CPE 323
MODULE 10
Synchronous Serial Interface (SPI) Communication

Aleksandar Milenković

Email: milenka@uah.edu

Web: http://www.ece.uah.edu/~milenka

Overview

This module discusses the SPI synchronous communication protocol and its implementation using the USCI peripheral (in MSP430F5529). Specifically, the following topics are covered: (a) System-view of SPI communication; (b) Configuration of the USCI peripheral device for SPI mode; and (b) Implementation of SPI communication between two Launchpad boards.

Objectives

- *Learners will understand hardware and software aspects of serial communication*
- *Learners will be able to configure and interact with serial communication interfaces*
- *Learners will be able to evaluate pros and cons of each serial communication protocol (speed complexity)*

Contents

1 Synchronous Communication

This document continues covering communication protocols used in embedded systems, with a focus on MSP430 family of microcontrollers. We have already discussed asynchronous communication in UART mode and used it to communicate between an MSP-EXP430 Luanchpad board and a development workstation. Asynchronous communication is most useful when communication must be established between two distinct systems that each have their own clocks and there is no clock sharing. 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 system when components can share a clock. Typically, these protocols are used for communication between components on a single board (intra-board communication), though they can also be used to connect multiple boards (inter-board communication). Synchronous Peripheral Interface or SPI is a synchronous serial bidirectional protocol often used for communication between microcontrollers and other components on the board (e.g., sensors, memory modules).

`Things to remember 1-1. Synchronous Peripheral Interface or SPI.`

SPI is a synchronous serial bidirectional interface, typically used to connect a microcontroller to other components (e.g., sensors, external memory modules) on a single board. The communicating parties have a shared clock, allowing for high data bit rates (in order of ~Mbps).

2 SPI

In SPI mode, serial data is transmitted and received by two or multiple devices using a shared clock provided by a master device. This is the simplest synchronous communication protocol. Unlike the other synchronous communication protocol commonly used in embedded systems, I^2C, SPI is not standardized and there are several variations of SPI. Thus, you must read the data sheet of the device to ensure that the details of the protocol are well understood.

Figure 1 illustrates a system view of SPI style of communication between two devices. Because SPI is a synchronous protocol, a communication clock is shared between the two devices. In SPI nomenclature, the device is called Master (M) if it provides the clock (SCLK) and initiates communication. The other device is called Slave (S). The S device receives the clock from the M device, but the assumption is that the S device can carry out communication steps at the given clock rate. SPI is a bidirectional communication protocol by design – that means that data flows from M to S and from S to M concurrently. The names of data lines are as follows:

- MISO/SOMI – Master In Slave Out/Slave Out Master In (carries data from S to M)
- MOSI/SIMO – Master Out Slave In/Slave In Master Out (carries data from M to S).

The minimum number of wires is thus 3 (SCLK, MISO, MOSI) and SPI is sometimes referred to as a 3-wire protocol. Please note how we connect data lines. Unlike in UART mode where TxD of one device connects to RxD of the other device (or vice versa), the SPI data lines imply direction of the data flow, so MOSI pin of the M device is connected to the MOSI pin of the S device and MISO pin of the M device is connected to MISO pin of the S device. This is admirably clear and makes the functions unambiguous.

The forth signal, SS#, can be used to select a slave device (as shown in Figure 1) or to enable master device if configured as an input for the M device. It is usually active low and labeled SS for slave select, CS for chip select, or CE for chip enable. An S device takes part in communication and drives its output data pin only when SS# is active; the output data pin 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.

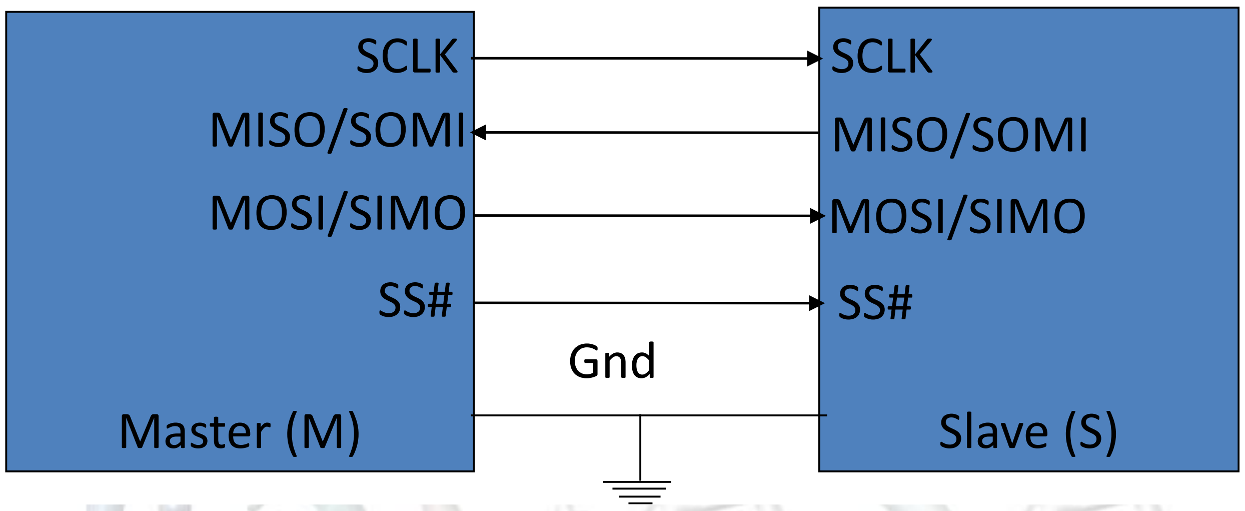


Figure 1. SPI communication: a system view.

`Things to remember 2-1. SPI.`

SPI involves at least 2 devices. A device that initiates communication and drives clock SCLK is called the master or M device. The other device is called slave or S device. On every SCLK clock one bit of data is sent from M to S over the MOSI data line and one bit of data is sent from S to M over the MISO data line. SCLK, MOSI, and SIMO are used in 3-wire configuration. In 4-wire configuration, an additional signal SS# is used to select a slave device in case that more than one S-device is connected in the system.

