1. CPE 325: Laboratory Assignment #9 Communications Part 2 – Synchronous Communication

Objectives: This tutorial will continue covering communication protocols used with the MSP430 and other devices. In the previous lab, asynchronous communications protocols were discussed, and RS-232 communication using the USCI peripheral in UART mode was implemented. In this lab, the SPI synchronous communications protocol is discussed, and it is implemented using the USCI peripheral and the USI peripheral.

- Configuration of the USCI peripheral device for SPI mode
- Configuration of the USI peripheral device for SPI mode
- Implementation of SPI communication between microcontrollers on the TI experimenter board

Note: All previous tutorials are required for successful completion of this lab. Especially, the tutorials introducing the TI experimenter's board and the IAR software development environment.

1.1. Synchronous communications

In the previous lab, asynchronous communication using the RS-232 protocol was discussed as a type of serial communication. Asynchronous communication is most useful when communication must be established between two distinct systems that each have their own clock. Examples of serial, asynchronous communication systems are USB, RS-232, Firewire (IEEE 1394), and Apple's Thunderbolt.

Synchronous communication protocols are best suited for parts of a distinct system when components can share a clock. Typically, these protocols could be used for communicating between memory modules, microcontrollers, sensors, and other board-level components. Today, we will be learning about the SPI communication protocol, the simplest to implement synchronous communications protocol. The I^2C protocol is perhaps more widely implemented, and you can learn more about it in your Davies textbook in pages 534-574.

In the lab today, we will see how we can develop microcontroller programs for the experimenter board that involve SPI communications between the two on-board microcontrollers (using a USCI in SPI mode for the 4618 and using the USI in SPI mode for the 2013) and also implement RS-232 communications as well (using a second USCI peripheral on the 4618).

1.2. Serial Peripheral Interface

In SPI mode, serial data is transmitted and received by multiple devices using a shared clock provided by the master. This is the simplest synchronous communication protocol in general use but faces the problem of not having a fixed standard like I2C. There are several variations of SPI and one must read the data sheet of the device closely and ensure that the details of the protocol are well understood. The Universal Serial Communication Interface (USCI) and Universal Serial Interface (USI) modules supports Serial Peripheral Interface (SPI) serial communication mode in MSP430FG4618 and MSP430F2013 respectively. One device is the

master and the other the slave. The master provides the clock for both devices and a signal to select (enable) the slave, but the path followed by the data is identical in each. In its full form SPI requires four wires (plus ground, which is essential but never counted) and transmits data simultaneously in both directions (full duplex) between two devices. The general nomenclature for the two data connections is “master in, slave out” (MISO) and “master out, slave in” (MOSI). This is admirably clear and makes the functions unambiguous. The two MISO pins should be connected together and likewise the two MOSI pins. Other terms are widely used, such as SDI, SI, or DIN for serial data in and SDO, SO, or DOUT for serial data out. In this case you connect an input on one device to an output on the other. There is similar variety in the names for the clock signal including SCLK (most popular), SPSCK, and SCK. The final signal selects the slave. This is usually active low and labeled SS for slave select, CS for chip select, or CE for chip enable. A slave should drive its output only when SS is active; the output should float at other times in case another slave is selected. In some modes of SPI, the first bit should be placed on the output when SS becomes active to start a new transfer.

1.2.1. USCI Operation – SPI Mode (MSP430FG4618)

SPI can be interfaced through USCI_B0 module present in MSP430FG4618. The following signals are used for SPI data exchange in USCI operation:

- UCB0SIMO Slave in, master out
  Master mode: UCB0SIMO is the data output line.
  Slave mode: UCB0SIMO is the data input line.

- UCB0SOMI Slave out, master in
  Master mode: UCB0SOMI is the data input line.
  Slave mode: UCB0SOMI is the data output line.

- UCB0CLK USCI SPI clock
  Master mode: UCB0CLK is an output.
  Slave mode: UCB0CLK is an input.

- UCB0STE Slave transmit enable. Used in 4-pin mode to allow multiple masters on a single bus. Not used in 3-pin mode.

