AP-476

APPLICATION NOTE

How to Implement I^2C Serial Communication Using Intel MCS-51 Microcontrollers

SABRINA D. QUARLES
APPLICATIONS ENGINEER

April 1993

Order Number: 272319-001

How to Implement I^{2}C Serial Communication Using Intel MCS-51 Microcontrollers

CONTENTS PAGE

INTRODUCTION

Did you know that you could implement I^2C functionality using the Intel MCS-51 family of microcontrollers? The I^2C-bus allows the designer to implement intelligent application-oriented control circuits without encountering numerous interfacing problems. This bus simplicity is maintained by being structured for economical, efficient and versatile serial communication. Proven I^2C applications are currently being implemented in digital control/signal processing circuits for audio and video systems, DTMF generators for telephones with tone dialing and ACCESS.bus, a lower-cost alternative for the RS-232C interface used for connecting peripherals to a host computer.

This application note describes a software emulation implementation of the I^2C-bus Master-Slave configuration using Intel MCS-51 microcontrollers. It is recommended that the reader become familiar with the Phillips Semiconductors I^2C-bus Specification and the Intel MCS-51 Architecture. However, it is possible to gain a basic understanding of the I^2C-bus and the I^2C emulation software from this application note.

I^2C-Bus System

The Inter-Integrated Circuit Bus commonly known as the I^2C-bus is a bi-directional two-wire serial communication standard. It is designed primarily for simple but efficient integrated circuit (IC) control. The system is comprised of two bus lines, SCL (Serial Clock) and SDA (Serial Data) that carry information between the ICs connected to them. Various communication configurations may be designed using this bus; however, this application note discusses only the Master-Slave system implementation.

Devices connected to the I^2C-bus system can operate as Masters and Slaves. The Master device controls bus communications by initiating/terminating transfers, sending information and generating the I^2C system clock. On the other hand, the Slave device waits to be addressed by the controlling Master. Upon being addressed, the Slave performs the specific function requested. An example of this configuration is a Master Controller sending display data to a LED Slave Receiver that would then output the requested display.

The configuration described above is the most common; however, at times the Slave can become a Transmitter and the Master a Receiver. For example, the Master may request information from an addressed Slave. This requires the Master to receive data from the Slave. It is important to understand that even during Master Receive/Slave Transmission, the generation of clock signals on the I^2C bus is always the responsibility of the Master. As a result, all events on the bus must be synchronized with the Master's SCL clock line.

I^2C Hardware Characteristics

Both SCL (Serial Clock) and SDA (Serial Data) are bi-directional lines that are connected to a positive supply voltage via pull-up resistors. Figure 1 displays a typical I^2C-bus configuration. Devices connected to the bus require open-drain or open-collector output stage interfaces. As a result of these interfaces, the resistors pull both lines HIGH when the bus is free. The free state is defined as SDA and SCL HIGH when the bus is not in use.

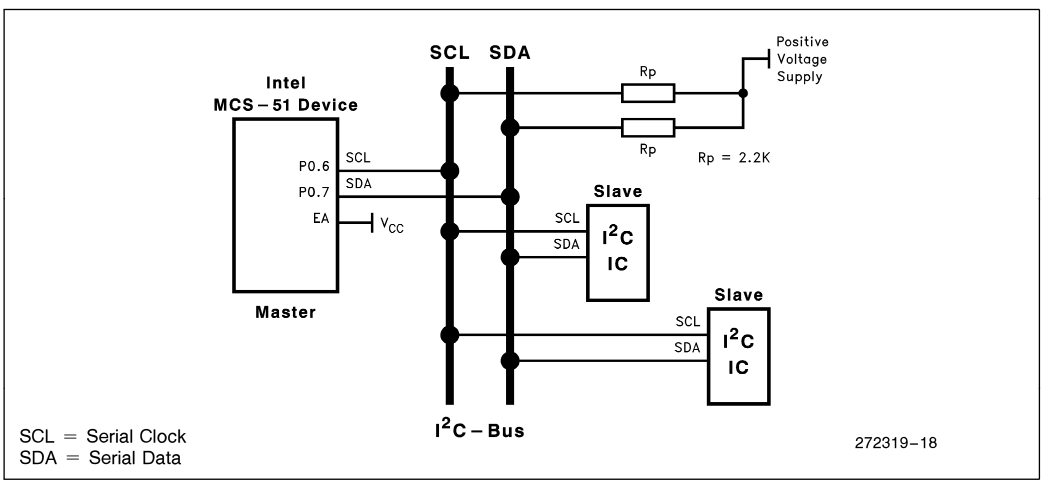


Figure 1. I^2C Master/Slave Bus System

One important bus characteristic enabled as a result of this hardware configuration is the wired-AND function. Similar to the logic AND truth table, when driven by connected ICs, I^{2}C-bus lines will not indicate the HIGH state until all devices verify that they too have achieved the same HIGH state. An I^{2}C-bus system relies on wired-AND functionality to maintain appropriate clock synchronization and to communicate effectively with extremely high and low speed devices. As a result, a relatively slow I^{2}C device can extend the system clock until it is ready to accept more data.

I^{2}C Protocol Characteristics

This section will explain a complete I^{2}C data transfer emphasizing data validity, information types, byte formats, and acknowledgment. Figure 2-1 displays the typical I^{2}C protocol data transfer frame. The important frame components are the START/STOP conditions, Slave Address, and Data with Acknowledgment. This frame structure remains constant except for the number of data bytes transferred and the transmission direction. It can be seen that all functionality except Acknowledgment is generated by the Master and current transmitter. Figure 2-2 displays a more detailed representation focusing on specific timing sequences of control signals and data transfers.

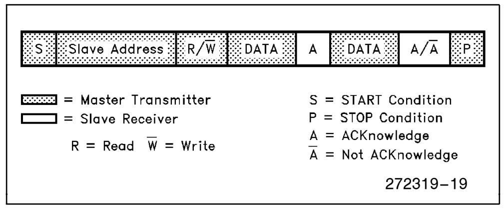


Figure 2-1. I^{2}C Protocol Data Transfer Frame

DATA VALIDITY

Figure 3 shows the bit transfer protocol that must be maintained on the I^{2}C-bus. The data on the SDA line must be stable during the HIGH period of the SCL clock. The HIGH or LOW state of SDA can only change when the clock signal on the SCL is LOW. In addition, these bus lines must meet required setup, hold and rise/fall times prescribed in the timing section of the I^{2}C protocol specifications.