Figure 2 illustrates SPI data transmission. Logically we can think about SPI data exchange as having two shift registers (M shift and S shift registers) connected in series. Let us assume that the M shift register contains character ‘M’ and the S shift register contains character ‘S’. Once the M shift register has data, it starts generating SCLK. On every SCLK clock cycle one bit of data flows from M to S (over MOSI data line) and in return one bit of data flows from S to M (over MISO data line). Thus in 8 clock cycles the M and S device will exchange data, the M shift register contains 'S' and the S shift register contains 'M'. Here we assume that each device has exactly one data register. This a bit simplified view because we typically have separate transmit and receive shift registers.

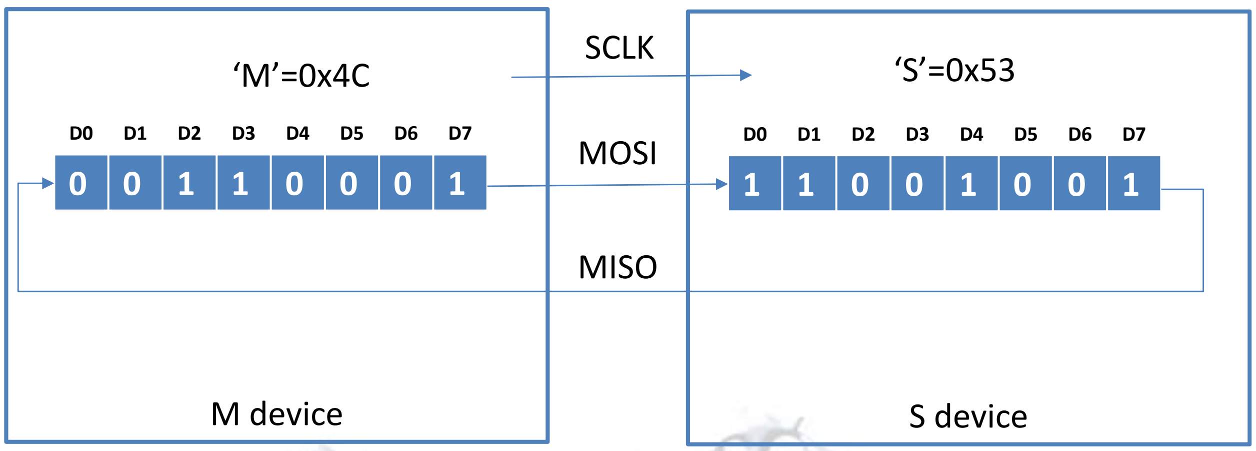


Figure 2. SPI data transmission.

3 USCI

The MSP430's Universal Serial Communication Interface (USCI) can be configured to work in SPI mode. Both channels A and B support SPI mode. An MSP430 may include more than one USCI device. For example an MSP430 with two USCI devices will have communication channels UCA0, UCB0, UCA1, UCB1, all capable to carry out SPI communication if the UCSYNC bit is set and SPI mode is selected with the UCMODEx bits (3-wire or 4-wire). SPI mode allows us to specify the following: 7-bit or 8-bit data length; LSB-first or MSB-first; 3-pin or 4-pin operation; M or S mode; selectable clock polarity and phase control; and a programmable clock frequency.

Figure 3 shows a block diagram of USCI when configured in SPI mode. Its resources are the same as seen in UART mode: the double-buffered transmit portion (TXBUF and the corresponding shift register), the double-buffered receive portion (RXBUF and the corresponding shift register), and the baud rate generator. The data pins and clock pins are: UCxSOMI, UCxSIMO, UCxCLK, and UCxSTE.

In SPI mode, serial data is transmitted and can be received by multiple devices using a shared clock provided by the M device. The signals are as follows:

- UCxSIMO – slave in, master out (M – output data pin, S – input data pin)
- UCxSOMI – slave out, master in (M – input data pin, S – output data pin)
- SCxCLK – USCI SPI clock (M – output clock, S – input clock)
- UCxSTE – slave transmit enable (unused in 3 wire mode).

