Using the PICmicro® MSSP Module for Master I^{2}C™ Communications

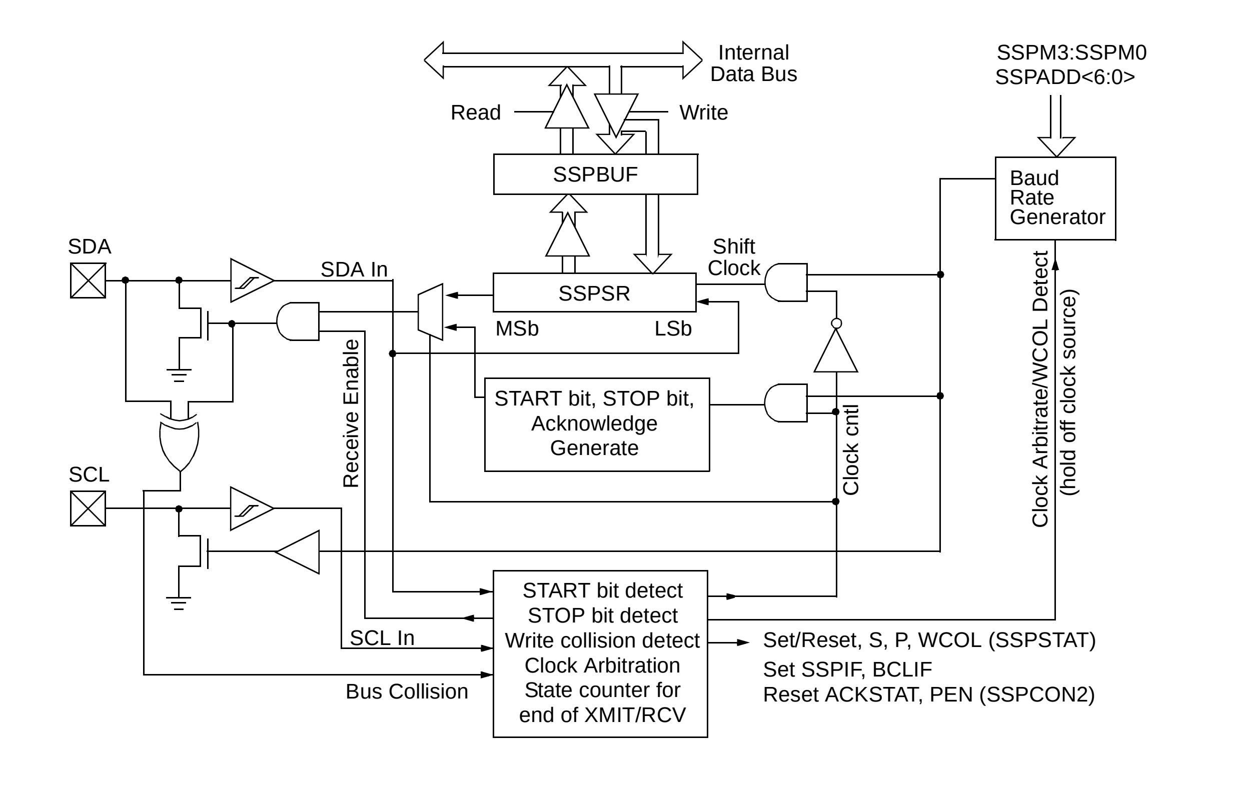
Author: Richard L. Fischer
Microchip Technology Inc.

INTRODUCTION

This application note describes the implementation of the PICmicro MSSP module for Master I^{2}C communications. The Master Synchronous Serial Port (MSSP) module is the enhanced Synchronous Serial Port developed by Microchip Technology and is featured on many of the PICmicro devices. This module provides for both the 4-mode SPI communications, as well as Master and Slave I^{2}C communications, in hardware.

For information on the SPI™ peripheral implementation see the PICmicro™ Mid-Range MCU Family Reference Manual, document DS33023. The MSSP module in I^{2}C mode fully implements all Master and Slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free I^{2}C bus (multi-master function). The MSSP module implements the standard mode specifications, as well as 7-bit and 10-bit addressing. Figure 1 depicts a functional block diagram of the I^{2}C Master mode. The application code for this I^{2}C example is developed for and tested on a PIC16F873, but can be ported over to a PIC17CXXX and PIC18CXXX PICmicro MCU which features a MSSP module.

FIGURE 1: I^{2}C MASTER MODE BLOCK DIAGRAM



THE I^2C BUS SPECIFICATION

Although a complete discussion of the I^2C bus specification is outside the scope of this application note, some of the basics will be covered here. For more information on the I^2C bus specification, you may refer to sources indicated in the *References* section.

The Inter-Integrated-Circuit, or I^2C bus specification was originally developed by Philips Inc. for the transfer of data between ICs at the PCB level. The physical interface for the bus consists of two open-collector lines; one for the clock (SCL) and one for data (SDA). The SDA and SCL lines are pulled high by resistors connected to the VDD rail. The bus may have a one Master/many Slave configuration or may have multiple master devices. The master device is responsible for generating the clock source for the linked Slave devices.

The I^2C protocol supports either a 7-bit addressing mode, or a 10-bit addressing mode, permitting 128 or 1024 physical devices to be on the bus, respectively. In practice, the bus specification reserves certain addresses so slightly fewer usable addresses are available. For example, the 7-bit addressing mode allows 112 usable addresses. The 7-bit address protocol is used in this application note.

All data transfers on the bus are initiated by the master device and are done eight bits at a time, MSb first. There is no limit to the amount of data that can be sent in one transfer. After each 8-bit transfer, a 9th clock pulse is sent by the master. At this time, the transmitting device on the bus releases the SDA line and the receiving device on the bus acknowledges the data sent by the transmitting device. An ACK (SDA held low) is sent if the data was received successfully, or a NACK (SDA left high) is sent if it was not received successfully. A NACK is also used to terminate a data transfer after the last byte is received.

According to the I^2C specification, all changes on the SDA line must occur while the SCL line is low. This restriction allows two unique conditions to be detected on the bus; a START sequence (**S**) and a STOP sequence (**P**). A START sequence occurs when the master device pulls the SDA line low while the SCL line is high. The START sequence tells all Slave devices on the bus that address bytes are about to be sent. The STOP sequence occurs when the SDA line goes high while the SCL line is high, and it terminates the transmission. Slave devices on the bus should reset their receive logic after the STOP sequence has been detected.

The I^2C protocol also permits a Repeated Start condition (**Rs**), which allows the master device to execute a START sequence without preceding it with a STOP sequence. The Repeated Start is useful, for example, when the Master device changes from a write operation to a read operation and does not release control of the bus.

MSSP MODULE SETUP, IMPLEMENTATION AND CONTROL

The following sections describe the setup, implementation and control of the PICmicro MSSP module for I^2C Master mode. Some key Special Function Registers (SFRs) utilized by the MSSP module are:

1. SSP Control Register1 (SSPCON1)
2. SSP Control Register2 (SSPCON2)
3. SSP Status Register (SSPSTAT)
4. Pin Direction Control Register (TRISC)
5. Serial Receive/Transmit Buffer (SSPBUF)
6. SSP Shift Register (SSPSR) - Not directly accessible
7. SSP Address Register (SSPADD)
8. SSP Hardware Event Status (PIR1)
9. SSP Interrupt Enable (PIE1)
10. SSP Bus Collision Status (PIR2)
11. SSP Bus Collision Interrupt Enable (PIE2)

Module Setup

To configure the MSSP module for Master I^2C mode, there are key SFR registers which must be initialized. Respective code examples are shown for each.

1. SSP Control Register1 (SSPCON1)
   - I^2C Mode Configuration
2. SSP Address Register (SSPADD<6:0>)
   - I^2C Bit Rate
3. SSP Status Register (SSPSTAT)
   - Slew Rate Control
   - Input Pin Threshold Levels (SMbus or I^2C)
4. Pin Direction Control (TRISC)
   - SCL/SDA Direction

To configure the MSSP module for Master I^2C mode, the SSPCON1 register is modified as shown in Example 1.

EXAMPLE 1: I^2C MODE CONFIGURATION

```
movlw   b’00101000’ ; setup value
                    ;   into W register
banksel SSPCON1     ; select SFR
                    ;   bank
movwf   SSPCON1     ; configure for
                    ;   Master I2C
```

With the two-wire synchronous I^2C bus, the Master generates all clock signals at a desired bit rate. Using the formula in Equation 1, the bit rate can be calculated and written to the SSPADD register. For a 400kHz bit rate @ Fosc = 16MHz, the SSPADD register is modified as shown in Example 2.

EQUATION 1: BIT RATE CALCULATION

$$SSPADD = \frac{\left(\frac{Fosc}{Bit\ Rate}\right)}{4} - 1$$

EXAMPLE 2: I²C BIT RATE SETUP

```
movlw   b’00001001’ ; setup value
                    ;  into W register
banksel SSPADD      ; select SFR bank
movwf   SSPADD      ; baud rate =
                    ;  400KHz @ 16MHz
```

To enable the slew rate control for a bit rate of 400kHz and select I^{2}C input thresholds, the SSPSTAT register is modified as shown in Example 3.

