Lab2 Report

This lab consisted of three parts, LED brightness controlled by User (throught PWM), out putting the values from an IR sensor to a serial monitor, and lastly controlling the brightness of the LED using the IR sensor with at least three levels.

> *To best view this document, use the link provided*

Table of Contents

Contributors

Student	ID
Christopher McArthur	`40004257`

Introduction

- Problem Statement: Using arduino and coding in 'bare metal' (c programming) create a 4-bt counter that displays the binary value of the counter on a set of LEDs. the solution should take in a count debouncing effect.
- Abbreviations and Acronyms
    - LED == Light Emiting Diode
    - DDRx, PBn, etc... Registers names
    - OCR == output compare register
    - IR == infra red

    - ADC == analog to digital conversion
    - PWM == pulse with modulation

Resources

- Hardware Resources
    - Arduino Nano
    - LEDs
    - Push Button
    - Resistors
    - Jumper Wires
    - IR Sensor
- Hardware Setup
    - I have not installed Fritzing

Software Resources

- Software setup
    - IDE is Visual Studio 2017 Enterprise
    - Compilers are the avr-gcc/avr-g++ for windows
    - Visual studio has an extension called Visual Micro which handles everything from compiling to upload (it just needs to be configured similar to the Arduino IDE)

Program Snipets

- timer 2 setup code

```
TCCR2A |= (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);  //clear on compare match, continue count to OxFF
TCCR2B |= (1 << CS02) | (1 << CS00); // prescale counter by 1024
DDRB |= (255); // turn all of PB's to output
OCR2A = 255;  //initial compare match value - max brightness
```

- serial reading loop (part 1)

```
if (Serial.available() > 0)
{
    // read the most recent byte (which will be from 0 to 255):
    brightness = Serial.read();

    // print the value
    Serial.print("I received: ");
    Serial.println(brightness, DEC);
    Serial.flush(); // hangs otherwise

    OCR2A = brightness; // set to user input
}
```

- ADC setup

```
ADMUX |= ADC4D | (1 << REFS0); // Vcc ( 5v) conversion scale
ADCSRA |= (1 << ADEN) | (1 << ADPS2); // enable with prescalar  at 16 (65kHz conversion c
lock speed)
```

- ADC controll

```
ADCSRA |= (1 << ADSC); // Start Conversion
while (!(ADCSRA & (1 << ADIF))) {} // wait for ADIF to go to 0, indicating conversion com
plete.
ADCSRA |= (1 << ADIF); // Reset ADIF to 1 for the next conversion
```

Reference Code Common

- the basic Arduino bare metal IO.h for access to pins and registers and basic definitions
- HardwareSerial.h for serial communitcation
- The supplied ADC code was extremely useful, I just specialized it to my particular case.

Discussion/Conclusion

This lab possed more challenges than expected… The largest of which is those with serial. However getting the IR sensor to work also took some time (special thanks to you Mohamed for the removing the resistor tip); I accedentally burnt a nano trying to find why the IR sensor had no output.

Issues with Serial

Having first proto-typed the Lab in arduino the transition to bare metal was not clean cut. Previously all my work for bare metal was in in C-code which is how I started this lab. The issue is that the arduino library is *C++* and the `Serial` object produced compilation errors; this actually took an hour of research because the the errors from the avr-gcc compiler are under equiped. The fix was to rename my file from .c to .cpp such that my visual studio extension would use the correct compiler.

With everything now compiling, running the code didnt even successfully output anything. It would hang after two bytes and never complete printing. Turns out after investigating there's a `Serial.flush()`, code here, whichs waits till the buffer is emptied but maybe helps in sending too. Either way adding this to each of my prints helped.

Now with lovely debug prints fully functional (code no longer hanging), I tried to pass information from the Serial dialogue to the Nano. This doesnt work, my…

```
if(Serial.available() > 0) { /* ... blah blah ... */ }
```

…was never true. Going back and for from arduino to bare metal, I tried testing `Serial.read()` without having printed anything, did have an effect, and a slue of other tweaks and modifications to never obtain a result. I was so faustered at why the arduino was not Rx anything I found this tutorial for doing Serial communication from C++ windows to arduino, and use it to manual check what was written. Guess what? It was written successfully.

Doing a `git reset --hard` to have both the arduino and bare metal the extact same and compiled both. The arduino was 1818 bytes and the bare metal was 1437; clearly the bare metal compilation is missing a key detail. Since all C/C++ programs have a `main()` function I set out to find it, took a while but again it was on github. There was a few snippets which stood out imediately which I did not know or expect…

```
#if defined(USBCON)
    USBDevice.attach();
#endif
```

…and within the loop…

```
if (serialEventRun) serialEventRun();
```

Sadly opening the arduino IDE itself and trying to use `USBDevice` did not work nor was `USBCON` defined. Now what does `serialEventRun()` do? This. and calling `serialEvent()` does what? according to this is is called when data is available. I have honestly no clue why but clearly this is where my issue was. This maps to the input function of the serial communication… I cut out the parirty check since it was the only possible fail case, recompiled the library and rebuilt my program and voila! Everything worked, minus the fact that my read was occasionally outputting garbage (one in every 10 inputs was wrong) it worked. okay so why was the parity/checksum wrong? I don't care that was the problem and its not my bug to fix so I submited the issue.