The USCI is reset by the UCSWRST bit. When set, the UCSWRST bit resets the UCB0RXIE, UCB0TXIE, UCB0RXIFG, UC0E, and UCFE bits and selects the UCB0TXIFG flag. Clearing UCSWRST releases the USCI for operation. The USCI module in SPI mode supports 7- and 8-bit character lengths selected by the UC7BIT bit. In 7-bit data mode, UCB0RXBUF is LSB justified and the MSB is always reset. The UCMSB bit controls the direction of the transfer and selects LSB or MSB first.

USCI Master: The USCI initiates data transfer when data is moved to the transmit data buffer UCB0TXBUF. The UCB0TXBUF data is moved to the TX shift register when the TX shift register is empty, initiating data transfer on UCB0SIMO starting with either the most-significant or least-significant bit depending on the UCMSB setting. Data on UCB0SOMI is shifted into the receive shift register on the opposite clock edge. When the character is received, the receive data is moved from the RX shift register to the received data buffer UCB0RXBUF and the receive interrupt flag, UCB0RXIFG, is set, indicating the RX/TX operation is complete. A set transmit interrupt flag, UCB0TXIFG, indicates that data has moved from UCB0TXBUF to the TX shift register and UCB0TXBUF is ready for new data. It does not indicate RX/TX completion. To receive data into the USCI in master mode, data must be written to UCB0TXBUF because receive and transmit operations operate concurrently.

USCI Slave: UCB0CLK is used as the input for the SPI clock and must be supplied by the external master. The data-transfer rate is determined by this clock and not by the internal bit clock generator. Data written to UCB0TXBUF and moved to the TX shift register before the start of UCB0CLK is transmitted on UCB0SOMI. Data on UCB0SIMO is shifted into the receive shift register on the opposite edge of UCB0CLK and moved to UCB0RXBUF when the set number of bits are received. When data is moved from the RX shift register to UCB0RXBUF, the UCB0RXIFG interrupt flag is set, indicating that data has been received. The overrun error bit, UCOE, is set when the previously received data is not read from UCB0RXBUF before new data is moved to UCB0RXBUF.

UCB0CLK is provided by the master on the SPI bus. When UCMST = 1, the bit clock is provided by the USCI bit clock generator on the UCB0CLK pin. The clock used to generate the bit clock is selected with the UCSSELx bits. When UCMST = 0, the USCI clock is provided on the UCB0CLK pin by the master, the bit clock generator is not used, and the UCSSELx bits are don’t care. The SPI receiver and transmitter operate in parallel and use the same clock source for data transfer. The 16-bit value of UCBRx in the bit rate control registers UCB0xBR1 and UCB0xBR0 is the division factor of the USCI clock source, BRCLK. The maximum bit clock that can be generated in master mode is BRCLK. Modulation is not used in SPI mode.

The USCI has one interrupt vector for transmission and one interrupt vector for reception. The UCB0TXIFG interrupt flag is set by the transmitter to indicate that UCB0TXBUF is ready to accept another character. An interrupt request is generated if UCB0TXIE and GIE are also set. UCB0TXIFG is automatically reset if a character is written to UCB0TXBUF. UCB0TXIFG is set after a PUC or when UCSWRST = 1. UCB0TXIE is reset after a PUC or when UCSWRST = 1. The UCB0RXIFG interrupt flag is set each time a character is received and loaded into UCB0RXBUF. An interrupt request is generated if UCB0RXIE and GIE are also set. UCB0RXIFG and UCB0RXIE are reset by a system reset PUC signal or when UCSWRST = 1. UCB0RXIFG is automatically reset when UCB0RXBUF is read.

1.2.2. USI Operation – SPI Mode (MSP430F2013)

The USI module provides the basic functionality to support synchronous serial communication. In its simplest form, it is an 8- or 16-bit shift register that can be used to output data streams, or when combined with minimal software, can implement serial communication. In addition, the USI includes built-in hardware functionality to ease the implementation of SPI communication. The USI module also includes interrupts to further reduce the necessary software overhead for serial communication and to maintain the ultra-low-power capabilities of the MSP430.

The USI module is a shift register and bit counter that includes logic to support SPI communication. The USI shift register (USISR) is directly accessible by software and contains the data to be transmitted or the data that has been received. The bit counter counts the number of sampled bits and sets the USI interrupt flag USIIFG when the USICNTx value becomes zero, either by decrementing or by directly writing zero to the USICNTx bits. Writing USICNTx with a value > 0 automatically clears USIIFG when USIIFGCC = 0, otherwise USIIFG is not affected. The USICNTx bits stop decrementing when they become 0. They will not underflow to 0FFh. Both the counter and the shift register are driven by the same shift clock. On a rising shift clock edge, USICNTx decrements and USISR samples the next bit input. The latch connected to the shift register's output delays the change of the output to the falling edge of shift clock. It can be made transparent by setting the USIGE bit. This setting will immediately output the MSB or LSB of USISR to the SDO pin, depending on the USILSB bit.