EXAMPLE 3: SLEW RATE CONTROL

```
movlw   b’00000000’ ; setup value
                    ;  into W register
movwf   SSPSTAT     ; slew rate
                    ;  enabled
banksel SSPSTAT     ; select SFR bank
```

The SSPSTAT register also provides for read-only status bits which can be utilized to determine the status of a data transfer, typically for the Slave data transfer mode. These status bits are:

- D/$\overline{A}$ - Data/Address
- P - STOP
- S - START
- R/$\overline{W}$ - Read/Write Information
- UA - Update Address (10-bit mode only)
- BF - Buffer Full

Finally, before selecting any I^{2}C mode, the SCL and SDA pins must be configured to inputs by setting the appropriate TRIS bits. Selecting an I^{2}C mode by setting the SSPEN bit (SSPCON1<5>), enables the SCL and SDA pins to be used as the clock and data lines in I^{2}C mode. A logic "1" written to the respective TRIS bits configure these pins as inputs. An example setup for a PIC16F873 is shown in Example 4. Always refer to the respective data sheet for the correct SCL and SDA TRIS bit locations.

EXAMPLE 4: SCL/SDA PIN DIRECTION SETUP

```
movlw   b’00011000’ ; setup value
                    ;  into W register
banksel TRISC       ; select SFR bank
iorwf   TRISC,f     ; SCL and SDA
                    ;  are inputs
```

The four remaining SFR’s can be used to provide for I^{2}C event completion and Bus Collision interrupt functionality.

1. SSP Event Interrupt Enable bit (SSPIE)
2. SSP Event Status bit (SSPIF)
3. SSP Bus Collision Interrupt Enable bit (BCLIE)
4. SSP Bus Collision Event Status bit (BCLIF)

Implementation and Control

Once the basic functionality of the MSSP module is configured for Master I^{2}C mode, the remaining steps relate to the implementation and control of I^{2}C events.

The Master can initiate any of the following I^{2}C bus events:

1. START
2. Restart
3. STOP
4. Read (Receive)
5. Acknowledge (after a read)
   - Acknowledge
   - Not Acknowledge (NACK)
6. Write

The first four events are initiated by asserting high the appropriate control bit in the SSPCON2<3:0> register. The Acknowledge bit event consists of first setting the Acknowledge state, ACKDT (SSPCON2<5>) and then asserting high the event control bit ACKEN (SSPCON2<4>).

Data transfer with acknowledge is obligatory. The acknowledge related clock is generated by the Master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse. This sequence is termed "ACK" or acknowledge.

When the Slave doesn’t acknowledge the Master during this acknowledge clock pulse (for any reason), the data line must be left HIGH by the Slave. This sequence is termed "NACK" or not acknowledge.

Example 5 shows an instruction sequence which will generate an acknowledge event by the Master.

EXAMPLE 5: ACKNOWLEDGE EVENT

```
banksel SSPCON2        ; select SFR
                       ;  bank
bcf     SSPCON2, ACKDT ; set ack bit
                       ;  state to 0
bsf     SSPCON2, ACKEN ; initiate ack
```

Example 6 shows an instruction sequence which would generate a not acknowledge (NACK) event by the Master.

EXAMPLE 6: NOT ACKNOWLEDGE EVENT

```
banksel SSPCON2          ; select SFR
                         ;  bank
bsf     SSPCON2, ACKDT   ; set ack bit
                         ;  state to 1
bsf     SSPCON2, ACKEN   ; initiate ack
```

The I^2C write event is initiated by writing a byte into the SSPBUF register. An important item to note at this point, is when implementing a Master I^2C controller with the MSSP module, no events can be queued. One event must be finished and the module IDLE before the next event can be initiated. There are a few of ways to ensure that the module is IDLE before initiating the next event. The first method is to develop and use a generic idle check subroutine. Basically, this routine could be called before initiating the next event. An example of this code module is shown in Example 7.

EXAMPLE 7: CODE MODULE FOR GENERIC IDLE CHECK

```
i2c_idle                 ; routine name
  banksel SSPSTAT        ; select SFR
                         ;  bank
  btfsc   SSPSTAT,R_W    ; transmit
                         ;  in progress?
  goto    $-1            ; module busy
                         ;  so wait
  banksel SSPCON2        ; select SFR
                         ;  bank
  movf    SSPCON2,w      ; get copy
                         ;  of SSPCON2
  andlw   0x1F           ; mask out
                         ;  non-status
  btfss   STATUS,Z       ; test for
                         ;  zero state
  goto    $-3            ; bus is busy
                         ;  test again
  return                 ; return
```

The second approach is to utilize a specific event idle check. For example, the Master initiates a START event and wants to know when the event completes. An example of this is shown in Example 8.

EXAMPLE 8: START EVENT COMPLETION CHECK

```
This code initiates an I2C start event
banksel SSPCON2       ; select SFR
                      ;  bank
bsf     SSPCON2,SEN   ; initiate
                      ;  I2C start

; This code checks for completion of I2C
; start event
btfsc   SSPCON2,SEN   ; test start
                      ;  bit state
goto    $-1           ; module busy
                      ;  so wait
```

Another example of this could be a read event completion check as shown in Example 9.

EXAMPLE 9: READ EVENT COMPLETION CHECK

```
This code initiates an I2C read event
banksel SSPCON2        ; select SFR
                       ;  bank
bsf     SSPCON2,RCEN   ; initiate
                       ;  I2C read

; This code checks for completion of I2C
; read event
btfsc   SSPCON2,RCEN   ; test read
                       ;  bit state
goto    $-1            ; module busy
                       ;  so wait
```

These examples can be modified slightly to reflect the other bus events, such as: Restart, STOP and the Acknowledge state after a read event. The bits for these events are defined in the SSPCON2 register.

For the I^{2}C write event, the idle check status bit is defined in the SSPSTAT register. An example of this is shown in Example 10.

EXAMPLE 10: WRITE EVENT COMPLETION CHECK

```
This code initiates an I$^2$C write event
banksel SSPBUF      ; select SFR bank
movlw   0xAA        ; load value
                    ;  into W
movwf   SSPBUF      ; initiate I$^2$C
                    ;  write cycle

; This code checks for completion of I$^2$C
; write event
banksel SSPSTAT     ; select SFR bank
btfsc   SSPSTAT,R_W ; test write bit
                    ;   state
goto    $-1         ; module busy
                    ;  so wait
```

The third approach is the implementation of interrupts. With this approach, the next I^{2}C event is initiated when an interrupt occurs. This interrupt is generated at the completion of the previous event. This approach will require a "state" variable to be used as an index into the next I^{2}C event (event jump table). An example of a possible interrupt structure is shown in Example 11 and the jump table is shown in Example 12. The entire code sequence is provided in Appendix A, specifically in the mastri2c.asm and i2ccomm.asm code files.

EXAMPLE 11: INTERRUPT SERVICE CODE EXCERPT

```
; Interrupt entry here
; Context Save code here.....

; I$^2$C ISR handler here

  bsf     STATUS,RP0    ; select SFR
                        ;  bank
  btfss   PIE1,SSPIE    ; test if
                        ;  interrupt is
                        ;  enabled
 goto    test_buscoll   ; no, so test for
                        ;  Bus Collision
  bcf     STATUS,RP0    ; select SFR
                        ;  bank
  btfss   PIR1,SSPIF    ; test for SSP
                        ;  H/W flag
  goto    test_buscoll  ; no, so test
                        ;  for Bus
                        ;  Collision Int
  bcf     PIR1,SSPIF    ; clear SSP
                        ;  H/W flag
  pagesel service_i2c   ; select page
                        ;  bits for
                        ; function
  call    service_i2c   ; service valid
                        ;  I$^2$C event

; Additional ISR handlers here

; Context Restore code here

  retfie                ; return
```

EXAMPLE 12: SERVICE I^{2}C JUMP TABLE CODE EXCERPT

```
service_i2c                 ; routine name

  movlw   high  I2CJump     ; fetch upper byte of jump table address
  movwf   PCLATH            ; load into upper PC latch
  movlw   i2cSizeMask       ;
  banksel i2cState          ; select GPR bank
  andwf   i2cState,w        ; retrieve current I2C state
  addlw   low (I2CJump + 1) ; calc state machine jump addr into W register
  btfsc   STATUS,C          ; skip if carry occured
  incf    PCLATH,f          ; otherwise add carry
I2CJump                     ; address were jump table branch occurs
  movwf   PCL               ; index into state machine jump table
                            ; jump to processing for each state = i2cState value
                            ;  for each state
; Jump Table entry begins here

  goto    WrtStart          ; start condition
  goto    SendWrtAddr       ; write address with R/W=1
  goto    WrtAckTest        ; test acknowledge state after address write
  goto    WrtStop           ; generate stop condition

  goto    ReadStart         ; start condition
  goto    SendReadAddr      ; write address with R/W=0
  goto    ReadAckTest       ; test acknowledge state after address write
  goto    ReadData          ; read more data
  goto    ReadStop          ; generate stop condition
```

Typical Master I^{2}C writes and reads using the MSSP module are shown in Figure 2 and Figure 3, respectively. Notice that the hardware interrupt flag bit, SSPIF (PIR1<3>), is asserted when each event completes. If interrupts are to be used, the SSPIF flag bit must be cleared before initiating the next event.