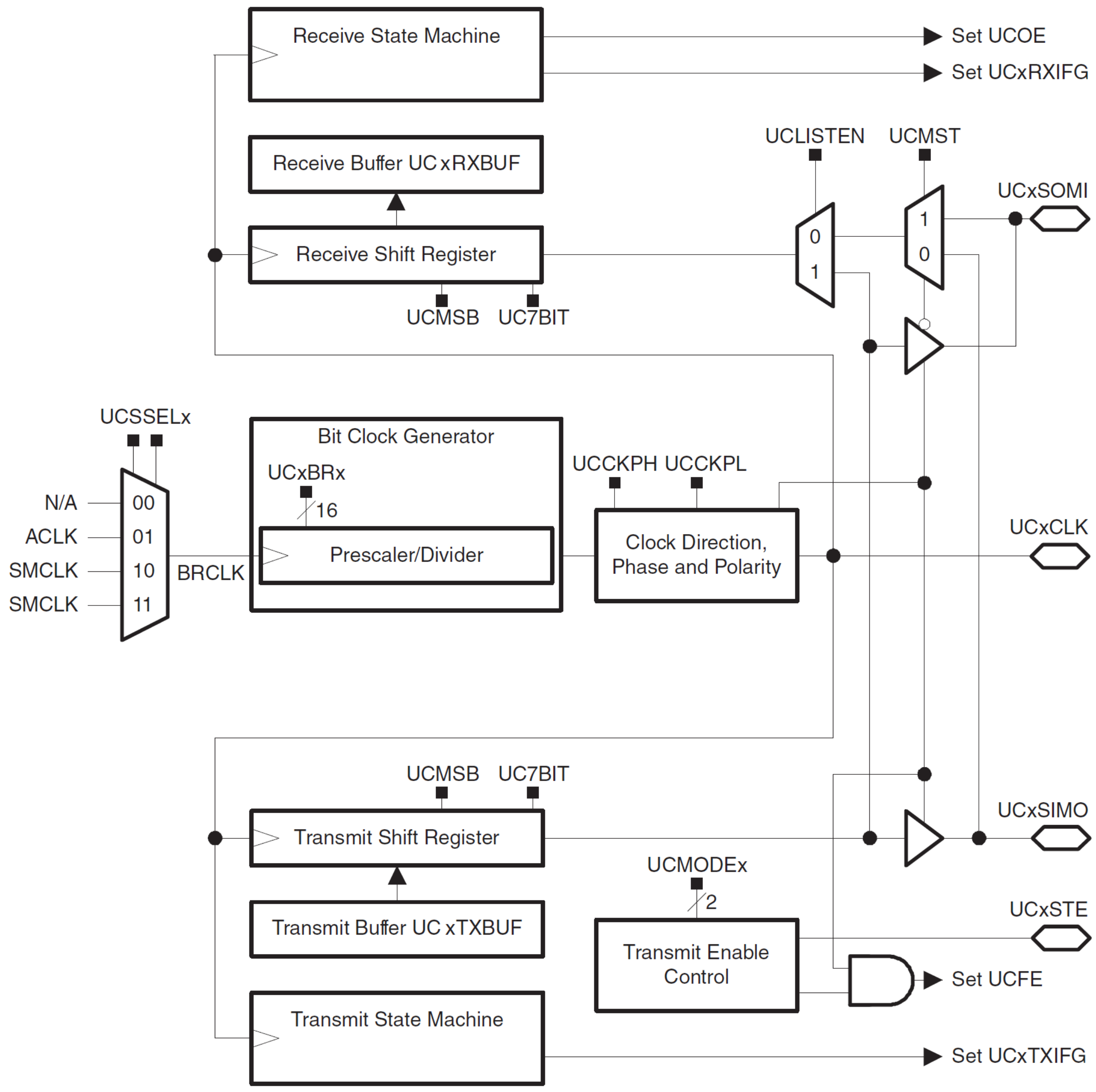


Figure 3. Block diagram of USCI in SPI mode.

The operation of UCxSTE is specified by Table 1. To support various implementation of SPI mode, the UCxSTE can be active at a logic ‘1’ (high) or at a logic ‘0’ (low). It is an input pin and can be used for both S and M devices.

Table 1. UCxSTE Operation. UCMODEx=01 means that UCxSTE is active high: 0 – inactive (S) / active (M), 1 – active (S) / inactive (M). UCMODEx=10 means that UCxSTE is active low: 0 – active (S) / inactive (M), 1 – inactive (S) / active (M)

UCMODEx	UCxSTE Active State	UCxSTE	Slave	Master
01	High	0	Inactive	Active
		1	Active	Inactive
10	Low	0	Active	Inactive
		1	Inactive	Active

SPI Master Mode. Figure 4 shows the USCI as a master in both 3-pin and 4-pin configurations. The USCI initiates data transfer when data is moved to UCxTXBUF (namely UCAxTXBUF or UCBxTXBUF). The data from the UCxTXBUF is moved into the transmit shift register when it is empty, and then it is transferred bit-by-bit over UCxSIMO pin (either MSB-first or LSB-first, depending on the UCMSB setting). Data bit on UCxSOMI is shifted into the receive shift register on the opposite edge of the clock. When the entire character is received (7-bit or 8-bit), the data is moved from the receive shift register to the UCxRXBUF and the receive interrupt flag UCRXIFG is set, indicating that the transfer is complete. Please note that to receive data from the S device, the M device must send something to the S device by writing into its UCxTXBUF (even though this data may not be useful to the S device).

In 4-pin master mode, UCxSTE is used to prevent conflicts with another master and controls the master as described in Table 1. Please note that the master may use digital I/O pins (Px.x.) connected to corresponding slaves' UCxSTE pins to select a particular slave in case it interfaces multiple slave devices.

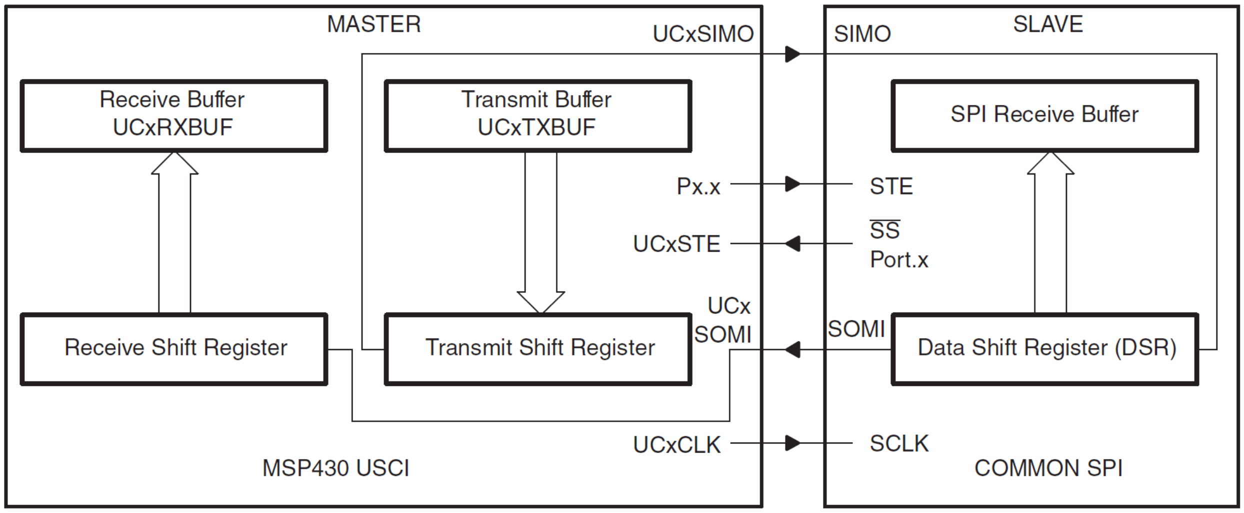


Figure 4. SPI Master Mode.

SPI Slave Mode. Figure 5 shows the USCI as a slave in both 3-pin and 4-pin configurations. UCxCLK is used as the input for the SPI clock and must be supplied by the external master. Data written in UCxTXBUF of the S device is moved into the transmit shift register before the start of UCxCLK. It is shifted out through UCxSOMI. Data on UCxSIMO is shifted into the receive shift register on the opposite edge of UCxCLK and moved to UCxRXBUF when the specified number of bits is received (UCRXIFG flag is set). In 4-pin slave mode, UCxSTE is used to enable transmit and receive operations and is provided by the master. When the UCxSTE is in the slave-active state, the slave operates normally. When UCxSTE is in the slave-inactive mode (see Table 1), any receive operation on UCxSIMO is halted and UCxSOMI is set to input direction.

