CPE 325: Intro to Embedded Computer System

Lab 09

Synchronous Serial Communications

Submitted by: Anshika Sinha & Charles Marmann

Date of Experiment: 03/05/25

Report Deadline: 03/23/25

Demonstration Deadline: 03/24/25

Theory

Topic 1: SPI vs UART

a) SPI (Serial Peripheral Interface) - SPI is a serial communication protocol that uses a clock signal for data transmission. It requires four wires: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select/Chip Select). Since it uses a clock signal, SPI operates at higher speeds compared to UART. It is commonly used for high-speed peripherals, such as SD cards, sensors, and display modules. Multiple slave devices can be connected to a single master device using CS (Chip Select) lines for each device.

b) UART (Universal Asynchronous Receiver-Transmitter) - UART is an asynchronous serial communication protocol, which does not require a clock signal. It uses two wires: TX (Transmit) and RX (Receive) for transmitting and receiving data. The devices must be set to the same baud rate for communication to work correctly. It is designed for point-to-point communication is used for simple and long-distance data transmission

Topic 2: Serial Communication Types

a) The MSP430F5529 supports multiple serial communication protocols including SPI, UART, and I2C.
b) SPI (Serial Peripheral Interface): A synchronous serial communication type that supports interfacing with external sensors, ADCs, DACs, and SD cards. It requires a clock signal, and uses four main signals: MOSI (Master Out Slave In), MISO (Master In Slave Out), SCLK (Serial Clock), and SS/CS (Slave Select/Chip Select). Serial data is transmitted and received by multiple devices that use a shared clock provided by the master. It supports full-duplex communication, transmitting data simultaneously in both directions.
c) UART (Universal Asynchronous Receiver-Transmitter): UART allows for asynchronous serial communication for communicating with PCs, Bluetooth modules, GPS modules, etc. It used start and stop bits to frame data, no clock signal is required. Both devices are required to have the same baud rate. UCAxTXBUF and UCAxRXBUF are the transmit and receive buffers respectively, supporting full-duplex communication.
d) I2C (Inter-Integrated Circuit): I2C is a widely implemented synchronous serial communication protocol used to interface with external sensors(temperature, accelerometer, etc.), EEPROMs, and other peripherals. It supports multi-master and multi-slave configuration and uses only two wires: SDA (Serial Data) and SCL (Serial Clock).

Results & Observation

Master:

Program Description:

The master program is a C program that interfaces with a serial monitor, slave MSP430, and LED2. It is a modified version of Lab09_D1_Master_5529.c that was provided to us. The modified code had the added functionality of being able to request a frequency and cycles from the user, which it would then use to blink LED2 as well as send a character over SPI to the slave MSP 430 to toggle its LED2. The result is that the user can set the master's and slave's LED2 to blink at a specific frequency for a certain number of cycles. The main changes that were made were the addition of UART communication so that the master could interface with a serial monitor and the addition of a timer using the watchdog timer in order to toggle LED2 at a specific frequency.

Slave (same as demo):

Program Description:

The slslave program is the same program that was provided in the lab manual. It works by initializing the SPI connection as a slave and then waits until it receives a character. Once it receives a character, it toggles LED2 and echos the character back.

Program Output:

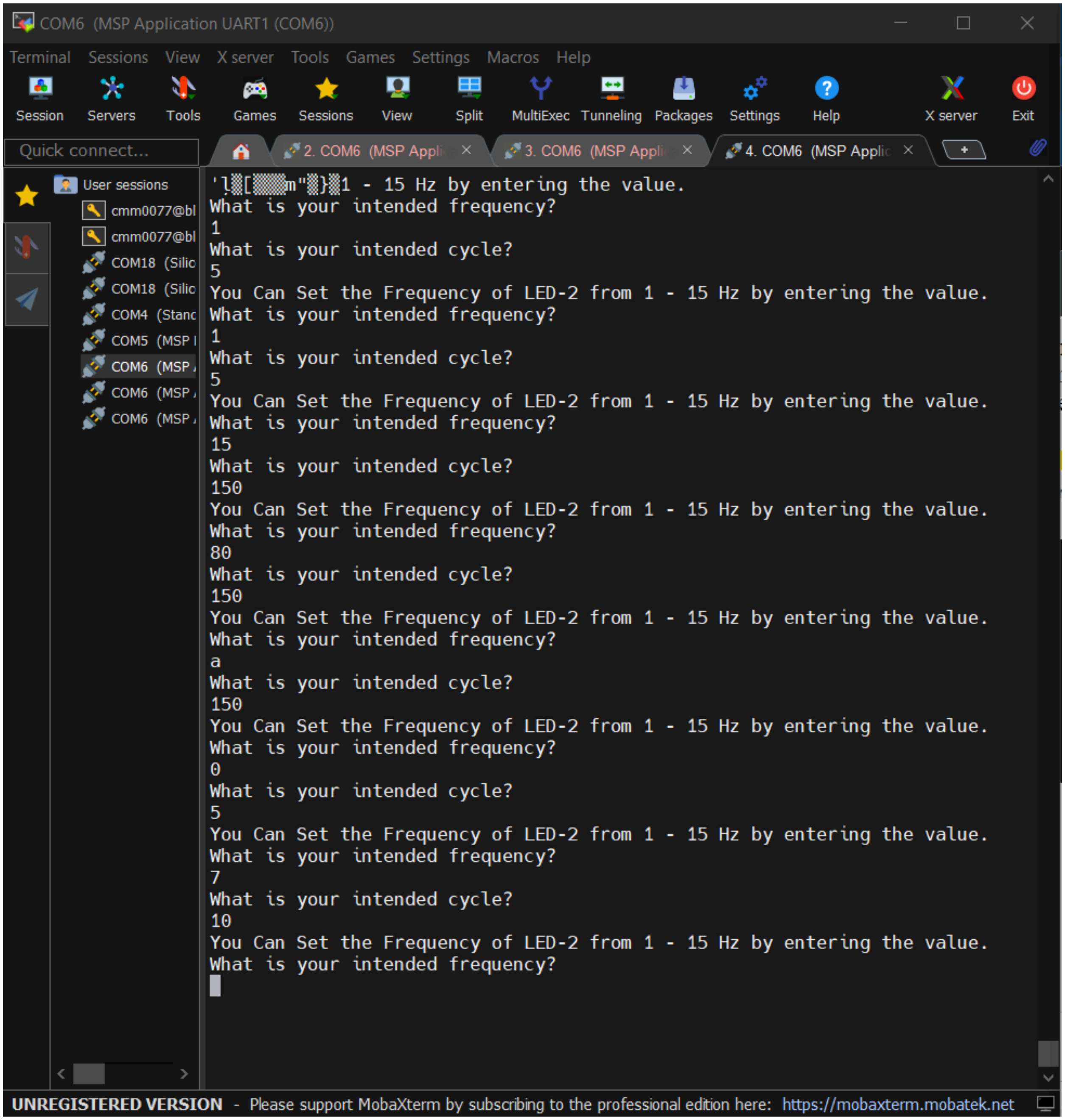
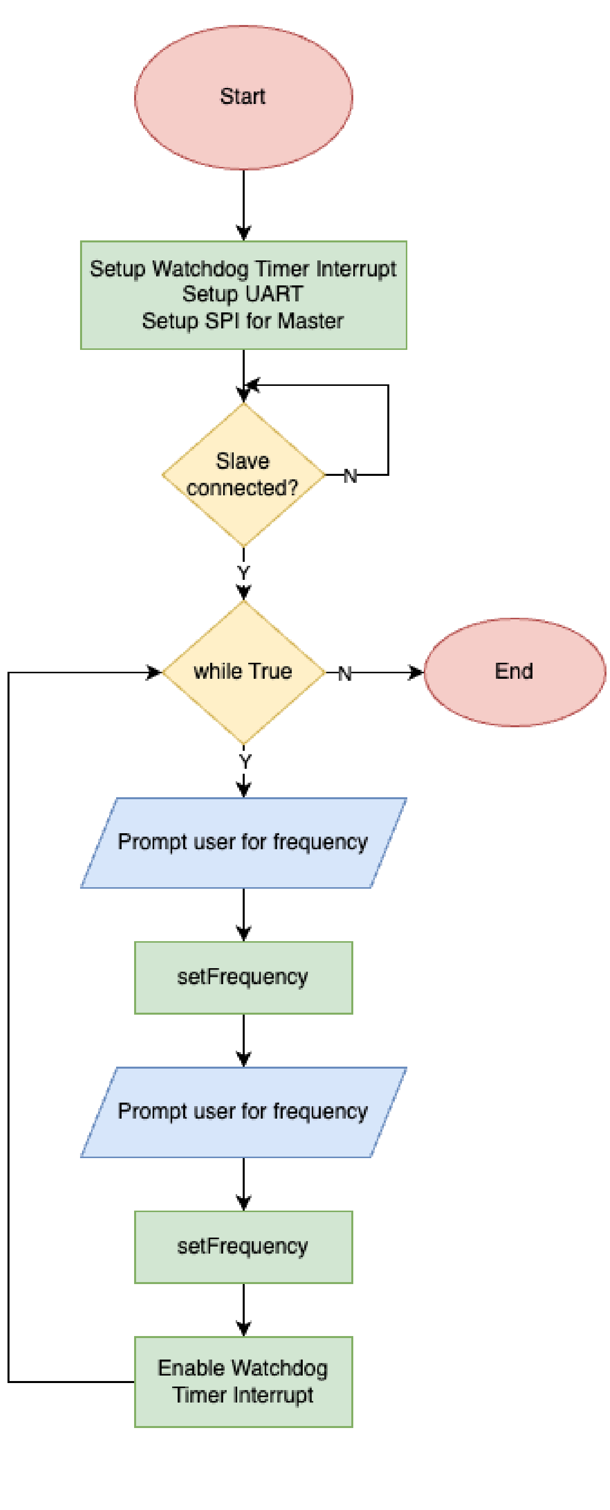