FIGURE 2: I^{2}C MASTER MODE WRITE TIMING (7 OR 10-BIT ADDRESS)

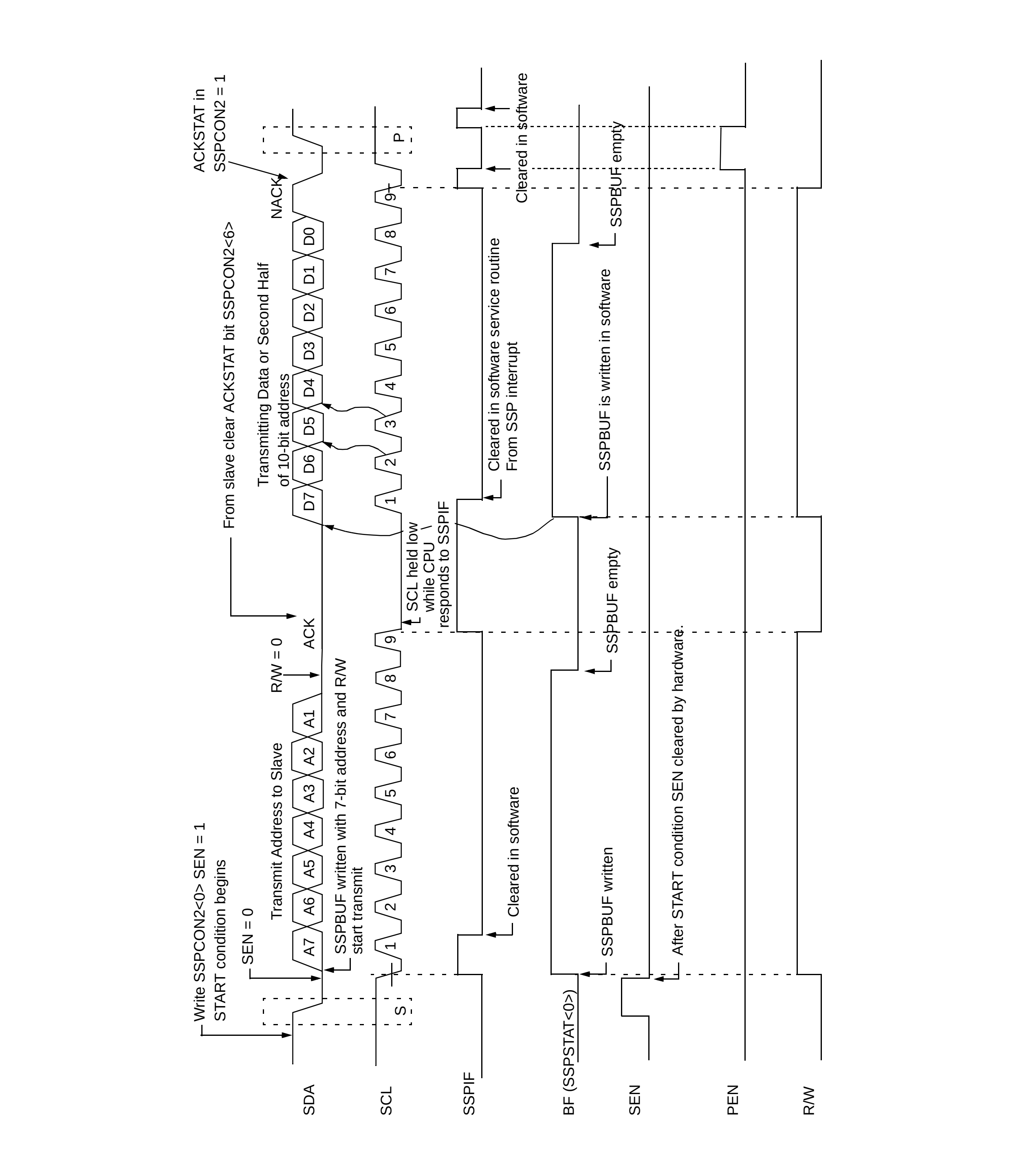
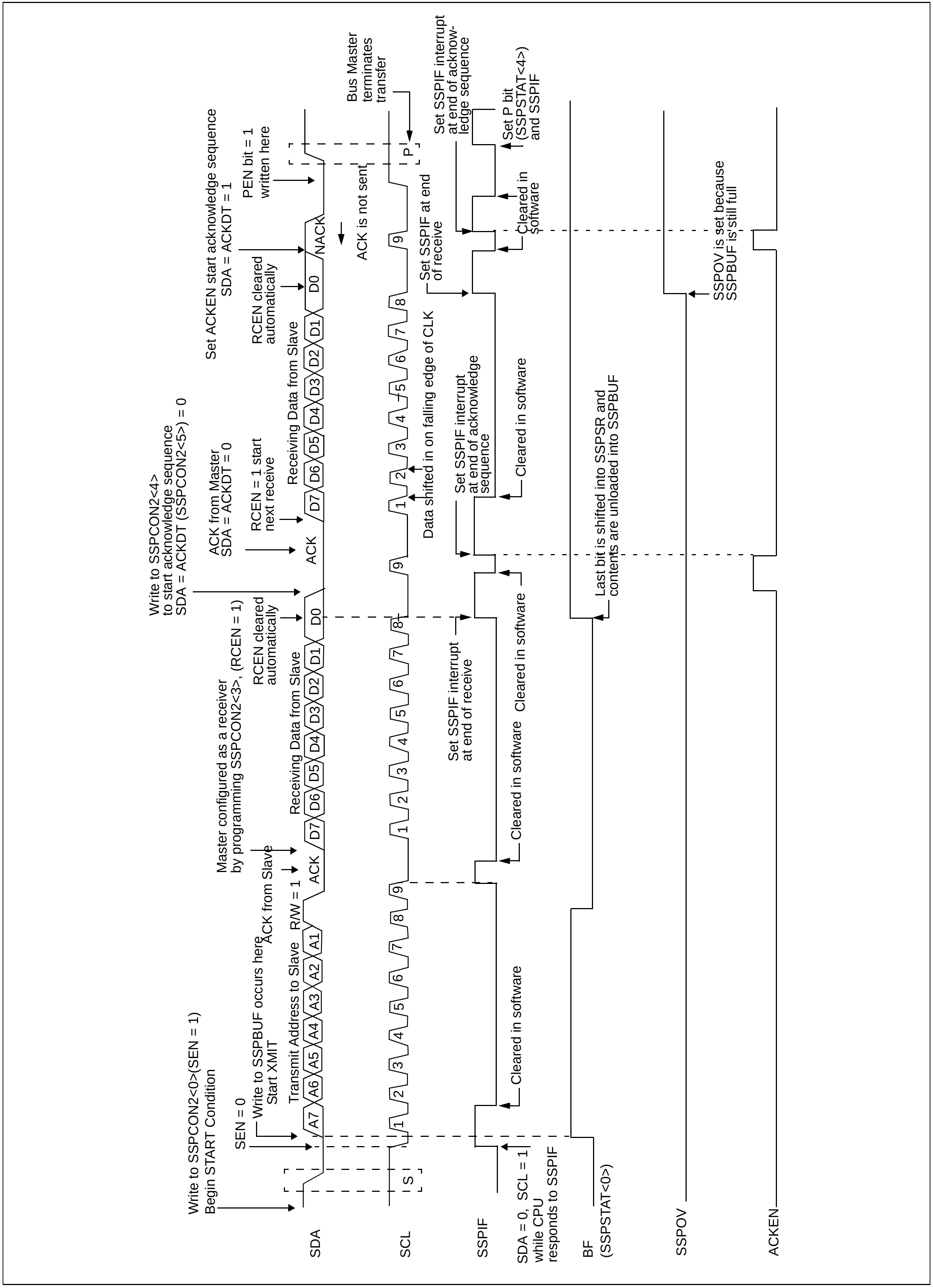


FIGURE 3: I^{2}C MASTER MODE READ TIMING (7-BIT ADDRESS)


SDA
SCL
SSPIF
SDA = 0, SCL = 1 while CPU responds to SSPIF
BF (SSPSTAT<0>)
SSPOV
ACKEN
Write to SSPCON2<0>(SEN = 1) Begin START Condition
SEN = 0
Write to SSPBUF occurs here Start XMIT
ACK from Slave
Transmit Address to Slave
R/W = 1
Master configured as a receiver by programming SSPCON2<3>, (RCEN = 1)
RCEN cleared automatically
Receiving Data from Slave
Write to SSPCON2<4> to start acknowledge sequence SDA = ACKDT (SSPCON2<5>) = 0
ACK from Master SDA = ACKDT = 0
RCEN = 1 start next receive
Receiving Data from Slave
RCEN cleared automatically
Set ACKEN start acknowledge sequence SDA = ACKDT = 1
PEN bit = 1 written here
ACK is not sent
Bus Master terminates transfer
S
A7
A6
A5
A4
A3
A2
A1
ACK
D7
D6
D5
D4
D3
D2
D1
D0
ACK
D7
D6
D5
D4
D3
D2
D1
D0
NACK
P
1
2
3
4
5
6
7
8
9
1
2
3
4
5
6
7
8
9
1
2
3
4
5
6
7
8
9
Data shifted in on falling edge of CLK
Set SSPIF interrupt at end of receive
Set SSPIF interrupt at end of acknowledge sequence
Set SSPIF at end of receive
Set SSPIF interrupt at end of acknow- ledge sequence
Set P bit (SSPSTAT<4>) and SSPIF
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Cleared in software
Last bit is shifted into SSPSR and contents are unloaded into SSPBUF
SSPOV is set because SSPBUF is still full

ERROR HANDLING

When the MSSP module is configured as a Master I^{2}C controller, there are a few operational errors which may occur and should be processed correctly. Each error condition should have a root cause and solution(s).

Write Collision (Master I^{2}C Mode)

In the event of a Write Collision, the WCOL bit (SSPCON1<7>) will be set high. This bit will be set if queueing of events is attempted. For example, an I^{2}C START event is initiated, as was shown in Example 8. Before this event completes, a write sequence is attempted by the Master firmware. As a result of not waiting for the module to be IDLE, the WCOL bit is set and the contents of the SSPBUF register are unchanged (the write doesn't occur).

> **Note:** Interrupts are not generated as a result of a write collision. The application firmware must monitor the WCOL bit for detection of this error.

