AN554

Software Implementation of I²C™ Bus Master

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

This application note describes the software implementation of I²C interface routines for the PIC16CXXX family of devices. Only the master mode of I²C interface is implemented in this application note. This implementation is for a single master communication to multiple slave I²C devices.

Some PIC16CXXX devices, such as the PIC16C64 and PIC16C74, have on-chip hardware which implements the I²C slave interface, while other PIC16CXXX devices, such as the PIC16C71 and PIC16C84, do not have the same 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 possible masters and multiple possible slaves connected to each other through two wires. The two wires consists of a clock line (SCL) and a data line (SDA) with both 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 PIC16CXXX (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 and 24LC01).

IMPLEMENTATION

Two levels of software routines are provided. The low-level routines "`i2c_low.asm`" are provided in Appendix A and the high level routines "`i2c_high.asm`" are provided in Appendix B.

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 in Table 1.

TABLE 1: DESCRIPTION OF ABBREVIATIONS USED

Abbreviations	Explanation
S	Start Condition
P	Stop Condition
SlvAR	Slave Address (for read operation)
SlvAW	Slave Address (for write operation)
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 is given. Every I^2C 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 I^2C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD_ADDR_8" (for 7-bit address slaves) or "LOAD_ADDR_10" macro (for 10-bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7- or 10-bit addressing modes. See the macros section for more details.

CLOCK STRETCHING

In the I^2C Bus, the clock (SCL line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built-in wait state for the I^2C Bus. This feature is implemented and can be turned on or off as an assembly time option (by configuring the _ENABLE_BUS_FREE_TIME flag to be TRUE or FALSE). If the clock is held low for too long, say 1 ms, then an error condition is assumed and a T0CKI interrupt is generated.

ARBITRATION

The I^2C 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 is 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 the SCL line and RB1 as the SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with '0's. To emulate the high state (passive), these lines are configured as inputs. To emulate the active low state, the pins are configured as outputs (with the assumption of having external pull-up resistors on both lines).

For devices that have the on-chip I^2C 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 the clock line, SCL, is stuck low.
2	_Txmt_Progress	1 = transmission in progress
3	_Rcv_Progress	1 = reception in progress
4	_Txmt_Success	1 = transmission successfully completed 0 = error condition
5	_Rcv_Success	1 = reception successfully completed 0 = error condition
6	_Fatal_Error	1 = FATAL error occurred (the communication was aborted).
7	_$\overline{\text{ACK}}$_Error	1 = slave sent $\overline{\text{ACK}}$ while the master was expecting an 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 $\overline{\text{ACK}}$.
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, indicating a bus error. On this time out, the operation is aborted.

