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^{2}C Devices can send a Status Byte upon reception of the control byte.
This Macro reads a Status byte from a slave to the master's RAM location _DestPointer_. This function is basically the same as `I2C_READ` for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when `I2C_READ_STATUS` function is called. On successful operation of this command, 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       _ClkOut         equ     (_ClkIn >> 2)

               ;
               ; Compute the delay constants for setup & hold times
               ;
00000010       _40uS_Delay     set     (_ClkOut/250000)
00000012       _47uS_Delay     set     (_ClkOut/212766)
00000014       _50uS_Delay     set     (_ClkOut/200000)

               #define _OPTION_INIT    (0xC0 | 0x03)           ; Prescaler to TMR0 for Appox 1 mSec timeout
               ;
               #define _SCL    PORTB,0
               #define _SDA    PORTB,1

               #define _SCL_TRIS       _trisb,0
               #define _SDA_TRIS       _trisb,1

               #define _WRITE_         0
               #define _READ_          1


               ;                                Register File Variables


                               CBLOCK  0x0C
0000000C                               SlaveAddr               ; Slave Addr must be loaded into this reg
0000000D                               SlaveAddrHi             ; for 10 bit addressing mode
0000000E                               DataByte                ; load this reg with the data to be transmitted
0000000F                               BitCount                ; The bit number (0:7) transmitted or received
00000010                               Bus_Status              ; Status Reg of I2C Bus for both TXMT & RCVE
00000011                               Bus_Control             ; control Register of I2C Bus
00000012                               DelayCount
00000013                               DataByteCopy            ; copy of DataByte for Left Shifts (destructive)

00000014                               SubAddr                 ; sub-address of slave (used in I2C_HIGH.ASM)
00000015                               SrcPtr                  ; source pointer for data to be transmitted

00000016                               tempCount               ; a temp variable for scratch RAM
00000017                               StoreTemp_1             ; a temp variable for scratch RAM, do not disturb contents

00000018                               _End_I2C_Ram            ; unused, only for ref of end of RAM allocation
                               ENDC

               ;*************************************************************************************
               ;                       I2C Bus Status Reg Bit Definitions
               ;*************************************************************************************

```

```
#define _Bus_Busy       Bus_Status,0
#define _Abort          Bus_Status,1
#define _Txmt_Progress  Bus_Status,2
#define _Rcv_Progress   Bus_Status,3
#define _Txmt_Success   Bus_Status,4
#define _Rcv_Success    Bus_Status,5
#define _Fatal_Error    Bus_Status,6
#define _ACK_Error      Bus_Status,7
;*************************************************************************************
;                       I2C Bus Control Register
;*************************************************************************************
#define _10BitAddr      Bus_Control,0
#define _Slave_RW       Bus_Control,1
#define _Last_Byte_Rcv  Bus_Control,2
#define _SlaveActive    Bus_Control,6
#define _TIME_OUT_      Bus_Control,7
;**********************************************************************************************************
;                               General Purpose Macros
;**********************************************************************************************************
RELEASE_BUS     MACRO
                        bsf     STATUS,RP0      ; select Bank1
                        bsf     _SDA            ; tristate SDA
                        bsf     _SCL            ; tristate SCL
;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                ENDM
;**********************************************************************************************************
;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
;
;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
;  on 8 or 10 bit addressing modes
;**********************************************************************************************************
LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
```

```
                            bsf     _10BitAddr                            ; Slave has 10 bit address
                            movlw   (SLAVE_ADDRESS & 0xff)
                            movwf   SlaveAddr                             ; load low byte of address
                            movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)    ; 10 bit addr is 11110XX0
                            movwf   SlaveAddr+1                           ; hi order  address

                            ENDM

              LOAD_ADDR_8   MACRO   SLAVE_ADDRESS

                            bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                            movlw   (SLAVE_ADDRESS & 0xff)
                            movwf   SlaveAddr

                            ENDM


                            CBLOCK  _End_I2C_Ram
  00000018                          SaveStatus                            ; copy Of STATUS Reg
  00000019                          SaveW register                        ; copy of W register
  0000001A                          byteCount
  0000001B                          HoldData
                            ENDC

                            CBLOCK  0x20
  00000020                          DataBegin                             ; Data to be read or written is stored here
                            ENDC

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

0000                        ORG     0x00

0000 2815                   goto    Start
              ;
0004                        ORG     0x04
              ;*********************************************************************************************************
              ;                                Interrupt Service Routine
              ;
              ;   For I2C Slave routines, only Rb0/INT interrupt is used for START Bit Detect From
              ;       Idle Mode
              ;
              ;*********************************************************************************************************

0004          Interrupt:
              ;
              ;  At first check if START Bit Detect (currently only INT pin interrupt is enabled, if other
              ;       Interrupts are enabled, then check for other interrupts)
```

```
                  ;
0004 1C8B                 btfss   INTCON,INTF
0005 0009                 retfie                        ; other interrupt, simply return & enable GIE
                  ;
                  ; Save Status
                  ;
0006 0099                 movwf   SaveW register
0007 0E03                 swapf   STATUS,W              ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0008 0098                 movwf   SaveStatus
                  ;
0009 1283                 bcf     STATUS,RP0
000A 1C06                 btfss   _SCL         ; Most likely a START Bit, Hold Time of START Bit must be valid from an INT to here
000B 280F                 goto    RestoreStatus         ; Not a Valid START Bit
000C 1886                 btfsc   _SDA                  ; If a valid Falling Edge on SDA when SCL is high, SDA must now be Low
000D 280F                 goto    RestoreStatus         ; Not a Valid START Bit
000E 282A                 goto    StartBitDetect        ; A Valid Start Bit Is Detected, process & then Branch to "Restore Status"
                  ;
                  ; Restore Status
                  ;
000F              RestoreStatus:

000F 0E18                 swapf   SaveStatus,W
0010 0083                 movwf   STATUS                ; restore STATUS Reg
0011 0E99                 swapf   SaveW register, F     ; save W register
0012 0E19                 swapf   SaveW register,W      ; restore W register
                  ;
0013 108B                 bcf     INTCON,INTF
0014 0009                 retfie
                  ;

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

0015              Start:

0015 2019                 call    Init_I2C_Slave        ; Initialize I2C Bus for Slave Mode, wait for START Bit detect
                  ;
0016 178B                 bsf     INTCON,GIE            ; Enable Interrupts
0017              wait:
0017 0064                 clrwdt                        ; User can write code here, will be interrupted by Falling Edge on INT
0018 2817                 goto    wait                  ; Loop until Start Bit Detect