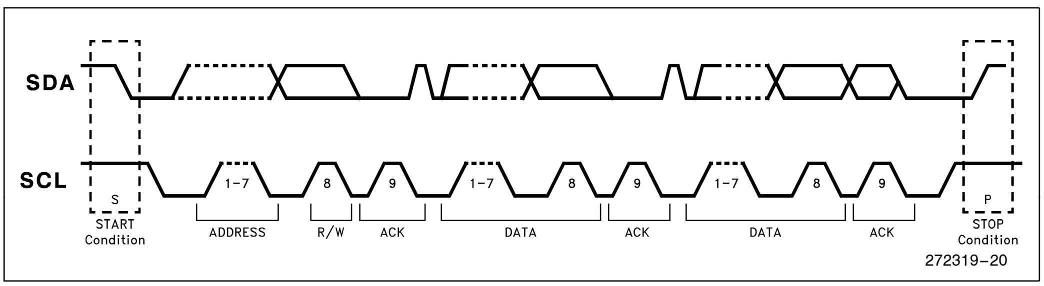


Figure 2-2. A Complete I^{2}C Data Transfer

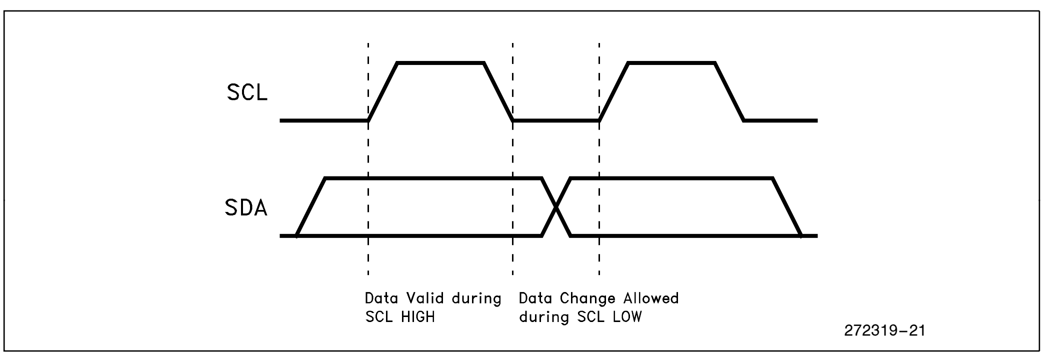


Figure 3. Bit Transfer on the I^{2}C-Bus

Control Signals

START and STOP conditions are used to signal the beginning and end of data communications. A Master generates a START condition (S) to obtain control of a free I^2C-bus by forcing a HIGH to LOW transition on the SDA line while maintaining SCL in its HIGH state. This condition is generated during software emulation in the MASTER—CONTROLLER subroutine described in another section. Again, START conditions may be generated by a Master only when the I^2C-bus is free. This free bus state exists only when no other Master devices have control of the bus (i.e. both SCL and SDA lines are pulled to their normal HIGH state).

Upon gaining control of the bus, the Master must transfer data across the system. After a complete data transfer, the Master must release the bus by generating a STOP (P) condition. The SEND_STOP subroutine described in a later section ends data communications by sending an I^2C STOP.

Data Transfers

The Slave address and data being transferred across the bus must conform to specific byte formats. The only byte transmission requirement is that data must be transferred with its Most Significant Bit (MSB) first. However, the number of bytes that can be transmitted per transfer is unrestricted. For both Master Transmit/Receive, the MASTER_CONTROLLER subroutine described in a later section performs these functions.

From Figure 4, it can be seen that the Slave address is one byte made up of a unique 7-bit address followed by a Read or Write data direction indicator bit. The Least Significant Bit (LSB) data direction indicator, always determines the direction of the message and type of transfer being requested by the Master—either Slave Receive or Slave Transmit. If the Master requests the Slave Receive functionality, the LSB of the addressed Slave would be set to "0" for Write. Therefore, the Master would Transmit or Write information to the selected Slave. On the other hand, if the Master was requesting the Slave Transmit functionality, the LSB would be set to "1" for Read. As a result, the Master would Receive or Read information from the Slave. SEND_DATA and RECV_DATA subroutines described later send and receive data bytes across the bus.

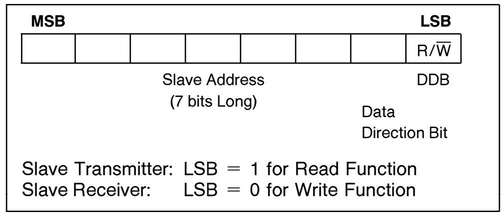


Figure 4. Slave Address Byte Format

Address Recognition

When an address is sent from the controlling Master, each device in a system compares the first 7 bits after the START condition with its predefined unique Slave address. If they match, the device considers itself addressed by the Master as either a Slave-Receiver or Slave-Transmitter, depending upon the data direction indicator. Due to the bus's serial configuration, only one device at a time may be addressed and communicated with at any given moment.

ACKNOWLEDGMENT

To ensure valid and reliable I^2C-bus communication, an obligatory data transfer acknowledgment procedure was devised. Figure 5 displays how acknowledgment

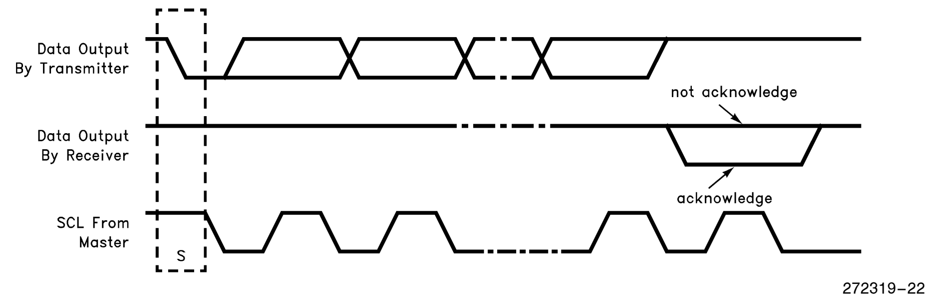


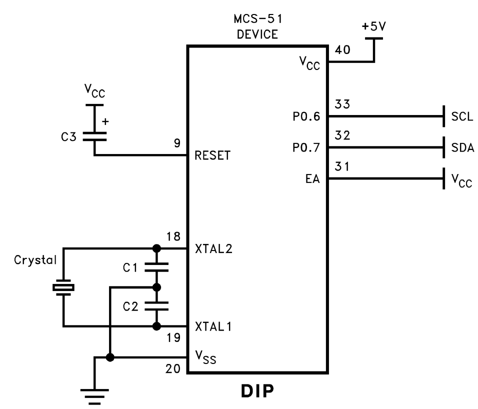
Figure 5. Acknowledgement of the I2C-Bus

always affects the Master, Transmitter and Receiver. Mter every byte transfer, the Master must generate an acknowledge related clock pulse. In Figure 1, this clock pulse is indicated as the 9th bit and labeled "ACK". Following the 8th data bit transmission, the active Transmitter must immediately release the SDA line enabling it to float HIGH. To receive another data byte, the Receiver must verify successful receipt of the previous byte by generating an acknowledgment. An acknowledge condition is delivered when the Receiver drives SDA LOW so that it remains stable LOW during the HIGH period of the SCL ACK pulse. Conversely, a not acknowledge condition is delivered when the Receiver leaves SDA HIGH. Set-up and hold times must always be taken into account and maintained for valid communications. SEND__BYTE and RECV__BYTE subroutines described later evaluate and/or generate acknowledgment conditions.