Lower Level Routines

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 prior 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 previously. In most cases only a few of the high level routines may be needed and the user can remove or not include the routines that are 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. If the slave under test is present, then "_SlaveActive" status bit (in Bus_Control file register) is set. If not, then this bit is cleared, 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 (# 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 I^2C Memory Device, the sub-addr specifies the starting address of the memory. The I^2C 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 ms
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
Delay 1 ms
•
•
•
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 I^2C 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 $\overline{ACK}$ 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_SUB    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 (# 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 another 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      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 0x60h) 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 I^2C 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, W register = '1' else W register = '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 I^2C 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 I^2C 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 I^2C Bus is a simple two wire serial protocol for inter-IC communications. Many peripheral devices (acting as slaves) are available in the market with I^2C interface (e.g., serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters). Although some of the PIC16CXXX devices do not have on-chip I^2C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I^2C bus can be implemented using software.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: I²C_TEST.H

```
MPASM 01.40.01 Intermediate  I2C_TEST.ASM   4-4-1997  14:47:18         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         Title           "I2C Slave Mode Implemetation"
                     00002         SubTitle        "Rev 0.2        :  04 April 1997"
                     00003
                     00004
                     00005
                     00006 ;******************************************************************
                     00007 ;
                     00008 ;                       I2C Slave Mode Using Software Polling
                     00009 ;
                     00010 ;  Start Bit Is detected by connecting SDA to RB0/INT Pin
                     00011 ;  Other bits, including STOP & Repeated Start Conditions are Software Polled
                     00012 ;
                     00013 ;    The software is implemented using PIC16C84 & thus can be ported to all
                     00014 ;       Enhanced core PIC16CXX products
                     00015 ;
                     00016 ;RB1 is SCL      (Any I/O Pin May Be used instead)
                     00017 ; RB0/INT is SDA (Must use this pin for START bit detect when in idle mode)
                     00018 ;
                     00019 ;   Since Slave Mode is always Application Dependent, as a Test Program, PIC16C84 is used to
                     00020 ;    emulate partial functionality of Microchip's 24Cxx Serial EEPROMs
                     00021 ;
                     00022 ;
                     00023 ;       Program:          I2C_TEST.ASM
                     00024 ;       Revision Date:    Rev 0.1       :  01 Mar 1993
                     00025 ;                         4-04-97      Compatibility with MPASMWIN 1.40
                     00026 ;
                     00027 ;******************************************************************
                     00028 ;
                     00029 #define _MY_ADDRESS     0xD6    ; This slave's address
                     00030 ;
                     00031 #define AtoD_Slave      1
                     00032 #define EE_Slave        0
                     00033
                     00034 ;
```

```
                    00035
                    00036                LIST    p = 16C71
                    00037                ERRORLEVEL  - 302
                    00038
                    00039                Radix   DEC
                    00040                EXPAND
                    00041
00F42400            00042 _ClkIn         equ     16000000
003D0900            00043 _ClkOut        equ     (_ClkIn >> 2)
                    00044 ;
                    00045 ;
00000020            00046 ControlByte    EQU     0x20
                    00047 #define        _STOP   ControlByte, 0
                    00048 #define        _START  ControlByte, 1
                    00049 #define        _RW     ControlByte, 2
                    00050 #define        _ACK    ControlByte, 3
00000021            00051 SlvStatus      EQU     0x21
                    00052 ;
                    00053 ;
                    00054
                    00055                include         “p16c71.inc”
                    00001        LIST
                    00002 ; P16C71.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                    00142        LIST
                    00056
00000008            00057 _eedata set    0x08
00000009            00058 _eeadr  set    0x09
                    00059
00000008            00060 _eecon1 set    0x08
00000009            00061 _eecon2 set    0x09            ; bank 1
                    00062
                    00063 #define        _rd     _eecon1,0
                    00064 #define        _wr     _eecon1,1
                    00065 #define        _wren   _eecon1,2
                    00066 #define        _wrerr  _eecon1,3
                    00067 #define        _eeif   _eecon1,4
                    00068
00000000            00069 LSB     equ    0
00000007            00070 MSB     equ    7
                    00071
                    00072                include “i2c.h”
                    00001 ;*******************************************************************************************************
                          *
                    00002 ;                               I2C Bus Header File
                    00003 ;*******************************************************************************************************
                          *
                    00004
```

```
003D0900          00005 _ClkOut         equ     (_ClkIn >> 2)
                  00006
                  00007 ;
                  00008 ; Compute the delay constants for setup & hold times
                  00009 ;
00000010          00010 _40uS_Delay     set     (_ClkOut/250000)
00000012          00011 _47uS_Delay     set     (_ClkOut/212766)
00000014          00012 _50uS_Delay     set     (_ClkOut/200000)
                  00013
                  00014 #define _OPTION_INIT    (0xC0 | 0x03)          ; Prescaler to TMR0 for Appox 1 mSec timeout
                  00015 ;
                  00016 #define _SCL    PORTB,0
                  00017 #define _SDA    PORTB,1
                  00018
                  00019 #define _SCL_TRIS       _trisb,0
                  00020 #define _SDA_TRIS       _trisb,1
                  00021
                  00022 #define _WRITE_         0
                  00023 #define _READ_          1
                  00024
                  00025
                  00026 ;                                       Register File Variables
                  00027
                  00028
                  00029                 CBLOCK  0x0C
0000000C          00030                         SlaveAddr               ; Slave Addr must be loaded into this reg
0000000D          00031                         SlaveAddrHi             ; for 10 bit addressing mode
0000000E          00032                         DataByte                ; load this reg with the data to be transmitted
0000000F          00033                         BitCount                ; The bit number (0:7) transmitted or received
00000010          00034                         Bus_Status              ; Status Reg of I2C Bus for both TXMT & RCVE
00000011          00035                         Bus_Control             ; control Register of I2C Bus
00000012          00036                         DelayCount
00000013          00037                         DataByteCopy            ; copy of DataByte for Left Shifts (destructive)
                  00038
00000014          00039                         SubAddr                 ; sub-address of slave (used in I2C_HIGH.ASM)
00000015          00040                         SrcPtr                  ; source pointer for data to be transmitted
                  00041
00000016          00042                         tempCount               ; a temp variable for scratch RAM
00000017          00043                         StoreTemp_1             ; a temp variable for scratch RAM, do not disturb contents
                  00044
00000018          00045                         _End_I2C_Ram            ; unused, only for ref of end of RAM allocation
                  00046                 ENDC
                  00047
                  00048 ;*****************************************************************************
                  00049 ;                       I2C Bus Status Reg Bit Definitions
                  00050 ;*****************************************************************************
                  00051
```

```
00052 #define _Bus_Busy       Bus_Status,0
00053 #define _Abort          Bus_Status,1
00054 #define _Txmt_Progress  Bus_Status,2
00055 #define _Rcv_Progress   Bus_Status,3
00056
00057 #define _Txmt_Success   Bus_Status,4
00058 #define _Rcv_Success    Bus_Status,5
00059 #define _Fatal_Error    Bus_Status,6
00060 #define _ACK_Error      Bus_Status,7
00061
00062 ;*******************************************************************************
00063 ;                       I2C Bus Control Register
00064 ;*******************************************************************************
00065 #define _10BitAddr      Bus_Control,0
00066 #define _Slave_RW       Bus_Control,1
00067 #define _Last_Byte_Rcv  Bus_Control,2
00068
00069 #define _SlaveActive    Bus_Control,6
00070 #define _TIME_OUT_      Bus_Control,7
00071
00072
00073
00074
00075 ;*****************************************************************************************************
      *
00076 ;                               General Purpose Macros
00077 ;*****************************************************************************************************
      *
00078
00079 RELEASE_BUS     MACRO
00080                         bsf     STATUS,RP0      ; select Bank1
00081                         bsf     _SDA            ; tristate SDA
00082                         bsf     _SCL            ; tristate SCL
00083 ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
00084                 ENDM
00085
00086 ;*****************************************************************************************************
      *
00087 ;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
00088 ;
00089 ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
00090 ;  on 8 or 10 bit addressing modes
00091 ;*****************************************************************************************************
      *
00092
00093 LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
00094
```

```
                    00095                bsf     _10BitAddr                                  ; Slave has 10 bit address
                    00096                movlw   (SLAVE_ADDRESS & 0xff)
                    00097                movwf   SlaveAddr                                   ; load low byte of address
                    00098                movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)      ; 10 bit addr is 11110XX0
                    00099                movwf   SlaveAddr+1                                 ; hi order  address
                    00100
                    00101                ENDM
                    00102
                    00103 LOAD_ADDR_8    MACRO   SLAVE_ADDRESS
                    00104
                    00105                bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                    00106                movlw   (SLAVE_ADDRESS & 0xff)
                    00107                movwf   SlaveAddr
                    00108
                    00109                ENDM
                    00073
                    00074
                    00075                CBLOCK  _End_I2C_Ram
  00000018          00076                        SaveStatus                                  ; copy Of STATUS Reg
  00000019          00077                        SaveW register                              ; copy of W register
  0000001A          00078                        byteCount
  0000001B          00079                        HoldData
                    00080                ENDC
                    00081
                    00082                CBLOCK  0x20
  00000020          00083                        DataBegin                                   ; Data to be read or written is stored here
                    00084                ENDC
                    00085
                    00086 ;*************************************************************************************************
                    00087
0000                00088                ORG     0x00
                    00089
0000 2815           00090                goto    Start
                    00091 ;
0004                00092                ORG     0x04
                    00093 ;*************************************************************************************************
                    00094 ;                               Interrupt Service Routine
                    00095 ;
                    00096 ;   For I2C Slave routines, only Rb0/INT interrupt is used for START Bit Detect From
                    00097 ;       Idle Mode
                    00098 ;
                    00099 ;*************************************************************************************************
                    00100
0004                00101 Interrupt:
                    00102 ;
                    00103 ;  At first check if START Bit Detect (currently only INT pin interrupt is enabled, if other
                    00104 ;       Interrupts are enabled, then check for other interrupts)
```

```
                    00105 ;
0004 1C8B           00106         btfss   INTCON,INTF
0005 0009           00107         retfie                          ; other interrupt, simply return & enable GIE
                    00108 ;
                    00109 ; Save Status
                    00110 ;
0006 0099           00111         movwf   SaveW register
0007 0E03           00112         swapf   STATUS,W                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0008 0098           00113         movwf   SaveStatus
                    00114 ;
0009 1283           00115         bcf     STATUS,RP0
000A 1C06           00116         btfss   _SCL          ; Most likely a START Bit, Hold Time of START Bit must be valid from an INT to here
000B 280F           00117         goto    RestoreStatus           ; Not a Valid START Bit
000C 1886           00118         btfsc   _SDA                    ; If a valid Falling Edge on SDA when SCL is high, SDA must now be Low
000D 280F           00119         goto    RestoreStatus           ; Not a Valid START Bit
000E 282A           00120         goto    StartBitDetect        ; A Valid Start Bit Is Detected, process & then Branch to “Restore Status”
                    00121 ;
                    00122 ; Restore Status
                    00123 ;
000F                00124 RestoreStatus:
                    00125
000F 0E18           00126         swapf   SaveStatus,W
0010 0083           00127         movwf   STATUS                  ; restore STATUS Reg
0011 0E99           00128         swapf   SaveW register, F       ; save W register
0012 0E19           00129         swapf   SaveW register,W        ; restore W register
                    00130 ;
0013 108B           00131         bcf     INTCON,INTF
0014 0009           00132         retfie
                    00133 ;
                    00134
                    00135 ;*****************************************************************************************************
                    00136 ;
                    00137 ;*****************************************************************************************************
                    00138
0015                00139 Start:
                    00140
0015 2019           00141         call    Init_I2C_Slave          ; Initialize I2C Bus for Slave Mode, wait for START Bit detect
                    00142 ;
0016 178B           00143         bsf     INTCON,GIE              ; Enable Interrupts
0017                00144 wait:
0017 0064           00145         clrwdt                          ; User can write code here, will be interrupted by Falling Edge on INT
0018 2817           00146         goto    wait                    ; Loop until Start Bit Detect
                    00147
                    00148
                    00149 ;
                    00150 ;*****************************************************************************************************
                    00151 ;
```

```
                    00152
0019                00153 Init_I2C_Slave:
                    00154
0019 01A0           00155                clrf    ControlByte
001A 1683           00156                bsf     STATUS,RP0
001B 30C3           00157                movlw   _OPTION_INIT
001C 0081           00158                movwf   OPTION_REG      ; initially set INT for Falling Edge INT
001D 1406           00159                bsf     _SCL
001E 1486           00160                bsf     _SDA            ; set TRIS of SCL & SDA to inputs, pulled up by external resistors
                    00161
001F 1283           00162                bcf     STATUS,RP0
0020 0806           00163                movf    PORTB,W
0021 39FC           00164                andlw   0xFC            ; do not use BSF & BCF on Port Pins
0022 0086           00165                movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris
                    00166
0023 0194           00167                clrf    SubAddr         ; Set Sub Address to Zero
                    00168 ;
                    00169 ; Initialize A/D Setup or EEPROM Control Regs
                    00170 ;
                    00171
                    00172 ;#if AtoD_Slave
                    00173 ;              movlw   0x01            ; Select Channel 0, and turn on A/D Module with Fosc/2 Sampling Rate
                    00174 ;              movwf   _adcon0         ; already in Bank0, STATUS,RP0 = 0
                    00175 ;              bsf     STATUS,RP0
                    00176 ;              clrf   _adcon1          ; 4 analog channels with internal Vref
                    00177 ;#else
0024 0189           00178                clrf    _eeadr          ; already in Bank0, init EEPROM Addr = 0
0025 1683           00179                bsf     STATUS,RP0
0026 0188           00180                clrf    _eecon1         ; clear err flag
                    00181 ;#endif
                    00182 ;
0027 018B           00183                clrf    INTCON
0028 160B           00184                bsf     INTCON,INTE     ; Enable Falling Edge Interrupt On SDA (connected on RB0/INT pin)
0029 0008           00185                return
                    00186
                    00187 ;
                    00188 ;*****************************************************************************************************
                    00189 ;                       In-Line Code For I2C-Slave
                    00190 ;  Returns to detect next Start Bit after a STOP Bit Is Detected
                    00191 ;  This implementation is very in-efficient if the Master is constantly sending a Repeated START Condition
                    00192 ;
                    00193 ;*****************************************************************************************************
                    00194
002A                00195 StartBitDetect:
002A 2045           00196                call    RcvByte         ; 1st byte received in DataByte
002B 0064           00197                clrwdt
002C 1820           00198                btfsc   _STOP
```

```
002D 283F          00199                goto    i2c_start_wait  ; STOP bit Detected, wait for another START
002E 18A0          00200                btfsc   _START
002F 282A          00201                goto    StartBitDetect  ; a Repeated START condition
                   00202 ;
                   00203 ; The received byte in DataByte contains R/W & 7 Address the address
                   00204 ; If address match send ACK bit else NACK
                   00205 ;
0030 1120          00206                bcf     _RW
0031 180E          00207                btfsc   DataByte,LSB
0032 1520          00208                bsf     _RW             ; LSB of the 1st byte received contains R/W info
0033 100E          00209                bcf     DataByte,LSB
                   00210
0034 080E          00211                movf    DataByte,W
0035 3AD6          00212                xorlw   _MY_ADDRESS     ; if match, then Z bit is set
0036 1D03          00213                btfss   STATUS,Z
0037 283D          00214                goto    SendNACK        ; No Address Match, NACK
                   00215 ;
                   00216 ; Address Match Occured, send ACK
                   00217
0038 1403          00218                bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0039 20B3          00219                call    SendAck
003A 1920          00220                btfsc   _RW
003B 287B          00221                goto    SendReqData     ; read operation, so send current Data
003C 2867          00222                goto    RcvReqData      ; receive 2 bytes of data, sub-addr & data byte
                   00223 ;
                   00224 ;*****************************************************************************************
                   00225 ;
003D               00226 SendNACK:
003D 1003          00227                bcf     STATUS,C        ; SendAck routine sends NACK if Carry == 0
003E 20B3          00228                call    SendAck         ; address not us, so wait for another start bit
                   00229 ;
003F               00230 i2c_start_wait:
003F 0064          00231                clrwdt
0040 1683          00232                bsf     STATUS,RP0
0041 108B          00233                bcf     INTCON,INTF
0042 1406          00234                bsf     _SCL            ; release CLK line, may be held in low by us
0043 1486          00235                bsf     _SDA            ; release SDA
0044 280F          00236                goto RestoreStatus
                   00237 ;
                   00238 ;*****************************************************************************************
                   00239 ;                               Receive A Byte Of Data
                   00240 ;*****************************************************************************************
                   00241
0045               00242 RcvByte:
0045 01A1          00243                clrf    SlvStatus
                   00244
0046 3008          00245                movlw   0x08
```

```
0047 008F          00246                movwf   BitCount
                   00247 ;
0048 1283          00248                bcf     STATUS,RP0
0049 1806          00249                btfsc   _SCL
004A 2849          00250                goto    $-1             ; wait until SCL is low and then Read the Control Byte
                   00251 ;
004B 1683          00252                bsf     STATUS,RP0
004C 1406          00253                bsf     _SCL            ; release CLK, possibly held low
004D 1283          00254                bcf     STATUS,RP0
                   00255 ;
004E               00256 RcvNextBit:
004E 1C06          00257                btfss   _SCL
004F 284E          00258                goto    $-1             ; wait until clock is high, for valid data
0050 1886          00259                btfsc   _SDA
0051 285A          00260                goto    Rcvd_1
0052               00261 Rcvd_0:
0052 1003          00262                bcf     STATUS,C
0053 0D8E          00263                rlf     DataByte, F     ; left shift data ( MSB first)
0054               00264 _WaitClkLo1:
0054 1C06          00265                btfss   _SCL
0055 2862          00266                goto    next1
0056 1C86          00267                btfss   _SDA            ; SDA must still be low when CLK is high, else may be a STOP
0057 2854          00268                goto    _WaitClkLo1
0058 1420          00269                bsf     _STOP
0059 0008          00270                return
                   00271 ;
005A               00272 Rcvd_1:
005A 1403          00273                bsf     STATUS,C
005B 0D8E          00274                rlf     DataByte, F
005C               00275 _WaitClkLo2:
005C 1C06          00276                btfss   _SCL
005D 2862          00277                goto    next1         ; CLK went low, process next bit
005E 1886          00278                btfsc   _SDA          ; SDA must still be high when CLK is high, else may be a Repeated START
005F 285C          00279                goto    _WaitClkLo2
0060 14A0          00280                bsf     _START
0061 0008          00281                return
                   00282 ;
0062               00283 next1:
0062 0B8F          00284                decfsz  BitCount, F
0063 284E          00285                goto    RcvNextBit
                   00286 ;
                   00287 ; A complete Byte Received, HOLD Master's Clock Line Low to force a wait state
                   00288 ;
0064 1683          00289                bsf     STATUS,RP0
0065 1006          00290                bcf     _SCL            ; force SCL Low for wait state
0066 0008          00291                return
                   00292 ;*******************************************************************************************************
```

```
                    00293 ;                       Write Operation Requested
                    00294 ;
                    00295 ; Read Sub Address and A Data Byte & Acknowledge, if no errors
                    00296 ; Currently Only One Byte Is Programmed At A Time, Buffering scheme is unimplemented
                    00297 ;
                    00298 ;*************************************************************************************************
                    00299
0067                00300 RcvReqData
0067 2045           00301                 call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0068 080E           00302                 movf    DataByte,W
0069 0094           00303                 movwf   SubAddr
006A 0064           00304                 clrwdt
006B 1820           00305                 btfsc   _STOP
006C 283F           00306                 goto    i2c_start_wait  ; STOP bit Detected, wait for another START
006D 18A0           00307                 btfsc   _START
006E 282A           00308                 goto    StartBitDetect  ; a Repeated START condition
006F 1403           00309                 bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0070 20B3           00310                 call    SendAck         ; Sub-Addr Received, Send an ACK
                    00311 ;
                    00312 ; Receive Data Byte to Write To EEPROM
                    00313 ;
0071 2045           00314                 call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0072 0064           00315                 clrwdt
0073 1820           00316                 btfsc   _STOP
0074 283F           00317                 goto    i2c_start_wait  ; STOP bit Detected, wait for another START
0075 18A0           00318                 btfsc   _START
0076 282A           00319                 goto    StartBitDetect  ; a Repeated START condition
0077 1403           00320                 bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0078 20B3           00321                 call    SendAck         ; Sub-Addr Received, Send an ACK
0079 20CD           00322                 call    EEpromWrite     ; Program EEPROM
007A 280F           00323                 goto    RestoreStatus   ; STOP bit Detected, wait for another START
                    00324
                    00325 ;*************************************************************************************************
                    00326 ;                               Read Operation Requested
                    00327 ;
                    00328 ; Send A/D Data Of Required Channel until NACK By Master
                    00329 ;
                    00330 ;*************************************************************************************************
                    00331
007B                00332 SendReqData:
                    00333
007B 1283           00334                 bcf     STATUS,RP0
007C 0814           00335                 movf    SubAddr,W
007D 0089           00336                 movwf   _eeadr          ; Load Sub_Addr Of EEPROM
                    00337
                    00338 ;
007E                00339 SendNextByte:
```

```
007E 1683          00340                bsf     STATUS,RP0
007F 1408          00341                bsf     _rd             ; read EEPROM
0080 1283          00342                bcf     STATUS,RP0
0081 0808          00343                movf    _eedata,W
0082 008E          00344                movwf   DataByte        ; DataByte = EEPROM(addr)
0083 0A89          00345                incf    _eeadr, F       ; auto-increment sub-address
                   00346
0084 208D          00347                call    TxmtByte        ; send out EEPROM Data
0085 0064          00348                clrwdt
0086 18A0          00349                btfsc   _START          ; check for abnormal START
0087 283F          00350                goto    i2c_start_wait
0088 20C0          00351                call    ReadACK
0089 0064          00352                clrwdt
008A 1DA0          00353                btfss   _ACK            ; _ACK == 1 if +ve ACK Rcvd
008B 283F          00354                goto    i2c_start_wait ; NACK Received, a START or STOP condition may occur
008C 287E          00355                goto    SendNextByte   ; continue to send until NACK
                   00356 ;
                   00357 ;*****************************************************************************************************
                   00358
008D               00359 TxmtByte:
008D 080E          00360                movf    DataByte,W
008E 0093          00361                movwf   DataByteCopy    ; make copy of DataByte
008F 3008          00362                movlw   0x08
0090 008F          00363                movwf   BitCount
0091 01A1          00364                clrf    SlvStatus
                   00365 ;
0092               00366 TxmtNextBit:
0092 1683          00367                bsf     STATUS,RP0
0093 0D93          00368                rlf     DataByteCopy, F ; MSB First
                   00369
0094 1803          00370                btfsc   STATUS,C
0095 28A1          00371                goto    Txmt_1
0096               00372 Txmt_0
0096 1086          00373                bcf     _SDA
0097 0000          00374                nop
0098 1406          00375                bsf     _SCL            ; release clock line, let master pull it high
0099 1283          00376                bcf     STATUS,RP0
009A 1C06          00377                btfss   _SCL
009B 289A          00378                goto    $-1             ; wait until clk goes high
009C 1806          00379                btfsc   _SCL
009D 289C          00380                goto    $-1             ; wait until clk goes low
009E 1683          00381                bsf     STATUS,RP0
009F 1006          00382                bcf     _SCL            ; clk went low, continue to hold it low
00A0 28AD          00383                goto    Q_TxmtNextBit
                   00384
                   00385
00A1               00386 Txmt_1
```

```
00A1 1486          00387                bsf     _SDA
00A2 0000          00388                nop
00A3 1406          00389                bsf     _SCL            ; release clock line, let master pull it high
00A4 1283          00390                bcf     STATUS,RP0
00A5 1C06          00391                btfss   _SCL
00A6 28A5          00392                goto    $-1             ; wait until clk goes high
00A7               00393 _IsClkLo_1
00A7 1C86          00394                btfss   _SDA
00A8 28B1          00395                goto    MayBeErr_Txmt   ; must never come here, illegal Repeated Start
00A9 1806          00396                btfsc   _SCL
00AA 28A7          00397                goto    _IsClkLo_1      ; wait until clk goes low
                   00398
00AB 1683          00399                bsf     STATUS,RP0
00AC 1006          00400                bcf     _SCL            ; clk went low, continue to hold it low
                   00401 ;
00AD               00402 Q_TxmtNextBit
00AD 0B8F          00403                decfsz  BitCount, F
00AE 2892          00404                goto    TxmtNextBit
00AF 1486          00405                bsf     _SDA            ; release SDA for Master's ACK
00B0 0008          00406                return
                   00407 ;
00B1               00408 MayBeErr_Txmt:
00B1 14A0          00409                bsf     _START          ; illegal Repeated START condition during a byte transfer
00B2 0008          00410                return
                   00411 ;
                   00412 ;*****************************************************************************************************
                   00413 ;                               Send ACK/NACK to Master
                   00414 ;
                   00415 ;  Prior to calling this routine, set CARRY bit to 1 for sending +ve ACK & set CARRY = 0, for NACK
                   00416 ;
                   00417 ;*****************************************************************************************************
                   00418
00B3               00419 SendAck:
00B3 1683          00420                bsf     STATUS,RP0
00B4 1803          00421                btfsc   STATUS,C        ; Carry bit == 1 for ACK else NACK
00B5 1086          00422                bcf     _SDA            ; pull SDA low to send +ve ACK
00B6 1406          00423                bsf     _SCL            ; release CLK line, let master clk it
00B7 1283          00424                bcf     STATUS,RP0
00B8 1C06          00425                btfss   _SCL
00B9 28B8          00426                goto    $-1             ; loop until Clk High
00BA 1806          00427                btfsc   _SCL
00BB 28BA          00428                goto    $-1             ; loop until Clk is Low, ACK bit sent
00BC 1683          00429                bsf     STATUS,RP0
00BD 1006          00430                bcf     _SCL            ; HOLD CLK Line LOW
00BE 1486          00431                bsf     _SDA            ; ACK over, release SDA line for Master control
                   00432 ;
                   00433 ;
```

```
00BF 0008           00434                return
                    00435 ;
                    00436 ;*******************************************************************************************************
                    00437 ;                                    Read ACK Sent By Master
                    00438 ;
                    00439 ; If +ve ACK then set _ACK bit in SlaveStatus Reg, else 0
                    00440 ;
                    00441 ;*******************************************************************************************************
                    00442
00C0                00443 ReadACK:
00C0 1683           00444                bsf     STATUS,RP0
00C1 1406           00445                bsf     _SCL            ; release clock
00C2 1283           00446                bcf     STATUS,RP0
00C3 1C06           00447                btfss   _SCL
00C4 28C3           00448                goto    $-1             ; wait until clock is high (9 the bit:ACK)
00C5 15A0           00449                bsf     _ACK            ; expecting a +ve ACK
00C6 1886           00450                btfsc   _SDA
00C7 11A0           00451                bcf     _ACK            ; NACK rcvd, stop transmission
00C8 1806           00452                btfsc   _SCL
00C9 28C8           00453                goto    $-1             ; wait until Clock is low
00CA 1683           00454                bsf     STATUS,RP0
00CB 1006           00455                bcf     _SCL            ; force Clock low
00CC 0008           00456                return
                    00457 ;*******************************************************************************************************
                    00458 ;                       Write One Byte Of Data To EEPROM Array at Sub-Addr
                    00459 ;
                    00460 ;*******************************************************************************************************
00CD                00461 EEpromWrite:
00CD 1283           00462                bcf     STATUS,RP0
00CE 0814           00463                movf    SubAddr,W
00CF 0089           00464                movwf   _eeadr          ; load sub-address
00D0 080E           00465                movf    DataByte,W
00D1 0088           00466                movwf   _eedata         ; load data to be written into EEDATA
00D2 1683           00467                bsf     STATUS,RP0
00D3 1508           00468                bsf     _wren           ; enable write operation
00D4 1188           00469                bcf     _wrerr          ; clear any previous error flags
00D5 3055           00470                movlw   0x55
00D6 0089           00471                movwf   _eecon2
00D7 30AA           00472                movlw   0xAA
00D8 0089           00473                movwf   _eecon2
00D9 1488           00474                bsf     _wr             ; start Write Operation
                    00475 ;
00DA 1406           00476                bsf     _SCL            ; release CLK line, may be held in low by us
00DB 1486           00477                bsf     _SDA            ; release SDA
                    00478 ;
00DC 0064           00479                clrwdt
00DD 1888           00480                btfsc   _wr             ; Poll until WR Over
```

```
00DE 28DC           00481                goto    $-2
                    00482 ;
00DF 1108           00483                bcf     _wren           ; disable write operations
00E0 0064           00484                clrwdt
00E1 108B           00485                bcf     INTCON,INTF
                    00486 ;
00E2 0008           00487                return
                    00488 ;*************************************************************************************************
                    00489
                    00490
                    00491     end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   224
Program Memory Words Free:   800


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     1 suppressed
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: TEST.ASM

