Johnson Le

January 12, 2019

CMPE 167/L

Lab 0 Report

Introduction

This introductory lab provides useful steps and requirements to help students learn basic hardware and microcontroller used in later labs. The main microcontroller used is the uno32 utilizing a custom bootloader. Within this lab, we are expected to print to serial, read A/D values, create low-pass filter, and control the speaker's pitch.

Part 1 – Hello World!

Method:

Part 1 is straightforward. Following the steps provided in the new project instructions pdf, I included the headers and source files for board and serial, wrote hello world code, and built. Using the ds30loader tool, the I took the .hex file created when building to write it onto the uno32.

Part2 – Hello A/D (reading an Analog Signal)

Method:

Adding onto the libraries needed in part 1, the AD library given is also needed here. To read the values of the on-board potentiometer, I needed to study the provided AD files and figured out that the reading is from the pin AD_A0. To test this, I printed out the values from the pin out onto serial with a delay. In a later part, I changed this to display on the on-board screen.

Part3 – Tone Out Speaker Hard Coded

Method:

To get a sound from the speaker, we are provided with a tone generating library. Upon reading the provided library, it is noted that the PWM pin is pin 3. To test this, I hooked up pin 3

to the oscilloscope to ensure that was the case. To wire up the speaker, pin 3 is connected to the positive terminal of the speaker, and the negative terminal goes to ground.

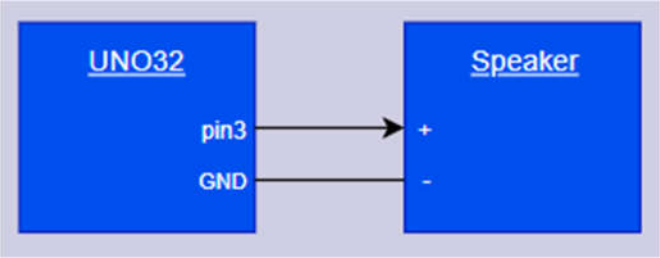

Fig 1: Part 2 Speaker Wiring

Part4 – Tone adjusted Via Pot

Method:

Part 4 is a combination of part 2 and 3 allowing us to control the speaker's pitch through the on-board potentiometer. Throwing the A/D reading into the function to control the PWM output, the speaker does change pitch but it sounds quite scratchy especially when changing the potentiometer. My hypothesis for the cause is bouncing generated by the potentiometer and having it constantly update the speaker. To remedy this, I implemented hysteresis that has its bounds changed based on the previous potentiometer value. This is done by only outputting a new pwm value to pin 3 when there is a big enough change in the current value from the potentiometer. The result is a lot less scratchy.

Part5 – Basic Filtering of Output (Single Pole RC to Smooth Tone Generation)

Method:

Low-pass filter is used to only allow a certain range of frequency blow a certain cut-off frequency. To implement this into our design, I needed to decide what value for the cut-off frequency I needed.