Bus Collision

In the event of a Bus Collision, the BCLIF bit (PIR2<3>) will be asserted high. The root cause of the bus collision may be one of the following:

- Bus Collision during a START event
- Bus Collision during a Repeated Start event
- Bus Collision during a STOP event
- Bus Collision during address/data transfer

When the Master outputs address/data bits onto the SDA pin, arbitration takes place when the Master outputs a '1' on SDA by letting SDA float high and another Master asserts a '0'. When the SCL pin floats high, data should be stable. If the expected data on SDA is a '1' and the data sampled on the SDA pin = '0', then a bus collision has taken place. The Master will set the Bus Collision Interrupt Flag, BCLIF and reset the I^{2}C port to its IDLE state. The next sequence should begin with a I^{2}C START event.

Not Acknowledge (NACK)

A NACK does not always indicate an error, but rather some operational state which must be recognized and processed. As defined in the I^{2}C protocol, the addressed Slave device should drive the SDA line low during ninth clock period if communication is to continue. A NACK event may be caused by various conditions, such as:

- There may be a software error with the addressed Slave I^{2}C device.
- There may be a hardware error with the addressed Slave I^{2}C device.
- The Slave device may experience, or even generate, a receive overrun. In this case, the Slave device will not drive the SDA line low and the Master device will detect this.

The response of the Master depends on the software error handling layer in the application firmware. One thing to note is that the I^{2}C bus is still held by the current Master. The Master has a couple of options at this point, which are:

- Generate an I^{2}C Restart event
- Generate an I^{2}C STOP event
- Generate an I^{2}C STOP/START event

If the Master wants to retain control of the bus (Multi-Master bus) then a I^{2}C Restart event should be initiated. If a I^{2}C STOP/START sequence is generated, it is possible to lose control of the bus in a Multi-Master system. This may not be an issue and is left up to the system designer to determine the appropriate solution.

MULTI-MASTER OPERATION

In a Mutli-Master system, there is a possibility that two or more Masters generate a START condition within the minimum hold time of the START condition, which results in a defined START condition to the bus.

Multi-Master mode support is achieved by bus arbitration. When the Master outputs address/data bits onto the SDA pin, arbitration takes place when the Master outputs a '1' on SDA by letting SDA float high and another Master asserts a '0'. When the SCL pin floats high, data should be stable. If the expected data on SDA is a '1' and the data sampled on the SDA pin = '0', then a bus collision has taken place. The Master will set the Bus Collision Interrupt Flag, BCLIF and reset the I^{2}C port to its IDLE state.

If a transmit was in progress when the bus collision occurred, the transmission is halted, the BF flag is cleared, the SDA and SCL lines are de-asserted, and the SSPBUF can be written to. When the user services the bus collision interrupt service routine, and if the I^{2}C bus is free, the user can resume communication by asserting a START condition.

If a START, Repeated Start, STOP, or Acknowledge condition was in progress when the bus collision occurred, the condition is aborted, the SDA and SCL lines are de-asserted, and the respective control bits in the SSPCON2 register are cleared. When the user services the bus collision interrupt service routine, and if the I^{2}C bus is free, the user can resume communication by asserting a START condition.

The Master will continue to monitor the SDA and SCL pins, and if a STOP condition occurs, the SSPIF bit will be set.

In Multi-Master mode, and when the MSSP is configured as a Slave, the interrupt generation on the detection of START and STOP conditions allows the determination of when the bus is free. Control of the I^{2}C bus can be taken when the P bit is set in the SSPSTAT register, or the bus is idle and the S and P bits are cleared.

When the MSSP is configured as a Master and it loses arbitration during the addressing sequence, it's possible that the winning Master is trying to address it. The losing Master must, therefore, switch over immediately to its Slave mode. While the MSSP module found on the PICmicro MCU does support Master I^{2}C, if it is the Master which lost arbitration and is also being addressed, the winning Master must restart the communication cycle over with a START followed by the device address. The MSSP Master I^{2}C mode implementation does not clock in the data placed on the bus during Multi-Master arbitration.

GENERAL CALL ADDRESS SUPPORT

The MSSP module supports the general call address mode when configured as a Slave (See Figure 4 below). The addressing procedure for the I^{2}C bus is such, that the first byte after the START condition usually determines which device will be the Slave addressed by the Master. The exception is the general call address, which can address all devices. When this address is used, all devices should, in theory, respond with an Acknowledge.

General call support can be useful if the Master wants to synchronize all Slaves, or wants to broadcast a message to all Slaves

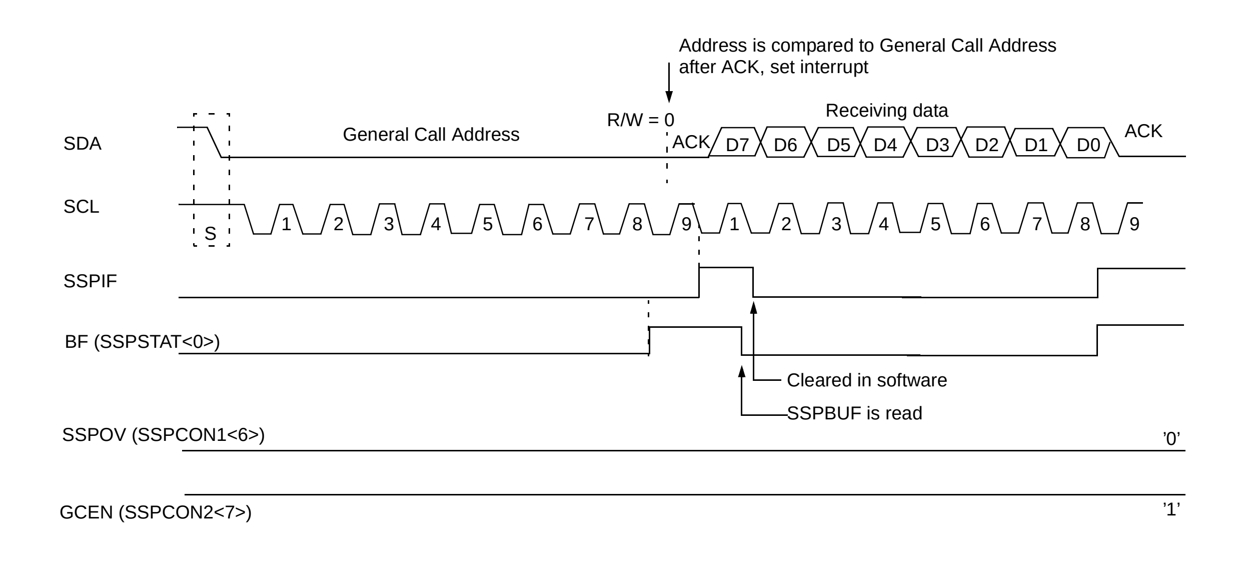
The general call address is one of eight addresses reserved for specific purposes by the I^{2}C protocol. It consists of all 0's with R/$\overline{W}$ = 0. The general call address is recognized when the General Call Enable bit (GCEN) is enabled (SSPCON2<7> set). Following a START bit detect, 8-bits are shifted into SSPSR and the address is compared against SSPADD, and is also compared to the general call address fixed in hardware.

If the general call address matches, the SSPSR is transferred to the SSPBUF, the BF flag bit is set (eighth bit) and on the falling edge of the ninth bit ($\overline{ACK}$ bit), the SSPIF interrupt flag bit is set.

When the interrupt is serviced, the source for the interrupt can be checked by reading the contents of the SSPBUF to determine if the address was device specific, or a general call address.

In 10-bit mode, the SSPADD is required to be updated for the second half of the address to match, and the UA bit is set (SSPSTAT<1>). If the general call address is sampled when the GCEN bit is set while the Slave is configured in 10-bit address mode, then the second half of the address is not necessary, the UA bit will not be set, and the Slave will begin receiving data.

FIGURE 4: SLAVE MODE GENERAL CALL ADDRESS SEQUENCE (7 OR 10-BIT ADDRESS MODE)



.

When the MSSP module is configured as a Master I^2C device, the operational characteristics of the SDA and SCL pins should be known. Table 1 below provides a summation of these pin characteristics.

TABLE 1: PICMICRO DEVICES WITH MSSP MODULE

Device	I^2C Pin Characteristics			
	Slew Rate Control[1]	Glitch Filter[1] on Inputs	Open Drain Pin Driver[2,3]	SMbus Compatible Input Levels[4]
PIC16C717	Yes	Yes	No	No
PIC16C770	Yes	Yes	No	No
PIC16C771	Yes	Yes	No	No
PIC16C773	Yes	Yes	No	No
PIC16C774	Yes	Yes	No	No
PIC16F872	Yes	Yes	No	Yes
PIC16F873	Yes	Yes	No	Yes
PIC16F874	Yes	Yes	No	Yes
PIC16F876	Yes	Yes	No	Yes
PIC16F877	Yes	Yes	No	Yes
PIC17C752	Yes	Yes	Yes	No
PIC17C756A	Yes	Yes	Yes	No
PIC17C762	Yes	Yes	Yes	No
PIC17C766	Yes	Yes	Yes	No
PIC18C242	Yes	Yes	No	No
PIC18C252	Yes	Yes	No	No
PIC18C442	Yes	Yes	No	No
PIC18C452	Yes	Yes	No	No

Note 1: A "glitch" filter is on the SCL and SDA pins when the pin is an input. The filter operates in both the 100 kHz and 400 kHz modes. In the 100 kHz mode, when these pins are an output, there is a slew rate control of the pin that is independent of device frequency

2: P-Channel driver disabled for PIC16C/FXXX and PIC18CXXX devices.

3: ESD/EOS protection diode to V_{DD} rail on PIC16C/FXXX and PIC18CXXX devices.

