Software Implementation of I²C™ Bus Master

INTRODUCTION

This application note describes the software implementation of I²C™ interface routines for the PIC16CXX family of devices. Only the master mode of I²C interface is implemented. This implementation is for a single master communication to multiple slave I²C devices.

Some PIC16CXX devices, such as the PIC16C64 and PIC16C74, have on-chip hardware which implements the I²C 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 I²C Bus specifications and the user is assumed to have an understanding of the I²C Bus. For detailed information on the bus, the user is advised to read the I²C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I²C 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 I²C 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 I²C slaves to the PIC16CXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I²C 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 I²C 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 I²C 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.

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

Message Format

In the high level routines, the basic structure of the message implemented is given. Every I²C 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-SlvAW-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 I²C 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_8" 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 I²C 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 I²C 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 I²C 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 I²C 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.

I²C 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 I²C Bus.

Bit #	Name	Description
0	_Bus_Busy	1 = Start Bit transmitted 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 succesfully completed. 0 = error condition.
5	_Rcv_Success	1 = reception succesfully completed. 0 = error condition.
6	_Fatal_Error	1 = FATAL error occurred. The communication was aborted.
7	_$\overline{\text{ACK}}$_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.

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 0 = 7 bit addressing.
1	_Slave_RW	1 = READ operation 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 I²C_TEST_DEVICE macro. See description of this I²C_TEST_DEVICE macro.
7	_TIME_OUT_	A status bit indicating if a clock is stretched low for more than 1 ms, indicationg a bus error. On this time out, the operation is aborted.

Low Level :

Function Name	Description
InitI²CBus_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 priot to calling this routine.

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.

I²C_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-SlvAW-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
```

I²C_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-SlvAW-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  ;
```

I²C_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 I²C_WR function, except that the starting address of the slave is also specified. For example, while writing to an I²C Memory Device, the sub-addr specifies the starting address of the memory. The I²C_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
I²C_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

I²C_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 I²C_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

I²C_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 I²C_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

I²C_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 I²C 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 : 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_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.

I²C_READ_SUB

Parameters : _BYTES_, _DestPointer_, _SubAddress

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
DestPointer	Data Start Buffer pointer in RAM (file Registers)
SubAddress	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
btfss     _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.

I²C_READ_BYTE or I²C_READ_STATUS

Parameters : _DestPointer_

DestPointer	Data Start Buffer pointer in RAM (file Registers)

Purpose : To read a Status Byte from Slave

Description : Several I²C 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*

I²C_WR_SUB_WR

Parameters : _Bytes1_, _SrcPtr1_, _SubAddr_, _Bytes2_, _SrcPtr2_

Bytes1	Number of bytes in the first data block
SrcPtr1	Starting address of first data block
SubAddr	Sub-address of the slave
Bytes2	Number of bytes in the second data block
SrcPtr2	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 I²C_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 I²C 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*

I²C_WR_SUB_RD

Parameters : _Count1_, _SrcPtr_, _SubAddr_, _Count2_, _DestPtr_

Count1	Length of the source buffer
SrcPtr	Source pointer address
SubAddr	Sub-address of the slave
Count2	Length of the destination buffer
DestPtr	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*

I²C_WR_COM_WR

Parameters : _Count1_, _SrcPtr1_, _Count2_, _SrcPtr2_

Count1	Length of the first data block
SrcPtr1	Source pointer of the first data block
Count2	Length of the second data block
SrcPtr2	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 I²C 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-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P*

APPLICATIONS

The I²C Bus is a simple two wire serial protocol for inter-IC communications. Many peripheral devices (acting as slaves) are available in the market with I²C 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 I²C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I²C bus can be implemented using software.

Author: Amar Palacherla,
Logic Products Division

Appendix A - I²C.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 Appox 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               ; 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
;*******************************************************************************************************

#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
```

```
    movlw   (SLAVE_ADDRESS & 0xff)              ; load low byte of address
    movwf   SlaveAddr                           ; 10 bit addr is 11110XX0
    movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)  ; hi order address
    movwf   SlaveAddr+1

    ENDM

LOAD_ADDR_8   MACRO  SLAVE_ADDRESS

    bcf     10BitAddr                           ; Set for 8 Bit Address Mode
    movlw   (SLAVE_ADDRESS & 0xff)
    movwf   SlaveAddr

    ENDM
```

Appendix B - TEST.ASM

```
  Title        “I2C Master Mode Implemetation”
       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\16Cxx.h”
;
#define _Slave_1_Addr  0xA0           ; Serial EEPROM #1
#define _Slave_2_Addr  0xAC           ; 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
```

```
                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 usesd only RTCC & INT Interrupt
;
MayBeOtherInt:
  NOP
;
RestoreIntStatus:                       ; Restore Interrupt Status
  swapf         SaveStatus,w
  movwf _       status                  ; restore STATUS Reg
  swapf         SaveWReg
  swapf         SaveWReg,w              ; restore WREG
  retfie
;
;*****************************************************************************************************
;                        Include I2C High Level & Low Level Routines if _INCLUDE_HIGH_LEVEL_I2C
        include         “i2c_high.asm”
```

```
  endif

;*****************************************************************************************************
;

ReadSlave1:

;
;  EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
;
       LOAD_ADDR_8  _Slave_1_Addr
wait1:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive    ; See If slave is responding
       goto            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
wait2:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive    ; See If slave is responding
       goto            wait2          ; if stuck for ever, recover from WDT, can use other schemes
       clrwdt
       I2C_READ_SUB  8, DataBegin+1, 0x60


  return
;
;*****************************************************************************************************

ReadSlave3:

       LOAD_ADDR_8  _Slave_3_Addr
wait3:
       I2C_TEST_DEVICE
       btfss   _       SlaveActive    ; See If slave is responding
       goto            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)
```

```
;
;*****************************************************************************************************

FillDataBuf:

  movlw        0x00                ; start address location of EEPROM array
  movwf        DataBegin           ; 1st byte of data to be sent is start address
  movlw        DataBegin+1         ; data starts following address (RAM Pointer)
  movwf _      fsr
  movlw        8                   ; fill RAM with 8 bytes , this data is written to EEPROM (slave)
  movwf        byteCount
  movlw        0x55                ; pattern to fill with is 0x55 & 0xAA
  movwf        HoldData
X1:
  comf         HoldData
  movf         HoldData,w
  movwf _      indf
  incf  _      fsr                 ; point to next location
  decfsz       byteCount
  goto         X1
  return
;
;*****************************************************************************************************
;
;                Main Routine  (Test Program)
;
;             SINGLE  MASTER, MULTIPLE SLAVES
;
;*****************************************************************************************************

Start:
  call         InitI2CBus_Master   ; initialize I2C Bus
  bsf   _      gie                 ; enable global interrupts
;

  call  FillDataBuf                ; fill data buffer with 8 bytes of data (0x55, 0xAA)
;
; 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            ; is Bus Free, ie. has a start & stop  bit been detected (only for multi master system)
  goto         $-1                 ; a very simple test, unused for now

  LOAD_ADDR_8  _Slave_1_Addr
  I2C_WR             0x09, DataBegin
;
; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
```

```
;
    btfsc       Bus_Busy        ; is Bus Free, ie. has a start & stop bit been detected (only for multi master system)
    goto        $-1             ; a very simple test, unused for now

    LOAD_ADDR_8  _Slave_2_Addr
    I2C_WR_SUB   0x08, DataBegin+1, 0x30

    call  ReadSlave1            ; read a byte from slave from current address
  ;
  ;   LOAD_ADDR_8  _Slave_3_Addr
  ;   movlw 0xCC
  ;      movwf  DataBegin
  ;   I2C_WR_SUB   0x01,DataBegin, 0x33
  ;
  ;   call  ReadSlave3          ; Read From Slave PIC
  ;
  ;
    self    clrwdt
      goto  self
  ;
  ;*****************************************************************************************************
  ;
      END
```

Appendix C - LOW.ASM

```
;*************************************************************************************************
;
;                       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:

  bcf   _       rp0
  movf  _       portb,w
  andlw         0xFC            ; do not use BSF & BCF on Port Pins
  movwf _       portb           ; set SDA & SCL to zero. From Now on, simply play with tris
  RELEASE_BUS
  clrf          Bus_Status      ; reset status reg
  clrf          Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
  return
;
;*************************************************************************************************
;                             Send Start Bit
;
;*************************************************************************************************

TxmtStartBit:
       bsf     _rp0            ; select page 1
       bsf     _SDA            ; set SDA high
       bsf     _SCL            ; clock is high
;
; Setup time for a REPEATED START condition (4.7 uS)
;
       call    Delay40uSec     ; only necesry for setup time
;
```

```
        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
  endif
;
        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
```

```
;  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 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 opreation
;
TenBitAddrRd:

  bcf          Slave_RW              ; temporarily set for WR operation
  call         TenBitAddrWR
  btfss        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
;
  btfss _      Txmt_Success          ; if not successful, generate STOP & abort transfer
  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
  movwf        DataByte              ; load addr to DatByte for transmission
  goto         EndTxmtAddr

SevenBitAddr:
  movf         SlaveAddr,W
  movwf        DataByte              ; load addr to DatByte for transmission
  bcf          DataByte,LSB
  btfsc _      Slave_RW              ; if skip then write operation
  bsf          DataByte,LSB          ; Read Operation

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
  retlw        FALSE                 ; Addr Txmt Unsuccessful, so return 0
;
; Address Not Acknowledged, so send STOP bit
;
  call         TxmtStopBit
  retlw        FALSE                 ; 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
; _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          ; make copy of DataByte
       bsf     Txmt_Progress         ; set Bus status for txmt progress
       bcf     Txmt_Success          ; reset status bit
       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           ; defined in I2C.H header file
       movwf   option
    endif

TxmtNextBit:
       clrwdt                        ; clear WDT, set for 18 mSec
       bcf     SCL
       rlf     DataByteCopy          ; MSB first, Note DataByte Is Lost
               bcf     _SDA
       btfsc   c
       bsf     SDA
       call    Delay47uSec           ; guareentee min LOW TIME tLOW & Setup time
       bsf     SCL                   ; set clock high , check if clock is high, else clock being stretched
       call    Delay40uSec           ; guareentee min HIGH TIME tHIGH
  if _CLOCK_STRETCH_CHECK
       bcf     rp0
       clrf    rtcc                  ; clear RTCC
       bcf     rtif                  ; clear any pending flags
       bsf     rtie                  ; elable RTCC Interrupt
       bcf     TIME_OUT_             ; reset timeout error flag
Check_SCL_1:
       btfsc   TIME_OUT_             ; if RTCC timeout or Error then Abort & return
       goto    Bus_Fatal_Error       ; Possible FATAL Error on Bus
```

