INTRODUCTION

Many devices in the PICmicro family have a Synchronous Serial Port (SSP) or Master Synchronous Serial Port (MSSP). These peripherals can be used to implement the SPI™ or I²C communication protocols. The purpose of this application note is to provide the reader with a better understanding of the I²C protocol and to show how PICmicro devices with the SSP or MSSP modules are used as a Slave device on an I²C bus.

For more information on the I²C bus specification, or the PICmicro SSP and MSSP peripherals, you may refer to sources indicated in the References section.

THE I²C BUS SPECIFICATION

Although a complete discussion of the I²C bus specification is outside the scope of this application note, some of the basics will be covered here. The Inter-Integrated-Circuit, or I²C 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 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 I²C protocol supports either a 7-bit addressing mode, or a 10-bit addressing mode, permitting up to 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.

All data transfers on the bus are initiated by the Master device, which always generates the clock signal on the bus. Data transfers are performed on the bus eight bits at a time, MSb first. There is no limit to the amount of data that can be sent in one transfer.

The I²C protocol includes a handshaking mechanism. 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.

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 pulls the SDA line low, while the SCL line is high. The START sequence tells all Slaves 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 I²C protocol also permits a Repeated Start condition (Rs), which allows the Master device on the bus to perform a START sequence, without a STOP sequence preceding it. The Repeated Start allows the Master device to start a new data transfer without releasing control of the bus.

A typical I²C write transmission would proceed as shown in Figure 1. In this example, the Master device will write two bytes to a Slave device. The transmission is started when the Master initiates a START condition on the bus. Next, the Master sends an address byte to the Slave. The upper seven bits of the address byte contain the Slave address. The LSB of the address byte specifies whether the I²C operation will be a read (LSb = 1), or a write (LSb = 0). On the ninth clock pulse, the Master releases the SDA line so the Slave can acknowledge the reception. If the address byte was received by the Slave and was the correct address, the Slave responds with an ACK by holding the SCL line low. By holding the SCL line, the Slave can take as much time as needed to prepare the data to be sent back to the Master. When the Slave is ready, it releases SCL and the Master device clocks the data from the Slave buffer.
On the ninth clock pulse, the Slave latches the value of the ACK bit received from the Master. If an ACK pulse was received, the Slave must prepare the next byte of data to be transmitted. If a NACK was received, the data transmission is complete. In this case, the Slave device should wait for the next START condition.

For many I²C peripherals, such as non-volatile EEPROM memory, an I²C write operation and a read operation are done in succession. For example, the write operation specifies the address to be read and the read operation gets the byte of data. Since the Master device does not release the bus after the memory address is written to the device, a Repeated Start sequence is performed to read the contents of the memory address.

THE PICmicro SSP MODULE

A block diagram of the SSP module for I²C Slave mode is shown in Figure 3. Key control and status bits required for I²C Slave communication are provided in the following special function registers:
- SSPSTAT
- SSPCON
- PIR1 (interrupt flag bit)
- PIE1 (interrupt enable bit)

Some of the bit functions in these registers vary, depending on whether the SSP module is used for I²C or SPI communications. The functionality of each of I²C mode is described here. For a complete description of each bit function, refer to the appropriate device data sheet.

FIGURE 1: TYPICAL I²C WRITE TRANSMISSION (7-BIT ADDRESS)

FIGURE 2: TYPICAL I²C READ TRANSMISSION (7-BIT ADDRESS) USING THE PICmicro SSP
**FIGURE 3: PICmicro SSP MODULE BLOCK DIAGRAM (I2C SLAVE MODE)**

**SSP Bits That Indicate Module Status**

**BF (SSPSTAT<0>)**
The BF (buffer full) bit tells the user whether a byte of data is currently in the SSP buffer register, SSPBUF. This bit is cleared automatically when the SSPBUF register is read, or when a byte to be transmitted is completely shifted out of the register. The BF bit will become set under the following circumstances:

- When an address byte is received with the LSb cleared. This will be the first byte sent by the Master device during an I^2^C write operation.
- Each time a data byte is received during an I^2^C write to the Slave device.
- Each time a byte of data is written to SSPBUF to be transmitted to the Master device. The BF bit will be cleared automatically when all bits have been shifted from SSPBUF to the Master device.

