**SOEN 422/2 Fall 2016**

**LAB REPORT #2-3**

**Team member:**

Jacqueline Luo (26938949)

TABLE OF CONTENTS

1. Introduction

1.1 Problem Statement

1.2 Abbreviations and Acronyms

2. Resources

2.1 Hardware Resources

2.2 Hardware Setup

2.3 Software Resources (compilers, loaders, etc.)

2.4 Software Setup (include the software you used to write and load the programs, including any special procedures

3. Programs

3.1 - Bare Metal Code (Atmega328)

3.1.1 Button Controlled LED

3.1.2 Timer Controlled LED

3.1.3 Button Controlled Timers

3.2.1 USB/UART Communication

3.2.2 SPI Communication

3.2.3 I2C Communication

4. Discussion/Conclusion

Note: For each code part, start with the question copied and pasted from the lab question paper. Give a brief explanatory note if appropriate, concerning any difficulty you encountered. Your code should include comments.

**Copy the report frame at the start of your work. Paste in the lab question followed by the code as you proceed.**

**1.1**

In these two labs, we were responsible for learning how to set up and program the Atmega328. In lab 2, we had to learn how interrupts work and how to program them. In lab 3, we learned about communication between devices. This includes SPI, I2C and UART communication.

**1.2**

FTDI: Future Technology Devices International

SPI: Serial Peripheral Interface

I2C: Inter-Integrated Circuit

USB: Universal Serial Bus

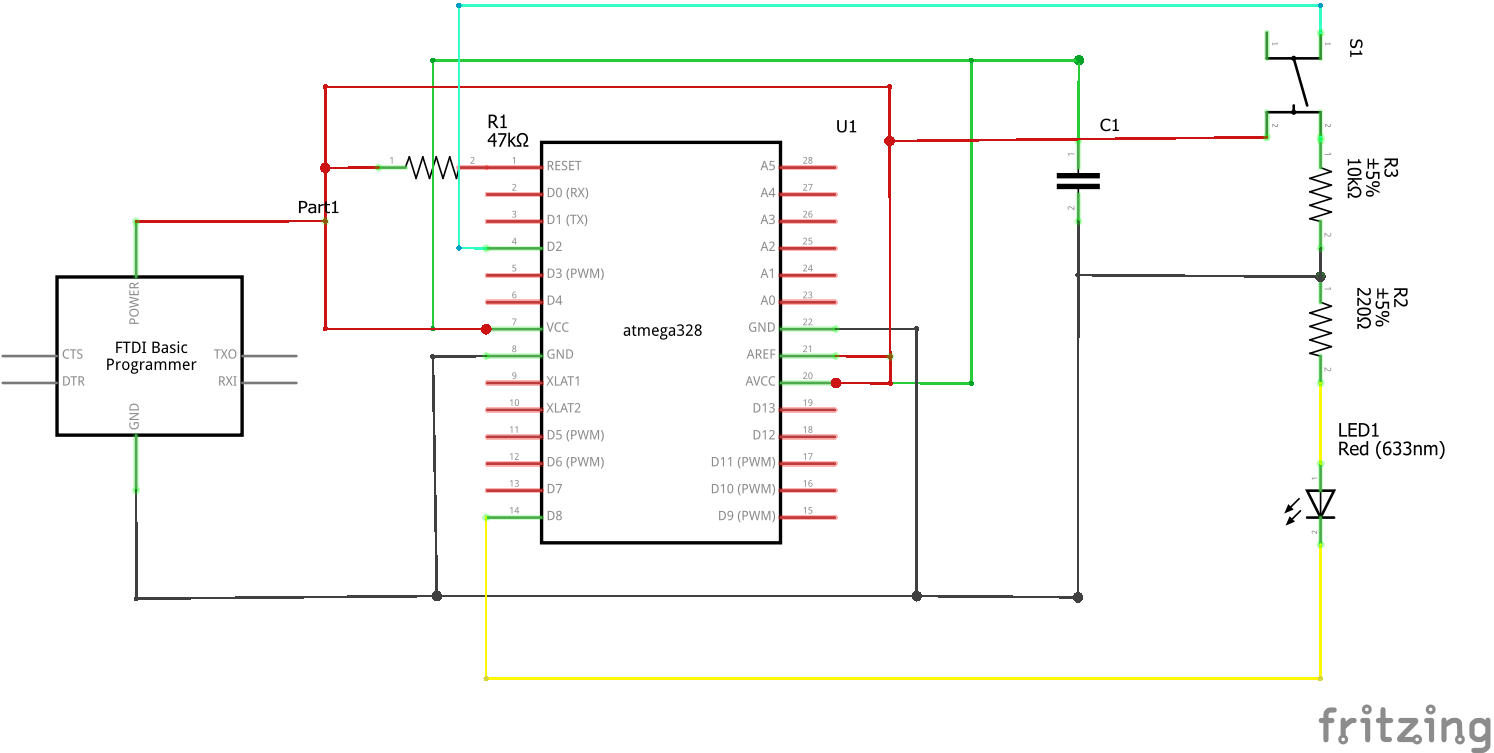
UART: Universal Asynchronous Receiver/Transmitter

