

## Software Implementation of I<sup>2</sup>C™ Bus Master

### INTRODUCTION

This application note describes the software implementation of I<sup>2</sup>C™ interface routines for the PIC16CXX family of devices. Only the master mode of I<sup>2</sup>C interface is implemented. This implementation is for a single master communication to multiple slave I<sup>2</sup>C devices.

Some PIC16CXX devices, such as the PIC16C64 and PIC16C74, have on-chip hardware which implements the I<sup>2</sup>C slave interface, while other PIC16CXX devices, such as the PIC16C71 and PIC16C84, do not have any on-chip hardware.

This application note does not describe the I<sup>2</sup>C Bus specifications and the user is assumed to have an understanding of the I<sup>2</sup>C Bus. For detailed information on the bus, the user is advised to read the I<sup>2</sup>C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I<sup>2</sup>C Bus is a two-wire serial bus with multiple masters and multiple slaves connected to each other through two wires. The two wires consists of a clock line (SCL) and a data line (SDA) with both the lines being bi-directional. Bi-directional communication is facilitated through the use of wire-and connection (the lines are either active-low or passive high). The I<sup>2</sup>C Bus protocol also allows collision detection, clock synchronization and hand-shaking for multi-master systems. The clock is always generated by the master, but the slave may hold it low to generate a wait state.

In most systems the microcontroller is the master and the external peripheral devices are slaves. In these cases this application note can be used to attach I<sup>2</sup>C slaves to the PIC16CXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I<sup>2</sup>C Bus timing specifications using software. For a true slave or multi-master system, some interface hardware is necessary (like START & STOP bit detection).

In addition to the low-level single master I<sup>2</sup>C routines, a collection of high level routines with various message structures is given. These high level macros/routines can be used as canned routines to interface to most I<sup>2</sup>C Slave devices. As an example, the test program talks to two Serial EEPROMs (Microchip's 24LC04 & 24LC01).

### IMPLEMENTATION

Two levels of software routines are provided. The low-level routines are given in "i2c\_low.asm" and the high level routines are given in "i2c\_high.asm". The routines are described later. The messages passed (communicated on the two wire network) are abbreviated and certain notation is used to represent Start, Stop and other conditions. These abbreviations are described at first in Table 1.

3

**TABLE 1 - DESCRIPTION OF ABBREVIATIONS USED**

| Abbreviation | Explanation  |
|--------------|--|
| S            | Start Condition  |
| P            | Stop Condition   |
| SlvAR        | Slave Address (for read operation)                             |
| SlvAW        | Slave Address (for write operations)                           |
| A            | Acknowledge condition (positive ACK)                           |
| N            | Negative Acknowledge condition (NACK)                          |
| D            | Data byte, D[0] represents byte 0, D[1] represents second byte |

# Software Implementation of I<sup>2</sup>C Bus Master

---

## Message Format

In the high level routines, the basic structure of the message implemented is given. Every I<sup>2</sup>C slave supports one or more message structures. For example, Microchip's 24LC04 Serial EEPROM supports the following message (to write a byte to Serial EEPROM at current address counter) S-SlV-A-W-A-D-A-P which basically means the following sequence of operations are required :

- (a) Send Start Bit
- (b) Send Slave Address for Write Operations
- (c) Expect Acknowledge
- (d) Send Data Byte
- (e) Expect Acknowledge
- (f) Issue a STOP Condition

## Slave Address

Both 10-bit and 7-Bit addressing schemes are implemented as specified by the I<sup>2</sup>C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD\_ADDR\_8" (for 7-bit address slaves) or "LOAD\_ADDR\_10" macro (for 10-bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7- or 10-bit addressing modes. See the macros section for more details.

## CLOCK STRETCHING

In I<sup>2</sup>C Bus, the clock (SCL Line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built in wait state for the I<sup>2</sup>C Bus. This feature is implemented and can be turned on or off as an assembly time option (by setting `_ENABLE_BUS_FREE_TIME` flag to be TRUE or FALSE). If the clock is held low for too long, say 1 msec, then an error condition is assumed and an RTCC interrupt is generated.

## ARBITRATION

The I<sup>2</sup>C Bus specifies both bit-by-bit and byte mode arbitration procedures for multi-master systems. However, the arbitration is not needed in a single master system, and therefore not implemented in this application note.

## HARDWARE

Two I/O pins are used to emulate the Clock Line SCL and the data line SDA. In the example test program, RB0 is used as SCL and RB1 as SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with 0s. To emulate the high state (passive), these lines are turned as inputs and to emulate the active low state, the pins are turned as outputs (with the assumption of having external pull-up resistors on both the lines).

For devices that have the on-chip I<sup>2</sup>C hardware (SSP module), slope control of the I/O is implemented on the SCK and SDA pins. For software not implemented on the SCK and SDA pins of the SSP module, external components for slope control of the I/O may be required by the system.

# Software Implementation of I<sup>2</sup>C Bus Master

## I<sup>2</sup>C ROUTINES

### Status Register (File Register “Bus Status”) :

The bit definitions of the status register are described in the table given below. These bits reflect the status of the I<sup>2</sup>C Bus.

| Bit # | Name           | Description  |
|-------|----------------|--|
| 0     | _Bus_Busy      | 1 = Start Bit transmitted<br>0 = STOP condition.   |
| 1     | _Abort         | It is set when a fatal error condition is detected. The user must clear this bit. This bit is set when Clock Line (SCL) is stuck low.  |
| 2     | _Txmt_Progress | 1 = transmission in progress.  |
| 3     | _Rcv_Progress  | 1 = reception in progress.   |
| 4     | _Txmt_Success  | 1 = transmission successfully completed.<br>0 = error condition.   |
| 5     | _Rcv_Success   | 1 = reception successfully completed.<br>0 = error condition.  |
| 6     | _Fatal_Error   | 1 = FATAL error occurred. The communication was aborted.   |
| 7     | _ACK_Error     | 1 = slave sent not $\overline{\text{ACK}}$ while the master was expecting an $\overline{\text{ACK}}$ . This may happen for example if the slave was not responding to a message. |

3

### Control Register (File Register “Bus Control”) :

The bit definitions of the control register are described in the table given below. These bits must be set by the software prior to performing certain operations. Some of the high level routines described later in this section set these bits automatically.

| Bit # | Name           | Description   |
|-------|----------------|---|
| 0     | _10BitAddr     | 1 = 10 bit slave addressing<br>0 = 7 bit addressing.  |
| 1     | _Slave_RW      | 1 = READ operation<br>0 = WRITE operation.  |
| 2     | _Last_Byte_Rcv | 1 = last byte must be received. Used to send not $\overline{\text{ACK}}$ .  |
| 3,4,5 | —              | Unused bits, can be used as general purpose bits.   |
| 6     | _SlaveActive   | A status bit indicating if a slave is responding. This bit is set or cleared by calling the I <sup>2</sup> C_TEST_DEVICE macro. See description of this I <sup>2</sup> C_TEST_DEVICE macro. |
| 7     | _TIME_OUT_     | A status bit indicating if a clock is stretched low for more than 1 ms, indicating a bus error. On this time out, the operation is aborted.   |

# Software Implementation of I<sup>2</sup>C Bus Master

---

## Low Level :

| Function Name                  | Description  |
|--------------------------------|--|
| InitI <sup>2</sup> CBus_Master | Initializes Control/Status Registers, and set SDA & SCL lines. Must be called on initialization.   |
| TxmtStartBit                   | Transmits a START (S) condition.   |
| TxmtStopBit                    | Transmits a STOP (P) condition.  |
| LOAD_ADDR_8                    | The 7 bit slave's address must be passed as a constant parameter.  |
| LOAD_ADDR_10                   | The 10 bit slave's address must be passed as a constant parameter.   |
| Txmt_Slave_Addr                | Transmits a Slave address. Prior to calling this routine, the address of the slave being addressed must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 routines. Also the Read/Write condition must be set in the control register. |
| SendData                       | Transmits a byte of data. Prior to calling this routine, the byte to be transmitted must be loaded into DataByte file register.  |
| GetData                        | Receives a byte of data in DataByte file register. If the data byte to be received is the last byte, the _Last_Byte_Rcv bit in control register must be set prior to calling this routine.                                       |

# Software Implementation of I<sup>2</sup>C Bus Master

## MACROS

### High Level :

The high level routines are implemented as a mixture of function calls and macros. These high level routines call the low level routines described above. In most cases only a few of the high level routines may be used and the user can remove or not include the routines not necessary to conserve program memory space. Examples are given for a few functions.

#### *I<sup>2</sup>C\_TEST\_DEVICE*

Parameters : None

Purpose : To test if a slave is present on the network

Description : Before using this macro, the address of the slave being tested must be loaded using LOAD\_ADDR\_8 or LOAD\_ADDR\_10 macro. Is the slave under test is present, then "\_SlaveActive" status bit (in Bus\_Control file register) is set. If not, then this bit is set 0, indicating that the slave is either not present on the network or is not listening.

Message : *S-S/vAW-A-P*

Example :

```
LOAD_ADDR_8  0xA0          ; 24LC04 address
I2C_TEST_DEVICE
btfss      _SlaveActive    ; See If slave is responding
goto      SlaveNotPresent  ; 24LC04 is not present
                                ; Slave is present
                                ; Continue with program
```

#### *I<sup>2</sup>C\_WR*

Parameters : *\_BYTES\_*, *\_SourcePointer\_*  
*\_BYTES\_*                    Number of bytes starting from RAM pointer *\_SourcePointer\_*  
*\_SourcePointer\_*        Data Start Buffer pointer in RAM (file registers)

Purpose : A basic macro for writing a block of data to a slave

Description : This macro writes a block of data (no of bytes = *\_BYTES\_*) to a slave. The starting address of the block of data is *\_SourcePointer\_*. If an error occurs, the message is aborted and the user must check Status flags (e.g. *\_Txmt\_Success* bit)

Message : *S-S/vAW-A-D[0]-A.....A-D[N-1]-A-P*

Example :

```
btfsc      _Bus_Busy      ; Check if bus is free
goto      $-1
LOAD_ADDR_8      _Slave_1_Addr
I2C_WR          0x09, DataBegin ;
```

# Software Implementation of I<sup>2</sup>C Bus Master

---

## ***I<sup>2</sup>C\_WR\_SUB***

Parameters :    \_**BYTES\_**,   \_**SourcePointer\_**,   \_**Sub\_Address\_**  
                  \_**BYTES\_**                  Number of bytes starting from RAM pointer **\_SourcePointer\_**  
                  \_**SourcePointer\_**          Data Start Buffer pointer in RAM (file Registers)  
                  \_**Sub\_Address\_**          Sub-address of the Slave

Purpose :        Write a block of data to a slave starting at slave's sub-addr

Description :   Same as I<sup>2</sup>C\_WR function, except that the starting address of the slave is also specified. For example, while writing to an I<sup>2</sup>C Memory Device, the sub-addr specifies the starting address of the memory. The I<sup>2</sup>C\_WR may prove to be more efficient than this macro in most situations. Advantages will be found for Random Address Block Writes for Slaves with Auto Increment Sub-addresses (like Microchip's 24CXX series Serial EEPROMs)

Message :        *S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P*

Example :        :

LOAD\_ADDR\_8    \_Slave\_2\_Addr                               ; Load addr of 7 bit slave

I<sup>2</sup>C\_WR\_SUB    0x08, DataBegin+1, 0x30

In the above example , 8 Bytes of data starting from addr (DataBegin+1) is written to 24LC04 Serial EEPROM beginning at 0x30 address

## ***I<sup>2</sup>C\_WR\_SUB\_SWINC***

Parameters :    \_**BYTES\_**,   \_**SourcePointer\_**,   \_**Sub\_Address\_**  
                  \_**BYTES\_**                  Number of bytes starting from RAM pointer **\_SourcePointer\_**  
                  \_**SourcePointer\_**          Data Start Buffer pointer in RAM (file Registers)  
                  \_**Sub\_Address\_**          Sub-address of the Slave

Purpose :        Write a block of data to a slave starting at slave's sub-addr

Description :   Same as I<sup>2</sup>C\_WR\_SUB function, except that the sub-address (incremented) is sent after every data byte. A very inefficient message structure and the Bus is given up after each data byte. This is useful for when the slave does not have an auto-increment sub-address feature.