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

```
0019                  Init_I2C_Slave:

0019 01A0                              clrf    ControlByte
001A 1683                              bsf     STATUS,RP0
001B 30C3                              movlw   _OPTION_INIT
001C 0081                              movwf   OPTION_REG      ; initially set INT for Falling Edge INT
001D 1406                              bsf     _SCL
001E 1486                              bsf     _SDA            ; set TRIS of SCL & SDA to inputs, pulled up by external resistors

001F 1283                              bcf     STATUS,RP0
0020 0806                              movf    PORTB,W
0021 39FC                              andlw   0xFC            ; do not use BSF & BCF on Port Pins
0022 0086                              movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris

0023 0194                              clrf    SubAddr         ; Set Sub Address to Zero
                      ;
                      ; Initialize A/D Setup or EEPROM Control Regs
                      ;

                      ;#if AtoD_Slave
                      ;                movlw   0x01            ; Select Channel 0, and turn on A/D Module with Fosc/2 Sampling Rate
                      ;                movwf   _adcon0         ; already in Bank0, STATUS,RP0 = 0
                      ;                bsf     STATUS,RP0
                      ;                clrf   _adcon1          ; 4 analog channels with internal Vref
                      ;#else
0024 0189                              clrf    _eeadr          ; already in Bank0, init EEPROM Addr = 0
0025 1683                              bsf     STATUS,RP0
0026 0188                              clrf    _eecon1         ; clear err flag
                      ;#endif
                      ;
0027 018B                              clrf    INTCON
0028 160B                              bsf     INTCON,INTE     ; Enable Falling Edge Interrupt On SDA (connected on RB0/INT pin)
0029 0008                              return

                      ;
                      ;*****************************************************************************************************
                      ;                      In-Line Code For I2C-Slave
                      ;  Returns to detect next Start Bit after a STOP Bit Is Detected
                      ;  This implementation is very in-efficient if the Master is constantly sending a Repeated START Condition
                      ;
                      ;*****************************************************************************************************

002A                  StartBitDetect:
002A 2045                              call    RcvByte         ; 1st byte received in DataByte
002B 0064                              clrwdt
002C 1820                              btfsc   _STOP
```

```
002D 283F                  goto    i2c_start_wait  ; STOP bit Detected, wait for another START
002E 18A0                  btfsc   _START
002F 282A                  goto    StartBitDetect  ; a Repeated START condition
          ;
          ; The received byte in DataByte contains R/W & 7 Address the address
          ; If address match send ACK bit else NACK
          ;
0030 1120                  bcf     _RW
0031 180E                  btfsc   DataByte,LSB
0032 1520                  bsf     _RW             ; LSB of the 1st byte received contains R/W info
0033 100E                  bcf     DataByte,LSB

0034 080E                  movf    DataByte,W
0035 3AD6                  xorlw   _MY_ADDRESS     ; if match, then Z bit is set
0036 1D03                  btfss   STATUS,Z
0037 283D                  goto    SendNACK        ; No Address Match, NACK
          ;
          ; Address Match Occured, send ACK

0038 1403                  bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0039 20B3                  call    SendAck
003A 1920                  btfsc   _RW
003B 287B                  goto    SendReqData     ; read operation, so send current Data
003C 2867                  goto    RcvReqData      ; receive 2 bytes of data, sub-addr & data byte
          ;
          ;*****************************************************************************************************
          ;
003D      SendNACK:
003D 1003                  bcf     STATUS,C        ; SendAck routine sends NACK if Carry == 0
003E 20B3                  call    SendAck         ; address not us, so wait for another start bit
          ;
003F      i2c_start_wait:
003F 0064                  clrwdt
0040 1683                  bsf     STATUS,RP0
0041 108B                  bcf     INTCON,INTF
0042 1406                  bsf     _SCL            ; release CLK line, may be held in low by us
0043 1486                  bsf     _SDA            ; release SDA
0044 280F                  goto RestoreStatus
          ;
          ;*****************************************************************************************************
          ;                               Receive A Byte Of Data
          ;*****************************************************************************************************

0045      RcvByte:
0045 01A1                  clrf    SlvStatus

0046 3008                  movlw   0x08
```

```
0047 008F                       movwf   BitCount
                        ;
0048 1283                       bcf     STATUS,RP0
0049 1806                       btfsc   _SCL
004A 2849                       goto    $-1             ; wait until SCL is low and then Read the Control Byte
                        ;
004B 1683                       bsf     STATUS,RP0
004C 1406                       bsf     _SCL            ; release CLK, possibly held low
004D 1283                       bcf     STATUS,RP0
                        ;
004E                    RcvNextBit:
004E 1C06                       btfss   _SCL
004F 284E                       goto    $-1             ; wait until clock is high, for valid data
0050 1886                       btfsc   _SDA
0051 285A                       goto    Rcvd_1
0052                    Rcvd_0:
0052 1003                       bcf     STATUS,C
0053 0D8E                       rlf     DataByte, F     ; left shift data ( MSB first)
0054                    _WaitClkLo1:
0054 1C06                       btfss   _SCL
0055 2862                       goto    next1
0056 1C86                       btfss   _SDA            ; SDA must still be low when CLK is high, else may be a STOP
0057 2854                       goto    _WaitClkLo1
0058 1420                       bsf     _STOP
0059 0008                       return
                        ;
005A                    Rcvd_1:
005A 1403                       bsf     STATUS,C
005B 0D8E                       rlf     DataByte, F
005C                    _WaitClkLo2:
005C 1C06                       btfss   _SCL
005D 2862                       goto    next1         ; CLK went low, process next bit
005E 1886                       btfsc   _SDA          ; SDA must still be high when CLK is high, else may be a Repeated START
005F 285C                       goto    _WaitClkLo2
0060 14A0                       bsf     _START
0061 0008                       return
                        ;
0062                    next1:
0062 0B8F                       decfsz  BitCount, F
0063 284E                       goto    RcvNextBit
                        ;
                        ; A complete Byte Received, HOLD Master’s Clock Line Low to force a wait state
                        ;
0064 1683                       bsf     STATUS,RP0
0065 1006                       bcf     _SCL            ; force SCL Low for wait state
0066 0008                       return
                        ;*********************************************************************************************************
```

```
                  ;                       Write Operation Requested
                  ;
                  ; Read Sub Address and A Data Byte & Acknowledge, if no errors
                  ; Currently Only One Byte Is Programmed At A Time, Buffering scheme is unimplemented
                  ;
                  ;*****************************************************************************************************

0067              RcvReqData
0067 2045                        call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0068 080E                        movf    DataByte,W
0069 0094                        movwf   SubAddr
006A 0064                        clrwdt
006B 1820                        btfsc   _STOP
006C 283F                        goto    i2c_start_wait  ; STOP bit Detected, wait for another START
006D 18A0                        btfsc   _START
006E 282A                        goto    StartBitDetect  ; a Repeated START condition
006F 1403                        bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0070 20B3                        call    SendAck         ; Sub-Addr Received, Send an ACK
                  ;
                  ; Receive Data Byte to Write To EEPROM
                  ;
0071 2045                        call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0072 0064                        clrwdt
0073 1820                        btfsc   _STOP
0074 283F                        goto    i2c_start_wait  ; STOP bit Detected, wait for another START
0075 18A0                        btfsc   _START
0076 282A                        goto    StartBitDetect  ; a Repeated START condition
0077 1403                        bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0078 20B3                        call    SendAck         ; Sub-Addr Received, Send an ACK
0079 20CD                        call    EEpromWrite     ; Program EEPROM
007A 280F                        goto    RestoreStatus   ; STOP bit Detected, wait for another START

                  ;*****************************************************************************************************
                  ;                                Read Operation Requested
                  ;
                  ; Send A/D Data Of Required Channel until NACK By Master
                  ;
                  ;*****************************************************************************************************

007B              SendReqData:

007B 1283                        bcf     STATUS,RP0
007C 0814                        movf    SubAddr,W
007D 0089                        movwf   _eeadr          ; Load Sub_Addr Of EEPROM

                  ;
007E              SendNextByte:
```

```
007E 1683                   bsf     STATUS,RP0
007F 1408                   bsf     _rd             ; read EEPROM
0080 1283                   bcf     STATUS,RP0
0081 0808                   movf    _eedata,W
0082 008E                   movwf   DataByte        ; DataByte = EEPROM(addr)
0083 0A89                   incf    _eeadr, F       ; auto-increment sub-address

0084 208D                   call    TxmtByte        ; send out EEPROM Data
0085 0064                   clrwdt
0086 18A0                   btfsc   _START          ; check for abnormal START
0087 283F                   goto    i2c_start_wait
0088 20C0                   call    ReadACK
0089 0064                   clrwdt
008A 1DA0                   btfss   _ACK           ; _ACK == 1 if +ve ACK Rcvd
008B 283F                   goto    i2c_start_wait ; NACK Received, a START or STOP condition may occur
008C 287E                   goto    SendNextByte   ; continue to send until NACK
          ;
          ;*****************************************************************************************************

008D      TxmtByte:
008D 080E                   movf    DataByte,W
008E 0093                   movwf   DataByteCopy    ; make copy of DataByte
008F 3008                   movlw   0x08
0090 008F                   movwf   BitCount
0091 01A1                   clrf    SlvStatus
          ;
0092      TxmtNextBit:
0092 1683                   bsf     STATUS,RP0
0093 0D93                   rlf     DataByteCopy, F ; MSB First

0094 1803                   btfsc   STATUS,C
0095 28A1                   goto    Txmt_1
0096      Txmt_0
0096 1086                   bcf     _SDA
0097 0000                   nop
0098 1406                   bsf     _SCL            ; release clock line, let master pull it high
0099 1283                   bcf     STATUS,RP0
009A 1C06                   btfss   _SCL
009B 289A                   goto    $-1             ; wait until clk goes high
009C 1806                   btfsc   _SCL
009D 289C                   goto    $-1             ; wait until clk goes low
009E 1683                   bsf     STATUS,RP0
009F 1006                   bcf     _SCL            ; clk went low, continue to hold it low
00A0 28AD                   goto    Q_TxmtNextBit


00A1      Txmt_1
```