Figure 01: Program output

Program Flowchart:



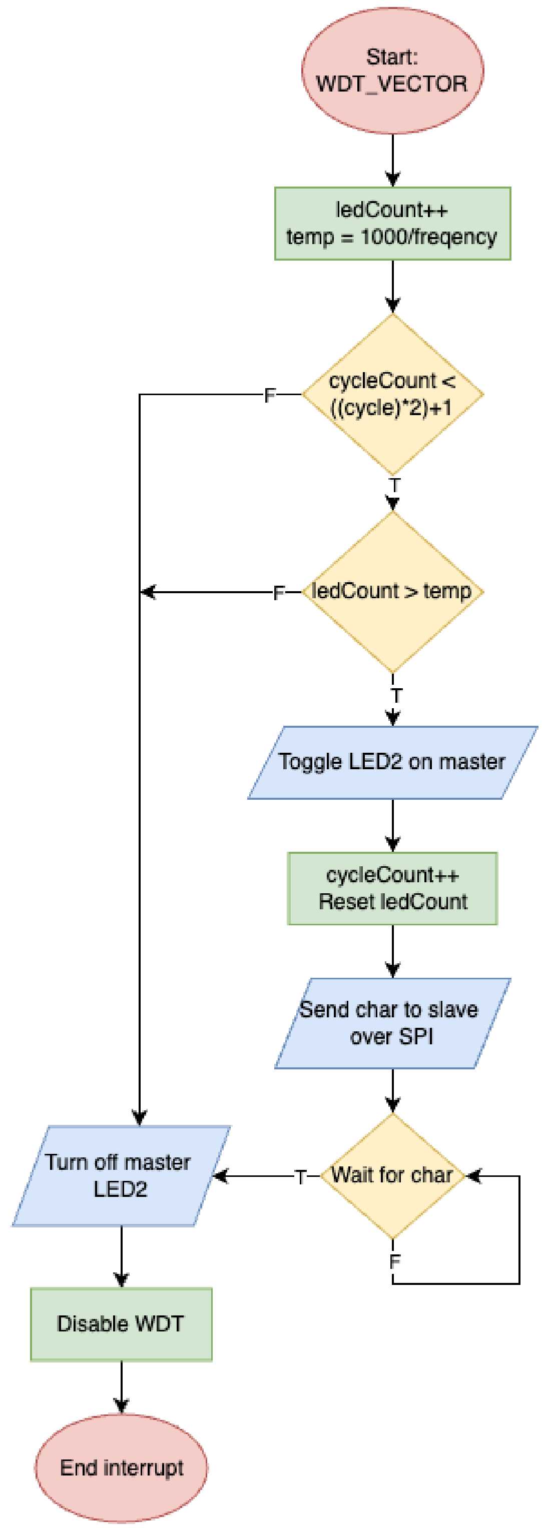


Figure 02: Program Master Flowchart

Appendix

Table 01: Program Master source code

```
/*--------------------------------------------------------------------------
 * File: main.c
 * Description: Prompts the user to input a frequency value and cycle value
 *              It then uses these values to blink LED2 at that frequency
 *              for that many cycles. In addition, each time it toggles LED2
 *              a character is sent over SPI to the slave MSP430 causing
 *              LED2 to toggle on the slave. This results in LED2 blinking
 *              at the same time on the master and slave board.
 *
 * Board: 5529
 * Input: UART serial monitor
 * Output: UART serial input, SPI output to slave MSP430, LED2
 * Author(s): Charles Marmann, Anshika Sinha
 * Date: March 23, 2025

*---------------------------------------------------------------------------*
/
#include <msp430.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned char MST_Data,SLV_Data;
char askFreq[] = "You Can Set the Frequency of LED-2 from 1 - 15 Hz by
entering the value.\n\rWhat is your intended frequency?\n\r"; //
char askCycle[] = "What is your intended cycle?\n\r"; //
char strIn[10];     // Holds the value of the read string
int frequency = 1;  // Frequency that the LEDs should blink at
int cycle = 0;      // Number of cycles that the LEDs should blink
int cycleCount = 1; // Number of cycles completed
int ledCount = 0;   // Iterator for the WDT ISR

void UART_setup(void) {
    //P3SEL |= BIT3 | BIT4;       // Set UC0TXD and UC0RXD to transmit and
receive data
    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 = 0x02;        // Modulation (UCBRS0=0x01, UCOS16=0)
    UCA1CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state
machine
}

void SPI_Master_UCB0_Setup(void) {
    P3SEL |= BIT0+BIT1+BIT2;                       // P3.0,1,2 option select
for SIMO,SOMI and CLK
    UCB0CTL1 |= UCSWRST;                      // **Put state machine in
reset**
```

```
    UCB0CTL0 |= UCMST+UCSYNC+UCCKPL+UCMSB;    // 3-pin, 8-bit SPI master
    UCB0CTL1 |= UCSSEL_2;                     // SMCLK
    UCB0BR0 = 18;
    UCB0BR1 = 0;                              //
    UCB0CTL1 &= ~UCSWRST;                     // **Initialize USCI state
machine**
}

void WDT_setup()
{
    WDTCTL = WDT_MDLY_0_5;      // Set WDT to all an interrupt every 0.5 ms
}

void SPI_sendChar(char bip)
{
    while (!(UCB0IFG&UCTXIFG)); // USCI_A0 TX buffer ready?
    UCB0TXBUF = bip; // Transmit first character
}

void UART_sendChar(char bip)
{
    while (!(UCA1IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA1TXBUF = bip;                  // TX -> RXed character
}

//
//  UART code from lab 8
//

char UART_getChar()