**2.1**

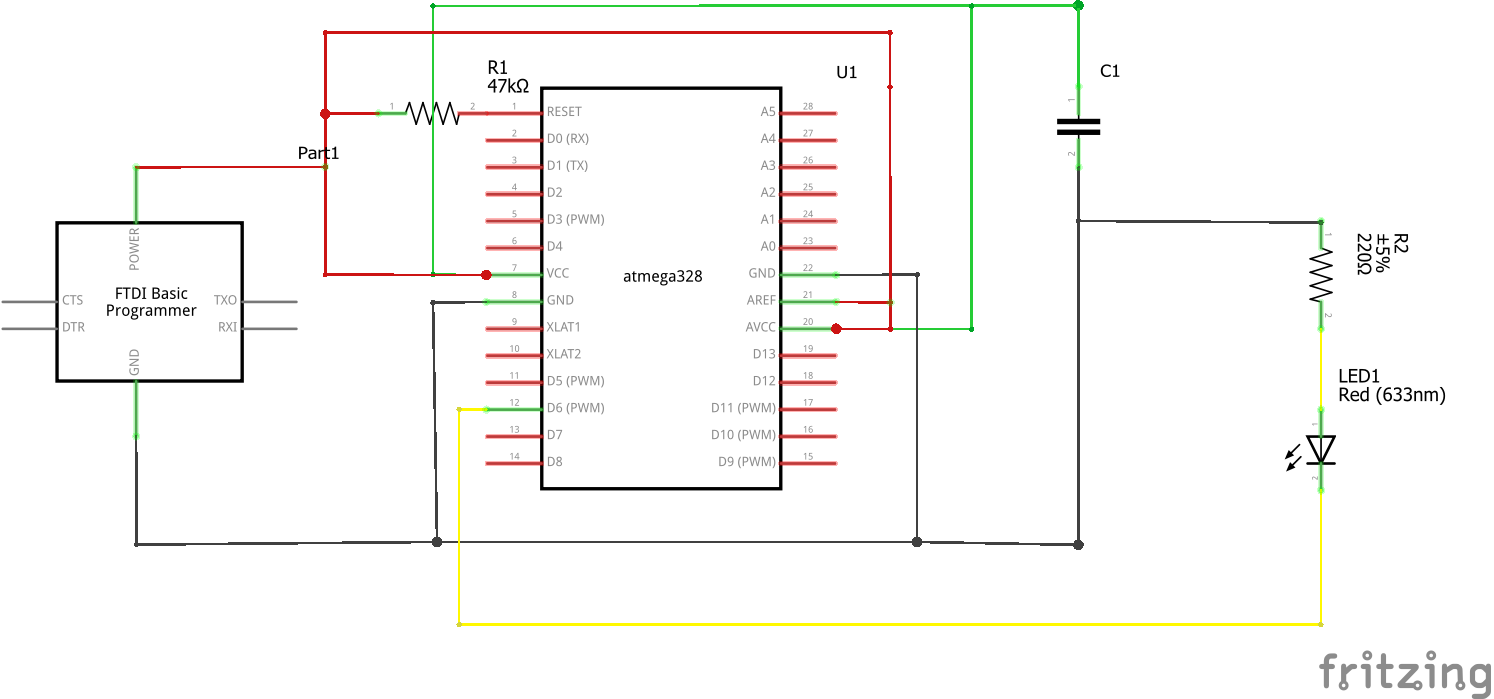
For these labs, we used:

