SPI Implementation Using Serial Mode 0 on EZ-USB

Introduction

This document demonstrates how to implement SPI (Serial Peripheral Interface) using one of the EZ-USB UARTS in serial mode 0. This demonstration uses the EZ-USB as the master, and transfers data to and from a 25LC320 EEPROM. The sample program demonstrates how to setup the UART. It includes a function to read an SPI byte, a function to write an SPI byte, and functions to read and write data to a 25LC320 EEPROM.

Background

UART Serial mode 0 provides synchronous, half-duplex serial communication. Serial data output appears on the RXD0OUT pin; serial data is received on the RXD0 pin; and the TXD0 pin provides the shift clock for both transmit and receive.

The serial mode 0 baud rate is either CLK24/12 or CLK24/4, depending on the state of the SM2_0 bit. When SM2_0 = 0, the baud rate is CLK24/12; when SM2_0 = 1, the baud rate is CLK24/4.

Data transmission begins when an instruction writes to the SBUF0 SFR. The UART shifts the data, LSB first, at the selected baud rate until the 8-bit value has been shifted out. TI_0 is set after the last bit is shifted out.

Data reception begins when the REN_0 bit is set and the RI_0 bit is cleared in the SCON_0 SFR. The shift clock is activated and the UART shifts data in on each rising edge of the shift clock until 8 bits have been received. After the last bit is shifted in, the RI_0 bit is set and reception stops until the software clears the RI_0 bit.

Initialization

The SCON_0 register initialization needs to address the following items:

- Serial Mode 0—Set SM0_0 = SM1_0 = 0.
- Baud Rate—In this case we are using CLK24/12 because the 25LC320 can not exceed 3 MHz clock rate, so SM2_0 = 0.
- Receive Enable—Set REN_0 = 1 in order to receive data.
- Interrupt Flags—Set TI_0 = RI_0 = 1 so that these flags are zero only when transmitting or receiving. A quick check of their state can determine if the UART is busy. Also, if using interrupts you can software trigger the first interrupt and transmit or receive your first byte.

The final initialization is to turn on the alternate pin functions associated with the serial port. In this example, we are using Serial Port 0. Set the following registers:

- PORTACFG = 0x40.
- PORTCCFG = 0x03.

Program

Data is transmitted one byte at a time by calling the **spiwritebyte** function. Transmitting a byte consists of:

- Clearing the TI_0 flag.
- Writing the byte to the SBUF_0 register.
- Waiting until the TI_0 flag is set indicating that the last bit has been shifted out.

This function also uses a look-up table, **swap**, which reverses the bit order. This is done because the UART transmits LSB first, but the 25LC320 is expecting MSB first.

Data is received one byte at a time by calling the **spireadbyte** function. Receiving a byte consists of:

- Clearing the RI_0 flag.
- Waiting until the RI_0 flag is set indicating that the last bit has been shifted in.
- Reading the SBUF_0 register.

This function also uses the **swap** look-up table to reverse the bit order.

Table 1. SCON_0 Register Init Values

SM0_0	SM1_0	SM2_0	REN_0	N/A	N/A	TI_0	RI_0
b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	1	0	0	1	1