While the USI software reset bit, USISWRST, is set, the flags USIIFG, USISTTIFG, USISTP, and USIAL will be held in their reset state. USISR and USICNTx are not clocked and their contents are not affected. To activate USI port functionality the corresponding USIPEx bits in the USI control register must be set. This will select the USI function for the pin and maintains the PxIN and PxIFG functions for the pin as well. With this feature, the port input levels can be read via the PxIN register by software and the incoming data stream can generate port interrupts on data transitions. This is useful, for example, to generate a port interrupt on a START edge.

The clock source can be selected from the internal clocks ACLK or SMCLK, from an external clock SCLK, as well as from the capture/compare outputs of Timer_A. In addition, it is possible to clock the module by software using the USISWCLK bit when USISSELx = 100. The USIDIVx bits can be used to divide the selected clock by a power of 2 up to 128. The generated clock, USICLK, is stopped when USIIFG = 1 or when the module operates in slave mode. The USICKPL bit is used to select the polarity of USICLK. When USICKPL = 0, the inactive level of USICLK is low. When USICKPL = 1 the inactive level of USICLK is high.

The USI module is configured in SPI mode when USII2C = 0. Control bit USICKPL selects the inactive level of the SPI clock while USICKPH selects the clock edge on which SDO is updated and SDI is sampled. USIPE5, USIPE6, and USIPE7 must be set to enable the SCLK, SDO, and SDI port functions.
USI Master: The USI module is configured as SPI master by setting the master bit USIMST and clearing the I2C bit USII2C. Since the master provides the clock to the slave(s) an appropriate

clock source needs to be selected and SCLK configured as output. When USIPE5 = 1, SCLK is automatically configured as an output. When USIIFG = 0 and USICNTx > 0, clock generation is enabled and the master will begin clocking in/out data using USISR. Received data must be read from the shift register before new data is written into it for transmission. In a typical application, the USI software will read received data from USISR, write new data to be transmitted to USISR, and enable the module for the next transfer by writing the number of bits to be transferred to USICNTx.

USI Slave: The USI module is configured as SPI slave by clearing the USIMST and the USII2C bits. In this mode, when USIPE5 = 1 SCLK is automatically configured as an input and the USI receives the clock externally from the master. If the USI is to transmit data, the shift register must be loaded with the data before the master provides the first clock edge. The output must be enabled by setting USIOE. When USICKPH = 1, the MSB will be visible on SDO immediately after loading the shift register. The SDO pin can be disabled by clearing the USIOE bit. This is useful if the slave is not addressed in an environment with multiple slaves on the bus. Once all bits are received, the data must be read from USISR and new data loaded into USISR before the next clock edge from the master. In a typical application, after receiving data, the USI software will read the USISR register, write new data to USISR to be transmitted, and enable the USI module for the next transfer by writing the number of bits to be transferred to USICNTx.

The 16-bit USISR is made up of two 8-bit registers, USISRL and USISRH. Control bit USI16B selects the number of bits of USISR that are used for data transmit and receive. When USI16B = 0, only the lower 8 bits, USISRL, are used. To transfer < 8 bits, the data must be loaded into USISRL such that unused bits are not shifted out. The data must be MSB- or LSB-aligned depending on USILSB. When USI16B = 1, all 16 bits are used for data handling. When using USISR to access both USISRL and USISRH, the data needs to be properly adjusted when < 16 bits are used.

There is one interrupt vector associated with the USI module, and one interrupt flag, USIIFG, relevant for SPI operation. When USIIE and the GIE bit are set, the interrupt flag will generate an interrupt request.

USIIFG is set when USICNTx becomes zero, either by counting or by directly writing 0 to the USICNTx bits. USIIFG is cleared by writing a value > 0 to the USICNTx bits when USIIFGCC = 0, or directly by software.