MCS-51 Hardware Requirements

The I2C protocol requires open-drain device outputs to drive the bus. To satisfy this specification, Port 0 on the Intel MCS-51 device was chosen. By using open-drain Port 0, no additional hardware is required to successfully interface to the I2C-bus. However, since Port 0 is designated as the I2C interface, it can no longer be used to interface with External Program Memory. In order for a MCS-51 device to communication in this environment, ASM51 software emulation modules were developed. This software can only execute out of Internal Memory. Port 0 is now configured for Input/Output functionality.

Figure 6 diagrams the necessary hardware connections of the development circuit. Internal Memory execution is accomplished by connecting the External Access (EA) DIP pin #31 to V_{CC}. The capacitor attached to RESET DIP pin #9 implements POWER ON RESET. While the capacitors and crystal attached to XTAL1&2 enable the on-chip oscillator, additional decoupling capacitors can be added to clean up any system noise. Additional MCS-51 information can be found in the 1992 Intel Embedded Microcontrollers and Processors Handbook Volume 1.



C1 = C2 = 30 pF
C3 = 10 pF

Figure 6. MCS-51 Hardware Requirements

The ASM51 software emulation modules described in this application note will occupy approximately 540 bytes of internal memory. The device's remaining memory may be programmed with user software. The following MCS-51 devices were tested for use in conjunction with the I^2C emulation modules:

MCS-51 Devices	Crystal Speeds (MHz)	ROM/ EPROM Size	Register RAM
8751BH	12	4K	128 bytes
87C51	12, 16, 20	4K	128 bytes
87C51-FX Core	12, 16, 20, 24	4K	128 bytes
87C51FA	12, 16, 20, 24	8K	256 bytes
87C51FB	12, 16, 20, 24	16K	256 bytes
87C51FC	12, 16, 20, 24	32K	256 bytes

NOTE:
The Internal memory setup described above eliminates the option of using Port 0 to interface to External Memory. However, this requirement should pose no problem for the system designer due to the diverse MCS-51 product line with various memory sizes offered by Intel.

MCS-51 I^2C Software Emulation Modules

When devices like the MCS-51 do not incorporate an on-chip I^2C port, I^2C functionality can be achieved through software emulation. The following software modules are based upon three distinct tasks: bus monitoring, time delays and bus control. Each task conforms to the I^2C protocol as specified by Philips Semiconductors.

The software modules designed to implement I^2C functionality are comprised of macros and subroutines, each independently developed, yet both networked to achieve a desired system function. For example, the use of macros was favored to implement certain timing delay loops. Macros are extremely flexible and can be changed to construct delays of varying lengths throughout the software. On the other hand, subroutines are verified routines that require no additional changes. To operate the bus at different frequencies, only the specific macros must be changed, not the predefined subroutines. The following ASM51 macros and subroutines are for Master-Slave system control:

Macro Names	**Functions**
DELAY_3_CYCLES	Delay loop for X seconds where X = time per cycle * 3
DELAY_4_CYCLES	Delay loop for X seconds where X = time per cycle * 4
*	*
*	*
DELAY_8_CYCLES	Delay loop for X seconds where X = time per cycle * 8
RELEASE_SCL_HIGH	Releases the SCL line HIGH and waits for any clock stretching requests from peripheral devices

Subroutine Names	**Functions**
MASTER_CONTROLLER	Sends an I^2C START condition and Slave Address during both a Master Transmit and Receive
SEND_DATA	Sends multiple data bytes during a Master Transmit
SEND_BYTE	Sends one data byte line during a Master Transmit
SEND_MSG	Sends a message across the I^2C bus using a predefined format
RECV_DATA	Receives multiple data bytes from an addressed Slave during a Master Receive
RECV_BYTE	Receives one data byte during a Master Receive
RECV_MSG	Receives a message from the I^2C bus using a predefined format
TRANSFER	Copies EPROM programmed data into Register RAM
SEND_STOP	Send an I^2C STOP condition during both a Master Transmit/Receive

These ASM51 modules are listed at the end of the application note in Appendix A.

MCS-51 and I²C-Bus Compatible IC's System Implementation

This section of the application note explains the Master/Slave system diagrammed in Figure 1. The Intel MCS-51 is the Master Controller communicating with two I²C Slave peripherals, the PCF8570 RAM chip and SAAI064 LED driver. Information related to communicating with these specific Slave devices can be found in the 1992 Philips I²C Peripherals for Microcontrollers Handbook.

The MCS-51 I²C Software Emulation Modules located in Appendix A are designed to demonstrate Master Controller functionality.

As described above, the Intel 51 Master Controller transmits data to the RAM device, receives it back and re-transmits it to the LED Slave driver. By using the SEND_MSG and RECV_MSG subroutines, both Master Transmit and Master Receive functionalities are demonstrated. Slave addresses used in these transfers are predefined values assigned by their manufacturer. These values can be found in their respective databooks.

An I²C Master Transmission consists of the following steps:

1. Master polls the bus to see if free state exists
2. Master generates a START condition on the bus
3. Master broadcasts the Slave Address expecting an Acknowledge from the addressed Slave
4. Master transmits data bytes expecting acknowledgment status following each byte
5. Master generates a STOP condition and releases the bus

An I²C Master/Receive transaction consists of the exact same steps stated above EXCEPT:

4. Master receives data bytes sending an ACK to the Slave Transmitter after receipt of each byte. The Master signals receipt of the last data byte by responding with the NOT Acknowledge condition.

MASTER TRANSMIT/RECEIVE

Bus transmission and evaluation is achieved by a nested loop structure. SEND_DATA represents the outer loop which directs data transfers. The MASTER_CONTROLLER subroutine polls the bus to determine if any transactions are in progress. Error checking is performed at this level by evaluating the following status flags, BUS_FAULT and I²C_BUSY. Based upon this information, the Master will either abort the transmit procedure or attempt to send information. If bus control is granted as indicated by cleared flags, the Master sends a START condition and the Slave address. On the other hand, if either flag is set, the transmit procedure is aborted.

