to send data (mailbox not yet read)
- DTRDYx = 0: Ready to send data (mailbox empty)

Receiving processor:

- DTRDYx = 1: Data available to read (but not yet read)
- DTRDYx = 0: Ready to receive data (mailbox empty or data stale)

## 7.6 Mailbox Data Register, Handshake Status Bits and Master/Slave Resets

The MSI1MBXnD registers are not subject to any device Reset other than POR/BOR, so data is therefore preserved should the receiver software be able use it. The assumption is that if a receiver read is already underway (i.e., interrupt triggered or DTRDYx polled and found to be set), it is preferable to return valid (if old) data instead of a Reset value.

However, all DTRDYx flow control bits (both Master and Slave) are subject to Master Resets. This is necessary to initialize the data flow protocol blocks upon Reset exit.

When the Master experiences a Reset, both the Master and Slave views of the Data Ready Status flags, DTRDYx (MSI1MBXS<7:0>), will be reset.

When MSRE (FSLV1DEVOPT<15>) = 0, the Slave Reset is decoupled from the Master, MSRE fuse = 0 (refer to **Section 10.0 "Master/Slave Reset Interaction"**), such that the Slave will continue to run in the event of a Master Reset should a Master (transmitter) write to the MSI1MBXnD register be immediately followed by a Master Reset, the Slave (receiver) side interrupt request will not occur.

When the Slave experiences a Reset, neither the Master nor Slave views of the Data Ready Status bits (DTRDYx) will be reset. Should a Slave (transmitter) write to the MSI1MBXnD register be immediately followed by a Slave Reset, the Master (receiver) side interrupt request will still occur as normal.

Resetting both the Master and Slave Data Ready Status (DTRDYx) bits with a Master Reset is also required to avoid a possible data collision condition. In the case of the Slave DTRDYx (MSI1MBXS<7:0>) flag, when the Master and Slave Resets are not coupled (Configuration fuse, MSRE = 0) and a Slave Reset will not disable the Slave (SSRE fuse = 0), a possible data collision condition could arise if the Slave DTRDYx flag were to be reset by a Slave Reset. If the Slave DTRDYx flag were to be reset on a Slave Reset, it could be possible for the Master to reset, resetting the Master view of the DTRDYx flag, but not that of the Slave. This would give the (still running) Slave the opportunity to service the Slave DTRDYx flag and read the corresponding mailbox, possibly while the Master is writing to it (assuming that it is empty because Master DTRDYx = 0.

## 7.7 Use of Mailboxes for Temporary Storage

A read from the MSI1MBXnD register by the receiver will only generate a data flow control protocol interrupt (Ready to Write) if the receiver DTRDYx = 1. If the receiver DTRDYx = 0 (which will be the case after the initial read of new data from the mailbox), a subsequent read of the mailbox by the receiver will have no effect (other than to return the data contents of the target mailbox). This allows the mailbox to be used by the receiver for temporary storage of the last data value that moved through it.

However, after data is read from a mailbox, its contents must be considered to be stale and subject to change (at any time) by the transmitter. Consequently, in order to manage mailbox temporary storage successfully, it is assumed that there is a software data transfer protocol in place, such that the data receiver can prevent the data transmitter from arbitrarily overwriting the contents of the mailbox with new data. For example, if the receiver had to request data from the transmitter (via another mailbox or an interrupt), the transmitter will not overwrite the mailbox.

## 7.8 Transaction Data Size

As is the case for any SFR, the MSI1MBXnD registers are both byte or word-assessable. In order to support both byte and word-sized data transactions when using data buffers, either a Most Significant Byte (MSB) or word write of the Transmitter Protocol register will set the corresponding DTRDYx flag. Similarly, either an MSB or word read of the Receiver Protocol Data register (on the other side of the MSI) will set the corresponding DTRDYx flag.

## 7.9 Mailbox Data Transfer Using the DMA Controller

The Mailbox Data registers may be accessed on the Master or Slave side of the MSI using DMA if available on the device. The mailbox data flow control protocol will generate interrupts that are compatible with DMA operation, allowing data within individual Mailbox registers to be transferred without CPU intervention.

## 7.10 DMA Data Transfer Sequence

For the first DMA data value (or block) to be transferred, the assigned transmitter DMA channel may be triggered by software or by manually writing the first data value (or block of data) in software. When the DMA writes to a Mailbox Protocol Data register (last write in the case of a block transfer), the corresponding DTRDYx flag will be set. Setting the transmitter DTRDYx flag will generate a Ready to Read interrupt on the receiver side of the interface.

The receiver Ready to Read interrupt (initiated after the Mailbox Protocol Data register is written by the transmitter, setting DTRDYx = 1) will trigger the corresponding receiver DMA channel and cause it to read the target mailbox (or mailboxes in the case of a block transfer). In doing so, it will clear the corresponding DTRDYx flag. Clearing the receiver DTRDYx flag will generate a transmitter Ready to Write interrupt on the transmitter side of the interface. This will trigger the assigned transmitter DMA channel to write the next data value (or block of data) and auto-set the DTRDYx flag, starting the sequence again.

# 8.0 FIFO TRANSFER MODE

## 8.1 FIFO Data Channels

The MSI contains a 2-channel FIFO; the FIFOs are used to coordinate data queues between the Master and Slave processors. Provided the FIFO does not become empty (or encounters an error condition), the Master and Slave may access it concurrently. A FIFO may therefore, offer a better throughput than a mailbox-based data pipe, which must be loaded by one processor before being read by the other.

Each FIFO channel data flow is unidirectional to simplify the design and operation; one channel is a dedicated read data channel, the other a dedicated write data channel. In the following sections, the data transmitter is the processor that writes data into a FIFO. Conversely, the data receiver is the processor that reads data from a FIFO.