The following programs in Figure 1 and Figure 2 illustrate utilization of SPI mode of communication between the microcontrollers MSP430FG4618 and MSP430F2013 both of which are present on the TI experimenter's board. Serial communication setup using UART mode of USCI between MSP430FG4618 and PC enables visualization and confirmation of the data transfer between the two microcontrollers using SPI. The programs in Figure 3 and Figure 4 are to be run on MSP430FG4618 and MSP430F2013 respectively as per the instructions provided in the program header. The MSP430FG4618 uses the USCI while the MSP430F2013 uses the USI. MSP430FG4618 communicates with PC via RS232 module using USCI Serial Communication

peripheral interface. This program takes user prompts the user to input a choice to turn ON or OFF the LED3 located on MSP430F2013. The user choice is communicated to MSP430FG4618 (master) via USCI serial interface and the corresponding action is communicated to MSP430F2013 (slave) via SPI. Based on the user choice, MSP430F2013 will turn ON or OFF the LED3. Open HyperTerminal application on your workstation with the settings as mentioned in the demo programs below. After creating a project for each program in IAR, download and run the program in Figure 3 by connecting the FET debugger to JTAG1 on the board. Stop debugging this project. Disconnect FET debugger from JTAG1 and connect it to JTAG2 on the board. Make sure the device selected is MSP430F2013. Now download and run the program in Figure 4. MSP430FG4618 sends a message to the HyperTerminal and awaits response from the user through keyboard to turn on or off the LED3. LED3 is connected to pin 0 of port 1 (P1.0) on MSP430F2013.The user input is then sent from MSP430FG4618 to MSP430F2013 via SPI. LED 3 will be turned on or off accordingly and the current state of LED is detected by MSP430FG4618 and sent to HyperTerminal via UART.

```
1. /*******************************************************************************
2. //              Lab 5 - Serial Communication and SPI Interface
3. //
4. // Description: Using the MSP-EXP430FG4618 Development Tool establish a

5. //               data exchange between the MSP430FG4618 and MSP430F2013
6. //               devices using the SPI mode. The MSP430FG4618 uses the USCI
7. //               module while the MSP430F2013 uses the USI module.
8. //              MSP430FG4618 communicates with PC via RS232 module using
9. //              USCI Serial Communication peripheral interface. This program
10.//        takes user prompts the user to input a choice to turn ON or OFF
11.//        the LED3 located on MSP430F2013. The user choice is communicated
12.//        to MSP430FG4618 (master) via USCI serial interface and the
13.//        corresponding action is communicated to MSP430F2013(slave) via
14.//        SPI. Based on the user choice, MSP430F2013 will turn ON or OFF
15.//        the LED3. This is the master code that runs on MSP430FG4618.
16.//
17.//                                Slave                   Master
18.//                            MSP430F2013             MSP430FG4618
19.//                         -----------------          -----------------
20.//                        |              XIN|-    /|\|              XIN|-
21.//                        |                 |      | |                 |  32kHz
   xtal
22.//                        |             XOUT|-     --|RST           XOUT|-
23.//                        |                 |        |                 |
24.//                    LED <-|P1.0             |        |                 |
25.//                        |         BF /P1.4|------->|P3.0/BF          |
26.//                        |         SDI/P1.7|<-------|P3.1/UCB0SIMO    |
27.//                        |         SDO/P1.6|------->|P3.2/UCB0SOMI    |
28.//                        |        SCLK/P1.5|<-------|P3.3/UCB0CLK     |
29.//
30.// Instructions:
31.//
32.// 1. Set the following parameters in hyperterminal
33.//                                  Port :       COM1
```

```
34. //                                  Baud rate :  19200
35. //                                  Data bits:   8
36. //                                  Parity:            None
37. //                                  Stop bits:   1
38. //                                  Flow Control:None
39. // 2. This lab requires to configure the USI module of MSP430F2013
40. //    as slave and MSP430FG4618 as master in SPI mode.
41. // 3. Connect the following jumpers on header 1 (H1) on the experimenter's
42. //    board.  [1-2], [3-4], [5-6], [7-8]
43. /*          H1
44.           _______
45.          1|-------|2
46.          3|-------|4
47.          5|-------|6