SEND_BYTE, the inner control loop, is responsible for transmitting 8 bits of each byte as well as monitoring Slave acknowledgment status. Each bit transfer from I²C-bus lines checks for possible serial wait states. Wait states occur when slower devices need to communicate on the bus with faster devices. Due to the wired-AND bus function, a Receiver can hold the clock line SCL LOW forcing the Transmitter into this state. Data transfer may continue when the Receiver is ready for another byte of data as indicated by releasing the clock line SCL HIGH.

As stated in its section above, acknowledgment is required to continue sending data bytes across the bus. However, situations may arise when a Receiver can not receive another byte of data until it has performed some other function like servicing internal interrupts. If the Slave Receiver does not respond to a Master Transmitter data byte, not acknowledge could indicate that it is performing some real-time function that prevents it from responding to I²C-bus communications. This situation shows the flexibility and versatility of the bus.

The Master Receive process also utilizes the MASTER_CONTROLLER subroutine to gain control of the bus. When accepting data from the addressed Slave, in this case, RECV_DATA is the outer control loop. RECV_BYTE, the inner control loop, is responsible for receiving 8 bits of each byte as well as generating the Master's acknowledgment condition. Similar to transmission, successful receipt of each byte is confirmed by driving SDA LOW so that it remains stable LOW during the HIGH period of the SCL ACK pulse. Therefore, the Master still drives both SCL and SDA lines since control of the system clock is its responsibility.

In both types of communication, Transmit/Receive, temporary RAM registers, BIT_CNT, BYTE_CNT, SLV_ADDR, and storage buffers, XMT_DAT, RCV_DAT, ALT_XMT, are integral parts of most subroutines because they are used for implementing the I²C protocol. Proper delays are implemented using the DELAY_X_CYCLES (X = any integer) macros. They give the designer flexibility to devise time delays of any required length to satisfy system requirements. For example, to achieve the maximum bus speeds described in the next section, Delay_X_Cycle macros were adjusted.

Lastly, the TRANSFER subroutine is provided to allow predefined communication data programmed in the microcontrollers EPROM to be transferred into Register RAM internal to the 51 device. It achieves this when used in conjunction with the SEND_MSG and RECV_MSG subroutines. However, when utilizing TRANSFER, the designer must conform their design to existing device Register RAM availability and to the following message format:

Slave Address, # of Bytes to be Transmitted/Received, Data Bytes (For Transmit Only)

The ASM-51 program demonstrating a complete Master Controller system is listed at the end of the application note in Appendix B. It writes the numeric data that represents the following display "_I^2C" to an I^2C compatible IC (PCF8570 RAM), reads the values back into a buffer and transmits this buffer out to the Philips I^2C SAA1064 LED driver to display the sequence.

I^2C Software Emulation Performance

As demonstrated above, the Intel MCS-51 product line can successfully implement the I^2C Master Controller functionality while maintaining data integrity and reliable performance. The system outlined in Figure 1 was evaluated for maximum bus performance and adherence to all I^2C-bus specifications. Performance characterization was conducted at various crystal speeds on all devices listed in the MCS-51 Hardware Requirements section of this application note.

When designing I^2C software emulation systems, keep in mind that the designer has the flexibility to implement large frequency ranges up to the I^2C-bus maximum. However, by making software changes to adjust bus frequencies, the newly modified program may no longer meet required specifications and desired reliability standards. Therefore, designers should first always take into consideration the bus performance level they want to reach. After deciding this, an appropriate crystal can be chosen to achieve that implementation speed. The table below gives a few examples of system performance for two of the MCS-51 devices:

MCS-51 Devices	Crystal Speed	I^2C Bus Maximum Performance
8751BH	12 MHz	66.7 kHz
87C51 (FX-Core)	24 MHz	80.0 kHz

CONCLUSION

As a result of this evaluation, Intel MCS-51 microcontrollers can be successfully interfaced to an I^2C-bus system as a Master controller. The interface communicates by ASM51 software emulation modules that have been tested on a wide array of I^2C devices ranging from serial RAMS, Displays and a DTMF generators. No compatibility problems have been seen to date. Therefore, when considering the implementation of your next I^2C-bus Master Controller serial communication system, you have the option of using the Intel MCS-51 Product Line.

REFERENCES

I^2CBITS.ASM, G. Goodhue, Philips Semiconductors, August 1992.

The I^2C-Bus and How to Use It (Including Specification), Philips Semiconductors, January 1992.

I^2C Peripherals for Microcontrollers, Philips Semiconductors, 1992 Data Handbook.

OM1016 I^2C Evaluation Board, E. Rodgers and G. Moss, Philips Components Applications Lab Auckland, New Zealand.

Programming the I^2C Interface, Mitchell Kahn, Senior Engineer, Intel Corporation.

APPENDIX A

```
;------------------------------------------------------------------------
;
;       INTEL MCS-51 MASTER CONTROLLER MODULES
;
;       The following ASM51 software emulation modules are used to develop I2C-bus
;       functionality with Intel MCS-51 microcontrollers.  They are described in detail in
;       FaxBACK document #2175 and BBS document AP476.ZIP.
;
;       Written By:     Sabrina Quarles
;                       Intel Corporation
;                       EMD 8-Bit Applications Engineering        Rev. 1.0
;
;       Date:           December 1, 1992
;
;------------------------------------------------------------------------



;------------------------------------------------------------------------
;
;       SEND STOP Subroutine
;
;       This program sends an I2C STOP condition to release the bus.
;
;------------------------------------------------------------------------

SEND_STOP:
        CLR     SDA_PIN                 ;Get SDA ready for stop.
        %RELEASE_SCL_HIGH               ;Set clock for stop.
        %DELAY_3_CYCLES                 ;Delay.
        SETB    SDA_PIN                 ;Send I2C STOP.
                                        ;Delay satisfied via software.
        CLR     I2C_BUSY                ;Clear I2C busy status.
        RET                             ;Bus should now be released.



;------------------------------------------------------------------------
;
;       SEND_MSG Subroutine
;
;       This subroutine sends a message across the I2C bus using the
;       information stored in the XMT_DAT Buffer in the following format:
;
;       Buffer @R0 = SlvAddr, # of Bytes to be Transferred, Data Bytes
;
;------------------------------------------------------------------------
```