C Program Listing

```
/*********************************************************************

    spi2.c         4-24-00        ott
    Used to test uart mode 0 spi functionality on EZ-USB
    This program will do a constant write and readback loop to an
    25C320 spi EEPROM with EZ-USB as the master
    Pins used:C0 - RxD0 (Data In)
                  C1 - TxD0 (Clk)
                  C2 - Chip select
                  A6 - RxD0OUT (Data Out)

*********************************************************************/

#include <ezusb.h>
#include <ezregs.h>

#define CSHIGH 0x04
#define CSLOW 0xFB
#define READ_CMD 3
#define WRITE_CMD 2
#define WRITE_ENABLE 6
#define WRITE_DISABLE 4
#define READ_STATUS 5
#define WRITE_STATUS 1

////////////////////////////////////// Look up table
code BYTE swap[] = {0,128,64,192,32,160,96,224,16,144,80,208,48,176,112,240,8,136,
72,200,40,168,104,232,24,152,88,216,56,184,120,248,4,132,68,196,36,164,100,
228,20,148,84,212,52,180,116,244,12,140,76,204,44,172,108,236,28,156,92,220,
60,188,124,252,2,130,66,194,34,162,98,226,18,146,82,210,50,178,114,242,10,
138,74,202,42,170,106,234,26,154,90,218,58,186,122,250,6,134,70,198,38,166,
102,230,22,150,86,214,54,182,118,246,14,142,78,206,46,174,110,238,30,158,94,
222,62,190,126,254,1,129,65,193,33,161,97,225,17,145,81,209,49,177,113,241,9,
137,73,201,41,169,105,233,25,153,89,217,57,185,121,249,5,133,69,197,37,165,
101,229,21,149,85,213,53,181,117,245,13,141,77,205,45,173,109,237,29,157,93,
221,61,189,125,253,3,131,67,195,35,163,99,227,19,147,83,211,51,179,115,243,11,
139,75,203,43,171,107,235,27,155,91,219,59,187,123,251,7,135,71,199,39,167,
103,231,23,151,87,215,55,183,119,247,15,143,79,207,47,175,111,239,31,159,95,
223,63,191,127,255};

////////////////////////////////////// Prototypes
void write_25LC320 (int a, BYTE d);
BYTE read_25LC320 (int a);
void enable_25LC320 (void);
BYTE status_25LC320 (void);
void spiwritebyte (BYTE d);
BYTE spireadbyte (void);

main()
{
    BYTE d;
    int a;
    BYTE t,x;

    PORTCCFG = 0x03;                          //Turn on uart pins rxd0, txd0
    PORTACFG = 0x40;                          //Turn on uart pin rxd0out
    OUTC |= CSHIGH;                           //Turn cs high
    OEC = 0x04;                               //Make CS# output
    SCON0 = 0x13;                             //Mode 0, baud 24/12, enable receive
    CKCON &= 0xF8;                            //Set stretch 0
    while(TRUE)
    {
           enable_25LC320();                  //Enable write
```

```
64              write_25LC320 (a,d);                    //Write byte
65              while (status_25LC320() & 1);           //Wait until done
66              t = read_25LC320 (a);                   //Try to read back
67              if (t != d)
68                      x=0;                            //Test for read back, set breakpoint here
69              a++;
70              d++;
71      }
72  }
73  void write_25LC320 (int a, BYTE d)
74  {
75      OUTC &= CSLOW;                                  //Turn cs low
76      spiwritebyte (WRITE_CMD);
77      spiwritebyte (a >> 8);
78      spiwritebyte (a);
79      spiwritebyte (d);
80      OUTC |= CSHIGH;                                 //Turn cs high
81  }
82
83  BYTE read_25LC320 (int a)
84  {
85      BYTE d;
86
87      OUTC &= CSLOW;                                  //Turn cs low
88      spiwritebyte (READ_CMD);
89      spiwritebyte (a >> 8);
90      spiwritebyte (a);
91      d = spireadbyte();
92      OUTC |= CSHIGH;                                 //Turn cs high
93      return (d);
94  }
95
96  void enable_25LC320 (void)
97  {
98      OUTC &= CSLOW;                                  //Turn cs low
99      spiwritebyte (WRITE_ENABLE);
100     OUTC |= CSHIGH;                                 //Turn cs high
101 }
102
103 BYTE status_25LC320 (void)
104 {
105     BYTE d;
106
107     OUTC &= CSLOW;                                  //Turn cs low
108     spiwritebyte (READ_STATUS);
109     d = spireadbyte();
110     OUTC |= CSHIGH;                                 //Turn cs high
111     return (d);
112 }
113
114 void spiwritebyte (BYTE d)
115 {
116     TI = FALSE;                                     //Clear flag
117     SBUF0 = swap[d];                                //Write byte
118     while (!TI);                                    //Wait until done transmitting
119 }
120
121 BYTE spireadbyte (void)
122 {
123     RI = FALSE;                                     //Clear flag
124     while (!RI);                                    //Wait until done receiving
125     return (swap[SBUF0]);                           //Return byte
126 }
```

Performance

This example uses the slower baud rate of CLK24/12, or 2 Mbits/s. This is at the limit of the 25LC320. At this rate, the **write_25LC320** function, which writes four consecutive bytes (command, address HIGH, address LOW, and data byte), takes approximately 37 µs to complete. This equates to a throughput greater than 840 kbits/s. Similar performance is seen with the **read_25LC320** function.

Numerous variations and optimizations can be made, depending on the application. For instance, multiple byte functions can be written to reduce some overhead. Also, interrupts can be used to keep the data flowing for large multiple byte read and writes.

