

Interfacing the 400KHz X24129 Serial EEPROM to the Motorola 68HC11 Microcontroller

by Applications Staff

This application note demonstrates how the Xicor X24129 Serial EEPROM can be interfaced to the 68HC11 microcontroller family when connected as shown in Figure 1. The interface uses two general

purpose port D pins to interface to the serial memory. The 68HC11 assembly code listing for this application note can be obtained from the Xicor website at <http://www.xicor.com>

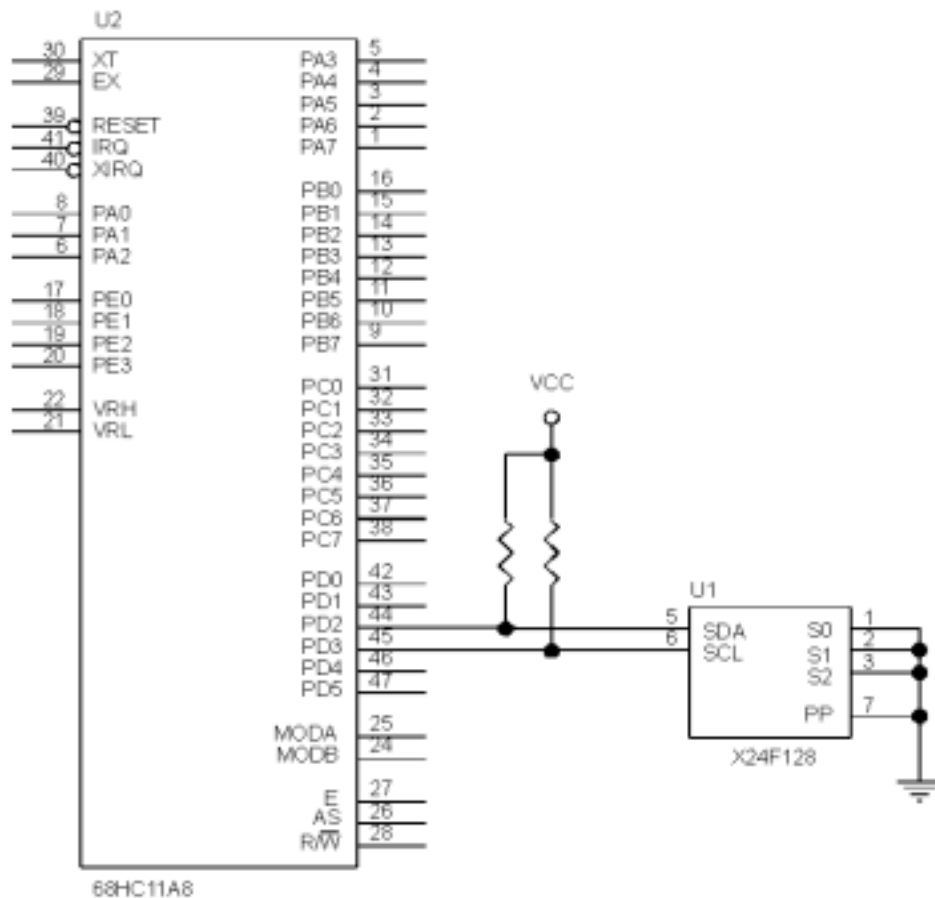


Figure 1. Typical hardware connection for interfacing an X24129EEPROM to the 68HC11 microcontroller.



Application Note

**

** DESCRIPTION:

**

** This file contains general utility routines written in 68HC11 assembly
** language used to interface the M68HC11 to the XICOR Two-wire X24129
** Serial E2PROM. The interface between the 68HC11 and X24129 device
** consists of a clock (SCL) and a bidirectional data line (SDA). The
** communication interface uses 2 pins from Port D (PD3 = SCL and PD2 = SDA).
** Other components may reside on this bus provided that they do not have the
** same device identifier byte as the Serial E2PROM.
** 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 from RAM buffer to serial memory page
** ProgByte: Transfer from RAM buffer to serial memory 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 program loop programs a test string into the Serial E2PROM. After
** 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