{
    while (!(UCA1IFG&UCRXIFG));
    if (UCA1RXBUF == '\r')
    {
        UART_sendChar('\r');
        UART_sendChar('\n');
    }
    else
    {
        UART_sendChar(UCA1RXBUF);
    }

    return UCA1RXBUF;
}

void UART_sendString(char* str)
{
    int i = 0;
    while(str[i] != '\0')
    {
        UART_sendChar(str[i]);
        i++;
    }
}

void UART_getLine(char* buf, int limit)
{
```

```
    int i = 0;
    while (limit != 0)
    {
        char temp = UART_getChar();
        if (temp == '\r')
        {
            break;
        }
        buf[i] = temp;
        i++;
        limit--;
    }
    // Add null character at the end
    buf[i] = '\0';
}

//
// Restrict both frequency and cycle
//

void setFreq(int input)
{
    if (input > 15)
    {
        frequency = 15;
    }
    else if (input < 1)
    {
        frequency = 1;
    }
    else
    {
        frequency = input;
    }
}

void setCycle(int input)
{
    if (input < 0)
    {
        cycle = 0;
    }
    else
    {
        cycle = input;
    }
    cycleCount = 1;
}

int main() {
    WDT_setup();
    UART_setup();                // Initialize USCI_A0 module in UART mode
    SPI_Master_UCB0_Setup();

    // Setup LEDs
    P1OUT = 0;
    P1DIR |= BIT0;
```

```
    P4DIR |= BIT7;
    P4OUT &= ~BIT7;
    P1IN &= ~BIT2;

    // Initialize interrupts
    _EINT();

    P1DIR |= BIT0;                // Set P1.0 to output direction

    while (!(P1IN&BIT2)); // wait for the slave to connect

    while(1)
    {
        UART_sendString(askFreq);   // Ask for frequency
        UART_getLine(strIn,10);     // Receive frequency
        setFreq(atoi(strIn));       // Convert to int

        UART_sendString(askCycle);  // Ask for cycle
        UART_getLine(strIn,10);     // Receive cycles
        setCycle(atoi(strIn));      // Convert to int, reset cycleCount

        SFRIE1 |= WDTIE;            // Enable WDT interrupt

    }
}

#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    ledCount++;                         // Increment ledCount
    int temp = 1000/frequency;          // 2 * 1000 * 0.5 ms = 1s

    if (cycleCount < ((cycle)*2)+1)     // If the number of cycles ran is less
than the desired ammount of cycles
    {

        if (ledCount > temp) {
            P4OUT ^= BIT7;              // Toggle master LED
            cycleCount++;               // increment cycleCount
            ledCount = 0;               // Reset ledCount
            SPI_sendChar('A');          // Send a character to toggle the
slave LED
            while(!(UCB0IFG&UCRXIFG));  // Wait for a character to come back
        }
    }
    else                                // All cycles completed
    {
        P4OUT &= ~BIT7;                 // Turn off master LED
        SFRIE1 &= ~WDTIE;               // Disable WDT ISR
    }


}
```