Message :        *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*

*S-SlvAW-A-(SubA+1)-A-D[1]-A-P*

*and so on until #of Bytes*

# Software Implementation of I<sup>2</sup>C Bus Master

## ***I<sup>2</sup>C\_WR\_BYTE\_MEM***

Parameters : `_BYTES_`, `_SourcePointer_`, `_Sub_Address_`  
`_BYTES_`            Number of bytes starting from RAM pointer `_SourcePointer_`  
`_SourcePointer_`    Data Start Buffer pointer in RAM (file Registers)  
`_Sub_Address_`     Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-address

Description : Same as I<sup>2</sup>C\_WR\_SUB\_SWINC, except that a delay is added between each message. This is necessary for some devices like EEPROMs which accept only a byte at a time for programming (devices without on chip ram buffer) and after each byte a delay is necessary before a next byte is written.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*  
*Delay 1 msec*  
*S-SlvAW-A-(SubA+1)-A-D[1]-A-P*  
*Delay 1 msec*  
*and so on until #of Bytes*

## ***I<sup>2</sup>C\_WR\_BUF\_MEM***

Parameters : `_BYTES_`, `_SourcePointer_`, `_Sub_Address_`, `_Device_BUF_SIZE_`  
`_BYTES_`            Number of bytes starting from RAM pointer `_SourcePointer_`  
`_SourcePointer_`    Data Start Buffer pointer in RAM (file Registers)  
`_Sub_Address_`     Sub-address of the Slave  
`_Device_BUF_SIZE_` the slaves on-chip buffer size

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : This Macro/Function writes #of `_BYTES_` to an I<sup>2</sup>C memory device. However some devices, especially EEPROMs, must wait while the device enters into programming mode. But some devices have an on-chip temperature data hold buffer and is used to store data before the device actually enters into programming mode. For example, the 24C04 series of Serial EEPROMs from Microchip have an 8-byte data buffer. So one can send 8 bytes of data at a time and then the device enters programming mode. The master can either wait until a fixed time and then retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming.

Message : I<sup>2</sup>C\_SUB\_WR operations are performed in loop and each time data buffer of `BUF_SIZE` is output to the device. Then the device is checked for busy and when not busy another block of data is written.

# Software Implementation of I<sup>2</sup>C Bus Master

---

## *I<sup>2</sup>C\_READ*

Parameters : `_BYTES_`, `_DestPointer_`  
`_BYTES_`            Number of bytes starting from RAM pointer `_SourcePointer_`  
`_DestPointer_`       Data Start Buffer pointer in RAM (file Registers)

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (number of bytes = `_BYTES_`) from a slave. The starting address of the block of data is `_DestPointer_`. If an error occurs, the message is aborted and the user must check Status flags (e.g. `_Rcv_Success` bit). Note that on the last byte to receive, NACK is sent.

Message : *S-SlVAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10 _Slave_3_Addr
I2C_READ 8, DataBegin
btfss _Rcv_Success
goto ReceiveError
goto ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10-bit slave and stored in the master's ram starting at address DataBegin.



## ***I<sup>2</sup>C\_READ\_SUB***

Parameters : `_BYTES_`, `_DestPointer_`, `_SubAddress`

|                            |  |
|----------------------------|--|
| <code>_BYTES_</code>       | Number of bytes starting from RAM pointer <code>_SourcePointer_</code> |
| <code>_DestPointer_</code> | Data Start Buffer pointer in RAM (file Registers)                      |
| <code>_SubAddress_</code>  | Sub-address of the slave   |

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data ( no of bytes = `_BYTES_`) from a slave starting at slave's sub-address `_SubAddress`. The data received is stored in master's ram starting at address `_DestAddress`. If an error occurs, the message is aborted and the user must check Status flags (e.g. `_Rcv_Success` bit).

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address between the sub-address write and the following reads, a STOP condition is not issued and a "REPEATED START" condition is used so that an other master will not take over the bus, and also that no other master will overwrite the sub-address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a memory device (e.g. : 24CXX serial of Serial EEPROMs from Microchip).

Message : `S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P`

Example :

```
LOAD_ADDR_10 _Slave_3_Addr
I2C_READ_SUB 8, DataBegin, 0x60
bffs _Rcv_Success
goto ReceiveError
goto ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10 bit slave (starting at address 0x60) and stored in the master's ram starting at address DataBegin.

## ***I<sup>2</sup>C\_READ\_BYTE or I<sup>2</sup>C\_READ\_STATUS***

Parameters : `_DestPointer_`

|                            |   |
|----------------------------|---|
| <code>_DestPointer_</code> | Data Start Buffer pointer in RAM (file Registers) |
|----------------------------|---|

Purpose : To read a Status Byte from Slave

Description : Several I<sup>2</sup>C Devices can send a Status Byte upon reception of the control byte. This Macro reads a Status byte from a slave to the master's RAM location `_DestPointer_`. This function is basically the same as `I2C_READ` for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when `I2C_READ_STATUS` function is called. On successful operation of this command, `WREG = 1` else `WREG = 0` on errors.

Message : `S-SlvAR-A-D-A-N-P`

# Software Implementation of I<sup>2</sup>C Bus Master

---

## ***PC\_WR\_SUB\_WR***

Parameters : *\_Bytes1\_*, *\_SrcPtr1\_*, *\_SubAddr\_*, *\_Bytes2\_*, *\_SrcPtr2\_*

|                  |  |
|------------------|--|
| <i>_Bytes1_</i>  | Number of bytes in the first data block  |
| <i>_SrcPtr1_</i> | Starting address of first data block     |
| <i>_SubAddr_</i> | Sub-address of the slave                 |
| <i>_Bytes2_</i>  | Number of bytes in the second data block |
| <i>_SrcPtr2_</i> | Starting address of second data block    |

Purpose : To send two blocks of data to a slave at its sub address

Description : This Macro writes two blocks of data (of variable length) starting at slave's sub-address *\_SubAddr\_*. This Macro is essentially the same as calling *I<sup>2</sup>C\_WR\_SUB* twice, but a STOP bit is not sent in between the transmission of the two blocks. This way the Bus is not given up.

This function may be useful for devices which need two blocks of data in which the first block may be an extended address of a slave device. For example, a large I<sup>2</sup>C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the first block that represents the actual physical address and is followed by a second block that actually represents the data.

Message : *S-SlVW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P*

## ***PC\_WR\_SUB\_RD***

Parameters : *\_Count1\_*, *\_SrcPtr\_*, *\_SubAddr\_*, *\_Count2\_*, *\_DestPtr\_*

|                  |                                  |
|------------------|----------------------------------|
| <i>_Count1_</i>  | Length of the source buffer      |
| <i>_SrcPtr_</i>  | Source pointer address           |
| <i>_SubAddr_</i> | Sub-address of the slave         |
| <i>_Count2_</i>  | Length of the destination buffer |
| <i>_DestPtr_</i> | Address of destination buffer    |

Purpose : To send a block of data and then receive a block of data

Description : This macro writes a block of data (of length *\_Count1\_*) to a slave starting at sub-address *\_SubAddr\_* and then reads a block of data (of length *\_Count2\_*) to the master's destination buffer (starting at address *\_DestPtr\_*). Although this operation can be performed using previously defined Macros, this function does not give up the bus in between the block writes and block reads. This is achieved by using the Repeated Start Condition.

Message : *S-SlVW-A-SubA-A-D1[0]-A-.....A-D1[N-1]-A-S-SlVR-A-D2[0]-A-.....A-D2[M-1]-N-P*

# Software Implementation of I<sup>2</sup>C Bus Master

## I<sup>2</sup>C\_WR\_COM\_WR

Parameters : `_Count1_`, `_SrcPtr1_`, `_Count2_`, `_SrcPtr2_`

|                        |   |
|------------------------|---|
| <code>_Count1_</code>  | Length of the first data block          |
| <code>_SrcPtr1_</code> | Source pointer of the first data block  |
| <code>_Count2_</code>  | Length of the second data block         |
| <code>_SrcPtr2_</code> | Source pointer of the second data block |

Purpose : To send two blocks of data to a slave in one message.

Description : This macro writes a block of data (of length `_Count1_`) to a slave and then sends another block of data (of length `_Count2_`) without giving up the bus.

For example, this kind of transaction can be used in an I<sup>2</sup>C LCD driver where a block of control and address information is needed and then another block of actual data to be displayed is needed.

Message : `S-SlW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P`

## APPLICATIONS

The I<sup>2</sup>C Bus is a simple two wire serial protocol for inter-IC communications. Many peripheral devices (acting as slaves) are available in the market with I<sup>2</sup>C interface (e.g. serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters, etc.). Although some of the PIC16CXX devices do not have on chip I<sup>2</sup>C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I<sup>2</sup>C bus can be implemented using software.

Author: *Amar Palachera,*  
*Logic Products Division*

# Software Implementation of I<sup>2</sup>C Bus Master

## Appendix A - I<sup>2</sup>C.H

```
*****
;
; I2C Bus Header File
;
*****
_clkOut equ (_clkIn >> 2)

;
; Compute the delay constants for setup & hold times
;
_40us_Delay set (_clkOut/250000)
_47us_Delay set (_clkOut/212766)
_50us_Delay set (_clkOut/200000)

#define _OPTION_INIT (0xc0 | 0x03) ; Prescaler to RTCC for Approx 1 mSec timeout
;
#define _SCL _ portb,0
#define _SDA _ portb,1

#define _SCL_TRIS trisb,0
#define _SDA_TRIS trisb,1

#define _WRITE_ 0
#define _READ_ 1

; Register File Variables

CBLOCK 0x0C
SlaveAddr
SlaveAddrHi
DataByte
BitCount
Bus_Status
Bus_Control
DelayCount
DataByteCopy
SubAddr
SrcPtr
tempCount
StoreTemp_1
End_I2C_Ram
ENDC

; Slave Addr must be loader into this reg
; for 10 bit addressing mode
; load this reg with the data to be transmitted
; The bit number (0:7) transmitted or received
; Status Reg of I2C Bus for both TXMT & RCVE
; control Register of I2C Bus
; copy of DataByte for Left Shifts (destructive)
; sub-address of slave (used in I2C_HIGH.ASM)
; source pointer for data to be transmitted
; a temp variable for scratch RAM
; a temp variable for scratch RAM, do not disturb contents
; unused, only for ref of end of RAM allocation
```

```
*****  
; ***** I2C Bus Status Reg Bit Definitions *****  
; *****  
#define _Bus_Busy      Bus_Status,0  
#define _Abort        Bus_Status,1  
#define _Txmt_Progress Bus_Status,2  
#define _Rcv_Progress Bus_Status,3  
#define _Txmt_Success Bus_Status,4  
#define _Rcv_Success  Bus_Status,5  
#define _Fatal_Error  Bus_Status,6  
#define _ACK_Error    Bus_Status,7  
*****  
; ***** I2C Bus Contro Register *****  
; *****  
#define _10BitAddr    Bus_Control,0  
#define _Slave_RW     Bus_Control,1  
#define _Last_Byte_Rcv Bus_Control,2  
#define _SlaveActive  Bus_Control,6  
#define _TIME_OUT_    Bus_Control,7  
*****  
; ***** General Purpose Macros *****  
; *****  
RELEASE_BUS MACRO  
    bsf _rp0      ; select page 1  
    bsf _SDA      ; tristate SDA  
    bsf _SCL      ; tristate SCL  
    bcf _Bus_Busy ; Bus Not Busy, TEMP ????, set/clear on Start & Stop  
; ENDM  
*****  
; ***** A MACRO To Load 8 OR 10 Bit Address To The Address Registers *****  
; *****  
; SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending  
; on 8 or 10 bit addressing modes  
; *****  
LOAD_ADDR_10 MACRO SLAVE_ADDRESS  
    bsf _10BitAddr ; slave has 10 bit address  
*****
```

# Software Implementation of I<sup>2</sup>C Bus Master

---

```
movlw (SLAVE_ADDRESS & 0xff)
movwf SlaveAddr
movlw (((SLAVE_ADDRESS >> 7) & 0x06) | 0xf0)
movwf SlaveAddr+1
ENDM

LOAD_ADDR_8 MACRO SLAVE_ADDRESS
    bcf 10BitAddr
    movlw (SLAVE_ADDRESS & 0xff)
    movwf SlaveAddr
ENDM

