Interface Software for Xicor SPI Serial Memories

by Applications Staff, June 2000

The following code is intended to provide generic C routines that can be adapted for use with a number of Xicor SPI serial EEPROMs. This code was written for and can be used "as-is" with the X25170, X25650, or X25138 serial SPI E^2PROMs. Figure 1 shows the circuit used to test and debug this code from the Centronics parallel printer port on a generic PC.

When using this code with SPI serial E^2PROMs, a single instruction sequence allows for any number of bytes to be read or up to 32 bytes to be written on a page before wrap-around on that page. With SPI memories, any number of bytes can be read in a single sequence as well, however, entire sectors of 32 bytes must be written when writing to these devices. This requires that all routine calls for WRITE_SPI be made in the following manner:

```
WRITE_SPI(32,addr,&array)
```

Otherwise, the code is entirely compatible, except for some changes in nomenclature (e.g. the WEL bit becomes the PEL bit, etc.).

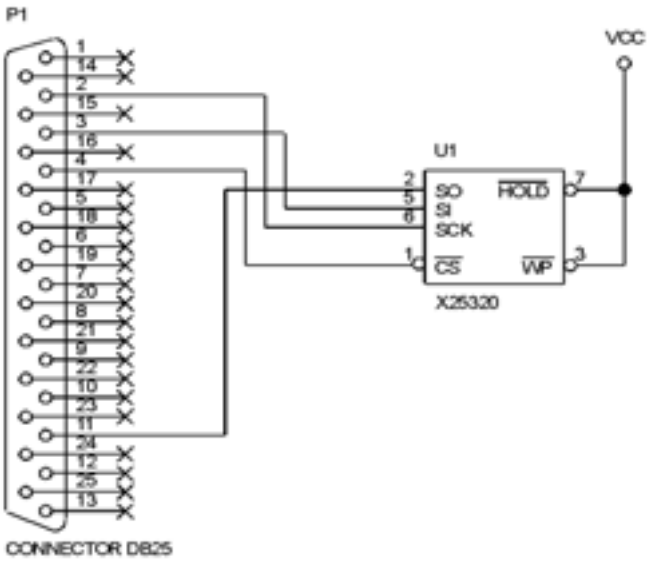

Figure 1. Hardware connection used to verify this code with an X25320 serial EEPROM

```
/*****************************************************************/
Turbo C code for interfacing Xicor second generation SPI serial       */
/* EEPROMs to the Centronics parallel printer port. These devices     */
/* include the X25160, X25320, X25640, X25642, and X25128.            */
/* SCK is connected to D0, SI to D1, SO to BUSY, /CS to D2, and both   */
/* /HOLD and /WP are assumed HIGH.                                    */
/*                                                                    */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>

int data_port = 0x0378;           /* typical output printer port */
int status_port = 0x0379;         /* typical input printer port */

unsigned char mask = 0xFC;        /* default bits X X X X X CS SI SCK */

void SCK_HIGH() {
      mask = mask | 0x01;         /* forces SCK pin HIGH */
      outportb(data_port,mask);
}

void SCK_LOW() {
      mask = mask & 0xFE;         /* forces SCK pin LOW */
      outportb(data_port,mask);
}

void SI_HIGH() {
      mask = mask | 0x02;         /* forces SI pin HIGH */
      outportb(data_port,mask);
}

void SI_LOW() {
      mask = mask & 0xFD;         /* forces SI pin LOW */
      outportb(data_port,mask);
}

unsigned char SO_SAMPLE() {
unsigned char SO_value;
      SO_value = inportb(status_port) & 0x80;  /* get value on SO pin */
                                               /* and isolate */
      SO_value = SO_value >> 7;                /* shift to LSB */
      SCK_HIGH();                              /* provide clock */
      SCK_LOW();
      return(SO_value);
}
void CS_HIGH() {
      mask = mask | 0x04;/* forces CS pin HIGH */
      outportb(data_port,mask);
}

void CS_LOW() {
mask = mask & 0xFB;        /* forces CS pin LOW */
outportb(data_port,mask);
}
```

```
/*****************************************************************/
/*                                                               */
/* Routine transmits a databyte to the SPI memory. The databyte  */
/* is passed to this routine directly when called.               */
/*                                                               */
/*****************************************************************/
void SEND_BYTE(unsigned char byte) {
char count;

      for (count = 0; count <= 7; count++) {  /* loop to pass each bit */
            if ((byte & 0x80) == 0)           /* is the bit LOW? */
                  SI_LOW();
            else
                  SI_HIGH();
            byte = byte << 1;                 /* rotate to get next bit */
            SCK_HIGH();                       /* provide clock */
            SCK_LOW();
      }
}

/*****************************************************************/
/*                                                               */
/* Routine receives a databyte from the SPI memory and passes it */
/* back to the calling routine as an unsigned char.              */
/*                                                               */
/*****************************************************************/
unsigned char GET_BYTE() {
int count;
unsigned char byte,temp;

      byte = 0;                               /* reset byte holder */
      for (count = 0; count <= 7; count++) {  /* loop to get each bit */
            byte = byte << 1;                 /* rotate for next bit */
            temp = SO_SAMPLE();               /* read SO pin */
            if (temp == 0)
                  byte = byte | 0x01;         /* reconstruct current bit */
      }
      return(byte);
}

/*****************************************************************/
/*                                                               */
/* Set the WEL bit in the status register.                       */
/*                                                               */
/*****************************************************************/
void WREN() {
      CS_LOW();
      SEND_BYTE(0x06);                        /* WREN instruction */
      CS_HIGH();
}

/*****************************************************************/
/*                                                               */
/* Reset the WEL bit in the status register.                     */
/*                                                               */
```

```
/*                                                                    */
/**********************************************************************/
void WRDI(){
      CS_LOW();