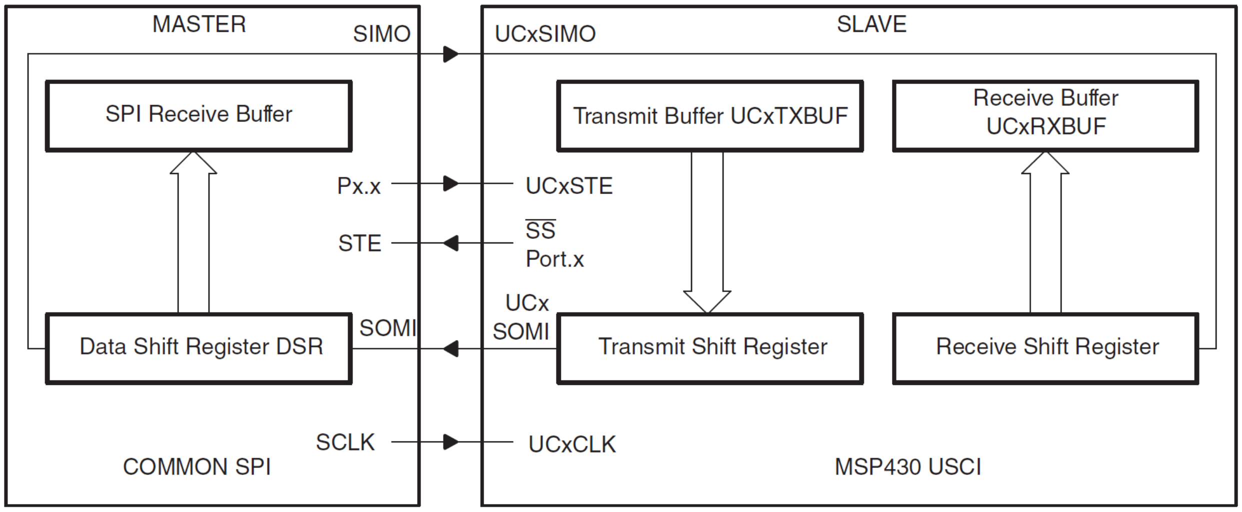


Figure 5. SPI Slave Mode.

The bit clock generator is activated when we write to the UCxTXBUF of the master device. In slave mode, transmission begins when a master provides a clock (providing UCxSTE is active in 4-pin mode). The 16-bit value of UCxBRx (UCxBR0 and UCxBR1) is the division factor of the USCI clock source, BRCLK. Modulation is not used in SPI mode, and UCAxMCTL should always be cleared. The clock frequency is determined as follows (if UCBRx=0, the bit clock is equal to source clock):

$$f_{BitClock} = f_{BRCLK}/UCBRx$$

The polarity and phase of the UCxCLK are independently configured via the UCCKPL and UCCKPH control bits of the USCI. Timing for each of the four possible cases is shown in Figure 6. As discussed above, SPI is not standardized, and 4 combinations for clock polarity and phase are available, so you can configure your USCI device to match any implementation that could be used by the other communicating party. Please note that not all combinations could be used or make sense in 3-wire mode (e.g., using CKKPH=1 in S mode in a 3-wire protocol does not make sense because there is not trigger to start data shifting).

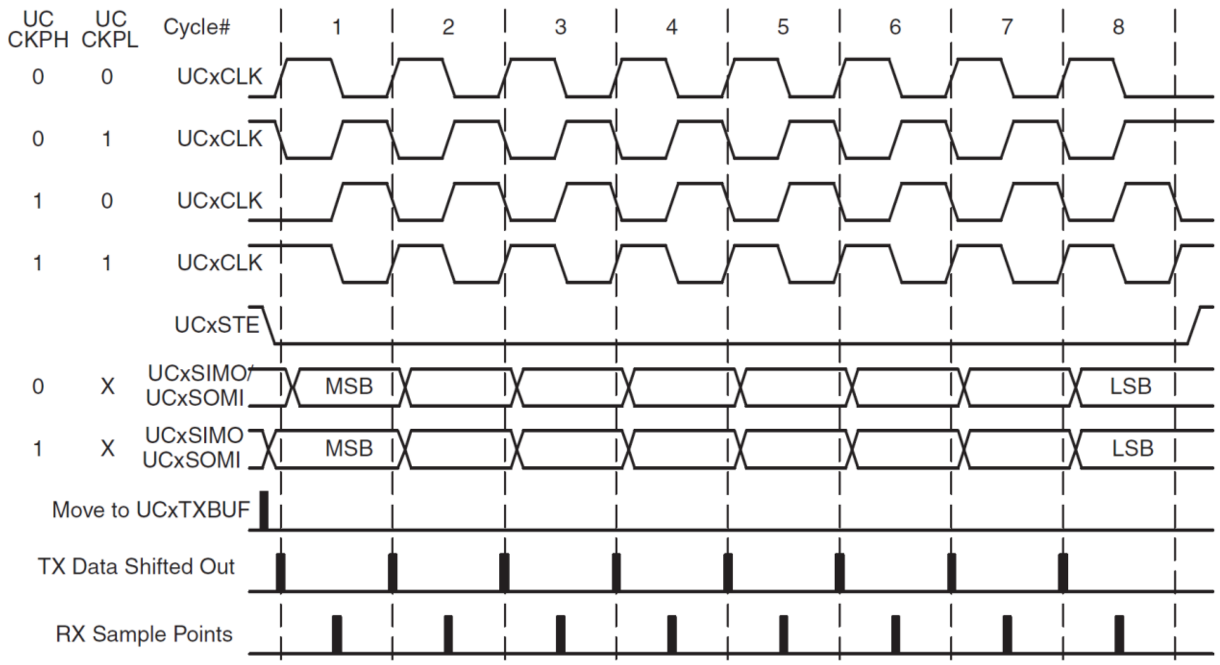
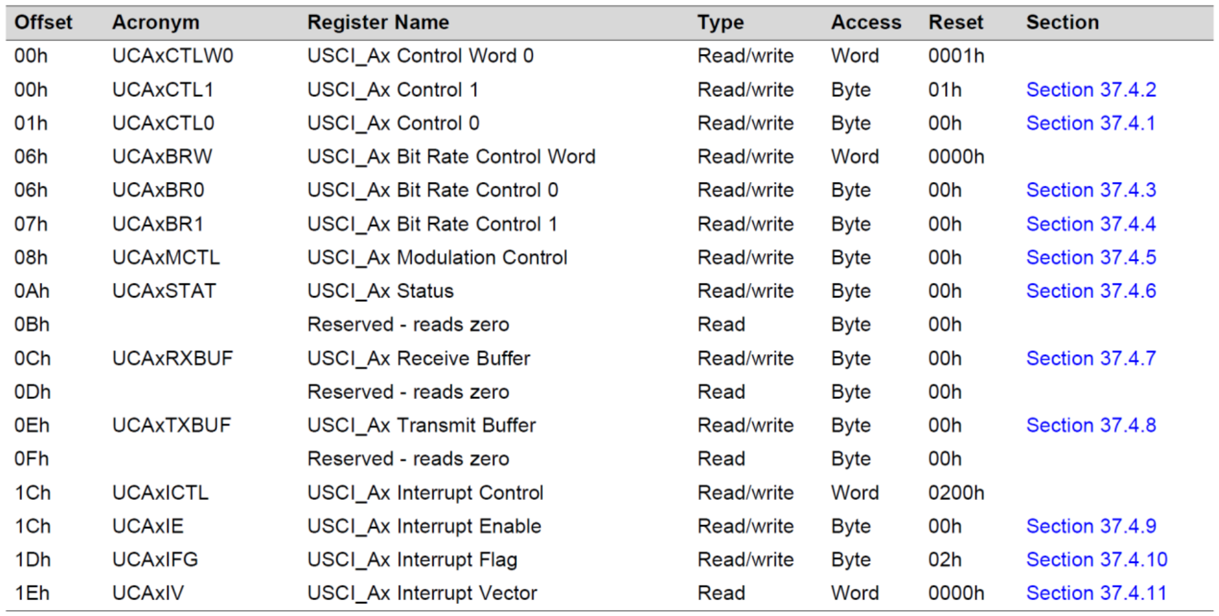