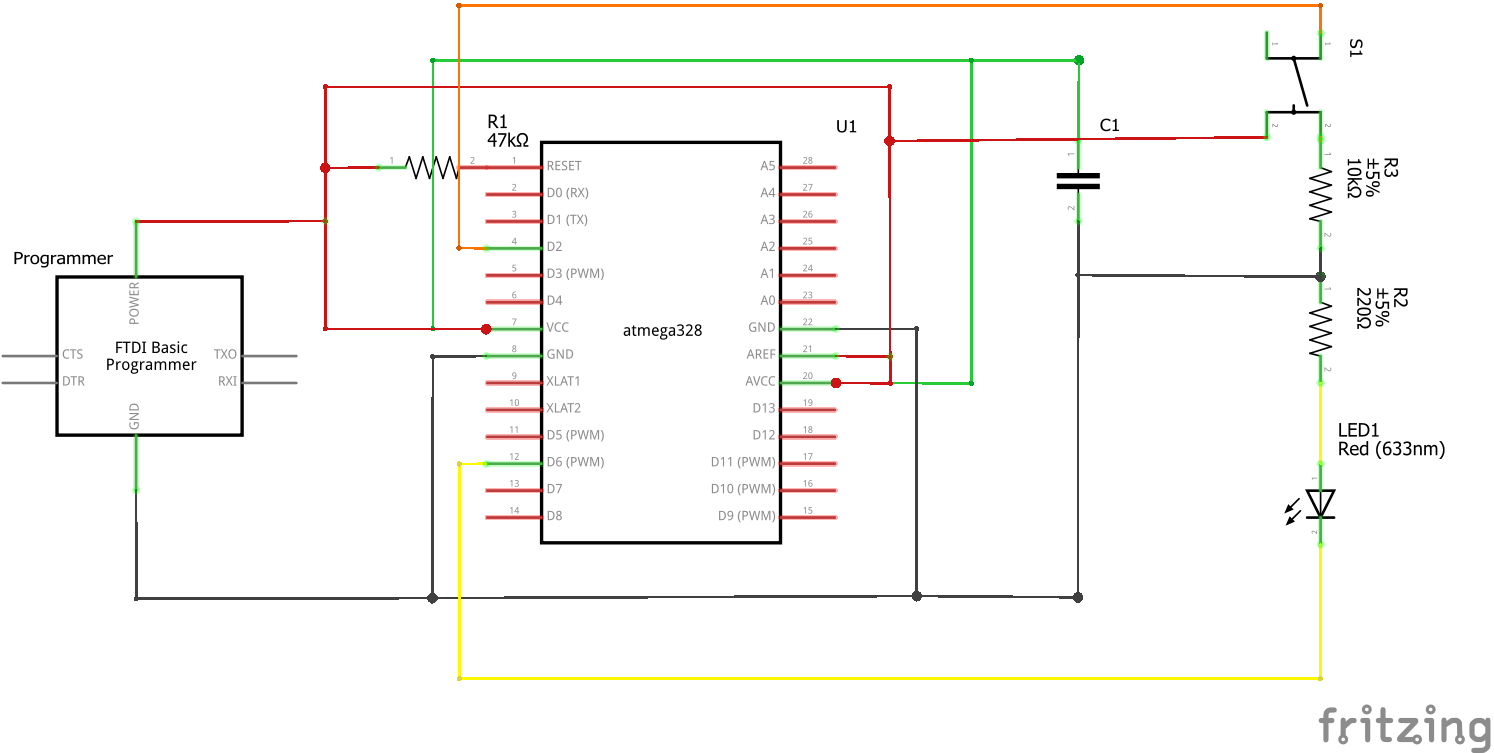
* Breadboard
* Wires
* Resistors
  + 47k Ohm resistor
  + 10k Ohm resistor
  + Bussed resistor pack
* 10 nF Capacitor
* LED pack
* Atmega328 microcontroller
* Push Button
* Sparkfun FTDI
* Sparkfun PGM-09825 AVR Pocket Programmer
* Atmega328 Programming Board
* SHARP 2Y0A02 distance sensor
* ISP cable
* USB to mini USB cable

