Interfacing the X24165/645 to 8051 Microcontrollers

by Applications Staff

This application note demonstrates how the Xicor X24165/645 family of serial memories can be interfaced to the 8051 microcontroller family when connected as shown in Figure 1. The interface uses the port 1 pins to interface to the serial memories. The 8051 assembly code listing for this application note can be obtained from Xicor's website http://www.xicor.com.

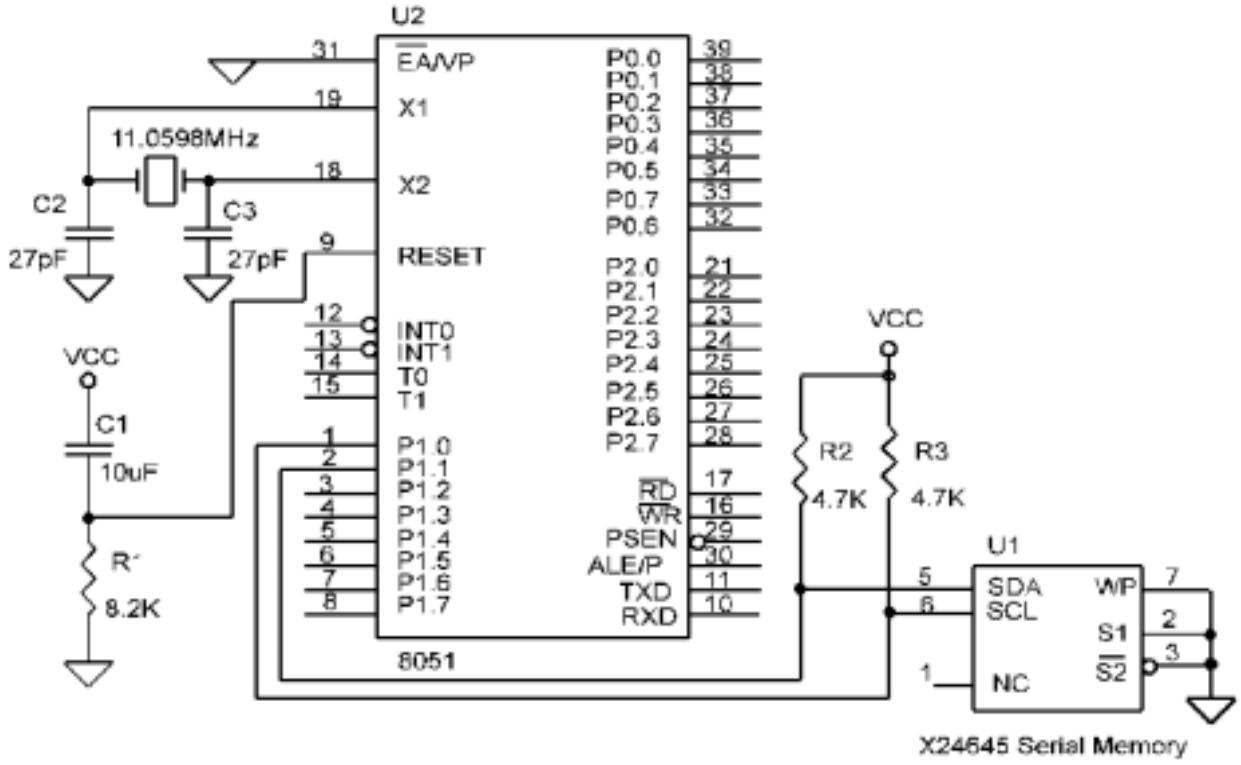

Figure 1. Typical hardware connection for interfacing an X24645 to the 8051 microcontroller.

;

```
;*******************************************************************************
;**
;**  DESCRIPTION:
;**
;**  This file contains general utility routines, written in the 80C51 assembly
;**  language, which are used to interface the 80C51 to Xicor's two-wire
;**  Serial Memory family (X24165/645). The interface between the 80C51 and
;**  X24xxx devices consists of a clock (SCL) and a bidirectional data line (SDA).
;**  The communication interface uses 2 pins from Port 1(P10 = SCL and P11 = SDA).
;**  Other two-wire bus compatible components may reside on this bus provided
;**  that they do not have the same device identifier byte.