There is one condition for the SSP module when the BF bit does not become set as one might expect. This condition occurs when the address byte for an I^2^C read operation is received by the Slave (LSb = 1). For read operations, the BF bit indicates the status of data written to SSPBUF for transmission to the Master device.

**UA (SSPSTAT<1>)**
The UA (Update Address) bit is used only in the 10-bit address modes. In the 10-bit address mode, an I^2^C Slave address must be sent in two bytes. The upper half of the 10-bit address (1111 0 A9 A8 0) first loaded into SSPADD for initial match detection. This particular address code is reserved in the I^2^C protocol for designating the upper half of a 10-bit address. When an address match occurs, the SSP module will set the UA bit to indicate that the lower half of the address should be loaded into SSPADD for match detection.

**R/W (SSPSTAT<2>)**
The R/W (Read/Write) bit tells the user whether the Master device is reading from, or writing to, the Slave device. This bit reflects the state of the LSb in the address byte that is sent by the Master. The state of the R/W bit is only valid for the duration of a particular I^2^C message and will be reset by a STOP condition, START condition, or a NACK from the Master device.

**S (SSPSTAT<3>)**
The S (START) bit is set if a START condition occurred last on the bus. The state of this bit will be the inverse of the P (STOP) bit, except when the module is first initialized and both bits are cleared.

**P (SSPSTAT<4>)**
The P (STOP) bit is set if a STOP condition occurred last on the bus. The state of this bit will be the inverse of the S (START) bit, except when the module is first initialized and both bits are cleared. The P bit can be used to determine when the bus is idle.

**D/A (SSPSTAT<5>)**
The D/A (Data/Address) bit indicates whether the last byte of data received by the SSP module was a data byte or an address byte. For read operations, the last byte sent to the Master device was a data byte when the D/A bit is set.

**WCOL (SSPCON<7>)**
The WCOL (Write Collision) bit indicates whether SSPBUF was written while the previously written word is still transmitting. The previous contents of SSPBUF are not changed when the write collision occurs. The WCOL bit must be cleared in software.
SSPOV (SSPCON<6>)
The SSPOV (SSP overflow bit) indicates that a new byte was received while SSPBUF was still holding the previous data. In this case, the SSP module will not generate an ACK pulse and SSPBUF will not be updated with the new data. Regardless of whether the data is to be used, the user must read SSPBUF whenever the BF bit becomes set, to avoid a SSP overflow condition. The user must read SSPBUF and clear the SSPOV bit to properly clear an overflow condition. If the user reads SSPBUF to clear the BF bit, but does not clear the SSPOV bit, the next byte of data received will be loaded into SSPBUF but the module will not generate an ACK pulse.

SSPIF (PIR1<3>)
The SSPIF (SSP interrupt flag) bit indicates that an I2C event has completed. The user must poll the status bits described here to determine what event occurred and the next action to be taken. The SSPIF bit must be cleared by the user.

SSP Bits for Module Control
SSPEN (SSPCON<5>)
The SSPEN (SSP enable bit) enables the SSP module and configures the appropriate I/O pins as serial port pins.

CKE (SSPSTAT<6>)
The CKE (Clock edge) bit has no function when the SSP module is configured for I2C mode and should be cleared.

SMP (SSPSTAT<7>)
The SMP (Sample phase) bit has no function when the SSP module is configured for I2C mode and should be cleared.

CKP (SSPCON<4>)
The CKP (Clock polarity) bit is used for clock stretching in the I2C protocol. When the CKP bit is cleared, the Slave device holds the SCL pin low so that the Master device on the bus is unable to send clock pulses. During clock stretching, the Master device will attempt to send clock pulses until the clock line is released by the Slave device.

Clock stretching is useful when the Slave device can not process incoming bytes quickly enough, or when SSPBUF needs to be loaded with data to be transmitted to the Master device. The SSP module performs clock stretching automatically when data is read by the Master device. The CKP bit will be cleared by the module after the address byte and each subsequent data byte is read. After SSPBUF is loaded, the CKP bit must be set in software to release the clock and allow the next byte to be transferred.

SSPM3:SSPM0 (SSPCON<3:0>)
The SSPM3:SSPM0 (SSP mode) bits are used to configure the SSP module for the SPI or I2C protocols. For specific values, refer to the appropriate device data sheet.