**2.2**

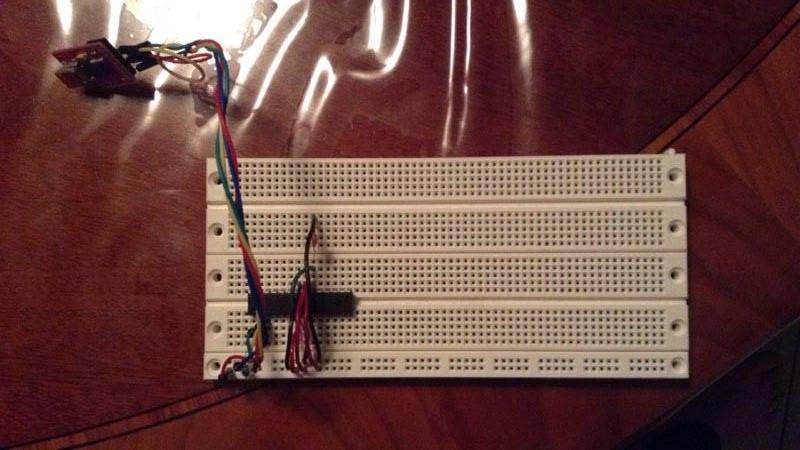
**Lab 2 part 1**

**Lab 2 part 2**

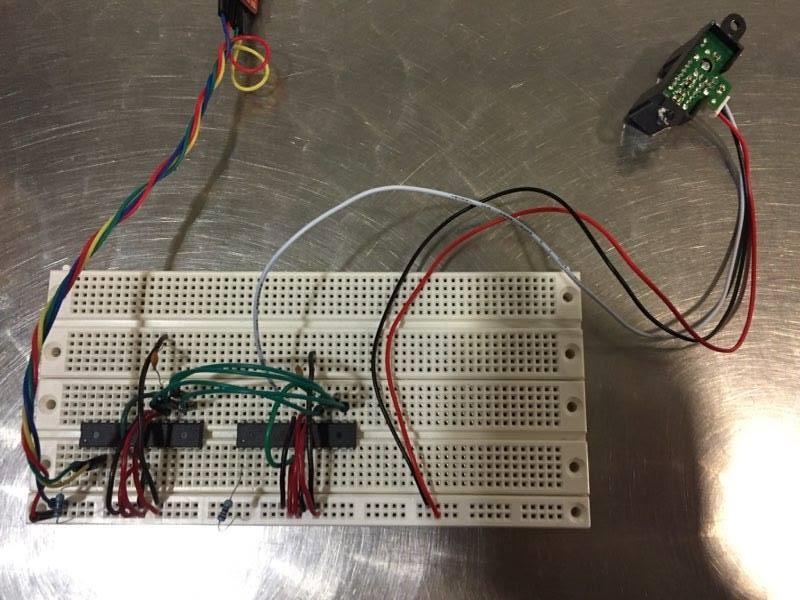


**Lab 2 part 3**

**Lab 3 part 1 (USB/UART)**



**Lab 3 part 2A (SPI)**



**Lab 3 part 2B (I2C)**

**2.3**

I used:

* AVRdude
* Notepad++
* Windows Command Line
* WinAVR
* PuTTY

**2.4**

I had to install WinAVR on my Windows laptop so that I could do development on the AVR controller. WinAVR already has the Atmel AVR toolchain included so there is no need to install that as well. I also had to install AVRdude in order to flash the code onto the microcontroller. I used notepad++ to do the coding because it is simple and there is no need to install an IDE if notepad++ fits the job. Plus it was already installed on my system. The first time I tested AVRdude there was an issue. For some reason, AVRdude does not recognize the Atmega328. However, it does have the 328p. To get AVRdude to recognize the Atmega328 chip, I had to do a special workaround that involved changing the signature of the Atmega328p into that of the Atmega328. All this was done in the avrdude.conf file. To do the compiling and loading, I used the Windows command line. PuTTY was used as a serial monitor for lab 3.

**3.1- Baremetal code**

**3.1.1 Part 1: Toggling an LED By Switch**

The first task is to toggle an LED by toggling a switch. The while loop in your main() function should be empty. The button must trigger INT0. You may use either an internal or external pull up resistor.

For this part, I originally had trouble setting the hardware up. I also had trouble getting this part to work because when the button would not work, I was uncertain if it was a hardware or software issue. I had to individually test the button, the LED, the chip to see if it was a hardware issue. Once I was sure it wasn’t, I had to check my code, which seemed correct, but as I am not familiar with programming on the Atmega328 using C++, I could not be certain. Ultimately, the problem was that my chip would work only if I solidly plugged in one side of the pins, as opposed to both sides, or neither sides.

#include <avr/io.h>

#include <avr/interrupt.h>

int main(void)

{

DDRD |= (0 << PD2); //set PD2 as input, may not be necessary but I want to be sure

DDRB |= (1 << PB0); //set PB0 as output

EICRA |= (1 << ISC01); // set INT0 to trigger on falling edge

EIMSK |= (1 << INT0); // Turns on INT0

sei(); //enable global interrupts

while (1)

{

//main loop

}

}

ISR(INT0\_vect)

{

cli(); //disable global interrupts

PORTB ^= (1 << PB0); //toggle LED

sei();

}

**3.1.2 Part 2: Timer Controlled LEDs**

The next task is to toggle an LED by using a timer and an interrupt. The LED should toggle state every 5 seconds.

For this part, my biggest difficulties was getting the timer to toggle every 5 seconds. The calculations needed were not intuitive to me, but I ultimately got it to work, give or take a few milliseconds.

#include <avr/io.h>

#include <avr/interrupt.h>

int main(void)

{

DDRD |= (1 << PD6); //set PD6 as output

TCCR1A = 0;

TCCR1B = 0;

// set up timer with prescaler = 256 and CTC mode

TCCR1B |= (1 << WGM12)|(1 << CS12);

// initialize counter

TCNT1 = 0;

// initialize compare value

//OCR1A = 15624; ~1 sec

OCR1A = 20000;

TIMSK1 = (1 << OCIE1A); //enable interrupt for compare

sei(); //enable global interrupts

while (1)

{

//main loop

}

}

ISR (TIMER1\_COMPA\_vect) // timer1 compare interrupt

{

PORTD ^= (1 << PD6); //event to be executed every 5s here

}

**3.1.3 Part 3: Button Controlled Timers**

The last task is to implement a method to toggle the timer used in Part 2. Assume that when the timer is stopped, it is reset back to 0.

This part was the easiest, since most of the heavy lifting was already done in the previous two parts. The reason I chose timer1 was because it is a 16 bit timer as opposed to timer0.