;**  The following table lists all the subroutines in this file with a brief
;**  description:
;**
;**          START: Generate the start condition
;**          STOP: Generate the stop condition
;**          RESET: Issues the appropriate commands to force device reset
;**          ProgPage: Transfer page from RAM buffer to Serial Memory
;**          ProgByte: Transfer byte from RAM buffer to Serial Memory
;**          SeqRead: Read multiple bytes, starting from current address pointer
;**          RandomRead: Read a byte from a specific memory location
;**          ACKPoll: Return when the write cycle completes.
;**          OutACK: Process the acknowledge output cycle
;**          GetACK: process the acknowledge from the slave device
;**
;**  The Main loop programs a test string into the Serial Memory. After the
;**  entire string is programmed, the content of the programmed page is read.
;**  The read data is stored in the internal RAM. A utility program can be
;**  written to verify that the buffer content matches the test string.
;**
;*******************************************************************************
;*******************************************************************************
;*                            INTERNAL RAM
;*******************************************************************************
R00         EQU     00H
R01         EQU     01H
RAMBuff     EQU     40H         ; RAM BUFFER ADDRESS
STACK       EQU     60H
;*******************************************************************************
;*                            PROGRAM CONSTANTS
;*******************************************************************************
SDAbit      EQU     01H         ; PORT-1 BITS FUNCTIONING AS BIDIRECTIONAL
SCLbit      EQU     00H         ; SERIAL DATA (SDA) AND SERIAL CLOCK OUTPUT (SCL)
PageNO      EQU     00H         ; PAGE NUMBER OF THE Serial Memory
BPX_0       EQU     03H         ; BPX BIT 0 POSITION IN WPR
BPX_1       EQU     04H         ; BPX BIT 1 POSITION IN WPR
WEL         EQU     01H         ; WEL BIT POSITION IN WPR
RWEL        EQU     02H         ; RWEL BIT POSITION IN WPR
WPEN        EQU     07H         ; WPEN BIT POSITION IN WPR
WELon       EQU     00000010b   ; WEL CONTROL BYTE
RWELon      EQU     00000110b   ; RWEL CONTROL BYTE
MaxDelay    EQU     0FFH        ; NUMBER OF TIMES TO CHECK ACKNOWLEDGE POLLING
ByteData    EQU     058H        ; CHANGES THE x TO AN X IN THE TEST PROGRAM
;*      PROGRAM CONSTANTS FOR X24645
SeqReadSize EQU     16          ; BYTE COUNTS TO SHIFT OUT USING SEQ. READ
```

```
DeviceID    EQU     080H        ; DEVICE SELECT
HiADDRmask  EQU     3FH         ; MASK FOR UPPER ADDRESS BYTE
WPR_ADDR    EQU     1FFFH       ; WPR PHYSICAL ADDRESS LOCATION (BYTE ACCESS)
PageSize    EQU     32          ; BYTES PER PAGE


;*******************************************************************************
;*                            START OF USER CODE                               *
;*******************************************************************************
     ORG 0000H
     ljmp    MAIN                   ; RESET VECTOR

     ORG 0200H
MAIN:
     mov     SP,#STACK              ; LOAD STACK POINTER

; INITIALIZE THE BUFFER BEFORE PROGRAMMING THE CONTENT TO A PAGE

     mov     R0,#RAMBuff            ; R0 = RAM BUFFER ADDRESS
     mov     DPTR,#TestString       ; DPTR = TEST STRING ADDRESS
InitRAM:
     clr     A
     movc    A,@A+DPTR              ; COPY THE TEST STRING TO
     mov     @R0,A                  ; RAM BUFFER
     inc     DPTR
     inc     R0
     jnz     InitRAM
     acall   Reset1                 ; RESET THE INTERFACE STATE MACHINE
     mov     DPTR,#WPR_ADDR         ; READ THE WPR CONTENT AND FIND THE
     acall   RandomRead             ; BLOCKS THAT ARE LOCKED. IF BOTH
                                    ; WPEN BIT AND WP PIN ARE HIGH THEN
     jnb     ACC.WPEN,WPEN_OFF      ; BPx BITS ARE PROTECTED (WRITES ARE
;  ... WARNING ...                  ; PERMITTED WHEN WP IS BROUGHT LOW).
