ECE 2049 – Laboratory Report

Embedded Computing in Engineering Design

Worcester Polytechnic Institute

C-Term 2021

Section: C03

Laboratory 4

SPI Communication

Submitted by:

Jonathan R. Lopez

ECE Mailbox #178

3/10/2021

Professor: Yarkin Doroz

Co-Instructor: Fatemeh (Saba) Ganji

Introduction:

The purpose of lab 4 is to make two SPI modules (one Master and one Slave) using our MSP430F5529, our keypad, our buzzer, potentiometer and our SHRP128 LCD screen. We will be using the 12-bit ADC converter on the MSP430 to get the voltages from the potentiometer.

Materials:

TI MSP430F5529 Launchpad based lab board.
Jumper wires
Buzzer
Breadboard
Potentiometer
Sharp128 LCD Screen
0-9, * and # Keypad
Micro-USB to USB-A cable
Computer with Code Composer Studio ver. 10.2

Discussion and Results:

Getting started:

Wiring is a little different with this lab. Figures 1 and 2 show the new wiring.

Approach:

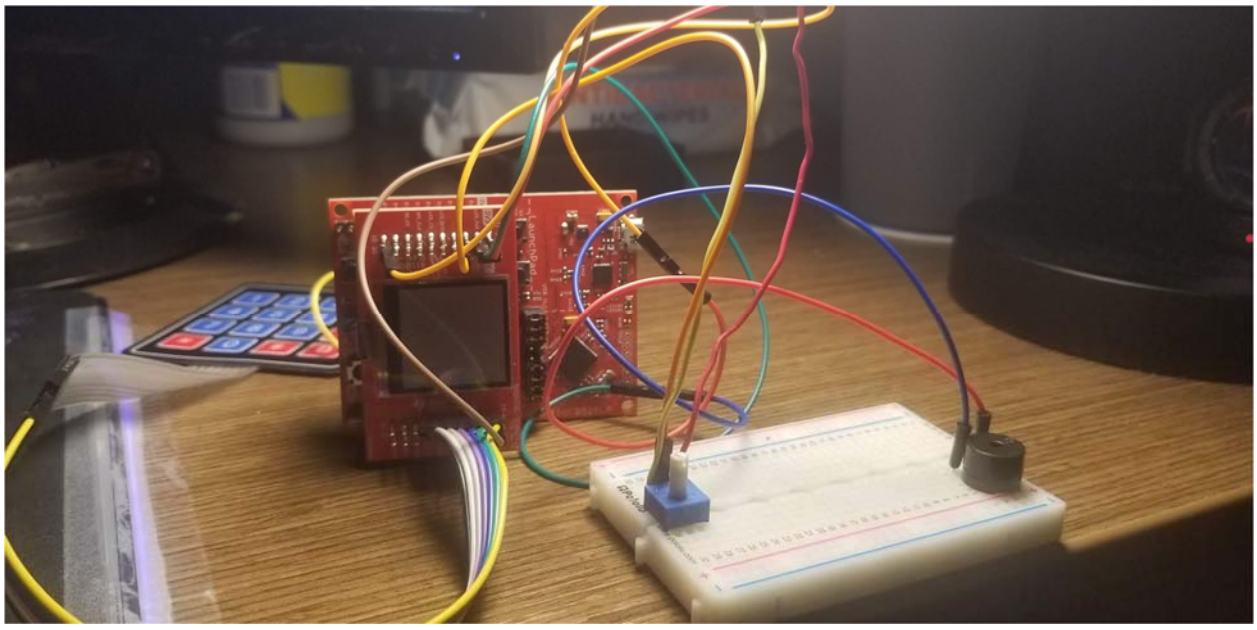

Figure 1: MSP430F5529 wired up and ready for Lab 4

To set up the Slave and master we needed to connect pins on the back of the MSP430 to each other. Figure 3: shows the pins for the Master/slave communication and Figure 2 shows the MSP430 wired up for SPI communication.

The potentiometer also needed to be wired up. We will use P6.4 which is input channel A4. We will keep a note of this when we set up the ADC for the potentiometer “scroll wheel”.

Figure 1: shows the front of the MSP with the potentiometer wired up. VCC is connected to the 3V3 pin for a source voltage of 3.3V and the wiper or middle pin is connected to P6.4. The VEE is connected to a ground pin the MSP.