48.          7|-------|8
49.           |_______|
50.
51.       Jumper must be present on      PWR1, PWR2 and JP2.
52.
53. // 4. Download and run this code by the connecting the FET debugger to
54. //    JTAG1 on the experimenter's board.
55. // 5. Make sure the device selected is MSP430FG4618 in the General Options of IAR.
56.
57. ******************************************************************************/
58.
59. #include  "msp430xG46x.h"
60. #include <stdio.h>
61.
62. #define LED_ON_STATE    0x31  // character '1'
63. #define LED_OFF_STATE   0x30    // character '0'
64. #define LED_NUL_STATE   0x00    // character NULL - used for dummy write operation
65.
66. #define LED_ON          0x01
67. #define LED_OFF         0x00
68.
69. unsigned char ch;  // hold char from UART RX
70. unsigned char rx_flag;   // receiver rx status flag
71.
72. char gm1[] = "Press 'y' to turn ON and 'n' to turn OFF the LED";
73. char gm2[] = "Type in 'y' or 'n'!";
74.
75. void SPISetup(void)
76. {
77.   UCB0CTL0 = UCMSB + UCMST + UCSYNC;       // sync. mode, 3-pin SPI, Master mode,
    8-bit data
78.   UCB0CTL1 = UCSSEL_2 + UCSWRST;     // SMCLK and Software reset
79.   UCB0BR0 = 0x02;                    // Data rate = SMCLK/2 ~= 500kHz
80.   UCB0BR1 = 0x00;
81.   P3SEL |= BIT1 + BIT2 + BIT3;             // P3.1,P3.2,P3.3 option select
82.   UCB0CTL1 &= ~UCSWRST;              // **Initialize USCI state machine**
83. }
84.
85. unsigned char SPIGetState(void)
```

```
{
    while((P3IN & 0x01));             // Verifies busy flag
    IFG2 &= ~UCB0RXIFG;
    UCB0TXBUF = LED_NUL_STATE;        // Dummy write to start SPI
    while (!(IFG2 & UCB0RXIFG));      // USCI_B0 TX buffer ready?
    return UCB0RXBUF;
}

void SPISetState(unsigned char State)
{
   while(P3IN & 0x01);                // Verifies busy flag
   IFG2 &= ~UCB0RXIFG;
   UCB0TXBUF = State;                 //  write new state
   while (!(IFG2 & UCB0RXIFG));       // USCI_B0 TX buffer ready?
   }

   void UART0_putchar(char c) {
     // wait for other character to transmit
     while (!(IFG2 & UCA0TXIFG));
     UCA0TXBUF = c;
   }

   void Serial_Initialize(void)
   {
     P2SEL |= BIT4+BIT5;            // Set UC0TXD and UC0RXD to transmit and
   receive data
     UCA0CTL1 |= BIT0;              // Software reset
     UCA0CTL0 = 0;            // USCI_A0 control register
     UCA0CTL1 |= UCSSEL_2;    // Clock source SMCLK - 1048576 Hz
     UCA0BR0=54;                    // baud rate - 1048576 Hz / 19200
     UCA0BR1=0;               //
     UCA0MCTL=0x0A;           // Modulation
     UCA0CTL1 &= ~BIT0;             // Software reset
     IE2 |=UCA0RXIE;               // Enable USCI_A0 RX interrupt
   }

   void main(void)
   {

     WDTCTL = WDTPW+WDTHOLD;               // Stop watchdog timer
     Serial_Initialize();
     SPISetup();

     _EINT();                           // Enable global interrupts
     // Greeting Message
     for(int i = 0; i < 49; i++) {
           ch = gm1[i];
           UART0_putchar(ch); // print the greeting message on hyperterminal
       }
     UART0_putchar('\n');    // newline
     UART0_putchar('\r');    // carriage return

     while(1)
```

```
138.     {
139.
140.       while(!(rx_flag&0x01)); // wait until receive the character from
   HyperTerminal
141.       rx_flag = 0;            // clear rx_flag
142.
143.       switch (ch)
144.       {
145.       case  'y' :
146.                 SPISetState(LED_ON_STATE);
147.                 for(int i = 1000; i > 0;i--); //delay
148.                 UART0_putchar(SPIGetState());  // prints the current state of LED
149.                                                 // '1' - ON ; '0' - OFF
150.             break;
151.       case  'n'  :
152.                 SPISetState(LED_OFF_STATE);
153.                 for(int i = 1000; i > 0;i--); //delay
154.                 UART0_putchar(SPIGetState()); // prints the current state of LED
155.                                       // '1' - ON ; '0' - OFF
156.             break;
157.       default     :
158.                   for(int i = 0; i < 20; i++) {
159.                       ch = gm2[i];
160.                       UART0_putchar(ch); // print the greeting message on
   hyperterminal
161.                   }
162.                   UART0_putchar('\n');    // newline
163.                   UART0_putchar('\r');    // carriage return
164.             break;
165.       }
166.     }
167.   }
168.
169.   // Interrupt for USCI Rx
170.   #pragma vector=USCIAB0RX_VECTOR
171.   __interrupt void USCIB0RX_ISR (void)
172.   {
173.     ch = UCA0RXBUF;  // character received is moved to a variable
174.     rx_flag=0x01;          // signal main function receiving a char
175.   }
176.
```