```
MPASM 01.40.01 Intermediate   I2CTEST.ASM   4-4-1997  14:45:51         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         Title           "I2C Master Mode Implemetation"
                     00002         SubTitle        "Rev 0.2        :  04 April 1997"
                     00003
                     00004 ;*****************************************************************************************************
                     00005 ;
                     00006 ;                       Software Implementation Of I2C Master Mode
                     00007 ;
                     00008 ;       * Master Transmitter & Master Receiver Implemented in software
                     00009 ;       * Slave Mode implemented in hardware
                     00010 ;
                     00011 ;       *       Refer to Signetics/Philips I2C-Bus Specification
                     00012 ;
                     00013 ;    The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX product
                     00014 ;
                     00015 ;       RB1 is SDA              (Any I/O Pin May Be used instead)
                     00016 ;       RB0/INT is SCL          (Any I/O Pin May Be used instead)
                     00017 ;
                     00018 ;
                     00019 ;       Program:          I2CTEST.ASM
                     00020 ;       Revision Date:    Rev 0.1       :  01 Mar 1993
                     00021 ;                         4-04-97      Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;*****************************************************************************************************
                     00024
                     00025                 LIST    p = 16C71
                     00026                 ERRORLEVEL -302
                     00027                 Radix   DEC
                     00028
  00F42400           00029 _ClkIn          equ     16000000        ; Input Clock Frequency Of PIC16C71
                     00030
                     00031                 include         <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                     00142         LIST
```

```
                    00032
00000001            00033 TRUE    equ     1
00000000            00034 FALSE   equ     0
                    00035
00000000            00036 LSB     equ     0
00000007            00037 MSB     equ     7
                    00038
                    00039 ;
                    00040 #define _Slave_1_Addr   0xA0            ; Serial EEPROM #1
                    00041 #define _Slave_2_Addr   0xAC            ; Serial EEPROM #2
                    00042 #define _Slave_3_Addr   0xD6            ; Slave PIC16CXX
                    00043
                    00044 #define _ENABLE_BUS_FREE_TIME   TRUE
                    00045 #define _CLOCK_STRETCH_CHECK    TRUE
                    00046 #define _INCLUDE_HIGH_LEVEL_I2C TRUE
                    00047
                    00048
                    00049                 include         “i2c.h”
                    00001 ;*******************************************************************************************************
                    00002 ;                               I2C Bus Header File
                    00003 ;*******************************************************************************************************
                    00004
003D0900            00005 _ClkOut         equ     (_ClkIn >> 2)
                    00006
                    00007 ;
                    00008 ; Compute the delay constants for setup & hold times
                    00009 ;
00000010            00010 _40uS_Delay     set     (_ClkOut/250000)
00000012            00011 _47uS_Delay     set     (_ClkOut/212766)
00000014            00012 _50uS_Delay     set     (_ClkOut/200000)
                    00013
                    00014 #define _OPTION_INIT    (0xC0 | 0x03)   ; Prescaler to TMR0 for Appox 1 mSec timeout
                    00015 ;
                    00016 #define _SCL    PORTB,0
                    00017 #define _SDA    PORTB,1
                    00018
                    00019 #define _SCL_TRIS       _trisb,0
                    00020 #define _SDA_TRIS       _trisb,1
                    00021
                    00022 #define _WRITE_         0
                    00023 #define _READ_          1
                    00024
                    00025
                    00026                                       ; Register File Variables
                    00027
                    00028
                    00029                 CBLOCK  0x0C
```

```
0000000C          00030                     SlaveAddr             ; Slave Addr must be loaded into this reg
0000000D          00031                     SlaveAddrHi           ; for 10 bit addressing mode
0000000E          00032                     DataByte              ; load this reg with the data to be transmitted
0000000F          00033                     BitCount              ; The bit number (0:7) transmitted or received
00000010          00034                     Bus_Status            ; Status Reg of I2C Bus for both TXMT & RCVE
00000011          00035                     Bus_Control           ; control Register of I2C Bus
00000012          00036                     DelayCount
00000013          00037                     DataByteCopy          ; copy of DataByte for Left Shifts (destructive)
                  00038
00000014          00039                     SubAddr               ; sub-address of slave (used in I2C_HIGH.ASM)
00000015          00040                     SrcPtr                ; source pointer for data to be transmitted
                  00041
00000016          00042                     tempCount             ; a temp variable for scratch RAM
00000017          00043                     StoreTemp_1           ; a temp variable for scratch RAM, do not disturb contents
                  00044
00000018          00045                     _End_I2C_Ram          ; unused, only for ref of end of RAM allocation
                  00046                 ENDC
                  00047
                  00048 ;*****************************************************************************
                  00049 ;                       I2C Bus Status Reg Bit Definitions
                  00050 ;*****************************************************************************
                  00051
                  00052 #define _Bus_Busy       Bus_Status,0
                  00053 #define _Abort          Bus_Status,1
                  00054 #define _Txmt_Progress  Bus_Status,2
                  00055 #define _Rcv_Progress   Bus_Status,3
                  00056
                  00057 #define _Txmt_Success   Bus_Status,4
                  00058 #define _Rcv_Success    Bus_Status,5
                  00059 #define _Fatal_Error    Bus_Status,6
                  00060 #define _ACK_Error      Bus_Status,7
                  00061
                  00062 ;*****************************************************************************
                  00063 ;                       I2C Bus Contro Register
                  00064 ;*****************************************************************************
                  00065 #define _10BitAddr      Bus_Control,0
                  00066 #define _Slave_RW       Bus_Control,1
                  00067 #define _Last_Byte_Rcv  Bus_Control,2
                  00068
                  00069 #define _SlaveActive    Bus_Control,6
                  00070 #define _TIME_OUT_      Bus_Control,7
                  00071
                  00072
                  00073
                  00074
                  00075 ;*************************************************************************************************
                  00076 ;                               General Purpose Macros
```

