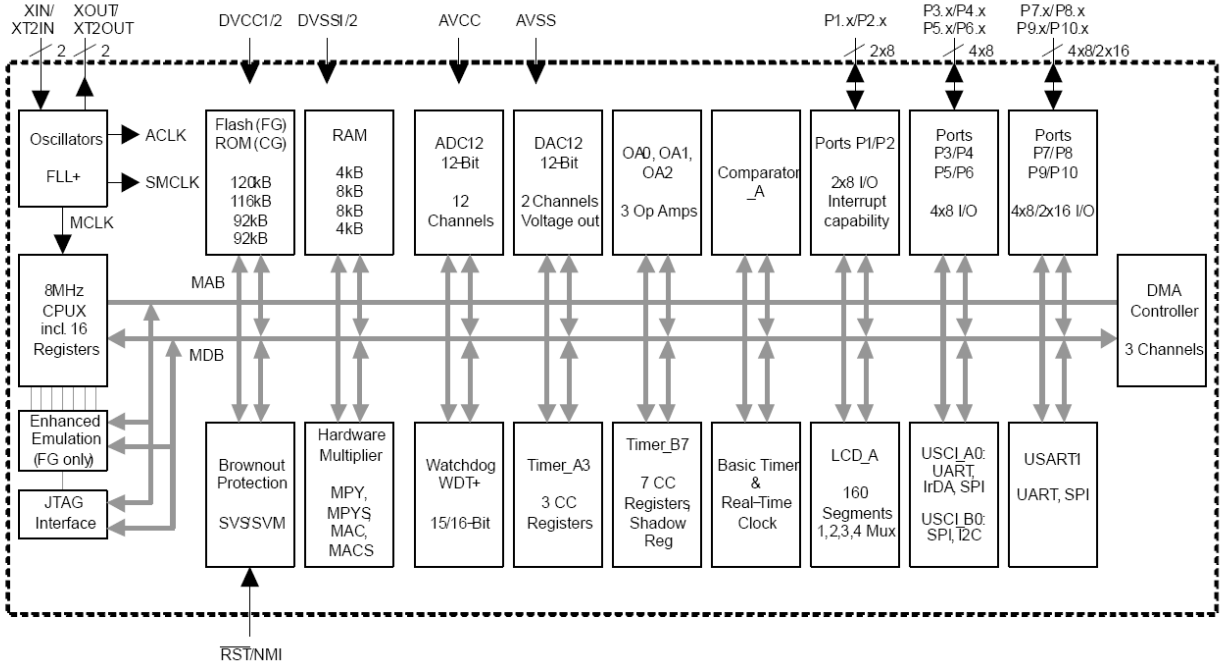
CPE 323
Intro to Embedded Computer Systems
SPI Serial Communication

Aleksandar Milenkovic

milenka@uah.edu

Admin

MSP430FG4618 Block Diagram



Communication

- Part of big 4
  - sense
  - process (compute)
  - store (memory)
  - communicate (UI, networks, ...)
- Communication in embedded systems
  - Between integrated circuits on PCB (e.g., μC ↔ sensors)
  - Between development platform and a workstation
  - Between embedded systems

Types of Communication

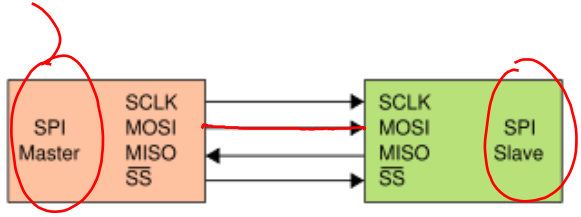
- Wired vs. wireless
- Serial vs. parallel
- Synchronous vs. asynchronous
- Unidirectional (simplex) vs. bidirectional (half-duplex and full-duplex)