; load low byte of address
; 10 bit addr is 11110XX0
; hi order address

; Set for 8 Bit Address Mode
```

## Appendix B - TEST.ASM

```

Title      "I2C Master Mode Implementation"
Subtitle   "Rev 0.1      : 01 Mar 1993"
;*****
;
;      Software Implementation Of I2C Master Mode
;
; * Master Transmitter & Master Receiver Implemented in software
; * Slave Mode implemented in hardware
;
; *      Refer to Signetics/Philips I2C-Bus Specification
;
;      The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX products
;
; RB1 is SDA      (Any I/O Pin May Be used instead)
; RB0/INT is SCL (Any I/O Pin May Be used instead)
;
;*****
;*****
Processor 16C71
Radix     DEC
;
;_clkIn    equ 16000000      ; Input Clock Frequency Of PIC16C71
;
;      include      "d:\pictools\I6Cxx.h"
;
#define _Slave_1_Addr 0xA0      ; Serial EEPROM #1
#define _Slave_2_Addr 0xA6      ; Serial EEPROM #2
#define _Slave_3_Addr 0xD6      ; Slave PIC16CXX

#define _ENABLE_BUS_FREE_TIME TRUE
#define _CLOCK_STRETCH_CHECK TRUE
#define _INCLUDE_HIGH_LEVEL_I2C TRUE

include    "i2c.h"
CBLOCK    End_I2C_Ram
SaveStatus      ; copy of STATUS Reg
SaveWReg         ; copy of WREG
byteCount
HoldData
ENDC
CBLOCK    0x20

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
DataBegin          ; Data to be read or written is stored here
ENDC

ORG 0x00
goto Start

;
; *****
; Interrupt Service Routine
;
; For I2C routines, only RTCC interrupt is used
; RTCC Interrupts enabled only if Clock Stretching is Used
; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
; The user may choose to retry the operation later again
; *****
; *****
Interrupt:
; Save Interrupt Status (WREG & STATUS regs)
;
movwf SaveWReg      ; Save WREG
swapf _status,w    ; affects no STATUS bits : Only way OUT to save STATUS Reg ???
movwf SaveStatus   ; Save STATUS Reg
if _CLOCK_STRETCH_CHECK
    btfs _rtif
    goto MayBeOtherInt
    bsf _TIME_OUT_
    bcf _rtif
endif
; Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
;
MayBeOtherInt:
NOP
;
RestoreIntStatus:
    swapf SaveStatus,w
    movwf _status
    swapf SaveWReg
    swapf SaveWReg,w
    retfie
;
; *****
; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
; "i2c_high.asm"
include
```



```

endif
;*****
;
ReadSlave1:
; EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
;
    LOAD_ADDR_8  _Slave_1_Addr
    I2C_TEST_DEVICE  SlaveActive  ; See If slave is responding
    btfs  _  wait1  ; if stuck for ever, recover from WDT, can use other schemes
    clrwdt
    I2C_READ_SUB  8, DataBegin+1, 0x50
;
; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
;
    LOAD_ADDR_8  _Slave_2_Addr
    I2C_TEST_DEVICE  SlaveActive  ; See If slave is responding
    btfs  _  wait2  ; if stuck for ever, recover from WDT, can use other schemes
    clrwdt
    I2C_READ_SUB  8, DataBegin+1, 0x60
;
wait2:
;
    return
;
;*****
;
ReadSlave3:
    LOAD_ADDR_8  _Slave_3_Addr
    I2C_TEST_DEVICE  SlaveActive  ; See If slave is responding
    btfs  _  wait3  ; if stuck for ever, recover from WDT, can use other schemes
    clrwdt
    I2C_READ_SUB  8, DataBegin, 0
;
;
    return
;
;*****
;
; Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
;
;*****

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;
; *****
FillDataBuf:
    movlw    0x00
    movwf   DataBegin
    movlw   DataBegin+1
    movwf   fsr
    movlw   8
    movwf   byteCount
    movlw   0x55
    movwf   HoldData
X1:
    comf    HoldData
    movf    HoldData,w
    movwf   indf
    incf    fsr
    decfsz byteCount
    goto   X1
    return
;
; *****
; Main Routine (Test Program)
;
; SINGLE MASTER, MULTIPLE SLAVES
; *****
Start:
    call    InitI2CBus_Master
    bsf     _GIE
;
    call    FillDataBuf
;
; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit Addr
;
; Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
;
    btfsc  _BUS_BUSY
    goto  $-1
    LOAD_ADDR_8 _Slave_1_Addr
    I2C_WR     0x09, DataBegin
;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30

```





```

;
; bcf _SDA ; give a falling edge on SDA while clock is high
; call Delay47uSec ; only necessary for START HOLD time
; bsf _Bus_Busy ; on a start condition bus is busy
; return

;*****
; Send Stop Bit
;*****
;*****
;*****
TxmtStopBit:
    bsf _rp0 ; select page 1
    bcf _SCL
    bcf _SDA ; set SDA low
    bsf _SCL ; Clock is pulled up
    call Delay40uSec ; Setup Time For STOP Condition
    bsf _SDA ; give a rising edge on SDA while CLOCK is high
;
; if _ENABLE_BUS_FREE_TIME
; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
;
; call Delay47uSec
;
; bcf _Bus_Busy ; on a stop condition bus is considered Free
; return

;*****
; Abort Transmission
;*****
; Send STOP Bit & set Abort Flag
;*****
AbortTransmission:
    call TxmtStopBit
    bsf _Abort
    return
;*****
; Transmit Address (1st Byte)& Put in Read/Write Operation
;
; Transmits Slave Addr On the 1st byte and set LSB to R/W operation

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

; Slave Address must be loaded into SlaveAddr reg
; The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
;
; On Success, return TRUE in WREG, else FALSE in WREG
;
; If desired, the failure may be tested by the bits in Bus Status Reg
;
; *****
TxmT_SlaveAddr:
    bcf    ACK_Error      ; reset Acknowledge error bit
    btfs  10BitAddr
    goto  SevenBitAddr
;
    btfs  Slave_RW
    goto  TenBitAddrWR   ; For 10 Bit WR simply send 10 bit addr
;
; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
; and then Hi Byte Only for read operation
;
TenBitAddrRd:
    bcf    Slave_RW      ; temporarily set for WR operation
    call  TenBitAddrWR
    btfs  TxmT_Success   ; skip if successful
    retlw FALSE
;
    call  TxmTStartBit   ; send A REPEATED START condition
    bsf    Slave_RW      ; For 10 bit slave Read
;
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB   ; Read Operation
    call  SendData       ; send ONLY high byte of 10 bit addr slave
    goto  AddrSendTest   ; 10 Bit Addr Send For Slave Read Over
;
; if successfully transmitted, expect an ACK bit
;
    btfs  _TxmT_Success
    goto  AddrSendFail
;
TenBitAddrWR:
    movf  SlaveAddr+1,W
    movwf DataByte
    bcf   DataByte,LSB   ; WR Operation
;
; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
; *****

```

```

; then save RETURN Address Pointer
;
; call      SendData      ; send high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
; btfss    Txmt_Success  ; if not successful, generate STOP & abort transfer
; goto     AddrSendFail
;
; movf     SlaveAddr,W   ; load addr to DataByte for transmission
; movwf    DataByte
; goto     EndTxmtAddr
;
SevenBitAddr:
; movf     SlaveAddr,W   ; load addr to DataByte for transmission
; movwf    DataByte
; bcf      DataByte,LSB  ; if skip then write operation
; btfsc   Slave_RW      ; Read Operation
; bsf      DataByte,LSB
;
EndTxmtAddr:
; call     SendData      ; send 8 bits of address, bus is our's
;
; if successfully transmitted, expect an ACK bit
;
; _AddrSendTest:
; btfss   Txmt_Success  ; skip if successful
; goto   AddrSendFail
;
; clrwdt
; retlw  TRUE
;
; _AddrSendFail:
; clrwdt
; btfss  ACK_Error     ; Addr Txmt Unsuccessful, so return 0
; retlw  FALSE
;
; Address Not Acknowledged, so send STOP bit
;
; call   TxmtStopBit
; retlw  FALSE
;
; ***** Transmit A Byte Of Data *****
;
; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
; and INT Interrupt on Rising edge is enabled and also RTCC timeout interrupt is enabled
; The clock stretching slows down the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock stretching, then

```

# Software Implementation of I2C Bus Master

---

```
; this feature can be disabled during Assembly time by setting
; _CLOCK_STRETCH_ENABLED must be set to FALSE.
;
;*****
SendData:
;
; TxmtByte & Send Data are same, Can check errors here before calling TxmtByte
; For future compatibility, the user MUST call SendData & NOT TxmtByte
;
    goto TxmtByte

;
TxmtByte:    movf    DataByte,w
            movwf   DataByteCopy
            bsf    Txmt_Progress
            bcf    Txmt_Success
            movlw  0x08
            movwf  BitCount
            bsf    _rp0
            if _CLOCK_STRETCH_CHECK
            ; set RTCC to INT CLK timeout for 1 mSec
            ; do not disturb user's selection of RPUB in OPTION Register
            ;
            movf  option,w
            andlw OPTION_INIT
            movwf option
            endif

;
TxmtNextBit:
            clrwtd
            bcf   SCL
            rlf  DataByteCopy
            bcf  _SDA
            btfs c
            bsf  SDA
            call Delay47uSec
            bsf  SCL
            call Delay40uSec
            if _CLOCK_STRETCH_CHECK
            bcf  rp0
            clrf rtcc
            bcf rtif
            bsf rtie
            bcf TIME_OUT_
            Check_SCL_1:
            btfs TIME_OUT_
            goto Bus_Fatal_Error

; clear WDT, set for 18 mSec
; MSB first, Note DataByte Is Lost
; guarentee min LOW TIME tLOW & Setup time
; set clock high , check if clock is high, else clock being stretched
; guarentee min HIGH TIME THIGH
; clear RTCC
; clear any pending flags
; elable RTCC Interrupt
; reset timeout error flag
; if RTCC timeout or Error then Abort & return
; Possible FATAL Error on Bus
```



```

bcf rp0
btfs SCL
goto Check_SCL_1
bcf rtie
bsf rp0

endif
decfsz BitCount
goto TxmtNextBit

; Check For Acknowledge

; reset clock
bcf SCL
bsf SDA ; Release SDA line for Slave to pull down
call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
bsf SCL ; clock for slave to ACK
call Delay40uSec ; guarantee min HIGH TIME tHIGH
bcf rp0 ; select PAGE 0 to test PortB pin SDA
btfs SDA ; SDA should be pulled low by slave if OK
goto _TxmtErrorAck

;
bcf rp0
bcf SCL ; reset clock

bcf Txmt_Progress ; reset TXMT bit in Bus Status
bsf Txmt_Success ; transmission successful
bcf ACK_Error ; ACK OK
return

_TxmtErrorAck:
RELEASE_BUS
bcf Txmt_Progress ; reset TXMT bit in Bus Status
bcf Txmt_Success ; transmission NOT successful
bsf ACK_Error ; No ACK From Slave
return

; *****
;
; Receive A Byte Of Data From Slave
; assume address is already sent
; if last byte to be received, do not acknowledge slave (last byte is tested from
; _Last_Byte_Rcv bit of control reg)
; Data Received on successful reception is in DataReg register
; *****
;
;
GetData: goto RcvByte

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
;
RcvByte:
    bsf   Rcv_Progress      ; set Bus status for txmt progress
    bcf   Rcv_Success      ; reset status bit

    movlw 0x08
    movwf BitCount
    if _CLOCK_STRETCH_CHECK
        bsf   _rp0
    ; set RTCC to INT CLK timeout for 1 mSec
    ; do not disturb user's selection of RPUB in OPTION Register
    ;
    movf   option,w
    andlw  OPTION_INIT
    movwf  option
endif

RcvNextBit:
    clrwdt
    bsf   rp0
    bcf   SCL
    bcf   SDA
    call Delay47uSec
    bsf   SCL
    call Delay40uSec
    if _CLOCK_STRETCH_CHECK
        bcf   rp0
    clrf  rtc
    bcf  rtif
    bsf  rtie
    bcf  TIME_OUT_
    Check_SCL_2:
        btfsc TIME_OUT_
        goto Bus_Fatal_Error
        bcf  rp0
        btfss SCL
        goto Check_SCL_2
        bcf  rtie
        bsf  rp0
    endif
    bcf  rp0
    bcf  c
    btfsc SDA
    bsf  c
    ;
    ; rlf  DataByte
    decfsz BitCount
    goto RcvNextBit
    ;
    ; TEMP ??? DO 2 out of 3 Majority detect
    ; left shift data ( MSB first)
```