```
                    00077 ;*******************************************************************************************************
                    00078
                    00079 RELEASE_BUS     MACRO
                    00080                         bsf     STATUS,RP0      ; select Bank1
                    00081                         bsf     _SDA            ; tristate SDA
                    00082                         bsf     _SCL            ; tristate SCL
                    00083 ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                    00084                 ENDM
                    00085
                    00086 ;*******************************************************************************************************
                    00087 ;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
                    00088 ;
                    00089 ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
                    00090 ;  on 8 or 10 bit addressing modes
                    00091 ;*******************************************************************************************************
                    00092
                    00093 LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
                    00094
                    00095                 bsf     _10BitAddr      ; Slave has 10 bit address
                    00096                 movlw   (SLAVE_ADDRESS & 0xff)
                    00097                 movwf   SlaveAddr       ; load low byte of address
                    00098                 movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)     ; 10 bit addr is 11110XX0
                    00099                 movwf   SlaveAddr+1     ; hi order  address
                    00100
                    00101                 ENDM
                    00102
                    00103 LOAD_ADDR_8     MACRO   SLAVE_ADDRESS
                    00104
                    00105                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                    00106                 movlw   (SLAVE_ADDRESS & 0xff)
                    00107                 movwf   SlaveAddr
                    00108
                    00109                 ENDM
                    00050
                    00051                 CBLOCK  _End_I2C_Ram
  00000018          00052                         SaveStatus              ; copy of STATUS Reg
  00000019          00053                         SaveW register          ; copy of W register
  0000001A          00054                         byteCount
  0000001B          00055                         HoldData
                    00056                 ENDC
                    00057
                    00058                 CBLOCK  0x20
  00000020          00059                         DataBegin               ; Data to be read or written is stored here
                    00060                 ENDC
                    00061
                    00062
0000                00063                 ORG     0x00
```

```
                    00064
0000 2956           00065                goto    Start
                    00066 ;
0004                00067                ORG     0x04
                    00068 ;*************************************************************************************************
                    00069 ;                              Interrupt Service Routine
                    00070 ;
                    00071 ;   For I2C routines, only TMR0 interrupt is used
                    00072 ; TMR0 Interrupts enabled only if Clock Stretching is Used
                    00073 ; On TMR0 timeout interrupt, disable TMR0 Interrupt, clear pending flags,
                    00074 ; MUST set _TIME_OUT_ flag saying possibly a FATAL error occurred
                    00075 ; The user may choose to retry the operation again later
                    00076 ;
                    00077 ;*************************************************************************************************
                    00078
0004                00079 Interrupt:
                    00080 ;
                    00081 ; Save Interrupt Status (W register & STATUS regs)
                    00082 ;
0004 0099           00083        movwf   SaveW register          ; Save W register
0005 0E03           00084        swapf   STATUS,W                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098           00085        movwf   SaveStatus              ; Save STATUS Reg
                    00086  if _CLOCK_STRETCH_CHECK               ; TMR0 Interrupts enabled only if Clock Stretching is Used
0007 1D0B           00087        btfss   INTCON,T0IF
0008 280B           00088        goto    MayBeOtherInt           ; other Interrupts
0009 1791           00089        bsf     _TIME_OUT_              ; MUST set this Flag, can take other desired actions here
000A 110B           00090        bcf     INTCON,T0IF
                    00091  endif
                    00092 ;
                    00093 ;  Check For Other Interrupts Here, This program usesd only TMR0 & INT Interrupt
                    00094 ;
000B                00095 MayBeOtherInt:
000B 0000           00096        NOP
                    00097 ;
000C                00098 RestoreIntStatus:                      ; Restore Interrupt Status
000C 0E18           00099        swapf   SaveStatus,W
000D 0083           00100        movwf   STATUS                  ; restore STATUS Reg
000E 0E99           00101        swapf   SaveW register, F
000F 0E19           00102        swapf   SaveW register,W        ; restore W register
0010 0009           00103        retfie
                    00104 ;
                    00105 ;*************************************************************************************************
                    00106 ;                               Include I2C High Level & Low Level Routines
                    00107   if _INCLUDE_HIGH_LEVEL_I2C
                    00108                include         “i2c_high.inc”
                    00001 ;*************************************************************************************************
                    00002 ;
```

```
00003 ;                               I2C Master : General Purpose Macros & Subroutines
00004 ;
00005 ;               High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
00006 ;
00007 ;
00008 ;       Program:          I2C_HIGH.ASM
00009 ;       Revision Date:
00010 ;                         1-16-97      Compatibility with MPASMWIN 1.40
00011 ;
00012 ;*******************************************************************************************************
00013
00014
00015 ;*******************************************************************************************************
00016 ;
00017 ;                               I2C_TEST_DEVICE
00018 ; MACRO
00019 ;
00020 ;   If Slave Device is listening, then _SlaveActive bit is set, else is cleared
00021 ;
00022 ;       Parameter :  NONE
00023 ;
00024 ;   Sequence Of Operations :
00025 ;       S-SlvAW-A-P
00026 ;   If A is +ve device is listening, else either busy, not present or error condition
00027 ;
00028 ;   This test may also be used to check for eample if a Serial EEPROM is in internal programming
00029 ;   mode
00030 ;
00031 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
00032 ;         mode addressing must be set
00033 ;*******************************************************************************************************
00034
00035 I2C_TEST_DEVICE         MACRO
00036
00037                         call    IsSlaveActive           ; TEMP ???? : Assembler Error with this MACRO
00038
00039                         ENDM
00040 ;
00041 ;
00042 ;               Test If A Device of SlaveAddr Is Present on Bus
00043 ;
00044 ;  The Slave Address Is put on the bus and if ACK it is present, if NACK not present
00045 ;  or maybe device is not responding. The presense can be checked constantly by a master
00046 ;  (for ex. the Operating System on an Access.Bus may constantly issue this command)
00047 ;
00048 ;   Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
00049 ;   Set  _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
```

```
                        00050 ;
                        00051 ;   Returns  1 in _SlaveActive Bit if slave is responding else a 0
                        00052 ;
                        00053 ;
                        00054
0011                    00055 IsSlaveActive
0011 1091               00056                 bcf     _Slave_RW       ; set for write operation
0012 2057               00057                 call    TxmtStartBit    ; send START bit
0013 206B               00058                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                        00059 ;
0014 1311               00060                 bcf     _SlaveActive
0015 1F90               00061                 btfss   _ACK_Error      ; skip if NACK, device is not present or not responding
0016 1711               00062                 bsf     _SlaveActive    ; ACK received, device present & listening
0017 205F               00063                 call    TxmtStopBit
0018 0008               00064                 return
                        00065 ;
                        00066 ;*****************************************************************************************************
                        00067 ;                               I2C_WRITE
                        00068 ;
                        00069 ;  A basic macro for writing a block of data to a slave
                        00070 ;
                        00071 ;  Parameters :
                        00072 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_
                        00073 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                        00074 ;
                        00075 ;   Sequence :
                        00076 ;               S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
                        00077 ;
                        00078 ;  If an error occurs then the routine simply returns and user should check for
                        00079 ;       flags in Bus_Status Reg (for eg. _Txmt_Success flag)
                        00080 ;
                        00081 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                        00082 ;               mode addressing must be set
                        00083 ;*****************************************************************************************************
                        00084
                        00085
                        00086 I2C_WR          MACRO   _BYTES_, _SourcePointer_
                        00087
                        00088                 movlw   _BYTES_
                        00089                 movwf   tempCount
                        00090                 movlw   _SourcePointer_
                        00091                 movwf   FSR
                        00092
                        00093                 call    _i2c_block_write
                        00094                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                        00095
                        00096                 ENDM
```

```
                    00097
0019                00098 _i2c_block_write:
0019 2057           00099                 call    TxmtStartBit    ; send START bit
001A 1091           00100                 bcf     _Slave_RW       ; set for write operation
001B 206B           00101                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                    00102 ;
001C                00103 _block_wr1_loop:
001C 1E10           00104                 btfss   _Txmt_Success
001D 0008           00105                 return
001E 0800           00106                 movf    INDF,W
001F 008E           00107                 movwf   DataByte        ; start from the first byte starting at _DataPointer_
0020 0A84           00108                 incf    FSR, F
0021 2095           00109                 call    SendData        ; send next byte, bus is our's !
0022 0B96           00110                 decfsz  tempCount, F
0023 281C           00111                 goto    _block_wr1_loop ; loop until desired bytes of data transmitted to slave
0024 0008           00112                 return
                    00113 ;
                    00114 ;*************************************************************************************************
                    00115
                    00116 ;*************************************************************************************************
                    00117 ;                           I2C_WRITE_SUB
                    00118 ;
                    00119 ;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
                    00120 ;  to a slave device.
                    00121 ;             Eg. : A serial EEPROM would need an address of memory location for Random Writes
                    00122 ;
                    00123 ;  Parameters :
                    00124 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
                    00125 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                    00126 ;               _Sub_Address_           Sub-address of Slave (constant)
                    00127 ;
                    00128 ;   Sequence :
                    00129 ;               S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P
                    00130 ;
                    00131 ;  If an error occurs then the routine simply returns and user should check for
                    00132 ;  flags in Bus_Status Reg (for eg. _Txmt_Success flag
                    00133 ;
                    00134 ;            Returns :  W register = 1 on success, else W register = 0
                    00135 ;
                    00136 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                    00137 ;         mode addressing must be set
                    00138 ;
                    00139 ;  COMMENTS :
                    00140 ;               I2C_WR may prove to be more efficient than this macro in most situations
                    00141 ;               Advantages will be found for Random Address Block Writes for Slaves with
                    00142 ;               Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMS)
                    00143 ;
```

```
00144 ;*****************************************************************************************************
00145
00146 I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
00147
00148                 movlw   (_BYTES_ + 1)
00149                 movwf   tempCount
00150
00151                 movlw   (_SourcePointer_ - 1)
00152                 movwf   FSR
00153
00154                 movf    INDF,W
00155                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00156                 movlw   _Sub_Address_
00157                 movwf   INDF            ; store temporarily the sub-address at (_SourcePointer_ -1)
00158
00159                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00160
00161                 movf    StoreTemp_1,W
00162                 movwf   (_SourcePointer_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00163
00164                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00165
00166                 ENDM
00167
00168 ;*****************************************************************************************************
00169 ;                               I2C_WR_SUB_SWINC
00170 ;
00171 ;  Parameters :
00172 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
00173 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
00174 ;               _Sub_Address_           Sub-address of Slave (constant)
00175 ;
00176 ;   Sequence :
00177 ;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
00178 ;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
00179 ;                       and so on until #of Bytes
00180 ;
00181 ;  If an error occurs then the routine simply returns and user should check for
00182 ;       flags in Bus_Status Reg (for eg. _Txmt_Success flag
00183 ;
00184 ;       Returns :       W register = 1 on success, else W register = 0
00185 ;
00186 ;  COMMENTS : Very In-efficient, Bus is given up after every Byte Write
00187 ;
00188 ;               Some I2C devices addresed with a sub-address do not increment automatically
00189 ;       after an access of each byte. Thus a block of data sent must have a sub-address
00190 ;       followed by a data byte.
```