4: SMbus input levels are not available on all PICmicro devices. Consult the respective data sheet for electrical specifications.

WHAT'S IN THE APPENDIX

Example assembly source code for the Master I^2C device is included in Appendix A. Table 2 lists the source code files and provides a brief functional description .The code is developed for and tested on a PIC16F873 but can be ported over to a PIC17CXXX and PIC18CXXX PICmicro MCU which features a MSSP module.

TABLE 2: SOURCE CODE FILES

File Name	Description
mastri2c.asm	Main code loop and interrupt control functions.
mastri2c.inc	Variable declarations & definitions.
i2ccomm1.inc	Reference linkage for variables utilized in i2ccomm.asm file.
i2ccomm.asm	Routines for communicating with the I^2C Slave device.
i2ccomm.inc	Variable declarations & definitions.
flags.inc	Common flag definitions utilized within the mastri2c.asm and i2ccomm.asm files.
init.asm	Routines for initializing the PICmicro peripherals and ports.
p16f873.inc	PICmicro SFR definition file.
16f873.lkr	Modified linker script file.

Note: The PICmicro MCU based source files were developed and tested with the following Microchip tools:

- MPLAB® version 5.11.00
- MPASM version 2.50.00
- MPLINK version 2.10.00

SUMMARY

The Master Synchronous Serial Port (MSSP) embedded on many of the PICmicro devices, provides for both the 4-mode SPI communications as well as Master and Slave I^2C communications in hardware. Hardware peripheral support removes the code overhead of generating I^2C based communications in the application firmware. Interrupt support of the hardware peripheral also allows for timely and efficient task management.

This application note has presented some key operational basics on the MSSP module which should aid the developer in the understanding and implementation of the MSSP module for I^2C based communications.

Note: Information contained in this application note, regarding device applications and the like, is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated, with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use, or otherwise.

REFERENCES

The I^2C – Bus Specification, Philips Semiconductor, Version 2.1, http://www-us.semiconductors.com/i2c/

AN736, An I^2C Network Protocol for Environmental Monitoring, Microchip Technology Inc., Document # DS00736

AN734, Using the PICmicro SSP Module for Slave I^2C Communications, Microchip Technology Inc., Document # DS00734

PICmicro™ Mid-Range MCU Reference Manual, Microchip Technology Inc., Document # DS33023