RAMBASE	EQU	\$0000	THE INTERNAL RAM BASE ADDRESS(Default)
RAMBuff	EQU	RAMBASE	RAM BUFFER ADDRESS
STACK	EQU	RAMBASE+\$FF	

*

PROGRAM CONSTANTS

DWOM	EQU	\$20	PORT D WOM CONTROL BIT
SDAbit	EQU	\$04	PORT D BITS FUNCTIONING AS BIDIRECTIONAL
SCLbit	EQU	\$08	SERIAL DATA (SDA) AND SERIAL CLOCK (SCL)
PageNO	EQU	\$00	PAGE NUMBER OF THE SERIAL MEMORY
BLX	EQU	\$18	BLX BITS POSITION IN WPR
WEL	EQU	\$02	WEL BIT POSITION IN WPR
RWEL	EQU	\$04	RWEL BIT POSITION IN WPR
WPEN	EQU	\$80	WPEN BIT POSITION IN WPR
WELon	EQU	0000010b	WEL CONTROL BYTE
RWELon	EQU	00000110b	RWEL CONTROL BYTE



Application Note

```
MaxDelay    EQU    $1000    NUMBER OF TIMES TO CHECK ACKNOWLEDGE POLLING
SeqReadSize EQU    16      BYTE COUNTS TO SHIFT OUT USING SEQ READ
```

```
DeviceID    EQU    $50      DEVICE SELECT SHIFTED ONE BIT TO THE RIGHT
HiADDRmask  EQU    $3F      MASK FOR UPPER ADDRESS BYTE
WPR_ADDR    EQU    $FFFF    WPR PHYSICAL ADDRESS LOCATION (BYTE ACCESS)
PageSize    EQU    32      BYTES PER PAGE
```

```
*****
*                INTERNAL REGISTERS AND CONTROL BLOCK
*****
```

```
PORTD      EQU    $08      PORT D DATA REGISTER
DDR        EQU    $09      DATA DIRECTION REGISTER FOR PORT D
SPCR       EQU    $28      SPI CONTROL REGISTER
```

```
*****
*                RESET VECTOR ENTRY POINT
*****
```

```
ORG    $FFFE    RESET VECTOR ADDRESS TO PROGRAM ENTRY
FDB    $E000    JUMP TO BEGINNING OF EXECUTABLE CODE
```

* ASSEMBLER REQUIREMENT- CPU TYPE

P68H11

```
*****
*                START OF USER CODE
*****
```

ORG \$E000

MAIN:

```
lds    #STACK    * LOAD STACK POINTER
```

* INITIALIZE THE BUFFER BEFORE PROGRAMMING THE CONTENT TO A PAGE

```
ldy    #RAMBuff    * IY = RAM BUFFER ADDRESS
ldx    #TestString * IX = TEST STRING ADDRESS
InitRAM:
ldaa   0,x         * COPY THE TEST STRING TO
staa   0,y         * RAM BUFFER
iny
inx
tsta
bne   InitRAM
ldx   #$1000      * SET REGISTER BASE
bset  SPCR,X,#DWOM * CONFIG. PORT-D AS OPEN DRAIN
ldaa  #$0C        * PD2 = SCL
staa  DDRD,X     * PD3 = SDA
ldaa  #$FF       * CONFIGURE PORT D
```