```
        bcf     rp0
        btfss   SCL             ; if clock not being stretched, it must be high
        goto    Check_SCL_1     ; loop until SCL high or RTCC timeout interrupt
        bcf     rtie            ; Clock good, diable RTCC interrupts
        bsf     rp0
  endif
        decfsz  BitCount
        goto    TxmtNextBit
;
; Check For Acknowledge
;
        bcf     SCL             ; reset clock
        bsf     SDA             ; Release SDA line for Slave to pull down
        call    Delay47uSec     ; guareentee min LOW TIME tLOW & Setup time
        bsf     SCL             ; clock for slave to ACK
        call    Delay40uSec     ; guareentee min HIGH TIME tHIGH
        bcf     rp0             ; select PAGE 0 to test PortB pin SDA
        btfsc   SDA             ; SDA should be pulled low by slave if OK
                goto    _TxmtErrorAck
;
        bsf     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 testted from
;  _Last_Byte_Rcv bit of control reg)
;  Data Received on successful reception is in DataReg register
;
;
;*************************************************************************************************************
;

GetData:
        goto    RcvByte
```

```
;
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             ; defined in I2C.H header file
        movwf   option
  endif

RcvNextBit:
        clrwdt                          ; clear WDT, set for 18 mSec
        bsf     rp0                     ; page 1 for TRIS manipulation
        bcf     SCL
        bsf     SDA                     ; can be removed from loop
        call    Delay47uSec             ; guareentee min LOW TIME tLOW & Setup time
        bsf     SCL                     ; clock high, data sent by slave
        call    Delay40uSec             ; guareentee min HIGH TIME tHIGH
  if _CLOCK_STRETCH_CHECK
        bcf     rp0
        clrf    rtcc                    ; clear RTCC
        bcf     rtif                    ; clear any pending flags
        bsf     rtie                    ; elable RTCC Interrupt
        bcf     TIME_OUT_               ; reset timeout error flag
Check_SCL_2:
        btfsc   TIME_OUT_               ; if RTCC timeout or Error then Abort & return
        goto    Bus_Fatal_Error         ; Possible FATAL Error on Bus
        bcf     rp0
        btfss   SCL                     ; if clock not being stretched, it must be high
        goto    Check_SCL_2             ; loop until SCL high or RTCC timeout interrupt
        bcf     rtie                    ; Clock good, diable RTCC interrupts
        bsf     rp0
  endif
        bcf     rp0                     ; select page 0 to read Ports
        bcf     c
        btfsc   SDA
        bsf     c
;                                       ; TEMP ???? DO 2 out of 3 Majority detect
                rlf     DataByte        ; left shift data ( MSB first)
        decfsz  BitCount
        goto    RcvNextBit
```

```
;
; Generate ACK bit if not last byte to be read,
; if last byte Gennerate NACK        ; do not send ACK on last byte, main routine will send a STOP bit
;
       bsf     rp0
       bcf     SCL
       bcf     SDA                 ; ACK by pulling SDA low
       btfsc   Last_Byte_Rcv
       bsf     SDA                 ; if last byte, send NACK by setting SDA high
       call    Delay47uSec         ; guarantee min LOW TIME tLOW & Setup time
       bsf     SCL
       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 appox 1mS
;   and RTCC times out ( appox 4096 cycles : 256 * 16 – prescaler of 16).
;
;*************************************************************************************************

Bus_Fatal_Error:

; diable 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 occured
       bcf     Txmt_Progress       ; Transmission Is Not in Progress
       bcf     Txmt_Success        ; Transmission Unsuccesful
;
       call    TxmtStopBit         ; Try sending a STOP bit, may be not successful
;
  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
;
;*****************************************************************************
```

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-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 eample 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
```

```
;
;
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
        btfss   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
```

```
        call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
;
_block_wr1_loop:
        btfss           Txmt_Success
        return
        movf            indf,w
        movwf           DataByte                ; start from the first byte starting at _DataPointer_
        incf            fsr
        call            SendData                ; send next byte, bus is our's !
        decfsz          tempCount
        goto    _       block_wr1_loop          ; loop until desired bytes of data transmitted to slave
        return
;
;*****************************************************************************************************
;
;*****************************************************************************************************
;                               I2C_WRITE_SUB
;
;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
;  to a slave device.
;               Eg. : A serial EEPROM would need an address of memory location for Random Writes
;
;  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-SlvAW-A-SubA-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
;
;       Returns :       WREG = 1 on success, else WREG = 0
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;               mode addressing must be set
;
;  COMMENTS :
;               I2C_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)
;
;*****************************************************************************************************

I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

        movlw           (_BYTES_ + 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   (_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-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
;
;  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 addresed 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          (_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
        goto    block_wr1_fail                ; end
        movf    SubAddr,w
        movwf   DataByte                      ; start from the first byte starting at _DataPointer_
        call    SendData                      ; send next byte
        btfss   Txmt_Success
        goto    block_wr1_fail                ; end
        movf    SrcPtr,w
        movwf   DataByte                      ; start from the first byte starting at _DataPointer_
        call    SendData                      ; send next byte
        btfss   Txmt_Success
        goto    block_wr1_fail                ; failed, return 0 in WREG
        goto    block_wr1_pass                ; successful, return 1 in WREG
;
; return back to called routine from either _block_wr1_pass or _block_wr1_fail
;
_block_wr1_fail:
        call    TxmtStopBit                   ; Issue a stop bit for slave to end transmission
        retlw   FALSE
_block_wr1_pass:
        call    TxmtStopBit                   ; Issue a stop bit for slave to end transmission
        retlw   TRUE
;

;*****************************************************************************************************************
;
;                               I2C_WR_MEM_BYTE
```

```
;
;  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 EERPOM 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-SlvAW-A-(SubA+0)-A-D[0]-A-P
;                      Delay 1 mSec              ; The user can chnage this value to desired delay
;              S-SlvAW-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    (_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
                      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 continiously 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++
```

```
                .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             ; -1 because, the last byte is used out of loop
        movlw   DestPointer_
        movwf   fsr                   ; FIFO destination address pointer

        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
        btfsc   Txmt_Success
        goto    block_rd1_loop        ; end
        call    TxmtStopBit           ; Issue a stop bit for slave to end transmission
        retlw   FALSE                 ; Error : may be device not responding
;
_block_rd1_loop:
        call    GetData
        movf    DataByte,w
        movwf   indf                  ; start receiving data, starting at Destination Pointer
        incf    fsr
        decfsz  tempCount
        goto    block_rd1_loop        ; loop until desired bytes of data transmitted to slave
        bsf     Last_Byte_Rcv         ; last byte to rcv, so send NACK
        call    GetData
        movf    DataByte,w
        movwf   indf
        call    TxmtStopBit           ; Issue a stop bit for slave to end transmission
        retlw   TRUE


;*****************************************************************************************************************
;
;                               I2C_READ_SUB
;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
;  Between the sub-addrers 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 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.
;               _BubAddress_    The sub-address of the slave
;
;  Sequence :
;       S-SlvAW-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                ; START address of EEPROM(slave 1)
        call    SendData                ; write sub address
;
; 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 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
        goto    byte_rd1_loop           ; read a byte
        call    TxmtStopBit             ; Issue a stop bit for slave to end transmission
        retlw   FALSE                   ; Error : may be device not responding
_byte_rd1_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
        btfss   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-A-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             ; 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 Data of length _COUNT2_ to destination
;       address pointer
;
;
;  Message Structure :
;             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
;
;  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 & addresss info is needed and then another block of actual data
;       to be displayed is needed.
;
;
;  Message Structure :
;               S-SlvW-A-D1[0]-A-.....A-D1[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_

        movlw   COUNT1_
        movwf   tempCount
        movlw   SourcePointer1_
        movwf   fsr
        call    i2c_block_write
;
```

```
; 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"
```