PIC16C717/770/771 Data Sheet, Microchip Technology Inc., Document # DS41120

PIC16F87X Data Sheet, Microchip Technology Inc., Document # DS30292

APPENDIX A: I^{2}C MASTER READ AND WRITE ROUTINES (ASSEMBLY)

```
;*********************************************************************
;                                                                    *
;     Implementing Master I2C with the MSSP module on a PICmicro    *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Filename:       mastri2c.asm                                     *
;   Date:           07/18/2000                                       *
;   Revision:       1.00                                             *
;                                                                    *
;   Tools:          MPLAB   5.11.00                                  *
;                   MPLINK  2.10.00                                  *
;                   MPASM   2.50.00                                  *
;                                                                    *
;  Author:          Richard L. Fischer                               *
;                                                                    *
;  Company:         Microchip Technology Incorporated                *
;                                                                    *
;*********************************************************************
;                                                                    *
;   System files required:                                           *
;                                                                    *
;                   mastri2c.asm                                     *
;                   i2ccomm.asm                                      *
;                   init.asm                                         *
;                                                                    *
;                   mastri2c.inc                                     *
;                   i2ccomm.inc                                      *
;                   i2ccomm1.inc                                     *
;                   flags.inc                                        *
;                                                                    *
;                   p16f873.inc                                      *
;                   16f873.lkr     (modified for interrupts)         *
;                                                                    *
```

```
;**********************************************************************
;                                                                     *
;   Notes:                                                            *
;                                                                     *
;   Device Fosc -> 8.00MHz                                            *
;   WDT -> on                                                         *
;   Brownout -> on                                                    *
;   Powerup timer -> on                                               *
;   Code Protect -> off                                               *
;                                                                     *
;   Interrupt sources -                                               *
;                       1. I2C events (valid events)                  *
;                       2. I2C Bus Collision                          *
;                       3. Timer1 - 100mS intervals                   *
;                                                                     *
;                                                                     *
**********************************************************************/
   list      p=16f873                    ; list directive to define processor
   #include <p16f873.inc>                ; processor specific variable definitions
    __CONFIG (_CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _HS_OSC & _WRT_ENABLE_ON
            & _LVP_OFF & _CPD_OFF)

   #include  "mastri2c.inc"             ; required include file
   #include  "i2ccomm1.inc"             ; required include file
   errorlevel -302                      ; suppress bank warning

 #define  ADDRESS     0x01              ; Slave I2C address



;-----------------------------------------------------------------------
;   ********************* RESET VECTOR LOCATION  ********************
;-----------------------------------------------------------------------
RESET_VECTOR  CODE    0x000         ; processor reset vector
   movlw  high  start               ; load upper byte of 'start' label
   movwf  PCLATH                    ; initialize PCLATH
   goto   start                     ; go to beginning of program

;-----------------------------------------------------------------------
;  ******************* INTERRUPT VECTOR LOCATION  *******************
;-----------------------------------------------------------------------
INT_VECTOR   CODE    0x004          ; interrupt vector location
   movwf   w_temp                   ; save off current W register contents
   movf    STATUS,w                 ; move status register into W register
   clrf    STATUS                   ; ensure file register bank set to 0
   movwf   status_temp              ; save off contents of STATUS register
   movf    PCLATH,w
   movwf   pclath_temp              ; save off current copy of PCLATH
   clrf    PCLATH                   ; reset PCLATH to page 0

; TEST FOR COMPLETION OF VALID I2C EVENT
   bsf     STATUS,RP0               ; select SFR bank
   btfss   PIE1,SSPIE               ; test is interrupt is enabled
   goto    test_buscoll             ; no, so test for Bus Collision Int
   bcf     STATUS,RP0               ; select SFR bank
   btfss   PIR1,SSPIF               ; test for SSP H/W flag
   goto    test_buscoll             ; no, so test for Bus Collision Int
   bcf     PIR1,SSPIF               ; clear SSP H/W flag

   pagesel service_i2c              ; select page bits for function
   call    service_i2c              ; service valid I2C event

; TEST FOR I2C BUS COLLISION EVENT
test_buscoll
   banksel PIE2                     ; select SFR bank
   btfss   PIE2,BCLIE               ; test if interrupt is enabled
   goto    test_timer1              ; no, so test for Timer1 interrupt
   bcf     STATUS,RP0               ; select SFR bank
   btfss   PIR2,BCLIF               ; test if Bus Collision occured
   goto    test_timer1              ; no, so test for Timer1 interrupt
   bcf     PIR2,BCLIF               ; clear Bus Collision H/W flag
   call    service_buscoll          ; service bus collision error

; TEST FOR TIMER1 ROLLOVER EVENT
test_timer1
   banksel PIE1                     ; select SFR bank
   btfss   PIE1,TMR1IE              ; test if interrupt is enabled
   goto    exit_isr                 ; no, so exit ISR
   bcf     STATUS,RP0               ; select SFR bank
   btfss   PIR1,TMR1IF              ; test if Timer1 rollover occured
   goto    exit_isr                 ; no so exit isr
   bcf     PIR1,TMR1IF              ; clear Timer1 H/W flag

   pagesel service_i2c              ; select page bits for function
   call    service_i2c              ; service valid I2C event
   banksel T1CON                    ; select SFR bank
   bcf     T1CON,TMR1ON             ; turn off Timer1 module
   movlw   0x5E                     ;
   addwf   TMR1L,f                  ; reload Timer1 low
   movlw   0x98                     ;
   movwf   TMR1H                    ; reload Timer1 high
   banksel PIE1                     ; select SFR bank
   bcf     PIE1,TMR1IE              ; disable Timer1 interrupt
   bsf     PIE1,SSPIE               ; enable SSP H/W interrupt

exit_isr
   clrf    STATUS                   ; ensure file register bank set to 0
   movf    pclath_temp,w
   movwf   PCLATH                   ; restore PCLATH
   movf    status_temp,w            ; retrieve copy of STATUS register
   movwf   STATUS                   ; restore pre-isr STATUS register contents
   swapf   w_temp,f                 ;
   swapf   w_temp,w                 ; restore pre-isr W register contents
   retfie                           ; return from interrupt


;---------------------------------------------------------------------
;   ******************* MAIN CODE START LOCATION  ******************
;---------------------------------------------------------------------
MAIN    CODE
start
   pagesel init_ports
   call    init_ports               ; initialize Ports
   call    init_timer1              ; initialize Timer1
   pagesel init_i2c
   call    init_i2c                 ; initialize I2C module

   banksel eflag_event              ; select GPR bank
   clrf    eflag_event              ; initialize event flag variable
   clrf    sflag_event              ; initialize event flag variable
   clrf    i2cState

   call    CopyRom2Ram              ; copy ROM string to RAM
   call    init_vars                ; initialize variables

   banksel PIE2                     ; select SFR bank
   bsf     PIE2,BCLIE               ; enable interrupt
   banksel PIE1                     ; select SFR bank
   bsf     PIE1,TMR1IE              ; enable Timer1 interrupt
   bsf     INTCON,PEIE              ; enable peripheral interrupt
   bsf     INTCON,GIE               ; enable global interrupt
```

```
;*******************************************************************
;                     MAIN LOOP BEGINS HERE
;*******************************************************************
main_loop
   clrwdt                           ; reset WDT

   banksel eflag_event              ; select SFR bank
   btfsc   eflag_event,ack_error    ; test for ack error event flag
   call    service_ackerror         ; service ack error

   banksel sflag_event              ; select SFR bank
   btfss   sflag_event,rw_done      ; test if read/write cycle complete
   goto    main_loop                ; goto main loop
   call    string_compare           ; else, go compare strings

   banksel T1CON                    ; select SFR bank
   bsf     T1CON,TMR1ON             ; turn on Timer1 module
   banksel PIE1                     ; select SFR bank
   bsf     PIE1,TMR1IE              ; re-enable Timer1 interrupts

   call    init_vars                ; re-initialize variables
   goto    main_loop                ; goto main loop


;----------------------------------------------------------------------
;   *************** Bus Collision Service Routine ******************
;----------------------------------------------------------------------
service_buscoll
   banksel i2cState                 ; select GPR bank
   clrf    i2cState                 ; reset I2C bus state variable
   call    init_vars                ; re-initialize variables
   bsf     T1CON,TMR1ON             ; turn on Timer1 module
   banksel PIE1                     ; select SFR bank
   bsf     PIE1,TMR1IE              ; enable Timer1 interrupt
   return


;----------------------------------------------------------------------
;   ************* Acknowledge Error Service Routine ***************
;----------------------------------------------------------------------
service_ackerror
   banksel eflag_event              ; select SFR bank
   bcf     eflag_event,ack_error    ; reset acknowledge error event flag
   clrf    i2cState                 ; reset bus state variable
   call    init_vars                 ; re-initialize variables
   bsf     T1CON,TMR1ON              ; turn on Timer1 module
   banksel PIE1                      ; select SFR bank
   bsf     PIE1,TMR1IE               ; enable Timer1 interrupt
   return



;----------------------------------------------------------------------
;   *****  INITIALIZE VARIABLES USED IN SERVICE_I2C FUNCTION  ******
;----------------------------------------------------------------------
init_vars
   movlw   D’21’                     ; byte count for this example
   banksel write_count               ; select GPR bank
   movwf   write_count               ; initialize write count
   movwf   read_count                ; initialize read count

   movlw   write_string              ; get write string array address
   movwf   write_ptr                 ; initialize write pointer
   movlw   read_string               ; get read string placement address
   movwf   read_ptr                  ; initialize read pointer

   movlw   ADDRESS                   ; get address of slave
   movwf   temp_address              ; initialize temporary address hold reg
   return



;----------------------------------------------------------------------
;  *******************  Compare Strings   ************************
;----------------------------------------------------------------------
;Compare the string written to and read back from the Slave
string_compare
   movlw   read_string
   banksel ptr1                      ; select GPR bank
   movwf   ptr1                      ; initialize first pointer
   movlw   write_string
   movwf   ptr2                      ; initialize second pointer

loop
   movf    ptr1,w                    ; get address of first pointer
   movwf   FSR                       ; init FSR
   movf    INDF,w                    ; retrieve one byte
   banksel temp_hold                 ; select GPR bank
   movwf   temp_hold                 ; save off byte 1
   movf    ptr2,w
   movwf   FSR                      ; init FSR
   movf    INDF,w                   ; retrieve second byte
   subwf   temp_hold,f              ; do comparison
   btfss   STATUS,Z                 ; test for valid compare
   goto    not_equal                ; bytes not equal
   iorlw   0x00                     ; test for null character
   btfsc   STATUS,Z
   goto    end_string               ; end of string has been reached
   incf    ptr1,f                   ; update first pointer
   incf    ptr2,f                   ; update second pointer
   goto    loop                     ; do more comparisons

not_equal
   banksel PORTB                    ; select GPR bank
   movlw   b’00000001’
   xorwf   PORTB,f
   goto    exit
end_string
   banksel  PORTB                    ; select GPR bank
   movlw    b’00000010’              ; no error
   xorwf    PORTB,f
exit
   banksel  sflag_event              ; select SFR bank
   bcf      sflag_event,rw_done      ; reset flag
   return



;----------------------------------------------------------------------
;  *******************  Program Memory Read   *******************
;----------------------------------------------------------------------
;Read the message from location MessageTable
CopyRom2Ram
   movlw   write_string
   movwf   FSR                      ; initialize FSR

   banksel EEADRH                   ; select SFR bank
   movlw   High (Message1)          ; point to the Message Table
   movwf   EEADRH                   ; init SFR EEADRH
   movlw   Low (Message1)
   movwf   EEADR                    ; init SFR EEADR

next1
   banksel EECON1                   ; select SFR bank
   bsf     EECON1,EEPGD             ; select the program memory
   bsf     EECON1,RD                  ; read word
   nop
   nop
   banksel EEDATA
   rlf     EEDATA,w                   ; get bit 7 in carry
   rlf     EEDATH,w                   ; get high byte in w

   movwf   INDF                       ; save it
   incf    FSR,f

   banksel EEDATA                     ; select SFR bank
   bcf     EEDATA,7                   ; clr bit 7
   movf    EEDATA,w                   ; get low byte and see = 0?
   btfsc   STATUS,Z                   ; end?
   return
   movwf   INDF                       ; save it
   incf    FSR,f                      ; update FSR pointer
   banksel EEADR                      ; point to address
   incf    EEADR,f                    ; inc to next location
   btfsc   STATUS,Z                   ; cross over 0xff
   incf    EEADRH,f                   ; yes then inc high
   goto    next1                      ; read next byte


;----------------------------------------------------------------------
;----------------------------------------------------------------------

Message1   DA      "Master and Slave I2C",0x00,0x00

   END                                ; required directive
```

```
;*********************************************************************
;                                                                    *
;     Implementing Master I2C with the MSSP module on a PICmicro     *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Filename:       i2ccomm.asm                                      *
;   Date:           07/18/2000                                       *
;   Revision:       1.00                                             *
;                                                                    *
;   Tools:          MPLAB   5.11.00                                  *
;                   MPLINK  2.10.00                                  *