The pin setup for the LCD is already configure in the `configDisplay ();` function

From Lab 3 we will use a timer that counts every second. We will utilize timmerA2 like in the previous 2 labs

Setup for timerA2 can be found in Figure 4 and the #pragma statement can be found in Figure 5.

The whole schematic of this lab can be found in Figure 6.

With the wiring out of the way we can launch CCS and start programming.

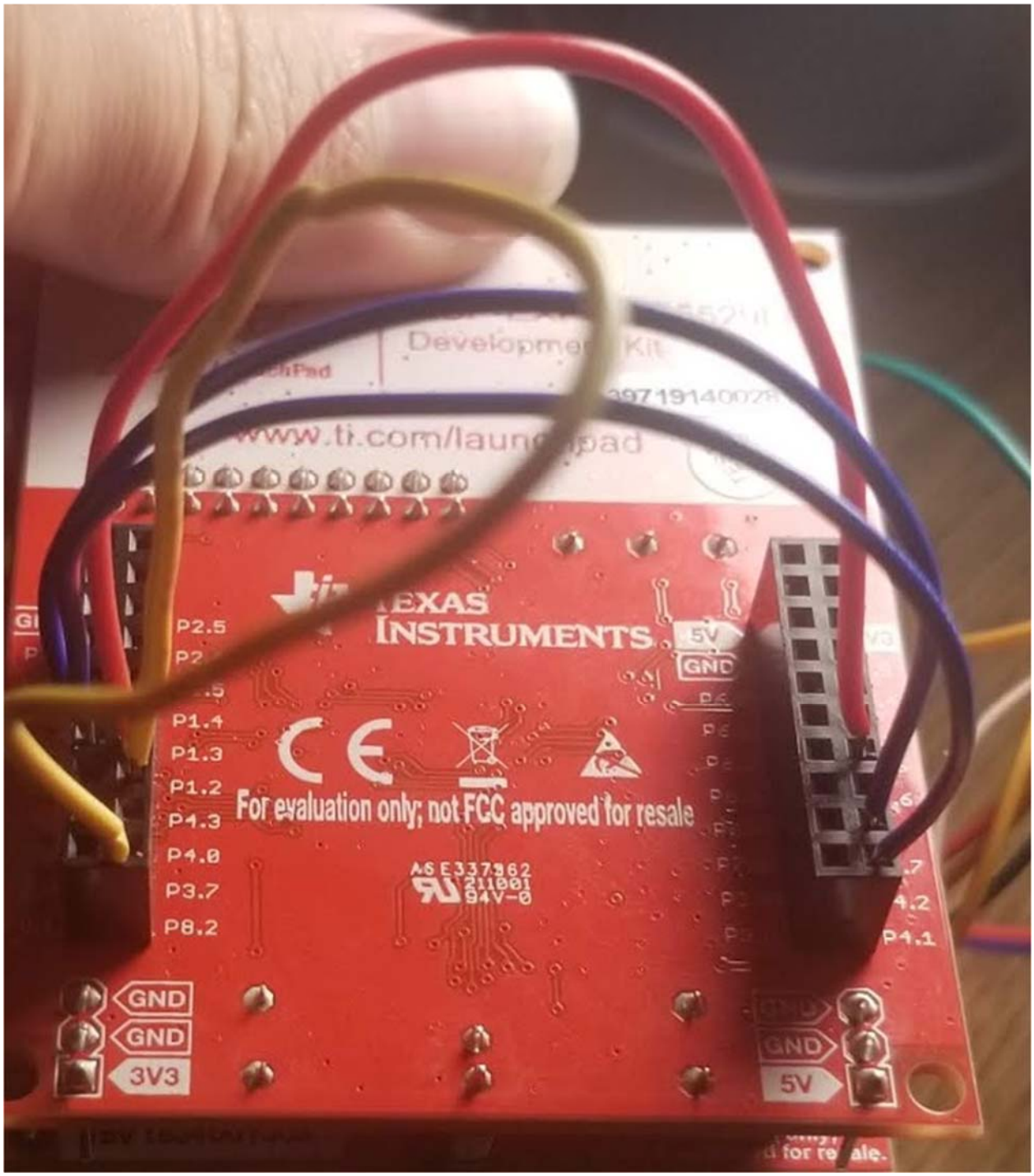

Figure 2: Pin setup for Master/Slave Communication

Pin setup for LCD:

UCB0 – LCD

SOMI 3.0

SIMO 3.1

CS 6.6

CLK 3.2

Pin setup for Master/Slave communication:

UCB0 – Master		UCB1 – Slave
SOMI	3.0	4.2
SIMO	3.1	4.1
CS	8.2 (you can use any other pin available)	4.0
CLK	3.2	4.3

Figure 3: Pin setup for Master/Slave Communication and Pin set up for LCD

```
173 void configTimerA2(){
174     TA2CTL = TASSEL_1 | MC_1 | ID_0;
175     TA2CCR0 = 32767;       //1 second
176     TA2CCTL0 = CCIE;
177 }
```

Figure 4: `void configTimerA2();`

```
//ISR
#pragma vector = TIMER2_A0_VECTOR
__interrupt void Timer_A2_ISR(void)
{
    currsecs++;

}
```

Figure 5: ISR

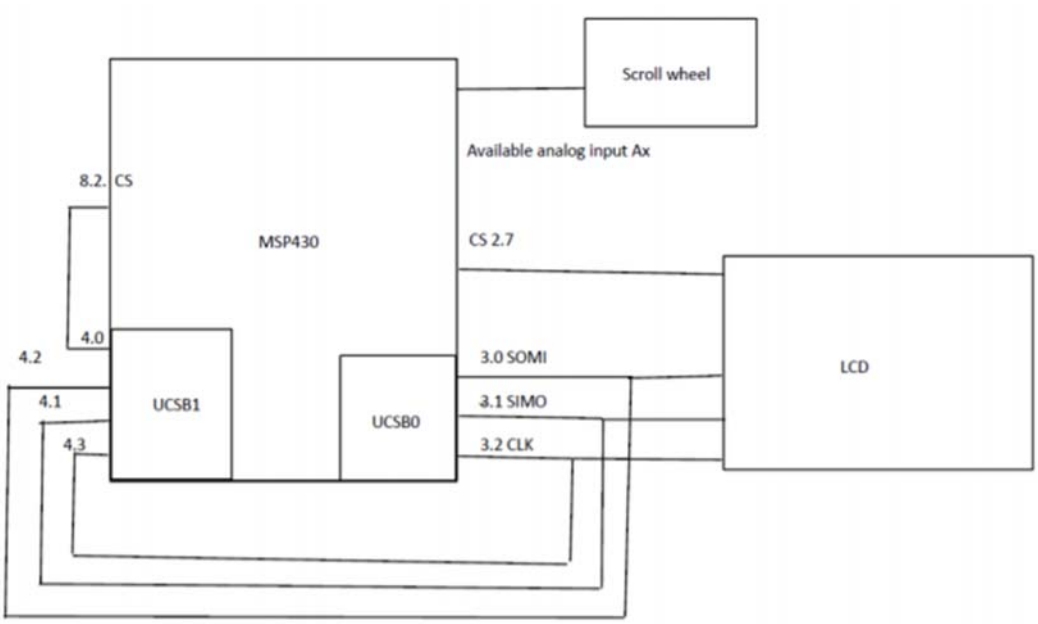

Figure 6: Lab 4 Schematic (Note Ax in this case is A4)

How will you map the output of the scroll wheel to voltage? Explain in your report?

We will map the scroll wheel to a voltage by taking the code and converting it to a voltage using the source voltage and the 12-bit ADC that's on board the MSP430. The configuration for the ADC can be found in Figure 7: and the `float potVolts();` function can be found in Figure 8: The functions take the code produced by the ADC and turn it into a float value for voltage across the potentiometer. We will then take this information and put it through the master and slave devices. It needs 4 reads and writes to get the information across since it can only take 1 byte at a time. There are 8 bits in a byte. Floats are 32 bits. There's some converting to get the floats into 4 bytes. Each byte I stored in an unsigned char to be put through the master and slave. This conversion is shown in Figure 13. After shifting the bits and getting them into their own variables, they are written by the master and read by the slave.

`void MasterSPIWrite (unsigned int data);` will take care of the writing and
`unsigned char SlaveSPIRead();` will take care of the reading.

After some debugging, I was able to get the test one working and then the timer and finally the voltage. The test SPI code can be found in Figure 9. The test one was verified using the debugger.

`void MasterSPIWrite (unsigned int data);` can be found in Figure 10.

`unsigned char SlaveSPIRead();` can be found in Figure 11.