Figure 6. USCI SPI Timing with UCMSB=1. UCCKPH (Clock Phase Select): 0 (Active/Inactive) - data is changed on the first UCxCLK edge and captured on the following edge; 1 (Inactive/Active) – data is captured on the first edge and changed on the following edge; CKPL (Clock Polarity): 0 - idles at 0, 1 - idles at 1.

The USCI registers visible to programmers for USIC_Ax are shown in Figure 7. USCI is an 8-bit peripheral device and all registers are 8-bit long. For USCI_A0 (UCA0), the notable registers are two control registers (UCA0CTL0 and UCA0CTL1), baud rate control registers (UCA0BR0 and UCA0BR1), modulation control register (UCA0MCTL), status register (UCA0STAT), receive buffer (UCA0RXBUF), and transmit buffer (UCA0TXBUF).



Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCAxCTLW0	USCI_Ax Control Word 0	Read/write	Word	0001h	
00h	UCAxCTL1	USCI_Ax Control 1	Read/write	Byte	01h	Section 37.4.2
01h	UCAxCTL0	USCI_Ax Control 0	Read/write	Byte	00h	Section 37.4.1
06h	UCAxBRW	USCI_Ax Bit Rate Control Word	Read/write	Word	0000h	
06h	UCAxBR0	USCI_Ax Bit Rate Control 0	Read/write	Byte	00h	Section 37.4.3
07h	UCAxBR1	USCI_Ax Bit Rate Control 1	Read/write	Byte	00h	Section 37.4.4
08h	UCAxMCTL	USCI_Ax Modulation Control	Read/write	Byte	00h	Section 37.4.5
0Ah	UCAxSTAT	USCI_Ax Status	Read/write	Byte	00h	Section 37.4.6
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCAxRXBUF	USCI_Ax Receive Buffer	Read/write	Byte	00h	Section 37.4.7
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCAxTXBUF	USCI_Ax Transmit Buffer	Read/write	Byte	00h	Section 37.4.8
0Fh		Reserved - reads zero	Read	Byte	00h	
1Ch	UCAxICTL	USCI_Ax Interrupt Control	Read/write	Word	0200h	
1Ch	UCAxIE	USCI_Ax Interrupt Enable	Read/write	Byte	00h	Section 37.4.9
1Dh	UCAxIFG	USCI_Ax Interrupt Flag	Read/write	Byte	02h	Section 37.4.10
1Eh	UCAxIV	USCI_Ax Interrupt Vector	Read	Word	0000h	Section 37.4.11