Application Note

```
staa    PORTD,X
jsr     Reset                * RESET THE INTERFACE STATE MACHINE

ldd     #WPR_ADDR            * READ THE WPR CONTENT AND FIND THE
jsr     RandomRead          * BLOCKS THAT ARE LOCKED. IF BOTH
bita    #WPEN                * WPEN BIT AND WP PIN ARE HIGH THEN
bne     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:
bita    #BLX                * SKIP IF THE BLx BITS ARE
beq     NO_Blx              * CLEAR (NO BLOCKS ARE PROTECTED)
clr     clra                * CLEAR THE BLOCK LOCK BITS (UNPROTECT
jsr     ProgBl              * THE ENTIRE DEVICE), WAIT FOR
jsr     ACKPoll            * WRITE OPERATION TO COMPLETE
NO_Blx:
jsr     SetWEL              * SET THE WRITE ENABLE BIT
ldd     #WPR_ADDR            * READ THE WPR CONTENT AND
jsr     RandomRead          * CHECK THAT WEL BIT
bita    #WEL                * IS SET HIGH
bne     WRITES_EN          * ELSE ITS A FAILURE
bra     *                   * CHECK THE DEVICE/CONNECTIONS*STOP*
WRITES_EN:
ldd     #PageNO              * D = PAGE NUMBER OF THE SERIAL MEMORY
ldy     #RAMBuff            * IY = RAM BUFFER ADDRESS
jsr     ProgPage            * TRANSFER BUFFER CONTENT TO THE PAGE
jsr     ACKPoll            * WAIT TILL COMPLETION OF PAGE PROG.
ldy     #RAMBuff            * IY = RAM BUFFER ADDRESS
ldaa    #$58                * LOAD THE DATA TO BE WRITTEN IN THE Y
staa    0,Y                 * REGISTER, ASCII 'X'
ldd     #PageNO              * LOAD THE ADDRESS TO THE ACCUM
jsr     ProgByte            * TRANSFER BUFFER CONTENT TO THE BYTE
jsr     ACKPoll            * WAIT TILL COMPLETION OF BYTE PROG.
jsr     ClrWEL              * RESET THE WRITE ENABLE BIT
ldd     #PageNO              * D = PAGE NUMBER OF THE SERIAL MEMORY
ldy     #RAMBuff            * IY = RAM BUFFER ADDRESS
jsr     RandomRead          * SETUP THE ADDRESS POINTER AND READ
staa    0,Y                 * FIRST BYTE, SAVE IT TO THE BUFFER
iny     *                   * ADJUST THE RAM BUFFER POINTER
ldaa    #.HIGH.PageNO       * LOAD THE UPPER BYTE OF ADDRESS
ldab    #$20                * SPECIFY BYTE COUNT FOR SEQ. READ OP
jsr     SeqRead             * READ/STORE THE REMAINING DATA
bra     *                   * 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 InByte routine.

*** Calls: Start, SlavAddr, InByte, OutACK, StopRead

*** Input: IY = RAM Buffer Base Address, A = High Order Address



Application Note

```
***                               B = Number of bytes to read
*** Output:                       None
*** Register Usage:               A, B, IY
*****
```

SeqRead:

```
    jsr    Start                * START
    sec                    * [C=1] READ OPERATION BIT
    jsr    SlavAddr            * SEND THE SLAVE ADDRESS BYTE
```

SeqReadNxt:

```
    jsr    InByte              * START READING FROM THE CURRENT ADDRESS
    staa   0,Y                 * TOTAL NUMBER OF BYTES TO READ OUT OF
    iny                    * SERIAL MEMORY
    decb                    *
    beq    SeqReadEnd          *
    jsr    OutACK              * SEND AN ACKNOWLEDGE TO THE DEVICE
    bra    SeqReadNxt
```

SeqReadEnd:

```
    jmp    StopRead           * END OF READ OPERATION
```

```
*****
```

```
*** 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 (D) register.
*** Calls:      Start, InByte, SlavAddr, OutByte, StopRead
*** Input:      D = Address of the byte
*** Output:     A = Read value
*** Register Usage: A
*****
```

RandomRead:

```
    psha                    * SAVE ADDRESS TO STACK
    jsr    Start            * START
    clc                    * [C=0] WRITE OPERATION BIT
    jsr    SlavAddr        * SEND THE SLAVE ADDRESS BYTE
    pula                    * RETRIEVE ADDRESS FROM STACK
    jsr    OutByte         * LOAD THE UPPER BYTE OF THE PAGE
    tba                    * LOAD THE LOWER BYTE OF THE PAGE
    jsr    OutByte         * ADDRESS AND SHIFT OUT TO THE DEVICE
    jsr    Start            * START
    sec                    * [C=1] READ OPERATION BIT
    jsr    SlavAddr        * SEND THE SLAVE ADDRESS BYTE
    jsr    InByte          * SHIFT IN A BYTE FROM THE DEVICE
    jmp    StopRead        * END OPERATION
```