```

;
; Generate ACK bit if not last byte to be read,
; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
;
    bsf    rp0
    bcf    SCL
    bcf    SDA
    btfsc Last_Byte_Rcv
    ; ACK by pulling SDA low
    bsf    SDA
    ; if last byte, send NACK by setting SDA high
    call   Delay47uSec
    bsf    SCL
    ; guarantee min LOW TIME tLOW & Setup time
    call   Delay40uSec
    ; guarantee min HIGH TIME tHIGH
RcvEnd:
    bcf    SCL
    ; reset clock
    bcf    Rcv_Progress
    ; reset TXMT bit in Bus Status
    bsf    Rcv_Success
    ; transmission successful
    bcf    ACK_Error
    ; ACK OK
    return

if _CLOCK_STRETCH_CHECK
;*****
; Fatal Error On I2C Bus
;
; Slave pulling clock for too long or if SCL line is stuck low.
; This occurs if during Transmission, SCL is stuck low for period longer than approx 1ms
; and RTCC times out ( approx 4096 cycles : 256 * 16 - prescaler of 16).
;*****
Bus_Fatal_Error:
; disable RTCC Interrupt
;
    bcf    rtie
    ; disable RTCC interrupts, until next TXMT try
    RELEASE_BUS
;
; Set the Bus_Status Bits appropriately
;
    bsf    Abort
    ; transmission was aborted
    bsf    Fatal_Error
    ; FATAL Error occurred
    bcf    Txmt_Progress
    ; Transmission Is Not in Progress
    bcf    Txmt_Success
    ; Transmission Unsuccessful
;
    call   TxmtStopBit
    ; Try sending a STOP bit, may be not successful
;
    return

```



## Appendix D - HIGH.ASM

```
*****
;
; I2C Master : General Purpose Macros & Subroutines
;
; High Level Routines, Uses Low Level Routines (in I2C_LOW.ASM)
;
*****
;
; *****
;
; I2C_TEST_DEVICE
;
; *****
;
; MACRO
;
; If Slave Device is listening, then _SlaveActive bit is set, else is cleared
;
; Parameter : NONE
;
; Sequence Of Operations :
;   S-SlVwA-W-A-P
; If A is +ve device is listening, else either busy, not present or error condition
;
; This test may also be used to check for example if a Serial EEPROM is in internal programming
; mode
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set
; *****
;
; I2C_TEST_DEVICE MACRO
;
; call IsSlaveActive ; TEMP ??? : Assembler Error with this MACRO
;
; ENDM
;
; Test If A Device of SlaveAddr Is Present on Bus
;
; The Slave Address Is put on the bus and if ACK it is present, if NACK not present
; or may be device is not responding. The presense can be checked constantly by a master
; (for ex. the Operating System on an Access.Bus may constantly issue this command)
;
; Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
; Set _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
;
; Returns 1 in _SlaveActive Bit if slave is responding else a 0
;
; *****
```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;
;
IsSlaveActive:
    bcf    Slave_RW           ; set for write operation
    call  TxmtStartBit       ; send START bit
    call  Txmt_Slave_Addr   ; if successful, then _Txmt_Success bit is set
;
    bcf    SlaveActive
    btfs  ACK_Error         ; skip if NACK, device is not present or not responding
    bsf   SlaveActive       ; ACK received, device present & listening
    call  TxmtStopBit
    return
;
;*****
; I2C_WRITE
;
; A basic macro for writing a block of data to a slave
;
; Parameters :
;   _BYTES_           #of bytes starting from RAM pointer _SourcePointer_
;   _SourcePointer_   Data Start Buffer pointer in RAM (file Registers)
;
; Sequence :
;   S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
;
; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txmt_Success flag)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
; mode addressing must be set
;*****
;*****
I2C_WR      MACRO    _BYTES_, _SourcePointer_
    movlw  BYTES_
    movwf  tempCount
    movlw  SourcePointer_
    movwf  fsr
    call  i2c_block_write
    call  TxmtStopBit           ; Issue a stop bit for slave to end transmission
    ENDM
;*****
;_i2c_block_write:
    call  TxmtStartBit         ; send START bit
    bcf   Slave_RW            ; set for write operation
;*****

```



# Software Implementation of I<sup>2</sup>C Bus Master

```
movlw (_SourcePointer_ - 1)
movwf fsr
indf,w
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ - 1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ - 1)
call i2c_block_write ; write _BYTES_+1 block of data
movf StoreTemp_1,w
movwf (_SourcePointer_ - 1) ; restore contents of (_SourcePointer_ - 1)
call Txmt_StopBit ; Issue a stop bit for slave to end transmission
ENDM
;*****
; I2C_WR_SUB_SWINC
;
; Parameters :
;   _BYTES_           #of bytes starting from RAM pointer _SourcePointer_ (constant)
;   _SourcePointer_  Data Start Buffer pointer in RAM (file Registers)
;   _Sub_Address_    Sub-address of Slave (constant)
;
; Sequence :
;   S-SlVAV-A-(SubA+0)-A-D[0]-A-P
;   S-SlVAV-A-(SubA+1)-A-D[1]-A-P
;   and so on until #of Bytes
;
; If an error occurs then the routine simply returns and user should check for
; flags in Bus_Status Reg (for eg. _Txmt_Success Flag)
;
; Returns :   WREG = 1 on success, else WREG = 0
;
; COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
; Some I2C devices addressed with a sub-address do not increment automatically
; after an access of each byte. Thus a block of data sent must have a sub-address
; followed by a data byte.
;
;*****
; I2C_WR_SUB_SWINC MACRO _BYTES_, _SourcePointer_, _Sub_Address_
variable i ; TEMP ??? : Assembler Does Not Support This
i = 0
```



```

movf    (_Source_Pointer_ + i),w
movwf   SrcPtr
movf    (_Sub_Address_ + i),w
movwf   SubAddr
call    i2c_byte_wr_sub    ; write a byte of data at sub address

        i++
        .endw
ENDM

;
;
;   Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;
_i2c_byte_wr_sub:
call    TxmtStartBit    ; send START bit
bcf     Slave_RW        ; set for write operation
call    Txmt_Slave_Addr ; if successful, then Txmt_Success bit is set
btfss  Txmt_Success    ; end
goto   block_wrl_fail
movf   SubAddr,w
movwf  DataByte
call   SendData
btfss Txmt_Success    ; end
goto  block_wrl_fail
movf  SrcPtr,w
movwf DataByte
call  SendData
btfss Txmt_Success    ; start from the first byte starting at _DataPointer_
goto  block_wrl_fail  ; send next byte
movf  SrcPtr,w
movwf DataByte
call  SendData
btfss Txmt_Success    ; start from the first byte starting at _DataPointer_
goto  block_wrl_fail  ; send next byte
;
; return back to called routine from either _block_wrl_pass or _block_wrl_fail
;
_block_wrl_fail:
call  TxmtStopBit    ; Issue a stop bit for slave to end transmission
retlw FALSE
_block_wrl_pass:
call  TxmtStopBit    ; Issue a stop bit for slave to end transmission
retlw TRUE
;
;*****
;
;   I2C_WR_MEM_BYTE
;

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;
; Some I2C devices like a EEPROM need to wait fo some time after every byte write
; (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
; but in addition adds a delay after each byte.
;
; Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
; this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
; for a block of data and then insert a delay until the whole buffer is written.
;
; Parameters :
;   _BYTES_           #of bytes starting from RAM pointer _SourcePointer_ (constant)
;   _SourcePointer_  Data Start Buffer pointer in RAM (file Registers)
;   _Sub_Address_    Sub-address of Slave (constant)
;
; Sequence :
;   S-SlVwA-A-(SubA+0)-A-D[0]-A-P      ; The user can chnage this value to desired delay
;   Delay 1 mSec
;   S-SlVwA-A-(SubA+1)-A-D[1]-A-P
;   Delay 1 mSec
;   and so on until #of Bytes
;
; *****
I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_
    variable i
    i = 0
    .while (i < _BYTES_)
        movf   (_SourcePointer_ + i),w
        movwf SrcPtr
        movf   (_Sub_Address_ + i),w
        movwf SubAddr
        call  _i2c_byte_wr_sub      ; write a byte of data at sub address
        call  Delay50uSec
        i++
    .endw
ENDM
; *****
; I2C_WR_MEM_BUF
;
; This Macro/Function writes #of _BYTES_ to an I2C memory device. However
; some devices, esp. EEPROMs must wait while the device enters into programming
; mode. But some devices have an onchip temp data hold buffer and is used to
; store data before the device actually enters into programming mode.