`void InitSlaveSPI();`can be found in Figure 12.

```
240 void configADC12(){
241     REFCTL0 &= ~REFMSTR;
242     ADC12CTL0 = ADC12SHT0_9 | ADC12REFON | ADC12ON ;
243     ADC12CTL1 = ADC12SHP;
244     P6SEL |= BIT4;
245     ADC12MCTL0 = ADC12SREF_0 | ADC12INCH_4;
246     __delay_cycles(100);
247     ADC12CTL0 |= ADC12ENC;
248 }
```

Figure 7: `void configADC12();`

```
250 float potVolts(){
251     float voltsHere;
252     ADC12CTL0 &~ ADC12SC;
253     ADC12CTL0 |= ADC12SC;
254     while (ADC12CTL1 & ADC12BUSY)
255         __no_operation();
256     volatile float potReading = ADC12MEM0 & 0x0FFF;
257     voltsHere = ((potReading/4095) * 3.3);
258     return voltsHere;
259 }
```

Figure 8: `float potVolts();`

```
// test SPI code
MasterSPIWrite(w);
r = SlaveSPIRead();
// r should be 0x55
```

Figure 9: Test SPI Code

```
308 void MasterSPIWrite (unsigned int data){
309     // Start SPI transmission by de-asserting CS
310
311     MSP_PORT_CS_OUT &= ~(MSP_PIN_CS);
312     // Write data/ 1-byte at a time
313     uint8_t byte = (unsigned char) ((data)&0xFF);
314     // Send byte
315     MSP_SPI_REG_TXBUF = byte;
316     // Wait for SPI peripheral to finish transmitting
317     while (!(MSP_SPI_REG_IFG & UCTXIFG)) {
318         _no_operation();
319     }
320     // Assert CS
321
322     MSP_PORT_CS_OUT |= MSP_PIN_CS;
323 }
```

Figure 10: `void MasterSPIWrite (unsigned int data);`

```
300 unsigned char SlaveSPIRead(){
301     unsigned char c;
302     while (!(SLAVE_SPI_REG_IFG & UCRXIFG)){
303         c = SLAVE_SPI_REG_RXBUF;
304     }
305     return (c & 0xFF);
306 }
```

Figure 11: `unsigned char SlaveSPIRead();`

```
271 void InitSlaveSPI(){
272     // Configure SCLK, MISO and MOSI for peripheral mode
273     SLAVE_PORT_SPI_SEL |=
274             (SLAVE_PIN_SPI_MOSI|SLAVE_PIN_SPI_MISO|SLAVE_PIN_SPI_SCLK);
275     // Configure the Slave CS as an Input P4.0
276     SLAVE_PORT_CS_SEL &= ~SLAVE_PIN_SPI_CS;
277     SLAVE_PORT_CS_DIR &= ~SLAVE_PIN_SPI_CS;
278     SLAVE_PORT_CS_REN |= SLAVE_PIN_SPI_CS;
279     SLAVE_PORT_CS_OUT |= SLAVE_PIN_SPI_CS;
280     // Configure the CS output of MSP430 P8.2. It will set P4.0 high or low.
281     MSP_PORT_CS_SEL &= ~MSP_PIN_CS;
282     MSP_PORT_CS_DIR |= MSP_PIN_CS;
283     MSP_PORT_CS_OUT |= MSP_PIN_CS;
284     // Disable the module so we can configure it
285     SLAVE_SPI_REG_CTL1 |= UCSWRST;
286     SLAVE_SPI_REG_CTL0 &= ~(0xFF); // Reset the controller config parameters
287     SLAVE_SPI_REG_CTL1 &= ~UCSSEL_3; // Reset the clock configuration
288     // Set SPI clock frequency (which is the same frequency as SMCLK so this can apparently be 0)
289     SPI_REG_BRL = ((uint16_t)SPI_CLK_TICKS) & 0xFF; // Load the low byte
290     SPI_REG_BRH = (((uint16_t)SPI_CLK_TICKS) >> 8) & 0xFF; // Load the high byte
291     //capture data on first edge
292     //inactive low polarity
293     //MSB first
294     //8 bit
295     //Slave Mode//
296     //4 wire - SPI active low
297     //Synchronous mode
298     SLAVE_SPI_REG_CTL0 |= UCCKPH + UCMSB + UCMODE_2 + UCSYNC;
299     // Reenable the module
300     SLAVE_SPI_REG_CTL1 &= ~UCSWRST;
301     SLAVE_SPI_REG_IFG &= ~UCRXIFG;
302 }
```

Figure 12: `void InitSlaveSPI();`

```
260 void breakUpFloat(float in){
261     unsigned char c[sizeof in];
262     memcpy(c, &in, sizeof in);
263     c[0] = bytes[0];
264     c[1] = bytes[1];
265     c[2] = bytes[2];
266     c[3] = bytes[3];
267 }
```

Figure13: Float to 4 bytes.