Figure 7. USCI_Ax SPI Mode Control and Status Registers

3.1 USCI Initialization: SPI Mode

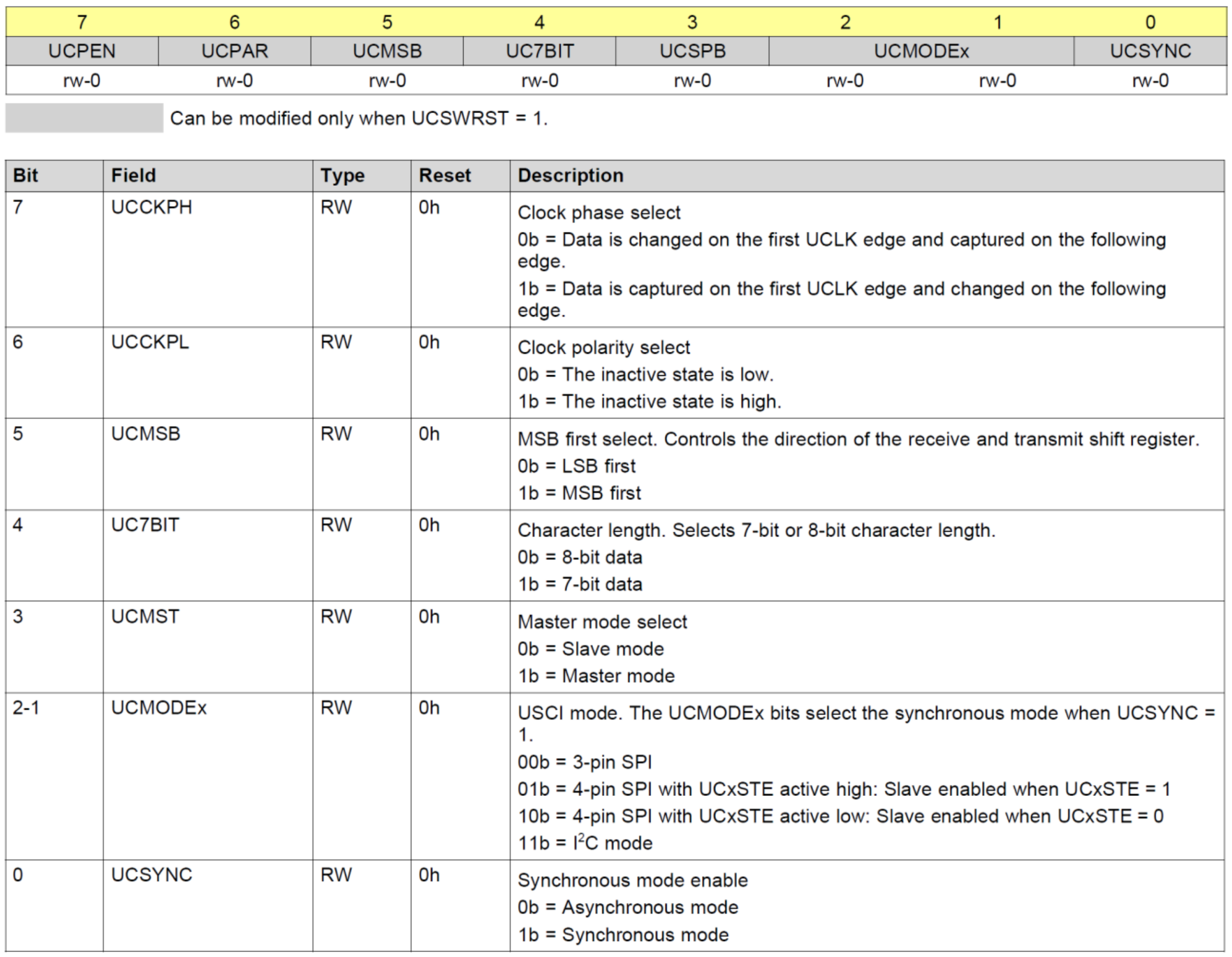
To initialize the USCI in SPI mode the following sequence of steps is recommended:

1. Set UCSWRST bit (software reset: BIS.B #UCSWRST, &UCAxCTL1) to reset the USCI state machine;
2. Initialize all USCI registers with UCSWRST=1 (UCxBRx, UCxCTL1);
3. Configure ports;
4. Clear UCSWRST (BIS.B #UCSWRST, &UCAxCTL1);
5. Enable interrupts (optional) by setting UCAxRXIE and UCAxTXIE.

3.2 USCI Control Registers in SPI Mode

You should already be familiar with programmer's view of the USCI device in UART mode. The following are programmer's view of these registers in SPI mode. Figure 8 and Figure 9 show the format and description of relevant bits for UCAxCTL0 and UCA0xCTL1, respectively. Figure 10 shows the format of modulation register that should always be cleared in SPI mode. Figure 11 shows the format of UCAxSTAT register. The UCAxBR0 and UCAxBR1 registers contain lower and upper byte, respectively, of the prescalar setting. Typically, the source clock is just divided by the value in these registers to create the SPI clock. These registers should be configured only if the device is working in the Master mode.

Figure 12 shows the format of registers related to the interrupts. The UCTXIFG interrupt flag is set by the transmitter to indicate that the UCAxTXBUF is ready to accept another character. The interrupt request is generated if UCTXIE and GIE are also set. UCTXIFG is automatically reset if a character is written to UCAxTXBUF. The UCRXIFG flag is set when a new character is received and loaded into UCAxRXBUF. An interrupt request is generated if UCRXIE and GIE are also set. UCRXIFG is automatically reset when UCAxRXBUF is read.

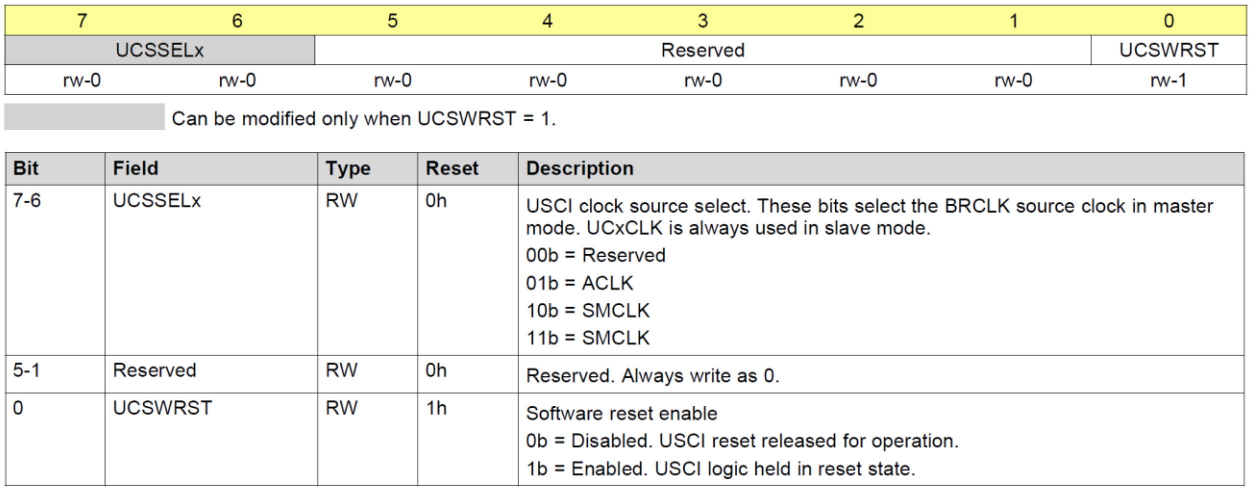


7	6	5	4	3	2	1	0
UCPEN	UCPAR	UCMSB	UC7BIT	UCSPB	UCMODEx		UCSYNC
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

▨ Can be modified only when UCSWRST = 1.