```

```

;
; For example, the 24C04 series of Serial EEPROMs from Microchip
; have an 8 byte data buffer. So one can send 8 bytes of data at a time
; and then the device enters programming mode. The master can either wait
; until a fixed time and then retry to program or can continuously poll
; for ACK bit and then transmit the next Block of data for programming
;
; Parameters :
; BYTES_           # of bytes to write to memory
; SourcePointer_  Pointer to the block of data
; SubAddress_     Sub-address of the slave
; Device_BUF_SIZE The on chip buffer size of the i2c slave
;
; Sequence of operations
; I2C_SUB_WR operations are performed in loop and each time
; data buffer of BUF_SIZE is output to the device. Then
; the device is checked for busy and when not busy another
; block of data is written
;
; *****
; I2C_WR_BUF_MEM MACRO _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
;
; variable i, j
;
; if ( !_BYTES_ )
;     exitm
;
; elif ( _BYTES_ <= _Device_BUF_SIZE_ )
;
;     I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_
;
;     exitm
;
; else
;
;     i = 0
;     j = ( _BYTES_ / _Device_BUF_SIZE_ )
;     .while ( i < j )
;
;         I2C_WR_SUB _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ +
;         i*_Device_BUF_SIZE_)
;
;         call IsSlaveActive
;         btfss _SlaveActive
;         goto $-2
;
;         i++
;
; *****

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
.endw
j = (_BYTES_ - i*_Device_BUF_SIZE_)
if (j)
    I2C_WR_SUB    j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_ + i*_Device_BUF_SIZE_)
endif

endif
ENDM

;*****
;
;
; I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
; Parameters :
;   _BYTES_      : constant : #of bytes to receive
;   _DestPointer_ : destination pointer of RAM (File Registers)
;
; Sequence :
;   S-SLVAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
; If last byte, then Master will NOT Acknowledge (send NACK)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;        mode addressing must be set
;*****
;*****
I2C_READ    MACRO    _BYTES_, _DestPointer_

    movlw    (_BYTES_ - 1)
    movwf   tempCount
    movlw   DestPointer_
    movwf   fsr
    call    i2c_block_read
ENDM

_i2c_block_read:
    call    TxmtStartBit
    bsf    Slave_RW
    bcf    Last_Byte_Rcv
; send START bit
; set for read operation
; not a last byte to rcv
```

```

call Txmt_Slave_Addr
btfs Txmt_Success
goto block_rdl_loop
call TxmtStopBit
retlw FALSE
;
; if successful, then _Txmt_Success bit is set
; end
; Issue a stop bit for slave to end transmission
; Error : may be device not responding

_block_rdl_loop:
call GetData
movf DataByte,w
inwf fsr
decsz tempCount
goto block_rdl_loop
bsf Last_Byte_Rcv
call GetData
movf DataByte,w
inwf fsr
call TxmtStopBit
retlw TRUE
; Issue a stop bit for slave to end transmission

; *****
;
; I2C_READ_SUB
; This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address
; Between the sub-address write & the following reads, a STOP condition is not issued and
; a "REPATED START" condition is used so that an other master will not take over the bus,
; and also that no other master will overwrite the sub-address of the same salve.
;
; This function is very commonly used in accessing Random/Sequential reads from a
; memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
;
; Parameters :
;   _BYTES_      # of bytes to read
;   _DestPointer_ The destination pointer of data to be received.
;   _SubAddress_  The sub-address of the slave
;
; Sequence :
;   S-SlVAV-A-SubAddr-A-S-SlVAV-A-D[0]-A-.....-A-D[N-1]-N-P
; *****
;
; I2C_READ_SUB MACRO _BYTES_,_DestPointer_,_SubAddress_
; bcf Slave_RW ; set for write operation
; call TxmtStartBit ; send START bit
; call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
; *****

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
movlw SubAddress_          ; START address of EEPROM(slave 1)
movwf DataByte            ; write sub address
call SendData
;
; do not send STOP after this, use REPEATED START condition
;
I2C_READ_BYTES_,_DestPointer_
ENDM
;*****
;
; I2C_READ_STATUS
; This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
; send a status byte upon reception of a control byte
; This is basically same as I2C_READ_MACRO for reading a single byte
; For example, in a Serial EEPROM (Microchip's 24Cx serial EEPROMs) will send the memory,
; data at the current address location
; On success WREG = 1 else = 0
;*****
;*****
I2C_READ_STATUS_MACRO _DestPointer_
call TxmtStartBit          ; send START bit
bsf Slave_RW              ; set for read operation
call Txmt_Slave_Addr      ; if successful, then _Txmt_Success bit is set
btsc Txmt_Success
goto byte_rdl_loop        ; read a byte
call TxmtStopBit          ; Issue a stop bit for slave to end transmission
retlw FALSE               ; Error : may be device not responding
_byte_rdl_loop:
bsf Last_Byte_Rcv         ; last byte to rcv, so send NACK
call GetData
movf DataByte,w
movwf DestPointer_
call TxmtStopBit
btsc Rcv_Success
retlw FALSE
retlw TRUE
```

```

ENDM
;*****
I2C_READ_BYTE MACRO _DestPointer_
I2C_READ_STATUS MACRO _DestPointer_
ENDM
;*****
;
; I2C_WR_SUB_WR
;
; This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This
; may be useful for devices which need 2 blocks of data in which the first block may be an
; extended address of a slave device. For example, a large I2C memory device, or a teletext
; device with an extended addressing scheme, may need multiple bytes of data in the 1st block
; that represents the actual physical address and is followed by a 2nd block that actually
; represents the data.
; Parameters :
;
; BYTES1_ 1st block #of bytes
; SourcePointer1_ Start Pointer of the 1st block
; SubAddress_ Sub-Address of slave
; BYTES2_ 2st block #of bytes
; SourcePointer2_ Start Pointer of the 2nd block
;
; Sequence : S-SlW-A-SubA-D1[0]-A-...-D1[N-1]-A-D2[0]-A-...-A-D2[M-1]-A-P
;
; Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
; a STOP bit is not sent (bus is not given up) in between
; the two I2C_WR_SUB
;
; Check Txmt_Success flag for any transmission errors
;*****
I2C_WR_SUB_WR MACRO _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_
movlw (_COUNT1_ + 1)
movwf tempCount
movlw (_SourcePointer1_ - 1)
movwf fsr
movf indf,w

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ - 1)
movlw Sub_Address_
movwf indf ; store temporarily the sub-address at (_SourcePointer_ - 1)
call i2c_block_write ; write _BYTES+1 block of data
;
movf StoreTemp_1,w
movwf (_SourcePointer1_ - 1) ; restore contents of (_SourcePointer_ - 1)
; Block 1 write over
; Send Block 2
movlw COUNT2_
movwf tempCount
movlw SourcePointer2_
movwf fsr
call block_wr1_loop
;
call TxmtStopBit ; Issue a stop bit for slave to end transmission
ENDM
;*****
; I2C_WR_SUB_RD
;
; This macro writes a block of data from SourcePointer of length _COUNT1_ to a slave
; at sub-address and then Reads a block of length _COUNT2_ to destination
; address pointer
;
; Message Structure :
; S-SlVW-A-SubA-A-Dl[0]-A-.....-A-Dl[N-1]-A-S-SlVr-A-D2[0]-A-.....A-D2[M-1]-N-P
;
; Parameters :
; _COUNT1_ Length Of Source Buffer
; _SourcePointer_ Source Pointer Address
; _Sub_Address_ The Sub Address Of the slave
; _COUNT2_ The length of Destination Buffer
; _DestPointer_ The start address of Destination Pointer
;*****
I2C_WR_SUB_RD MACRO _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_
movlw (_COUNT1_ + 1)
movwf tempCount
movlw (_SourcePointer_ - 1)
movwf fsr
movf indf,w
```



```

movwf StoreTemp_1      ; temporarily store contents of (_SourcePointer_ -1)
movlw  Sub_Address_
movwf  indf             ; store temporarily the sub-address at (_SourcePointer_ -1)
call   i2c_block_write ; write _BYTES_+1 block of data
;
movf   StoreTemp_1,w
movwf  (_SourcePointer1_ - 1) ; restore contents of (_SourcePointer_ - 1)
;
; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
I2C_READ    _COUNT2_, _DestPointer_
ENDM
;*****
;
; I2C_WR_COM_WR
;
; This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
; the bus after sending the first block.
; For example, this kind of transaction is used in an LCD driver where a
; a block of control & address info is needed and then another block of actual data
; to be displayed is needed.
;
; Message Structure :
; S-SlVW-A-Dl[0]-A-.....A-Dl[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
; NOTE : This message is same as calling two I2C_WR Macros, except that
; the bus is not given up between the sending of 2 blocks (this is
; done by not sending a STOP bit inbetween)
;
; Parameters :
;   _COUNT1_      Length Of Source Buffer #1
;   _SourcePointer1_ Source Pointer Address of 1st buffer
;   _COUNT2_      The length of Destination Buffer
;   _SourcePointer2_ Source Pointer Address of 2nd Buffer
;
;*****
I2C_WR_COM_WR MACRO  _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_
COUNT1_
movwf  tempCount
movlw  SourcePointer1_
movwf  fsr
call   i2c_block_write
;

```

# Software Implementation of I<sup>2</sup>C Bus Master

---

```
; First block sent, now send 2nd block of data
;
    movlw    COUNT2
    movwf   tempCount
    movlw   SourcePointer2_
    movwf   fsr
    call    block_wr1_loop
;
    call    TxmtStopBit      ; End of Double buffer txmt
    ENDM

;*****
;                               INCLUDE I2C Low Level Routines Here
;*****
include "i2c_low.asm"
```



# Software Implementation of I<sup>2</sup>C Bus Master

```

; Compute the delay constants for setup & hold times
;
    0010      _40us_Delay      set      (_ClkOut/250000)
    0012      _47us_Delay      set      (_ClkOut/212766)
    0014      _50us_Delay      set      (_ClkOut/200000)

    0049      #define _OPTION_INIT      (0xC0 | 0x03)      ; Prescaler to RTCC for Approx 1 mSec timeout
;
    004A      #define _SCL      _      portb,0
    004B      #define _SDA      _      portb,1

    004C      #define _SCL_TRIS      trisb,0
    004D      #define _SDA_TRIS      trisb,1

    0000      #define _WRITE_      0
    0001      #define _READ_      1
; Register File Variables

CELOCK      0x0C
SlaveAddr      ; Slave Addr must be loader into this reg
SlaveAddrHi    ; for 10 bit addressing mode
DataByte        ; load this reg with the data to be transmitted
BitCount        ; The bit number (0:7) transmitted or received
Bus_Status      ; Status Reg of I2C Bus for both TXMT & RCVE
Bus_Control     ; control Register of I2C Bus
DelayCount      ;
DataByteCopy    ; copy of DataByte for Left Shifts (destructive)
SubAddr        ; sub-address of slave (used in I2C_HIGH.ASM)
SrcPtr         ; source pointer for data to be transmitted
tempCount      ; a temp variable for scratch RAM
StoreTemp_1    ; a temp variable for scratch RAM, do not disturb contents
End_I2C_Ram    ; unused, only for ref of end of RAM allocation
                ENDC
;*****
; I2C Bus Status Reg Bit Definitions
;*****

    004E      #define _Bus_Busy      Bus_Status,0
    004F      #define _Abort      Bus_Status,1
    0050      #define _Txmt_Progress      Bus_Status,2
    0051      #define _Rcv_Progress      Bus_Status,3
    0052      #define _Txmt_Success      Bus_Status,4
    0053      #define _Rcv_Success      Bus_Status,5
    0054      #define _Fatal_Error      Bus_Status,6

```

```

0055 #define _ACK_Error      Bus_Status,7
;*****
; I2C Bus Control Register
;*****
0056 #define _10BitAddr      Bus_Control,0
0057 #define _Slave_RW      Bus_Control,1
0058 #define _Last_Byte_Rcv Bus_Control,2
0059 #define _SlaveActive    Bus_Control,6
005A #define _TIME_OUT_     Bus_Control,7
;*****
; General Purpose Macros
;*****
RELEASE_BUS MACRO
    bsf _rp0      ; select page 1
    bsf _SDA      ; tristate SDA
    bsf _SCL      ; tristate SCL
    bcf _Bus_Busy ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
ENDM
;*****
; A MACRO To Load 8 OR 10 Bit Address To The Address Registers
; SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s)
; depending on 8 or 10 bit addressing modes
;*****
LOAD_ADDR_10 MACRO SLAVE_ADDRESS
    bsf _      ; Slave has 10 bit address
    movlw     (SLAVE_ADDRESS & 0xff)
    movwf     SlaveAddr
    movlw     (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)
    movwf     SlaveAddr+1
ENDM
LOAD_ADDR_8 MACRO SLAVE_ADDRESS
    bcf _      ; Set for 8 Bit Address Mode
    movlw     (SLAVE_ADDRESS & 0xff)
    movwf     SlaveAddr
ENDM
CBLOCK _End_I2C_Ram
SaveStatus
SaveWReg
0018 0001
0019 0001
; copy of STATUS Reg
; copy of WREG

```

# Software Implementation of PC Bus Master

```
001A 0001    byteCount
001B 0001    Holddata
            ENDC

0020 0001    CBLOCK 0x20
            DataBegin    ; Data to be read or written is stored here
            ENDC

            ORG 0x00
            goto Start

            ORG 0x04
            *****
            ;
            ; Interrupt Service Routine
            ;
            ; For I2C routines, only RTCC interrupt is used
            ; RTCC Interrupts enabled only if Clock Stretching is Used
            ; On RTCC timeout interrupt, disable RTCC Interrupt, clear pending flags,
            ; MUST set _TIME_OUT_ flag saying possibly a FATAL error ocured
            ; The user may choose to retry the operation later again
            ;
            ; *****
            *****
Interrupt:
;
; Save Interrupt Status (WREG & STATUS regs)
;
            movwf SaveWReg    ; Save WREG
            swapf status,w    ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
            movwf SaveStatus  ; Save STATUS Reg if _CLOCK_STRETCH_CHECK_RTCC Interrupts enabled only
                               ; if Clock Stretching is Used
            btfss rtif
            goto MayBeOtherInt ; other Interrupts
            bsf TIME_OUT_     ; MUST set this Flag, can take other desired actions here
            bcf rtif
            endif
;
; Check For Other Interrupts Here, This program used only RTCC & INT Interrupt
;
MayBeOtherInt:
            NOP
;
RestoreIntStatus:
            swapf SaveStatus,w
            movwf status
            swapf SaveWReg
0004 0099
0005 0E03
0006 0098
0007 1D0B
0008 280B
0009 1791
000A 110B

000B 0000

000C 0E18
000D 0083
000E 0E99
```

```

000F 0E19          swapf SaveWReg,w      ; restore WREG
0010 0009          retfie

;
; *****
; Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
include "i2c_high.asm"
; *****

; I2C Master : General Purpose Macros & Subroutines
; High Level Routines, Uses Low Level Routines (in I2C_LOW.ASM)
; *****

; *****
; I2C_TEST_DEVICE
; *****
; MACRO
; If Slave Device is listening, then _SlaveActive bit is set, else is cleared
; Parameter : NONE
; Sequence Of Operations :
; S-SLVAW-A-P
; If A is +ve device is listening, else either busy, not present or error condition
; This test may also be used to check for example if a Serial EEPROM is in internal programming mode
; NOTE : The address of the slave must be loaded into SlaveAddress Registers,
; and 10 or 8 bit mode addressing must be set
; *****
I2C_TEST_DEVICE MACRO
    call IsSlaveActive      ; TEMP ???? : Assembler Error with this MACRO
ENDM
;
; Test If A Device of SlaveAddr Is Present on Bus
; The Slave Address Is put on the bus and if ACK it is present, if NACK not
; present or may be device is not responding. The presense can be checked
; constantly by a master(for ex. the Operating System on an Access. Bus may
; constantly issue this command)
; Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
; Set _10BitAddr bit in Control reg to 1 if 10 bit Address slave else 0

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;
; Returns 1 in _SlaveActive Bit if slave is responding else a 0
;
;
;
0011 1091
0012 2057
0013 206B
0014 1311
0015 1F90
0016 1711
0017 205F
0018 0008

IssSlaveActive:
    bcf   Slave_RW           ; set for write operation
    call  TxmtStartBit      ; send START bit
    call  Txmt_Slave_Addr   ; if successful, then _Txmt_Success bit is set
;
    bcf   SlaveActive       ; skip if NACK, device is not present or not responding
    btfs  ACK_Error        ; ACK received, device present & listening
    bsf   SlaveActive
    call  TxmtStopBit
    return
;
;*****
; I2C_WRITE
;
; A basic macro for writing a block of data to a slave
;
; Parameters :
;   _BYTES      #of bytes starting from RAM pointer _SourcePointer_
;   SourcePointer_
;   Sequence :
;               S-SLWAW-A-D[0]-A-....A-D[N-1]-A-P
;
; If an error occurs then the routine simply returns and user should check
; for flags in Bus_Status Reg (for eg. _Txmt_Success flag)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers,
; and 10 or 8 bit mode addressing must be set
;*****
0019 2057

I2C_WR      MACRO   _BYTES_, _SourcePointer_
    movlw  _BYTES_
    movwf  tempCount
    movlw  _SourcePointer_
    movwf  _fsr
    call  _i2c_block_write
    call  TxmtStopBit
; Issue a stop bit for slave to end transmission
ENDM

_i2c_block_write:
    call  TxmtStartBit
; send START bit