www.DataSheet4U.com

```
00A1 1486                bsf     _SDA
00A2 0000                nop
00A3 1406                bsf     _SCL            ; release clock line, let master pull it high
00A4 1283                bcf     STATUS,RP0
00A5 1C06                btfss   _SCL
00A6 28A5                goto    $-1             ; wait until clk goes high
00A7      _IsClkLo_1
00A7 1C86                btfss   _SDA
00A8 28B1                goto    MayBeErr_Txmt   ; must never come here, illegal Repeated Start
00A9 1806                btfsc   _SCL
00AA 28A7                goto    _IsClkLo_1      ; wait until clk goes low

00AB 1683                bsf     STATUS,RP0
00AC 1006                bcf     _SCL            ; clk went low, continue to hold it low
          ;
00AD      Q_TxmtNextBit
00AD 0B8F                decfsz  BitCount, F
00AE 2892                goto    TxmtNextBit
00AF 1486                bsf     _SDA            ; release SDA for Master's ACK
00B0 0008                return
          ;
00B1      MayBeErr_Txmt:
00B1 14A0                bsf     _START          ; illegal Repeated START condition during a byte transfer
00B2 0008                return
          ;
          ;*****************************************************************************************************
          ;                               Send ACK/NACK to Master
          ;
          ;  Prior to calling this routine, set CARRY bit to 1 for sending +ve ACK & set CARRY = 0, for NACK
          ;
          ;*****************************************************************************************************

00B3      SendAck:
00B3 1683                bsf     STATUS,RP0
00B4 1803                btfsc   STATUS,C        ; Carry bit == 1 for ACK else NACK
00B5 1086                bcf     _SDA            ; pull SDA low to send +ve ACK
00B6 1406                bsf     _SCL            ; release CLK line, let master clk it
00B7 1283                bcf     STATUS,RP0
00B8 1C06                btfss   _SCL
00B9 28B8                goto    $-1             ; loop until Clk High
00BA 1806                btfsc   _SCL
00BB 28BA                goto    $-1             ; loop until Clk is Low, ACK bit sent
00BC 1683                bsf     STATUS,RP0
00BD 1006                bcf     _SCL            ; HOLD CLK Line LOW
00BE 1486                bsf     _SDA            ; ACK over, release SDA line for Master control
          ;
          ;
```

```
00BF 0008                      return
                      ;
                      ;****************************************************************************************
                      ;                                       Read ACK Sent By Master
                      ;
                      ; If +ve ACK then set _ACK bit in SlaveStatus Reg, else 0
                      ;
                      ;****************************************************************************************

00C0                  ReadACK:
00C0 1683                      bsf     STATUS,RP0
00C1 1406                      bsf     _SCL            ; release clock
00C2 1283                      bcf     STATUS,RP0
00C3 1C06                      btfss   _SCL
00C4 28C3                      goto    $-1             ; wait until clock is high (9 the bit:ACK)
00C5 15A0                      bsf     _ACK            ; expecting a +ve ACK
00C6 1886                      btfsc   _SDA
00C7 11A0                      bcf     _ACK            ; NACK rcvd, stop transmission
00C8 1806                      btfsc   _SCL
00C9 28C8                      goto    $-1             ; wait until Clock is low
00CA 1683                      bsf     STATUS,RP0
00CB 1006                      bcf     _SCL            ; force Clock low
00CC 0008                      return
                      ;****************************************************************************************
                      ;                        Write One Byte Of Data To EEPROM Array at Sub-Addr
                      ;
                      ;****************************************************************************************
00CD                  EEpromWrite:
00CD 1283                      bcf     STATUS,RP0
00CE 0814                      movf    SubAddr,W
00CF 0089                      movwf   _eeadr          ; load sub-address
00D0 080E                      movf    DataByte,W
00D1 0088                      movwf   _eedata         ; load data to be written into EEDATA
00D2 1683                      bsf     STATUS,RP0
00D3 1508                      bsf     _wren           ; enable write operation
00D4 1188                      bcf     _wrerr          ; clear any previous error flags
00D5 3055                      movlw   0x55
00D6 0089                      movwf   _eecon2
00D7 30AA                      movlw   0xAA
00D8 0089                      movwf   _eecon2
00D9 1488                      bsf     _wr             ; start Write Operation
                      ;
00DA 1406                      bsf     _SCL            ; release CLK line, may be held in low by us
00DB 1486                      bsf     _SDA            ; release SDA
                      ;
00DC 0064                      clrwdt
00DD 1888                      btfsc   _wr             ; Poll until WR Over
```

```
00DE 28DC                goto    $-2
          ;
00DF 1108                bcf     _wren           ; disable write operations
00E0 0064                clrwdt
00E1 108B                bcf     INTCON,INTF
          ;
00E2 0008                return
          ;*****************************************************************************************************

               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
```

```
00000001  TRUE    equ     1
00000000  FALSE   equ     0

00000000  LSB     equ     0
00000007  MSB     equ     7

          ;
          #define _Slave_1_Addr   0xA0            ; Serial EEPROM #1
          #define _Slave_2_Addr   0xAC            ; Serial EEPROM #2
          #define _Slave_3_Addr   0xD6            ; Slave PIC16CXX

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


                          include         "i2c.h"
          ;*********************************************************************************************************
          ;                               I2C Bus Header File
          ;*********************************************************************************************************

003D0900  _ClkOut         equ     (_ClkIn >> 2)

          ;
          ; Compute the delay constants for setup & hold times
          ;
00000010  _40uS_Delay     set     (_ClkOut/250000)
00000012  _47uS_Delay     set     (_ClkOut/212766)
00000014  _50uS_Delay     set     (_ClkOut/200000)

          #define _OPTION_INIT    (0xC0 | 0x03)   ; Prescaler to TMR0 for Appox 1 mSec timeout
          ;
          #define _SCL    PORTB,0
          #define _SDA    PORTB,1

          #define _SCL_TRIS       _trisb,0
          #define _SDA_TRIS       _trisb,1

          #define _WRITE_         0
          #define _READ_          1


                                                  ; Register File Variables


                          CBLOCK  0x0C
```


