Austin Bumbalough
CPE 325-08
Lab 09
10/31/2019

# Lab 9 Solution

In this lab, I wrote a C program to configure the USCI peripheral of the MPS430FG4618 for SPI communication with the MSP430F2013. The MSP430FG4618 is the master device and the MSP430F2013 is the slave device. Sending a pause value between 1 and 100 will update the duty cycle of a square wave that drives LED3 connected to the slave device. Sending the number 0 will cause the slave device to respond with the current pause value. Sending 255 will turn the LED off. All other messages will have no effect.

## Master Device Source Code

```
void SPI_setup(void);

char SPI_getChar(void);

void SPI_putChar(char);

void SPI_getState(void);

void SPI_setState(char);

char UART_getChar(void);

void UART_getStr(char*, int);

void UART_putChar(char);

void UART_putStr(char*);

void UART_setup(void);
```

```c
char m1[] = "Beacon pause: ";

char uartRx[5];

char spiRx;

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;     // Stop watchdog timer
    unsigned char pause;

    UART_setup();
    SPI_setup();

    while(1) {
        UART_putStr(m1); // Send prompt
        UART_getStr(uartRx,5);
        if (!(strcmp(uartRx,"?"))) {
            SPI_getState();
        } else if (!(strcmp(uartRx,"-"))) {
            SPI_putChar(0xFF);
        } else {
            pause = (unsigned char)atoi(uartRx);
            if ((pause>0) & (pause<101)) {
                SPI_setState(pause);
            } else {
                UART_putStr("\r\nInvalid pause entered.");
            }
        }
        UART_putStr("\r\n");
    }
}

char SPI_getChar(void) {
    while(!(IFG2 & UCB0RXIFG)); // Wait until character is ready to be receive
    IFG2 &= ~UCB0RXIFG;
    UCB0TXBUF = 0x80;     // Dummy write to start SPI
    while (!(IFG2 & UCB0RXIFG));   // USCI_B0 TX buffer ready?
    return UCB0RXBUF;
}

void SPI_getState(void) {
    char pauseStr[20];
    SPI_putChar(0x00);
    for (int k=1000;k>0;k--); // Give slave time to load transmit register
    spiRx = SPI_getChar();
    sprintf(pauseStr,"\r\nCurrent pause: %u",spiRx);
    UART_putStr(pauseStr);
}

void SPI_putChar(char c) {
    while (!(IFG2 & UCB0TXIFG));  // Wait for previous character to transmit
    UCB0TXBUF = c;                // Put character into tx buffer
}

void SPI_setState(char state) {
    SPI_putChar(state);
}

void SPI_setup(void) {
    UCB0CTL0 = UCMSB + UCMST + UCSYNC;// Sync. mode, 3-pin SPI, Master mode, 8
    UCB0CTL1 = UCSSEL_2 + UCSWRST; // SMCLK and Software reset
    UCB0BR0 = 0x02;                // Data rate = SMCLK/2 ~= 500kHz
    UCB0BR1 = 0x00;
    P3SEL |= BIT1 + BIT2 + BIT3;   // P3.1,P3.2,P3.3 option select
    UCB0CTL1 &= ~UCSWRST;          // **Initialize USCI state machine**
}

char UART_getChar(void) {
    while (!(IFG2 & UCA0RXIFG)); // Wait until a character is ready to be reac
    IFG2 &= ~UCA0RXIFG;
    return UCA0RXBUF;
}

void UART_getStr(char* rxBuf, int limit) {
    char c;
    unsigned int i = 0;

    c = UART_getChar();
    while ((c != '\r') & (i < limit-1)) {
        rxBuf[i++] = c; // Store received character in receive buffer
        UART_putChar(c); // Echo character back
        c = UART_getChar(); // Get next character
    }
    rxBuf[i] = (char)0x00; // Terminate string with null character
}

void UART_putChar(char c) {
    while (!(IFG2 & UCA0TXIFG));  // Wait for previous character to transmit
    UCA0TXBUF = c;                // Put character into tx buffer
}

void UART_putStr(char* message) {
    int i;
    for(i = 0; message[i] != 0; i++) {
        UART_putChar(message[i]);
    }
}

void UART_setup(void) {
    P2SEL |= BIT4 + BIT5;          // Set UC0TXD and UC0RXD to transmit and re
    UCA0CTL1 |= UCSWRST;           // 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;               // remainder 10
    UCA0CTL1 &= ~UCSWRST;          // Clear software reset
}
```

## Slave Device Source Code

```
void initLED(void);

void SPI_setup(void);

void sysInit(void);

unsigned char pause;

unsigned char spiRx;

unsigned char spiTx;

void main(void) {
    sysInit();
    SPI_setup();                  // Setup USI module in SPI mode
    while(1) {
        _BIS_SR(LPM0_bits + GIE); // LPM0 w/ Interrupt
        if (spiRx == 0) {
            USISRL = pause;
        } else if (spiRx < 101) {
            pause = spiRx;
        } else if (spiRx == 0xff) {
            P1DIR &= ~BIT0; // Turn LED3 off
        }
    }
}

void initLED(void) {
    P1DIR |= BIT0;                // P1.0 as output - LED3
    P1OUT |= BIT0; // Initialize LED3 on
}

void SPI_setup(void) {
    USICTL0 |= USISWRST;          // Set UCSWRST -- needed for re-configuratio
    USICTL0 |= USIPE5 + USIPE6 + USIPE7 + USIOE; // SCLK-SDO-SDI port enable,M
    USICTL1 = USIIE;              // USI Counter Interrupt enable
    USICTL0 &= ~USISWRST;         // **Initialize USCI state machine**
    USICNT = 8;                   // Load bit counter, clears IFG
}

void sysInit(void) {
    WDTCTL = WDT_MDLY_32;     // 32 ms interval mode
    BCSCTL1 = CALBC1_1MHZ;        // Set DCO
    DCOCTL = CALDCO_1MHZ;
}

#pragma vector = USI_VECTOR
__interrupt void USI_ISR(void) {
    P1DIR |= BIT0;
    spiRx = USISRL;           // Read new command
    USICNT = 8;                   // Load bit counter for next TX
    _BIC_SR_IRQ(LPM0_bits);       // Exit from LPM0 on RETI
}

#pragma vector = WDT_VECTOR
__interrupt void WDT_ISR(void) {
    static int i = 0;
    if (i == 1) {
        P1OUT &= ~BIT0;
    } else if (i == pause+1) {
        P1OUT |= BIT0;
        i = 0;
    } else i++;
    _BIS_SR(LPM0_bits + GIE);
}
```