Interfacing the 400KHz X24128 Serial EEPROM to 8051 Microcontrollers

by Applications Staff

This application note demonstrates how the Xicor X24128 serial EEPROM 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 EEPROM. The 8051 assembly code listing for this application note can be obtained from the Xicor website.

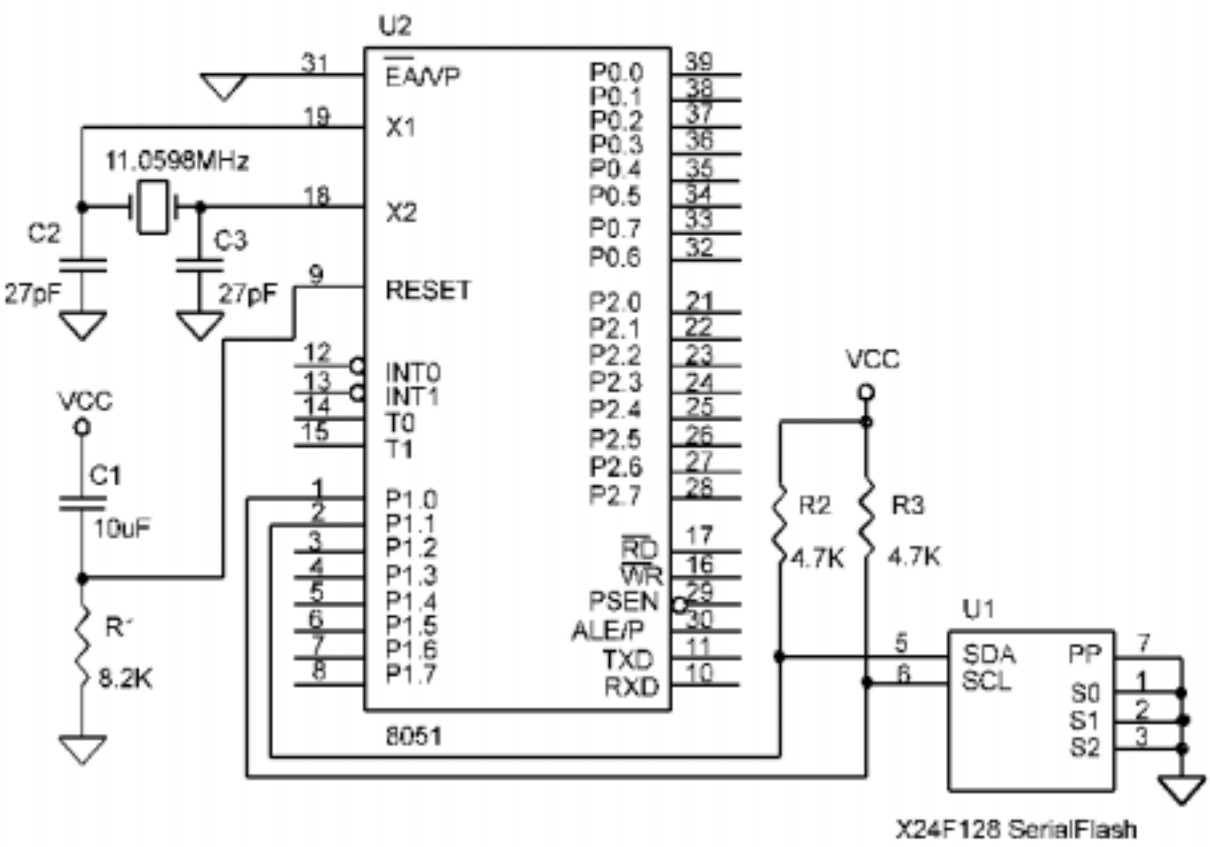

Figure 1. Typical hardware connection for interfacing an X24128 to the 8051 microcontroller.

```
;*******************************************************************************
;**
;**  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
;**  X24129. The interface between the 80C51 and X24129
;**  device 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
;**          RESETDev: Issues the appropriate commands to force device reset
;**          ProgPage: Transfer from RAM buffer to EEPROM page
;**          ProgByte: Transfer from RAM buffer to EEPROM byte
;**          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 EEPROM. 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 EEPROM
BLX_0       EQU     03H         ; BLX BIT 0 POSITION IN WPR
BLX_1       EQU     04H         ; BLX 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
;*   PROGRAM CONSTANTS FOR X24128
SeqReadSize EQU     32          ; BYTE COUNTS TO SHIFT OUT USING SEQ. READ
DeviceID    EQU     50H         ; SLAVE ADDRESS SHIFTED ONE BIT TO THE RIGHT
```

```
HiADDRmask  EQU     3FH         ; MASK FOR UPPER ADDRESS BYTE
WPR_ADDR    EQU     0FFFFH      ; 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   RESETDev                ; 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       ; BLx 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.BLX_0,CLR_BLX
        jnb     ACC.BLX_1,NO_BLX        ; SKIP IF BLx BITS ARE CLEAR
CLR_BLX:
        clr     A                       ; CLEAR THE BLOCK LOCK BITS (UNPROTECT
        acall   ProgBL                  ; THE ENTIRE DEVICE)
        acall   ACKPoll                 ; WAIT TILL THE OPERATION IS COMPLETED
NO_BLX:
        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 EEPROM INDICATED BY DPTR
        acall   ACKPoll                 ; WAIT TILL COMPLETION OF PAGE PROG.
        mov     DPTR,#PageNO            ; DPTR = ADDRESS
        mov     R0,#RAMBuff             ; R0 = RAM BUFFER ADDRESS
```

```
        mov       @R0,#58H                    ; TRANSFER DATA BYTE TO RO IN RAM