```
*****
```

```
*** Name: StopRead
*** Description: Terminate read operation
*** Function: This subroutine is called at the end of a read operation. The
***             routine generates the last ACK clock cycle followed by a stop
***             command. The last ACK bit clock cycle differs from the normal
***             ACK bit in that the SDA line is held high. This action notifies
***             the serial memory that it should suspend operation.
*** Calls:      ClockPulse, Stop
*** Input:      None
*** Output:     None
```



Application Note

```
*** Register Usage:      None
*****
```

```
StopRead:
  bset    PORTD,X,#SDAbit      * MAKE SURE THAT THE DATA LINE IS HIGH
  bset    DDRD,X,#SDAbit      * CHANGE THE PDX DIRECTION TO OUTPUT
  jsr     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:      IY = RAM Buffer Base Address, D(AB) = Page Number
*** Output:     None
*** Register Usage: A,B
*****
```

```
ProgPage:
  psha                    * SAVE ADDRESS TO THE STACK
  jsr    Start            * START
  clc                    * [C=0] WRITE OPERATION BIT
  jsr    SlavAddr        * SEND THE SLAVE ADDRESS BYTE
  pula                    * RETRIEVE ADDRESS FROM STACK
  jsr    OutByte         * LOAD THE UPPER BYTE OF THE PAGE ADDRESS
  tba                    * LOAD THE LOWER BYTE OF THE PAGE ADDRESS
  anda  #$0E0            * MASK OUT THE UNWANTED LOWER BITS
  jsr    OutByte         * AND SHIFT OUT TO THE DEVICE
  ldab  #PageSize       * TRANSFER CONTENT OF THE RAM BUFFER

ProgPageNxt:
  ldaa  0,Y              * TO THE SERIAL MEMORY MEMORY
  jsr    OutByte         * IY SHOULD BE POINTING TO THE BUFFER
  ldaa  #$0FF            * COVER UP YOUR TRACKS AS BUFFER IS
  staa  0,Y              * READ AND STORED TO THE SERIAL MEMORY
  iny                    * TOTAL NUMBER OF BYTES TRANSFERED
  decb                    * TO THE SERIAL MEMORY SHOULD NOT EXCEED
  bne   ProgPageNxt     * THE PAGE SIZE
  jmp   Stop             * END OF THE OPERATION
```

```
*****
```

```
*** Name: ProgByte
*** Description: Update a byte of the serial memory
*** Function: This subroutine transfers a byte located in the Y register to the
***             serial memory.
*** Calls:      Start, SlavAddr, OutByte, Stop
*** Input:      IY = RAM Buffer Base Address,
*** Output:     None
*** Register Usage: A,B
*****
```

```
ProgByte:
```



Application Note

```
psha          * SAVE ADDRESS TO THE STACK
jsr   Start   * START
clc          * [C=0] WRITE OPERATION BIT
jsr   SlavAddr * SEND THE SLAVE ADDRESS BYTE
pula          * RETRIEVE ADDRESS FROM STACK
jsr   OutByte * LOAD THE UPPER BYTE OF THE PAGE ADDRESS
tba          * LOAD THE LOWER BYTE OF THE PAGE ADDRESS
jsr   OutByte * AND SHIFT OUT TO THE DEVICE
ldaa   0,Y    * TO THE SERIAL MEMORY MEMORY
jsr   OutByte * IY SHOULD BE POINTING TO THE BYTE DATA
jmp   Stop    * END OF THE OPERATION
```

```
*****
*** Name: EnProgWPR
*** Description: Enable updates to Write Protect Register (WPR)
*** Function: This subroutine writes the appropriate sequence to the serial memory
***             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 serial memory.
*** Calls:      RandomRead, SetWEL, SetRWEL
*** Input:      None
*** Output:     A = INITIAL WPR VALUE
*** Register Usage: A, B
*****
```