;                   MPASM   2.50.00                                  *
;                                                                    *
;  Author:          Richard L. Fischer                               *
;                   John E. Andrews                                  *
;                                                                    *
;  Company:         Microchip Technology Incorporated                *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Files required:                                                  *
;                                                                    *
;                   i2ccomm.asm                                      *
;                                                                    *
;                   i2ccomm.inc                                      *
;                   flags.inc    (referenced in i2ccomm.inc file)    *
;                   i2ccomm1.inc (must be included in main file)     *
;                   p16f873.inc                                      *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Notes:   The routines within this file are used to read from     *
;   and write to a Slave I2C device. The MSSP initialization         *
;   function is also contained within this file.                     *
;                                                                    *
;********************************************************************/

   #include <p16f873.inc>               ; processor specific definitions
   #include  "i2ccomm.inc"              ; required include file
   errorlevel -302                      ; suppress bank warning


 #define  FOSC        D’8000000’         ; define FOSC to PICmicro
 #define  I2CClock    D'400000'          ; define I2C bite rate
 #define  ClockValue  (((FOSC/I2CClock)/4) -1) ;



;-------------------------------------------------------------------
;   ***********************  I2C Service  ************************
;-------------------------------------------------------------------
I2C_COMM   CODE
service_i2c

   movlw   high  I2CJump           ; fetch upper byte of jump table address
   movwf   PCLATH                  ; load into upper PC latch
   movlw   i2cSizeMask
   banksel i2cState                ; select GPR bank
   andwf   i2cState,w              ; retrieve current I2C state
   addlw   low  (I2CJump + 1)      ; calc state machine jump addr into W
   btfsc   STATUS,C                ; skip if carry occured
   incf    PCLATH,f                ; otherwise add carry
I2CJump                                ; address were jump table branch occurs,
                                   ;  this addr also used in fill
   movwf   PCL                     ; index into state machine jump table

; jump to processing for each state = i2cState value

   goto    WrtStart                ; write start sequence           =  0
   goto    SendWrtAddr             ; write address, R/W=1           =  1
   goto    WrtAckTest              ; test acknowledge after address =  2
   goto    WrtStop                 ; generate stop sequence         =  3

   goto    ReadStart               ; write start sequence           =  4
   goto    SendReadAddr            ; write address, R/W=0           =  5
   goto    ReadAckTest             ; test acknowledge after address =  6
   goto    ReadData                ; read more data                 =  7
   goto    ReadStop                ; generate stop sequence         =  8

I2CJumpEnd
        Fill (return),  (I2CJump-I2CJumpEnd) + i2cSizeMask
```

```
;----------------------------------------------------------------------
;   ********************* Write data to Slave   *********************
;----------------------------------------------------------------------
; Generate I2C bus start condition               [ I2C STATE -> 0 ]
WrtStart
   banksel  write_ptr                ; select GPR bank
   movf     write_ptr,w              ; retrieve ptr address
   movwf    FSR                      ; initialize FSR for indirect access
   incf     i2cState,f               ; update I2C state variable
   banksel  SSPCON2                  ; select SFR bank
   bsf      SSPCON2,SEN             ; initiate I2C bus start condition
   return

; Generate I2C address write (R/W=0)             [ I2C STATE -> 1 ]
SendWrtAddr
   banksel temp_address             ; select GPR bank
   bcf     STATUS,C                 ; ensure carry bit is clear
   rlf     temp_address,w           ; compose 7-bit address
   incf    i2cState,f               ; update I2C state variable
   banksel SSPBUF                   ; select SFR bank
   movwf   SSPBUF                   ; initiate I2C bus write condition
   return

; Test acknowledge after address and data write  [ I2C STATE -> 2 ]
WrtAckTest
   banksel SSPCON2                  ; select SFR bank
   btfss   SSPCON2,ACKSTAT          ; test for acknowledge from slave
   goto    WrtData                  ; go to write data module
   banksel eflag_event              ; select GPR bank
   bsf     eflag_event,ack_error    ; set acknowledge error
   clrf    i2cState                 ; reset I2C state variable
   banksel SSPCON2                  ; select SFR bank
   bsf     SSPCON2,PEN              ; initiate I2C bus stop condition
   return

; Generate I2C write data condition
WrtData
   movf    INDF,w                   ; retrieve byte into w
   banksel write_count              ; select GPR bank
   decfsz  write_count,f            ; test if all done with writes
   goto    send_byte                ; not end of string
   incf    i2cState,f               ; update I2C state variable
send_byte
   banksel SSPBUF                   ; select SFR bank
   movwf   SSPBUF                   ; initiate I2C bus write condition
   incf    FSR,f                    ; increment pointer
   return

; Generate I2C bus stop condition                [ I2C STATE -> 3 ]
WrtStop
   banksel SSPCON2                  ; select SFR bank
   btfss   SSPCON2,ACKSTAT          ; test for acknowledge from slave
   goto    no_error                 ; bypass setting error flag
   banksel eflag_event              ; select GPR bank
   bsf     eflag_event,ack_error    ; set acknowledge error
   clrf    i2cState                 ; reset I2C state variable
   goto    stop
no_error
   banksel i2cState                 ; select GPR bank
   incf    i2cState,f               ; update I2C state variable for read
stop
   banksel SSPCON2                  ; select SFR bank
   bsf     SSPCON2,PEN              ; initiate I2C bus stop condition
   return



;----------------------------------------------------------------------
;   ********************* Read data from Slave   *********************
;----------------------------------------------------------------------
; Generate I2C start condition                   [ I2C STATE -> 4 ]
ReadStart
   banksel  read_ptr                 ; select GPR bank
   movf     read_ptr,W               ; retrieve ptr address
   movwf    FSR                      ; initialize FSR for indirect access
   incf     i2cState,f               ; update I2C state variable
   banksel  SSPCON2                  ; select SFR bank
   bsf      SSPCON2,SEN              ; initiate I2C bus start condition
   return

; Generate I2C address write (R/W=1)             [ I2C STATE -> 5 ]
SendReadAddr
   banksel temp_address             ; select GPR bank
   bsf     STATUS,C                 ; ensure cary bit is clear
   rlf     temp_address,w           ; compose 7 bit address
   incf    i2cState,f               ; update I2C state variable
   banksel SSPBUF                   ; select SFR bank
   movwf   SSPBUF                   ; initiate I2C bus write condition
   return
```

```
; Test acknowledge after address write           [ I2C STATE -> 6 ]
ReadAckTest
   banksel SSPCON2                  ; select SFR bank
   btfss   SSPCON2,ACKSTAT          ; test for not acknowledge from slave
   goto    StartReadData            ; good ack, go issue bus read
   banksel eflag_event              ; ack error, so select GPR bank
   bsf     eflag_event,ack_error    ; set ack error flag
   clrf    i2cState                 ; reset I2C state variable
   banksel SSPCON2                  ; select SFR bank
   bsf     SSPCON2,PEN              ; initiate I2C bus stop condition
   return

StartReadData
   bsf     SSPCON2,RCEN             ; generate receive condition
   banksel i2cState                 ; select GPR bank
   incf    i2cState,f               ; update I2C state variable
   return

; Read slave I2C                                 [ I2C STATE -> 7 ]
ReadData
   banksel SSPBUF                   ; select SFR bank
   movf    SSPBUF,w                 ; save off byte into W
   banksel read_count               ; select GPR bank
   decfsz  read_count,f             ; test if all done with reads
   goto    SendReadAck              ; not end of string so send ACK

; Send Not Acknowledge
SendReadNack
   movwf   INDF                     ; save off null character
   incf    i2cState,f               ; update I2C state variable
   banksel SSPCON2                  ; select SFR bank
   bsf     SSPCON2,ACKDT            ; acknowledge bit state to send (not ack)
   bsf     SSPCON2,ACKEN            ; initiate acknowledge sequence
   return

; Send Acknowledge
SendReadAck
   movwf   INDF                     ; no, save off byte
   incf    FSR,f                    ; update receive pointer
   banksel SSPCON2                  ; select SFR bank
   bcf     SSPCON2,ACKDT            ; acknowledge bit state to send
   bsf     SSPCON2,ACKEN            ; initiate acknowledge sequence
   btfsc   SSPCON2,ACKEN            ; ack cycle complete?
   goto    $-1                      ; no, so loop again
   bsf     SSPCON2,RCEN             ; generate receive condition
   return

; Generate I2C stop condition                    [ I2C STATE -> 8 ]
ReadStop
   banksel SSPCON2                  ; select SFR bank
   bcf     PIE1,SSPIE               ; disable SSP interrupt
   bsf     SSPCON2,PEN              ; initiate I2C bus stop condition
   banksel i2cState                 ; select GPR bank
   clrf    i2cState                 ; reset I2C state variable
   bsf   sflag_event,rw_done        ; set read/write done flag
   return



;----------------------------------------------------------------------
;   ******************* Generic bus idle check ***********************
;----------------------------------------------------------------------
; test for i2c bus idle state; not implemented in this code (example only)
i2c_idle
   banksel SSPSTAT                  ; select SFR bank
   btfsc   SSPSTAT,R_W              ; test if transmit is progress
   goto    $-1                      ; module busy so wait
   banksel SSPCON2                  ; select SFR bank
   movf    SSPCON2,w                ; get copy of SSPCON2 for status bits
   andlw   0x1F                     ; mask out non-status bits
   btfss   STATUS,Z                      ; test for zero state, if Z set, bus is idle
   goto    $-3                      ; bus is busy so test again
   return                           ; return to calling routine



;----------------------------------------------------------------------
;  ******************* INITIALIZE MSSP MODULE  *******************
;----------------------------------------------------------------------

init_i2c
   banksel SSPADD                   ; select SFR bank
   movlw   ClockValue               ; read selected baud rate
   movwf   SSPADD                   ; initialize I2C baud rate
   bcf     SSPSTAT,6                ; select I2C input levels
   bcf     SSPSTAT,7                ; enable slew rate

   movlw   b’00011000’
   iorwf   TRISC,f                  ; ensure SDA and SCL are inputs
   bcf     STATUS,RP0               ; select SFR bank
```

```
movlw   b'00111000'
movwf   SSPCON                   ; Master mode, SSP enable
return                           ; return from subroutine


END                            ; required directive
```

```
;*******************************************************************
;                                                                  *
;     Implementing Master I2C with the MSSP module on a PICmicro    *
;                                                                  *
;*******************************************************************
;                                                                  *
;   Filename:      init.asm                                        *
;   Date:          07/18/2000                                      *
;   Revision:      1.00                                            *
;                                                                  *
;   Tools:         MPLAB   5.11.00                                 *
;                  MPLINK  2.10.00                                 *
;                  MPASM   2.50.00                                 *
;                                                                  *
;  Author:         Richard L. Fischer                              *
;                                                                  *
;  Company:        Microchip Technology Incorporated               *
;                                                                  *
;*******************************************************************
;                                                                  *
;   Files required:                                                *
;                                                                  *
;                   init.asm                                       *
;                                                                  *
;                   p16f873.inc                                    *
;                                                                  *
;                                                                  *
;*******************************************************************
;                                                                  *
;   Notes:                                                         *
;                                                                  *
;                                                                  *
;******************************************************************/

   #include <p16f873.inc>         ; processor specific variable definitions
   errorlevel -302                ; suppress bank warning



   GLOBAL  init_timer1            ; make function viewable for other modules
   GLOBAL  init_ports             ; make function viewable for other modules
```

```
;----------------------------------------------------------------------
;  ******************* INITIALIZE PORTS  *************************
;----------------------------------------------------------------------
INIT_CODE   CODE

init_ports
   banksel PORTA                  ; select SFR bank
   clrf    PORTA                  ; initialize PORTS
   clrf    PORTB
   clrf    PORTC

   bsf     STATUS,RP0             ; select SFR bank
   movlw   b’00000110’
   movwf   ADCON1                 ; make PORTA digital
   clrf    TRISB
   movlw   b’000000’
   movwf   TRISA
   movlw   b’00011000’
   movwf   TRISC
   return


;----------------------------------------------------------------------
;  ******************* INITIALIZE TIMER1 MODULE  *******************
;----------------------------------------------------------------------
init_timer1
   banksel T1CON                  ; select SFR bank
   movlw   b’00110000’            ; 1:8 prescale, 100mS rollover
   movwf   T1CON                  ; initialize Timer1

   movlw   0x5E
   movwf   TMR1L                  ; initialize Timer1 low
   movlw   0x98
   movwf   TMR1H                  ; initialize Timer1 high

   bcf     PIR1,TMR1IF            ; ensure flag is reset
   bsf     T1CON,TMR1ON           ; turn on Timer1 module
   return

   END                            ; required directive
```

```
;*********************************************************************
;                                                                    *
;   Filename:       mastri2c.inc                                     *
;   Date:           07/18/2000                                       *
;   Revision:       1.00                                             *
;                                                                    *
;   Tools:          MPLAB   5.11.00                                  *
;                   MPLINK  2.10.00                                  *
;                   MPASM   2.50.00                                  *
;                                                                    *
;*********************************************************************



;*******    INTERRUPT CONTEXT SAVE/RESTORE VARIABLES
INT_VAR        UDATA   0x20              ; create uninitialized data "udata" section