           acall   ProgByte                   ; TRANSFER BUFFER CONTENT TO THE BYTE
                                              ; IN EEPROM INDICATED BY DPTR
        acall   ACKPoll                       ; WAIT TILL COMPLETION OF BYTE PROG.
        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 EEPROM.
;** Function: This subroutine extracts contents of the EEPROM 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                        ; EEPROM
        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 EEPROM 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,DPH                   ; LOAD THE UPPER BYTE OF THE PAGE
        acall   OutByte                 ; ADDRESS AND SHIFT OUT TO THE DEVICE
        mov     A,DPL                         ; LOAD THE LOWER BYTE OF THE PAGE
        acall   OutByte                 ; ADDRESS AND SHIFT OUT TO THE DEVICE
        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
;**             EEPROM.
;** 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: ProgByte
;** Description: Update a byte of the EEPROM.
;** Function: This subroutine transfers the contents of the given buffer to the
;**            EEPROM. The caller program must supply the address
;**            of the EEPROM to update and the base address of the
;**            RAM buffer.
;** Calls:               Start, SlavAddr, OutByte, Stop
;** Input:               R0 = RAM Buffer Base Address, DPTR = Byte Address
;** Output:              None
;** Register Usage:      A, R0, R1
;*************************************************************************
ProgByte:
       acall   Start                    ; START
       clr     C                        ; [C=0] WRITE OPERATION BIT
       acall   SlavAddr                 ; SEND THE SLAVE ADDRESS BYTE
       mov     A,DPH                    ; LOAD THE UPPER BYTE OF THE PAGE
       acall   OutByte                  ; AND SHIFT OUT TO THE DEVICE
       mov     A,DPL                    ; LOAD THE LOWER BYTE OF THE PAGE ADDRESS
       acall   OutByte                  ; AND SHIFT OUT TO THE DEVICE
       mov     A,@R0                    ; TO THE EEPROM
       acall   OutByte                  ; R0 SHOULD BE POINTING TO THE BUFFER
       ajmp    Stop                     ; END OF THE OPERATION

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