272319–1

```
SEND_MSG:
        MOV SLV_ADDR, @R0               ;Initializes Slave Address.
        INC R0                          ;Next address.
        MOV BYTE_CNT,@R0                ;Initializes BYTE_CNT.
        INC R0                          ;Next address.
        ACALL SEND_DATA                 ;Send Data.
        RET                             ;Return from Subroutine.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;   MASTER CONTROLLER Subroutine
;
;   This subroutine sends an I2C START condition and Slave Address to
;   begin I2C communications.
;
;     SDA = Receive/Transmit Data
;     SCL = Generate/Control Clock Line
;
;
;   SLV_ADDR = Slave Address
;
;   Verification
;     Issues before MASTER TRANSMIT
;        * No Bus Fault = Bus Not Busy = SCL & SDA HIGH
;
;
;      Issues during MASTER TRANSMIT
;        *  ACK Received after every Byte Transmission
;
;
;     SUBROUTINES Used
;        SEND_BYTE
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

MASTER_CONTROLLER:

        SETB   I2C_BUSY                 ;Indicate that I2C frame is in progress.
        CLR    NO_ACK                   ;Clear error status flags.
        CLR    BUS_FAULT
        JNB    SCL_PIN, FAULT           ;Check for bus clear.
        JNB    SDA_PIN, FAULT
        CLR    SDA_PIN                  ;Begin I2C start.
        %DELAY_3_CYCLES                 ;Delay.
        CLR    SCL_PIN                  ;Complete I2C START.
        %DELAY_3_CYCLES                 ;Delay.
        MOV    A, SLV_ADDR              ;Get slave address.
        ACALL  SEND_BYTE                ;Send slave address.
        RET


FAULT:
        SETB   BUS_FAULT                ;Set fault status.
        RET                             ; and return.
```

272319–2

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;   MASTER TRANSMIT ~ SEND_BYTE Subroutine
;
;       This subroutine sends 1 byte of information located in the ACCumulator
;          ACC = Byte to be Transmitted
;
;     Verification Issues
;        * ACK Received after transmission of Byte
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SEND_BYTE:
        MOV    BIT_CNT, #8              ;Set bit count value.

SB_LOOP:
        RLC    A                        ;Send one data bit.
        MOV    SDA_PIN, C               ;Put data bit on pin.
        %RELEASE_SCL_HIGH               ;Drive SCL HIGH.
        %DELAY_3_CYCLES                 ;Delay.

        CLR    SCL_PIN                  ;Clear SCL.
        %DELAY_3_CYCLES                 ;Delay.
        DJNZ   BIT_CNT,SB_LOOP          ;Repeat until all bits sent.

        SETB   SDA_PIN                  ;Release data line for acknowledge.
        %RELEASE_SCL_HIGH               ;Send clock for acknowledge.
        %DELAY_4_CYCLES                 ;Delay.
        JNB    SDA_PIN, SB_EX           ;Check for valid acknowledge bit.
        SETB   NO_ACK                   ;Set status for no acknowledge.

SB_EX:
        CLR    SCL_PIN                  ;Finish acknowledge bit.
        %DELAY_3_CYCLES                 ;Delay.
        RET                             ;Return.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;  MASTER TRANSMIT ~ SEND DATA Subroutine
;
;  This subroutine transmits multiple data bytes over the SDA line.
;  The following locations must be initialized before the transmission.
;
;     BYTE_CNTR = # of bytes to be transmitted
;     SLV_ADDR  = Slave Address
;     @R0       = Data to be Transmitted
;                 ~ includes any additional subaddresses, control, etc
;                   specific to certain devices
;
;     SUBROUTINES Used
;        MASTER_XMIT
;           SEND_BYTE
;        SEND_BYTE
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

272319–3

```
SEND_DATA:
        ACALL   MASTER_CONTROLLER       ;Acquire bus and send slave address.
        JB      NO_ACK,SDEX             ;Check for slave not responding.
SD_LOOP:
        MOV     A, @R0                  ;Get data byte from buffer.
        ACALL   SEND_BYTE               ;Send next data byte.
        INC     R0                      ;Advance buffer pointer.
        JB      NO_ACK,SDEX             ;Check for slave not responding.
        DJNZ    BYTE_CNT, SD_LOOP       ;All bytes sent?

SDEX:
        ACALL   SEND_STOP               ;Done, send an I2C stop.
        RET                             ;Return.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       TRANSFER Subroutine
;
;       This subroutine copies data from the EPROM referenced by DPTR into a
;       Buffer referenced by R1.
;
;       DPTR = String stored into EPROM
;       R1 = Buffer in which data shall be stored
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TRANSFER:
        CLR A                           ;Clears ACC.

        MOVC A, @A+DPTR                 ;Moves contents of DPTR into A.
        MOV @R1, A                      ;Copies A into Buffer.
        INC R1                          ;Next address.
        INC DPTR                        ;Next location.
        CLR A                           ;Clears ACC.

        MOVC A, @A+DPTR                 ;Moves contents of DPTR into A.
        MOV @R1, A                      ;Copies A into Buffer.
        MOV R0, A                       ;Copies A into R0 (# of bytes).
        INC R1                          ;Next address.
        INC DPTR                        ;Next location.
        CLR A                           ;Clears A.

NEXT:
        MOVC A, @A+DPTR                 ;Moves contents of DPTR into A.
        DEC R0                          ;Decrease # of remaining bytes.
        MOV @R1, A                      ;Copies A  into Buffer.
        INC R1                          ;Next address.
        INC DPTR                        ;Next location.
        CLR A                           ;Clears A.
        CJNE R0, #0, NEXT               ;Compare # of bytes remaining.
        RET                             ;If all bytes copied, return.
```

272319–4

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;        RECV_MSG Subroutine
;
;        This subroutine receives a message from the I2C bus using SLV_ADDR
;        and BYTE_CNT as indicators as to what Slave will be sending info and
;        how many bytes to expect to receive, and places the data into the
;        RCV_DAT buffer.  The RCV_DAT Buffer is configured to receive a max. 8 bytes.
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RECV_MSG:
        MOV SLV_ADDR, @R1               ;Moves SLV_ADDR from Buffer R0 points to.
        INC R1                          ;Next buffer location.
        MOV BYTE_CNT, @R1               ;Moves BYTE_CNT value into memory location.
        ACALL RCV_DATA                  ;Calls RCV_DATA Subroutine.
        RET                             ;Returns from Receive Msg subroutine.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;        MASTER RECEIVE ~ RECEIVE BYTE Subroutine
;
;        This subroutine receives a byte from an addressed I2C slave
;        device and places into the ACC register.
;
;        ACC = Data Byte Received
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RECV_BYTE:
        MOV    BIT_CNT,#8               ;Set bit count.
RB_LOOP:
        %RELEASE_SCL_HIGH               ;Read one data bit.
        %DELAY_3_CYCLES                 ;Delay.
        MOV    C, SDA_PIN               ;Get data bit from pin.
        RLC    A                        ;Rotate bit into result byte.
        CLR    SCL_PIN                  ;Clear SCL pin.
        %DELAY_3_CYCLES                 ;Delay.
        DJNZ   BIT_CNT, RB_LOOP         ;Repeat until all bits received.