Appendix E - I²C test.lst

```
MPASM B0.24                                                          PAGE  1
“I2C Master Mode Implemetation”
“Rev 0.1        :  01 Mar 1993”


                          Title           “I2C Master Mode Implemetation”
                          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

00F4 2400            _ClkIn          equ     16000000        ; Input Clock Frequency Of PIC16C71

                       include    “d:\pictools\16Cxx.h”
;
00A0              #define _Slave_1_Addr   0xA0                    ; Serial EEPROM #1
00AC              #define _Slave_2_Addr   0xAC                    ; Serial EEPROM #2
00D6              #define _Slave_3_Addr   0xD6                    ; Slave PIC16CXX
0046              #define _ENABLE_BUS_FREE_TIME   TRUE
0047              #define _CLOCK_STRETCH_CHECK    TRUE
0048              #define _INCLUDE_HIGH_LEVEL_I2C TRUE

                      include    “i2c.h”
                  ;*************************************************************************************************
                  ;             I2C Bus Header File
                  ;*************************************************************************************************

003D 0900            _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 Appox 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           ; 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
        ;*****************************************************************************

#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
```

```
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 _   10BitAddr                                ; Slave has 10 bit address
                               movlw   (SLAVE_ADDRESS & 0xff)
                               movwf   SlaveAddr                                ; load low byte of address
                               movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)   ; 10 bit addr is 11110XX0
                               movwf   SlaveAddr+1                              ; hi order  address

                            ENDM
                        LOAD_ADDR_8     MACRO   SLAVE_ADDRESS

                               bcf _   10BitAddr                                ; Set for 8 Bit Address Mode
                               movlw   (SLAVE_ADDRESS & 0xff)
                               movwf   SlaveAddr
                            ENDM


                         CBLOCK  _End_I2C_Ram
0018 0001                       SaveStatus                                       ; copy of STATUS Reg
0019 0001                       SaveWReg                                         ; copy of WREG
```

```
001A 0001                       byteCount
001B 0001                       HoldData
                        ENDC

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


                         ORG     0x00

0000 2956                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)
                ;
0004 0099                       movwf   SaveWReg               ; Save WREG
0005 0E03                       swapf   status,w               ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098                       movwf   SaveStatus             ; Save STATUS Reg if _CLOCK_STRETCH_CHECK RTCC Interrupts enabled only
                                                               ;    if Clock Stretching is Used
0007 1D0B                       btfss   rtif
0008 280B                       goto    MayBeOtherInt          ; other Interrupts
0009 1791                       bsf     TIME_OUT_              ; MUST set this Flag, can take other desired actions here
000A 110B                       bcf     rtif
                                endif
                ;
                ;  Check For Other Interrupts Here, This program usesd only RTCC & INT Interrupt
                ;
                MayBeOtherInt:
000B 0000                       NOP
                ;
                RestoreIntStatus:                              ; Restore Interrupt Status
000C 0E18                       swapf   SaveStatus,w
000D 0083                       movwf   status                 ; restore STATUS Reg
000E 0E99                       swapf   SaveWReg
```

```
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 eample 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
                        ;
                        ;

                        IsSlaveActive:
0011 1091                       bcf     Slave_RW           ; set for write operation
0012 2057                       call    TxmtStartBit               ; send START bit
0013 206B                       call    Txmt_Slave_Addr            ; if successful, then _Txmt_Success bit is set
                        ;
0014 1311                       bcf     SlaveActive
0015 1F90                       btfss   ACK_Error          ; skip if NACK, device is not present or not responding
0016 1711                       bsf     SlaveActive        ; ACK received, device present & listening
0017 205F                       call    TxmtStopBit
0018 0008                       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:
0019 2057                          call       TxmtStartBit             ; send START bit
```

```
001A 1091                   bcf         Slave_RW                ; set for write operation
001B 206B                   call        Txmt_Slave_Addr         ; if successful, then _Txmt_Success bit is set
                          ;
                          _block_wr1_loop:
001C 1E10                   btfss       Txmt_Success
001D 0008                   return
001E 0800                   movf        indf,w
001F 008E                   movwf       DataByte                ; start from the first byte starting at DataPointer_
0020 0A84                   incf        fsr
0021 2095                   call        SendData                ; send next byte, bus is our's !
0022 0B96                   decfsz      tempCount
0023 281C                   goto        block_wr1_loop          ; loop until desired bytes of data transmitted to slave
0024 0008                   return
                    ;
                    ;*****************************************************************************************************

                    ;*****************************************************************************************************
                    ;                           I2C_WRITE_SUB
                    ;
                    ;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address  to a slave device.
                    ;  Eg. : A serial EEPROM would need an address of memory location for Random Writes
                    ;  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-SlvAW-A-SubA-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
                    ;
                    ;       Returns :       WREG = 1 on success, else WREG = 0
                    ;
                    ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers,
                    ;         and 10 or 8 bit mode addressing must be set
                    ;
                    ;  COMMENTS :
                    ;  I2C_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)
                    ;
                    ;*****************************************************************************************************

                    I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                        movlw   (_BYTES_ + 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   (_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-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
;
;  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 addresed 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    (_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:
0025 2057           call    TxmtStartBit       ; send START bit
0026 1091           bcf     Slave_RW           ; set for write operation
0027 206B           call    Txmt_Slave_Addr    ; if successful, then _Txmt_Success bit is set
0028 1E10           btfss   Txmt_Success
0029 2835           goto    block_wr1_fail     ; end
002A 0814           movf    SubAddr,w
002B 008E           movwf   DataByte           ; start from the first byte starting at DataPointer_
002C 2095           call    SendData           ; send next byte
002D 1E10           btfss   Txmt_Success
002E 2835           goto    block_wr1_fail     ; end
002F 0815           movf    SrcPtr,w
0030 008E           movwf   DataByte           ; start from the first byte starting at DataPointer_
0031 2095           call    SendData           ; send next byte
0032 1E10           btfss   Txmt_Success
0033 2835           goto    block_wr1_fail     ; failed, return 0 in WREG
0034 2837           goto    block_wr1_pass     ; successful, return 1 in WREG
                    ;
                    ; return back to called routine from either _block_wr1_pass or block_wr1_fail
                    block_wr1_fail:
0035 205F           call    TxmtStopBit        ; Issue a stop bit for slave to end transmission
0036 3400           retlw   FALSE
                    block_wr1_pass:
0037 205F           call    TxmtStopBit        ; Issue a stop bit for slave to end transmission
0038 3401           retlw    TRUE
                    ;


;*****************************************************************************************************

                    ;
                    ;                               I2C_WR_MEM_BYTE
                    ;
```

```
;  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 EERPOM 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-SlvAW-A-(SubA+0)-A-D[0]-A-P
;      Delay 1 mSec       ; The user can chnage this value to desired delay
;              S-SlvAW-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    (_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
        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 continiously 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++
        .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          ; -1 because, the last byte is used out of loop
                        movlw   DestPointer_
                        movwf   fsr                 ; FIFO destination address pointer

                        call    i2c_block_read
                        ENDM

                        _i2c_block_read:
0039 2057               call    TxmtStartBit        ; send START bit
003A 1491               bsf     Slave_RW            ; set for read operation
003B 1111               bcf     Last_Byte_Rcv       ; not a last byte to rcv
003C 206B               call    Txmt_Slave_Addr     ; if successful, then _Txmt_Success bit is set
003D 1A10               btfsc   Txmt_Success
003E 2841               goto    block_rd1_loop      ; end
003F 205F               call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
0040 3400               retlw   FALSE               ; Error : may be device not responding
                                                    ;
                        _block_rd1_loop:
0041 20CC               call    GetData
```