www.DataSheet4U.com

```
0000000C                          SlaveAddr         ; Slave Addr must be loaded into this reg
0000000D                          SlaveAddrHi       ; for 10 bit addressing mode
0000000E                          DataByte          ; load this reg with the data to be transmitted
0000000F                          BitCount          ; The bit number (0:7) transmitted or received
00000010                          Bus_Status        ; Status Reg of I2C Bus for both TXMT & RCVE
00000011                          Bus_Control       ; control Register of I2C Bus
00000012                          DelayCount
00000013                          DataByteCopy      ; copy of DataByte for Left Shifts (destructive)

00000014                          SubAddr           ; sub-address of slave (used in I2C_HIGH.ASM)
00000015                          SrcPtr            ; source pointer for data to be transmitted

00000016                          tempCount         ; a temp variable for scratch RAM
00000017                          StoreTemp_1       ; a temp variable for scratch RAM, do not disturb contents

00000018                          _End_I2C_Ram      ; unused, only for ref of end of RAM allocation
                          ENDC

         ;*************************************************************************************
         ;                        I2C Bus Status Reg Bit Definitions
         ;*************************************************************************************

         #define _Bus_Busy       Bus_Status,0
         #define _Abort          Bus_Status,1
         #define _Txmt_Progress  Bus_Status,2
         #define _Rcv_Progress   Bus_Status,3

         #define _Txmt_Success   Bus_Status,4
         #define _Rcv_Success    Bus_Status,5
         #define _Fatal_Error    Bus_Status,6
         #define _ACK_Error      Bus_Status,7

         ;*************************************************************************************
         ;                        I2C Bus Contro Register
         ;*************************************************************************************
         #define _10BitAddr      Bus_Control,0
         #define _Slave_RW       Bus_Control,1
         #define _Last_Byte_Rcv  Bus_Control,2

         #define _SlaveActive    Bus_Control,6
         #define _TIME_OUT_      Bus_Control,7




         ;*****************************************************************************************************
         ;                                General Purpose Macros
```

```
         ;*****************************************************************************************************
         
         RELEASE_BUS     MACRO
                                 bsf     STATUS,RP0      ; select Bank1
                                 bsf     _SDA            ; tristate SDA
                                 bsf     _SCL            ; tristate SCL
         ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                         ENDM
         
         ;*****************************************************************************************************
         ;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
         ;
         ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
         ;  on 8 or 10 bit addressing modes
         ;*****************************************************************************************************
         
         LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
         
                         bsf     _10BitAddr      ; Slave has 10 bit address
                         movlw   (SLAVE_ADDRESS & 0xff)
                         movwf   SlaveAddr       ; load low byte of address
                         movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)     ; 10 bit addr is 11110XX0
                         movwf   SlaveAddr+1     ; hi order  address
         
                         ENDM
         
         LOAD_ADDR_8     MACRO   SLAVE_ADDRESS
         
                         bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                         movlw   (SLAVE_ADDRESS & 0xff)
                         movwf   SlaveAddr
         
                         ENDM
         
                         CBLOCK  _End_I2C_Ram
00000018                         SaveStatus              ; copy of STATUS Reg
00000019                         SaveW register          ; copy of W register
0000001A                         byteCount
0000001B                         HoldData
                         ENDC
         
                         CBLOCK  0x20
00000020                         DataBegin              ; Data to be read or written is stored here
                         ENDC
         
         
0000                     ORG     0x00
```

```
                                      
0000 2956                         goto    Start
                          ;
0004                              ORG     0x04
                          ;*****************************************************************************************************
                          ;                                Interrupt Service Routine
                          ;
                          ;   For I2C routines, only TMR0 interrupt is used
                          ; TMR0 Interrupts enabled only if Clock Stretching is Used
                          ; On TMR0 timeout interrupt, disable TMR0 Interrupt, clear pending flags,
                          ; MUST set _TIME_OUT_ flag saying possibly a FATAL error occurred
                          ; The user may choose to retry the operation again later
                          ;
                          ;*****************************************************************************************************

0004                      Interrupt:
                          ;
                          ; Save Interrupt Status (W register & STATUS regs)
                          ;
0004 0099                         movwf   SaveW register          ; Save W register
0005 0E03                         swapf   STATUS,W                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098                         movwf   SaveStatus              ; Save STATUS Reg
                           if _CLOCK_STRETCH_CHECK                ; TMR0 Interrupts enabled only if Clock Stretching is Used
0007 1D0B                         btfss   INTCON,T0IF
0008 280B                         goto    MayBeOtherInt           ; other Interrupts
0009 1791                         bsf     _TIME_OUT_              ; MUST set this Flag, can take other desired actions here
000A 110B                         bcf     INTCON,T0IF
                           endif
                          ;
                          ;  Check For Other Interrupts Here, This program usesd only TMR0 & INT Interrupt
                          ;
000B                      MayBeOtherInt:
000B 0000                         NOP
                          ;
000C                      RestoreIntStatus:                       ; Restore Interrupt Status
000C 0E18                         swapf   SaveStatus,W
000D 0083                         movwf   STATUS                  ; restore STATUS Reg
000E 0E99                         swapf   SaveW register, F
000F 0E19                         swapf   SaveW register,W        ; restore W register
0010 0009                         retfie
                          ;
                          ;*****************************************************************************************************
                          ;                                Include I2C High Level & Low Level Routines
                            if _INCLUDE_HIGH_LEVEL_I2C
                                  include         "i2c_high.inc"
                          ;*****************************************************************************************************
                          ;
```

```
;                               I2C Master : General Purpose Macros & Subroutines
;
;               High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
;
;
;       Program:          I2C_HIGH.ASM
;       Revision Date:
;                         1-16-97      Compatibility with MPASMWIN 1.40
;
;*********************************************************************************************************


;*********************************************************************************************************
;
;                               I2C_TEST_DEVICE
; MACRO
;
;   If Slave Device is listening, then _SlaveActive bit is set, else is cleared
;
;       Parameter :  NONE
;
;   Sequence Of Operations :
;       S-SlvAW-A-P
;   If A is +ve device is listening, else either busy, not present or error condition
;
;   This test may also be used to check for eample if a Serial EEPROM is in internal programming
;   mode
;
;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
;         mode addressing must be set
;*********************************************************************************************************

I2C_TEST_DEVICE         MACRO

                        call    IsSlaveActive           ; TEMP ???? : Assembler Error with this MACRO

                        ENDM
;
;
;               Test If A Device of SlaveAddr Is Present on Bus
;
;  The Slave Address Is put on the bus and if ACK it is present, if NACK not present
;  or maybe device is not responding. The presense can be checked constantly by a master
;  (for ex. the Operating System on an Access.Bus may constantly issue this command)
;
;   Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
;   Set  _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
```

```
                  ;
                  ;   Returns  1 in _SlaveActive Bit if slave is responding else a 0
                  ;
                  ;

0011              IsSlaveActive
0011 1091                         bcf     _Slave_RW       ; set for write operation
0012 2057                         call    TxmtStartBit    ; send START bit
0013 206B                         call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                  ;
0014 1311                         bcf     _SlaveActive
0015 1F90                         btfss   _ACK_Error      ; skip if NACK, device is not present or not responding
0016 1711                         bsf     _SlaveActive    ; ACK received, device present & listening
0017 205F                         call    TxmtStopBit
0018 0008                         return
                  ;
                  ;*****************************************************************************************************
                  ;                               I2C_WRITE
                  ;
                  ;  A basic macro for writing a block of data to a slave
                  ;
                  ;  Parameters :
                  ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_
                  ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                  ;
                  ;   Sequence :
                  ;               S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
                  ;
                  ;  If an error occurs then the routine simply returns and user should check for
                  ;       flags in Bus_Status Reg (for eg. _Txmt_Success flag)
                  ;
                  ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                  ;               mode addressing must be set
                  ;*****************************************************************************************************


                  I2C_WR          MACRO   _BYTES_, _SourcePointer_

                                  movlw   _BYTES_
                                  movwf   tempCount
                                  movlw   _SourcePointer_
                                  movwf   FSR

                                  call    _i2c_block_write
                                  call    TxmtStopBit     ; Issue a stop bit for slave to end transmission

                                  ENDM
```

```
                   
0019               _i2c_block_write:
0019 2057                         call    TxmtStartBit    ; send START bit
001A 1091                         bcf     _Slave_RW       ; set for write operation
001B 206B                         call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                   ;
001C               _block_wr1_loop:
001C 1E10                         btfss   _Txmt_Success
001D 0008                         return
001E 0800                         movf    INDF,W
001F 008E                         movwf   DataByte        ; start from the first byte starting at _DataPointer_
0020 0A84                         incf    FSR, F
0021 2095                         call    SendData        ; send next byte, bus is our’s !
0022 0B96                         decfsz  tempCount, F
0023 281C                         goto    _block_wr1_loop ; loop until desired bytes of data transmitted to slave
0024 0008                         return
                   ;
                   ;*******************************************************************************************************

                   ;*******************************************************************************************************
                   ;                            I2C_WRITE_SUB
                   ;
                   ;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
                   ;  to a slave device.
                   ;             Eg. : A serial EEPROM would need an address of memory location for Random Writes
                   ;
                   ;  Parameters :
                   ;               _BYTES_                #of bytes starting from RAM pointer _SourcePointer_ (constant)
                   ;               _SourcePointer_        Data Start Buffer pointer in RAM (file Registers)
                   ;               _Sub_Address_          Sub-address of Slave (constant)
                   ;
                   ;   Sequence :
                   ;               S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P
                   ;
                   ;  If an error occurs then the routine simply returns and user should check for
                   ;  flags in Bus_Status Reg (for eg. _Txmt_Success flag
                   ;
                   ;            Returns :  W register = 1 on success, else W register = 0
                   ;
                   ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                   ;         mode addressing must be set
                   ;
                   ;  COMMENTS :
                   ;               I2C_WR may prove to be more efficient than this macro in most situations
                   ;               Advantages will be found for Random Address Block Writes for Slaves with
                   ;               Auto Increment Sub-Addresses (like Microchip’s 24CXX series Serial EEPROMS)
                   ;
```

