CPE 325: Intro to Embedded Computer System

Lab 8

Synchronous Communication, UART Communication with DMA

Submitted by: Hannah Kelley

Date of Experiment: October 27, 2025

Report Deadline: November 3, 2025

Demonstration Deadline: November 3, 2025

Theory

Topic 1: SPI vs UART

Serial Peripheral Interface (SPI) and Universal Asynchronous Receiver/Transmitter (UART) are both serial communication protocols. SPI uses separate data and clock lines—specifically the Master In Slave Out (MISO), Master Out Slave In (MOSI), and Serial Clock (SCLK)—to communicate using high-speed, full-duplex communication between two or more devices. In contrast, UART is an asynchronous protocol that uses two lines—the transfer (TX) and receive (RX)—without a shared clock signal. Instead of a clock signal both devices must be configured to use the same baud rate to ensure correct timing. Overall, SPI is generally faster and provides better synchronization while UART is simpler and uses less hardware.

Topic 2: Serial Communication Types

Serial communication is a method of transmitting data one bit at a time over a single channel or wire pair. It has two main types, synchronous and asynchronous communication. In synchronous communication data is transferred using a shared clock signal. This shared clock allows for faster and more reliable communication between connected devices. SPI and I2C are examples of serial synchronous communication. In asynchronous communication there is no shared clock. Instead, devices are configured to use the same baud rate and start and stop bits are added to the data to ensure each data frame is sent and received in the correct order. An example of serial asynchronous communication is UART. Additionally, serial communication can be classified as half-duplex or full-duplex. Both types allow for the transmission of data in both directions. Full-duplex allows for the transmission of data in both directions simultaneously while half-duplex can only transmit data in one direction at a time.

Results & Observation

Program 1:

Program Description:

For this program, two MSP430F5529 LaunchPad boards were connected and configured to communicate via SPI using the UCB0 module. Parallel ports P3.0, P3.1, and P3.2 were used for the SIMO, SOMI, and SCLK lines, respectively, while a handshaking signal was implemented on P1.2. The slave module code was retained from the provided demonstration, while the master code was modified using elements from both the previous lab and the master demo to interface with the user through UART. After receiving user input for the desired LED blinking frequency and number of cycles, the master and slave modules initiated data transmission and synchronized LED blinking to indicate successful data exchange.

Program Output:

```
You can set the frequency of LED-2 from 1 - 10 Hz by entering the value.
                                                    What is your intended frequency?
                                                                                    1
What is your intended cycle?
                            10
```

Figure 01: Program 1 Terminal Output

Appendix

Table 01: Program 1 source code

```
/*-------------------------------------------------------------------------
 * File:       Lab08_Master.c
 * Description: send data between 2 lab boards using UART for
 *  user interface and SPI for communication between boards
 * Board:      5529
 * Input:       user defined frequency and cycles
 * Output:     LEDs and UART terminal (Moba)
 * Author:     Hannah Kelley, Margaret Roberts
 * Date:       October 27, 2025
 *-------------------------------------------------------------------------*/
#include <msp430.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>


unsigned char MST_Data, SLV_Data;
unsigned char temp;
unsigned int cycles;
unsigned int freq;
unsigned long int delay;
char response[3];

void SPI_Master_UCB0_Setup(void);
void UART_Setup(void);
void UART_sendChar(char bip);
char UART_getChar();
void UART_sendString(char* str);
void UART_getLine(char* buf, int limit);
void UART_getUserInput(void);
void trim(char* s);

int main(void) {
   WDTCTL = WDTPW+WDTHOLD;
   P1OUT = 0;
   P1DIR |= BIT0;
   P4DIR |= BIT7;
   P4OUT &= ~BIT7;
   P1IN &= ~BIT2;
   SPI_Master_UCB0_Setup();

   UART_Setup();

   while(1) {
```

```
    UART_getUserInput();

    // delay calcs
    delay = 50000 / (2 * freq);
    printf("%lu", delay);

    // Wait for Slave
    while (!(P1IN&BIT2)); // Wait until Slave is ready
      P1OUT |= BIT0; // If ready, light LED1
      //__delay_cycles(delay);
      int j;
      for (j = delay; j > 0; j--);
      P1OUT &= ~BIT0;
      MST_Data = 0x01; // Initialize data values
      SLV_Data = 0x00;
      int i;
      for (i = 2 * cycles; i > 0; i--) {
        while (!(UCB0IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
        UCB0TXBUF = MST_Data; // Transmit first character
        while(!(UCB0IFG&UCRXIFG)); // Wait for data back
        if (UCB0RXBUF==SLV_Data){ // Test for correct character RX'd
          P1OUT |= BIT0; // If correct, light LED1
        }
        else {
          P1OUT &= ~BIT0; // If incorrect, turn off LED1
        }
        P4OUT ^= BIT7; // heart bit on LED2
        MST_Data = (MST_Data + 1) % 50;
        SLV_Data = (SLV_Data + 1) % 50;
        for (j = delay * 2; j > 0; j--);
      }
    }
}
// set up UART for user interface
void UART_Setup(void) {
    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
}

// set up SPI
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**
}

// send a char via UART
void UART_sendChar(char bip) {
   while (!(UCA1IFG & UCTXIFG)); // wait for TX buffer to be ready
   UCA1TXBUF = bip;
}

// retrieve a char via UART
char UART_getChar() {
   while (!(UCA1IFG & UCRXIFG));  // Wait for RX buffer to be ready
   return UCA1RXBUF;
}

// send a string via UART
void UART_sendString(char* str) {
   while(*str) {
      UART_sendChar(*str++);
   }
}

// receive a line of user input via UART (w/ backspace handling)
void UART_getLine(char* buf, int limit) {
   int i = 0;
   char ch;

   while (1) {
      ch = UART_getChar();

      if ((ch == '\r') || (ch == '\n')) { // enter key pressed
         buf[i] = '\0'; // null-terminate str
         UART_sendString("\r\n");
         break;
      } else if (ch == '\b' || ch == 127) { // backspace pressed
         if (i > 0) {
            i--; // move buffer ptr back
            UART_sendString("\b \b"); // clear char on screen
         }
      } else if ( i < limit - 1) { // normal char input
         buf[i++] = ch;
         UART_sendChar(ch); // echo print char
      }
```

```
    }
}

// get user inputs from Putty/Moba
void UART_getUserInput(void) {
    UART_sendString("You can set the frequency of LED-2 from 1 - 10 Hz by entering the value. \nWhat
is your intended frequency?\n");
    UART_getLine(response, 3);
    printf("%s", response);
    //trim(response);
    freq = atoi(response);
    UART_sendString("What is your intended cycle?\n");
    UART_getLine(response, 10);
    printf("%s", response);
    //trim(response);
    cycles = atoi(response);
}

void trim(char *s) {
    char *p = s;
    char *end;

    // Skip leading whitespace
    while (isspace((unsigned char)*p)) p++;

    // Shift the string back to the beginning
    if (p != s)
        memmove(s, p, strlen(p) + 1);

    // Now remove trailing whitespace
    end = s + strlen(s) - 1;
    while (end >= s && isspace((unsigned char)*end)) end--;
    *(end + 1) = '\0';
}
```