### 8.1.1 FIFO ENABLE

The FIFO will be disabled when the corresponding Write FIFO Enable bit is cleared (WFEN (MSI1FIFOCS<15>) for the Master Write FIFO and RFEN (MSI1FIFOCS<7>) for the Read FIFO). The FIFO enable control bits are cleared during a device Reset.

The Master is ultimately responsible for initializing and enabling the Slave, and associated mailboxes using the Configuration bits, so the Master is also responsible for engaging (or not) the MSI FIFOs, irrespective of data flow direction.

Under normal operating conditions, the FIFOs will remain enabled. However, in the event of a FIFO error, or if the Slave processor has reset (or has stopped responding and needs to be reset), the WFEN (MSI1FIFOCS<15>) and RFEN (MSI1FIFOCS<7>) control bits can be used to flush and re-initialize the FIFOs as necessary.

When disabled, the FIFO contents are wiped (reset to logic '0') and the Address Pointers are initialized to the FIFO empty state, where both Address Pointers are set equal to each other (in this case, all '0's). The Write FIFO Empty Status bit (MSI1FIFOCS<8>) is also set for the Write FIFO, and the RFEMPTY (MSI1FIFOCS<0>) is set for the Read FIFO.

After the FIFO is enabled, the Empty Status bit will remain set until such time that the first data value is written into the FIFO.

The FIFO Empty Status flags are set to '1' in the event of a Master Reset or whenever the FIFOs are disabled. However, FIFO empty interrupts are disabled whenever the FIFO is disabled. A FIFO empty interrupt will therefore, never be pending upon Reset exit or when a FIFO is disabled. When disabled, the FIFO Underflow and Overflow flags are cleared.

**Note:** FIFO underflow is detected on the FIFO read side of the interface. This status must be synchronized to the write side clock before it can be observed by the FIFO write side of the interface. Consequently, on the write side of the interface, the underflow status will be delayed from when the FIFO is actually detected as underflowed.

A further implication of this delay is that, when the user disables the Write FIFO, they must wait for the underflow status to propagate before re-enabling the FIFO. This is because the FIFOs can only be disabled from the Master side of the interface. Therefore, the WFEN signal must propagate to the Slave side to clear the WFUF flag and then the WFUF state must propagate back to the Master side before it is seen to be cleared.

## 8.2 FIFO Data Register Accessibility

Data to be passed from the Master to the Slave processor is written by the Master processor into the Master Write FIFO Data register (MWSRFDATA<15:0>). The Slave can then read the data from the Slave Read FIFO Data register (SRMWFDATA<15:0>).

Data to be passed from the Slave to the Master processor is written by the Slave processor into the Slave Write FIFO Data register, SWMRFDATA<15:0>. The Master can then read the data from the Master Read FIFO Data register (MRSWFDATA). Because each Data register access modifies the data channel FIFO Address Pointers, data is to be written and read as a single entity (i.e., a word or byte).

The Write FIFO Data registers (Master MWSRFDATA<15:0> and Slave SWMRFDATA<15:0>) are write-only registers. Reading these registers will return all '0's and not affect the FIFO Address Pointers. The Read FIFO Data registers (Master MRSWFDATA<15:0> and Slave SRMWFDATA<15:0>) are read-only. Writes to these registers will have no effect.

| | |
|---|---|
| **Note:** | Read-Modify-Write operations should not be executed on the MWSRFDATA/ MRSWFDATA or SRMWFDATA/SWMRFDATA registers. |

### 8.2.1 FIFO DATA SIZE

As is the case for any SFR, the FIFO Data registers are both byte or word-assessable. In order to support both byte and word-sized data sizes when writing into the FIFOs, either an MSB or word write of the Write FIFO Data register will modify the data channel FIFO Write Address Pointer. Similarly, either an MSB or word read of the Read FIFO Data register will modify the data channel FIFO Read Address Pointer.

When using FIFOs for byte-sized data transfers, the FIFO Data register Least Significant Byte (LSB) must be accessed prior to the MSB.

### 8.2.2 FIFO SIZE AND ADDRESSING

The FIFOs operate as circular buffers, each using a Write and Read Address Pointer to determine the next write and read address, respectively. The Address Pointers are both modified after their respective operation completes. Consequently, after each write operation, the Write Address Pointer will point to the next free location within the FIFO, and prior to each read operation, the Read Address Pointer will point to the next data location to read.

| | |
|---|---|
| **Note:** | The FIFO for the dsPIC33CH families is 32 words deep. For the FIFO size for the specific device, please refer to the device data sheet. The FIFO Address Pointers are not externally controllable or observable by the user (other than being able to reset them (FIFO flush) by disabling the FIFO). |

When the Read and Write Address Pointer are equal in value, the circular buffer is deemed to be empty, and the FIFO Empty Status bit is set (WFEMPTY (MSI1FIFOCS<8>) for the Write FIFO and RFEMPTY (MSI1FIFOCS<0>) for the Read FIFO). This will also generate a data request interrupt (MSIWFEIF (IFS8<11>)) to the data transmitter processor on the write side of the interface.

### 8.2.3 WRITING TO A FIFO DATA CHANNEL

Data is written to the next free location within a FIFO when the data transmitter writes to the Write FIFO Data register (MWSRFDATA/SWMRFDATA for the Write FIFO and MRSWFDATA/ SRMWFDATA for the Read FIFO).