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

I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                movlw   (_BYTES_ + 1)
                movwf   tempCount

                movlw   (_SourcePointer_ - 1)
                movwf   FSR

                movf    INDF,W
                movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
                movlw   _Sub_Address_
                movwf   INDF           ; store temporarily the sub-address at (_SourcePointer_ -1)

                call    _i2c_block_write        ; write _BYTES_+1 block of data

                movf    StoreTemp_1,W
                movwf   (_SourcePointer_ - 1)   ; restore contents of (_SourcePointer_ - 1)

                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission

                ENDM

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

```
                   ;
                   ;********************************************************************************************************
                   
                   I2C_WR_SUB_SWINC        MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
                   
                                          variable i              ; TEMP ???? : Assembler Does Not Support This
                   
                                          i = 0
                   
                                            while (i < _BYTES_)
                                                   movf    (_Source_Pointer_ + i),W
                                                   movwf   SrcPtr
                                                   movf    (_Sub_Address_ + i),W
                                                   movwf   SubAddr
                                                   call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                   
                                                   i++
                                          endw
                   
                                           ENDM
                   
                   ;
                   ;
                   ;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
                   ;
                   
0025               _i2c_byte_wr_sub:
0025 2057                          call    TxmtStartBit    ; send START bit
0026 1091                          bcf     _Slave_RW       ; set for write operation
0027 206B                          call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
0028 1E10                          btfss   _Txmt_Success
0029 2835                          goto    _block_wr1_fail ; end
002A 0814                          movf    SubAddr,W
002B 008E                          movwf   DataByte        ; start from the first byte starting at _DataPointer_
002C 2095                          call    SendData        ; send next byte
002D 1E10                          btfss   _Txmt_Success
002E 2835                          goto    _block_wr1_fail ; end
002F 0815                          movf    SrcPtr,W
0030 008E                          movwf   DataByte        ; start from the first byte starting at _DataPointer_
0031 2095                          call    SendData        ; send next byte
0032 1E10                          btfss   _Txmt_Success
0033 2835                          goto    _block_wr1_fail ; failed, return 0 in W register
0034 2837                          goto    _block_wr1_pass ; successful, return 1 in W register
                   ;
                   ; return back to called routine from either _block_wr1_pass or _block_wr1_fail
                   ;
0035               _block_wr1_fail:
```

```
0035 205F                    call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0036 3400                    retlw   FALSE
0037          _block_wr1_pass:
0037 205F                    call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0038 3401                    retlw   TRUE
              ;

              ;*****************************************************************************************************
              ;
              ;                               I2C_WR_MEM_BYTE
              ;
              ;  Some I2C devices like a EEPROM need to wait fo some time after every byte write
              ;  (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
              ;  but in addition adds a delay after each byte.
              ;          Some EERPOM memories (like Microchip’s 24Cxx Series have on-chip data buffer), and hence
              ;               this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
              ;               for a block of data and then insert a delay until the whole buffer is written.
              ;
              ;  Parameters :
              ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
              ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
              ;               _Sub_Address_           Sub-address of Slave (constant)
              ;
              ;   Sequence :
              ;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
              ;               Delay 1 mSec            ; The user can chnage this value to desired delay
              ;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
              ;               Delay 1 mSec
              ;               and so on until #of Bytes
              ;
              ;*****************************************************************************************************

              I2C_WR_BYTE_MEM         MACRO   _BYTES_, _SourcePointer_, _Sub_Address_

                                     variable i              ; TEMP ???? : Assembler Does Not Support This

                                     i = 0

                                     while (i < _BYTES_)
                                              movf    (_Source_Pointer_ + i),W
                                              movwf   SrcPtr
                                              movf    (_Sub_Address_ + i),W
                                              movwf   SubAddr
                                              call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                                              call    Delay50uSec

                                              i++
```

```
                       endw


                        ENDM

;*********************************************************************************************************
;                     I2C_WR_MEM_BUF
;
;   This Macro/Function writes #of _BYTES_ to an I2C memory device. However
;   some devices, esp. EEPROMs must wait while the device enters into programming
;   mode. But some devices have an onchip temp data hold buffer and is used to
;   store data before the device actually enters into programming mode.
;   For example, the 24C04 series of Serial EEPROMs from Microchip
;   have an 8 byte data buffer. So one can send 8 bytes of data at a time
;   and then the device enters programming mode. The master can either wait
;   until a fixed time and then retry to program or can continiously poll
;   for ACK bit and then transmit the next Block of data for programming
;
;  Parameters :
;               _BYTES_                 # of bytes to write to memory
;               _SourcePointer_         Pointer to the block of data
;               _SubAddress_            Sub-address of the slave
;               _Device__BUF_SIZE_      The on chip buffer size of the i2c slave
;
;  Sequence of operations
;                       I2C_SUB_WR operations are performed in loop and each time
;                       data buffer of BUF_SIZE is output to the device. Then
;                       the device is checked for busy and when not busy another
;                       block of data is written
;
;
;*********************************************************************************************************

I2C_WR_BUF_MEM  MACRO   _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_


                variable i, j


                if ( !_BYTES_)
                        exitm

                elif ( _BYTES_ <=  _Device_BUF_SIZE_)

                        I2C_WR_SUB      _BYTES_, _SourcePointer_, _SubAddress_

                        exitm
```

```

                else

                        i = 0
                        j = (_BYTES_ / _Device_BUF_SIZE_)
                        while (i < j)

I2C_WR_SUB  _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_),(_SubAddress_ + i*_Device_BUF_SIZE_)

                                call    IsSlaveActive
                                btfss   _SlaveActive
                                goto    $-2

                          i++
                          endw

                        j = (_BYTES_ - i*_Device_BUF_SIZE_)

                        if (j)
I2C_WR_SUB j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_  + i*_Device_BUF_SIZE_)
                        endif

                endif

                ENDM


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