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

0044 0A84                incf    fsr
0045 0B96                decfsz  tempCount
0046 2841                goto    block_rd1_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 preceeded by a write of the sub-address Between the
;  sub-addrers 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.
;       BubAddress_     The sub-address of the slave
;
;  Sequence :
;       S-SlvAW-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               ; START address of EEPROM(slave 1)
    call     SendData               ; write sub address
;
; 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 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
    goto   byte_rd1_loop        ; read a byte
    call   TxmtStopBit          ; Issue a stop bit for slave to end transmission
    retlw  FALSE                ; Error : may be device not responding
    _byte_rd1_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
    btfss  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-A-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             ; 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 Data of length _COUNT2_ to
;  destination address pointer
;
;
;  Message Structure :
;  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
;
;  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
;  block of control & addresss info is needed and then another block of actual
;  data to be displayed is needed.
;
;
;  Message Structure :
;           S-SlvW-A-D1[0]-A-.....A-D1[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_

movlw           COUNT1_
movwf           tempCount
movlw           SourcePointer1_
movwf           fsr
call            i2c_block_write
;
; 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”
;*****************************************************************************************************
```

```
                    ;
                    ;             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        ; do not use BSF & BCF on Port Pins
0050 0086               movwf       portb       ; set SDA & SCL to zero. From Now on, simply play with tris
                                                ; RELEASE_BUS
0051 1683
0051 1486
0052 1406
0054 0190               clrf    Bus_Status      ; reset status reg
0055 0191               clrf    Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008               return
                    ;
                    ;*************************************************************************************************
                    ;                 Send Start Bit
                    ;

                    ;*************************************************************************************************