```
                     00191 ;
                     00192 ;*******************************************************************************************************
                     00193
                     00194 I2C_WR_SUB_SWINC        MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
                     00195
                     00196                        variable i              ; TEMP ???? : Assembler Does Not Support This
                     00197
                     00198                        i = 0
                     00199
                     00200                          while (i < _BYTES_)
                     00201                                 movf    (_Source_Pointer_ + i),W
                     00202                                 movwf   SrcPtr
                     00203                                 movf    (_Sub_Address_ + i),W
                     00204                                 movwf   SubAddr
                     00205                                 call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                     00206
                     00207                                 i++
                     00208                        endw
                     00209
                     00210                         ENDM
                     00211
                     00212 ;
                     00213 ;
                     00214 ;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
                     00215 ;
                     00216
0025                 00217 _i2c_byte_wr_sub:
0025 2057            00218                 call    TxmtStartBit    ; send START bit
0026 1091            00219                 bcf     _Slave_RW       ; set for write operation
0027 206B            00220                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
0028 1E10            00221                 btfss   _Txmt_Success
0029 2835            00222                 goto    _block_wr1_fail ; end
002A 0814            00223                 movf    SubAddr,W
002B 008E            00224                 movwf   DataByte        ; start from the first byte starting at _DataPointer_
002C 2095            00225                 call    SendData        ; send next byte
002D 1E10            00226                 btfss   _Txmt_Success
002E 2835            00227                 goto    _block_wr1_fail ; end
002F 0815            00228                 movf    SrcPtr,W
0030 008E            00229                 movwf   DataByte        ; start from the first byte starting at _DataPointer_
0031 2095            00230                 call    SendData        ; send next byte
0032 1E10            00231                 btfss   _Txmt_Success
0033 2835            00232                 goto    _block_wr1_fail ; failed, return 0 in W register
0034 2837            00233                 goto    _block_wr1_pass ; successful, return 1 in W register
                     00234 ;
                     00235 ; return back to called routine from either _block_wr1_pass or _block_wr1_fail
                     00236 ;
0035                 00237 _block_wr1_fail:
```

```
0035 205F          00238                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0036 3400          00239                retlw   FALSE
0037               00240 _block_wr1_pass:
0037 205F          00241                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0038 3401          00242                retlw   TRUE
                   00243 ;
                   00244
                   00245 ;*************************************************************************************************
                   00246 ;
                   00247 ;                               I2C_WR_MEM_BYTE
                   00248 ;
                   00249 ;  Some I2C devices like a EEPROM need to wait fo some time after every byte write
                   00250 ;  (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
                   00251 ;  but in addition adds a delay after each byte.
                   00252 ;          Some EERPOM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
                   00253 ;               this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
                   00254 ;               for a block of data and then insert a delay until the whole buffer is written.
                   00255 ;
                   00256 ;  Parameters :
                   00257 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
                   00258 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                   00259 ;               _Sub_Address_           Sub-address of Slave (constant)
                   00260 ;
                   00261 ;   Sequence :
                   00262 ;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
                   00263 ;               Delay 1 mSec            ; The user can chnage this value to desired delay
                   00264 ;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
                   00265 ;               Delay 1 mSec
                   00266 ;               and so on until #of Bytes
                   00267 ;
                   00268 ;*************************************************************************************************
                   00269
                   00270 I2C_WR_BYTE_MEM         MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
                   00271
                   00272                       variable i              ; TEMP ???? : Assembler Does Not Support This
                   00273
                   00274                       i = 0
                   00275
                   00276                       while (i < _BYTES_)
                   00277                               movf    (_Source_Pointer_ + i),W
                   00278                               movwf   SrcPtr
                   00279                               movf    (_Sub_Address_ + i),W
                   00280                               movwf   SubAddr
                   00281                               call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                   00282                               call    Delay50uSec
                   00283
                   00284                               i++
```

```
00285                       endw
00286
00287
00288                        ENDM
00289
00290 ;*****************************************************************************************************
00291 ;                   I2C_WR_MEM_BUF
00292 ;
00293 ;   This Macro/Function writes #of _BYTES_ to an I2C memory device. However
00294 ;   some devices, esp. EEPROMs must wait while the device enters into programming
00295 ;   mode. But some devices have an onchip temp data hold buffer and is used to
00296 ;   store data before the device actually enters into programming mode.
00297 ;   For example, the 24C04 series of Serial EEPROMs from Microchip
00298 ;   have an 8 byte data buffer. So one can send 8 bytes of data at a time
00299 ;   and then the device enters programming mode. The master can either wait
00300 ;   until a fixed time and then retry to program or can continiously poll
00301 ;   for ACK bit and then transmit the next Block of data for programming
00302 ;
00303 ;  Parameters :
00304 ;               _BYTES_                 # of bytes to write to memory
00305 ;               _SourcePointer_         Pointer to the block of data
00306 ;               _SubAddress_            Sub-address of the slave
00307 ;               _Device__BUF_SIZE_      The on chip buffer size of the i2c slave
00308 ;
00309 ;  Sequence of operations
00310 ;                       I2C_SUB_WR operations are performed in loop and each time
00311 ;                       data buffer of BUF_SIZE is output to the device. Then
00312 ;                       the device is checked for busy and when not busy another
00313 ;                       block of data is written
00314 ;
00315 ;*****************************************************************************************************
00316
00317 I2C_WR_BUF_MEM  MACRO   _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
00318
00319
00320               variable i, j
00321
00322
00323               if ( !_BYTES_)
00324                       exitm
00325
00326               elif ( _BYTES_ <=  _Device_BUF_SIZE_)
00327
00328                       I2C_WR_SUB      _BYTES_, _SourcePointer_, _SubAddress_
00329
00330                       exitm
00331
```

```
00332
00333                 else
00334
00335                         i = 0
00336                         j = (_BYTES_ / _Device_BUF_SIZE_)
00337                         while (i < j)
00338
00339 I2C_WR_SUB  _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_),(_SubAddress_ + i*_Device_BUF_SIZE_)
00340
00341                                 call    IsSlaveActive
00342                                 btfss   _SlaveActive
00343                                 goto    $-2
00344
00345                           i++
00346                           endw
00347
00348                         j = (_BYTES_ - i*_Device_BUF_SIZE_)
00349
00350                         if (j)
00351 I2C_WR_SUB j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_  + i*_Device_BUF_SIZE_)
00352                         endif
00353
00354                 endif
00355
00356                 ENDM
00357
00358
00359 ;*****************************************************************************************************
00360 ;
00361 ;                               I2C_READ
00362 ;
00363 ; The basic MACRO/procedure to read a block message from a slave device
00364 ;
00365 ;   Parameters :
00366 ;               _BYTES_         :  constant : #of bytes to receive
00367 ;               _DestPointer_   :  destination pointer of RAM (File Registers)
00368 ;
00369 ;   Sequence :
00370 ;               S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
00371 ;
00372 ;   If last byte, then Master will NOT Acknowledge (send NACK)
00373 ;
00374 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
00375 ;         mode addressing must be set
00376 ;
00377 ;*****************************************************************************************************
00378
```

```
                    00379 I2C_READ        MACRO   _BYTES_, _DestPointer_
                    00380
                    00381
                    00382                 movlw   (_BYTES_ -1)
                    00383                 movwf   tempCount              ; -1 because, the last byte is used out of loop
                    00384                 movlw   _DestPointer_
                    00385                 movwf   FSR                    ; FIFO destination address pointer
                    00386
                    00387                 call    _i2c_block_read
                    00388
                    00389                 ENDM
                    00390
0039                00391 _i2c_block_read:
0039 2057           00392                 call    TxmtStartBit    ; send START bit
003A 1491           00393                 bsf     _Slave_RW       ; set for read operation
003B 1111           00394                 bcf     _Last_Byte_Rcv  ; not a last byte to rcv
003C 206B           00395                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
003D 1A10           00396                 btfsc   _Txmt_Success
003E 2841           00397                 goto    _block_rd1_loop ; end
003F 205F           00398                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0040 3400           00399                 retlw   FALSE           ; Error : may be device not responding
                    00400 ;
0041                00401 _block_rd1_loop:
0041 20CC           00402                 call    GetData
0042 080E           00403                 movf    DataByte,W
0043 0080           00404                 movwf   INDF            ; start receiving data, starting at Destination Pointer
0044 0A84           00405                 incf    FSR, F
0045 0B96           00406                 decfsz  tempCount, F
0046 2841           00407                 goto    _block_rd1_loop ; loop until desired bytes of data transmitted to slave
0047 1511           00408                 bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
0048 20CC           00409                 call    GetData
0049 080E           00410                 movf    DataByte,W
004A 0080           00411                 movwf   INDF
004B 205F           00412                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
004C 3401           00413                 retlw   TRUE
                    00414
                    00415
                    00416 ;*****************************************************************************************************
                    00417 ;
                    00418 ;                                 I2C_READ_SUB
                    00419 ;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
                    00420 ;  Between the sub-address write & the following reads, a STOP condition is not issued and
                    00421 ;  a “REPEATED START” condition is used so that an other master will not take over the bus,
                    00422 ;  and also that no other master will overwrite the sub-address of the same slave.
                    00423 ;
                    00424 ;   This function is very commonly used in accessing Random/Sequential reads from a
                    00425 ;   memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
```

```
00426 ;
00427 ;  Parameters :
00428 ;               _BYTES_          # of bytes to read
00429 ;               _DestPointer_    The destination pointer of data to be received.
00430 ;               _BubAddress_     The sub-address of the slave
00431 ;
00432 ;  Sequence :
00433 ;               S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
00434 ;
00435 ;
00436 ;*************************************************************************************************
00437
00438 I2C_READ_SUB    MACRO   _BYTES_, _DestPointer_, _SubAddress_
00439
00440                 bcf     _Slave_RW       ; set for write operation
00441                 call    TxmtStartBit    ; send START bit
00442                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
00443
00444                 movlw   _SubAddress_
00445                 movwf   DataByte        ; START address of EEPROM(slave 1)
00446                 call    SendData        ; write sub address
00447 ;
00448 ; do not send STOP after this, use REPEATED START condition
00449 ;
00450
00451                 I2C_READ _BYTES_, _DestPointer_
00452
00453
00454                 ENDM
00455
00456
00457
00458 ;*************************************************************************************************
00459 ;
00460 ;                               I2C_READ_STATUS
00461 ;
00462 ;  This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
00463 ;  send a status byte upon reception of a control byte
00464 ;  This is basically same as I2C_READ MACRO for reading a single byte
00465 ;
00466 ;   For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
00467 ;   data at the current address location
00468 ;
00469 ;  On success W register = 1 else = 0
00470 ;
00471 ;*************************************************************************************************
00472
```

```
00473 I2C_READ_STATUS MACRO   _DestPointer_
00474
00475
00476                 call    TxmtStartBit    ; send START bit
00477                 bsf     _Slave_RW       ; set for read operation
00478                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
00479                 btfsc   _Txmt_Success
00480                 goto    _byte_rd1_loop  ; read a byte
00481                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00482                 retlw   FALSE           ; Error : may be device not responding
00483 _byte_rd1_loop:
00484                 bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
00485                 call    GetData
00486                 movf    DataByte,W
00487                 movwf   _DestPointer_
00488                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00489                 btfss   _Rcv_Success
00490                 retlw   FALSE
00491                 retlw   TRUE
00492
00493                 ENDM
00494
00495 ;*****************************************************************************************************
00496
00497 I2C_READ_BYTE   MACRO   _DestPointer_
00498
00499                 I2C_READ_STATUS MACRO   _DestPointer_
00500
00501                 ENDM
00502
00503
00504 ;*****************************************************************************************************
00505 ;
00506 ;                               I2C_WR_SUB_WR
00507 ;
00508 ;  This Macro writes 2 Blocks of Data (variable length) to a slave at a sub-address. This
00509 ;  may be useful for devices which need 2 blocks of data in which the first block may be an
00510 ;  extended address of a slave device. For example, a large I2C memory device, or a teletext
00511 ;  device with an extended addressing scheme, may need multiple bytes of data in the 1st block
00512 ;  that represents the actual physical address and is followed by a 2nd block that actually
00513 ;  represents the data.
00514 ;
00515 ;  Parameters :
00516 ;
00517 ;               _BYTES1_                1st block #of bytes
00518 ;               _SourcePointer1_        Start Pointer of the 1st block
00519 ;               _SubAddress_            Sub-Address of slave
```