When the addressed FIFO location is loaded, the Write Address Pointer is adjusted to point to the next free location within the circular buffer. If there are no remaining free locations, the Write FIFO Full Status bit (WFFULL (MSI1FIFOCS<9>) is set for the Write FIFO and RFFULL (MSI1FIFOCS<1>) for the Read FIFO).

| | |
|---|---|
| **Note:** | The application must test the status of the FIFO Empty Status flag prior to reading data from the FIFO. Reading data from an empty FIFO Data register will result in a data underflow. |

### 8.2.4 FIFO INTERRUPTS

The FIFO Empty Status bits are used to generate interrupts to both the Master and Slave processors. These interrupts are intended to be used as part of the data transfer protocol. However, if not required by the application, they may be disabled within the interrupt controller (MSIxIE bits). Table 8-1 shows the interrupts associated with the FIFO.

**Table 8-1: MSI FIFO Interrupts**

| Core | Interrupt | Comment |
|---|---|---|
| Master | MSIFLTIF | Master Read or Write FIFO Fault Interrupt for Overflow or Underflow |
| Slave | MSIFLTIF | Slave Read or Write FIFO Fault Interrupt for Overflow or Underflow |
| Master | MSIWFEIF | Master Write FIFO Empty Interrupt |
| Slave | MSIWFEIF | Slave Write FIFO Empty Interrupt |
| Master | MSIDTIF | Master FIFO Data Ready Interrupt |
| Slave | MSIDTIF | Slave FIFO Data Ready Interrupt |

### 8.2.5 FIFO EMPTY INTERRUPT (MSIWFEIF)

When a FIFO is deemed to be empty, the FIFO Empty Status flag is set and a FIFO empty interrupt is generated for the data transmitter processor. The interrupt is generated on the logic '0' to logic '1' transition of the FIFO Empty Status flag (WFEMPTY (MSI1FIFOCS<8>) for the Write FIFO and RFEMPTY (MSI1FIFOCS<0>) for the Read FIFO). Writing data to the FIFO will clear the FIFO Empty Status flag and send a data valid interrupt to the receiver.

The FIFO Empty Status flags are set to logic '1' in the event of a Master Reset or whenever the FIFOs are disabled. However, FIFO empty interrupts are disabled whenever the FIFO is disabled. Therefore, a FIFO empty interrupt will never be pending upon Reset exit or when a FIFO is disabled.

### 8.2.6 FIFO DATA VALID (READY) INTERRUPT (MSIDTIF)

When data is written into a previously empty FIFO, the FIFO Empty Status flag is cleared and a FIFO data valid interrupt is generated for the data receiver processor. The interrupt is generated based on various thresholds of data availability in the FIFO.

There are 4 ways the interrupt logic can function. The logic is determined by the RFITSEL<1:0> bits (MSI1CON<11:10> for Master and SI1CON<11:10> for Slave).

The interrupt can occur after a Master write to FIFO or a Slave write to FIFO and data is ready in the FIFO with the following conditions:

1. Interrupt when 1st FIFO data is ready.
2. Interrupt when FIFO is 50% full.
3. Interrupt when FIFO is 75% full.
4. Interrupt when FIFO is 100% full.

### 8.2.7 FIFO OVERFLOW AND UNDERFLOW STATUS AND INTERRUPTS

In the event that a data transmitter writes data to the FIFO after the FIFO Full Status bit is set (WFFULL (MSI1FIFOCS<9>) for the Write FIFO and RFFULL (MSI1FIFOCS<1>) for the Read FIFO), the FIFO occupancy logic will detect an overflow condition and set the FIFO Overflow flag (WFOF (MSI1FIFOCS<11>) for the Write FIFO and RFOF (MSI1FIFOCS<3>) for the Read FIFO). Note that the data write will be ignored, and the FIFO Write Pointer will not be modified, preserving the contents of the FIFO.

Similarly, in the event that a data receiver attempts to read data from the FIFO after the FIFO Empty Status bit is set, the FIFO occupancy logic will detect an underflow condition and set the FIFO Underflow flag (WFUF (MSI1FIFOCS<10>) for the Write FIFO and RFUF (MSI1FIFOCS<2>) for the Read FIFO). The FIFO Read Pointer will not be adjusted prior to the read (as would be typical), resulting in a re-read of the most recently read FIFO address.

The FIFOs may be used in a variety of ways, some of which are described below. Data may be requested or pushed to a processor. The data Acknowledge may be implied directly (using the FIFO Empty Status bit state or processor interrupt).