                       TxmtStartBit:
0057 1683               bsf     rp0             ; select page 1
0058 1486               bsf     SDA             ; set SDA high
0059 1406               bsf     SCL             ; clock is high
                                                ;
                                                ; Setup time for a REPEATED START condition (4.7 uS)
                                                ;
005A 210D               call    Delay40uSec     ; only necesry for setup time
                                                ;
005B 1086               bcf     SDA             ; give a falling edge on SDA while clock is high
                                                ;
005C 210B               call    Delay47uSec     ; only necessary for START HOLD time
                                                ;
005D 1410               bsf     Bus_Busy        ; on a start condition bus is 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
```

```
                ;  Write & 1 for Read
                ;  On Success, return TRUE in WREG, else FALSE in WREG
                ;
                ;   If desired, the failure may tested by the bits in Bus Status Reg
                ;
                ;*************************************************************************************************

                Txmt_Slave_Addr:
006B 1390           bcf         ACK_Error               ; reset Acknowledge error bit
006C 1C11           btfss       10BitAddr
006D 2886           goto        SevenBitAddr
                                                        ;
006E 1C91           btfss       Slave_RW
006F 287D           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 opreation
                                                        ;
                TenBitAddrRd:

0070 1091           bcf         Slave_RW                ; temporarily set for WR operation
0071 207D           call        TenBitAddrWR
0072 1E10           btfss       Txmt_Success            ; skip if successful
0073 3400           retlw       FALSE
0074 2057           call        TxmtStartBit            ; send A REPEATED START condition
0075 1491           bsf         Slave_RW                ; For 10 bit slave Read

0076 080D           movf        SlaveAddr+1,W
0077 008E           movwf       DataByte
0078 140E           bsf         DataByte,LSB            ; Read Operation
0079 2095           call        SendData                ; send ONLY high byte of 10 bit addr slave
007A 288C           goto        AddrSendTest            ; 10 Bit Addr Send For Slave Read Over
                                                        ;
                                                        ; if successfully transmitted, expect an ACK bit
                                                        ;
007B 1E10           btfss       Txmt_Success            ; if not successful, generate STOP & abort transfer
007C 2890           goto    _   AddrSendFail
                                                        ;

                TenBitAddrWR:

007D 080D           movf    SlaveAddr+1,W
007E 008E           movwf   DataByte
007F 100E           bcf     DataByte,LSB                ; WR Operation
                                                        ;
                                                        ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW
                                                        ; Enabled) then save RETURN Address Pointer
                                                        ;
```

```
0080 2095          call    SendData                    ; send high byte of 10 bit addr slave
                                                       ;
                                                       ; if successfully transmitted, expect an ACK bit
                                                       ;
0081 1E10          btfss   Txmt_Success                ; if not successful, generate STOP & abort transfer
0082 2890          goto    AddrSendFail
                                                       ;
0083 080C          movf    SlaveAddr,W
0084 008E          movwf   DataByte                    ; load addr to DatByte for transmission
0085 288B          goto    EndTxmtAddr

                SevenBitAddr:
0086 080C          movf    SlaveAddr,W
0087 008E          movwf   DataByte                    ; load addr to DatByte for transmission
0088 100E          bcf     DataByte,LSB
0089 1891          btfsc   Slave_RW                    ; if skip then write operation
008A 140E          bsf     DataByte,LSB                ; Read Operation

                EndTxmtAddr:
008B 2095          call    SendData                    ; send 8 bits of address, bus is our's
                                                       ;
                                                       ; if successfully transmitted, expect an ACK bit
                                                       ;
                _AddrSendTest:
008C 1E10          btfss   Txmt_Success                ; skip if successful
008D 2890          goto    AddrSendFail
008E 0064          clrwdt
008F 3401          retlw   TRUE
                                                       ;
                _AddrSendFail:
0090 0064          clrwdt
0091 1F90          btfss   ACK_Error
0092 3400          retlw   FALSE                       ; Addr Txmt Unsuccessful, so return 0
                                                       ;
                                                       ; Address Not Acknowledged, so send STOP bit
                                                       ;
0093 205F          call    TxmtStopBit
0094 3400          retlw   FALSE                       ; 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
```

```
                ;   _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
                ;
0095 2896           goto    TxmtByte

                                                ;
                TxmtByte:
0096 080E           movf    DataByte,w
0097 0093           movwf   DataByteCopy        ; make copy of DataByte
0098 1510           bsf     Txmt_Progress       ; set Bus status for txmt progress
0099 1210           bcf     Txmt_Success        ; reset status bit
009A 3008           movlw   0x08
009B 008F           movwf   BitCount
009C 1683           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
                                                ;
009D 0801           movf    option,w
009E 39C3           andlw   OPTION_INIT         ; defined in I2C.H header file
009F 0081           movwf   option
                    endif

                TxmtNextBit:
00A0 0064           clrwdt                      ; clear WDT, set for 18 mSec
00A1 1006           bcf     SCL
00A2 0D93           rlf     DataByteCopy        ; MSB first, Note DataByte Is Lost
00A3 1086           bcf     SDA
00A4 1803           btfsc   c
00A5 1486           bsf     SDA
00A6 210B           call    Delay47uSec         ; guarantee min LOW TIME tLOW & Setup time
00A7 1406           bsf     SCL                 ; set clock high , check if clock is high, else
                                                ; clock being stretched
00A8 210D           call    Delay40uSec         ; guareentee min HIGH TIME tHIGH
                    if _CLOCK_STRETCH_CHECK
00A9 1283           bcf     rp0
00AA 0181           clrf    rtcc                ; clear RTCC
00AB 110B           bcf     rtif                ; clear any pending flags
00AC 168B           bsf     rtie                ; elable RTCC Interrupt
00AD 1391           bcf     TIME_OUT_           ; reset timeout error flag
                Check_SCL_1:
00AE 1B91           btfsc   TIME_OUT_           ; if RTCC timeout or Error then Abort & return
00AF 28FC           goto    Bus_Fatal_Error     ; Possible FATAL Error on Bus
```