;       MAKE SURE THAT WP PIN IS LOW BEFORE ATTEMPTING TO WRITE NEW VALUE TO THE
;       WPR WHEN WPEN BIT IS SET.
WPEN_OFF:
     jb      ACC.BPX_0,CLR_BPX
     jnb     ACC.BPX_1,NO_BPX       ; SKIP IF BPx BITS ARE CLEAR
CLR_BPX:
     clr     A                      ; CLEAR THE BLOCK LOCK BITS (UNPROTECT
     acall   ProgBP                 ; THE ENTIRE DEVICE)
     acall   ACKPoll                ; WAIT TILL THE OPERATION IS COMPLETED
NO_BPX:
     acall   SetWEL                 ; SET THE WRITE ENABLE BIT
     mov     DPTR,#WPR_ADDR         ; READ THE WPR CONTENT AND
     acall   RandomRead             ; CHECK THAT WEL BIT
     jb     ACC.WEL,WRITES_EN       ; IS SET HIGH, ELSE ITS A FAILURE
     ajmp    $                      ; CHECK THE DEVICE/CONNECTIONS*STOP*
WRITES_EN:
     mov     DPTR,#PageNO           ; DPTR = PAGE NUMBER
     mov     R0,#RAMBuff            ; R0 = RAM BUFFER ADDRESS
     acall   ProgPage               ; TRANSFER BUFFER CONTENT TO THE PAGE
                                    ; IN Serial Memory INDICATED BY DPTR
     acall   ACKPoll                ; WAIT TILL COMPLETION OF PAGE PROG.
     mov     DPTR,#00H              ; ADDRESS 0 IS THE LOCATION FOR THE BYTE TO BE WRITTEN
```

```
                                    ; THIS WILL WRITE OVER THE FIRST BYTE IN THE PAGEPROG
     acall   ProgByte               ; WRITE BYTE TO Serial Memory
     acall   ACKPoll
     acall   ClrWEL                 ; RESET THE WRITE ENABLE BIT
     mov     DPTR,#PageNO           ; DPTR= PAGE NUMBER
     mov     R0,#RAMBuff            ; R0 = RAM BUFFER ADDRESS
     acall   RandomRead             ; SETUP THE ADDRESS POINTER AND READ
     mov     @R0,A                  ; THE FIRST BYTE IN THE PAGE, SAVE
     inc     R0                     ; IT TO THE BUFFER
     mov     A,# HIGH PageNO        ; LOAD THE UPPER BYTE OF ADDRESS
     mov     R1,#1FH                ; BYTE COUNT
     acall   SeqRead                ; READ/STORE THE REMAINING DATA
     ajmp    $                      ; END OF MAIN
;*****************************************************************************
;** Name: SeqRead
;** Description: Read sequentially from the Serial Memory.
;** Function: This subroutine extracts contents of the Serial Memory and stores
;**            them into the specified RAM buffer. The total number of bytes to
;**            read should be provided along with the buffer address. This
;**            routine assumes that the address pointer has already been
;**            initialized using the RandomRead routine.
;**
;** Calls:              Start, SlavAddr, InByte, StopRead
;** Input:              R0 = RAM Buffer Base Address, DPH = High Order Address
;**                     R1 = Number of bytes to read
;** Output:             None
;** Register Usage:     A, R0, R1
;*****************************************************************************
SeqRead:
     acall   Start                  ; START
     setb    c                      ; [C=1] READ OPERATION BIT
     acall   SlavAddr               ; SEND THE SLAVE ADDRESS BYTE
SeqReadLoop:
     acall   InByte                 ; START READING FROM THE CURRENT ADDRESS
     mov     @R0,A                  ; TOTAL NUMBER OF BYTES TO READ OUT OF
     inc     R0                     ; Serial Memory
     djnz    R01,SeqReadNxt
     ajmp    StopRead               ; END OF READ OPERATION
SeqReadNxt:
     acall   OutACK                 ; SEND AN ACKNOWLEDGE TO THE DEVICE
     ajmp    SeqReadLoop
;*****************************************************************************
;** Name: RandomRead
;** Description: Reads content of the Serial Memory at a specific location.