w_temp           RES     1
status_temp      RES     1
pclath_temp      RES     1


INT_VAR1       UDATA   0xA0             ; reserve location 0xA0
w_temp1          RES     1


;*******    GENERAL PURPOSE VARIABLES
GPR_DATA         UDATA
temp_hold        RES     1              ; temp variable for string compare
ptr1             RES     1              ; used as pointer in string compare
ptr2             RES     1              ; used as pointer in string compare


STRING_DATA      UDATA
write_string     RES      D’30’
read_string      RES      D’30’



   EXTERN   init_timer1                 ; reference linkage for function
   EXTERN   init_ports                  ; reference linkage for function
```

```
;*********************************************************************
;                                                                    *
;   Filename:       i2ccomm1.inc                                     *
;   Date:           07/18/2000                                       *
;   Revision:       1.00                                             *
;                                                                    *
;   Tools:          MPLAB   5.11.00                                  *
;                   MPLINK  2.10.00                                  *
;                   MPASM   2.50.00                                  *
;                                                                    *
;*********************************************************************
;                                                                    *
;   Notes:                                                           *
;                                                                    *
;   This file is to be included in the <main.asm> file. The          *
;   <main.asm> notation represents the file which has the            *
;   subroutine calls for the functions ’service_i2c’ and ’init_i2c’. *
;                                                                    *
;                                                                    *
;********************************************************************/

   #include  "flags.inc"         ; required include file



   GLOBAL   write_string         ; make variable viewable for other modules
   GLOBAL   read_string          ; make variable viewable for other modules

   EXTERN   sflag_event          ; reference linkage for variable
   EXTERN   eflag_event          ; reference linkage for variable
   EXTERN   i2cState             ; reference linkage for variable
   EXTERN   read_count           ; reference linkage for variable
   EXTERN   write_count          ; reference linkage for variable
   EXTERN   write_ptr            ; reference linkage for variable
   EXTERN   read_ptr             ; reference linkage for variable
   EXTERN   temp_address         ; reference linkage for variable

   EXTERN   init_i2c             ; reference linkage for function
   EXTERN   service_i2c          ; reference linkage for function


;*********************************************************************
;                                                                    *
;   Additional notes on variable usage:                              *
;                                                                    *
;   The variables listed below are used within the function          *
;   service_i2c. These variables must be initialized with the         *
;   appropriate data from within the calling file. In this            *
;   application code the main file is 'mastri2c.asm'. This file       *
;   contains the function calls to service_i2c. It also contains      *
;   the function for initializing these variables, called 'init_vars'*
;                                                                     *
;   To use the service_i2c function to read from and write to an      *
;   I2C slave device, information is passed to this function via      *
;   the following variables.                                          *
;                                                                     *
;                                                                     *
;   The following variables are used as function parameters:          *
;                                                                     *
;   read_count   - Initialize this variable for the number of bytes  *
;                  to read from the slave I2C device.                 *
;   write_count  - Initialize this variable for the number of bytes  *
;                  to write to the slave I2C device.                  *
;   write_ptr    - Initialize this variable with the address of the  *
;                  data string or data byte to write to the slave     *
;                  I2C device.                                        *
;   read_ptr     - Initialize this variable with the address of the  *
;                  location for storing data read from the slave I2C *
;                  device.                                            *
;   temp_address - Initialize this variable with the address of the  *
;                  slave I2C device to communicate with.              *
;                                                                     *
;                                                                     *
;   The following variables are used as status or error events        *
;                                                                     *
;   sflag_event  - This variable is implemented for status or         *
;                  event flags. The flags are defined in the file     *
;                  'flags.inc'.                                       *
;   eflag_event  - This variable is implemented for error flags. The *
;                  flags are defined in the file 'flags.inc'.         *
;                                                                     *
;                                                                     *
;   The following variable is used in the state machine jumnp table. *
;                                                                     *
;   i2cState     - This variable holds the next I2C state to execute.*
;                                                                     *
;*********************************************************************
```

```
;*********************************************************************
;                                                                   *
;   Filename:       flags.inc                                       *
;   Date:           07/18/2000                                      *
;   Revision:       1.00                                            *
;                                                                   *
;   Tools:          MPLAB   5.11.00                                 *
;                   MPLINK  2.10.00                                 *
;                   MPASM   2.50.00                                 *
;                                                                   *
;*********************************************************************
;                                                                   *
;   Notes:                                                          *
;                                                                   *
;   This file defines the flags used in the i2ccomm.asm file.       *
;                                                                   *
;                                                                   *
;********************************************************************/



; bits for variable sflag_event
#define  sh1         0                   ; place holder
#define  sh2         1                   ; place holder
#define  sh3         2                   ; place holder
#define  sh4         3                   ; place holder
#define  sh5         4                   ; place holder
#define  sh6         5                   ; place holder
#define  sh7         6                   ; place holder
#define  rw_done     7                   ; flag bit


; bits for variable eflag_event
#define  ack_error   0                   ; flag bit
#define  eh1         1                   ; place holder
#define  eh2         2                   ; place holder
#define  eh3         3                   ; place holder
#define  eh4         4                   ; place holder
#define  eh5         5                   ; place holder
#define  eh6         6                   ; place holder
#define  eh7         7                   ; place holder
```

```
;**********************************************************************
;                                                                     *
;   Filename:       i2ccomm.inc                                       *
;   Date:           07/18/2000                                        *
;   Revision:       1.00                                              *
;                                                                     *
;   Tools:          MPLAB   5.11.00                                   *
;                   MPLINK  2.10.00                                   *
;                   MPASM   2.50.00                                   *
;                                                                     *
;**********************************************************************
;                                                                     *
;   Notes:                                                            *
;                                                                     *
;   This file is to be included in the i2ccomm.asm file               *
;                                                                     *
;                                                                     *
;*********************************************************************/

  #include  "flags.inc"            ; required include file


i2cSizeMask  EQU  0x0F


   GLOBAL   sflag_event          ; make variable viewable for other modules
   GLOBAL   eflag_event          ; make variable viewable for other modules
   GLOBAL   i2cState             ; make variable viewable for other modules
   GLOBAL   read_count           ; make variable viewable for other modules
   GLOBAL   write_count          ; make variable viewable for other modules
   GLOBAL   write_ptr            ; make variable viewable for other modules
   GLOBAL   read_ptr             ; make variable viewable for other modules
   GLOBAL   temp_address         ; make variable viewable for other modules

   GLOBAL   init_i2c             ; make function viewable for other modules
   GLOBAL   service_i2c          ; make function viewable for other modules



;*******    GENERAL PURPOSE VARIABLES
GPR_DATA   UDATA
sflag_event     RES     1        ; variable for i2c general status flags
eflag_event     RES     1        ; variable for i2c error status flags
i2cState        RES     1        ; I2C state machine variable
```

```
read_count      RES     1        ; variable used for slave read byte count
write_count     RES     1        ; variable used for slave write byte count
write_ptr       RES     1        ; variable used for pointer (writes to)
read_ptr        RES     1        ; variable used for pointer (reads from)
temp_address    RES     1        ; variable used for passing address to functions



;*********************************************************************
;                                                                    *
;   Additional notes on variable usage:                              *
;                                                                    *
;   The variables listed below are used within the function          *
;   service_i2c. These variables must be initialized with the        *
;   appropriate data from within the calling file. In this           *
;   application code the main file is 'mastri2c.asm'. This file      *
;   contains the function calls to service_i2c. It also contains     *
;   the function for initializing these variables, called 'init_vars'*
;                                                                    *
;   To use the service_i2c function to read from and write to an     *
;   I2C slave device, information is passed to this function via     *
;   the following variables.                                         *
;                                                                    *
;                                                                    *
;   The following variables are used as function parameters:         *
;                                                                    *
;   read_count   - Initialize this variable for the number of bytes  *
;                  to read from the slave I2C device.                *
;   write_count  - Initialize this variable for the number of bytes  *
;                  to write to the slave I2C device.                 *
;   write_ptr    - Initialize this variable with the address of the  *
;                  data string or data byte to write to the slave    *
;                  I2C device.                                       *
;   read_ptr     - Initialize this variable with the address of the  *
;                  location for storing data read from the slave I2C *
;                  device.                                           *
;   temp_address - Initialize this variable with the address of the  *
;                  slave I2C device to communicate with.             *
;                                                                    *
;                                                                    *
;   The following variables are used as status or error events       *
;                                                                    *
;   sflag_event  - This variable is implemented for status or        *
;                  event flags. The flags are defined in the file    *
;                  ’flags.inc’.                                      *
;   eflag_event  - This variable is implemented for error flags. The *
;                  flags are defined in the file ’flags.inc’.        *
;                                                                    *
;                                                                    *
;   The following variable is used in the state machine jumnp table. *
;                                                                    *
;   i2cState     - This variable holds the next I2C state to execute.*
;                                                                    *
;*********************************************************************
```

NOTES:

Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products 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 PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be 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 product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

Printed on recycled paper.

DNV Certification, Inc.
USA

ANSI★RAB
QMS
ACCREDITED

DNV

DNV MSC
The Netherlands
Accredited by the RvA

ISO 9001 / QS-9000
REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

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: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-7456

Atlanta

500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York

150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose

Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia

Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing

Microchip Technology Consulting (Shanghai)
Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu

Microchip Technology Consulting (Shanghai)
Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-6766200 Fax: 86-28-6766599

China - Fuzhou

Microchip Technology Consulting (Shanghai)
Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai

Microchip Technology Consulting (Shanghai)
Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen

Microchip Technology Consulting (Shanghai)
Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-2350361 Fax: 86-755-2366086

Hong Kong

Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India

Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan

Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea

Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore

Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan

Microchip Technology Taiwan
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark

Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France

Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany

Microchip Technology GmbH
Gustav-Heinemann Ring 125
D-81739 Munich, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy

Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom

Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

01/18/02