That unsigned int for the timer will then be put through the same functions as the bytes for Voltage. It will also take 4 reads and writes to get the information across since `unsigned long int` are also 32 bits. It takes 1 byte at a time. There are 8 bits in a byte. Each byte will be stored in an `unsigned int`

```
124         //TIME
125         idkyet = 0x00000000;
126         timer_cpy = currsecs;
127         timer_cpy1 = (timer_cpy & 0xFF000000)>>24;
128         timer_cpy2 = (timer_cpy & 0x00FF0000)>>16;
129         timer_cpy3 = (timer_cpy & 0x0000FF00)>>8;
130         timer_cpy4 = (timer_cpy & 0x000000FF);
131
132         MasterSPIWrite(timer_cpy1);
133         idkyet  |= ((long unsigned int) (SlaveSPIRead()) << 24);
134         MasterSPIWrite(timer_cpy2);
135         idkyet  |= ((long unsigned int) (SlaveSPIRead()) << 16);
136         MasterSPIWrite(timer_cpy3);
137         idkyet  |= ((long unsigned int) (SlaveSPIRead()) << 8);
138         MasterSPIWrite(timer_cpy4);
139         idkyet  |= ((long unsigned int) (SlaveSPIRead()));
140
141         //VOLTAGE
142         voltsCopy = volts1;
143         breakUpFloat(voltsCopy);
144         //bytes now set
145         //SPI time
146         MasterSPIWrite(bytes[0]);
147         idkyet1[0]  |= ((long unsigned int) (SlaveSPIRead()) << 24);
148         MasterSPIWrite(bytes[1]);
149         idkyet1[1]  |= ((long unsigned int) (SlaveSPIRead()) << 16);
150         MasterSPIWrite(bytes[2]);
151         idkyet1[2]  |= ((long unsigned int) (SlaveSPIRead()) << 8);
152         MasterSPIWrite(bytes[3]);
153         idkyet1[3]  |= ((long unsigned int) (SlaveSPIRead()));
```

Figure 14: SPI for both the timer and voltage

Figure 15: Timer and Voltage displayed on the MSP430

I've attached the signoff video on canvas along with this report and code in a zip file.

Summary and Conclusion:

In conclusion, this lab had the purpose of getting students more familiar with the interrupts, timers, SPI, the C programming language and the environment of Code Composer Studio as well as get the hang of using the builder, debugger and the variable window within Eclipse/CCS. This concludes this laboratory experiment with the SPI Communication lab on the MSP430F5529 using the buzzer, potentiometer, keypad, and Code Composer Studio.

Appendices

ASCII table:

Lab 4 Handout:
https://canvas.wpi.edu/courses/23392/files/3597862?wrap=1

Demo Project:
https://canvas.wpi.edu/courses/23392/files/3416786?wrap=1

CCS download:
https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html

CCS tutorial:
https://www.youtube.com/watch?v=2W0ZHO0vzJE

MSP430 user guide:
https://canvas.wpi.edu/courses/23392/files/3432903?wrap=1

C Review pdf
https://canvas.wpi.edu/courses/23392/files/3439859?module_item_id=542026

ECE 2049 Lab BOM:
https://canvas.wpi.edu/courses/23392/files/3411701?module_item_id=527923

ECE 2049 Course Information:
https://www.wpi.edu/academics/calendar-courses/course-descriptions/17856/electrical-computer-engineering#ECE-2049

Potentiometer Data Sheet:
https://www.digchip.com/datasheets/parts/datasheet/1064/WIW3362-R-102-pdf.php

ECE2049 C-2021 Lab 4
Sign-off Sheet

***Report due*: 18/03/21**

Student 1: _____ ECE mailbox: _____

Student 2: _____ ECE mailbox: _____

YOU ARE RESPONSIBLE FOR *ALL* THE REQUIREMENTS and ***QUESTIONS*** LISTED IN THE ASSIGNMENT!

Welcome screen with timer and voltage level	5	
Initilize Slave device with initSlaveSPI	20	
Read Voltage from Scrolling Wheel with readVoltage() and voltage changes with scroll wheel	10	
Send data using masterSpiWrite()	10	
Read data using slaveSpiRead()	10	
Send voltage and time information using read/write functions multiple times and construct the correct values from the character values and displaying them on screen	20	
Report	25	
Total points	100	