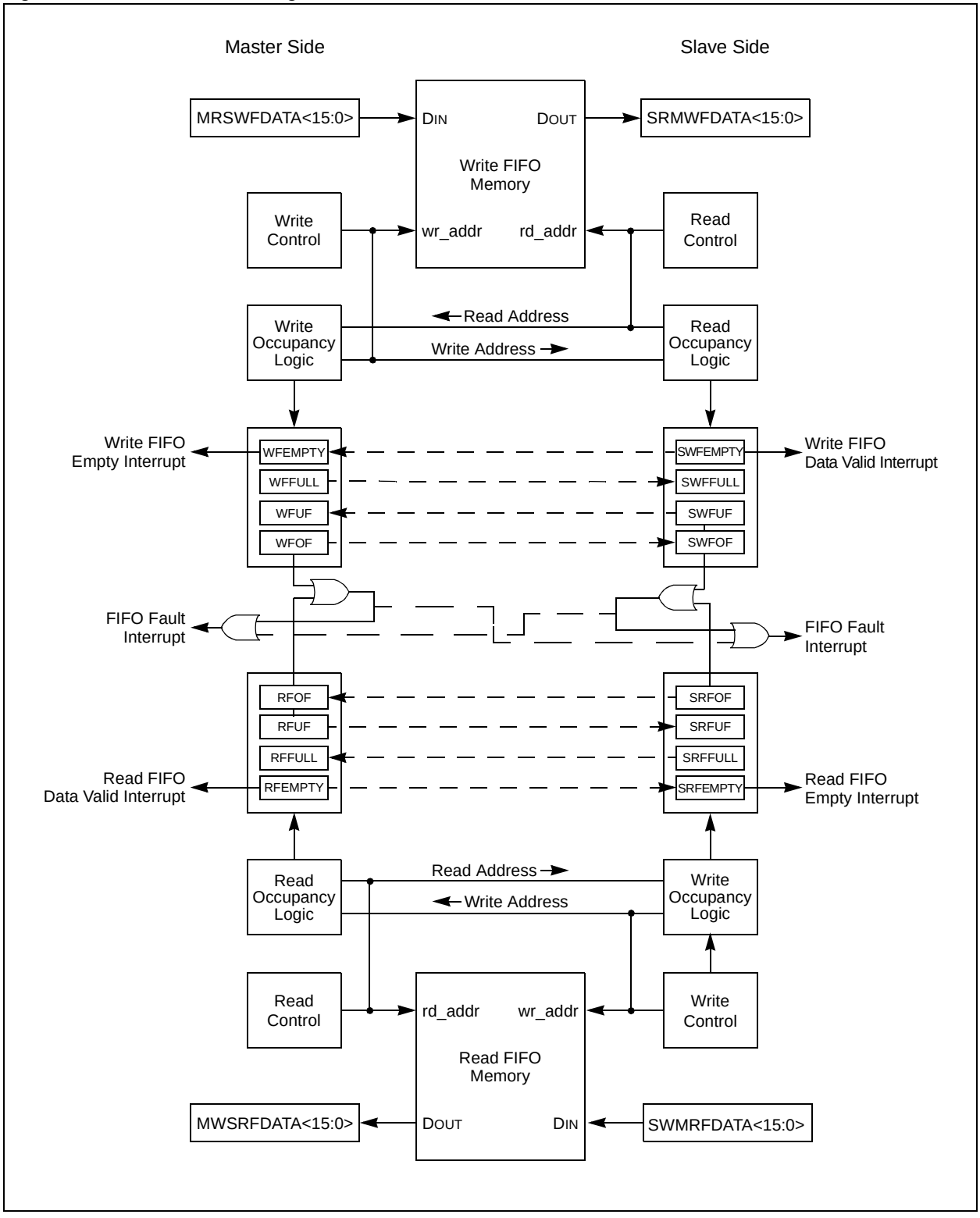
**Example 8-1: Master to Slave Write**

```
////////////////////////////////////////////////////////////////////////////
Example of the Master project to write data to the Slave

#include "p33CH128RA508.h"
unsigned char Count;
int      main()
{
             MSI1FIFOCSbits.WFEN=1;
             Count=1;
while(Count<=32                          //buffer size of 32
{
             MWSRFDATA=Count;            //Master write to FIFO
             Count++;
}
while(1);

}

////////////////////////////////////////////////////////////////////////////
Example of the Slave project to Read the data written by the Master

#include "p33CH128RA508S1.h"
unsigned int  SRdata[32];
unsigned char Count;
int      main(void) {

         while(SI1FIFOCSbits.SRFFULL==0);  // Wait till the 32 but buffer is full
         Count=32;
         while(Count!=0)
         {
              SRdata[Count]=SRMWFDATA;
              Count--;
         }
         while(1);

}
```

**Example 8-2: Slave to Master Write**

```
////////////////////////////////////////////////////////////////////////////
Example of the Slave project to Write data to Master (Master enables the Slave write bit)

#include "p33CH128RA508S1.h"

unsigned char Count;
int     main(void)
{
        while(SI1FIFOCSbits.SWFEN==0); // wait till Master enables the Masters Read FIFO
        Count=1;
        while(Count<=32)               // Fill till the buffer is full
        {
        SWMRFDATA=Count;
        Count++;
        }

        while(1);
    }

////////////////////////////////////////////////////////////////////////////
Example of the Master project to read the data written by slave

#include "p33CH128RA508.h"
unsigned char Count;
unsigned int  MRdata[32];
int     main()
{
        MSI1FIFOCSbits.RFEN=1;     // enable the read (Slave SWFEN)
        while(MSI1FIFOCSbits.RFFULL==0); // wait till the read buffer is full
        Count=32;
        while(Count!=0)
        {
        MRdata[Count]=MRSWFDATA;
        Count--;
        }
while(1);

}
////////////////////////////////////////////////////////////////////////////
```

**Figure 8-1: FIFO Block Diagram**



### 8.2.8 USING A FIFO WITH DMA

The MSI FIFOs can be used with the DMA module. For FIFO write operations using the DMA, the FIFO empty interrupt is used to trigger the corresponding data write DMA channel. This interrupt will be asserted once when the FIFO becomes empty. The DMA channel can then transfer the next block of data into the FIFO. The block can be of any size, up to the capacity of the FIFO.

For FIFO read operations using the DMA, the FIFO data valid interrupt is used to trigger the corresponding data read DMA channel. This interrupt will be asserted whenever the FIFO is no longer empty and will remain asserted until the FIFO becomes empty. This will allow the DMA channel to be retriggered, and continues moving data until such time that all data has been moved (FIFO empty), or the DMA deems that the transfer is complete. Whenever the data read DMA channel empties the FIFO, the FIFO empty interrupt will be asserted and cause the data write DMA channel to reload the FIFO as described above.

### 8.2.9 USING THE FIFO ERROR INTERRUPTS

The MSI FIFOs will generate a FIFO error interrupt to both the Master and Slave processors in the event of a FIFO overflow or underflow condition.

The data transmitter processor is responsible for correcting (write related) overflow errors. At some point, after detecting the error and prior to using the FIFO again, the corresponding (sticky) overflow status bit must be cleared. In addition, a data transmitter processor can also observe the state of the FIFO underflow error. Although the data transmitter cannot correct a read underflow, it can stop sending data into the FIFO while the error persists and/or interrogates the data receiver processor.

