CPE 325: Embedded Systems Laboratory

Lab09

Synchronous Serial Communications

Submitted by: Esther Shore

Date of Experiment: July 12, 2023

Report Deadline: July 19, 2023

Demonstration Deadline: July 24, 2023

**Introduction**

This lab introduces the SPI communication protocol to control a slave device from a master device. It also utilizes UART to accept input through MobaXterm. The master program will accept a range of multiplier values to send to the slave device and change the frequency of the blinking LED. Additionally, it can retrieve the current multiplier value or set the LED on without blinking when certain inputs are entered.

**Theory Topics**

1. SPI vs UART

Serial peripheral interface (SPI) and universal asynchronous receiver/transmitter (UART) are two serial communication methods that are commonly used for connecting peripherals for embedded systems. Since they are both serial methods, that means they transmit data bit by bit. SPI is synchronous, following a clock, whereas UART is asynchronous, using bits to indicate start and stop points. Following the SPI protocol, a master device controls a slave device or multiple slave devices. Following the UART protocol, two devices communicate back and forth with each other at a specified baud rate. On one hand, SPI requires four wires (SCLK, MOSI, MISO, and SS/CS) and UART only requires two (TX and RX).

1. DMA controller

A direct memory access (DMA) controller is a peripheral device that enables the transfer of data between memory and other peripherals directly. This method offloads the CPU to do other operations, resulting in more efficient data transfer than alternative methods.

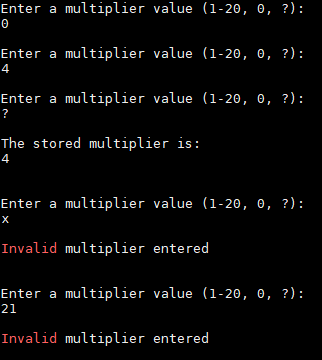
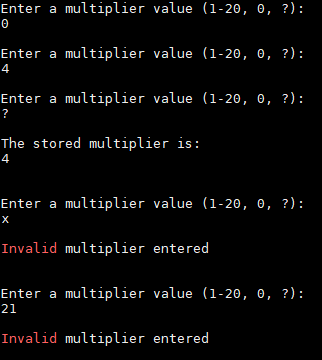
**Program 1 (‘FG4618 Master Program)**

***Program Description:***

This is the master program, which accepts a multiplier value via UART and sends valid inputs via SPI to the slave. These inputs are then interpreted by the slave program to alter the frequency of the blinking LED. When a question mark is entered, the stored multiplier value will be received.

***Program Output:***

Screenshots of valid, invalid and ? inputs



**Program 2 (‘F2013 Slave Program)**

***Program Description:***

This is the slave program, which blinks LED3 at a frequency dependent on the given multiplier (1-20) sent through SPI from the master to the slave (0.25Hz \* multiplier). When the program receives 0, LED3 is turned on continuously and when the program receives 255, it replies with the current value of the multiplier.

Calculations:

wdt\_interval = 0.5 ms

target\_freq = multiplier / 4

target\_period = 1 / target\_freq \* 1000

wdt\_cycles = round(target\_period / 2 / wdt\_interval) (delay between toggles is half a period)

**Conclusion**

This lab exposed a lot about how the different wired communication protocols work and how they can work together to accomplish a goal. There was a steep learning curve to it, but the demo code helped a lot. Switching between frequencies in the slave program took a bit of time because of troubles with division operations to dynamically calculate. First, a massive case statement was attempted but that seemed to add a stall between switching multipliers, so finally figured out that accessing the corresponding element of an array with all the precalculated cycle counts worked. Definitely not the most elegant solution but it works.