;** Function: This subroutine sends out the command to read the content of a
;**             memory location specified in the DPTR.
;** Calls:              Start, InByte, SlavAddr, OutByte
;** Input:              DPTR = Address of the byte
;** Output:             A = Read value
;** Register Usage:     A
;*****************************************************************************
RandomRead:
     acall   Start                  ; START
     clr     C                      ; [C=0] WRITE OPERATION BIT
     acall   SlavAddr               ; SEND THE SLAVE ADDRESS BYTE
```

```
     mov     A,DPL                  ; LOAD THE LOWER BYTE OF THE PAGE
     acall   OutByte                ; ADDRESS AND SHIFT OUT TO THE DEVICE
     mov     A,DPH
     acall   Start                  ; START
     setb    C                      ; [C=1] READ OPERATION BIT
     acall   SlavAddr               ; SEND THE SLAVE ADDRESS BYTE
     acall   InByte                 ; SHIFT IN A BYTE FROM THE DEVICE
     ajmp    StopRead               ; END OPERATION

;*************************************************************************
;** Name: StopRead
;** Description: Terminate read operation.
;** Function: This subroutine sends out the command to end reading content of a
;**             Serial Memory.
;** Calls:              ClockPulse, Stop
;** Input:              None
;** Output:             None
;** Register Usage:     None

;*************************************************************************
StopRead:
     setb    P1.SDAbit              ; MAKE SURE THAT THE DATA LINE IS HIGH   ...
     acall   ClockPulse
     jmp     Stop                   ; END OPERATION

;*************************************************************************
;** Name: ProgPage
;** Description: Update a page of the Serial Memory.
;** Function: This subroutine transfers the contents of the given buffer to the
;**            Serial Memory. The caller program must supply the page number
;**            of the Serial Memory to update and the base address of the
;**            RAM buffer.
;** Calls:              Start, SlavAddr, OutByte, Stop
;** Input:              R0 = RAM Buffer Base Address, DPTR = Page Number
;** Output:             None
;** Register Usage:     A, R0, R1

;*************************************************************************
ProgPage:
     acall   Start                  ; START
     clr     C                      ; [C=0] WRITE OPERATION BIT
     acall   SlavAddr               ; SEND THE SLAVE ADDRESS BYTE
     mov     A,DPL                  ; LOAD THE LOWER BYTE OF THE PAGE ADDRESS
     anl     A,#0E0H                ; MASK OUT THE UNWANTED LOWER BITS
     acall   OutByte                ; AND SHIFT OUT TO THE DEVICE
     mov     R1,#PageSize           ; TRANSFER CONTENT OF THE RAM BUFFER
ProgPageNxt:
     mov     A,@R0                  ; TO THE Serial MEMORY
     acall   OutByte                ; R0 SHOULD BE POINTING TO THE BUFFER
     mov     A,#0FFH                ; COVER UP YOUR TRACKS AS BUFFER IS
     mov     @R0,A                  ; READ AND WRITTEN TO THE Serial Mem.
     inc     R0                     ; TOTAL NUMBER OF BYTES TRANSFERRED
     djnz    R01,ProgPageNxt        ; TO THE Serial SHOULD NOT EXCEED
                                    ; THE PAGE SIZE
     ajmp     Stop                  ; END OF THE OPERATION
```

```
;*************************************************************************
;** Name: ProgByte
;** Description: Write a byte to Serial Memory.
;** Function: This subroutine transfers the contents of Bytedata to the Serial
;**           Memory. The address written to is contained in the slave address
;**           (DPH) and the lower byte is contained in (DPL).  A two byte
;**           address is required for both page and byte write commands.
;**
;** Calls:              Start, SlavAddr, Outbyte, Stop
;** Input:              Bytedata = byte to be written, DPTR = Address
;** Output:             None
;** Register Usage:     A, DPTR
;*************************************************************************
ProgByte:
     acall   Start                  ; START
     clr     c                      ; [C=0] WRITEOPERATION BIT
     acall   SlavAddr               ; SEND THE SLAVE ADDRESS BYTE
     mov     A,DPL                  ; LOAD THE LOWER BYTE OF THE PAGE ADDRESS
     acall   OutByte                ; SEND OUT THE LOWER BYTE OF THE ADDRESS
     mov     A,#ByteData            ; LOAD THE DATA TO BE SENT IN THE ACC
     acall   OutByte                ; SEND OUT THE DATA TO THE SERIAL MEMORY
     ajmp    Stop                   ; END OF THE OPERATION
;*************************************************************************
;** Name: EnProgWPR
;** Description: Enable updates to Write Protect Register (WPR) of the Serial Memory.