```
00B0 1283     bcf    rp0                     ; if clock not being stretched, it must be high
00B1 1C06     btfss  SCL                     ; loop until SCL high or RTCC timeout interrupt
00B2 28AE     goto   Check_SCL_1             ; clock good, disable RTCC interrupts
00B3 128B     bcf    rtie
00B4 1683     bsf    rp0

              endif
00B5 0B8F     decfsz BitCount
00B6 28A0     goto   TxmtNextBit
                                             ;
                                             ; Check For Acknowledge
                                             ;
00B7 1006     bcf    SCL                     ; reset clock
00B8 1486     bsf    SDA                     ; Release SDA line for Slave to pull down
00B9 210B     call   Delay47uSec             ; guareentee min LOW TIME tLOW & Setup time
00BA 1406     bsf    SCL                     ; clock for slave to ACK
00BB 210D     call   Delay40uSec             ; guareentee min HIGH TIME tHIGH
00BC 1283     bcf    rp0                     ; select PAGE 0 to test PortB pin SDA
00BD 1886     btfsc  SDA                     ; SDA should be pulled low by slave if OK
00BE 28C5     goto   TxmtErrorAck
                                             ;
00BF 1683     bsf    rp0
00C0 1006     bcf    SCL                     ; reset clock

00C1 1110     bcf    Txmt_Progress           ; reset TXMT bit in Bus Status
00C2 1610     bsf    Txmt_Success            ; transmission successful
00C3 1390     bcf    ACK_Error               ; ACK OK
00C4 0008     return
              TxmtErrorAck:
              RELEASE_BUS

00C5 1683
00C5 1486
00C6 1406
00C8 1110     bcf    Txmt_Progress           ; reset TXMT bit in Bus Status
00C9 1210     bcf    Txmt_Success            ; transmission NOT successful
00CA 1790     bsf    ACK_Error               ; No ACK From Slave
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
              ;
              ;
```

```
;*****************************************************************************************************************
;
;
GetData:
   goto   RcvByte
;
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              ; defined in I2C.H header file
   andlw  OPTION_INIT
   movwf  option
      endif

   RcvNextBit:
   clrwdt                       ; clear WDT, set for 18 mSec
   bsf    rp0                   ; page 1 for TRIS manipulation
   bcf    SCL
   bsf    SDA                   ; can be removed from loop
   call   Delay47uSec           ; guarantee min LOW TIME tLOW & Setup time
   bsf    SCL                   ; clock high, data sent by slave
   call   Delay40uSec           ; guarantee min HIGH TIME tHIGH
         if _CLOCK_STRETCH_CHECK
   bcf    rp0                   ; clear RTCC
   clrf   rtcc
   bcf    rtif                  ; clear any pending flags
   bsf    rtie                  ; elable RTCC Interrupt
   bcf    TIME_OUT_             ; reset timeout error flag

      Check_SCL_2:
   btfsc  TIME_OUT_             ; if RTCC timeout or Error then Abort & return
   goto   Bus_Fatal_Error       ; Possible FATAL Error on Bus
   bcf    rp0
   btfss  SCL                   ; if clock not being stretched, it must be high
   goto   Check_SCL_2           ; loop until SCL high or RTCC timeout interrupt
   bcf    rtie                  ; Clock good, diable RTCC interrupts
   bsf    rp0
      endif

   bcf    rp0                   ; select page 0 to read Ports
   bcf    c
   btfsc  SDA
   bsf    c
```

```
                ;                           ; TEMP ???? DO 2 out of 3 Majority detect
00EC 0D8E           rlf     DataByte            ; left shift data ( MSB first)
00ED 0B8F           decfsz  BitCount
00EE 28D5           goto    RcvNextBit
                                            ;
                                            ; 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           bsf     rp0
00F0 1006           bcf     SCL
00F1 1086           bcf     SDA                 ; ACK by pulling SDA low
00F2 1911           btfsc   Last_Byte_Rcv
00F3 1486           bsf     SDA                 ; if last byte, send NACK by setting SDA high
00F4 210B           call    Delay47uSec         ; guarantee min LOW TIME tLOW & Setup time
00F5 1406           bsf     SCL
00F6 210D           call    Delay40uSec         ; guarantee min HIGH TIME tHIGH
                      RcvEnd:
00F7 1006           bcf     SCL                 ; reset clock
00F8 1190           bcf     Rcv_Progress        ; reset TXMT bit in Bus Status
00F9 1690           bsf     Rcv_Success         ; transmission successful
00FA 1390           bcf     ACK_Error           ; ACK OK
00FB 0008       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 (appox 4096 cycles : 256 * 16 –
                ;  prescaler of 16).
                      ;*************************************************************************************************

                      Bus_Fatal_Error:

                      ; disable RTCC Interrupt
                      ;
00FC 128B             bcf     rtie                    ; disable RTCC interrupts, until next TXMT try

                RELEASE_BUS
00FD 1683
00FD 1486
00FE 1406
                      ;
                      ;  Set the Bus_Status Bits appropriately
                      ;
0100 1490             bsf     Abort                   ; transmission was aborted
0101 1710             bsf     Fatal_Error             ; FATAL Error occured
0102 1110             bcf     Txmt_Progress           ; Transmission Is Not in Progress
0103 1210             bcf     Txmt_Success            ; Transmission Unsuccesful
```