```
00520 ;               _BYTES2_                2st block #of bytes
00521 ;               _SourcePointer2_        Start Pointer of the 2nd block
00522 ;
00523 ;  Sequence :
00524 ;               S-SlvW-A-SubA-A-D1[0]-A-.....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
00525 ;
00526 ;  Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
00527 ;               a STOP bit is not sent (bus is not given up) in between
00528 ;               the two I2C_WR_SUB
00529 ;
00530 ;  Check Txmt_Success flag for any transmission errors
00531 ;
00532 ;*************************************************************************************************
00533
00534 I2C_WR_SUB_WR   MACRO   _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_
00535
00536                 movlw   (_COUNT1_ + 1)
00537                 movwf   tempCount
00538                 movlw   (_SourcePointer1_ - 1)
00539                 movwf   FSR
00540
00541                 movf    INDF,W
00542                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00543                 movlw   _Sub_Address_
00544                 movwf   INDF            ; store temporarily the sub-address at (_SourcePointer_ -1)
00545                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00546 ;
00547                 movf    StoreTemp_1,W
00548                 movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00549 ; Block 1 write over
00550 ; Send Block 2
00551                 movlw   _COUNT2_
00552                 movwf   tempCount
00553                 movlw   _SourcePointer2_
00554                 movwf   FSR
00555                 call    _block_wr1_loop
00556 ;
00557                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00558
00559                 ENDM
00560
00561
00562 ;*************************************************************************************************
00563 ;
00564 ;                               I2C_WR_SUB_RD
00565 ;
00566 ;  This macro writes a block of data from SourcePointer of length _COUNT1_ to a slave
```

```
00567 ;       at sub-address and then Reads a block of Data of length _COUNT2_ to destination
00568 ;       address pointer
00569 ;
00570 ;
00571 ;  Message Structure :
00572 ;              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
00573 ;
00574 ;  Parameters :
00575 ;               _COUNT1_                Length Of Source Buffer
00576 ;               _SourcePointer_         Source Pointer Address
00577 ;               _Sub_Address_           The Sub Address Of the slave
00578 ;               _COUNT2_                The length of Destination Buffer
00579 ;               _DestPointer_           The start address of Destination Pointer
00580 ;
00581 ;*************************************************************************************************
00582
00583 I2C_WR_SUB_RD   MACRO   _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_
00584
00585                 movlw   (_COUNT1_ + 1)
00586                 movwf   tempCount
00587                 movlw   (_SourcePointer_ - 1)
00588                 movwf   FSR
00589
00590                 movf    INDF,W
00591                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00592                 movlw   _Sub_Address_
00593                 movwf   INDF           ; store temporarily the sub-address at (_SourcePointer_ -1)
00594                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00595 ;
00596                 movf    StoreTemp_1,W
00597                 movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00598 ;
00599 ; Without sending a STOP bit, read a block of data by using a REPEATED
00600 ; Start Condition
00601 ;
00602                 I2C_READ        _COUNT2_, _DestPointer_
00603
00604                 ENDM
00605
00606
00607 ;*************************************************************************************************
00608 ;
00609 ;                               I2C_WR_COM_WR
00610 ;
00611 ;  This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
00612 ;  the bus after sending the first block.
00613 ;  For example, this kind of transaction is used in an LCD driver where a
```

```
00614 ;       block of control & addresss info is needed and then
00615 ;       another block of actual data to be displayed is needed.
00616 ;
00617 ;
00618 ;  Message Structure :
00619 ;              S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.......-A-D2[M-1]-A-P
00620 ;  NOTE : This message is same as calling two I2C_WR Macros, except that
00621 ;       the bus is not given up between the sending of 2 blocks (this is
00622 ;       done by not sending a STOP bit inbetween)
00623 ;
00624 ;  Parameters :
00625 ;              _COUNT1_               Length Of Source Buffer #1
00626 ;              _SourcePointer1_       Source Pointer Address of 1st buffer
00627 ;              _COUNT2_               The length of Destination Buffer
00628 ;              _SourcePointer2_       Source Pointer Address of 2nd Buffer
00629 ;
00630 ;*************************************************************************************************
00631
00632 I2C_WR_COM_WR   MACRO   _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_
00633
00634                movlw   _COUNT1_
00635                movwf   tempCount
00636                movlw   _SourcePointer1_
00637                movwf   FSR
00638                call    _i2c_block_write
00639 ;
00640 ; First block sent, now send 2nd block of data
00641 ;
00642                movlw   _COUNT2_
00643                movwf   tempCount
00644                movlw   _SourcePointer2__
00645                movwf   FSR
00646                call    _block_wr1_loop
00647 ;
00648                call    TxmtStopBit     ; End of Double buffer txmt
00649
00650                ENDM
00651
00652
00653 ;*************************************************************************************************
00654 ;                               INCLUDE I2C Low Level Routines Here
00655 ;*************************************************************************************************
00656
00657                        include “i2c_low.inc”
00001 ;*************************************************************************************************
00002 ;
00003 ;                               Low Level I2C Routines
```

```
                    00004 ;
                    00005 ;       Single Master Transmitter & Single Master Receiver Routines
                    00006 ;       These routines can very easily be converted to Multi-Master System
                    00007 ;       when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
                    00008 ;       detection is available.
                    00009 ;
                    00010 ;   The generic high level routines are given in I2C_HIGH.ASM
                    00011 ;
                    00012 ;
                    00013 ;       Program:          I2C_LOW.ASM
                    00014 ;       Revision Date:
                    00015 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                    00016 ;
                    00017 ;***********************************************************************************************
                    00018
                    00019
                    00020
                    00021
                    00022
                    00023 ;***********************************************************************************************
                    00024 ;                               I2C Bus Initialization
                    00025 ;
                    00026 ;***********************************************************************************************
004D                00027 InitI2CBus_Master:
                    00028
004D 1283           00029         bcf     STATUS,RP0
004E 0806           00030         movf    PORTB,W
004F 39FC           00031         andlw   0xFC            ; do not use BSF & BCF on Port Pins
0050 0086           00032         movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris
                    00033         RELEASE_BUS
0051 1683               M         bsf     STATUS,RP0      ; select Bank1
0052 1486               M         bsf     _SDA            ; tristate SDA
0053 1406               M         bsf     _SCL            ; tristate SCL
                        M ;       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
0054 0190           00034         clrf    Bus_Status      ; reset status reg
0055 0191           00035         clrf    Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008           00036         return
                    00037 ;
                    00038 ;***********************************************************************************************
                    00039 ;                                     Send Start Bit
                    00040 ;
                    00041 ;***********************************************************************************************
                    00042
0057                00043 TxmtStartBit:
0057 1683           00044          bsf     STATUS,RP0      ; select Bank1
0058 1486           00045          bsf     _SDA            ; set SDA high
0059 1406           00046          bsf     _SCL            ; clock is high
```

```
                    00047 ;
                    00048 ; Setup time for a REPEATED START condition (4.7 uS)
                    00049 ;
005A 210D           00050                 call    Delay40uSec     ; only necessary for setup time
                    00051 ;
005B 1086           00052                 bcf     _SDA            ; give a falling edge on SDA while clock is high
                    00053 ;
005C 210B           00054                 call    Delay47uSec     ; only necessary for START HOLD time
                    00055 ;
005D 1410           00056                 bsf     _Bus_Busy       ; on a start condition bus is busy
                    00057 ;
005E 0008           00058                 return
                    00059
                    00060
                    00061 ;*****************************************************************************************************
                    00062 ;                                       Send Stop Bit
                    00063 ;
                    00064 ;*****************************************************************************************************
                    00065
005F                00066 TxmtStopBit:
005F 1683           00067                 bsf     STATUS,RP0      ; select Bank1
0060 1006           00068                 bcf     _SCL
0061 1086           00069                 bcf     _SDA            ; set SDA low
0062 1406           00070                 bsf     _SCL            ; Clock is pulled up
0063 210D           00071                 call    Delay40uSec     ; Setup Time For STOP Condition
0064 1486           00072                 bsf     _SDA            ; give a rising edge on SDA while CLOCK is high
                    00073 ;
                    00074   if _ENABLE_BUS_FREE_TIME
                    00075 ; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
                    00076 ;
0065 210B           00077                 call    Delay47uSec
                    00078   endif
                    00079 ;
0066 1010           00080                 bcf     _Bus_Busy       ; on a stop condition bus is considered Free
                    00081 ;
0067 0008           00082                 return
                    00083
                    00084
                    00085 ;*****************************************************************************************************
                    00086 ;                                       Abort Transmission
                    00087 ;
                    00088 ;   Send STOP Bit & set Abort Flag
                    00089 ;*****************************************************************************************************
                    00090
0068                00091 AbortTransmission:
                    00092
0068 205F           00093                 call    TxmtStopBit
```

```
0069 1490          00094                bsf     _Abort
006A 0008          00095                return
                   00096
                   00097 ;*****************************************************************************************************
                   00098 ;                                Transmit Address (1st Byte)& Put in Read/Write Operation
                   00099 ;
                   00100 ;  Transmits Slave Addr On the 1st byte and set LSB to R/W operation
                   00101 ;  Slave Address must be loaded into SlaveAddr reg
                   00102 ;  The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
                   00103 ;
                   00104 ;  On Success, return TRUE in W register, else FALSE in W register
                   00105 ;
                   00106 ;   If desired, the failure may tested by the bits in Bus Status Reg
                   00107 ;
                   00108 ;*****************************************************************************************************
                   00109
006B               00110 Txmt_Slave_Addr:
006B 1390          00111         bcf     _ACK_Error              ; reset Acknowledge error bit
006C 1C11          00112         btfss   _10BitAddr
006D 2886          00113         goto    SevenBitAddr
                   00114 ;
006E 1C91          00115         btfss   _Slave_RW
006F 287D          00116         goto    TenBitAddrWR            ; For 10 Bit WR simply send 10 bit addr
                   00117 ;
                   00118 ; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
                   00119 ; and then Hi Byte Only for read opreation
                   00120 ;
0070               00121 TenBitAddrRd:
                   00122
0070 1091          00123         bcf     _Slave_RW               ; temporarily set for WR operation
0071 207D          00124         call    TenBitAddrWR
0072 1E10          00125         btfss   _Txmt_Success           ; skip if successful
0073 3400          00126         retlw   FALSE
                   00127
0074 2057          00128         call    TxmtStartBit            ; send A REPEATED START condition
0075 1491          00129         bsf     _Slave_RW               ; For 10 bit slave Read
                   00130
0076 080D          00131         movf    SlaveAddr+1,W
0077 008E          00132         movwf   DataByte
0078 140E          00133         bsf     DataByte,LSB            ; Read Operation
0079 2095          00134         call    SendData                ; send ONLY high byte of 10 bit addr slave
007A 288C          00135         goto    _AddrSendTest           ; 10 Bit Addr Send For Slave Read Over
                   00136 ;
                   00137 ; if successfully transmitted, expect an ACK bit
                   00138 ;
007B 1E10          00139         btfss   _Txmt_Success           ; if not successful, generate STOP & abort transfer
007C 2890          00140         goto    _AddrSendFail
```

```
                    00141 ;
                    00142
007D                00143 TenBitAddrWR:
                    00144
007D 080D           00145         movf    SlaveAddr+1,W
007E 008E           00146         movwf   DataByte
007F 100E           00147         bcf     DataByte,LSB            ; WR Operation
                    00148 ;
                    00149 ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
                    00150 ; then save RETURN Address Pointer
                    00151 ;
0080 2095           00152         call    SendData                ; send high byte of 10 bit addr slave
                    00153 ;
                    00154 ; if successfully transmitted, expect an ACK bit
                    00155 ;
0081 1E10           00156         btfss   _Txmt_Success           ; if not successful, generate STOP & abort transfer
0082 2890           00157         goto    _AddrSendFail
                    00158 ;
0083 080C           00159         movf    SlaveAddr,W
0084 008E           00160         movwf   DataByte                ; load addr to DatByte for transmission
0085 288B           00161         goto    EndTxmtAddr
                    00162
0086                00163 SevenBitAddr:
0086 080C           00164         movf    SlaveAddr,W
0087 008E           00165         movwf   DataByte                ; load addr to DatByte for transmission
0088 100E           00166         bcf     DataByte,LSB
0089 1891           00167         btfsc   _Slave_RW               ; if skip then write operation
008A 140E           00168         bsf     DataByte,LSB            ; Read Operation
                    00169
008B                00170 EndTxmtAddr:
008B 2095           00171         call    SendData                ; send 8 bits of address, bus is our's
                    00172 ;
                    00173 ; if successfully transmitted, expect an ACK bit
                    00174 ;
008C                00175 _AddrSendTest:
008C 1E10           00176         btfss   _Txmt_Success           ; skip if successful
008D 2890           00177         goto    _AddrSendFail
008E 0064           00178         clrwdt
008F 3401           00179         retlw   TRUE
                    00180 ;
0090                00181 _AddrSendFail:
0090 0064           00182         clrwdt
0091 1F90           00183         btfss   _ACK_Error
0092 3400           00184         retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                    00185 ;
                    00186 ; Address Not Acknowledged, so send STOP bit
                    00187 ;
```