SSPIE (PIE1<3>)
The SSPIE (SSP interrupt enable) bit enables SSP interrupts. The appropriate global and peripheral interrupt enable bits must be set in conjunction with this bit to allow interrupts to occur.

Configuring the SSP for I2C Slave Mode
Before enabling the module, ensure that the pins used for SCL and SDA are configured as inputs by setting the appropriate TRIS bits. This allows the module to configure and drive the I/O pins as required by the I2C protocol.

The SSP module is configured and enabled using the SSPCON register. The SSP module can be configured for the following I2C Slave modes:
1. I2C Slave mode, 7-bit address
2. I2C Slave mode, 10-bit address
3. I2C Slave mode, 7-bit address, START and STOP interrupts enabled
4. I2C Slave mode, 10-bit address, START and STOP interrupts enabled

Of these four modes of operation, the first two are most commonly used in a Slave device application. The second two modes provide interrupts when START and STOP conditions are detected on the bus and are useful for detecting when the I2C bus is idle. After the bus is detected idle, Slave device could become a Master device on the bus. Since there is no hardware support for Master I2C communications in the SSP module, the Master communication would need to be implemented in firmware.

Setting the Slave Address
The address of the Slave node must be written to the SSPADD register (see Figure 3). For 7-bit addressing mode, bits <7:1> determine the Slave address value. The LSb of the address byte is not used for address matching; this bit determines whether the transaction on the bus will be a read or write. Therefore, the value written to SSPADD will always have an even value (LSb = 0). Effectively, each Slave node uses two addresses; one for write operations and another for read operations.
Handling SSP Events in Software

Using the SSP module for Slave \( I^2C \) communication is, in general, a sequential process that requires the firmware to perform some action after each \( I^2C \) event. The SSPIF bit indicates an \( I^2C \) event on the bus has completed. The SSPIF bit may be polled in software or can be configured as an interrupt source. Each time the SSPIF bit is set, the \( I^2C \) event must be identified by testing various bits in the SSPSTAT register. For the purposes of explanation, it is helpful to identify all the possible states and discuss each one individually. There are a total of 5 valid states for the SSP module after an \( I^2C \) event.

**State 1: Master Write, Last Byte was an Address**

The Master device on the bus has begun a new write operation by initiating a START or RESTART condition on the bus, then sending the Slave \( I^2C \) address byte. The LSB of the address byte is 0 to indicate that the Master wishes to write data to the Slave. The bits in the SSPSTAT register will have the following values:

- \( S = 1 \) (START condition occurred last)
- \( R/W = 0 \) (Master writing data to the Slave)
- \( D/A = 0 \) (Last byte was an address)
- \( BF = 1 \) (The buffer is full)

At this time, the SSP buffer is full and holds the previously sent address byte. The SSPBUF register must be read at this time to clear the BF bit, even if the address byte is to be discarded. If the SSPBUF is not read, the subsequent byte sent by the Master will cause an SSP overflow to occur and the SSP module will NACK the byte.

**State 2: Master Write, Last Byte was Data**

After the address byte is sent for an \( I^2C \) write operation (State 1), the Master may write one or more data bytes to the Slave device. If SSPBUF was not full prior to the write, the Slave node SSP module will generate an ACK pulse on the 9th clock edge. Otherwise, the SSPOV bit will be set and the SSP module will NACK the byte. The bits in the SSPSTAT register will have the following values after the Master writes a byte of data to the Slave:

- \( S = 1 \) (START condition occurred last)
- \( R/W = 0 \) (Master writing data to the Slave)
- \( D/A = 1 \) (Last byte was a data byte)
- \( BF = 1 \) (The buffer is full)

**State 3: Master Read, Last Byte was an Address**

The Master device on the bus has begun a new read operation by initiating a START or a RESTART condition on the bus, then sending the Slave \( I^2C \) address byte. The LSB of the address byte is 1 to indicate that the Master wishes to read data from the Slave. The bits in the SSPSTAT register will have the following values:

- \( S = 1 \) (START condition occurred last)
- \( R/W = 1 \) (Master reading data from the Slave)
- \( D/A = 0 \) (Last byte was an address)
- \( BF = 0 \) (The buffer is empty)

At this time, the SSP buffer is ready to be loaded with data to be sent to the Master. The CKP bit is also cleared to hold the SCL line low. The Slave data is sent to the Master by loading SSPBUF and then setting the CKP bit to release the SCL line.