Similarly, the data receiver processor is responsible for correcting (read related) underflow errors. At some point, after detecting the error and prior to using the FIFO again, the corresponding (sticky) underflow status bit must be cleared. In addition, a data receiver processor can also observe the state of the FIFO overflow error. Although the data receiver cannot correct a write overflow, it can stop reading data from the FIFO while the error persists and/or interrogates the data transmitter processor.

### 8.2.10 FIFO CHANNEL LATENCY

There will be a delay between when the data was written and when it becomes available to be received. This is referred to as the FIFO channel latency. The synchronization delay is 3 destination clocks. Therefore, for signals/data moving from the transmitter to the receiver, the delay would be 3 receiver clocks. Similarly, for signals/data moving from the receiver to the transmitter, the delay would be 3 transmitter clocks.

Time taken to write will be:

1. FIFO write cycle (1 transmitter clock).
2. 3 sync cycles (3 receiver clocks).

### 8.2.11 SLAVE RESET TO THE FIFOs

Because both the Read and Write FIFOs are controlled from the Master side of the interface, the Master must be made aware of the Slave Reset in order to restart the FIFO channel(s). Consequently, unless a mailbox-based protocol is in place to communicate to the Master that the Slave has just been reset, the Master processor should monitor the Slave Reset Status bit, SLVRST (MSI1STAT<15>), or enable the Slave Reset Event Interrupt Enable bit, SRSTIE (MSI1CON<7> = 1), and restart the enabled FIFO if a Slave Reset is detected.

## 9.0 INTER-PROCESSOR INTERRUPTS

The Master and Slave processors may interrupt each other directly. The Master may issue an interrupt request to the Slave by asserting the MTSIRQ control bit (MS1CON<9>). Similarly, the Slave may issue an interrupt request to the Master by asserting the STMIRQ control bit (SI1CON<9>) control bit.

The interrupts are Acknowledged through the use of the interrupt Acknowledge control bits (MTSIACK (MSI1CON<8>) for the Master to Slave interrupt request and STMIACK (SI1CON<8>) for the Slave to Master interrupt request).

All Master/Slave interrupt control/status bits are readable by either processor. The interrupt request bits are read/write by the requesting processor and the interrupt Acknowledge bits are read/write by the interrupted processor through the MSI1CON control register. The interrupt request bits are read-only by the interrupted processor and the interrupt Acknowledge bits are read-only by the requesting processor through the MSI1STAT status register.

### 9.1 Master to Slave Interrupt Protocol

When the Master asserts the MTSIRQ bit, it is synchronized with the Slave clock to become a Slave interrupt. Once the Master sets the MTSIRQ bit (MSI1CON<9>), the Slave will get an interrupt with the MSIMIF bit getting set in the IFSx register. From the Slave perspective, the interrupt will set a read-only status bit (SI1STAT<9>). The Slave must Acknowledge the interrupt by setting the STMIACK bit at some point within the handler when servicing the interrupt.

After synchronization into the Master clock domain, the Master will observe that MTSIACK (MSI1STAT<8>) = 1 and then clear (its view of) the MTSIRQ bit, rescinding the request. The handshake is completed by the Slave when it observes that STMIRQ> = 0. At that point, the Slave clears STMIACK to rescind the Acknowledge and the interrupt handler may then exit.

**Example 9-1: Master to Slave Interrupt Protocol**

```
//////////////////Master code Master to Slave interrupt Protocol//////////////////////

while(1)
{
    MSI1CONbits.MSTIRQ=1;                // Interrupt to slave
    while(MSI1STATbits.MTSIACK==0);      // wait till slave acknowledges
    MSI1CONbits.MSTIRQ=0;                // clear the interrupt to repeat the next
    while(MSI1STATbits.MTSIACK==1);      //  wait till slave clears the acknowledge

}

/////////////////Slave code for Master to Slave interrupt Protocol/////////////////
while(1)
{
    while(IFS8bits.MSIMIF==0);           // wait for the interrupt
    IFS8bits.MSIMIF=0;
    SI1CONbits.MTSIACK=1;                // Acknowledge the master interrupt
    while(SI1STATbits.MTSIRQ==1);        //wait till master clears the interrupt request
    SI1CONbits.MTSIACK=0;                //
}
```

**Note:** The user must clear MTSIRQ in order to be able to generate another interrupt. That is, writing a logic ‘1’ to the MTSIRQ bit when it is already set, will not generate another interrupt pulse.

The Master should (but is not required to) wait for the Slave to rescind STMIACK, prior to asserting MTSIRQ again (to generate another interrupt, assuming MTSIRQ has been cleared beforehand). When using the handshake described above, failure to wait for the Slave to rescind STMIACK will just leave the new interrupt pending until the current one has exited.

## 9.2 Slave to Master Interrupt Protocol

When the Slave asserts the STMIRQ bit, it is synchronized with the Master clock to become a Master Interrupt (MSISxIF). Once the Slave sets the STMIRQ bit (SI1CON<9>), the Master will get an interrupt with the MSISxIF bit getting set in the IFSx register. From the Master perspective, the interrupt will set a read-only status bit (MSI1STAT<9>).The Master must Acknowledge the interrupt by setting the MTSIACK bit at the end of the handler when servicing of the interrupt is complete.

After synchronization into the Slave clock domain, the Slave will observe that STMIACK (SI1STAT<8>) = 1 and then clear (its view of) the STMIRQ (SI1CON<9>) bit, rescinding the request. The handshake is completed by the Master when it observes that STMIRQ (MSI1STAT<9>) = 0. At that point, the Master clears STMIACK (MSI1CON<8>) to rescind the Acknowledge and the interrupt handler may then exit.