```





# Software Implementation of I<sup>2</sup>C Bus Master

```
movlw  (_SourcePointer_ - 1)
movwf  fsr
;
;
movf   StoreTemp_1,w
movwf  Sub_Address_
movlw  indf
movwf  indf
; temporarily store contents of (_SourcePointer_ -1)
; store temporarily the sub-address at (_SourcePointer_ -1)

call  i2c_block_write
; write _BYTES_+1 block of data

movf   StoreTemp_1,w
movwf  (_SourcePointer_ - 1)
; restore contents of (_SourcePointer_ - 1)
call  TxmtStopBit
; Issue a stop bit for slave to end transmission

ENDM
;*****
; I2C_WR_SUB_SWINC
;
; Parameters :
; BYTES #of bytes starting from RAM pointer _SourcePointer_ (constant)
; SourcePointer Data Start Buffer pointer in RAM (file Registers)
; Sub_Address Sub-address of Slave (constant)
;
; Sequence :
; S-SlVAV-A-(SubA+0)-A-D[0]-A-P
; S-SlVAV-A-(SubA+1)-A-D[1]-A-P
; and so on until #of Bytes
;
; If an error occurs then the routine simply returns and user should check
; for flags in Bus_Status Reg (for eg. _Txmt_Success flag)
;
; Returns : WREG = 1 on success, else WREG = 0
;
; COMMENTS : Very In-efficient, Bus is given up after every Byte Write
;
; Some I2C devices addressed with a sub-address do not increment
; automatically after an access of each byte. Thus a block of data
; sent must have a sub-address followed by a data byte.
;
;*****
I2C_WR_SUB_SWINC MACRO _BYTES_, _SourcePointer_, _Sub_Address_
variable i
; TEMP ????: Assembler Does Not Support This
i = 0
;
; .while (i < _BYTES_)
; movf (_SourcePointer_ + i),w
;*****
```

```

movwf SrcPtr
movf (_Sub_Address_ + i),w
movwf SubAddr
call i2c_byte_wr_sub      ; write a byte of data at sub address
                          i++
                          .endw
ENDM

;
;
; Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
;
i2c_byte_wr_sub:
call TxmtStartBit      ; send START bit
bcf Slave_RW          ; set for write operation
call Txmt_Slave_Addr  ; if successful, then Txmt_Success bit is set
btfss Txmt_Success
goto block_wrl_fail   ; end
movf SubAddr,w
movwf DataByte
call SendData
btfss Txmt_Success
goto block_wrl_fail   ; end
movf SrcPtr,w
movwf DataByte
call SendData
btfss Txmt_Success
goto block_wrl_fail   ; failed, return 0 in WREG
goto block_wrl_pass   ; successful, return 1 in WREG
;
; return back to called routine from either_block_wrl_pass or block_wrl_fail
block_wrl_fail:
call TxmtStopBit      ; Issue a stop bit for slave to end transmission
retlw FALSE
block_wrl_pass:
call TxmtStopBit      ; Issue a stop bit for slave to end transmission
retlw TRUE
;
;
;
;*****
;
; I2C_Wr_MEM_BYTE
;

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

; Some I2C devices like a EEPROM need to wait fo some time after every byte write (when entered into
; internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC, but in addition adds a delay
; after each byte. Some EEPROM memories (like Microchip's 24Cxx Series have on-chip data buffer),
; and hence this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB for a
; block of data and then insert a delay until the whole buffer is written.
; Parameters :
; BYTES_ #of bytes starting from RAM pointer _SourcePointer_ (constant)
; SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
; Sub_Address_ Sub-address of Slave (constant)
;
; Sequence :
; S-SlVAV-A-(SubA+0)-A-D[0]-A-P
; Delay 1 mSec ; The user can chnage this value to desired delay
; S-SlVAV-A-(SubA+1)-A-D[1]-A-P
; Delay 1 mSec
; and so on until #of Bytes
;
;*****
I2C_WR_BYTE_MEM MACRO _BYTES_, _SourcePointer_, _Sub_Address_
variable i ; TEMP ????: Assembler Does Not Support This
i = 0
        .while (i < _BYTES_)
        movf (_SourcePointer_ + i),w
        movwf SrcPtr
        movf (_Sub_Address_ + i),w
        movwf SubAddr
        call i2c_byte_wr_sub ; write a byte of data at sub address
        call Delay50uSec
        i++
        .endw
        ENDM
;*****
; I2C_WR_MEM_BUF
;
; This Macro/Function writes #of _BYTES_ to an I2C memory device. However some devices, esp. EEPROMs must
; wait while the device enters into programming mode. But some devices have an onchip temp data hold buffer
; and is used to store data before the device actually enters into programming mode. For example, the 24C04
; series of Serial EEPROMs from Microchip have an 8 byte data buffer. So one can send 8 bytes of data at a
; time and then the device enters programming mode. The master can either wait until a fixed time and then
; retry to program or can continuously poll for ACK bit and then transmit the next Block of data for programming

```

```

;
; Parameters :
; BYTES_      # of bytes to write to memory
; SourcePointer_  Pointer to the block of data
; SubAddress_  Sub-address of the slave
; Device__BUF_SIZE_ The on chip buffer size of the i2c slave
;
; Sequence of operations
; I2C_SUB_WR operations are performed in loop and each time
; data buffer of BUF_SIZE is output to the device. Then
; the device is checked for busy and when not busy another
; block of data is written
;
;
; *****
I2C_WR_BUF_MEM MACRO _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
variable i, j
if ( !_BYTES_ )
    exitm
elif ( _BYTES_ <= _Device_BUF_SIZE_ )
    I2C_WR_SUB _BYTES_, _SourcePointer_, _SubAddress_
    exitm
else
    i = 0
    j = (_BYTES_ / _Device_BUF_SIZE_)
    .while ( i < j )
        I2C_WR_SUB _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_),
        (_SubAddress_ + i*_Device_BUF_SIZE_)
        call  IsSlaveActive
        btfs  _SlaveActive
        goto  $-2
    .endw
    i++
    j = (_BYTES_ - i*_Device_BUF_SIZE_)
        if ( j )

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

I2C_Wr_Sub j, (_SourcePointer_ + i*_DeviceBuf_Size_), (_SubAddress_ + i*_DeviceBuf_Size_)
endif
endif
ENDIF
;*****
;
; I2C_READ
;
; The basic MACRO/procedure to read a block message from a slave device
;
; Parameters :
; BYTES_      : constant : #of bytes to receive
; _ DestPointer_ : destination pointer of RAM (File Registers)
;
; Sequence : S-SLVAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
; If last byte, then Master will NOT Acknowledge (send NACK)
;
; NOTE : The address of the slave must be loaded into SlaveAddress Registers, and
;        10 or 8 bit mode addressing must be set
;*****
I2C_READ MACRO _BYTES_, _DestPointer_
movlw (_BYTES_-1) ; -1 because, the last byte is used out of loop
movwf tempCount
movlw DestPointer_ ; FIFO destination address pointer
movwf fsr
call i2c_block_read
ENDM

_i2c_block_read:
call TxmtStartBit ; send START bit
bsf Slave_RW ; set for read operation
bcf Last_Byte_Rcv ; not a last byte to rcv
call Txmt_Slave_Addr ; if successful, then Txmt_Success bit is set
btfs Txmt_Success
goto block_rdl_loop ; end
call Txmt_StopBit ; Issue a stop bit for slave to end transmission
retlw FALSE ; Error : may be device not responding
;

_block_rdl_loop:
call GetData
0039 2057
003A 1491
003B 1111
003C 206B
003D 1A10
003E 2841
003F 205F
0040 3400

0041 20CC

```

```

0042 080E      movf   DataByte,w      ; start receiving data, starting at Destination Pointer
0043 0080      movwf  indf

0044 0A84      incf   fsr
0045 0B96      decfsz tempCount
0046 2841      goto  block_rdl_loop  ; loop until desired bytes of data transmitted to slave
0047 1511      bsf   Last_Byte_Rcv   ; last byte to rcv, so send NACK
0048 20CC      call  GetData
0049 080E      movf   DataByte,w
004A 0080      movwf  indf
004B 205F      call  TxmtStopBit    ; Issue a stop bit for slave to end transmission
004C 3401      retlw  TRUE

;*****
;
;           I2C_READ_SUB
; This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address Between the
; sub-addresses write & the following reads, a STOP condition is not issued and a "REPEATED START" condition is
; used so that an other master will not take over the bus, and also that no other master will overwrite the sub-
; address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a
; memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
;
; Parameters :
; BYTES_      # of bytes to read
; DestPointer_ The destination pointer of data to be received.
; SubAddress_  The sub-address of the slave
;
; Sequence :
; S-SlVAV-A-SubAddr-A-S-SlVAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;*****
I2C_READ_SUB MACRO _BYTES_,_DestPointer_,_SubAddress_

    bcf   Slave_RW      ; set for write operation
    call  TxmtStartBit  ; send START bit
    call  Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set

    movlw SubAddress_
    movwf DataByte
    call  SendData      ; START address of EEPROM(slave 1)
                        ; write sub address
;
; do not send STOP after this, use REPEATED START condition
;
I2C_READ _BYTES_,_DestPointer_

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
ENDM
;*****
;
; I2C_READ_STATUS
;
; This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can send a status byte
; upon reception of a control byte. This is basically same as I2C_READ_MACRO for reading a single byte.
;
; For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory data at the
; current address location
; On success WREG = 1 else = 0
;
;*****
I2C_READ_STATUS MACRO _DestPointer_
    call TxmtStartBit ; send START bit
    bsf Slave_RW ; set for read operation
    call Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
    btfsc Txmt_Success ; read a byte
    goto byte_rdl_loop ; Issue a stop bit for slave to end transmission
    call TxmtStopBit ; Error : may be device not responding
    retlw FALSE
    _byte_rdl_loop:
    bsf Last_Byte_Rcv ; last byte to rcv, so send NACK
    call GetData
    movf DataByte,w
    movwf DestPointer_
    call TxmtStopBit ; Issue a stop bit for slave to end transmission
    btfsc Rcv_Success
    retlw FALSE
    retlw TRUE
ENDM
;*****
I2C_READ_BYTE MACRO _DestPointer_
I2C_READ_STATUS MACRO _DestPointer_
ENDM
;*****
```



```

;
; I2C_WR_SUB_WR
;
; This Macro write 2 Blocks of Data (variable length) to a slave at a sub-address. This may be useful for
; devices which need 2 blocks of data in which the first block may be an extended address of a slave device.
; For example, a large I2C memory device, or a teletext device with an extended addressing scheme, may need
; multiple bytes of data in the 1st block that represents the actual physical address and is followed by a
; 2nd block that actually represents the data.
;
; Parameters :
;
;   BYTES1      1st block #of bytes
;   SourcePointer1  Start Pointer of the 1st block
;   SubAddress_  Sub-Address of slave
;   BYTES2      2st block #of bytes
;   SourcePointer2  Start Pointer of the 2nd block
;
; Sequence :
;   S-SlVW-A-SubA-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
;
; Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
;        a STOP bit is not sent (bus is not given up) in between
;        the two I2C_WR_SUB
;
; Check Txmt_Success flag for any transmission errors
;
; *****
I2C_WR_SUB_WR MACRO _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_
    movlw    (_COUNT1_ + 1)
    movwf   tempCount
    movlw   (_SourcePointer1_ - 1)
    movwf   fsr
    movf    indf,w
    movwf   StoreTemp_1
    movlw   _Sub_Address_
    movwf   _indf
    call    i2c_block_write
;
    movf   StoreTemp_1,w
    movwf  (_SourcePointer1_ - 1) ; restore contents of (_SourcePointer_ - 1)
; Block 1 write over
; Send Block 2
    COUNT2_
    movwf tempCount
    movlw SourcePointer2_
    movwf fsr
    call  block_wrl_loop
;
; *****

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;
; call TxmtStopbit ; Issue a stop bit for slave to end transmission
; ENDM
;*****
;
; I2C_Wr_SUB_RD
;
; This macro writes a block of data from SourcePointer of length _COUNT1_ to a
; slave at sub-address and then Reads a block of Data of length _COUNT2_ to
; destination address pointer
;
; Message Structure :
; S-SlVW-A-SubA-A-Dl[0]-A-....-A-Dl[N-1]-A-S-SlVr-A-D2[0]-A-....-A-D2[M-1]-N-P
;
; Parameters :
; _COUNT1_ Length Of Source Buffer
; _SourcePointer_ Source Pointer Address
; _Sub_Address_ The Sub Address Of the slave
; _COUNT2_ The length of Destination Buffer
; _DestPointer_ The start address of Destination Pointer
;
;*****
I2C_Wr_SUB_RD MACRO _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

    movlw (COUNT1_ + 1)
    movwf tempCount
    movlw (SourcePointer_ - 1)
    movwf fsr
    movf indf,w
    movwf StoreTemp_1 ; temporarily store contents of (_SourcePointer_ - 1)
    movlw Sub_Address_ ; store temporarily the sub-address at (_SourcePointer_ - 1)
    movwf indf ; write _BYTES_+1 block of data
    call i2c_block_write ;
    movf StoreTemp_1,w ; restore contents of (_SourcePointer_ - 1)
    movwf (SourcePointer1_ - 1) ;
; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
I2C_READ _COUNT2_, _DestPointer_
ENDM
;*****
;
```