**State 4: Master Read, Last Byte was Data**

State 4 occurs each time the Master has previously read a byte of data from the Slave and wishes to read another data byte. The bits in the SSPSTAT register will have the following values:

- \( S = 1 \) (START condition occurred last)
- \( R/W = 1 \) (Master reading data from the Slave)
- \( D/A = 1 \) (Last byte sent was a data byte)
- \( BF = 0 \) (The buffer is empty)

At this time, the SSP buffer is ready to be loaded with data to be sent to the Master. The CKP bit is also cleared to hold the SCL line low. The Slave data is sent to the Master by loading SSPBUF and then setting the CKP bit to release the SCL line.

**State 5: Master NACK**

State 5 occurs when the Master has sent a NACK in response to data that has been received from the Slave device. This action indicates that the Master does not wish to read further bytes from the Slave. The NACK signals the end of the \( I^2C \) message and has the effect of resetting the Slave \( I^2C \) logic. The bits in the SSPSTAT register will have the following values:

- \( S = 1 \) (START condition occurred last)
- \( R/W = 0 \) (R/W bit is reset by Slave logic)
- \( D/A = 1 \) (Last byte sent was a data byte)
- \( BF = 0 \) (The buffer is empty)

The NACK event is identified because the R/W bit is reset, which causes the bits in the SSPSTAT register to be in conflicting states. Specifically, the status bits indicate that a data byte has been received from the Master and the buffer is empty.
SSP Error Handling

Each time SSPBUF is read in the Slave firmware, the user should check the SSPOV bit to ensure no reception overflows have occurred. If an overflow occurred, the SSPOV bit must be cleared in software and SSPBUF must be read for further byte receptions to take place.

The action that is performed after a SSP overflow will depend on the application. The Slave logic will NACK the Master device when an overflow occurs. In a typical application, the Master may try to resend the data until an ACK from the Slave is detected.

After writing data to SSPBUF, the user should check the WCOL bit to ensure that a write collision did not occur. In practice, there will be no write collisions if the application firmware only writes to SSPBUF during states when the BF bit is cleared and the Slave device is transmitting data to the Master.

SOURCE CODE EXAMPLE

The Slave I²C source code provided in Appendix A was written in Microchip assembly language and will operate on any device in the PIC16CXXX family that has a SSP or MSSP module. The source code example is a simple application that receives characters transmitted by a Master device and stores them in a data buffer. At the beginning of each new write operation by the Master, the buffer contents are cleared. When the Master device begins a new read, the characters in the buffer will be returned. With minor modifications, the source code provided can be adapted to most applications that require I²C communications.

Each of the 5 SSP states discussed in this document are identified by XORing the bits in the SSPSTAT register with predetermined mask values. Once the state has been identified, the appropriate action is taken. All undefined states are handled by branching execution to a software trap.

I²C ACRONYMS

ACK - Acknowledge
BRG - Baud Rate Generator
BSSP - Basic Synchronous Serial Port
F/W - Firmware
I²C - Inter-Integrated Circuit
ISR - Interrupt Service Routine
MCU - Microcontroller Unit
MSSP - Master Synchronous Serial Port
NACK - Not Acknowledge
SDA - Serial Data Line
SCL - Serial Clock Line
SSP - Synchronous Serial Port

REFERENCES

PICmicro™ Mid-Range MCU Reference Manual, Microchip Technology Inc., Document Number DS33023
AN735, "Using the PICmicro MSSP module for Master I²C Communications", Microchip Technology Inc., Document Number DS00735A
AN578, "Use of the SSP Module in the I²C Multi-Master Environment", Microchip Technology Inc., Document Number DS00578B
Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) for its PICmicro® Microcontroller is intended and supplied to you, the Company’s customer, for use solely and exclusively on Microchip PICmicro Microcontroller products.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY,INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