```
EnProgWPR:
  ldd   #WPR_ADDR      * READ THE WPR CONTENT AND
  jsr   RandomRead     * TEST THE STATUS OF
  bita  #WEL           * THE WEL BIT AND
  bne   ProgWPR_1     * SKIP IF ITS SET
  psha          * ALL WRITES TO THE WPR ARE DISALLOWED
  jsr   SetWEL        * WHEN THE WEL IS CLEAR, SEND SET WEL
  pula          * COMMAND
ProgWPR_1:
  bita  #RWEL         * CHECK THE RWEL BIT AND
  bne   ProgWPR_2     * SKIP IF ITS SET
  psha          * WRITING TO BLOCK-LOCK BITS OR WPEN
  jsr   SetRWEL       * BIT REQUIRE THAT RWEL TO BE SET,
  pula          * SEND SET RWEL COMMAND
ProgWPR_2:
  rts
```

```
*****
*** Name: ProgBL
*** 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 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, IY
```



Application Note

```
ProgBL:
  anda    #$03          * MASK OUT THE UNWANTED BITS
  asla
  asla          * SHIFT THE BLx BITS TO THE
  asla          * BIT POSITIONS 4:3
  psha
  jsr      EnProgWPR    * SAVE THE BLx NEW VALUES AND
  anda    #$9A          * ENABLE WRITING TO THE WPR
  oraa    #$02          * CREATE THE DATA PATTERN BY MASKING
  tsy      * IN THE DESIRED BIT PATTERN AND
  oraa    0,y          * SAVING STATUS OF WPEN BIT
  staa    0,y          * SET THE BLx BITS PER REQUESTED PATTERN
  jsr      AddrWPR      * SAVE THE WPR VALUE ONTO THE STACK
  pula    * GENERATE WPR WRITE COMMAND
  jsr      OutByte      * SHIFT OUT WPR PATTERN
  jmp     Stop          * TO THE DEVICE
```

```
*** 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 BL1:0 bits are preserved.
*** Calls:      AddrWPR, EnProgWPR, OutByte, Stop
*** Input:      C
*** Output:     None
*** Register Usage: A, IY
```

```
ProgWPEN:
  clra          * LOAD THE STATUS FLAGS
  rora          * MASK OUT THE UNWANTED BITS
  psha
  jsr      EnProgWPR    * SAVE THE WPEN BIT NEW VALUE AND
  anda    #$9A          * ENABLE WRITING TO THE WPR
  oraa    #$02          * CREATE THE DATA PATTERN BY MASKING
  tsy      * IN THE DESIRED BIT PATTERN AND
  oraa    0,y          * SAVING STATUS OF WPEN BIT
  staa    0,y          * SET THE WPEN BIT PER AS REQUESTED
  jsr      AddrWPR      * SAVE THE WPR VALUE ONTO THE STACK
  pula    * GENERATE WPR WRITE COMMAND
  jsr      OutByte      * SHIFT OUT WPR PATTERN
  jmp     Stop          * TO THE DEVICE
```

```
*** 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
```



Application Note

```
SetWEL:
  jsr    AddrWPR          * GENERATE WPR WRITE COMMAND
  ldaa   #WELon          * SHIFT OUT WEL-ON PATTERN
  jsr    OutByte         * TO THE DEVICE
  jmp    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:
  jsr    AddrWPR          * GENERATE WPR WRITE COMMAND
  clra                   * SHIFT OUT WEL-OFF PATTERN
  jsr    OutByte         * TO THE DEVICE
  jmp    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:
  jsr    AddrWPR          * GENERATE WPR WRITE COMMAND
  ldaa   #RWELon         * SHIFT OUT RWEL-ON PATTERN
  jsr    OutByte         * TO THE DEVICE
  jmp    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,B
*****
```

```
AddrWPR:
  ldd    #WPR_ADDR      * SAVE ADDRESS TO STACK
  psha
```



Application Note