Serial Communication in MSP430

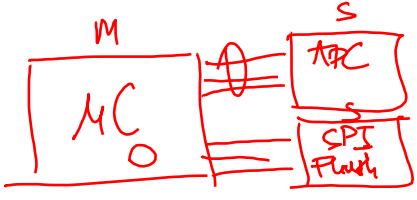
- Communication protocols
  - UART (Universal Asynchronous Receiver/Transmitter)
  - SPI (Serial Parallel Interface)
  - I^2C (Inter Integrated Circuit)
  - Infrared
- Peripheral devices
  - USCI – Universal Serial Communication Interface
  - USI – Universal Serial Interface
  - USART – Universal Synchronous/Asynchronous Receiver/Transmitter

SPI – Serial Peripheral Interface

- 4-wire vs. 3-wire (de-facto...)
- Signals
  - SCLK
  - MOSI/SIMO
  - MISO/SOMI
  - SS

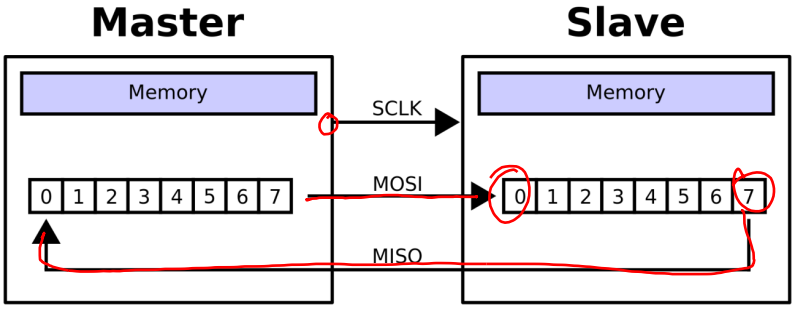


Serial Clock
Data line: Master Out Slave In / Slave In Master Out
Data line: Master In Slave Out / Slave Out Master In
Slave select