#include <avr/io.h>

#include <avr/interrupt.h>

int main(void)

{

DDRD |= (0 << PD2); //set PD2 as input

DDRD |= (1 << PD6); //set PD6 as output

TCCR1A = 0;

TCCR1B = 0;

// set up timer with prescaler = 256 and CTC mode

TCCR1B |= (1 << WGM12)|(1 << CS12);

// initialize counter

TCNT1 = 0;

// initialize compare value

//OCR1A = 15624; ~1 sec

OCR1A = 20000;

TIMSK1 = (1 << OCIE1A); //enable interrupt for compare

EICRA |= (1 << ISC01); // set INT0 to trigger on falling edge

EIMSK |= (1 << INT0); // Turns on INT0

sei(); //enable global interrupts

while (1)

{

//main loop

}

}

ISR(INT0\_vect)

{

cli(); //disable global interrupt

TCCR1B ^= (1 << WGM12)|(1 << CS12); //toggles timer

TCNT1 = 0; //resets timer to 0

sei();

}

ISR (TIMER1\_COMPA\_vect) // timer1 compare interrupt

{

PORTD ^= (1 << PD6); //event to be executed every 5s here

}

**3.2.1 Part 1 USB/UART Communication**

The objective of this part of the lab is to be able to display 3 different greetings on the Serial Monitor. The user should be able to send a number and the ATmega328 should respond with a greeting. You may use the Serial Monitor in the Arduino IDE to interact with the ATmega328.

I used the code that was given as a base to start. What was tricky is that strings are arrays of chars in c++ unlike java. Another thing that I realized was that UDR0 takes a byte.

#include<avr/io.h>

#define F\_CPU 1000000UL

#include <util/delay.h>

#include <string.h>

#define USART\_BAUDRATE 4800

#define BAUD\_PRESCALE (((F\_CPU/(USART\_BAUDRATE\*16UL)))-1)

int main(void)

{

char received\_byte; //The byte that is read and sent

char message1[20] = "Hello there!"; //A greeting

char message2[20] = "Oh hai dere!"; //Another greeting

char message3[20] = "Greetings, wow."; //A third greeting

UCSR0B |= (1<<RXEN0) | (1<<TXEN0); //Initialization for serial communication

UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01);

UBRR0H = (BAUD\_PRESCALE >> 8);

UBRR0L = BAUD\_PRESCALE;

for(;;)

{

// wait until a byte is ready to read

while( ( UCSR0A & ( 1 << RXC0 ) ) == 0 ){}

// grab the byte from the serial port

received\_byte = UDR0;

// wait until the port is ready to be written to

while( ( UCSR0A & ( 1 << UDRE0 ) ) == 0 ){}

if(received\_byte == '0') //If what was typed was 0

{

for(int i=0; i<strlen(message1); i++)//a loop to go through the entire array of chars that message 1 is comprised of

{

UDR0 = message1[i]; //send char by char

\_delay\_ms(10); //need delay to print all

}

UDR0 = 0x0A; //new line

UDR0 = 0x0D; //carraige return, aka make cursor go to front

}

else if(received\_byte == '1') //If what was typed was 1

{

for(int i=0; i<strlen(message2); i++)

{

UDR0 = message2[i];

\_delay\_ms(10);

}

UDR0 = 0x0A;

UDR0 = 0x0D;

}

else //for anything else that is typed

{

for(int i=0; i<strlen(message3); i++)

{

UDR0 = message3[i];

\_delay\_ms(10);

}

UDR0 = 0x0A;

UDR0 = 0x0D;

}

}

return 0; /\* never reached \*/

}

**3.2.2 Part 2A: SPI Communication**

Implement the Master/Slave configuration using SPI.

I had difficulty getting the sensor to work properly. I ended up taking some code from online and from notes for the sensor and the SPI and messing around with it. At some point, there was issues opening a serial port. I fixed that by unplugging and replugging the FTDI mini USB. I set the master as the one that is taking in the sensor info because when it takes in info, it then sends that to the slave. The master is the one who knows when it is taking in info so it chooses when to inform the slave.