```
jsr Start          * START [ C = OPERATION BIT ]
clc                * [C=0] WRITE OPERATION BIT
jsr   SlavAddr    * SEND THE SLAVE ADDRESS BYTE
pula              * RETRIEVE ADDRESS FROM STACK
jsr   OutByte     * LOAD THE UPPER BYTE OF ADDRESS
tba              * LOAD THE LOWER BYTE OF ADDRESS
jmp   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:    D(AB) = Page number
***           C = COMMAND BIT (=0 WRITE, =1 READ)
*** Output:   None
*** Register Usage: A
```

```
SlavAddr:
  ldaa #$50 * LOAD DEVICEID INTO ACCUM
  rola * MERGE THE COMMAND BIT
  jmp 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 D.
*** Calls:    ClockPulse, GetACK
*** Input:    A = Byte to be sent
*** Return Value: None
*** Register Usage: A
```

```
OutByte:
  bset DDRD,X,#SDAbit * CHANGE THE PDx DIRECTION TO OUTPUT
  sec
OutByteNxt:
  rola * SHIFT OUT THE BYTE, MSB FIRST
  bcc OutByte0
  bset PORTD,X,#SDAbit
  bra OutByte1
OutByte0:
  bclr PORTD,X,#SDAbit
OutByte1:
  jsR ClockPulse * CLOCK THE DATA INTO THE SERIAL MEMORY
  cmpa #10000000b * MEMORY
  clc * LOOP IF ALL THE BITS HAVE
  bne OutByteNxt * NOT BEEN SHIFTED OUT
  jmp GetACK * CHECK FOR AN ACK FROM THE DEVICE
```



Application Note

```
*****
*** 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 D. After the byte is received
***             this subroutine does not send out an ACK bit to the serial memory.
*** Calls:      ClockPulse
*** Input:      None
*** Return Value: A = Received byte
*** Register Usage: A
*****
```

```
InByte:
    ldaa    #00000001b
    bclr    DDRD,X,#SDAbit        * CHANGE THE PDx DIRECTION TO INPUT
InByteNxt:
    jsr     ClockPulse            * CLOCK THE SERIAL MEMORY MEMORY AND SHIFT
    rola    * INTO ACC. THE LOGIC LEVEL ON THE SDA
    bcc     InByteNxt            * LINE. THE DEVICE OUTPUTS DATA ON SDA,
    rts     * MSB FIRST
```

```
*****
*** Name: ClockPulse
*** Description: Generate a clock pulse
*** Function: This subroutine forces a high-low transition on the
***             assigned SCL line on port D. It also samples the SDA
***             line state during high clock period.
*** Calls:      None
*** Input:      None
*** Return Value: C = SDA line status
*** Register Usage: None
*****
```

```
ClockPulse:
    bset    PORTD,X,#SCLbit        * FORCE SCL LINE HIGH. BASED
    nop     * ON AN 8MHz CRYSTAL FREQ. THE SYSTEM
    nop
    clc     * BUS CYCLE TIME IS 0.5 MICROSEC.
    brclr   PORTD,X,#SDAbit,ClockPulseLo *
    sec
ClockPulseLo:
    bclr    PORTD,X,#SCLbit        * LOWER THE CLOCK LINE
    rts
```

```
*****
*** Name: OutACK
*** Description: Send out an ACK bit to the serial memory
*** Function: This subroutine changes the direction of the SDA pin on port D
***             and then 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). The direction of the SDA pin is programmed as input
***             prior to returning to the caller.
*** Calls:      ClockPulse
*** Input:      None
```



Application Note

```
*** Return Value:      None
*** Register Usage:    None
*****
```

OutACK:

```
bclr    PORTD,X,#SDAbit      * MAKE SURE THAT THE DATA LINE IS LOW
bset    DDRD,X,#SDAbit      * CHANGE THE PDx DIRECTION TO OUTPUT
jmp     ClockPulse          *
```