```
;** Name: ProgPage
;** Description: Update a page of the EEPROM.
;** Function: This subroutine transfers the contents of the given buffer to the
;**            EEPROM. The caller program must supply the pagenumber
;**            of the EEPROM 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,DPH        ; LOAD THE UPPER BYTE OF THE PAGE
      acall    OutByte      ; AND SHIFT OUT TO THE DEVICE
      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 EEPROM
       acall   OutByte       ; R0 SHOULD BE POINTING TO THE BUFFER
               A,  0FFH
       mov     @R0,A
       inc R0                ; TOTAL NUMBER OF BYTES TRANSFERRED
       djnz R01,ProgPageNxt  ; TO THE EEPROM SHOULD NOT EXCEED
       ajmp Stop             ; END OF THE OPERATION



;*************************************************************************
;** Name: EnProgWPR
;** Description: Enable updates to Program Protect Register (WPR) of the EEPROM.
;** Function: This subroutine writes the appropriate sequence to the EEPROM
;**            to enable updating of the WPR. The ProgWPEN and ProgBL 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 EEPROM.
;**
;** 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: ProgBL
;** Description: Update Block Lock bits in WPR of the EEPROM.
;** Function: This subroutine writes to the WPR of the EEPROM and
;**             changes the BL1:0. The caller program must supply the new values
;**             for the BL1:0 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
;*************************************************************************
ProgBL:
        anl     A,#03H                  ; MASK OUT THE UNWANTED BITS
        swap    A                       ; SHIFT THE BLx BITS TO THE
        rr      A                       ; BIT POSITIONS 4:3
        mov     R0,A                    ; SAVE THE BLx 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 BLx 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 Program Protect Enable bit in WPR of the EEPROM.
;** Function: This subroutine writes to the WPR of the EEPROM and
;**             changes the WPEN bit. The caller program must supply the new
;**             value of the WPEN bit. The state of the BL1:0 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 EEPROM.
;** Function: This subroutine writes to the WPR of the EEPROM 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 EEPROM.
;** Function: This subroutine writes to the WPR of the EEPROM 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 EEPROM.
;** Function: This subroutine writes to the WPR of the EEPROM 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 EEPROM.
;** Function: This subroutine issues the WPR address and write instruction
;**            to the EEPROM.
;** 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,DPH                   ; LOAD THE UPPER BYTE OF ADDRESS
        acall   OutByte                 ; AND SHIFT OUT TO THE DEVICE
        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 EEPROM.
;** 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 EEPROM.
;** Calls:              OutByte
;** Input:              DPH = Page number high addr.
;**                     C = COMMAND BIT (=0 WRITE, =1 READ)
;** Output:             None
;** Register Usage:     A
;*************************************************************************
SlavAddr:
      mov      A,#DeviceID      ; SLAVE ADDRESS SHIFTED ONE BIT TO THE RIGHT
      rlc      A                ; MERGE THE COMMAND BIT, SHIFTING TO THE LEFT
      ajmp     OutByte          ; SEND THE SLAVE ADDRESS

;*************************************************************************
;** Name: OutByte
;** Description: Sends a byte to the EEPROM
;** 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 EEPROM
        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 EEPROM
;** 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 EEPROM.
;** 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 EEPROM 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 EEPROM
;** Function: This subroutine clocks an ACK bit to the EEPROM.
;**             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 EEPROM for ACK cycle
;** Function: This subroutine returns the sampled logic level on the SDA during
;**             high clock cycle. The EEPROM 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 EEPROM
;** Function: This subroutine monitors the EEPROM 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 EEPROM
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 EEPROM. 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 EEPROM
;** 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 EEPROM
;** 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: RESETDev
;** Description: Resets the EEPROM
;** 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 EEPROM. These are unpredictable and
;**             random events that may leave the EEPROM interface
;**             logic in an unknown state. Issuing a Stop command may not be
;**             sufficient to reset the EEPROM.
;** Calls:              Start, Stop
;** Input:              None
;** Return Value:       None
;** Register Usage:     R0
;*************************************************************************
RESETDev:
        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  X24129  EEPROM INTERFACE SOURCE CODE
;*************************************************************************

        END
```