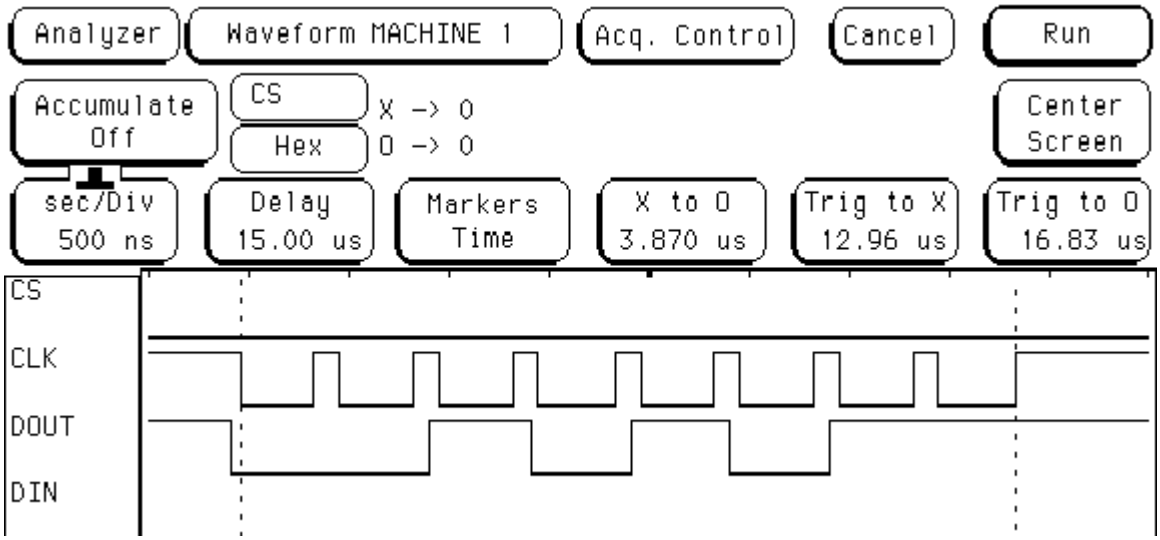

Figure 1. Typical Byte Write Sequence

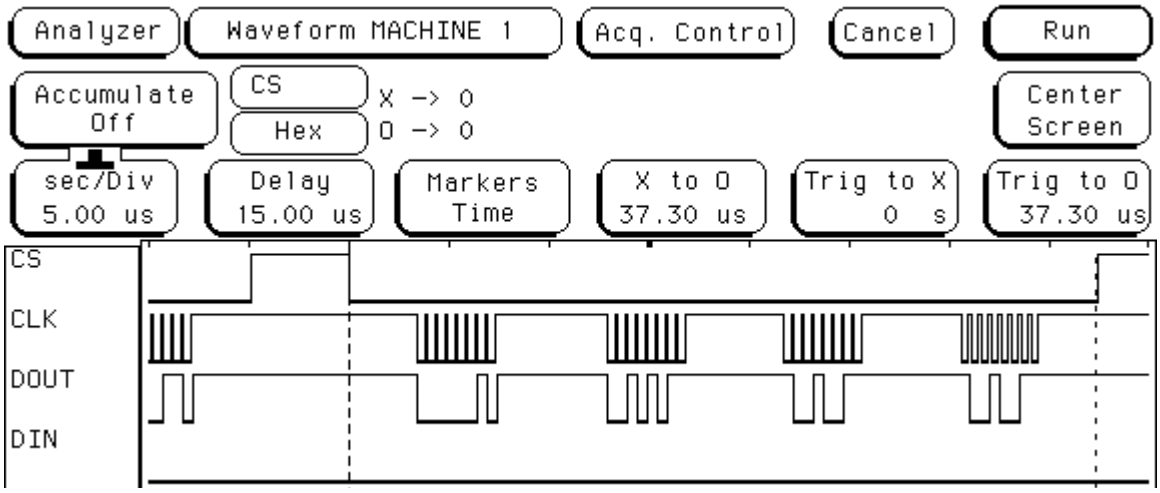

Figure 2. Entire write_25LC320 Sequence

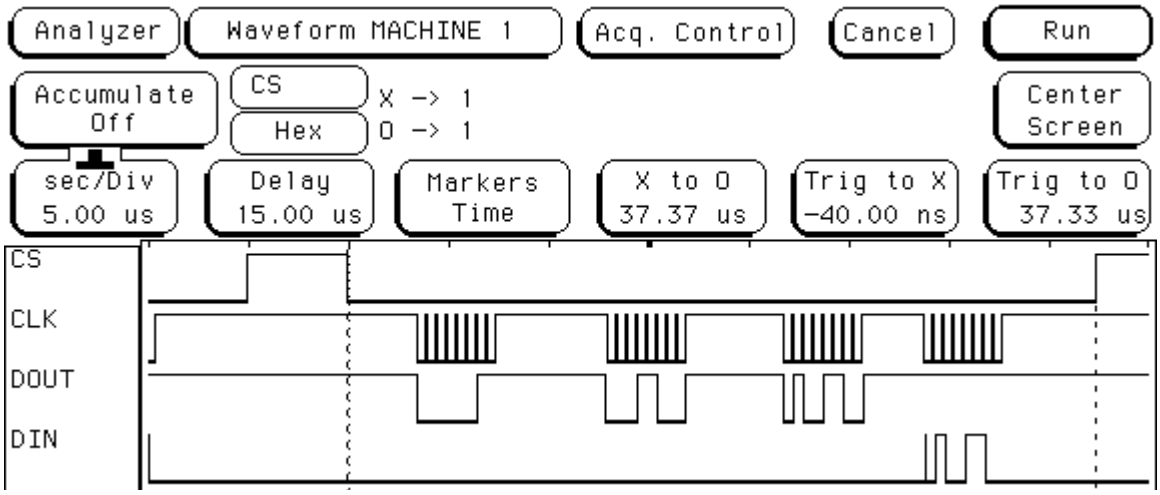

Figure 3. Entire read_25LC320 Sequence

Conclusion

This document demonstrates an easy way to implement SPI communications using the EZ-USB Serial Ports in mode 0. A minimum of code is required. Very respectable performance can be achieved as demonstrated with a 25LC320 EEPROM. Application specific modifications and optimizations can be made to communicate with other SPI peripherals and to increase performance.