```
*****
```

```
*** Name: GetACK
*** Description: Clock the serial memory for an ACK cycle
*** Function: This subroutine changes the direction of the SDA pin on port D
***             and then clocks the serial memory. It returns the sampled
***             logic level on the SDA during high clock cycle. The serial memory
***             acknowledges a properly received command/data by lowering the
***             SDA line during this period (9th clock cycle of a transmitted
***             byte). If the SDA state is HIGH, it signifies that either it
***             did not receive the correct number of clocks or it's stuck in
***             previously initiated write command,
*** Calls:      ClockPulse
*** Input:      None
*** Return Value: C = ACKnowledge bit
*** Register Usage: None
*****
```

GetACK:

```
bclr    DDRD,X,#SDAbit      * CHANGE THE PDx DIRECTION TO INPUT
jsr     ClockPulse          * CLOCK THE SERIAL MEMORY
bset    DDRD,X,#SDAbit      * CHANGE THE PDx DIRECTION TO OUTPUT
rts
```

```
*****
```

```
*** Name: ACKPoll
*** Description: Wait for an ACK from the serial memory
*** Function: This subroutine sends a slave address to the serial memory and
***             monitors the SDA for an ACK signal. It returns if a low
***             logic level is detected on the SDA during high clock cycle of
***             the acknowledge cycle. The serial memory does not respond to any
***             commands with an acknowledge bit while the store operation
***             is in progress. If no ACK is received another slave address is
***             sent to the serial memory. The number of iteration is specified
***             by the MaxDelay constant.
*** Calls:      Start, SlavAddr, Stop
*** Input:      None
*** Return Value: C = ACKnowledge bit [=0 ACK ,=1 No ACK was received]
*** Register Usage: A, B, IY
*****
```

ACKPoll:

```
ldy     #MaxDelay          * LOAD MAX NO. OF ACK POLLING CYCLE
```

ACKPollnxt:

```
jsr     Start              * START THE ACK POLL CYCLE AND
ldd     #PageNO            * D = PAGE NUMBER OF THE SERIAL MEMORY
clc     [C=0] WRITE OPERATION BIT
jsr     SlavAddr           * SEND THE SLAVE ADDRESS. THEN
```



Application Note

```
jsr      Stop                * OPERATION BY A STOP CONDITION.
bcc      ACKPollExit        * EXIT IF THE ACK WAS RECEIVED
dey
bne      ACKPollnxt        * LOOP WHILE THE MAXIMUM NO. OF CYCLES
ACKPollExit:
rts
```

```
*****
*** 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:
bset     PORTD,X,#SDAbit    * FORCE THE SDA LINE HIGH
bset     PORTD,X,#SCLbit    * FORCE THE SCL CLOCK LINE HIGH
bclr     PORTD,X,#SDAbit    * BEFORE TAKING THE SDA LOW
nop
nop
nop
nop
bclr     PORTD,X,#SCLbit    * FORCE THE SCL LOW
rts
```

```
*****
*** 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:
bclr     PORTD,X,#SDAbit    * FORCE THE SDA LOW BEFORE TAKING
bset     PORTD,X,#SCLbit    * THE SCL CLOCK LINE HIGH
nop
nop
nop
nop
bset     PORTD,X,#SDAbit    * FORCE THE SDA HIGH (IDLE STATE)
rts
```

```
*****
*** Name: Reset
```



Application Note

*** 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: B

```
Reset:
    ldab    #$0A          * APPLY 10 CLOCKS TO THE DEVICE. EACH
ResetNxt:
    jsr     Start        * CYCLE CONSISTS OF A START/STOP
    jsr     Stop         * THIS WILL TERMINATE PENDING WRITE
    decb   *             * COMMAND AND PROVIDES ENOUGH CLOCKS
    bne    ResetNxt     * FOR UNSHIFTED BITS OF A READ
    rts                    * OPERATION
```

```
TestString: FCC    'XICOR MAKES IT MEMORABLE!'
             FCB    $00
```

*** END OF X24320/640/128 SERIAL MEMORY INTERTERFACE SOURCE CODE

END