Figure 3. Program to be run on MSP430FG4618 for SPI data transfer

```
1   /*****************************************************************************/
177.   //              Lab 5 - Serial Communication and SPI Interface
178.   //

179.   // Description: Using the MSP-EXP430FG4618 Development Tool establish a

180.   //           data exchange between the MSP430FG4618 and MSP430F2013
181.   //           devices using the SPI mode. The MSP430FG4618 uses the USCI
182.   //           module while the MSP430F2013 uses the USI module.
183.   //              MSP430FG4618 communicates with PC via RS232 module using
184.   //              USCI Serial Communication peripheral interface. This program
```

```
185.  //           takes user prompts the user to input a choice to turn ON or OFF
186.  //           the LED3 located on MSP430F2013. The user choice is communicated
187.  //           to MSP430FG4618 (master) via USCI serial interface and the
188.  //           corresponding action is communicated to MSP430F2013(slave) via
189.  //           SPI. Based on the user choice, MSP430F2013 will turn ON or OFF
190.  //           the LED3. This is the salve code that runs on MSP430F2013.
191.  //
192.  //                                Slave                    Master
193.  //                            MSP430F2013              MSP430FG4618
194.  //                         -----------------          -----------------
195.  //                        |              XIN|-    /|\|              XIN|-
196.  //                        |                 |      | |                 |  32kHz
   xtal
197.  //                        |             XOUT|-     --|RST           XOUT|-
198.  //                        |                 |        |                 |
199.  //                  LED <-|P1.0             |        |                 |
200.  //                        |         BF /P1.4|------->|P3.0/BF          |
201.  //                        |         SDI/P1.7|<-------|P3.1/UCB0SIMO    |
202.  //                        |         SDO/P1.6|------->|P3.2/UCB0SOMI    |
203.  //                        |        SCLK/P1.5|<-------|P3.3/UCB0CLK     |
204.  //
205.  // Instructions:
206.  //
207.  // 1. Set the following parameters in hyperterminal
208.  //                                      Port :       COM1
209.  //                                      Baud rate :  19200
210.  //                                      Data bits:   8
211.  //                                      Parity:             None
212.  //                                      Stop bits:   1
213.  //                                      Flow Control:None
214.  // 2. This lab requires to configure the USI module of MSP430F2013
215.  //    as slave and MSP430FG4618 as master in SPI mode.
216.  // 3. Connect the following jumpers on header 1 (H1) on the experimenter's
217.  //    board.  [1-2], [3-4], [5-6], [7-8]
218.  /*              H1
219.                 _______
220.               1|-------|2
221.               3|-------|4
222.               5|-------|6
223.               7|-------|8
224.                |_______|
225.
226.        Jumper must be present on   PWR1, PWR2 and JP2.
227.
228.  // 4. Download and run this code by the connecting the FET debugger to
229.  //    JTAG2 on the experimenter's board.
230.  // 5. Make sure the device selected is MSP430F2013 in the General Options of
   IAR.
231.
232.  *******************************************************************/
233.
```

```
234.
235.   #include "msp430x20x3.h"
236.
237.   #define LED_ON_STATE    0x31  // character '1'
238.   #define LED_OFF_STATE   0x30  // character '0'
239.   #define LED_NUL_STATE   0x00  // character NULL - used for dummy write operation
240.
241.   #define LED_ON          0x01
242.   #define LED_OFF         0x00
243.
244.   #define SET_BUSY_FLAG()   P1OUT |= 0x10;
245.   #define RESET_BUSY_FLAG() P1OUT &= ~0x10;