;** Function: This subroutine writes the appropriate sequence to the Serial Memory
;**           to enable updating of the WPR. The ProgWPEN and ProgBP routines
;**           must call this subroutine before writes to the WPR are allowed.
;**           Once this sequence is activated, the only way to exit this mode
;**           is by writing to the WPR or resetting the Serial Memory.
;**
;** Calls:              RandomRead, SetWEL, SetRWEL
;** Input:              None
;** Output:             A = INITIAL WPR VALUE
;** Register Usage:     A, B, DPTR
;*************************************************************************
EnProgWPR:
     mov     DPTR,#WPR_ADDR         ; READ THE WPR CONTENT AND
     acall   RandomRead             ; TEST THE STATUS OF
     mov     B,A
     jnb     B.WEL,ProgWPR_1        ; THE WEL BIT AND SKIP IF ITS SET
     acall   SetWEL                 ; SEND SET WEL COMMAND
ProgWPR_1:
     jnb     B.RWEL,ProgWPR_2       ; CHECK THE RWEL BIT AND SKIP IF ITS SET
     acall   SetRWEL                ; SEND SET RWEL COMMAND
ProgWPR_2:
     mov     A,B
     ret

;*************************************************************************
;** Name: ProgBP
;** Description: Update Block Lock bits in WPR of the Serial Memory.
;** Function: This subroutine writes to the WPR of the Serial Memory and
;**           changes the BP1:0. The caller program must supply the new values
```

```
;**            for the BP1:0 bits. This routine retains the original state of
;**            the WPEN bit.
;**
;** Calls:              AddrWPR, EnProgWPR, OutByte, Stop
;** Input:              A[1:0] = BL[1:0]
;** Output:             None
;** Register Usage:     A, R0
;*******************************************************************************
ProgBP:
     anl     A,#03H                 ; MASK OUT THE UNWANTED BITS
     swap    A                      ; SHIFT THE BPx BITS TO THE
     rr      A                      ; BIT POSITIONS 4:3
     mov     R0,A                   ; SAVE THE BPx NEW VALUES AND
     acall   EnProgWPR              ; ENABLE WRITING TO THE WPR
     anl     A,#9AH                 ; CREATE THE DATA PATTERN BY MASKING
     orl     A,#02H                 ; IN THE DESIRED BIT PATTERN AND
     orl     A,R0                   ; SAVING STATUS OF WPEN BIT
     mov     R0,A                   ; SET THE BPx BITS PER REQUESTED PATTERN
     acall   AddrWPR                ; GENERATE WPR WRITE COMMAND
     mov     A,R0                   ; SHIFT OUT WPR PATTERN
     acall   OutByte                ; TO THE DEVICE
     ajmp    Stop

;*******************************************************************************
;** Name: ProgWPEN
;** Description: Update Write Protect Enable bit in WPR of the Serial Memory.
;** Function: This subroutine writes to the WPR of the Serial Memory and
;**            changes the WPEN bit. The caller program must supply the new
;**            value of the WPEN bit. The state of the BP1:0 bits are preserved.
;**
;** Calls:              AddrWPR, EnProgWPR, OutByte, Stop
;** Input:              C
;** Output:             None
;** Register Usage:     A, R0
;*******************************************************************************
ProgWPEN:
     clr     A                      ; LOAD THE STATUS FLAGS
     rrc     A                      ; MASK OUT THE UNWANTED BITS
     mov     R0,A                   ; SAVE THE WPEN BIT NEW VALUE AND
     acall   EnProgWPR              ; ENABLE WRITING TO THE WPR
     anl     A,#9AH                 ; CREATE THE DATA PATTERN BY MASKING
     orl     A,#02H                 ; IN THE DESIRED BIT PATTERN AND
     orl     A,R0                   ; SAVING STATUS OF WPEN BIT
     mov     R0,A                   ; SET THE WPEN BIT PER AS REQUESTED
     acall   AddrWPR                ; GENERATE WPR WRITE COMMAND
     mov     A,R0                   ; SHIFT OUT WPR PATTERN
     acall   OutByte                ; TO THE DEVICE
     ajmp    Stop

;*******************************************************************************
;** Name: SetWEL
;** Description: Set the Write Enable Latch (WEL) bit in the WPR of the Serial Memory.