//Code for master

#include <avr/io.h>

#include <stdlib.h>

#define F\_CPU 1000000UL // Set CPU speed for delay

#include <util/delay.h>

#define BAUDRATE 4800

#define BAUD\_PRESCALLER (((F\_CPU / (BAUDRATE \* 16UL))) - 1)

uint16\_t adc\_value; //Variable used to store the value read from the ADC

void adc\_init(void); //Function to initialize/configure the ADC

uint16\_t read\_adc(uint8\_t channel); //Function to read an arbitrary analogic channel/pin

uint16\_t read\_adc(uint8\_t channel){

ADMUX &= 0xF0; //Clear the older channel that was read

ADMUX |= channel; //Defines the new ADC channel to be read

ADCSRA |= (1<<ADSC); //Starts a new conversion

while(ADCSRA & (1<<ADSC)); //Wait until the conversion is done

return ADCW; //Returns the ADC value of the chosen channel

}

void SPIMasterSend(uint8\_t data)

{

PORTB &= ~(1<<PB2); //turn BP2 low, therefore activating slave

SPDR = data; //send data

while (!(SPSR &(1<<SPIF))); //wait for transmition complete

PORTB |= (1<<PB2); //turn PB2 high, therefore deactivating slave

}

int main(void)

{

ADCSRA |= ((1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)); //16Mhz/128 = 125Khz the ADC reference clock

ADMUX |= (1<<REFS0); //Voltage reference from Avcc (5v)

ADCSRA |= (1<<ADEN); //Turn on ADC

ADCSRA |= (1<<ADSC); //Do an initial conversion

DDRB |= (1<<PB3)|(1<<PB5)|(1<<PB2); //set MOSI, SCK and SS as output

PORTB |= (1<<PB2); //set SS to high

SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0); //enable master SPI at clock rate Fck/16

while (1)

{

adc\_value = read\_adc(0); //Read ADC0

SPIMasterSend(adc\_value); //send ADC0 value to slave

\_delay\_ms(500);

}

}

--------------------------------------------------------------------------------------------------------------------------

//Code for Slave

#include <avr/io.h>

#include <stdlib.h>

#define F\_CPU 1000000UL

#include <util/delay.h>

#include <avr/interrupt.h>

#define BAUDRATE 4800

#define BAUD\_PRESCALLER (((F\_CPU / (BAUDRATE \* 16UL))) - 1)

char buffer[5];

void USART\_init(void);

void USART\_send( unsigned char data);

void USART\_putstring(char\* StringPtr);

void USART\_send( unsigned char data) //USART send to UDR0

{

while(!(UCSR0A & (1<<UDRE0)));

UDR0 = data;

}

void USART\_putstring(char\* StringPtr) //send string as characters

{

while(\*StringPtr != 0x00)

{

USART\_send(\*StringPtr);

StringPtr++;

}

}

int main(void)

{

UBRR0H = (uint8\_t)(BAUD\_PRESCALLER>>8);

UBRR0L = (uint8\_t)(BAUD\_PRESCALLER);

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

UCSR0C = (3<<UCSZ00);

DDRB |= (1<<PB4); //set MISO as output

SPCR = (1<<SPE)|(1<<SPIE); //enable SPI and enable SPI interrupt

TCNT0=0; //Set Initial Timer value

OCR0A=0; //Place compare value to Output compare register

TCCR0A|=(1<<COM0A1)|(1<<WGM01)|(1<<WGM00); //Set fast PWM mode

TCCR0B|=(1<<CS01)|(1<<CS00); //Set prescaler to 64 and start timer

sei(); //enable global interrupts

while (1)

{

//main loop

}

}

ISR(SPI\_STC\_vect) //Interrupt for when serial transfer is finished

{

USART\_putstring("IR Sensor Reading: ");

itoa(SPDR, buffer, 10); //Convert from int to string

USART\_putstring(buffer); //Send the converted value to the terminal

USART\_putstring(" ");

USART\_send('\r');

USART\_send('\n');

}

**3.2.3 Part 2B: I2C Communication**

Implement the Master/Slave configuration using I2C.

**4. Discussion/ Conclusion**

A lot of the times, my biggest trouble was figuring out what went wrong. It’s hard to tell if it is the hardware or software that is not working since sometimes code snippets that are given do not work and sometimes there is a wonky wire or a wonky pin. I did learn about the interrupts’ specific vectors and how to work with the datasheet in order to know which pins to set high. For communication, I got serial connections working and SPI communication, but I could not get I2C to work.