        PUSH   ACC                      ;Save accumulator.
        MOV    A, BYTE_CNT              ;Copies byte count into A.
        CJNE   A, #1, RB_ACK            ;Check for last byte of frame.
        SETB   SDA_PIN                  ;Send no acknowledge on last byte.
        SJMP   RB_ACLK                  ;No ACK on last byte; jump to RB_ACLK.
RB_ACK:
        CLR    SDA_PIN                  ;Send acknowledge bit.
RB_ACLK:
        %RELEASE_SCL_HIGH               ;Send acknowledge clock.
        POP    ACC                      ;Restore accumulator.
        %DELAY_3_CYCLES                 ;Delay.
        CLR    SCL_PIN                  ;Clear SCL pin.
        SETB   SDA_PIN                  ;Clear acknowledge bit.
        %DELAY_4_CYCLES                 ;Delay.
        RET                             ;Return from RECV_BYTE.
```

272319–5

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       MASTER RECEIVE ~ RECEIVE DATA BYTES Subroutine
;
;       This subroutine receives multiple data bytes from an addressed
;       I2C slave device into the buffer pointed to by R0.
;
;       BYTE_CNT = # of bytes to be received
;       SLV_ADDR  = Slave addres
;
;       @R0 = location of received data
;
;       SUBROUTINES Used
;               MASTER_XMIT
;               RCV_BYTE
;
;       Note:  To receive with a subaddress, use SEND_DATA to set the
;               subaddress first (no provision for repeated start).
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RCV_DATA:
        INC  SLV_ADDR                   ;Set for READ of slave.
        ACALL   MASTER_CONTROLLER       ;Acquire bus and send slave address.
        JB      NO_ACK,RDEX             ;Check for slave not responding.

RDLoop:
        ACALL   RECV_BYTE               ;Recieve next data byte.
        MOV     @R0,A                   ;Save data byte in buffer.
        INC     R0                      ;Advance buffer pointer.
        DJNZ    BYTE_CNT,RDLoop         ;Repeat untill all bytes received.

RDEX:
        ACALL   SEND_STOP               ;Done, send an I2C stop.
        RET                             ;Return from RCV_DATA Subroutine.


;~~~~~~~~~~~~~~~~~~~~~~~ INTEL CORPORATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ~~~~~~~~~~~~~~~~~~~~~~~~~ I2C MACROS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       These macros are to be used in conjunction with the I2CDEMO.ASM
;       ASM51 program that implements the I2C Master Controller functionality.
;
;
;       Written By:     Sabrina Quarles
;                       Intel Corporation
;                       EMD 8-Bit Applications Engineering      Rev. 1.0
;
;       Date:           December 1, 1992
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

272319–6

```
%*DEFINE(Delay_2_Cycles)(
    NOP
    NOP
    )

%*DEFINE(Delay_3_Cycles)(
    NOP
    NOP
    NOP
    )

%*DEFINE(Delay_4_Cycles)(
    NOP
    NOP
    NOP
    NOP
    )

%*DEFINE(Delay_5_Cycles)(
    NOP
    NOP
    NOP
    NOP
    NOP
    )

%*DEFINE(Delay_6_Cycles)(
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    )

%*DEFINE(Delay_7_Cycles)(
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    )
```

272319–7

```
%*DEFINE(Delay_8_Cycles)(
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   )


%*DEFINE(Release_SCL_High)(
   SETB    SCL_Pin
   JNB     SCL_Pin, $
   )
```

272319–24

APPENDIX B

```
$TITLE(INTEL_I2C_SOFTWARE EMULATION_MASTER CONTROLLER)
$INCLUDE(A:MACRO.PDF)
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       INTEL MCS-51 MASTER  CONTROLLER MODULE
;
;
;       This ASM51 program demonstrates I2C Bus communication between
;       the Intel MCS-51 product line and I2C compatible ICs located on the Philips
;       OM1016 I2C Evaluation Board.
;
;       This program writes the numeric data that represents the following display "_I2C"
;       to an I2C compatible IC (PCF8570 RAM), reads the values back into a buffer and
;       transmits this buffer out to the Philips I2C SAA1064 LED driver to display the sequence.
;
;       Note:  ASM51 macro file MACRO.PDF is referenced for use with this program.
;
;       Written By:     Sabrina Quarles
;                       Intel Corporation
;                       EMD 8-Bit Applications Engineering       Rev 1.0
;       Date:           December 21, 1992
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;     DEFINITIONS
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


;~~~~~~~~~~ I2C Philips Address of compatible devices on I2C Eval Board ~~~

I2C_RAM         EQU    0AEh                    ;Slave address for PCF8570 RAM chip.
I2C_IO          EQU    4Eh                     ;Slave address for PCF8574 I/O expandor.
I2C_LED         EQU    76h                     ;Slave address for SAA1064 LED driver.


;~~~~~~~~ RAM DATA STORAGE BUFFERS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

BIT_CNT         DATA   8h                      ;Bit counter for I2C routines.
BYTE_CNT        DATA   9h                      ;Byte counter for I2C routines.
SLV_ADDR        DATA   0Ah                     ;Slave address for I2C routines.

XMT_DAT         DATA   0Ch                     ;I2C transmit buffer, 12 bytes max.
RCV_DAT         DATA   18h                     ;I2C receive buffer, 8 bytes max.
ALT_XMT         DATA   20h                     ;Alternate I2C transmit buffer, 8 bytes max.

FLAGS           DATA   28h                     ;Location for bit flags.
NO_ACK          BIT    FLAGS.0                 ;I2C no acknowledge flag.
BUS_FAULT       BIT    FLAGS.1                 ;I2C bus fault flag.
I2C_BUSY        BIT    FLAGS.2                 ;I2C busy flag.
```

272319–8

```
;~~~~~~~~~ I2C DECLATIONS ON PORT 0 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SINK        BIT   P0.0             ;Sink pin for oscope triggering.
SCL_PIN     BIT   P0.6             ;I2C serial clock line.
SDA_PIN     BIT   P0.7             ;I2C serial data line.


;~~~~~~~~~~~~~~~ RESET ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

        ORG    0
        AJMP   I2C_RESET

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       SUBROUTINES
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

       ORG    30h

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       SEND STOP Subroutine
;
;       This program sends an I2C STOP condition to release the bus.
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SEND_STOP:
        CLR    SDA_PIN             ;Get SDA ready for stop.
        %RELEASE_SCL_HIGH          ;Set clock for stop.
        %DELAY_3_CYCLES            ;Delay.
        SETB   SDA_PIN             ;Send I2C STOP.
                                   ;Delay satisfied via software.
        CLR    I2C_BUSY            ;Clear I2C busy status.
        RET                        ;Bus should now be released.