```

```
            I2C_READ        MACRO   _BYTES_, _DestPointer_


                            movlw   (_BYTES_ -1)
                            movwf   tempCount               ; -1 because, the last byte is used out of loop
                            movlw   _DestPointer_
                            movwf   FSR                     ; FIFO destination address pointer

                            call    _i2c_block_read

                            ENDM

0039        _i2c_block_read:
0039 2057                   call    TxmtStartBit    ; send START bit
003A 1491                   bsf     _Slave_RW       ; set for read operation
003B 1111                   bcf     _Last_Byte_Rcv  ; not a last byte to rcv
003C 206B                   call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
003D 1A10                   btfsc   _Txmt_Success
003E 2841                   goto    _block_rd1_loop ; end
003F 205F                   call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0040 3400                   retlw   FALSE           ; Error : may be device not responding
            ;
0041        _block_rd1_loop:
0041 20CC                   call    GetData
0042 080E                   movf    DataByte,W
0043 0080                   movwf   INDF            ; start receiving data, starting at Destination Pointer
0044 0A84                   incf    FSR, F
0045 0B96                   decfsz  tempCount, F
0046 2841                   goto    _block_rd1_loop ; loop until desired bytes of data transmitted to slave
0047 1511                   bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
0048 20CC                   call    GetData
0049 080E                   movf    DataByte,W
004A 0080                   movwf   INDF
004B 205F                   call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
004C 3401                   retlw   TRUE


            ;*********************************************************************************************************
            ;
            ;                               I2C_READ_SUB
            ;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
            ;  Between the sub-address write & the following reads, a STOP condition is not issued and
            ;  a “REPEATED START” condition is used so that an other master will not take over the bus,
            ;  and also that no other master will overwrite the sub-address of the same slave.
            ;
            ;   This function is very commonly used in accessing Random/Sequential reads from a
            ;   memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
```

```
;
;  Parameters :
;                _BYTES_         # of bytes to read
;                _DestPointer_   The destination pointer of data to be received.
;                _BubAddress_    The sub-address of the slave
;
;  Sequence :
;                S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
;
;
;*****************************************************************************************************

I2C_READ_SUB    MACRO   _BYTES_, _DestPointer_, _SubAddress_

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

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

                I2C_READ _BYTES_, _DestPointer_


                ENDM



;*****************************************************************************************************
;
;                               I2C_READ_STATUS
;
;  This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
;  send a status byte upon reception of a control byte
;  This is basically same as I2C_READ MACRO for reading a single byte
;
;   For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
;   data at the current address location
;
;  On success W register = 1 else = 0
;
;*****************************************************************************************************

```

```
I2C_READ_STATUS MACRO   _DestPointer_


                call    TxmtStartBit    ; send START bit
                bsf     _Slave_RW       ; set for read operation
                call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                btfsc   _Txmt_Success
                goto    _byte_rd1_loop  ; read a byte
                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                retlw   FALSE           ; Error : may be device not responding
_byte_rd1_loop:
                bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
                call    GetData
                movf    DataByte,W
                movwf   _DestPointer_
                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                btfss   _Rcv_Success
                retlw   FALSE
                retlw   TRUE

                ENDM

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

I2C_READ_BYTE   MACRO   _DestPointer_

                I2C_READ_STATUS MACRO   _DestPointer_

                ENDM


;*****************************************************************************************************
;
;                               I2C_WR_SUB_WR
;
;  This Macro writes 2 Blocks of Data (variable length) to a slave at a sub-address. This
;  may be useful for devices which need 2 blocks of data in which the first block may be an
;  extended address of a slave device. For example, a large I2C memory device, or a teletext
;  device with an extended addressing scheme, may need multiple bytes of data in the 1st block
;  that represents the actual physical address and is followed by a 2nd block that actually
;  represents the data.
;
;  Parameters :
;
;               _BYTES1_                1st block #of bytes
;               _SourcePointer1_        Start Pointer of the 1st block
;               _SubAddress_            Sub-Address of slave
```

```
;               _BYTES2_                2st block #of bytes
;               _SourcePointer2_        Start Pointer of the 2nd block
;
;  Sequence :
;               S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
;
;  Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
;               a STOP bit is not sent (bus is not given up) in between
;               the two I2C_WR_SUB
;
;  Check Txmt_Success flag for any transmission errors
;
;*****************************************************************************************************

I2C_WR_SUB_WR   MACRO   _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_

                movlw   (_COUNT1_ + 1)
                movwf   tempCount
                movlw   (_SourcePointer1_ - 1)
                movwf   FSR

                movf    INDF,W
                movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
                movlw   _Sub_Address_
                movwf   INDF           ; store temporarily the sub-address at (_SourcePointer_ -1)
                call    _i2c_block_write        ; write _BYTES_+1 block of data
;
                movf    StoreTemp_1,W
                movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
; Block 1 write over
; Send Block 2
                movlw   _COUNT2_
                movwf   tempCount
                movlw   _SourcePointer2_
                movwf   FSR
                call    _block_wr1_loop
;
                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission

                ENDM


;*****************************************************************************************************
;
;                               I2C_WR_SUB_RD
;
;  This macro writes a block of data from SourcePointer of length _COUNT1_ to a slave
```

```
;       at sub-address and then Reads a block of Data of length _COUNT2_ to destination
;       address pointer
;
;
;  Message Structure :
;              S-SlvW-A-SubA-A-D1[0]-A-.....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-......A-D2[M-1]-N-P
;
;  Parameters :
;               _COUNT1_                Length Of Source Buffer
;               _SourcePointer_         Source Pointer Address
;               _Sub_Address_           The Sub Address Of the slave
;               _COUNT2_                The length of Destination Buffer
;               _DestPointer_           The start address of Destination Pointer
;
;*****************************************************************************************************

I2C_WR_SUB_RD   MACRO   _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_

                movlw   (_COUNT1_ + 1)
                movwf   tempCount
                movlw   (_SourcePointer_ - 1)
                movwf   FSR

                movf    INDF,W
                movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
                movlw   _Sub_Address_
                movwf   INDF           ; store temporarily the sub-address at (_SourcePointer_ -1)
                call    _i2c_block_write        ; write _BYTES_+1 block of data
;
                movf    StoreTemp_1,W
                movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
;
; Without sending a STOP bit, read a block of data by using a REPEATED
; Start Condition
;
                I2C_READ        _COUNT2_, _DestPointer_

                ENDM


;*****************************************************************************************************
;
;                               I2C_WR_COM_WR
;
;  This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
;  the bus after sending the first block.
;  For example, this kind of transaction is used in an LCD driver where a
```

```
;       block of control & addresss info is needed and then
;       another block of actual data to be displayed is needed.
;
;
;  Message Structure :
;               S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-......-A-D2[M-1]-A-P
;  NOTE : This message is same as calling two I2C_WR Macros, except that
;       the bus is not given up between the sending of 2 blocks (this is
;       done by not sending a STOP bit inbetween)
;
;  Parameters :
;               _COUNT1_                Length Of Source Buffer #1
;               _SourcePointer1_        Source Pointer Address of 1st buffer
;               _COUNT2_                The length of Destination Buffer
;               _SourcePointer2_        Source Pointer Address of 2nd Buffer
;
;*****************************************************************************************************

I2C_WR_COM_WR   MACRO   _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_

                movlw   _COUNT1_
                movwf   tempCount
                movlw   _SourcePointer1_
                movwf   FSR
                call    _i2c_block_write
;
; First block sent, now send 2nd block of data
;
                movlw   _COUNT2_
                movwf   tempCount
                movlw   _SourcePointer2__
                movwf   FSR
                call    _block_wr1_loop
;
                call    TxmtStopBit     ; End of Double buffer txmt

                ENDM


;*****************************************************************************************************
;                               INCLUDE I2C Low Level Routines Here
;*****************************************************************************************************

                        include “i2c_low.inc”
;****************************************************************************************************
;
;                               Low Level I2C Routines
```

```
                ;
                ;       Single Master Transmitter & Single Master Receiver Routines
                ;       These routines can very easily be converted to Multi-Master System
                ;       when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
                ;       detection is available.
                ;
                ;   The generic high level routines are given in I2C_HIGH.ASM
                ;
                ;
                ;       Program:          I2C_LOW.ASM
                ;       Revision Date:
                ;                         1-16-97      Compatibility with MPASMWIN 1.40
                ;
                ;*****************************************************************************************





                ;*****************************************************************************************
                ;                              I2C Bus Initialization
                ;
                ;*****************************************************************************************
004D            InitI2CBus_Master:

004D 1283               bcf     STATUS,RP0
004E 0806               movf    PORTB,W
004F 39FC               andlw   0xFC            ; do not use BSF & BCF on Port Pins
0050 0086               movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris
                        RELEASE_BUS
0051 1683           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               clrf    Bus_Status      ; reset status reg
0055 0191               clrf    Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008               return
                ;
                ;*****************************************************************************************
                ;                                       Send Start Bit
                ;
                ;*****************************************************************************************

0057            TxmtStartBit:
0057 1683                bsf     STATUS,RP0      ; select Bank1
0058 1486                bsf     _SDA            ; set SDA high
0059 1406                bsf     _SCL            ; clock is high
```

```
                         ;
                         ; Setup time for a REPEATED START condition (4.7 uS)
                         ;
005A 210D                                call    Delay40uSec     ; only necessary for setup time
                         ;
005B 1086                                bcf     _SDA            ; give a falling edge on SDA while clock is high
                         ;
005C 210B                                call    Delay47uSec     ; only necessary for START HOLD time
                         ;
005D 1410                                bsf     _Bus_Busy       ; on a start condition bus is busy
                         ;
005E 0008                                return


                         ;*****************************************************************************************
                         ;                                        Send Stop Bit
                         ;
                         ;*****************************************************************************************

005F                     TxmtStopBit:
005F 1683                                bsf     STATUS,RP0      ; select Bank1
0060 1006                                bcf     _SCL
0061 1086                                bcf     _SDA            ; set SDA low
0062 1406                                bsf     _SCL            ; Clock is pulled up
0063 210D                                call    Delay40uSec     ; Setup Time For STOP Condition
0064 1486                                bsf     _SDA            ; give a rising edge on SDA while CLOCK is high
                         ;
                           if _ENABLE_BUS_FREE_TIME
                         ; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
                         ;
0065 210B                                call    Delay47uSec
                           endif
                         ;
0066 1010                                bcf     _Bus_Busy       ; on a stop condition bus is considered Free
                         ;
0067 0008                                return


                         ;*****************************************************************************************
                         ;                                        Abort Transmission
                         ;
                         ;   Send STOP Bit & set Abort Flag
                         ;*****************************************************************************************

0068                     AbortTransmission:

0068 205F                                call    TxmtStopBit
```

```
0069 1490                     bsf     _Abort
006A 0008                     return

        ;*****************************************************************************************************
        ;                              Transmit Address (1st Byte)& Put in Read/Write Operation
        ;
        ;  Transmits Slave Addr On the 1st byte and set LSB to R/W operation
        ;  Slave Address must be loaded into SlaveAddr reg
        ;  The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
        ;
        ;  On Success, return TRUE in W register, else FALSE in W register
        ;
        ;   If desired, the failure may tested by the bits in Bus Status Reg
        ;
        ;*****************************************************************************************************

006B            Txmt_Slave_Addr:
006B 1390               bcf     _ACK_Error              ; reset Acknowledge error bit
006C 1C11               btfss   _10BitAddr
006D 2886               goto    SevenBitAddr
                ;
006E 1C91               btfss   _Slave_RW
006F 287D               goto    TenBitAddrWR            ; For 10 Bit WR simply send 10 bit addr
                ;
                ; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
                ; and then Hi Byte Only for read opreation
                ;
0070            TenBitAddrRd:

0070 1091               bcf     _Slave_RW               ; temporarily set for WR operation
0071 207D               call    TenBitAddrWR
0072 1E10               btfss   _Txmt_Success           ; skip if successful
0073 3400               retlw   FALSE

0074 2057               call    TxmtStartBit            ; send A REPEATED START condition
0075 1491               bsf     _Slave_RW               ; For 10 bit slave Read

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