246.
247.   #define SET_LED()       P1OUT |= 0x01;
248.   #define RESET_LED()     P1OUT &= ~0x01;
249.
250.   unsigned char LEDState ;
251.   unsigned char NextState;
252.
253.   void SPISetup(void)
254.   {
255.
256.     USICTL0 |= USISWRST;      //Set UCSWRST -- needed for re-configuration process
257.     USICTL0 |= USIPE5 + USIPE6 + USIPE7 + USIOE; // SCLK-SDO-SDI port enable,MSB
   first
258.     USICTL1 = USIIE;          // USI  Counter Interrupt enable
259.     USICTL0 &= ~USISWRST;     // **Initialize USCI state machine**
260.   }
261.
262.
263.
264.   void InitComm(void)
265.   {
266.    USICNT = 8;                // Load bit counter, clears IFG
267.    USISRL = LEDState;         // set LED state
268.    RESET_BUSY_FLAG();         // reset busy flag
269.   }
270.
271.   void LEdInit(unsigned char state)
272.   {
273.     if (state == LED_OFF_STATE)
274.     {
275.       RESET_LED();
276.       LEDState = LED_OFF_STATE;
277.     }
278.     else
279.     {
280.       SET_LED();
281.       LEDState = LED_ON_STATE;
282.     }
283.     P1DIR |= 0x11;                        //  P1.0,4 output
284.   }
285.
```

```
286.  void SystemInit()
287.  {
288.
289.    WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
290.
291.   BCSCTL1 = CALBC1_1MHZ;                // Set DCO
292.   DCOCTL = CALDCO_1MHZ;
293.  }
294.
295.  void main(void)
296.  {
297.
298.    WDTCTL = WDTPW + WDTHOLD;          // Stop watchdog timer
299.    LEdInit(LED_OFF_STATE);           //LED state initialization
300.    SPISetup();                       //USi module in SPI mode initialization
301.    InitComm();                       //Communication initialization
302.    while(1)
303.    {
304.      _BIS_SR(LPM0_bits + GIE);           // Enter LPM0 w/ interrupt
305.
306.      switch (NextState)
307.      {
308.        case 0x00 :
309.          break;
310.        default :
311.          LEDState = NextState;          // new state
312.          break;
313.       }
314.      if (LEDState == LED_OFF_STATE)
315.      {
316.        RESET_LED();
317.      }
318.      else
319.      {
320.        SET_LED();
321.      }
322.      USISRL = LEDState;            // prepares new communication with new state
323.      RESET_BUSY_FLAG();          // clears busy flag - ready for new
   communication
324.
325.    }
326.  }
327.
328.  #pragma vector=USI_VECTOR
329.  __interrupt void USI_ISR(void)
330.  {
331.    SET_BUSY_FLAG();            // set busy flag - slave is ready with a new
   communication
332.    NextState = USISRL;         // read new command
333.    USICNT = 8;                 // Load bit counter for next TX
334.    _BIC_SR_IRQ(LPM4_bits);     // Exit from LPM4 on RETI
335.  }
```

Figure 4. Program to be run on MSP430F2013 for SPI data transfer

1.3. References

In order to understand more about UART communication and the USCI peripheral device, please access the following references:

- Davies Text, pages 497-520 and pages 520-534 (examples)
- MSP430FG4618 User's Guide, Chapter 20, pages 587-610 (USCI in SPI mode)
- MSP430F2013 User's Guide, Chapter 14, pages 405-420 (USI in SPI mode)

1.4. Assignment

Implement SPI communication between MSP430FG4618 and MSP430F2013 on TI experimenter's board. MSP430FG4618 should communicate with user via UART. User must be asked to press '1', '2', '3', or '4' to set the blink frequency of LED3 (after each selection, the user should be prompted again for a new selection). At power-on, LED3 should be blinking at a rate of 1x. Then, if a '2' is sent through the UART, the blinking rate should change to 2x. If a '3' is sent, the blink rate should go to 3x. If a '4' is sent, the blink rate should go to 4x. The initial blink rate (or the 1x blink rate) is up to you, but it should obvious that it is blinking and changing rates. You must use the SPI connection between the MSP430FG4618 and the MSP430F2013 to send the new blinking rates.