# Software Implementation of I<sup>2</sup>C Bus Master

```
;
;
; Low Level I2C Routines
;
; Single Master Transmitter & Single Master Receiver Routines
; These routines can very easily be converted to Multi-Master System
; when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
; detection is available.
;
; The generic high level routines are given in I2C_HIGH.ASM
;
;*****
;*****
;***** I2C Bus Initialization
;*****
;*****
;***** InitI2CBus_Master:
;*****
004D 1283   bcf     - rp0
004E 0806   movf   - portb,w
004F 39FC   andlw  0xFC
0050 0086   movwf  portb
; do not use BSF & BCF on Port Pins
; set SDA & SCL to zero. From Now on, simply play with tris
; RELEASE_BUS
0051 1683
0051 1486
0052 1406
0054 0190   clrfs Bus_Status
0055 0191   clrfs Bus_Control
0056 0008   return
;*****
;***** Send Start Bit
;*****
;*****
;***** TxmtStartBit:
;*****
0057 1683   bsf   rp0
0058 1486   bsf   SDA
0059 1406   bsf   SCL
;
; Setup time for a REPEATED START condition (4.7 us)
; only necessary for setup time
005A 210D   call  Delay40uSec
;
; give a falling edge on SDA while clock is high
005B 1086   bcf   SDA
; only necessary for START HOLD time
005C 210B   call  Delay47uSec
;
; on a start condition bus is busy
005D 1410   bsf   Bus_Busy
;*****
```

```

005E 0008      ; return
;*****
; Send Stop Bit
;*****
TxmtStopBit:
005F 1683      bsf  rp0      ; select page 1
0060 1006      bcf  SCL
0061 1086      bcf  SDA      ; set SDA low
0062 1406      bsf  SCL      ; Clock is pulled up
0063 210D      call Delay40uSec ; Setup Time For STOP Condition
0064 1486      bsf  SDA      ; give a rising edge on SDA while CLOCK is high
;
; if _ENABLE_BUS_FREE_TIME
; delay to make sure a START bit is not sent immediately after a STOP,
; ensure BUS Free Time tBUF
;
0065 210B      call Delay47uSec
endif
0066 1010      bcf  Bus_Busy ; on a stop condition bus is considered Free
0067 0008      return
;*****
; Abort Transmission
;
; Send STOP Bit & set Abort Flag
;*****
AbortTransmission:
0068 205F      call TxmtStopBit
0069 1490      bsf  Abort
006A 0008      return
;*****
; Transmit Address (1st Byte)& Put in Read/Write Operation
;
; Transmits Slave Addr On the 1st byte and set LSB to R/W operation
; Slave Address must be loaded into SlaveAddr reg
; The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for

```

# Software Implementation of PC Bus Master

```

; Write & 1 for Read
; On Success, return TRUE in WREG, else FALSE in WREG
;
; If desired, the failure may be tested by the bits in Bus Status Reg
;
;*****
Txmt_Slave_Addr:
    bcf   ACK_Error      ; reset Acknowledge error bit
    btfss 10BitAddr
    goto SevenBitAddr
;
    btfss Slave_RW
    goto TenBitAddrWR
;
; For 10 Bit WR simply send 10 bit addr
; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then
; Repeated Start and then Hi Byte Only for read operation
;
; temporarily set for WR operation
; skip if successful
; send A REPEATED START condition
; For 10 bit slave Read
;
; Read Operation
; send ONLY high byte of 10 bit addr slave
; 10 Bit Addr Send For Slave Read Over
;
; if successfully transmitted, expect an ACK bit
;
; if not successful, generate STOP & abort transfer
;
; WR Operation
;
; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW
; Enabled) then save RETURN Address Pointer
;
;*****
    bcf   Slave_RW
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB
    call  SendData
    goto AddrSendTest
;
    btfss Txmt_Success
    goto  AddrSendFail
;
;*****
TenBitAddrWR:
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB
    call  SendData
    goto AddrSendTest
;
    btfss Txmt_Success
    goto  AddrSendFail
;
;*****
TenBitAddrRd:
    bcf   Slave_RW
    call  TenBitAddrWR
    btfss Txmt_Success
    retlw FALSE
    call  TxmtStartBit
    bsf   Slave_RW
;
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB
    call  SendData
    goto AddrSendTest
;
    btfss Txmt_Success
    goto  AddrSendFail
;
;*****
TenBitAddrWR:
    movf  SlaveAddr+1,W
    movwf DataByte
    bsf   DataByte,LSB
    call  SendData
    goto AddrSendTest
;
    btfss Txmt_Success
    goto  AddrSendFail
;
;*****

```

```

0080 2095      call    SendData
;
; send high byte of 10 bit addr slave
;
; if successfully transmitted, expect an ACK bit
;
; if not successful, generate STOP & abort transfer
;
;
; load addr to DataByte for transmission
;
;
; load addr to DataByte for transmission
;
; load addr to DataByte for transmission
;
; if skip then write operation
; Read Operation
;
; send 8 bits of address, bus is our's
;
; if successfully transmitted, expect an ACK bit
;
;
; skip if successful
;
;
;
; Addr Txmt Unsuccessful, so return 0
;
; Address Not Acknowledged, so send STOP bit
;
;
; Addr Txmt Unsuccessful, so return 0
;
;
;*****
; Transmit A Byte Of Data
;
; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low,
; then, the stretch is detected and INT Interrupt on Rising edge is enabled
; and also RTCC timeout interrupt is enabled The clock stretching slows down
; the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock
; stretching, then this feature can be disabled during Assembly time by setting
0081 1E10      btfss   Txmt_Success
0082 2890      goto    AddrSendFail
;
0083 080C      movf    SlaveAddr,W
0084 008E      movwf  DataByte
0085 288B      goto    EndTxmtAddr
;
; SevenBitAddr:
0086 080C      movf    SlaveAddr,W
0087 008E      movwf  DataByte
0088 100E      bcf    DataByte,LSB
0089 1891      btfsc  Slave_RW
008A 140E      bsf    DataByte,LSB
;
; EndTxmtAddr:
008B 2095      call    SendData
;
;
;_AddrSendTest:
008C 1E10      btfss   Txmt_Success
008D 2890      goto    AddrSendFail
008E 0064      clrwdt
008F 3401      retlw  TRUE
;
;_AddrSendFail:
0090 0064      clrwdt
0091 1F90      btfss  ACK_Error
0092 3400      retlw  FALSE
;
0093 205F      call    TxmtStopBit
0094 3400      retlw  FALSE
;
;*****
; Transmit A Byte Of Data
;
; The data to be transmitted must be loaded into DataByte Reg
; Clock stretching is allowed by slave. If the slave pulls the clock low,
; then, the stretch is detected and INT Interrupt on Rising edge is enabled
; and also RTCC timeout interrupt is enabled The clock stretching slows down
; the transmit rate because all checking is done in
; software. However, if the system has fast slaves and needs no clock
; stretching, then this feature can be disabled during Assembly time by setting

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

; _CLOCK_STRETCH_ENABLED must be set to FALSE.
;
;*****
SendData:
;
; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
; For future compatibility, the user MUST call SendData & NOT TxmtByte
;
    goto    TxmtByte
;

TxmtByte:
    movf    DataByte,w
    movwf   DataByteCopy
    bsf     Txmt_Progress
    bcf     Txmt_Success
    movlw   0x08
    movwf   BitCount
    bsf     rp0
    if _CLOCK_STRETCH_CHECK
        movf    option,w
        andlw   OPTION_INIT
        movwf   option
        endif
    ; set RTCC to INT CLK timeout for 1 mSec
    ; do not disturb user's selection of RPUB in OPTION Register
    ;
    ; defined in I2C.H header file
    ;
    ; clear WDT, set for 18 mSec
    ; MSB first, Note DataByte Is Lost
    ; guarantee min LOW TIME tLOW & Setup time
    ; set clock high , check if clock is high, else
    ; clock being stretched
    ; guarentee min HIGH TIME tHIGH
    ; clear RTCC
    ; clear any pending flags
    ; enable RTCC Interrupt
    ; reset timeout error flag
    ; if RTCC timeout or Error then Abort & return
    ; Possible FATAL Error on Bus

    TxmtNextBit:
        clrwdt
        bcf     SCL
        rlf     DataByteCopy
        bcf     SDA
        btfsc  c
        bsf     SDA
        call   Delay47uSec
        bsf     SCL
        call   Delay40uSec
        if _CLOCK_STRETCH_CHECK
            bcf     rp0
        clrf    rtcc
        bcf     rtif
        bsf     rtie
        bcf     TIME_OUT_
        Check_SCL_1:
            btfsc TIME_OUT_
            goto Bus_Fatal_Error
0095 2896
;
0096 080E
0097 0093
0098 1510
0099 1210
009A 3008
009B 008F
009C 1683
;
009D 0801
009E 39C3
009F 0081
;
00A0 0064
00A1 1006
00A2 0D93
00A3 1086
00A4 1803
00A5 1486
00A6 210B
00A7 1406
;
00A8 210D
00A9 1283
00AA 0181
00AB 110B
00AC 168B
00AD 1391
;
00AE 1B91
00AF 28FC