Data Transmission

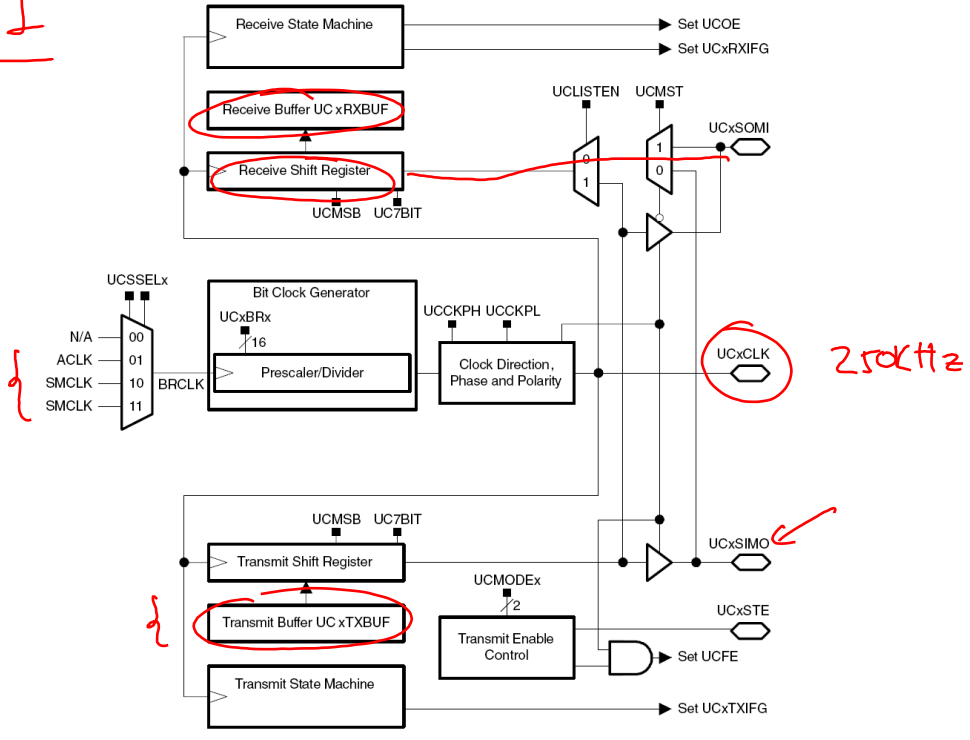
Synchronous, Serial



1. Configure itself as Master device
2. Configure clock (make sure the selected S device can work at that frequency)
3. Optionally, select the slave device (SS#)
4. Transmission
   Each SPI clock: a full-duplex transmission occurs (M: sends a bit on MOSI, receive a bit from MISO)

1. Configure itself as Slave device
2. If selected (SS# active) and SCLK is active
   Each SPI clock: S: sends a bit on MISO, receive a bit from MOSI)

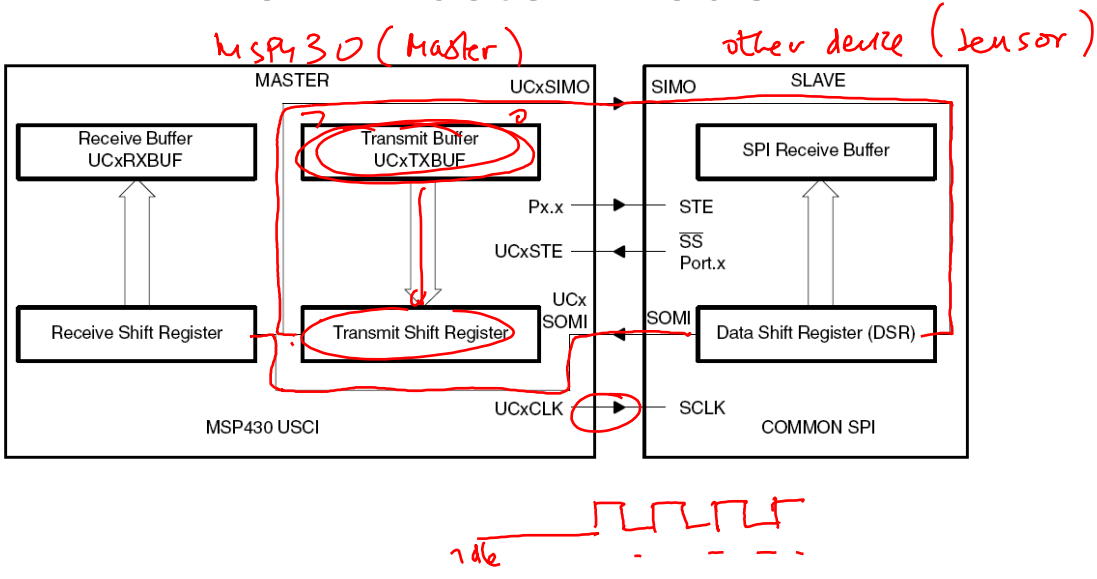
USCI



USCI

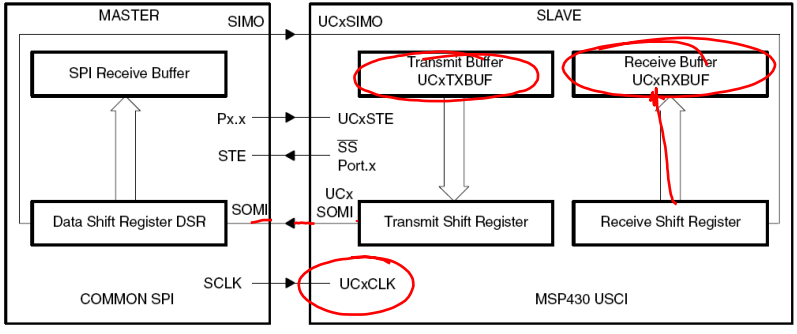
UCMODEx	UCxSTE Active State	UCxSTE	Slave	Master
01	high	0	inactive	active
		1	active	inactive
10	low	0	active	inactive
		1	inactive	active