```
                  ;

007D              TenBitAddrWR:

007D 080D                 movf    SlaveAddr+1,W
007E 008E                 movwf   DataByte
007F 100E                 bcf     DataByte,LSB            ; WR Operation
                  ;
                  ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
                  ; then save RETURN Address Pointer
                  ;
0080 2095                 call    SendData                ; send high byte of 10 bit addr slave
                  ;
                  ; if successfully transmitted, expect an ACK bit
                  ;
0081 1E10                 btfss   _Txmt_Success           ; if not successful, generate STOP & abort transfer
0082 2890                 goto    _AddrSendFail
                  ;
0083 080C                 movf    SlaveAddr,W
0084 008E                 movwf   DataByte                ; load addr to DatByte for transmission
0085 288B                 goto    EndTxmtAddr

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

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

```
0093 205F               call    TxmtStopBit
0094 3400               retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                  ;
                  ;*****************************************************************************************************
                  ;                              Transmit A Byte Of Data
                  ;
                  ; The data to be transmitted must be loaded into DataByte Reg
                  ; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
                  ; and INT Interrupt on Rising edge is enabled and also TMR0 timeout interrupt is enabled
                  ;       The clock stretching slows down the transmit rate because all checking is done in
                  ;       software. However, if the system has fast slaves and needs no clock stretching, then
                  ;       this feature can be disabled during Assembly time by setting
                  ;       _CLOCK_STRETCH_ENABLED must be set to FALSE.
                  ;
                  ;*****************************************************************************************************
0095              SendData:

                  ;
                  ; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
                  ; For future compatibility, the user MUST call SendData & NOT TxmtByte
                  ;
0095 2896               goto    TxmtByte

                  ;
0096              TxmtByte:
0096 080E                       movf    DataByte,W
0097 0093                       movwf   DataByteCopy    ; make copy of DataByte
0098 1510                       bsf     _Txmt_Progress  ; set Bus status for txmt progress
0099 1210                       bcf     _Txmt_Success   ; reset status bit
009A 3008                       movlw   0x08
009B 008F                       movwf   BitCount
009C 1683                       bsf     STATUS,RP0
                      if _CLOCK_STRETCH_CHECK
                  ;        set TMR0 to INT CLK timeout for 1 mSec
                  ;        do not disturb user's selection of RPUB in OPTION Register
                  ;
009D 0801                       movf    OPTION_REG,W
009E 39C3                       andlw   _OPTION_INIT    ; defined in I2C.H header file
009F 0081                       movwf   OPTION_REG
                      endif

00A0              TxmtNextBit:
00A0 0064                       clrwdt                  ; clear WDT, set for 18 mSec
00A1 1006                       bcf     _SCL
00A2 0D93                       rlf     DataByteCopy, F ; MSB first, Note DataByte Is Lost
00A3 1086                       bcf     _SDA
00A4 1803                       btfsc   STATUS,C
```

```
00A5 1486                  bsf     _SDA
00A6 210B                  call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00A7 1406                  bsf     _SCL            ; set clock high, check if clock is high, else clock being stretched
00A8 210D                  call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                   if _CLOCK_STRETCH_CHECK
00A9 1283                  bcf     STATUS,RP0
00AA 0181                  clrf    TMR0            ; clear TMR0
00AB 110B                  bcf     INTCON,T0IF     ; clear any pending flags
00AC 168B                  bsf     INTCON,T0IE     ; elable TMR0 Interrupt
00AD 1391                  bcf     _TIME_OUT_      ; reset timeout error flag
00AE           Check_SCL_1:
00AE 1B91                  btfsc   _TIME_OUT_      ; if TMR0 timeout or Error then Abort & return
00AF 28FC                  goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00B0 1283                  bcf     STATUS,RP0
00B1 1C06                  btfss   _SCL            ; if clock not being stretched, it must be high
00B2 28AE                  goto    Check_SCL_1     ; loop until SCL high or TMR0 timeout interrupt
00B3 128B                  bcf     INTCON,T0IE     ; Clock good, disable TMR0 interrupts
00B4 1683                  bsf     STATUS,RP0
                   endif
00B5 0B8F                  decfsz  BitCount, F
00B6 28A0                  goto    TxmtNextBit
               ;
               ; Check For Acknowledge
               ;
00B7 1006                  bcf     _SCL            ; reset clock
00B8 1486                  bsf     _SDA            ; Release SDA line for Slave to pull down
00B9 210B                  call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00BA 1406                  bsf     _SCL            ; clock for slave to ACK
00BB 210D                  call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
00BC 1283                  bcf     STATUS,RP0      ; select Bank0 to test PortB pin SDA
00BD 1886                  btfsc   _SDA            ; SDA should be pulled low by slave if OK
00BE 28C5                  goto    _TxmtErrorAck
               ;
00BF 1683                  bsf     STATUS,RP0
00C0 1006                  bcf     _SCL            ; reset clock

00C1 1110                  bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
00C2 1610                  bsf     _Txmt_Success   ; transmission successful
00C3 1390                  bcf     _ACK_Error      ; ACK OK
00C4 0008                  return
00C5           _TxmtErrorAck:
                           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                  bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
```

```
00C9 1210                 bcf     _Txmt_Success   ; transmission NOT successful
00CA 1790                 bsf     _ACK_Error      ; No ACK From Slave
00CB 0008                 return
          ;
          ;*****************************************************************************************************
          ;
          ;                               Receive  A Byte Of Data From Slave
          ;
          ;  assume address is already sent
          ;  if last byte to be received, do not acknowledge slave (last byte is testted from
          ;  _Last_Byte_Rcv bit of control reg)
          ;  Data Received on successful reception is in DataReg register
          ;
          ;
          ;*****************************************************************************************************
          ;

00CC      GetData:
00CC 28CD                 goto    RcvByte
          ;
00CD      RcvByte:

00CD 1590                 bsf     _Rcv_Progress   ; set Bus status for txmt progress
00CE 1290                 bcf     _Rcv_Success    ; reset status bit

00CF 3008                 movlw   0x08
00D0 008F                 movwf   BitCount
                  if _CLOCK_STRETCH_CHECK
00D1 1683                 bsf     STATUS,RP0
          ;        set TMR0 to INT CLK timeout for 1 mSec
          ;        do not disturb user's selection of RPBU in OPTION Register
          ;
00D2 0801                 movf    OPTION_REG,W
00D3 39C3                 andlw   _OPTION_INIT    ; defined in I2C.H header file
00D4 0081                 movwf   OPTION_REG
                  endif

00D5      RcvNextBit:
00D5 0064                 clrwdt                  ; clear WDT, set for 18 mSec
00D6 1683                 bsf     STATUS,RP0      ; Bank1 for TRIS manipulation