;** Function: This subroutine writes to the WPR of the Serial Memory and
;**            sets the WEL bit.
;**
```

```
;** Calls:              AddrWPR, OutByte, Stop
;** Input:              NONE
;** Output:             NONE
;** Register Usage:     A
;*************************************************************************
SetWEL:
     acall   AddrWPR                ; GENERATE WPR WRITE COMMAND
     mov     A,#WELon               ; SHIFT OUT WEL-ON PATTERN
     acall   OutByte                ; TO THE DEVICE
     ajmp    Stop

;*************************************************************************
;** Name: ClrWEL
;** Description: Reset the Write Enable Latch (WEL) bit in the WPR of the Serial Memory.
;** Function: This subroutine writes to the WPR of the Serial Memory and
;**            resets the WEL bit.
;**
;** Calls:              AddrWPR, OutByte, Stop
;** Input:              NONE
;** Output:             NONE
;** Register Usage:     A
;*************************************************************************
ClrWEL:
     acall   AddrWPR                ; GENERATE WPR WRITE COMMAND
     clr     A                      ; SHIFT OUT WEL-OFF PATTERN
     acall   OutByte                ; TO THE DEVICE
     ajmp    Stop


;*************************************************************************
;** Name: SetRWEL
;** Description: Set Register Write Enable Latch bit in the WPR of the Serial Memory.
;** Function: This subroutine writes to the WPR of the Serial Memory and
;**            sets the RWEL bit.
;** Calls:              AddrWPR, OutByte, Stop
;** Input:              NONE
;** Output:             NONE
;** Register Usage:     A
;*************************************************************************
SetRWEL:
     acall   AddrWPR                ; GENERATE WPR WRITE COMMAND
     mov     A,#RWELon              ; SHIFT OUT RWEL-ON PATTERN
    acall   OutByte                ; TO THE DEVICE
    ajmp    Stop

;*************************************************************************
;** Name: AddrWPR
;** Description: Initiate write operation to the WPR of the Serial Memory.
;** Function: This subroutine issues the WPR address and write instruction
;**            to the Serial Memory.
;** Calls:              Start, SlavAddr, OutByte
;** Input:              NONE
;** Output:             NONE
;** Register Usage:     A
;*************************************************************************
```

```
AddrWPR:
     mov     DPTR,#WPR_ADDR
     acall   Start                   ; START [ C = OPERATION BIT ]
     clr     C                       ; [C=0] WRITE OPERATION BIT
     acall   SlavAddr                ; SEND THE SLAVE ADDRESS BYTE
     mov     A,DPL                   ; LOAD THE LOWER BYTE OF ADDRESS
     ajmp    OutByte                 ; AND SHIFT OUT TO THE DEVICE

;*************************************************************************
;** Name: SlavAddr
;** Description: Build the slave address for the Serial Memory.
;** Function: This subroutine concatenates the bit fields for Device ID,
;**            the high address bits and the command bit. The resultant
;**            byte is then transmitted to the Serial Memory.
;** Calls:               OutByte
;** Input:               DPH = Page number high addr.
;**                      C = COMMAND BIT (=0 WRITE, =1 READ)
;** Output:              None
;** Register Usage:      A
;*************************************************************************
SlavAddr:
     mov     A,DPH
     rlc     A                       ; MERGE THE COMMAND BIT
     xrl     A,#DeviceID             ; AND THE DEVICE SELECT BITS
     anl     A,#HiADDRmask           ; WITH THE UPPER BYTE OF
     xrl     A,#DeviceID             ; PAGE ADDRESS
     ajmp    OutByte                 ; SEND THE SLAVE ADDRESS

;*************************************************************************
;** Name: OutByte
;** Description: Sends a byte to the Serial Memory
;** Function: This subroutine shifts out a byte, MSB first, through the
;**             assigned SDA/SCL lines on port 1.