Bit	Field	Type	Reset	Description
7	UCCKPH	RW	0h	Clock phase select 0b = Data is changed on the first UCLK edge and captured on the following edge. 1b = Data is captured on the first UCLK edge and changed on the following edge.
6	UCCKPL	RW	0h	Clock polarity select 0b = The inactive state is low. 1b = The inactive state is high.
5	UCMSB	RW	0h	MSB first select. Controls the direction of the receive and transmit shift register. 0b = LSB first 1b = MSB first
4	UC7BIT	RW	0h	Character length. Selects 7-bit or 8-bit character length. 0b = 8-bit data 1b = 7-bit data
3	UCMST	RW	0h	Master mode select 0b = Slave mode 1b = Master mode
2-1	UCMODEx	RW	0h	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00b = 3-pin SPI 01b = 4-pin SPI with UCxSTE active high: Slave enabled when UCxSTE = 1 10b = 4-pin SPI with UCxSTE active low: Slave enabled when UCxSTE = 0 11b = I^2C mode
0	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

Figure 8. UCAxCTL0.



7	6	5	4	3	2	1	0
UCSSELx		Reserved					UCSWRST
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-1

▨ Can be modified only when UCSWRST = 1.

Bit	Field	Type	Reset	Description
7-6	UCSSELx	RW	0h	USCI clock source select. These bits select the BRCLK source clock in master mode. UCxCLK is always used in slave mode. 00b = Reserved 01b = ACLK 10b = SMCLK 11b = SMCLK
5-1	Reserved	RW	0h	Reserved. Always write as 0.
0	UCSWRST	RW	1h	Software reset enable 0b = Disabled. USCI reset released for operation. 1b = Enabled. USCI logic held in reset state.

Figure 9. UCAxCTL1.

7	6	5	4	3	2	1	0
Reserved							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Figure 10. UCAxMCTL. Should be always cleared in SPI mode.

7	6	5	4	3	2	1	0
UCLISTEN	UCFE	UCOE	Reserved				UCBUSY
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	r-0

▨ Can be modified only when UCSWRST = 1.

Bit	Field	Type	Reset	Description
7	UCLISTEN	RW	0h	Listen enable. The UCLISTEN bit selects loopback mode. 0b = Disabled 1b = Enabled. The transmitter output is internally fed back to the receiver.
6	UCFE	RW	0h	Framing error flag. This bit indicates a bus conflict in 4-wire master mode. UCFE is not used in 3-wire master or any slave mode. 0b = No error 1b = Bus conflict occurred.
5	UCOE	RW	0h	Overrun error flag. This bit is set when a character is transferred into UCxRXBUF before the previous character was read. UCOE is cleared automatically when UCxRXBUF is read, and must not be cleared by software. Otherwise, it does not function correctly. 0b = No error 1b = Overrun error occurred
4-1	Reserved	R	0h	Reserved. Always reads as 0.
0	UCBUSY	R	0h	USCI busy. This bit indicates if a transmit or receive operation is in progress. 0b = USCI inactive 1b = USCI transmitting or receiving

Figure 11. UCAxSTAT.

7	6	5	4	3	2	1	0
Reserved						UCTXIFG	UCRXIFG
r-0	r-0	r-0	r-0	r-0	r-0	rw-1	rw-0

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	UCTXIFG	RW	1h	Transmit interrupt flag. UCTXIFG is set when UCAxTXBUF empty. 0b = No interrupt pending 1b = Interrupt pending
0	UCRXIFG	RW	0h	Receive interrupt flag. UCRXIFG is set when UCAxRXBUF has received a complete character. 0b = No interrupt pending 1b = Interrupt pending

7	6	5	4	3	2	1	0
Reserved						UCTXIE	UCRXIE
r-0	r-0	r-0	r-0	r-0	r-0	rw-0	rw-0

Bit	Field	Type	Reset	Description
7-2	Reserved	R	0h	Reserved. Always reads as 0.
1	UCTXIE	RW	0h	Transmit interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled
0	UCRXIE	RW	0h	Receive interrupt enable 0b = Interrupt disabled 1b = Interrupt enabled

15	14	13	12	11	10	9	8
UCIVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
UCIVx							
r0	r0	r0	r-0	r-0	r-0	r-0	r0

Bit	Field	Type	Reset	Description
15-0	UCIVx	R	0h	USCI interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: Data received; Interrupt Flag: UCRXIFG; Interrupt Priority: Highest 04h = Interrupt Source: Transmit buffer empty; Interrupt Flag: UCTXIFG; Interrupt Priority: Lowest

Figure 12. UCAxIFG, UCAxIE, and UCAxIV Registers.

4 Code Examples

Code 1 shows a demo program for the SPI master that carries out communication in SPI mode between two MSP-EX430F529 launchpad boards using the USCI0, channel A (UCA). The code initializes the master for SPI Master-mode in the SPI_Master_UCA0_Setup() subroutine as follows:

- Specify port special functions: P3.3 is SIMO, P3.4 is SOMI, and P2.7 is SCLK;
- Master mode, 8-bit data, clock polarity is high, MSB-first;
- Bit clock is set to SMCLK/2 (2^{19} Hz).