SPI Master Mode



SPI Slave Mode

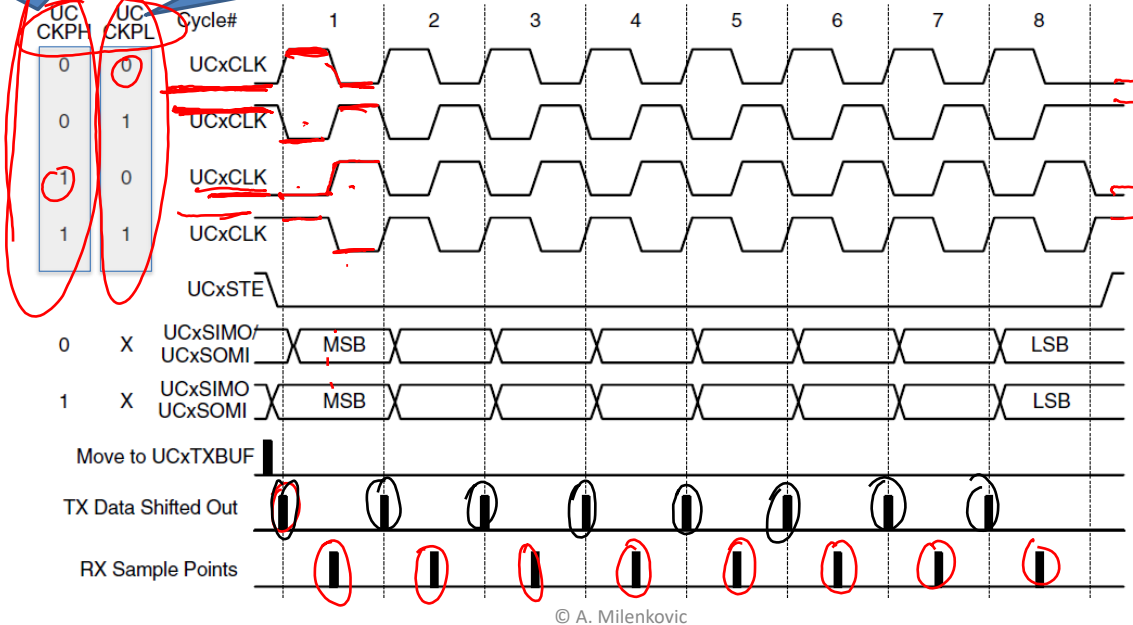
MSP430 (slave)



SPI Timing: Clock Polarity and Phase

Phase
0: Active/Inactive
1: Inactive/Active

Clock Polarity
0 – Idles at 0;
1 – Idles at 1



Clock phase select (UCCKPH)

- 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.

UCAxCTL0, UCAxCTL1

8-bit PER

Figure 37-5. UCAxCTL0 Register

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	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.

Table 37-3. UCAxCTL0 Register Description

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 = I²C mode
0	UCSYNC	RW	0h	Synchronous mode enable 0b = Asynchronous mode 1b = Synchronous mode

Figure 37-6. UCAxCTL1 Register

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.

Table 37-4. UCAxCTL1 Register Description

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.

BR1

Figure 37-7. UCAxBR0 Register

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when UCSWRST = 1.