00D7 1006                 bcf     _SCL
00D8 1486                 bsf     _SDA            ; can be removed from loop
00D9 210B                 call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00DA 1406                 bsf     _SCL            ; clock high, data sent by slave
00DB 210D                 call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                  if _CLOCK_STRETCH_CHECK
00DC 1283                 bcf     STATUS,RP0
```

```
00DD 0181                        clrf    TMR0             ; clear TMR0
00DE 110B                        bcf     INTCON,T0IF      ; clear any pending flags
00DF 168B                        bsf     INTCON,T0IE      ; enable TMR0 Interrupt
00E0 1391                        bcf     _TIME_OUT_       ; reset timeout error flag
00E1                Check_SCL_2:
00E1 1B91                        btfsc   _TIME_OUT_       ; if TMR0 timeout or Error then Abort & return
00E2 28FC                        goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00E3 1283                        bcf     STATUS,RP0
00E4 1C06                        btfss   _SCL             ; if clock not being stretched, it must be high
00E5 28E1                        goto    Check_SCL_2      ; loop until SCL high or TMR0 timeout interrupt
00E6 128B                        bcf     INTCON,T0IE      ; Clock good, diable TMR0 interrupts
00E7 1683                        bsf     STATUS,RP0
                         endif
00E8 1283                        bcf     STATUS,RP0       ; select Bank0 to read Ports
00E9 1003                        bcf     STATUS,C
00EA 1886                        btfsc   _SDA
00EB 1403                        bsf     STATUS,C
                    ;                                     ; TEMP ???? DO 2 out of 3 Majority detect
00EC 0D8E                        rlf     DataByte, F      ; left shift data ( MSB first)
00ED 0B8F                        decfsz  BitCount, F
00EE 28D5                        goto    RcvNextBit
                    ;
                    ; Generate ACK bit if not last byte to be read,
                    ; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
                    ;
00EF 1683                        bsf     STATUS,RP0
00F0 1006                        bcf     _SCL
00F1 1086                        bcf     _SDA             ; ACK by pulling SDA low
00F2 1911                        btfsc   _Last_Byte_Rcv
00F3 1486                        bsf     _SDA             ; if last byte, send NACK by setting SDA high
00F4 210B                        call    Delay47uSec      ; guarantee min LOW TIME tLOW & Setup time
00F5 1406                        bsf     _SCL
00F6 210D                        call    Delay40uSec      ; guarantee min HIGH TIME tHIGH
00F7                RcvEnd:
00F7 1006                        bcf     _SCL             ; reset clock

00F8 1190                        bcf     _Rcv_Progress    ; reset TXMT bit in Bus Status
00F9 1690                        bsf     _Rcv_Success     ; transmission successful
00FA 1390                        bcf     _ACK_Error       ; ACK OK

00FB 0008                        return

                      if _CLOCK_STRETCH_CHECK
                    ;*****************************************************************************************************
                    ;                               Fatal Error On I2C Bus
                    ;
                    ;  Slave pulling clock for too long or if SCL Line is stuck low.
```

```
                  ;  This occurs if during Transmission, SCL is stuck low for period longer than appox 1mS
                  ;   and TMR0 times out ( appox 4096 cycles : 256 * 16 -- prescaler of 16).
                  ;
                  ;*****************************************************************************************************

00FC              Bus_Fatal_Error:

                  ; disable TMR0 Interrupt
                  ;
00FC 128B                 bcf     INTCON,T0IE                     ; disable TMR0 interrupts, until next TXMT try

                          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
                  ;
                  ;  Set the Bus_Status Bits appropriately
                  ;
0100 1490                 bsf     _Abort                          ; transmission was aborted
0101 1710                 bsf     _Fatal_Error                    ; FATAL Error occured
0102 1110                 bcf     _Txmt_Progress                  ; Transmission Is Not in Progress
0103 1210                 bcf     _Txmt_Success                   ; Transmission Unsuccesful
                  ;
0104 205F                 call    TxmtStopBit                     ; Try sending a STOP bit, may be not successful
                  ;
0105 0008                 return
                  ;
                  ;*****************************************************************************************************
                    endif


                  ;*****************************************************************************************************
                  ;                       General Purpose Delay Routines
                  ;
                  ;  Delay4uS     is wait loop for 4.0 uSec
                  ;  Delay47uS    is wait loop for 4.7 uSec
                  ;  Delay50uS    is wait loop for 5.0 uSec
                  ;
                  ;*****************************************************************************************************
                  ;

0106              Delay50uSec:
0106 3006                 movlw   ((_50uS_Delay-5)/3 + 1)
0107              DlyK
0107 0092                 movwf   DelayCount
0108 0B92                 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
                     ;

0146 0008                    return

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

0147                 FillDataBuf:

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

0156                 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

0175 210F                      call    ReadSlave1          ; read a byte from slave from current address
                   ;
                       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                      movlw   0xCC
017A 00A0                      movwf   DataBegin
                       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
```

```
                      ;
0187 2134                     call    ReadSlave3              ; Read From Slave PIC
                      ;

0188 0064             self    clrwdt
0189 2988                     goto    self
                      ;
                      ;*****************************************************************************************************

                              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 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
```

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

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

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

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

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75248
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

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

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

New York
Microchip Technology Inc.
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

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing 100027 PRC
Tel: 86-10-85282100 Fax: 86-10-85282104

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
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
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700 Fax: 86 21-6275-5060

ASIA/PACIFIC (continued)

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

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

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

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

Italy
Arizona 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

11/15/99

DNV Certification, Inc. USA
DNV MSC The Netherlands Accredited by the RvA
ANSI • RAB QMS ACCREDITED
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.