;** Calls:               ClockPulse, GetACK
;** Input:               A = Byte to be sent
;** Return Value:        None
;** Register Usage:      A
;*************************************************************************
OutByte:
     setb    C
OutByteLoop:
     rlc     A                       ; SHIFT OUT THE BYTE, MSB FIRST
     jnc     OutByte0

     setb    P1.SDAbit
     ajmp    OutByte1
OutByte0:
     clr     P1.SDAbit
OutByte1:
     acall   ClockPulse              ; CLOCK THE DATA INTO THE Serial Memory
     cjne    A,#10000000b,OutByteNxt ; MEMORY, SKIP IF NOT DONE
     jmp     GetACK                  ; CHECK FOR AN ACK FROM THE DEVICE
OutByteNxt:
     clr     C                       ; LOOP IF ALL THE BITS HAVE
     ajmp    OutByteLoop             ; NOT BEEN SHIFTED OUT
```

```
;*************************************************************************
;** Name: InByte
;** Description: Shifts in a byte from the Serial Memory
;** Function: This subroutine shifts in a byte, MSB first, through the
;**             assigned SDA/SCL lines on port 1. After the byte is received
;**             an ACK bit is sent to the Serial Memory.
;** Calls:              ClockPulse
;** Input:              None
;** Return Value:       A = Received byte
;** Register Usage:     A
;*************************************************************************
InByte:
     mov     A,#00000001b
     setb    P1.SDAbit               ; SDA LINE FORCED HIGH
InByteNxt:
     acall   ClockPulse              ; CLOCK THE Serial MEMORY AND SHIFT
     rlc     A                       ; INTO ACC. THE LOGIC LEVEL ON THE SDA
     jnc     InByteNxt               ; LINE. THE DEVICE OUTPUTS DATA ON SDA
     ret                             ; MSB FIRST

;*************************************************************************
;** Name: ClockPulse
;** Description: Generate a clock pulse
;** Function: This subroutine forces a high-low transition on the assigned SCL
;**             pin of port 1. It also samples and returns the SDA line state
;**             during high clock period.
;** Calls:              None
;** Input:              None
;** Return Value:       C = SDA line status
;** Register Usage:     None
;*************************************************************************
ClockPulse:
     setb    P1.SCLbit               ; BASED ON A 12MHz SYSTEM CRYSTAL THE
     nop                             ; BUS CYCLE TIME IS 1 MICROSEC.
     nop
     clr     C                       ;
     jnb     P1.SDAbit,ClockPulseLo  ;
     setb    C
ClockPulseLo:
     clr     P1.SCLbit               ; LOWER THE CLOCK LINE
     ret
;*************************************************************************
;** Name: OutACK
;** Description: Send out an ACK bit to the Serial Memory
;** Function: This subroutine clocks an ACK bit to the Serial Memory.
;**             The ACK cycle acknowledges a properly received data by lowering
;**             the SDA line during this period (9th clock cycle of a received
;**             byte).
;** Calls:              ClockPulse
;** Input:              None
;** Return Value:       None
;** Register Usage:     None
;*************************************************************************
OutACK:
```

```
     clr     P1.SDAbit               ; MAKE SURE THAT THE DATA LINE IS LOW   ...
     jmp     ClockPulse              ; CLOCK OUT THE ACK CYCLE

;*************************************************************************
;** Name: GetACK
;** Description: Clock the Serial Memory for ACK cycle
;** Function: This subroutine returns the sampled logic level on the SDA during
;**             high clock cycle. The Serial Memory holds the SDA line HIGH if
;**             it does not receive the correct number of clocks or it's stuck in
;**             a previously initiated write command.
;** Calls:              ClockPulse
;** Input:              None
;** Return Value:       C = ACKnowledge bit
;** Register Usage:     None
;*************************************************************************
GetACK:
     setb    P1.SDAbit               ; FORCE SDA LINE HIGH
     acall   ClockPulse              ; GENERATE ONE CLOCK PULSE
     ret

;*************************************************************************
;** Name: ACKPoll
;** Description: Wait for an ACK from the Serial Memory
;** Function: This subroutine monitors the Serial Memory response
;**             following a dummy write command. The program returns when it
;**             either receives an ACK bit or the maximum number of tries is
;**             reached.