Table 37-5. UCAxBR0 Register Description

Bit	Field	Type	Reset	Description
7-0	UCBRx	RW	undefined	Bit clock prescaler low byte. The 16-bit value of (UCAxBR0 + UCAxBR1 × 256) forms the prescaler value UCBRx. $f_{BRClock} = f_{BRCLK} / UCBRx$ If UCBRx = 0, $f_{BRClock} = f_{BRCLK}$

USCI Interrupts

Figure 37-13. UCAxIE Register

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

Table 37-11. UCAxIE Register Description

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

Figure 37-14. UCAxIFG Register

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

Table 37-12. UCAxIFG Register Description

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

Figure 37-15. UCAxIV Register

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

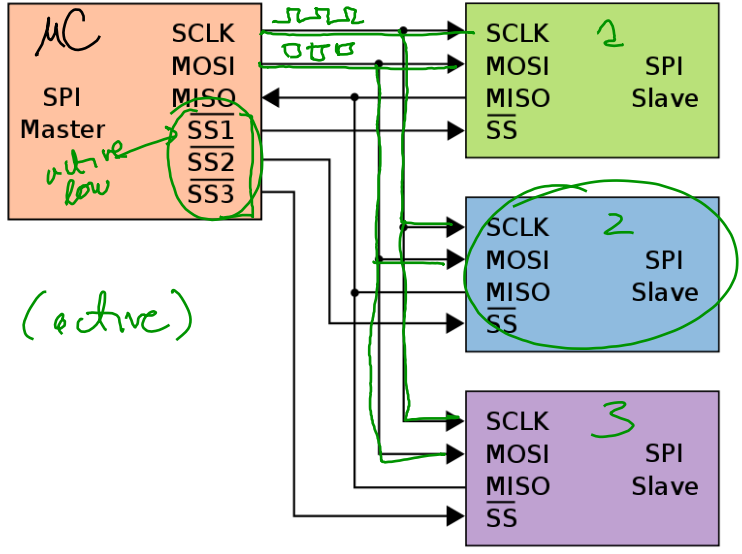
Table 37-13. UCAxIV Register Description

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

```
USCI_SPI_ISR
        ADD     &UCB0IV, PC  ; Add offset to jump table
        RETI                 ; Vector 0: No interrupt
        JMP     RXIFG_ISR    ; Vector 2: RXIFG
TXIFG_ISR                    ; Vector 4: TXIFG
        ...                  ; Task starts here
        RETI                 ; Return
RXIFG_ISR                    ; Vector 2
        ...                  ; Task starts here
        RETI                 ; Return
```

Multiple Slaves: Independent Configuration

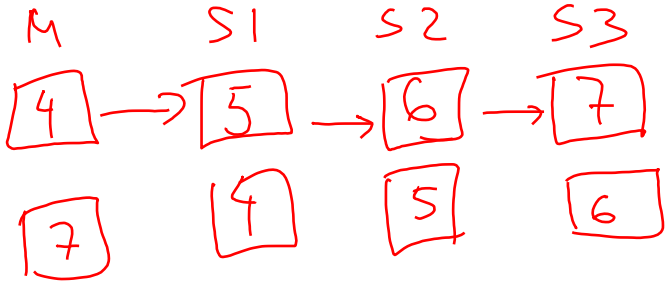
- SS for each S device
- One is active at a time