```
                        ;
0104 205F                   call    TxmtStopBit                 ; Try sending a STOP bit, may be not successful
                        ;
0105 0008                         return
                        ;
                        ;*****************************************************************************************************
                                  endif
                        ;*****************************************************************************************
                        ;         General Purpose Delay Routines
                        ;
                        ;  Delay4uSis wait loop for 4.0 uSec
                        ;  Delay47uS   is wait loop for 4.7 uSec
                        ;  Delay50uS   is wait loop for 5.0 uSec
                        ;
                        ;*****************************************************************************************************
                        ;

                        Delay50uSec:
0106 3006                   movlw       ((_50uS_Delay-5)/3 + 1)
                            DlyK
0107 0092                   movwf       DelayCount
0108 0B92                   decfsz      DelayCount
0109 2908                   goto        $-1
010A 0008                   return
                        ;
                        Delay47uSec:
010B 3004                   movlw       ((47uS_Delay-8)/3 + 1)
010C 2907                   goto        DlyK
                        ;
                        Delay40uSec:
010D 3003                   movlw       ((_40uS_Delay-8)/3 + 1)
010E 2907                   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
```

```
                        wait1:
                    I2C_TEST_DEVICE

0112 2011                   btfss   SlaveActive     ; See If slave is responding
0113 1F11                   goto    wait1           ; if stuck for ever, recover from WDT, can use other schemes
0114 2912                   clrwdt
0115 0064           I2C_READ_SUB  8, DataBegin+1, 0x50

0116 1091
0116 2057
0117 206B
0119 3050
0119 008E
011A 2095
011C 3007
011C 0096
011D 3021
011E 0084
0120 2039

                    ;
                    ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
                    ;
                    LOAD_ADDR_8   _Slave_2_Addr

0121 1011
0121 30AC
0122 008C

                        wait2:
                    I2C_TEST_DEVICE

0124 2011                   btfss   SlaveActive     ; See If slave is responding
0125 1F11                   goto    wait2           ; if stuck for ever, recover from WDT, can use other schemes
0126 2924                   clrwdt
0127 0064           I2C_READ_SUB  8, DataBegin+1, 0x60

0128 1091
0128 2057
0129 206B
012B 3060
012B 008E
```

```
012C 2095
012E 3007
012E 0096
012F 3021
0130 0084
0132 2039
0133 0008                       return
                        ;
                        ;*****************************************************************************************

                                ReadSlave3:

                        LOAD_ADDR_8  _Slave_3_Addr

0134 1011
0134 30D6
0135 008C

                                wait3:
                        I2C_TEST_DEVICE

0137 2011
0138 1F11                       btfss   SlaveActive             ; See If slave is responding
0139 2937                       goto    wait3                   ; if stuck for ever, recover from WDT, can use other schemes
013A 0064                       clrwdt
                        I2C_READ_SUB  8, DataBegin, 0

013B 1091
013B 2057
013C 206B
013E 3000
013E 008E
013F 2095
0141 3007
0141 0096
0142 3020
0143 0084
0145 2039

                        ;

0146 0008               return

                        ;*****************************************************************************************
                        ;
                        ;       Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
                        ;
                        ;*****************************************************************************************
```

```
                            FillDataBuf:

0147 3000                   movlw   0x00                ; start address location of EEPROM array
0148 00A0                   movwf   DataBegin           ; 1st byte of data to be sent is start address
0149 3021                   movlw   DataBegin+1         ; data starts following address (RAM Pointer)
014A 0084                   movwf   fsr
014B 3008                   movlw   8                   ; fill RAM with 8 bytes , this data is written to
                                                      ; EEPROM (slave)
014C 009A                   movwf   byteCount
014D 3055                   movlw   0x55                ; pattern to fill with is 0x55 & 0xAA
014E 009B                   movwf   HoldData
                            X1:
014F 099B                   comf    HoldData
0150 081B                   movf    HoldData,w
0151 0080                   movwf   indf
0152 0A84                   incf    fsr                 ; point to next location
0153 0B9A                   decfsz  byteCount
0154 294F                   goto    X1
0155 0008                   return

                ;
                ;*****************************************************************************************************
                ;
                ;             Main Routine  (Test Program)
                ;
                ;          SINGLE  MASTER, MULTIPLE SLAVES
                ;
                ;*****************************************************************************************************

                            Start:
0156 204D                   call    InitI2CBus_Master   ; initialize I2C Bus
0157 178B                   bsf     gie                 ; enable global interrupts
                                                        ;
0158 2147                   call    FillDataBuf         ; fill data buffer with 8 bytes of data (0x55, 0xAA)
                                                        ;
                ; 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
                ;

0159 1810                       btfsc   Bus_Busy        ; is Bus Free, ie. has a start & stop  bit been
                                                        ; detected (only for multi master system)
015A 2959                       goto    $-1             ; a very simple test, unused for now

                LOAD_ADDR_8  _Slave_1_Addr

015B 1011
```

```
015B 30A0
015C 008C


                I2C_WR      0x09, DataBegin

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

                                                        ;
                                                        ; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
                                                        ;
0164 1810                       btfsc   Bus_Busy        ; is Bus Free, ie. has a start & stop bit been
                                                        ; detected (only for multi master system)
0165 2964                       goto    $-1             ; a very simple test, unused for now

                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                       call    ReadSlave1      ; read a byte from slave from current address
                                                        ;
                LOAD_ADDR_8  _Slave_3_Addr

0176 1011
0176 30D6
```

```
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          call   ReadSlave3          ; Read From Slave PIC
                   ;
                   ;
0188 0064          self   clrwdt
0189 2988          goto   self
                  ;
                  ;*****************************************************************************
                   END
```

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