The program first waits for the slave device to get ready (a positive pulse on P1.2). The MST_Data is initially set to 0x01 and SLV_Data to 0x00. The communication is carried out in the infinite loop using polling. The master waits for the transmit buffer to get ready, and then writes MST_Data into UCA0TXBUF. This will trigger SPI exchange as described above. The program waits for RXIFG to get ready, and then reads data received from the slave. The received data should be equal to the character that the master previously sent to the slave (in the previous exchange). The correct exchange keeps LED1 on and toggles LED2. If the received data does not match, the LED1 is off and LED2 is not toggled. The MST_Data and SLV_Data are updated and the cycle repeats after a delay of 100,000 clock cycles. This delay is inserted just for us to be able to observe data exchange.

```c
//******************************************************************************
//   MSP430F5529 Demo Program - USCI_A0, SPI 3-Wire Master Incremented Data
//
//   Description: SPI master talks to SPI slave using 3-wire mode. Incrementing
//   data is sent by the master starting at 0x01. Received data is expected to
//   be same as the previous transmission.
//   Once UCA0 is initialized in SPI Master mode, as follows:
//   BRCLK=SMCLK/2, 3-wire mode, clock polarity is high, MSB is sent first.
//   The main loop is entered if P1.2 is at logic 1 which indicates that
//   the slave device is ready.
//   Communication is handled in the infinite loop, as follows:
//      A new character in MST_Data is written into TXBUF if it is empty.
//      if the received data corresponds to previously sent character,
//      the communication is carried out properly, LED1 is on, LED2 toggles.
//      Otherwise, LED1 is off, LED2 is off.
//      The MST_Data and SLV_Data are updated, delay is applied so we
//      can verify program behavior through LED1&LED2.
//
//                   MSP430F552x
//                 -----------------
//                |                 |
//                |                 |
//                |             P1.0|-> LED1
//                |                 |
//                |             P3.3|-> Data Out (UCA0SIMO)
//                |                 |
//                |             P3.4|<- Data In (UCA0SOMI)
//                |                 |
//  Slave RDY   ->|P1.2         P2.7|-> Serial Clock Out (UCA0CLK)
//
//
//   A. Milenkovic, milenkovic@computer.org
//
//   October 2022
//******************************************************************************

#include <msp430.h>

unsigned char MST_Data,SLV_Data;
unsigned char temp;

void SPI_Master_UCA0_Setup(void) {
      P3SEL |= BIT3+BIT4;                       // P3.3,4 option select
      P2SEL |= BIT7;                            // P2.7 option select

      UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
      UCA0CTL0 |= UCMST+UCSYNC+UCCKPL+UCMSB;    // 3-pin, 8-bit SPI master
                                                // Clock polarity high, MSB
      UCA0CTL1 |= UCSSEL_2;                     // SMCLK
      UCA0BR0 = 0x02;                           // /2
      UCA0BR1 = 0;                              //
      UCA0MCTL = 0;                             // No modulation
      UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
}
```

```
int main(void) {

  WDTCTL = WDTPW+WDTHOLD;                   // Stop watchdog timer

  P1OUT = 0;                                // LED1 is OFF
  P1DIR |= BIT0;                            // Set P1.0 as output
  P4DIR |= BIT7;
  P4OUT &= ~BIT7;
  SPI_Master_UCA0_Setup();                  // Initialize SPI interface

  // Wait for Slave
  while (!(P1IN&BIT2));                     // Wait until Slave is ready

  MST_Data = 0x01;                          // Initialize data values
  SLV_Data = 0x00;                          //
  for (;;) {
     while (!(UCA0IFG&UCTXIFG));            // USCI_A0 TX buffer ready?
     UCA0TXBUF = MST_Data;                  // Transmit first character

     while(!(UCA0IFG&UCRXIFG));             // Wait for data back
     if (UCA0RXBUF==SLV_Data){              // Test for correct character RX'd
       P1OUT |= BIT0;                       // If correct, light LED1
       P4OUT ^= BIT7;                       // heart bit on LED2
     } else {
       P1OUT &= ~BIT0;                      // If incorrect, turn off LED1
     }
     MST_Data = (MST_Data + 1) % 50;
     SLV_Data = (SLV_Data + 1) % 50;
     __delay_cycles(1000000);
  }
}
```

Code 1. SPI Demo Connecting two Launchpad boards– Master Code.

Code 2 shows the slave program that carries out the SPI communication described above. Please note that the slave does not specify the clock in the USCI setup procedure. The slave generates a positive pulse on P1.2 to indicate that it is ready, and then enters the main loop. It checks when a new character is received and then when TXBUF is ready, echoes the character back to the master.

```
//******************************************************************************
//   MSP430F552x Demo - USCI_A0, SPI 3-Wire Slave Data Echo
//
//   Description: SPI slave demo using 3-wire mode. Incrementing
//   data is sent by the master starting at 0x01. Received data is expected to
//   be same as the previous transmission.
//   Initialize SPI Slave mode, as follows:
//   3-wire mode, clock polarity is high, MSB is sent first.
//   Slave generated a logic high pulse on P1.2 indicating it is ready.
```

```c
//   Communication is handled in the infinite loop, as follows:
//      Once a new character is received it is echoed if TXBUF is ready.
//      LED2 is toggled providing visual indication of communication.
//
//                   MSP430F552x
//                 -----------------
//           LED1<-|P1.0             |
//                 |                 |
//Slave is Ready<-|P1.2             |
//                 |                 |
//                 |             P3.3|-> Data Out (UCA0SIMO)
//                 |                 |
//                 |             P3.4|<- Data In (UCA0SOMI)
//                 |                 |
//                 |             P2.7|-> Serial Clock Out (UCA0CLK)
//
//
//   Author: A. Milenkovic, milenkovic@computer.org
//
//   Date: October 2022
//******************************************************************************

#include <msp430.h>

void SPI_Slave_UCA0_Setup(void) {
    P3SEL |= BIT3+BIT4;                         // P3.3,4 option select
    P2SEL |= BIT7;                              // P2.7 option select
    UCA0CTL1 |= UCSWRST;                        // **Put state machine in reset**
    UCA0CTL0 |= UCSYNC+UCCKPL+UCMSB;            // 3-pin, 8-bit SPI slave,
                                                // Clock polarity high, MSB
    UCA0CTL1 &= ~UCSWRST;                       // **Initialize USCI state machine**
}

int main(void) {
  WDTCTL = WDTPW+WDTHOLD;                       // Stop watchdog timer

  SPI_Slave_UCA0_Setup();
  P1DIR |= BIT2 + BIT0;                         // Set P1.0 and P1.2 as outputs
  P1OUT |= BIT2 + BIT0;                         // LED1 is on, P1.2 is set
  __delay_cycles(100);
  P1OUT &= ~BIT2 ;                              // LED is on, P1.2 is off
  // P4.7 is heartbeat of the application (toggles on each received char)
  P4DIR |= BIT7;
  P4OUT = 0;

  for(;;) {
      while(!(UCA0IFG&UCRXIFG));  // wait for a new character
      while(!(UCA0IFG&UCTXIFG));  // new character is received, is TXBUF ready?
      UCA0TXBUF = UCA0RXBUF;      // echo character back if ready
      P4OUT ^= BIT7;              // Toggle LED2
  }
}
```

Code 2. SPI Demo Connecting two Launchpad boards – Slave Code.

5 Exercises