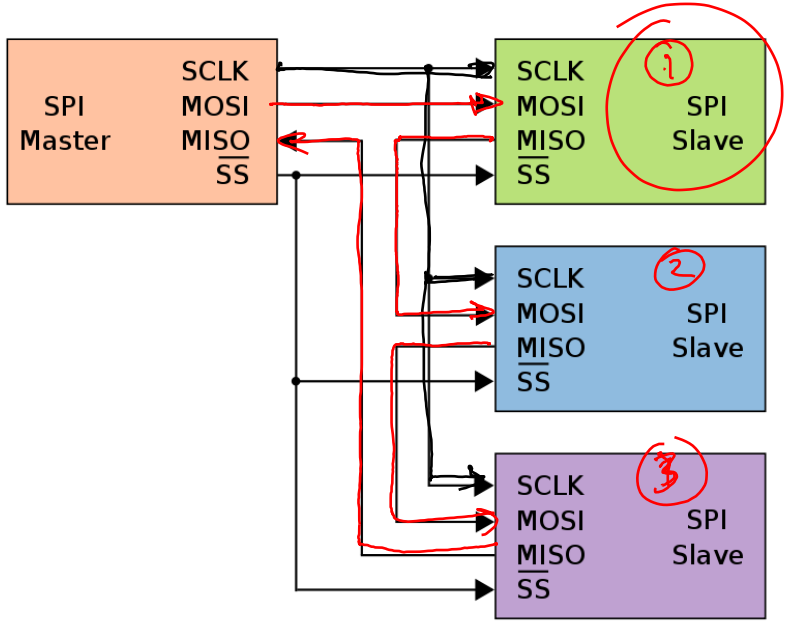


SPI bus

select slave 1 { $\overline{SS1} = 0$ (active), $\overline{SS2} = 1$, $\overline{SS3} = 1$

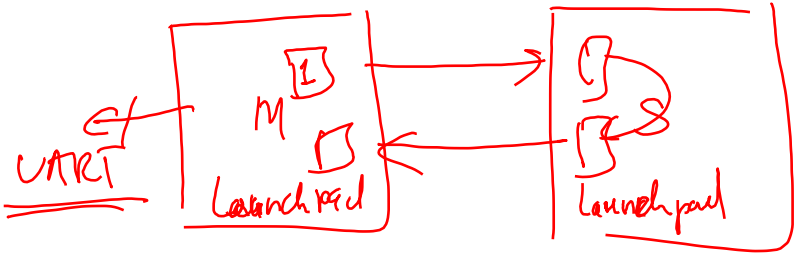
Multiple Slaves: Daisy-Chained SPI Bus





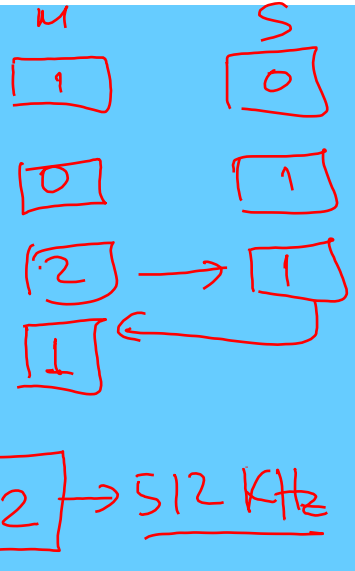
Problem: Setup SPI link between two boards

- Master: Send printable characters to slave over SPI link, verify the received character from the slave (should be equal to previously sent one), send the character to UART
- Slave: Echo a character received back
- Block diagram



Master Code

```
/*-------------------------------------------------------------------------------
 * File:         SPI_Master.c
 * Function:     Send printable characters to SPI-Slave device
 * Description:  This program sends one by one character to SPI-Slave device
 *               .. and receives data back from the slave device.
 *               .. If the expected data is received, LED1 is turned ON
 *               .. .. else LED1 is turned OFF
 *               All the characters received from the slave device are sent to UART
 *               .. using USCIA1 (no UART-USB connections required,
 *               .. works through JTAG)
 *
 * Instruction:  Set the following parameters in putty/hyperterminal
 * Port:         COMx
 * Baud rate:    115200
 * Data bits:    8
 * Parity:       None
 * Stop bits:    1
 * Flow Ctrl:    None
 *
 * ACLK = ~32.768kHz, MCLK = SMCLK = DCO ~ 1048kHz.  BRCLK = SMCLK/2
 *
```



Master Code

```
* Note:
*          1. PLEASE PROGRAM THE SLAVE DEVICE
*          2. CONNECT THE P1.2 WITH RESET PIN OF SLAVE DEVICE
*          3. MAKE OTHER CONNECTIONS BETWEEN MASTER AND SLAVE
*          4. PROGRAM THE MASTER DEVICE
*
*              MSP430F552x (Master)
*               -----------------
*           /|\|                 |
*            | |                 |
*            --|RST          P1.0|-> LED1
*              |                 |
*              |             P3.3|-> Data Out (UCA0SIMO)
*              |                 |
*              |             P3.4|<- Data In (UCA0SOMI)
*              |                 |
*Slave reset <-|P1.2         P2.7|-> Serial Clock Out (UCA0CLK)
*
*
* Code Written By:   Bhargavi Nisarga (TI)
* Modified By:       Prawar Poudel, prawar.poudel@uah.edu
*
*******************************************************************************/
```