**Example 9-2: Slave to Master Interrupt Protocol**

```
//////////////////Master code Slave to Master Interrupt Protocol//////////////////////

while(1)
{
    while(IFS8bits.MSIS1IF==0);          // wait for the Slave interrupt
    IFS8bits.MSIS1IF=0;
    MSI1CONbits.STMIACK=1;               // ACK the slave
    while(MSI1STATbits.STMIRQ==1);       // wait till the slave clears the Interrupt request
    MSI1CONbits.STMIACK=0;

}

/////////////////Slave code for Slave to Master Interrupt Protocol/////////////////
while(1)
{
    SI1CONbits.STMIRQ=1;                 // Interrupt the Master
    while(SI1STATbits.STMIACK==0)        // Wait for ACK from the Master
    SI1CONbits.STMIRQ=0;                 // Clear the interrupt request
    while(SI1STATbits.STMIACK==1)        //wait till Master clears the acknowledge
}
```

## 10.0 MASTER/SLAVE RESET INTERACTION

When operating in any mode, the user may choose how the remaining Run-Time Resets (defined as any Reset that is not a POR, BOR, $\overline{\text{MCLR}}$, or in Dual Debug mode, $\overline{\text{SMCLR}}$ Reset) from the Master and Slave will affect the SLVEN (MSI1CON<15>) control bit, based on the state of 2 fuses: Master Slave Reset Enable bit (MSRE, FSLV1DEVOPT<15>) and Slave Reset Enable bit (SSRE, FSLV1DEVOPT<14>).

The SLVEN bit is essentially a Slave Reset control, so these fuses may be used to effectively couple or decouple the Master and Slave Run-Time Resets (MSRE), and additionally determine whether the Slave continues operation or disables itself in the event of a Slave Run-Time Reset (SSRE). The default state (when both MSRE and SSRE are unprogrammed) is to allow both Master and Slave Resets to reset SLVEN and disable the Slave.

| | |
|---|---|
| **Note:** | When MSRE = 1, any Master Reset will reset the Slave (Op Code Reset, Watchdog Timer Time-out Reset, Trap Reset, Illegal Instruction Reset). When MSRE = 0, the Slave can run independently without a Reset when the Master encounters a Reset. The SSRE bit determines if the SLVEN bit is disabled during a Slave Reset. If SSRE = 1, the Slave generated Resets will reset the Slave Reset Enable bit. If SSRE = 0, the Slave generated Resets will not reset the Slave Reset Enable bit in the MSI module. |

### 10.1 Slave Reset Coupling Control

In all operating modes, the user may couple or decouple the Master Run-Time Resets to the Slave Reset by using the Master Slave Reset Enable (MSRE) fuse. The Resets are effectively coupled by directing the selected Reset source to the SLVEN bit Reset.

In all operating modes, the user may also choose whether the SLVEN bit is reset or not in the event of a Slave Run-Time Reset by using the Slave Reset Enable (SSRE) fuse.

A user may choose to reset SLVEN in the event of a Slave Reset because that event could be an indicator of a problem with Slave execution. The Slave would be placed in Reset and the Master alerted (via the Slave Reset event interrupt) to attempt to rectify the problem. The Master must re-enable the Slave by setting the SLVEN bit again.

Alternatively, the user may choose to not halt the Slave in the event of a Slave Reset, and just allow it to restart execution after a Reset and continue operation as soon as possible. The Slave Reset event interrupt would still occur, but could be ignored by the Master.

**Table 10-1: Application Mode SLVEN Reset Control Truth Table**

| MSRE | SSRE | SLVEN Bit Reset Source | Application Effect |
|---|---|---|---|
| 0 | 0 | POR/BOR/$\overline{\text{MCLR}}$ | • Slave is reset and disabled in the event of a POR, BOR or $\overline{\text{MCLR}}$ Reset. Master must re-enable Slave.<br>• Slave Run-Time Resets will not disable Slave. Slave will reset and continue execution (and may optionally interrupt Master). |
| 1 | 0 | Master Resets(1) | • Slave is reset and disabled in the event of any Master Reset. Master must re-enable Slave.<br>• Slave Run-Time Resets will not disable Slave. Slave will reset and continue execution (and may optionally interrupt Master). |
| 0 | 1 | Slave Resets(2) | • Slave is reset and disabled in the event of any Slave Run-Time Reset (and may optionally interrupt Master). Master must re-enable Slave to execute the Slave code.<br>• Master Run-Time Resets will not affect Slave operation. |
| 1 | 1 | Master Resets(1)/ Slave Resets(2) | • Slave is reset and disabled in the event of any Slave Run-Time Reset or Master Reset. Master must re-enable Slave. This represents the default state (MSRE and SSRE are unprogrammed). |

**Note 1:** Master Resets include any Master Reset, such as POR/BOR/$\overline{\text{MCLR}}$ Resets.

**2:** Slave Resets include any Slave Reset, plus POR/BOR/$\overline{\text{MCLR}}$ Resets (in Application mode).

## 11.0 INTER-PROCESSOR OPERATING MODE STATUS

The application operating mode status of all processors is made available through the MSI1STAT register. Each Slave can observe the operating status of the Master and the Master can observe the operating status of each Slave. A Slave processor cannot directly view the operating state of any other Slave processor.

### 11.1 Slave Processor Reset Status (for Master)

The state of the Slave processor Reset is available to the Master processor by observing the state of the Master view of the SLVRST bit. The bit will remain set until the Slave exits the Reset state. When the Slave is disabled (SLVEN> = 0), it is held in the Reset state, so SLVRST will be set.