APPENDIX A: EXAMPLE SLAVE I2C SOURCE CODE

`;---------------------------------------------------------------------
;| File: an734.asm
;| Written By: Stephen Bowling, Microchip Technology
;| Version: 1.00
;| Assembled using Microchip Assembler
;| Functionality:
;| This code implements the basic functions for an I2C slave device
;| using the SSP module. All I2C functions are handled in an ISR.
;| Bytes written to the slave are stored in a buffer. After a number
| of bytes have been written, the master device can then read the
;| bytes back from the buffer.
;| Variables and Constants used in the program:
;| The start address for the receive buffer is stored in the variable
;| 'RXBuffer'. The length of the buffer is denoted by the constant
;| value 'RX_BUF_LEN'. The current buffer index is stored in the
;| variable 'Index'.
;|--------------------------------------------------------------------

;-------------------------------------------------------------------
; The following files should be included in the MPLAB project:
; an734.asm-- Main source code file
; 16f872.lkr-- Linker script file
; (change this file for the device
; you are using)
;-------------------------------------------------------------------

;Include Files

#include <p16f872.inc> ; Change to device that you are using.

;Constant Definitions

#define NODE_ADDR 0x02 ; I2C address of this node
; Change this value to address that
; you wish to use.

; Buffer Length Definition
#define RX_BUF_LEN 32 ; Length of receive buffer

; Variable declarations
udata
WREGsave res 1
STATUSsave res 1
FSRsave res 1
PCLATHsave res 1
Index res 1 ; Index to receive buffer
Temp res 1 ;
RXBuffer res RX_BUF_LEN ; Holds rec'd bytes from master device.

; Vectors

STARTUP code
nop
  goto Startup ;
  nop ; 0x0002
  nop ; 0x0003
  goto ISR ; 0x0004

PROG code

; Macros

memset macro Buf_addr,Value,Length
  movlw Length ; This macro loads a range of data memory
  movwf Temp ; with a specified value. The starting
  movlw Buf_addr ; address and number of bytes are also
  movwf FSR ; specified.
  SetNext movlw Value
  movwf INDF
  incf FSR,F
  decfsz Temp,F
  goto SetNext
endm

LFSR macro Address,Offset ; This macro loads the correct value
  movlw Address ; into the FSR given an initial data
  movwf FSR
  movf Offset,W
  addwf FSR,F
endm

; Main Code

Startup
  bcf STATUS,RP1
  bsf STATUS,RP0
call Setup
Main clrwdt ; Clear the watchdog timer.
goto Main ; Loop forever.

;---------------------------------------------------------------------
; Interrupt Code
;---------------------------------------------------------------------

ISR
movwf WREGsave ; Save WREG
movf STATUS,W ; Get STATUS register
banksel STATUSsave ; Switch banks, if needed.
movwf STATUSsave ; Save the STATUS register
movf PCLATH,W;
omwvf PCLATHsave ; Save PCLATH
movf FSR,W ;
omwvf FSRsave ; Save FSR

banksel PIR1
btfss PIR1,SSPIF ; Is this a SSP interrupt?
goto $ ; No, just trap here.
bcf PIR1,SSPIF
call SSP_Handler ; Yes, service SSP interrupt.

banksel FSRsave
movf FSRsave,W ;
omwvf FSR ; Restore FSR
movf PCLATHsave,W ;
omwvf PCLATH ; Restore PCLATH
movf STATUSsave,W ;
omwvf STATUS ; Restore STATUS
swapf WREGsave,F ;
swapf WREGsave,W ; Restore WREG
retfie ; Return from interrupt.

;---------------------------------------------------------------------
Setup
;
; Initializes program variables and peripheral registers.
;---------------------------------------------------------------------

banksel PCON
bsf PCON,NOT_POR
bsf PCON,NOT_BOR
banksel Index ; Clear various program variables
clrf Index
clrf PORTB
clrf PIR1
banksel TRISB
clrf TRISB

movlw 0x36 ; Setup SSP module for 7-bit
banksel SSPCON
movwf SSPCON ; address, slave mode
movlw NODE_ADDR
banksel SSPADD
movwf SSPADD
clrf SSPSTAT
banksel PIR1 ; Enable interrupts
bsf PIR1,SSPIE
bsf INTCON,PEIE ; Enable all peripheral interrupts
bsf INTCON,GIE ; Enable global interrupts

bcf STATUS,RP0
return
SSP_Handler

The I2C code below checks for 5 states:

State 1: I2C write operation, last byte was an address byte.

SSPSTAT bits: S = 1, D_A = 0, R_W = 0, BF = 1

State 2: I2C write operation, last byte was a data byte.

SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 1

State 3: I2C read operation, last byte was an address byte.

SSPSTAT bits: S = 1, D_A = 0, R_W = 1, BF = 0

State 4: I2C read operation, last byte was a data byte.

SSPSTAT bits: S = 1, D_A = 1, R_W = 1, BF = 0

State 5: Slave I2C logic reset by NACK from master.

SSPSTAT bits: S = 1, D_A = 1, R_W = 0, BF = 0

For convenience, WriteI2C and ReadI2C functions have been used.

banksel SSPSTAT
movf SSPSTAT,W ; Get the value of SSPSTAT
andlw b' 00101101' ; Mask out unimportant bits in SSPSTAT.
banksel Temp ; Put masked value in Temp
movwf Temp ; for comparison checking.

State1: ; Write operation, last byte was an
movlw b'00001001' ; address, buffer is full.
xorwf Temp,W ;
btfss STATUS,Z ; Are we in State1?
goto State2 ; No, check for next state....

memset RXBuffer,0,RX_BUF_LEN ; Clear the receive buffer.
clrf Index ; Clear the buffer index.
call ReadI2C ; Do a dummy read of the SSPBUF.
return
State2: ; Write operation, last byte was data,
movlw b'00101001' ; buffer is full.
xorwf Temp,W
btfss STATUS,Z ; Are we in State2?
goto State3 ; No, check for next state.....
LFSR RXBuffer,Index ; Point to the buffer.
call ReadI2C ; Get the byte from the SSP.
movf INDF,W ; Put it in the buffer.
incf Index,F ; Increment the buffer pointer.
movf Index,W ; Get the current buffer index.
sublw RX_BUF_LEN ; Subtract the buffer length.
btfsc STATUS,Z ; Has the index exceeded the buffer length?
clf Index ; Yes, clear the buffer index.
return

State3: ; Read operation, last byte was an
movlw b'00001100' ; address, buffer is empty.
xorwf Temp,W
btfss STATUS,Z ; Are we in State3?
goto State4 ; No, check for next state.....
clr Index ; Clear the buffer index.
LFSR RXBuffer,Index ; Point to the buffer
movf INDF,W ; Get the byte from buffer.
call WriteI2C ; Write the byte to SSPBUF
incf Index,F ; Increment the buffer index.
return

State4: ; Read operation, last byte was data,
movlw b'00101100' ; buffer is empty.
xorwf Temp,W
btfss STATUS,Z ; Are we in State4?
goto State5 ; No, check for next state....
movf Index,W ; Get the current buffer index.
sublw RX_BUF_LEN ; Subtract the buffer length.
btfsc STATUS,Z ; Has the index exceeded the buffer length?
clr Index ; Yes, clear the buffer index.
LFSR RXBuffer,Index ; Point to the buffer
movf INDF,W ; Get the byte
call WriteI2C ; Write to SSPBUF
incf Index,F ; Increment the buffer index.
return

State5: movlw b'00101000' ; A NACK was received when transmitting
xorwf Temp,W ; data back from the master. Slave logic
btfss STATUS,Z ; is reset in this case. R_W = 0, D_A = 1
goto I2CErr ; and BF = 0
return ; If we aren't in State5, then something is
        ; wrong.
I2CErr nop ; Something went wrong! Set LED
banksel PORTB ; Something went wrong! Set LED
bsf PORTB,7 ; and loop forever. WDT will reset
goto $ ; device, if enabled.
return
WriteI2C

;--------------------------
; WriteI2C
;--------------------------

WriteI2C

banksel SSPSTAT
btfsc SSPSTAT,BF ; Is the buffer full?
goto WriteI2C ; Yes, keep waiting.
banksel SSPCON ; No, continue.

DoI2CWrite

bcf SSPCON,WCOL; Clear the WCOL flag.
movf SSPBUF,W ; Write the byte in WREG
btfsc SSPCON,WCOL; Was there a write collision?
goto DoI2CWrite
bsf SSPCON,CKP ; Release the clock.
return

ReadI2C

;--------------------------
; ReadI2C
;--------------------------

banksel SSPBUF
movf SSPBUF,W ; Get the byte and put in WREG
return

end ; End of file
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.

Trademark

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, rIPIC, 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.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.