;** Calls:              Start, SlavAddr, Stop
;** Input:              None
;** Return Value:       C = ACKnowledge bit [=0 ACK ,=1 No ACK was received]
;** Register Usage:     A, R1, DPTR
;*************************************************************************
ACKPoll:
     mov     R1,#MaxDelay            ; LOAD MAX NO. OF ACK POLLING CYCLE
     mov     DPTR,#PageNO            ; D = PAGE NUMBER OF THE Serial Memory
ACKPollnxt:
     acall   Start                   ; START THE ACK POLL CYCLE AND
     clr     C                       ; [C=0] WRITE OPERATION BIT
     acall   SlavAddr                ; SEND THE SLAVE ADDRESS. THEN
                                     ; MONITOR THE SDA LINE FOR AN ACK FROM
                                     ; THE Serial Memory. TERMINATE THE
     acall   Stop                    ; OPERATION BY A STOP CONDITION.
     jnc     ACKPollExit             ; EXIT IF THE ACK WAS RECEIVED

     djnz    R1,ACKPollnxt           ; LOOP WHILE THE MAXIMUM NO. OF CYCLES
                                     ; HAVE NOT EXPIRED. ELSE RETURN WITH C=1
ACKPollExit:
     ret
;*************************************************************************
;** Name: Start
;** Description: Send a start command to the Serial Memory
;** Function: This subroutine generates a start condition on the bus. The start
;**             condition is defined as a high-low transition on the SDA
;**             line while the SCL is high. The start is used at the beginning
;**             of all transactions.
```

```
;** Calls:              None
;** Input:              None
;** Return Value:       None
;** Register Usage:     None
;*************************************************************************
Start:
     setb    P1.SDAbit              ; FORCE THE SDA LINE HIGH
     setb    P1.SCLbit              ; FORCE THE SCL CLOCK LINE HIGH
     clr     P1.SDAbit              ; BEFORE TAKING THE SDA LOW
     nop
     nop
     nop
     nop
     clr     P1.SCLbit              ; FORCE THE SCL LOW
     ret

;*************************************************************************
;** Name: Stop
;** Description: Send stop command to the Serial Memory
;** Function: This subroutine generates a stop condition on the bus. The stop
;**             condition is defined as a low-high transition on the SDA
;**             line while the SCL is high. The stop is used to indicate end
;**             of current transaction.
;** Calls:              None
;** Input:              None
;** Return Value:       None
;** Register Usage:     None
;*************************************************************************
Stop:
     clr     P1.SDAbit              ; FORCE THE SDA LOW BEFORE TAKING
     setb    P1.SCLbit              ; THE SCL CLOCK LINE HIGH
     nop
     nop
     nop
     nop
     setb    P1.SDAbit              ; FORCE THE SDA HIGH (IDLE STATE)
     ret

;*************************************************************************
;** Name: Reset
;** Description: Resets the Serial Memory
;** Function: This subroutine is written for the worst case. System interruptions
;**             caused by brownout or soft error conditions that reset the main
;**             CPU may have no effect on the internal Vcc sensor and reset
;**             circuit of the Serial Memory. These are unpredictable and
;**             random events that may leave the Serial Memory interface
;**             logic in an unknown state. Issuing a Stop command may not be
;**             sufficient to reset the Serial Memory.
;** Calls:              Start, Stop
;** Input:              None
;** Return Value:       None
;** Register Usage:     R0
;*************************************************************************
Reset1:
     mov     R0,#0AH                ; APPLY 10 CLOCKS TO THE DEVICE. EACH
```

```
ResetNxt:
     acall   Start                  ; CYCLE CONSISTS OF A START/STOP
     acall   Stop                   ; THIS WILL TERMINATE PENDING WRITE
     djnz    R00,ResetNxt           ; COMMAND AND PROVIDES ENOUGH CLOCKS
     ret                            ; FOR REMAINING BITS OF A READ OPERATION

TestString: DB     'xICOR MAKES IT MEMORABLE!'
          DB     00

;*************************************************************************
;** END OF  X24xxx  Serial Memory INTERFACE SOURCE CODE
;*************************************************************************

          END
```

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.