;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       SEND_MSG Subroutine
;
;       This subroutine sends a message across the I2C bus using the
;       information stored in the XMT_DAT Buffer in the following format:
;
;       Buffer @R0 = SlvAddr, # of Bytes to be Transferred, Data Bytes
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

272319–9

```
SEND_MSG:
        MOV SLV_ADDR, @R0                ;Initializes Slave Address.
        INC R0                           ;Next address.
        MOV BYTE_CNT,@R0                 ;Initializes BYTE_CNT.
        INC R0                           ;Next address.
        ACALL SEND_DATA                  ;Send Data.
        RET                              ;Return from Subroutine.


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;   MASTER CONTROLLER Subroutine
;
;   This subroutine sends an I2C START condition and Slave Address to
;   begin I2C communications.
;
;     SDA = Receive/Transmit Data
;     SCL = Generate/Control Clock Line
;
;
;   SLV_ADDR = Slave Address
;
;   Verification
;     Issues before MASTER TRANSMIT
;        * No Bus Fault = Bus Not Busy = SCL & SDA HIGH
;
;
;      Issues during MASTER TRANSMIT
;         *  ACK Received after every Byte Transmission
;
;
;     SUBROUTINES Used
;        SEND_BYTE
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

MASTER_CONTROLLER:

        SETB   I2C_BUSY                  ;Indicate that I2C frame is in progress.
        CLR    NO_ACK                    ;Clear error status flags.
        CLR    BUS_FAULT
        JNB    SCL_PIN, FAULT            ;Check for bus clear.
        JNB    SDA_PIN, FAULT
        CLR    SDA_PIN                   ;Begin I2C start.
        %DELAY_3_CYCLES                  ;Delay.
        CLR    SCL_PIN                   ;Complete I2C START.
        %DELAY_3_CYCLES                  ;Delay.
        MOV    A, SLV_ADDR               ;Get slave address.
        ACALL  SEND_BYTE                 ;Send slave address.
        RET

FAULT:
        SETB   BUS_FAULT                 ;Set fault status.
        RET                              ; and return.
```

272319–10

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;   MASTER TRANSMIT ~ SEND_BYTE Subroutine
;
;       This subroutine sends 1 byte of information located in the ACCumulator
;          ACC = Byte to be Transmitted
;
;    Verification Issues
;       * ACK Received after transmission of Byte
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;

SEND_BYTE:
        MOV    BIT_CNT, #8                ;Set bit count value.

SB_LOOP:
        RLC    A                          ;Send one data bit.
        MOV    SDA_PIN, C                 ;Put data bit on pin.
        %RELEASE_SCL_HIGH                 ;Drive SCL HIGH.
        %DELAY_3_CYCLES                   ;Delay.

        CLR    SCL_PIN                    ;Clear SCL.
        %DELAY_3_CYCLES                   ;Delay.
        DJNZ   BIT_CNT,SB_LOOP            ;Repeat until all bits sent.

        SETB   SDA_PIN                    ;Release data line for acknowledge.
        %RELEASE_SCL_HIGH                 ;Send clock for acknowledge.
        %DELAY_4_CYCLES                   ;Delay.
        JNB    SDA_PIN, SB_EX             ;Check for valid acknowledge bit.
        SETB   NO_ACK                     ;Set status for no acknowledge.

SB_EX:
        CLR    SCL_PIN                    ;Finish acknowledge bit.
        %DELAY_3_CYCLES                   ;Delay.
        RET                               ;Return.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;
;  MASTER TRANSMIT ~ SEND DATA Subroutine
;
;  This subroutine transmits multiple data bytes over the SDA line.
;  The following locations must be initialized before the transmission.
;
;     BYTE_CNTR = # of bytes to be transmitted
;     SLV_ADDR  = Slave Address
;     @R0       = Data to be Transmitted
;                 ~ includes any additional subaddresses, control, etc
;                   specific to certain devices
;
;     SUBROUTINES Used
;        MASTER_XMIT
;           SEND_BYTE
;        SEND_BYTE
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
```

272319–11

```
SEND_DATA:
        ACALL   MASTER_CONTROLLER           ;Acquire bus and send slave address.
        JB      NO_ACK,SDEX                 ;Check for slave not responding.
SD_LOOP:
        MOV     A, @R0                      ;Get data byte from buffer.
        ACALL   SEND_BYTE                   ;Send next data byte.
        INC     R0                          ;Advance buffer pointer.
        JB      NO_ACK,SDEX                 ;Check for slave not responding.
        DJNZ    BYTE_CNT, SD_LOOP           ;All bytes sent?

SDEX:
        ACALL   SEND_STOP                   ;Done, send an I2C stop.
        RET                                 ;Return.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       TRANSFER Subroutine
;
;       This subroutine copies data from the EPROM referenced by DPTR into a
;       Buffer referenced by R1.
;
;       DPTR = String stored into EPROM
;       R1 = Buffer in which data shall be stored
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TRANSFER:
        CLR A                               ;Clears ACC.

        MOVC A, @A+DPTR                     ;Moves contents of DPTR into A.
        MOV @R1, A                          ;Copies A into Buffer.
        INC R1                              ;Next address.
        INC DPTR                            ;Next location.
        CLR A                               ;Clears ACC.

        MOVC A, @A+DPTR                     ;Moves contents of DPTR into A.
        MOV @R1, A                          ;Copies A into Buffer.
        MOV R0, A                           ;Copies A into R0 (# of bytes).
        INC R1                              ;Next address.
        INC DPTR                            ;Next location.
        CLR A                               ;Clears A.