This bit is not mapped into the Slave side of the interface and is R-0 from the Master side. A device POR, BOR or MCLR Reset will always reset both the Master and Slave, and therefore, the SLVRST bit. The bit otherwise represents the state of the Slave Reset. Consequently, if the Slave is also reset by the Master Reset (or was already in Reset), or the Slave is disabled (SLVEN = 0) either as the result of the Master Reset or prior to it, the SLVRST bit will appear to be reset to logic '1'. If the Slave is already enabled (SLVEN = 1), it is unaffected by the Master Reset and the SLVRST bit will be reset to logic '0'.

When the Master wishes to take action whenever the Slave resets, the SLVRST bit may be used to generate a 'Slave Reset Event' interrupt. For this interrupt to work, the SRSTIE (MSI1CON<7> bit should be set. When enabled, a Slave Reset event interrupt will be generated for the Master upon the leading edge (only) of any Slave Run-Time Reset event (i.e., not POR/BOR or MCLR) that occurs.

| | |
|---|---|
| **Note:** | The associated 'Slave Reset Event' interrupt flag in the Master interrupt controller must be cleared by the Interrupt Service Routine (ISR) prior to returning to avoid re-entry.<br><br>To avoid an unwanted 'Slave Reset Event' interrupt when intentionally disabling the Slave, the user must clear the Slave Reset Event Interrupt Enable bit (SRSTIE (MSI1CON<7>) = 0) prior to disabling the Slave (SLVEN = 0). |

The SLVRST bit is intended to provide a means for the Master to check if the Slave is able to respond prior to attempting to communicate with it. As such, it remains asserted throughout the Slave Reset event and cannot be cleared by the Master. However, it is also an interrupt event source, but only when SLVRST transitions from a '0' to a '1'. No subsequent interrupts will occur if it remains asserted or when it is cleared (i.e., when the Slave Reset state is exited).

In the event that SLVRST = 1, the Master may:

- Wait for SLVRST = 0 within the ISR
- Log the event and return to application operation while periodically checking the state of the SLVRST status bit
- Re-initialize the Slave by disabling it (SLVEN = 0) and reloading the Slave PRAM prior to re-enabling it

#### 11.1.1 SLVRST WHEN MASTER IS IN SLEEP MODE

Should the Master be in Sleep mode and a Slave Reset event sets SLVRST = 1, the resulting 'Slave Reset Event' interrupt will be able to wake-up the Master.

## 11.2 Master Processor Reset Status (for Slave)

The state of the Master processor Reset is available to the Slave processor by observing the state of the Slave view of the MSTRST (SI1STAT<15>) bit. The bit will remain set until the Master exits the Reset state.

This bit is not mapped into the Master side of the interface and is R-0 from the Slave side. It will always read as '0' unless the MSRE fuse = 0 (because the Slave will also be reset whenever the Master is reset when MSRE = 1). A device POR or BOR Reset will always reset both the Slave, and therefore, the MSTRST bit. The bit otherwise represents the state of the Master Reset (i.e., when MSRE = 0).

When the Slave wishes to take action whenever the Master resets, the MSTRST bit may be used to generate a 'Master Reset Event' interrupt. This interrupt is subject to being enabled by setting STMIRQ (SI1CON<9> = 1). When enabled, a 'Master Reset Event' interrupt will be generated for the Slave upon the leading edge of any Master Run-Time Reset (i.e., not POR/BOR/$\overline{\text{MCLR}}$) event that occurs. The interrupt will set the associated interrupt flag in the Master interrupt controller macro.

| | |
|---|---|
| **Note:** | The 'Master Reset Event' interrupt is edge-sensitive, occurring only when the Master enters the Reset state when the interrupt enable bit is set. However, the associated 'Master Reset Event' interrupt flag in the Slave interrupt controller must be cleared by the ISR prior to return to avoid re-entry. |

### 11.2.1 MSTRST USAGE EXAMPLE

The MSTRST bit is intended to provide a means for the Slave (when independently reset because the MSRE fuse = 0) to check if the Master is able to respond prior to attempting to communicate with it. As such, it remains asserted throughout the Master Reset event and cannot be cleared by the Slave. However, it is also an interrupt event source, but only when MSTRST transitions from a '0' to a '1'. No subsequent interrupts will occur if it remains asserted or when it is cleared (i.e., when the Master Reset state is exited).

In the event that MSTRST = 1, the Slave may:

- Log the event and return to application operation while periodically checking the state of the MSTRST status bit
- Wait for MSTRST = 0 within the ISR (effectively placing the entire device in a Halted state)
- Restart (in case it is misreading the status due to a temporary Fault condition)
- Validate PRAM contents (e.g., checksum) and halt or restart as a result

If the user requires knowledge of past Slave Reset events, this could be garnered by using the associated ISR code to log the events.

### 11.2.2 MSTRST WHEN SLAVE IS IN SLEEP MODE

If the MSRE (Master Slave Reset Enable) fuse is programmed (to logic '0'), the Master and Slave Resets are decoupled. Should this be the case, and the Slave is in Sleep mode, a Master Reset event will set MSTRST = 1 and the resulting 'Master Reset Event' interrupt will be able to wake-up the Slave. If MSRE = 1, Master and Slave Resets are coupled so a Master Reset will also reset the Slave (and exit Sleep mode).

## 11.3 System Watchdog Timer Status

The state of the Slave processor Watchdog Timer (WDT) Reset is available to the Master processor by observing the state of the Master view of the SLVWDRST (MSI1STAT<14>) bit. If the WDT has timed out and forced a Slave Reset, this bit will be set; it will remain set until cleared by the Master. This bit is not mapped into the Slave side of the interface and is R/C (Read or Clear only) from the Master side.