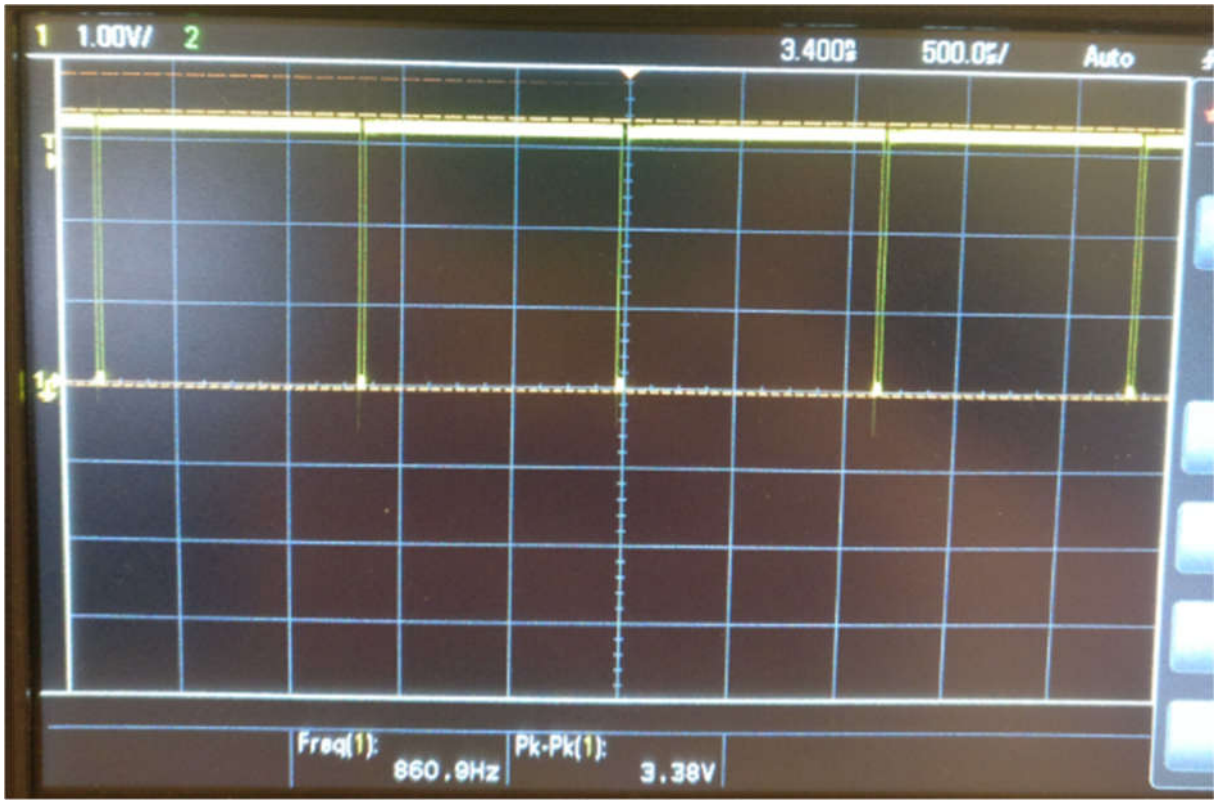

Fig 2: PWM output at 860HZ

To do this, I first displayed the output from pin 3 from the uno32 on to the oscilloscope as seen in figure 2. Altering the potentiometer, I notice the square wave gets inverted/disappears at around 880hz and above. Therefore, I needed a cut-off frequency of about 880hz or a bit lower. I did not want to lose too much gain, so I went with a decently large capacitor of 10uF for the design. After some calculations, an 18ohm resistor gets me close to the desired cut-off frequency of 880hz.