      SEND_BYTE(0x04);             /* WRDI instruction */
      CS_HIGH();
}

/**********************************************************************/
/*                                                                    */
/* Read the status register.                                          */
/*                                                                    */
/**********************************************************************/
unsigned char RDSR() {
unsigned char byte;
      CS_LOW();
      SEND_BYTE(0x05);            /* RDSR instruction */
      byte = GET_BYTE();          /* retrieve SR databyte */
      CS_HIGH();
      return(byte);
}

/**********************************************************************/
/*                                                                    */
/* Poll the status of the WIP bit to determine the early              */
/* completion of a nonvolatile write cycle.                           */
/*                                                                    */
/**********************************************************************/
void WIP_POLL() {
unsigned char byte;
      byte = 0;                   /* reset byte holder */
      do {
            byte = RDSR();        /* read the SR */
            byte = byte & 0x01;   /* isolate WIP bit */
      } while (byte != 0);        /* repeat until WIP bit is LOW */
}
/**********************************************************************/
/*                                                                    */
/* Write the status register with the value passed to the routine     */
/* in the following format: WPEN|x|x|x|BP1|BP0|x|x                    */
/*                                                                    */
/**********************************************************************/
void WRSR(unsigned char byte) {
      WREN();                     /* set WEL bit */
      byte = byte & 0x8C;         /* force other bits to 0 */
      CS_LOW();
      SEND_BYTE(0x01);            /* WRSR instruction */
      SEND_BYTE(byte);            /* send SR data byte */
      CS_HIGH();
      WIP_POLL();                 /* poll for completion of write cycle */
}

/**********************************************************************/
/*                                                                     */
/* Routine to read multiple consecutive bytes from the SPI memory. */
```

```
/* The specified number of bytes (no_bytes) are read starting from */
/* location (addr) and are stored to the array pointed to by       */
/* (*bytes).                                                       */
/*                                                                 */
/*******************************************************************/
void READ_SPI(int no_bytes,int addr,unsigned char *bytes) {
unsigned char addrhi,addrlo;
int n;
      CS_LOW();
      SEND_BYTE(0x03);                    /* READ instruction */
      addrhi = ((addr & 0x0FF00) >> 8);   /* decompose addr into 2 bytes */
      addrlo = addr & 0x0FF;
      SEND_BYTE(addrhi);                  /* send 2 address bytes */
      SEND_BYTE(addrlo);
      for (n = 0; n < no_bytes; n++) {    /* loop to read bytes */
            bytes[n]=GET_BYTE();          /* read next byte into the array */
      }
      CS_HIGH();
}

/*******************************************************************/
/*                                                                 */
/* Routine to write multiple consecutive bytes to the SPI memory.  */
/* The specified number of bytes (no_bytes) are written starting   */
/* at location (addr) and are taken from the array pointed to by   */
/* (*bytes).                                                       */
/*                                                                 */
/*******************************************************************/
void WRITE_SPI(int no_bytes,int addr,unsigned char *bytes) {
unsigned char addrhi,addrlo,byte;
int n;
      WREN();                             /* set WEL bit */
      CS_LOW();
      SEND_BYTE(0x02);                    /* WRITE instruction */
      addrhi = ((addr & 0x0FF00) >> 8);   /* decompose addr into 2 bytes */
      addrlo = addr & 0x0FF;
      SEND_BYTE(addrhi);                  /* send 2 address bytes */
      SEND_BYTE(addrlo);
      for (n = 0; n < no_bytes; n++) {    /* loop to write bytes */
            byte = bytes[n];              /* retrieve next byte to write */
            SEND_BYTE(byte);              /* write it to the SPI memory */
}
CS_HIGH();
WIP_POLL();                               /* poll for completion of write cycle */
}

/*******************************************************************/
/*                                                                 */
/* Sample MAIN program to demonstrate the use of these routines    */
/*                                                                 */
/*******************************************************************/
main() {
unsigned char data1[]={10,20,30,40,50,60,70,80,90,100};
unsigned char data2[]={15,25,35,45,55,65,75,85,95,105};
unsigned char data3[256];
```

```
unsigned char data4[128];

      outportb(data_port,mask);         /* initialize SI, CS, and SCK on power-up */
      WRITE_SPI(5,513,&data1);          /* write 5 bytes from 513 */
      READ_SPI(1,45,&data4);            /* read 1 byte from 45 */
      WRITE_SPI(1,45,&data1);           /* write 1 byte at 45 */
      WRITE_SPI(8,524,&data2);          /* write 8 bytes at 524 */
      READ_SPI(71,500,&data3);          /* read 71 bytes from 500 */
}
```

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.