The SLVWDRST bit is reset (together with the rest of the Slave) when the Master Reset is asserted. Consequently, a Master WDT Reset status (for the Slave) is not meaningful.

| **Note:** | SLVWDRST is not affected should the Slave be disabled (SLVEN = 0). |
|---|---|

### 11.3.1 LOW-POWER OPERATING MODE STATUS

The Slave processor Low-Power Operating mode status is indicated by the SLVPWR<1:0> (MSI1STAT<13:12>) bits. These bits are not visible from the Slave side of the interface and are read-only from the Master side. Similarly, the Master processor Low-Power Operating mode status is indicated by the MSTPWR<1:0> (SI1STAT<13:12>) bits.These bits are not mapped into the Master side of the interface and are read-only from the Slave side.

## 12.0 RELATED APPLICATION NOTES

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC33/PIC24 device families, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Master Slave Interface (MSI) module are:

| Title | Application Note # |
|---|---|
| No related application notes at this time. | N/A |

**Note:** Visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the dsPIC33/PIC24 device families.

## 13.0 REVISION HISTORY

### Revision A (August 2016)

This is the initial version of this document.

### Revision B (March 2018)

- Tables:
  - Updated Table 2-1 and Table 2-2.
- Examples:
  - Added Example 4-2, Example 9-1 and Example 9-2.
  - Updated Example 7-1.
- Registers:
  - Updated Register 2-1 and Register 2-9.
- Sections:
  - Updated **Section 8.2.6 “FIFO Data Valid (Ready) Interrupt (MSIDTIF)”**.

**NOTES:**

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.



*Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

**QUALITY MANAGEMENT SYSTEM**
**CERTIFIED BY DNV**
**═ ISO/TS 16949 ═**

**Trademarks**



ISBN: 978-1-5224-2802-2

# Worldwide Sales and Service

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

**Atlanta**
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

**Austin, TX**
Tel: 512-257-3370

**Boston**
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

**Chicago**
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

**Dallas**
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

**Detroit**
Novi, MI
Tel: 248-848-4000

**Houston, TX**
Tel: 281-894-5983

**Indianapolis**
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

**Raleigh, NC**
Tel: 919-844-7510

**New York, NY**
Tel: 631-435-6000

**San Jose, CA**
Tel: 408-735-9110
Tel: 408-436-4270

**Canada - Toronto**
Tel: 905-695-1980
Fax: 905-695-2078

## ASIA/PACIFIC

**Australia - Sydney**
Tel: 61-2-9868-6733

**China - Beijing**
Tel: 86-10-8569-7000

**China - Chengdu**
Tel: 86-28-8665-5511

**China - Chongqing**
Tel: 86-23-8980-9588

**China - Dongguan**
Tel: 86-769-8702-9880

**China - Guangzhou**
Tel: 86-20-8755-8029

**China - Hangzhou**
Tel: 86-571-8792-8115

**China - Hong Kong SAR**
Tel: 852-2943-5100

**China - Nanjing**
Tel: 86-25-8473-2460

**China - Qingdao**
Tel: 86-532-8502-7355

**China - Shanghai**
Tel: 86-21-3326-8000

**China - Shenyang**
Tel: 86-24-2334-2829

**China - Shenzhen**
Tel: 86-755-8864-2200

**China - Suzhou**
Tel: 86-186-6233-1526

**China - Wuhan**
Tel: 86-27-5980-5300

**China - Xian**
Tel: 86-29-8833-7252

**China - Xiamen**
Tel: 86-592-2388138

**China - Zhuhai**
Tel: 86-756-3210040

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-3090-4444

**India - New Delhi**
Tel: 91-11-4160-8631

**India - Pune**
Tel: 91-20-4121-0141

**Japan - Osaka**
Tel: 81-6-6152-7160

**Japan - Tokyo**
Tel: 81-3-6880- 3770

**Korea - Daegu**
Tel: 82-53-744-4301

**Korea - Seoul**
Tel: 82-2-554-7200

**Malaysia - Kuala Lumpur**
Tel: 60-3-7651-7906

**Malaysia - Penang**
Tel: 60-4-227-8870

**Philippines - Manila**
Tel: 63-2-634-9065

**Singapore**
Tel: 65-6334-8870

**Taiwan - Hsin Chu**
Tel: 886-3-577-8366

**Taiwan - Kaohsiung**
Tel: 886-7-213-7830

**Taiwan - Taipei**
Tel: 886-2-2508-8600

**Thailand - Bangkok**
Tel: 66-2-694-1351

**Vietnam - Ho Chi Minh**
Tel: 84-28-5448-2100

## EUROPE

**Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

**Denmark - Copenhagen**
Tel: 45-4450-2828
Fax: 45-4485-2829

**Finland - Espoo**
Tel: 358-9-4520-820

**France - Paris**
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

**Germany - Garching**
Tel: 49-8931-9700

**Germany - Haan**
Tel: 49-2129-3766400

**Germany - Heilbronn**
Tel: 49-7131-67-3636

**Germany - Karlsruhe**
Tel: 49-721-625370

**Germany - Munich**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Germany - Rosenheim**
Tel: 49-8031-354-560

**Israel - Ra'anana**
Tel: 972-9-744-7705

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Italy - Padova**
Tel: 39-049-7625286

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**Norway - Trondheim**
Tel: 47-7289-7561

**Poland - Warsaw**
Tel: 48-22-3325737

**Romania - Bucharest**
Tel: 40-21-407-87-50

**Spain - Madrid**
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

**Sweden - Gothenberg**
Tel: 46-31-704-60-40

**Sweden - Stockholm**
Tel: 46-8-5090-4654

**UK - Wokingham**
Tel: 44-118-921-5800
Fax: 44-118-921-5820