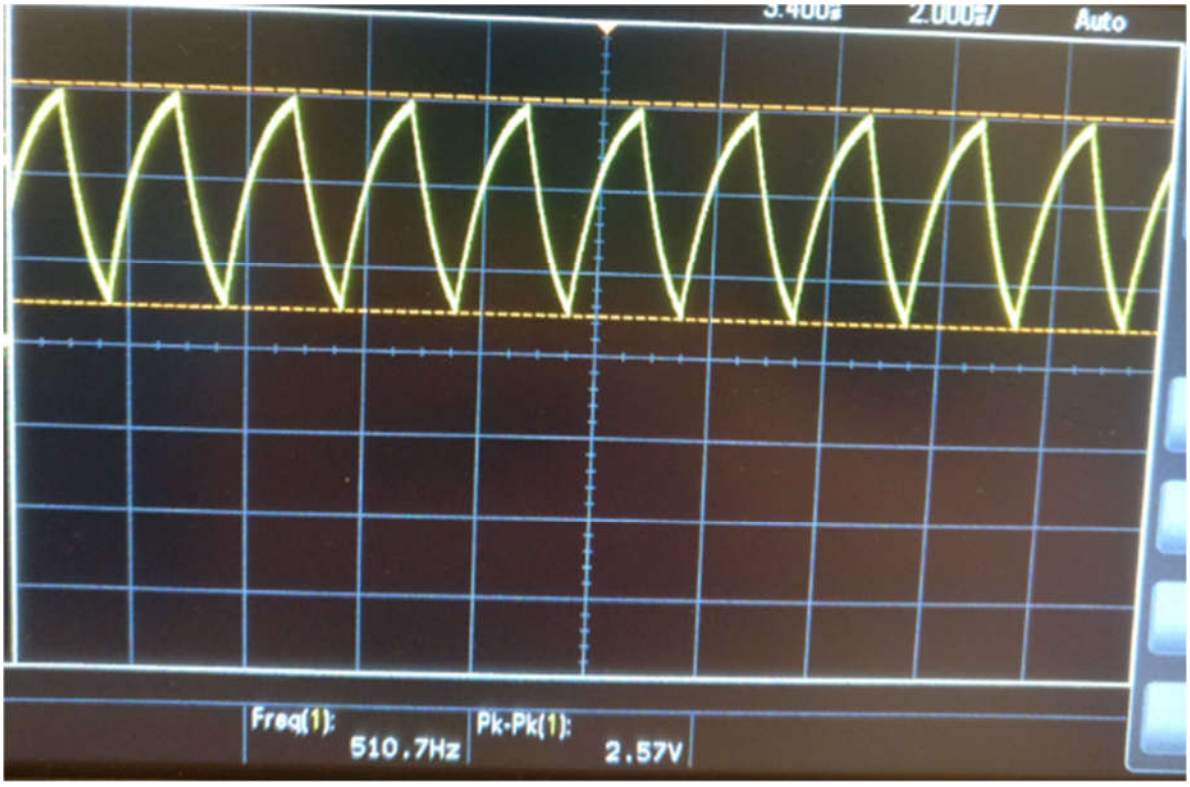

Fig 3: 50% Duty Cycle from PWM after Low-Pass Filter

Connecting pin 3 to the input of the low-pass filter, and the speaker to the output, the sound is now dampened slightly but has become "smoother." Figure 3 shows the observed result of the square wave from the PWM signal going into the low-pass filter at about 50% duty cycle. Hooking up the sparkfun audio amplifier, the speaker is now much louder. Figure 3 below shows how I wired up the design.

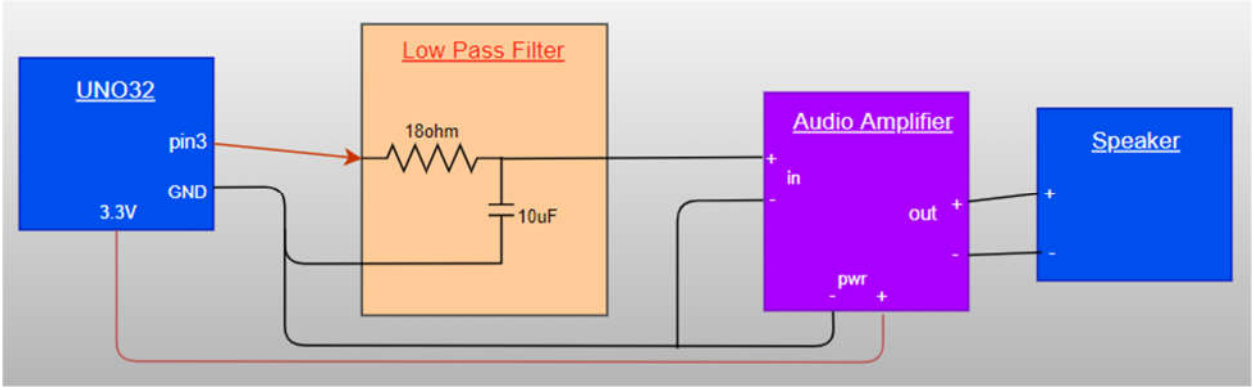

Fig 4: Circuit Diagram for Part 5 and 6

I chose not to implement a potentiometer to control the amplification(volume) from the audio amplifier. This causes the volume pins to act like a short allowing the signal to be amplified to the maximum that is allowed.

Part6 – Make Some Music

Method:

Part 6 is like part 5 but now buttons are involved. To read button states, there is a macro written in the BOARD.h file that allows us to grab the states of all four buttons at once. To be able to grab the state of each button individually, bit-masking is involved. I denoted each button to have a tone that is a frequency of 50 apart starting from 150 till 300. The potentiometer is used to offset these values. Because we are utilizing the on-board buttons, there is no extra circuitry involved and the setup is the same as part 5.

Result:

Each button represents a different tone. Potentiometer alters the tones but a certain offset. When a button is not pressed, there is no sound.

Conclusion

This lab is a good introductory lab allowing us to dive into hardware and microcontroller without much headache. Each part gives us enough information to figure out the next parts providing a good linear approach. This lab covered how to control a speaker through buttons and potentiometers and filtering out unwanted sounds. Fun lab to start the quarter off.