```
0093 205F          00188         call    TxmtStopBit
0094 3400          00189         retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                   00190 ;
                   00191 ;*******************************************************************************************************
                   00192 ;                                 Transmit A Byte Of Data
                   00193 ;
                   00194 ; The data to be transmitted must be loaded into DataByte Reg
                   00195 ; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
                   00196 ; and INT Interrupt on Rising edge is enabled and also TMR0 timeout interrupt is enabled
                   00197 ;       The clock stretching slows down the transmit rate because all checking is done in
                   00198 ;       software. However, if the system has fast slaves and needs no clock stretching, then
                   00199 ;       this feature can be disabled during Assembly time by setting
                   00200 ;       _CLOCK_STRETCH_ENABLED must be set to FALSE.
                   00201 ;
                   00202 ;*******************************************************************************************************
0095               00203 SendData:
                   00204
                   00205 ;
                   00206 ; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
                   00207 ; For future compatibility, the user MUST call SendData & NOT TxmtByte
                   00208 ;
0095 2896          00209         goto    TxmtByte
                   00210
                   00211 ;
0096               00212 TxmtByte:
0096 080E          00213                 movf    DataByte,W
0097 0093          00214                 movwf   DataByteCopy    ; make copy of DataByte
0098 1510          00215                 bsf     _Txmt_Progress  ; set Bus status for txmt progress
0099 1210          00216                 bcf     _Txmt_Success   ; reset status bit
009A 3008          00217                 movlw   0x08
009B 008F          00218                 movwf   BitCount
009C 1683          00219                 bsf     STATUS,RP0
                   00220     if _CLOCK_STRETCH_CHECK
                   00221 ;        set TMR0 to INT CLK timeout for 1 mSec
                   00222 ;        do not disturb user's selection of RPUB in OPTION Register
                   00223 ;
009D 0801          00224                 movf    OPTION_REG,W
009E 39C3          00225                 andlw   _OPTION_INIT    ; defined in I2C.H header file
009F 0081          00226                 movwf   OPTION_REG
                   00227     endif
                   00228
00A0               00229 TxmtNextBit:
00A0 0064          00230                 clrwdt                  ; clear WDT, set for 18 mSec
00A1 1006          00231                 bcf     _SCL
00A2 0D93          00232                 rlf     DataByteCopy, F ; MSB first, Note DataByte Is Lost
00A3 1086          00233                 bcf     _SDA
00A4 1803          00234                 btfsc   STATUS,C
```

```
00A5 1486           00235                   bsf     _SDA
00A6 210B           00236                   call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00A7 1406           00237                   bsf     _SCL            ; set clock high, check if clock is high, else clock being stretched
00A8 210D           00238                   call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                    00239           if _CLOCK_STRETCH_CHECK
00A9 1283           00240                   bcf     STATUS,RP0
00AA 0181           00241                   clrf    TMR0            ; clear TMR0
00AB 110B           00242                   bcf     INTCON,T0IF     ; clear any pending flags
00AC 168B           00243                   bsf     INTCON,T0IE     ; elable TMR0 Interrupt
00AD 1391           00244                   bcf     _TIME_OUT_      ; reset timeout error flag
00AE                00245 Check_SCL_1:
00AE 1B91           00246                   btfsc   _TIME_OUT_      ; if TMR0 timeout or Error then Abort & return
00AF 28FC           00247                   goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00B0 1283           00248                   bcf     STATUS,RP0
00B1 1C06           00249                   btfss   _SCL            ; if clock not being stretched, it must be high
00B2 28AE           00250                   goto    Check_SCL_1     ; loop until SCL high or TMR0 timeout interrupt
00B3 128B           00251                   bcf     INTCON,T0IE     ; Clock good, disable TMR0 interrupts
00B4 1683           00252                   bsf     STATUS,RP0
                    00253           endif
00B5 0B8F           00254                   decfsz  BitCount, F
00B6 28A0           00255                   goto    TxmtNextBit
                    00256 ;
                    00257 ; Check For Acknowledge
                    00258 ;
00B7 1006           00259                   bcf     _SCL            ; reset clock
00B8 1486           00260                   bsf     _SDA            ; Release SDA line for Slave to pull down
00B9 210B           00261                   call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00BA 1406           00262                   bsf     _SCL            ; clock for slave to ACK
00BB 210D           00263                   call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
00BC 1283           00264                   bcf     STATUS,RP0      ; select Bank0 to test PortB pin SDA
00BD 1886           00265                   btfsc   _SDA            ; SDA should be pulled low by slave if OK
00BE 28C5           00266                   goto    _TxmtErrorAck
                    00267 ;
00BF 1683           00268                   bsf     STATUS,RP0
00C0 1006           00269                   bcf     _SCL            ; reset clock
                    00270
00C1 1110           00271                   bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
00C2 1610           00272                   bsf     _Txmt_Success   ; transmission successful
00C3 1390           00273                   bcf     _ACK_Error      ; ACK OK
00C4 0008           00274                   return
00C5                00275 _TxmtErrorAck:
                    00276                   RELEASE_BUS
00C5 1683               M                   bsf     STATUS,RP0      ; select Bank1
00C6 1486               M                   bsf     _SDA            ; tristate SDA
00C7 1406               M                   bsf     _SCL            ; tristate SCL
                        M ;                 bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
00C8 1110           00277                   bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
```

```
00C9 1210          00278                bcf     _Txmt_Success   ; transmission NOT successful
00CA 1790          00279                bsf     _ACK_Error      ; No ACK From Slave
00CB 0008          00280                return
                   00281 ;
                   00282 ;*****************************************************************************************************
                   00283 ;
                   00284 ;                                Receive  A Byte Of Data From Slave
                   00285 ;
                   00286 ;  assume address is already sent
                   00287 ;  if last byte to be received, do not acknowledge slave (last byte is testted from
                   00288 ;  _Last_Byte_Rcv bit of control reg)
                   00289 ;  Data Received on successful reception is in DataReg register
                   00290 ;
                   00291 ;
                   00292 ;*****************************************************************************************************
                   00293 ;
                   00294
00CC               00295 GetData:
00CC 28CD          00296                goto    RcvByte
                   00297 ;
00CD               00298 RcvByte:
                   00299
00CD 1590          00300                bsf     _Rcv_Progress   ; set Bus status for txmt progress
00CE 1290          00301                bcf     _Rcv_Success    ; reset status bit
                   00302
00CF 3008          00303                movlw   0x08
00D0 008F          00304                movwf   BitCount
                   00305        if _CLOCK_STRETCH_CHECK
00D1 1683          00306                bsf     STATUS,RP0
                   00307 ;        set TMR0 to INT CLK timeout for 1 mSec
                   00308 ;        do not disturb user's selection of RPBU in OPTION Register
                   00309 ;
00D2 0801          00310                movf    OPTION_REG,W
00D3 39C3          00311                andlw   _OPTION_INIT    ; defined in I2C.H header file
00D4 0081          00312                movwf   OPTION_REG
                   00313        endif
                   00314
00D5               00315 RcvNextBit:
00D5 0064          00316                clrwdt                  ; clear WDT, set for 18 mSec
00D6 1683          00317                bsf     STATUS,RP0      ; Bank1 for TRIS manipulation
00D7 1006          00318                bcf     _SCL
00D8 1486          00319                bsf     _SDA            ; can be removed from loop
00D9 210B          00320                call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00DA 1406          00321                bsf     _SCL            ; clock high, data sent by slave
00DB 210D          00322                call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                   00323        if _CLOCK_STRETCH_CHECK
00DC 1283          00324                bcf     STATUS,RP0
```

```
00DD 0181          00325                clrf    TMR0            ; clear TMR0
00DE 110B          00326                bcf     INTCON,T0IF     ; clear any pending flags
00DF 168B          00327                bsf     INTCON,T0IE     ; enable TMR0 Interrupt
00E0 1391          00328                bcf     _TIME_OUT_      ; reset timeout error flag
00E1               00329 Check_SCL_2:
00E1 1B91          00330                btfsc   _TIME_OUT_      ; if TMR0 timeout or Error then Abort & return
00E2 28FC          00331                goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00E3 1283          00332                bcf     STATUS,RP0
00E4 1C06          00333                btfss   _SCL            ; if clock not being stretched, it must be high
00E5 28E1          00334                goto    Check_SCL_2     ; loop until SCL high or TMR0 timeout interrupt
00E6 128B          00335                bcf     INTCON,T0IE     ; Clock good, diable TMR0 interrupts
00E7 1683          00336                bsf     STATUS,RP0
                   00337         endif
00E8 1283          00338                bcf     STATUS,RP0      ; select Bank0 to read Ports
00E9 1003          00339                bcf     STATUS,C
00EA 1886          00340                btfsc   _SDA
00EB 1403          00341                bsf     STATUS,C
                   00342 ;                                      ; TEMP ???? DO 2 out of 3 Majority detect
00EC 0D8E          00343                rlf     DataByte, F     ; left shift data ( MSB first)
00ED 0B8F          00344                decfsz  BitCount, F
00EE 28D5          00345                goto    RcvNextBit
                   00346 ;
                   00347 ; Generate ACK bit if not last byte to be read,
                   00348 ; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
                   00349 ;
00EF 1683          00350                bsf     STATUS,RP0
00F0 1006          00351                bcf     _SCL
00F1 1086          00352                bcf     _SDA            ; ACK by pulling SDA low
00F2 1911          00353                btfsc   _Last_Byte_Rcv
00F3 1486          00354                bsf     _SDA            ; if last byte, send NACK by setting SDA high
00F4 210B          00355                call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00F5 1406          00356                bsf     _SCL
00F6 210D          00357                call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
00F7               00358 RcvEnd:
00F7 1006          00359                bcf     _SCL            ; reset clock
                   00360
00F8 1190          00361                bcf     _Rcv_Progress   ; reset TXMT bit in Bus Status
00F9 1690          00362                bsf     _Rcv_Success    ; transmission successful
00FA 1390          00363                bcf     _ACK_Error      ; ACK OK
                   00364
00FB 0008          00365                return
                   00366
                   00367   if _CLOCK_STRETCH_CHECK
                   00368 ;*************************************************************************************************
                   00369 ;                                Fatal Error On I2C Bus
                   00370 ;
                   00371 ;  Slave pulling clock for too long or if SCL Line is stuck low.
```

```
                    00372 ;  This occurs if during Transmission, SCL is stuck low for period longer than appox 1mS
                    00373 ;   and TMR0 times out ( appox 4096 cycles : 256 * 16 -- prescaler of 16).
                    00374 ;
                    00375 ;*****************************************************************************************************
                    00376
00FC                00377 Bus_Fatal_Error:
                    00378
                    00379 ; disable TMR0 Interrupt
                    00380 ;
00FC 128B           00381         bcf     INTCON,T0IE                     ; disable TMR0 interrupts, until next TXMT try
                    00382
                    00383         RELEASE_BUS
00FD 1683               M                         bsf     STATUS,RP0      ; select Bank1
00FE 1486               M                         bsf     _SDA            ; tristate SDA
00FF 1406               M                         bsf     _SCL            ; tristate SCL
                        M ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                    00384 ;
                    00385 ;  Set the Bus_Status Bits appropriately
                    00386 ;
0100 1490           00387         bsf     _Abort                          ; transmission was aborted
0101 1710           00388         bsf     _Fatal_Error                    ; FATAL Error occured
0102 1110           00389         bcf     _Txmt_Progress                  ; Transmission Is Not in Progress
0103 1210           00390         bcf     _Txmt_Success                   ; Transmission Unsuccesful
                    00391 ;
0104 205F           00392         call    TxmtStopBit                     ; Try sending a STOP bit, may be not successful
                    00393 ;
0105 0008           00394         return
                    00395 ;
                    00396 ;*****************************************************************************************************
                    00397   endif
                    00398
                    00399
                    00400 ;*****************************************************************************************************
                    00401 ;                       General Purpose Delay Routines
                    00402 ;
                    00403 ;  Delay4uS     is wait loop for 4.0 uSec
                    00404 ;  Delay47uS    is wait loop for 4.7 uSec
                    00405 ;  Delay50uS    is wait loop for 5.0 uSec
                    00406 ;
                    00407 ;*****************************************************************************************************
                    00408 ;
                    00409
0106                00410 Delay50uSec:
0106 3006           00411         movlw   ((_50uS_Delay-5)/3 + 1)
0107                00412 DlyK
0107 0092           00413         movwf   DelayCount
0108 0B92           00414         decfsz  DelayCount, F
```