NEXT:
        MOVC A, @A+DPTR                     ;Moves contents of DPTR into A.
        DEC R0                              ;Decrease # of remaining bytes.
        MOV @R1, A                          ;Copies A  into Buffer.
        INC R1                              ;Next address.
        INC DPTR                            ;Next location.
        CLR A                               ;Clears A.
        CJNE R0, #0, NEXT                   ;Compare # of bytes remaining.
        RET                                 ;If all bytes copied, return.
```

272319–12

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;        RECV_MSG Subroutine
;
;        This subroutine receives a message from the I2C bus using SLV_ADDR
;        and BYTE_CNT as indicators as to what Slave will be sending info and
;        how many bytes to expect to receive, and places the data into the
;        RCV_DAT buffer.  The RCV_DAT Buffer is configured to receive a max. 8 bytes.
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RECV_MSG:
        MOV SLV_ADDR, @R1               ;Moves SLV_ADDR from Buffer R0 points to.
        INC R1                          ;Next buffer location.
        MOV BYTE_CNT, @R1               ;Moves BYTE_CNT value into memory location.
        ACALL RCV_DATA                  ;Calls RCV_DATA Subroutine.
        RET                             ;Returns from Receive Msg subroutine.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;
;        MASTER RECEIVE ~ RECEIVE BYTE Subroutine
;
;        This subroutine receives a byte from an addressed I2C slave
;        device and places into the ACC register.
;
;        ACC = Data Byte Received
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RECV_BYTE:
        MOV    BIT_CNT,#8               ;Set bit count.
RB_LOOP:
        %RELEASE_SCL_HIGH               ;Read one data bit.
        %DELAY_3_CYCLES                 ;Delay.
        MOV    C, SDA_PIN               ;Get data bit from pin.
        RLC    A                        ;Rotate bit into result byte.
        CLR    SCL_PIN                  ;Clear SCL pin.
        %DELAY_3_CYCLES                 ;Delay.
        DJNZ   BIT_CNT, RB_LOOP         ;Repeat until all bits received.

        PUSH   ACC                      ;Save accumulator.
        MOV    A, BYTE_CNT              ;Copies byte count into A.
        CJNE   A, #1, RB_ACK            ;Check for last byte of frame.
        SETB   SDA_PIN                  ;Send no acknowledge on last byte.
        SJMP   RB_ACLK                  ;No ACK on last byte; jump to RB_ACLK.
RB_ACK:
        CLR    SDA_PIN                  ;Send acknowledge bit.
RB_ACLK:
        %RELEASE_SCL_HIGH               ;Send acknowledge clock.
        POP    ACC                      ;Restore accumulator.
        %DELAY_3_CYCLES                 ;Delay.
        CLR    SCL_PIN                  ;Clear SCL pin.
        SETB   SDA_PIN                  ;Clear acknowledge bit.
        %DELAY_4_CYCLES                 ;Delay.
        RET                             ;Return from RECV_BYTE.
```

272319–13

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;       MASTER RECEIVE ~ RECEIVE DATA BYTES Subroutine
;
;       This subroutine receives multiple data bytes from an addressed
;       I2C slave device into the buffer pointed to by R0.
;
;       BYTE_CNT = # of bytes to be received
;       SLV_ADDR  = Slave addres
;
;       @R0 = location of received data
;
;       SUBROUTINES Used
;               MASTER_XMIT
;               RCV_BYTE
;
;       Note:  To receive with a subaddress, use SEND_DATA to set the
;               subaddress first (no provision for repeated start).
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RCV_DATA:
        INC  SLV_ADDR                   ;Set for READ of slave.
        ACALL   MASTER_CONTROLLER       ;Acquire bus and send slave address.
        JB      NO_ACK,RDEX             ;Check for slave not responding.

RDLoop:
        ACALL   RECV_BYTE               ;Recieve next data byte.
        MOV     @R0,A                   ;Save data byte in buffer.
        INC     R0                      ;Advance buffer pointer.
        DJNZ    BYTE_CNT,RDLoop         ;Repeat untill all bytes received.

RDEX:
        ACALL   SEND_STOP               ;Done, send an I2C stop.
        RET                             ;Return from RCV_DATA Subroutine.

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;                       Main Program
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


I2C_RESET:

        MOV     SP,#2Fh                 ;Set stack to start at 30h.

        MOV DPTR, #RAM_LED              ;Points to RAM_LED string.
        MOV R1, #XMT_DAT                ;Points to the XMT_DAT Buffer.
        ACALL TRANSFER                  ;Transfers RAM_LED into XMT_DAT.

        MOV DPTR, #RAM_SLC              ;Points to RAM_SLC string to select. RAM.
        MOV R1, #ALT_XMT                ;Buffer to transfer string to.
        ACALL TRANSFER                  ;Transfer RAM_SLC into ALT_XMT.
```

272319–14

```
TEST_LOOP:

        CLR     SINK                        ;Trigger point for oscope.
        SETB    SINK

        MOV R0, #XMT_DAT                    ;Points to XMT_DAT Buffer.
        ACALL SEND_MSG                      ;Calls SEND_MSG Subroutine.
                                            ;Writes Data to I2C RAM.
                                            ;(1 Subaddr + 8 data bytes).

        MOV R0, #ALT_XMT                    ;Points to ALTXMT Buffer.
        ACALL SEND_MSG                      ;Calls SEND_MSG Subroutine.
                                            ;Writes Subaddress to Select RAM

        MOV R0, #RCV_DAT                    ;Points to RECEIVE Buffer.
        MOV R1, #XMT_DAT                    ;Points to XMTDAT Buffer.
        ACALL RECV_MSG                      ;Calls RECV_MSG Subroutine.
                                            ;Receives data from I2C RAM into
                                            ;Intel MCS-51 Device.

        MOV R0, #RCV_DAT                    ;Points to RECEIVE Buffer.
        ACALL SEND_MSG                      ;Calls SEND_MSG Subroutine.
                                            ;Transfers RCV_DAT Buffer to LED.
                                            ;(info encoded into string).

        AJMP    TEST_LOOP                   ;Repeat operation for oscope monitoring.

;~~~~~~~~~~~~~~~~~~~~~ I2C STRINGS ~~~~~~~~~~~~~~~~~~~~~~~~~~~

RAM_SLC:    DB      I2C_RAM, 1, 0
RAM_LED:    DB      I2C_RAM, 9, 0, I2C_LED, 6, 0, 37H, 0H, 48H, 3EH, 35H
        END
```

272319–15

```
$TITLE(I2C_MACROS_FOR_THE_80C51)
;~~~~~~~~~~~~~~~~~~~~~~ INTEL CORPORATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ~~~~~~~~~~~~~~~~~~~~~~~~ I2C MACROS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;
;        These macros are to be used in conjunction with the I2CDEMO.ASM
;        ASM51 program that implements the I2C Master Controller functionality.
;
;
;        Written By:      Sabrina Quarles
;                         Intel Corporation
;                         EMD 8-Bit Applications Engineering        Rev. 1.0
;
;        Date:            December 1, 1992
;
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


%*DEFINE(Delay_2_Cycles)(
   NOP
   NOP
   )

%*DEFINE(Delay_3_Cycles)(
   NOP
   NOP
   NOP
   )

%*DEFINE(Delay_4_Cycles)(
   NOP
   NOP
   NOP
   NOP
   )

%*DEFINE(Delay_5_Cycles)(
   NOP
   NOP
   NOP
   NOP
   NOP
   )

%*DEFINE(Delay_6_Cycles)(
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   )
```

272319–16

```
%*DEFINE(Delay_7_Cycles)(
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   )


%*DEFINE(Delay_8_Cycles)(
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   )


%*DEFINE(Release_SCL_High)(
   SETB    SCL_Pin
   JNB     SCL_Pin, $
   )
```

272319–17