Master: Configure SPI link

```
#include <msp430.h>

unsigned char MST_Data;
unsigned char temp;

void setup_master_SPI() {
    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**
    UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt
}
```

Master: Configure UART

```
// This UART does not need external connector (JTAG)
void setup_UART_application() {
    P4SEL |= BIT4 + BIT5;   // Set USCI_A1 RXD/TXD to receive/transmit data
    UCA1CTL1 |= UCSWRST;    // Set software reset during initialization
    UCA1CTL0 = 0;           // USCI_A1 control register
    UCA1CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA1BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA1BR1 = 0x00;         // upper byte
    UCA1MCTL |= UCBRS0;     // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA1CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
}
```

Master: Main

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

    P1DIR |= 0x05;                          // Set P1.0&2 to output direction
    P1OUT |= 0x04;                          // Set P1.0 for LED
                                            // Set P1.2 for slave reset

    setup_master_SPI();
    setup_UART_application();

    P1OUT &= ~0x04;                         // Now with SPI signals initialized
    __delay_cycles(100);
    P1OUT |= 0x04;                          // reset slave

    __delay_cycles(10000);                  // Wait for slave to initialize

    MST_Data = '0';                          // Initialize data values

    while (!(UCA0IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA0TXBUF = MST_Data;                   // Transmit first character

    __bis_SR_register(LPM0_bits + GIE);     // CPU off, enable interrupts
}
```

Master: USCI SPI ISR

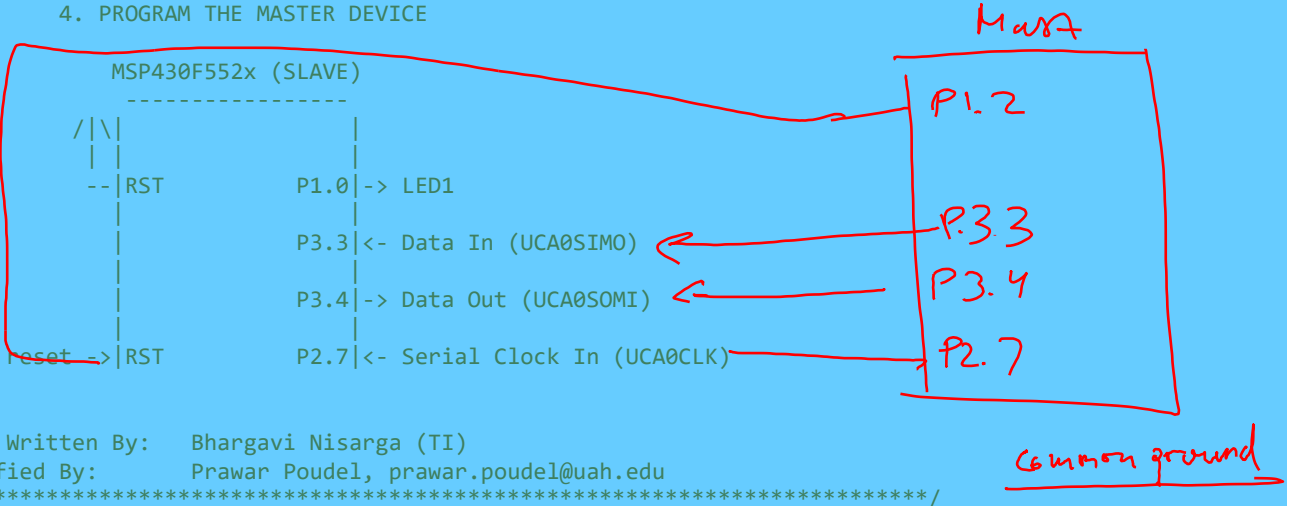
```
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCA0IV,4)) {
        case 0: break;                          // Vector 0 - no interrupt
        case 2:                                 // Vector 2 - RXIFG
            while (!(UCA0IFG&UCTXIFG));         // USCI_A0 TX buffer ready?
            temp = UCA0RXBUF;
            if (temp==MST_Data-1)               // Test for correct character RX'd
                                                // .. the correct value is data sent in prev cycle
                P1OUT |= 0x01;                  // If correct, light LED
            else
                P1OUT &= ~0x01;                 // If incorrect, clear LED
            MST_Data++;                         // Increment data to send nextcycle
            while(!(UCA1IFG&UCTXIFG));          // Wait until TXBUF is ready
            UCA1TXBUF = temp;                   // TXBUF <-- RXBUF
            if(MST_Data>126)
                MST_Data = 33;
            UCA0TXBUF = MST_Data;               // Send next value