```
0109 2908           00415         goto    $-1
010A 0008           00416         return
                    00417 ;
010B                00418 Delay47uSec:
010B 3004           00419         movlw   ((_47uS_Delay-8)/3 + 1)
010C 2907           00420         goto    DlyK
                    00421 ;
010D                00422 Delay40uSec:
010D 3003           00423         movlw   ((_40uS_Delay-8)/3 + 1)
010E 2907           00424         goto    DlyK
                    00425 ;
                    00426 ;*************************************************************************************************
                    00109   endif
                    00110
                    00111 ;*************************************************************************************************
                    00112 ;
                    00113
010F                00114 ReadSlave1:
                    00115
                    00116 ;
                    00117 ;  EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
                    00118 ;
                    00119                 LOAD_ADDR_8  _Slave_1_Addr
                        M
010F 1011               M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0110 30A0               M                 movlw   (0xA0 & 0xff)
0111 008C               M                 movwf   SlaveAddr
                        M
0112                00120 wait1:
                    00121                 I2C_TEST_DEVICE
                        M
0112 2011               M                 call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                        M
0113 1F11           00122                 btfss   _SlaveActive    ; See If slave is responding
0114 2912           00123                 goto    wait1           ; if stuck forever, recover from WDT, can use other schemes
0115 0064           00124                 clrwdt
                    00125                 I2C_READ_SUB  8, DataBegin+1, 0x50
                        M
0116 1091               M                 bcf     _Slave_RW       ; set for write operation
0117 2057               M                 call    TxmtStartBit    ; send START bit
0118 206B               M                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                        M
0119 3050               M                 movlw   0x50
011A 008E               M                 movwf   DataByte        ; START address of EEPROM(slave 1)
011B 2095               M                 call    SendData        ; write sub address
                        M ;
                        M ; do not send STOP after this, use REPEATED START condition
```

```
                          M ;
                          M
                          M                 I2C_READ 8, DataBegin+1
                          M
                          M
011C 3007                 M                 movlw   (8 -1)
011D 0096                 M                 movwf   tempCount             ; -1 because, the last byte is used out of loop
011E 3021                 M                 movlw   DataBegin+1
011F 0084                 M                 movwf   FSR                   ; FIFO destination address pointer
                          M
0120 2039                 M                 call    _i2c_block_read
                          M
                          M
                          M
                        00126 ;
                        00127 ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
                        00128 ;
                        00129                 LOAD_ADDR_8  _Slave_2_Addr
                          M
0121 1011                 M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0122 30AC                 M                 movlw   (0xAC & 0xff)
0123 008C                 M                 movwf   SlaveAddr
                          M
0124                    00130 wait2:
                        00131                 I2C_TEST_DEVICE
                          M
0124 2011                 M                 call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                          M
0125 1F11               00132                 btfss   _SlaveActive    ; See If slave is responding
0126 2924               00133                 goto    wait2           ; if stuck forever, recover from WDT, can use other schemes
0127 0064               00134                 clrwdt
                        00135                 I2C_READ_SUB  8, DataBegin+1, 0x60
                          M
0128 1091                 M                 bcf     _Slave_RW       ; set for write operation
0129 2057                 M                 call    TxmtStartBit    ; send START bit
012A 206B                 M                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                          M
012B 3060                 M                 movlw   0x60
012C 008E                 M                 movwf   DataByte        ; START address of EEPROM(slave 1)
012D 2095                 M                 call    SendData        ; write sub address
                          M ;
                          M ; do not send STOP after this, use REPEATED START condition
                          M ;
                          M
                          M                 I2C_READ 8,       DataBegin+1
                          M
                          M
```

```
012E 3007               M                movlw   (8 -1)
012F 0096               M                movwf   tempCount               ; -1 because, the last byte is used out of loop
0130 3021               M                movlw   DataBegin+1
0131 0084               M                movwf   FSR                     ; FIFO destination address pointer
                        M
0132 2039               M                call    _i2c_block_read
                        M
                        M
                        M
                     00136
                     00137
0133 0008            00138        return
                     00139 ;
                     00140 ;*****************************************************************************************************
                     00141
0134                 00142 ReadSlave3:
                     00143
                     00144                LOAD_ADDR_8  _Slave_3_Addr
                        M
0134 1011               M                bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0135 30D6               M                movlw   (0xD6 & 0xff)
0136 008C               M                movwf   SlaveAddr
                        M
0137                 00145 wait3:
                     00146                I2C_TEST_DEVICE
                        M
0137 2011               M                call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                        M
0138 1F11            00147                btfss   _SlaveActive    ; See If slave is responding
0139 2937            00148                goto    wait3           ; if stuck forever, recover from WDT, can use other schemes
013A 0064            00149                clrwdt
                     00150                I2C_READ_SUB  8, DataBegin, 0
                        M
013B 1091               M                bcf     _Slave_RW       ; set for write operation
013C 2057               M                call    TxmtStartBit    ; send START bit
013D 206B               M                call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                        M
013E 3000               M                movlw   0
013F 008E               M                movwf   DataByte        ; START address of EEPROM(slave 1)
0140 2095               M                call    SendData        ; write sub address
                        M ;
                        M ; do not send STOP after this, use REPEATED START condition
                        M ;
                        M
                        M                I2C_READ 8, DataBegin
                        M
                        M
```

```
0141 3007              M                movlw   (8 -1)
0142 0096              M                movwf   tempCount             ; -1 because, the last byte is used out of loop
0143 3020              M                movlw   DataBegin
0144 0084              M                movwf   FSR                   ; FIFO destination address pointer
                       M
0145 2039              M                call    _i2c_block_read
                       M
                       M
                       M
                    00151 ;
                    00152
0146 0008           00153         return
                    00154
                    00155 ;*******************************************************************************************************
                    00156 ;
                    00157 ;                       Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
                    00158 ;
                    00159 ;*******************************************************************************************************
                    00160
0147                00161 FillDataBuf:
                    00162
0147 3000           00163         movlw   0x00            ; start address location of EEPROM array
0148 00A0           00164         movwf   DataBegin       ; 1st byte of data to be sent is start address
0149 3021           00165         movlw   DataBegin+1     ; data starts following address (RAM Pointer)
014A 0084           00166         movwf   FSR
014B 3008           00167         movlw   8               ; fill RAM with 8 bytes , this data is written to EEPROM (slave)
014C 009A           00168         movwf   byteCount
014D 3055           00169         movlw   0x55            ; pattern to fill with is 0x55 & 0xAA
014E 009B           00170         movwf   HoldData
014F                00171 X1:
014F 099B           00172         comf    HoldData, F
0150 081B           00173         movf    HoldData,W
0151 0080           00174         movwf   INDF
0152 0A84           00175         incf    FSR, F          ; point to next location
0153 0B9A           00176         decfsz  byteCount, F
0154 294F           00177         goto    X1
0155 0008           00178         return
                    00179 ;
                    00180 ;*******************************************************************************************************
                    00181 ;
                    00182 ;                          Main Routine  (Test Program)
                    00183 ;
                    00184 ;                       SINGLE  MASTER, MULTIPLE SLAVES
                    00185 ;
                    00186 ;*******************************************************************************************************
                    00187
0156                00188 Start:
```

```
0156 204D           00189         call    InitI2CBus_Master       ; initialize I2C Bus
0157 178B           00190         bsf     INTCON,GIE              ; enable global interrupts
                    00191 ;
                    00192
0158 2147           00193         call    FillDataBuf             ; fill data buffer with 8 bytes of data (0x55, 0xAA)
                    00194 ;
                    00195 ; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit Addr
                    00196 ;
                    00197 ;  Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
                    00198 ;
                    00199
0159 1810           00200  btfsc  _Bus_Busy ; is Bus Free, ie.has a start & stop bit been detected (only for multi master system)
015A 2959           00201         goto    $-1                     ; a very simple test, unused for now
                    00202
                    00203         LOAD_ADDR_8  _Slave_1_Addr
                        M
015B 1011               M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
015C 30A0               M                 movlw   (0xA0 & 0xff)
015D 008C               M                 movwf   SlaveAddr
                        M
                    00204         I2C_WR  0x09, DataBegin
                        M
015E 3009               M                 movlw   0x09
015F 0096               M                 movwf   tempCount
0160 3020               M                 movlw   DataBegin
0161 0084               M                 movwf   FSR
                        M
0162 2019               M                 call    _i2c_block_write
0163 205F               M                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                        M
                    00205 ;
                    00206 ; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
                    00207 ;
                    00208
0164 1810           00209  btfsc   _Bus_Busy  ; is Bus Free, ie. has a start & stop bit been detected (only for multi master system)
0165 2964           00210  goto    $-1             ; a very simple test, unused for now
                    00211
                    00212  LOAD_ADDR_8  _Slave_2_Addr
                        M
0166 1011               M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0167 30AC               M                 movlw   (0xAC & 0xff)
0168 008C               M                 movwf   SlaveAddr
                        M
                    00213         I2C_WR_SUB    0x08, DataBegin+1, 0x30
                        M
0169 3009               M                 movlw   (0x08 + 1)
016A 0096               M                 movwf   tempCount
```

```
                        M
016B 3020               M               movlw   (DataBegin+1 - 1)
016C 0084               M               movwf   FSR
                        M
016D 0800               M               movf    INDF,W
016E 0097               M               movwf   StoreTemp_1         ; temporarily store contents of (_SourcePointer_ -1)
016F 3030               M               movlw   0x30
0170 0080               M               movwf   INDF                ; store temporarily the sub-address at (_SourcePointer_ -1)
                        M
0171 2019               M               call    _i2c_block_write    ; write _BYTES_+1 block of data
                        M
0172 0817               M               movf    StoreTemp_1,W
0173 00A0               M               movwf   (DataBegin+1 - 1)   ; restore contents of (_SourcePointer_ - 1)
                        M
0174 205F               M               call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
                        M
                     00214
0175 210F            00215              call    ReadSlave1          ; read a byte from slave from current address
                     00216 ;
                     00217      LOAD_ADDR_8  _Slave_3_Addr
                        M
0176 1011               M               bcf     _10BitAddr          ; Set for 8 Bit Address Mode
0177 30D6               M               movlw   (0xD6 & 0xff)
0178 008C               M               movwf   SlaveAddr
                        M
0179 30CC            00218              movlw   0xCC
017A 00A0            00219              movwf   DataBegin
                     00220      I2C_WR_SUB      0x01,DataBegin, 0x33
                        M
017B 3002               M               movlw   (0x01 + 1)
017C 0096               M               movwf   tempCount
                        M
017D 301F               M               movlw   (DataBegin  - 1)
017E 0084               M               movwf   FSR
                        M
017F 0800               M               movf    INDF,W
0180 0097               M               movwf   StoreTemp_1        ; temporarily store contents of (_SourcePointer_ -1)
0181 3033               M               movlw   0x33
0182 0080               M               movwf   INDF               ; store temporarily the sub-address at (_SourcePointer_ -1)
                        M
0183 2019               M               call    _i2c_block_write   ; write _BYTES_+1 block of data
                        M
0184 0817               M               movf    StoreTemp_1,W
0185 009F               M               movwf   (DataBegin - 1)   ; restore contents of (_SourcePointer_ - 1)
                        M
0186 205F               M               call    TxmtStopBit       ; Issue a stop bit for slave to end transmission
                        M
```

```
                    00221 ;
0187 2134           00222         call    ReadSlave3              ; Read From Slave PIC
                    00223 ;
                    00224
0188 0064           00225 self    clrwdt
0189 2988           00226         goto    self
                    00227 ;
                    00228 ;*************************************************************************************************************
                    00229
                    00230         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXX------ ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   391
Program Memory Words Free:   633


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     4 suppressed
```

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

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable”.
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

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

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV MSC The Netherlands Accredited by the RvA — DNV — ISO 9001 / QS-9000 REGISTERED FIRM

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

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

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

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

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

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

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

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

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

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

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

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

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

ASIA/PACIFIC

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

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

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

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

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

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

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

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

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

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

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

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

EUROPE

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

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

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

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

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

01/18/02