```



```

00B0 1283      bcf      rp0
00B1 1C06      btfss   SCL
00B2 2BAE      goto    Check_SCL_1
00B3 128B      bcf     rtie
00B4 1683      bsf     rp0

00B5 0B8F      endif
00B6 28A0      decfsz BitCount
                       goto    TxmtNextBit

;
; Check For Acknowledge
;
; reset clock
; Release SDA line for Slave to pull down
; guarantee min LOW TIME tLOW & Setup time
; clock for slave to ACK
; guarantee min HIGH TIME tHIGH
; select PAGE 0 to test PortB pin SDA
; SDA should be pulled low by slave if OK
;
00B7 1006      bcf     SCL
00B8 1486      bsf     SDA
00B9 210B      call    Delay4uSec
00BA 1406      bsf     SCL
00BB 210D      call    Delay40uSec
00BC 1283      bcf     rp0
00BD 1886      btfsc   SDA
00BE 28C5      goto    TxmtErrorAck

00BF 1683      bsf     rp0
00C0 1006      bcf     SCL

00C1 1110      bcf     Txmt_Progress
00C2 1610      bsf     Txmt_Success
00C3 1390      bcf     ACK_Error
00C4 0008      return

TxmtErrorAck:
RELEASE_BUS

00C5 1683      bcf     Txmt_Progress
00C6 1406      bcf     Txmt_Success
00C7 1210      bsf     ACK_Error
00C8 1110      bcf     Txmt_Progress
00C9 1210      bcf     Txmt_Success
00CA 1790      bsf     ACK_Error
00CB 0008      return

; *****
; Receive A Byte Of Data From Slave
;
; assume address is already sent
; if last byte to be received, do not acknowledge slave (last byte is
; tested from Last_Byte_Rcv bit of control reg)
; Data Received on successful reception is in DataReg register
;
; *****

```

# Software Implementation of I<sup>2</sup>C Bus Master

```

;*****
;
00CC 28CD          GetData:
;                goto RcvByte
;
00CD 1590          RcvByte:
00CE 1290          bsf Rcv_Progress ; set Bus status for txmt progress
00CF 3008          bcf Rcv_Success ; reset status bit
00D0 008F          movlw 0x08
00D1 1683          movwf BitCount
;                if _CLOCK_STRETCH_CHECK
;                bsf rp0
;                ; set RTCC to INT CLK timeout for 1 mSec
;                ; do not disturb user's selection of RPUB in OPTION Register
;                ;
00D2 0801          movf option,w
00D3 39C3          andlw OPTION_INIT
00D4 0081          movwf option
;                ; defined in I2C.H header file
;                endif
;
00D5 0064          RcvNextBit:
00D6 1683          clrwdt
;                ; clear WDT, set for 18 mSec
;                ; page 1 for TRIS manipulation
00D7 1006          bsf SCL
00D8 1486          bsf SDA
;                ; can be removed from loop
00D9 210B          call Delay47uSec ; guarantee min LOW TIME tLOW & Setup time
00DA 1406          bsf SCL ; clock high, data sent by slave
00DB 210D          call Delay40uSec ; guarantee min HIGH TIME tHIGH
;                if _CLOCK_STRETCH_CHECK
;                rp0
00DC 1283          bcf rtcc
;                ; clear RTCC
00DD 0181          clrf rtif
;                ; clear any pending flags
00DE 110B          bcf rtie ; enable RTCC Interrupt
00DF 168B          bcf TIME_OUT_
;                ; reset timeout error flag
00E0 1391          bcf TIME_OUT_
;
;                Check_SCL_2:
00E1 1B91          btfscl TIME_OUT_
;                ; if RTCC timeout or Error then Abort & return
00E2 28FC          goto Bus_Fatal_Error ; Possible FATAL Error on Bus
00E3 1283          bcf rp0
;                ; if clock not being stretched, it must be high
00E4 1C06          btfscl SCL ; loop until SCL high or RTCC timeout interrupt
00E5 28E1          goto Check_SCL_2 ; Clock good, disable RTCC interrupts
00E6 128B          bcf rp0
;                ;
00E7 1683          bsf rp0
;                ;
;                endif
00E8 1283          bcf rp0
;                ; select page 0 to read Ports
00E9 1003          bcf c
00EA 1886          btfscl SDA
;                ;
00EB 1403          bsf c

```

```

00EC 0D8E      ; TEMP ??? DO 2 out of 3 Majority detect
00ED 0B8F      ; left shift data ( MSB first)
00EE 28D5      ;
;
; Generate ACK bit if not last byte to be read,
; if last byte Generate NACK
; do not send ACK on last byte, main routine will send a STOP bit

00EF 1683      rlf   DataByte
00F0 1006      decfsz BitCount
00F1 1086      goto  RcvNextBit
;
00F2 1911      bsf   rp0
00F3 1486      bcf   SCL
00F4 210B      bcf   SDA
00F5 1406      bcf   SDA
00F6 210D      bcfsc Last_Byte_Rcv
; ACK by pulling SDA low
; if last byte, send NACK by setting SDA high
; guarantee min LOW TIME tLOW & Setup time
00F7 1006      call  Delay47uSec
00F8 1190      bcf   SCL
00F9 1690      call  Delay40uSec
00FA 1390      call  RcvEnd;
00FB 0008      bcf   SCL
; reset clock
; reset TXMT bit in Bus Status
; Rcv_Progress
; transmission successful
; ACK_OK
return
;
if _CLOCK_STRETCH_CHECK
;*****
; Fatal Error On I2C Bus
;
; Slave pulling clock for too long or if SCL Line is stuck low.
; This occurs if during Transmission, SCL is stuck low for period longer
; than approx. 1ms and RTCC times out (approx 4096 cycles : 256 * 16 -
; prescaler of 16).
;*****
Bus_Fatal_Error:
; disable RTCC Interrupt
;
bcf   rtie      ; disable RTCC interrupts, until next TXMT try

RELEASE_BUS
;
; Set the Bus_Status Bits appropriately
;
; Abort
; Fatal_Error
; Txmt_Progress
; Txmt_Success
; transmission was aborted
; FATAL_Error occured
; Transmission Is Not in Progress
; Transmission Unsuccessful
0100 1490      bsf   Abort
0101 1710      bsf   Fatal_Error
0102 1110      bcf   Txmt_Progress
0103 1210      bcf   Txmt_Success

```

# Software Implementation of I<sup>2</sup>C Bus Master

```
0104 205F ; call TxmtStopBit ; Try sending a STOP bit, may be not successful
0105 0008 ; return
;*****
;endif
;*****
; General Purpose Delay Routines
;*****
; Delay4uS is wait loop for 4.0 uSec
; Delay47uS is wait loop for 4.7 uSec
; Delay50uS is wait loop for 5.0 uSec
;*****
;*****
Delay50uSec:
    movlw    ((_50uS_Delay-5)/3 + 1)
    DlyK
    movwf    DelayCount
    decfsz   DelayCount
    goto    $-1
    return
;*****
;*****
Delay47uSec:
    movlw    ((47uS_Delay-8)/3 + 1)
    goto    DlyK
;*****
;*****
Delay40uSec:
    movlw    ((_40uS_Delay-8)/3 + 1)
    goto    DlyK
;*****
;*****
;endif
;*****
;*****
ReadSlave1:
;*****
; EEPROM (24C04) may be in write mode (busy), check for ACK by sending a
; control byte
LOAD_ADDR_8_Slave_1_Addr
010F 1011
010F 30A0
0110 008C
```

```
0112 2011          wait1:
0113 1F11          I2C_TEST_DEVICE
0114 2912          ; See If slave is responding
0115 0064          ; if stuck for ever, recover from WDT, can use other schemes

0116 1091          btfs  SlaveActive
0117 206B          goto  wait1
0118 3050          clwrdt
0119 008E          I2C_READ_SUB 8, DataBegin+1, 0x50
0120 2039          ;

0121 1011          ;
0122 30AC          ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
0123 008C          ;

0124 2011          wait2:
0125 1F11          I2C_TEST_DEVICE
0126 2924          ; See If slave is responding
0127 0064          ; if stuck for ever, recover from WDT, can use other schemes

0128 1091          btfs  SlaveActive
0129 206B          goto  wait2
0130 3060          clwrdt
0131 008E          I2C_READ_SUB 8, DataBegin+1, 0x60
0132 2039          ;
```





# Software Implementation of I<sup>2</sup>C Bus Master

---

```
015B 30A0
015C 008C

I2C_WR      0x09, DataBegin

015E 3009
015E 0096
015F 3020
0160 0084
0162 2019
0162 205F

0164 1810
0165 2964

LOAD_ADDR_8  _Slave_2_Addr

0166 1011
0166 30AC
0167 008C

I2C_WR_SUB  0x08, DataBegin+1, 0x30

0169 3009
0169 0096
016B 3020
016B 0084
016D 0800
016D 0097
016E 3030
016F 0080
0171 2019
0172 0817
0172 00A0
0174 205F
0175 210F

LOAD_ADDR_8  _Slave_3_Addr

0176 1011
0176 30D6

;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
;
; is Bus Free, ie. has a start & stop bit been
; detected (only for multi master system)
; a very simple test, unused for now

;
; read a byte from slave from current address
;
```



```
0177 008C          movlw   0xCC
0179 30CC          movwf   DataBegin
017A 00A0          I2C_Wr_SUB 0x01,DataBegin, 0x33

017B 3002
017B 0096
017D 301F
017D 0084
017F 0800
017F 0097
0180 3033
0181 0080
0183 2019
0184 0817
0184 009F
0186 205F
0187 2134

0188 0064          self   clrwdt
0189 2988          goto   self

;*****
;
;          call   ReadSlave3      ; Read From Slave PIC
;
;          self   clrwdt
;          goto   self
;*****
END
```

# Software Implementation of I<sup>2</sup>C Bus Master

---

NOTES:

---

---

# WORLDWIDE SALES & SERVICE

---

---

## AMERICAS

### Corporate Office

Microchip Technology Inc.  
2355 West Chandler Blvd.  
Chandler, AZ 85224-6199  
Tel: 602 786-7200 Fax: 602 786-7277  
Technical Support: 602 786-7627  
Web: <http://www.mchip.com/microhip>

### Atlanta

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

### Boston

Microchip Technology Inc.  
5 Mount Royal Avenue  
Marlborough, MA 01752  
Tel: 508 480-9990 Fax: 508 480-8575

### Chicago

Microchip Technology Inc.  
333 Pierce Road, Suite 180  
Itasca, IL 60143  
Tel: 708 285-0071 Fax: 708 285-0075

### Dallas

Microchip Technology Inc.  
14651 Dallas Parkway, Suite 816  
Dallas, TX 75240-8809  
Tel: 214 991-7177 Fax: 214 991-8588

### Dayton

Microchip Technology Inc.  
35 Rockridge Road  
Englewood, OH 45322  
Tel: 513 832-2543 Fax: 513 832-2841

### Los Angeles

Microchip Technology Inc.  
18201 Von Karman, Suite 455  
Irvine, CA 92715  
Tel: 714 263-1888 Fax: 714 263-1338

### New York

Microchip Technology Inc.  
150 Motor Parkway, Suite 416  
Hauppauge, NY 11788  
Tel: 516 273-5305 Fax: 516 273-5335

## AMERICAS (continued)

### San Jose

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

## ASIA/PACIFIC

### Hong Kong

Microchip Technology  
Unit No. 3002-3004, Tower 1  
Metroplaza  
223 Hing Fong Road  
Kwai Fong, N.T. Hong Kong  
Tel: 852 2 401 1200 Fax: 852 2 401 3431

### Korea

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

### Singapore

Microchip Technology  
200 Middle Road  
#10-03 Prime Centre  
Singapore 188980  
Tel: 65 334 8870 Fax: 65 334 8850

### Taiwan

Microchip Technology  
10F-1C 207  
Tung Hua North Road  
Taipei, Taiwan, ROC  
Tel: 886 2 717 7175 Fax: 886 2 545 0139

## EUROPE

### United Kingdom

Arizona Microchip Technology Ltd.  
Unit 6, The Courtyard  
Meadow Bank, Furlong Road  
Bourne End, Buckinghamshire SL8 5AJ  
Tel: 44 0 1628 851077 Fax: 44 0 1628 850259

### France

Arizona Microchip Technology SARL  
2 Rue du Buisson aux Fraises  
91300 Massy - France  
Tel: 33 1 69 53 63 20 Fax: 33 1 69 30 90 79

### Germany

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

### Italy

Arizona Microchip Technology SRL  
Centro Direzionale Colleoni  
Palazzo Pegaso Ingresso No. 2  
Via Paracelso 23, 20041  
Agrate Brianza (MI) Italy  
Tel: 39 039 689 9939 Fax: 39 039 689 9883

## JAPAN

Microchip Technology Intl. Inc.

Benex S-1 6F  
3-18-20, Shin Yokohama  
Kohoku-Ku, Yokohama  
Kanagawa 222 Japan  
Tel: 81 45 471 6166 Fax: 81 45 471 6122

9/22/95

All rights reserved. © 1995, Microchip Technology Incorporated, USA.

Information contained in this publication 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. 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. The Microchip logo and name are registered trademarks of Microchip Technology Inc. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.