            __delay_cycles(100000);             // Add time between transmissions to
                                                // make sure slave can process information
            break;
        case 4: break;                          // Vector 4 - TXIFG
        default: break;
    }
}
```

Slave: Header

```
/*****************************************************************************
 * File:        SPI_Slave.c
 * Function:    Receive a character from SPI master device
 * Description: This program receives a character to SPI master device
 *              .. and responds with the previously received character
 *              .. (delayed echo).
 *              USCI RX ISR is used to handle communication,
 *              .. CPU is in LPM4 normally.
 *              Prior to initial data exchange, master pulses
 *              .. slaves RST for complete reset.
 *
 * ACLK = ~32.768kHz, MCLK = SMCLK = DCO ~ 1048kHz.  BRCLK = SMCLK/2
 *
```

Slave: Header

```
* Note:
*          1. PLEASE PROGRAM THE SLAVE DEVICE
*          2. CONNECT THE P1.2 WITH RESET PIN OF SLAVE DEVICE
*          3. MAKE OTHER CONNECTIONS BETWEEN MASTER AND SLAVE
*          4. PROGRAM THE MASTER DEVICE
*
*                MSP430F552x (SLAVE)
*                 -----------------
*             /|\|                 |
*              | |                 |
*              --|RST          P1.0|-> LED1
*                |                 |
*                |             P3.3|<- Data In (UCA0SIMO)
*                |                 |
*                |             P3.4|-> Data Out (UCA0SOMI)
*                |                 |
*Slave reset ->|RST          P2.7|<- Serial Clock In (UCA0CLK)
*
*
* Code Written By:   Bhargavi Nisarga (TI)
* Modified By:       Prawar Poudel, prawar.poudel@uah.edu
*******************************************************************************/
```



Slave: Main

```
#include <msp430.h>

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

    P1DIR |= BIT0;                          // LED to indicate start of program
    P1OUT |= BIT0;                          // .. will toggle in every character receive

    while(!(P2IN&0x80));                    // If clock sig from mstr stays low,
                                            // it is not yet in SPI mode
    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**
    UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt

    __bis_SR_register(LPM4_bits + GIE);     // Enter LPM4, enable interrupts
}
```

Slave USCI ISR

```
// Echo character
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void) {
    switch(__even_in_range(UCA0IV,4)) {
        case 0:break;                           // Vector 0 - no interrupt
        case 2:                                 // Vector 2 - RXIFG
          while (!(UCA0IFG&UCTXIFG));           // USCI_A0 TX buffer ready?
          P1OUT ^= BIT0;                        // Toggle P1.0 for every character received
          UCA0TXBUF = UCA0RXBUF;                // .. this will be sent in next cycle of reception

          break;
        case 4:break;                           // Vector 4 - TXIFG
        default: break;
    }
}
```

echo