Table 02: Program Slave source code (same as demo)

```
//***************************************************************************
//   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.
//   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               |
//                |                   |
//                |                   |
//                |                   |
//                |                   |
//                |               P3.0|<- Data In (UCA0SIMO)
//                |                   |
//                |               P3.1|-> Data Out (UCA0SOMI)
//                |               P3.2|-> Serial Clock Out (UCB0CLK)
//
//   Author: A. Milenkovic, milenkovic@computer.org
//
//   Date: October 2022
//***************************************************************************#i
nclude <msp430.h>

void SPI_Slave_UCB0_Setup(void) {
   P3SEL |= BIT0+BIT1+BIT2; // P3.3,4 option select
   //P2SEL |= BIT7; // P2.7 option select
   UCB0CTL1 |= UCSWRST; // **Put state machine in reset**
   UCB0CTL0 |= UCSYNC+UCCKPL+UCMSB; // 3-pin, 8-bit SPI slave,
   // Clock polarity high, MSB
   UCB0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**
```

```
}
int main(void) {
   WDTCTL = WDTPW+WDTHOLD;

   SPI_Slave_UCB0_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(!(UCB0IFG&UCRXIFG));  // wait for a new character
       while(!(UCB0IFG&UCTXIFG));  // new character is received, is TXBUF
ready?
       UCB0TXBUF = UCB0RXBUF;      // echo character back if ready
       P4OUT ^= BIT7;
   }
}
```

Make sure to submit a single pdf file only