***Appendix:***

**Table 1:** Program 1 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab09\_master.c  \* Function: SPI Interface (MPS430Fg4618)  \* Input: Character 0-20 and ? from MobaXterm UART  \* Output: Send integer state to SPI slave  \*----------------------------------------------------------------------------\*/  #include "msp430xG46x.h"  #include <stdio.h>  #include <ctype.h>  #include <stdlib.h>  #include <string.h>  #define LED\_CUR\_STATE 0xFF // Character NULL - used for dummy write operation  char greetMessage1[] = "Enter a multiplier value (1-20, 0, ?): ";  char greetMessage2[] = "Invalid multiplier entered";  void SPI\_setup(void) {  UCB0CTL0 = UCMSB + UCMST + UCSYNC;// Sync. mode, 3-pin SPI, Master mode, 8-bit data  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\*\*  }  unsigned char SPI\_getState(void) {  while((P3IN & 0x01)); // Verifies busy flag  IFG2 &= ~UCB0RXIFG;  UCB0TXBUF = LED\_CUR\_STATE; // Dummy write to start SPI  while (!(IFG2 & UCB0RXIFG)); // USCI\_B0 TX buffer ready?  return UCB0RXBUF;  }  void SPI\_setState(unsigned char State) {  while(P3IN & 0x01); // Verifies busy flag  IFG2 &= ~UCB0RXIFG;  UCB0TXBUF = State; // Write new state  while (!(IFG2 & UCB0RXIFG)); // USCI\_B0 TX buffer ready?  }  void UART\_putCharacter(char c) {  while (!(IFG2 & UCA0TXIFG)); // Wait for previous character to transmit  UCA0TXBUF = c; // Put character into tx buffer  }  void UART\_sendMessage(char\* message) {  int i;  for(i = 0; message[i] != 0; i++) {  UART\_putCharacter(message[i]);  }  UART\_putCharacter('\n'); // Newline  UART\_putCharacter('\r'); // Carriage return  }  void UART\_getWord(char\* buffer, int limit)  {  int i = 0;  char received\_char;  while (i < limit) {  while (!(IFG2 & UCA0RXIFG)); // USCI\_A0 RX buffer ready?  received\_char = UCA0RXBUF; // RX -> Get received character  UART\_putCharacter(received\_char);  if (received\_char == '\r') {  break;  }  buffer[i] = received\_char;  i++;  }  buffer[i] = '\0'; // Terminate the string with null character  }  void UART\_setup(void) {  P2SEL |= BIT4 + BIT5; // Set UC0TXD and UC0RXD to transmit and receive data  UCA0CTL1 |= UCSWRST; // Software reset  UCA0CTL1 |= UCSSEL\_2; // Clock source SMCLK - 1048576 Hz  UCA0BR0 = 18; // Baud rate - 1048576 Hz / 57600  UCA0BR1 = 0;  UCA0MCTL = UCBRS\_1 + UCBRF\_0; // Modulation  UCA0CTL1 &= ~UCSWRST; // Software reset  }  void main(void) {  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer  UART\_setup(); // Setup USCI\_A0 module in UART mode  SPI\_setup(); // Setup USCI\_B0 module in SPI mode  \_EINT(); // Enable global interrupts  int z;  char input[3];  char multiplier[3];  for(z = 100; z > 0; z--); // Delay to allow baud rate stabilize    while(1) {  UART\_sendMessage(greetMessage1);  UART\_getWord(input, 3);  UART\_sendMessage("\r\n");  if (strcmp(input, "?") == 0) {  UART\_sendMessage("The stored multiplier is:");  snprintf(multiplier, 3, "%u", SPI\_getState());  UART\_sendMessage(multiplier);  UART\_sendMessage("\r\n");  } else if (strcmp(input, "0") == 0) {  int target = atoi(input);  SPI\_setState(target);  } else {  int target = atoi(input);  if (target > 0 && target <= 20) {  SPI\_setState(target);  } else {  UART\_sendMessage(greetMessage2);  UART\_sendMessage("\r\n");  }  }  }  } |
| --- |

**Table 2:** Program 2 Source Code

| /\*------------------------------------------------------------------------------  \* File: Lab09\_slave.c  \* Function: SPI Interface (MPS430F2013)  \* Input: Character 0-20, 255 from master  \* Output: Alter LED3 blink frequency (0.25Hz \* m) or return current state  \*------------------------------------------------------------------------------\*/  #include "msp430x20x3.h"  #include <math.h>  #define SET\_BUSY\_FLAG() P1OUT |= 0x10  #define RESET\_BUSY\_FLAG() P1OUT &= ~0x10  unsigned char LEDState = 4; // set default multiplier  unsigned char NextState = 4;  unsigned int wdt\_cycles, blink;  void SPI\_setup(void) {  USICTL0 |= USISWRST; // Set UCSWRST -- needed for re-configuration process  USICTL0 |= USIPE5 + USIPE6 + USIPE7 + USIOE; // SCLK-SDO-SDI port enable,MSB first  USICTL1 = USIIE; // USI Counter Interrupt enable  USICTL0 &= ~USISWRST; // \*\*Initialize USCI state machine\*\*  }  void SPI\_initComm(void) {  USICNT = 8; // Load bit counter, clears IFG  USISRL = LEDState; // Set LED state  RESET\_BUSY\_FLAG(); // Reset busy flag  }  void systemInitialize() {  BCSCTL1 = CALBC1\_1MHZ; // Set DCO  DCOCTL = CALDCO\_1MHZ;  }  void main(void) {  systemInitialize();  SPI\_setup(); // Setup USI module in SPI mode  SPI\_initComm(); // Initialization communication  WDTCTL = WDT\_MDLY\_0\_5; // use watchdog interval 0.5ms  P1DIR |= BIT0; // set LED3 as output  P1DIR |= BIT4; // P1.4 as output - Busy flag  IE1 |= WDTIE;  blink = 1; // default 1 Hz blink  LEDState = 4;  NextState = 4;  wdt\_cycles = 1000;  while(1) {  \_BIS\_SR(LPM0\_bits + GIE );// Enter LPM0 with interrupt  switch (NextState) {  case 0xFF: // Dummy operation; no change in the state  break;  default:  LEDState = NextState; // New state  break;  }  unsigned int wdt\_cycles\_array[] = {  4000, 2000, 1333, 1000, 800, 667, 571, 500, 444, 400,  364, 333, 308, 286, 267, 250, 235, 222, 210, 200  }; // see lab report for calculation method  // Change the wdt\_cycles depending on the command  if (LEDState > 0 && LEDState <= 20) {  blink = 1; // blink true  unsigned int array\_index = LEDState - 1;  wdt\_cycles = wdt\_cycles\_array[array\_index]; // get corresponding cycles  } else if (LEDState == 0) {  P1OUT |= BIT0; // turn ON LED3  blink = 0; // blink false  }  USISRL = LEDState; // Prepares reply to master with new state  RESET\_BUSY\_FLAG(); // Clears busy flag - ready for new communication  }  }  #pragma vector = USI\_VECTOR  \_\_interrupt void USI\_ISR(void) {  SET\_BUSY\_FLAG(); // Set busy flag - busy with new communication  NextState = 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 watchdog\_timer(void) {  static int i = 0;  if (blink == 1) { // if blink is true  i++;  if (i == wdt\_cycles) { // only toggle if number of cycles complete  P1OUT ^= BIT0;  i = 0;  }  }  IFG1 &= ~WDTIFG;  } |
| --- |