Leonid Shuster

ECE 167

1/12/20

Lab Report 0

**Introduction**

In this lab, we were tasked with familiarizing ourselves with the Uno32 microcontroller, as well as learning how to generate a tone through the Uno32 and make it sound better using software filtering or a low-pass filter. In the first part, we simply printed “Hello World” from the microcontroller onto the terminal. In the second part, we read the value of the potentiometer on the Uno32 and displayed its value onto the Uno32’s display screen. In the third part, we produced a tone using the Uno32 and audio amplifier and speaker, which we then modified in the fourth part to change based on the potentiometer’s value, and cleaned the sound using software filtering. In the fifth part, we cleaned the sound using a low-pass filter, and put it all together in the sixth part to create music using the buttons on the Uno32.

**Part 1: Hello World!**

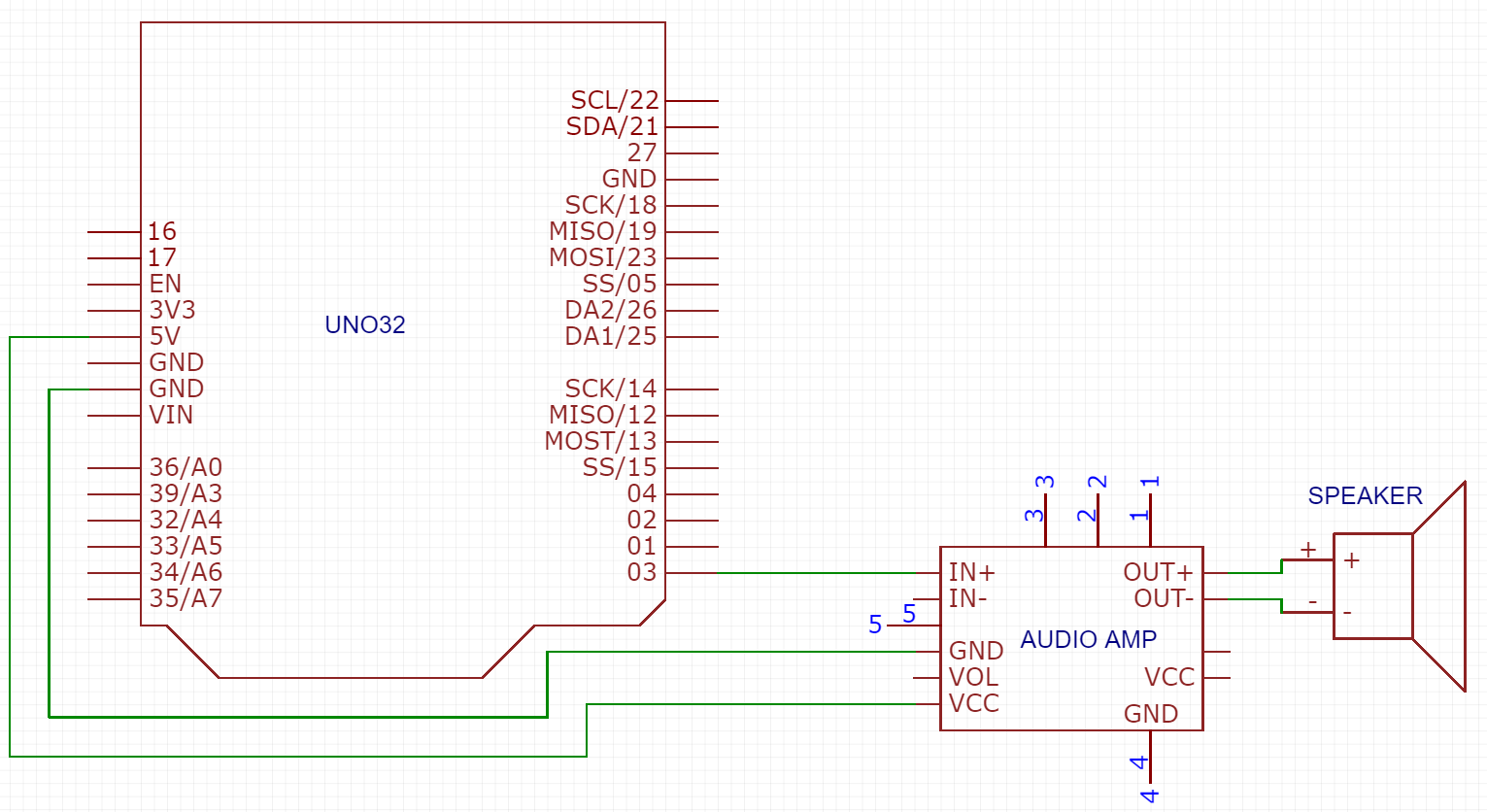
In the first part of the lab, we were told to print “Hello World” via serial communication from the Uno32 to the ds30 terminal. After burning the ds30 bootloader onto the Uno32, in Lab0\_Part1Main.c, I printed “Hello World” using printf(), ran the program, and saw that “Hello World” was printed on the ds30 terminal as expected.

**Part 2: Hello A/D (Reading an analog signal)**

In the second part of the lab, we were told to display the value of the potentiometer on the Uno32. To do so, in Lab0\_Part2Main.c, I added the analog pin A0 which corresponded to the potentiometer, and in a while loop, I read from the analog pin using the AD\_ReadPin() function from the AD library, put the value of the potentiometer into a string using the sprintf() function from the stdio library, and used the OledClear(), OledDrawString(), and OledUpdate() functions from the Oled library to display it onto the onboard display screen.

One issue I had at first was that nothing was being displayed on the screen. I found the problem to be that I was calling BOARD\_Init() after AD\_Init() and OLEDInit(), which caused the entire board to reinitialize everything including whatever AD\_Init() and OLEDInit() did. After putting BOARD\_Init() first, I was able to finally see something on the screen. Another issue I ran into was that I didn’t add the analog pin corresponding to the potentiometer, so I was getting an error reading from it. This issue was fixed by simply calling AD\_AddPins() on that analog pin which allowed it to be read from.

**Part 3: Tone out speaker hard coded** In the third part of the lab, we were tasked with generating a tone from the Uno32 through an amplifier and speaker. On the outside of the Uno32, I connected the Uno32’s 5 V and ground pins to the amplifier’s power and ground, as well as pin 3 of the Uno32 to the amplifier’s In + pin since that was the PWM pin the tone generator library controlled. The amplifier’s In - pin was connected to ground, while its Out + and Out - pins went to the speaker’s + and - pins (see Figure 1).



*Part 3 External Schematic*

In Lab0\_Part3Main.c, I used the ToneGeneration\_SetFrequency() function from the tone generation library and gave a certain frequency as the argument for it in order to produce a tone at that frequency.

**Part 4: Tone adjusted via POT**

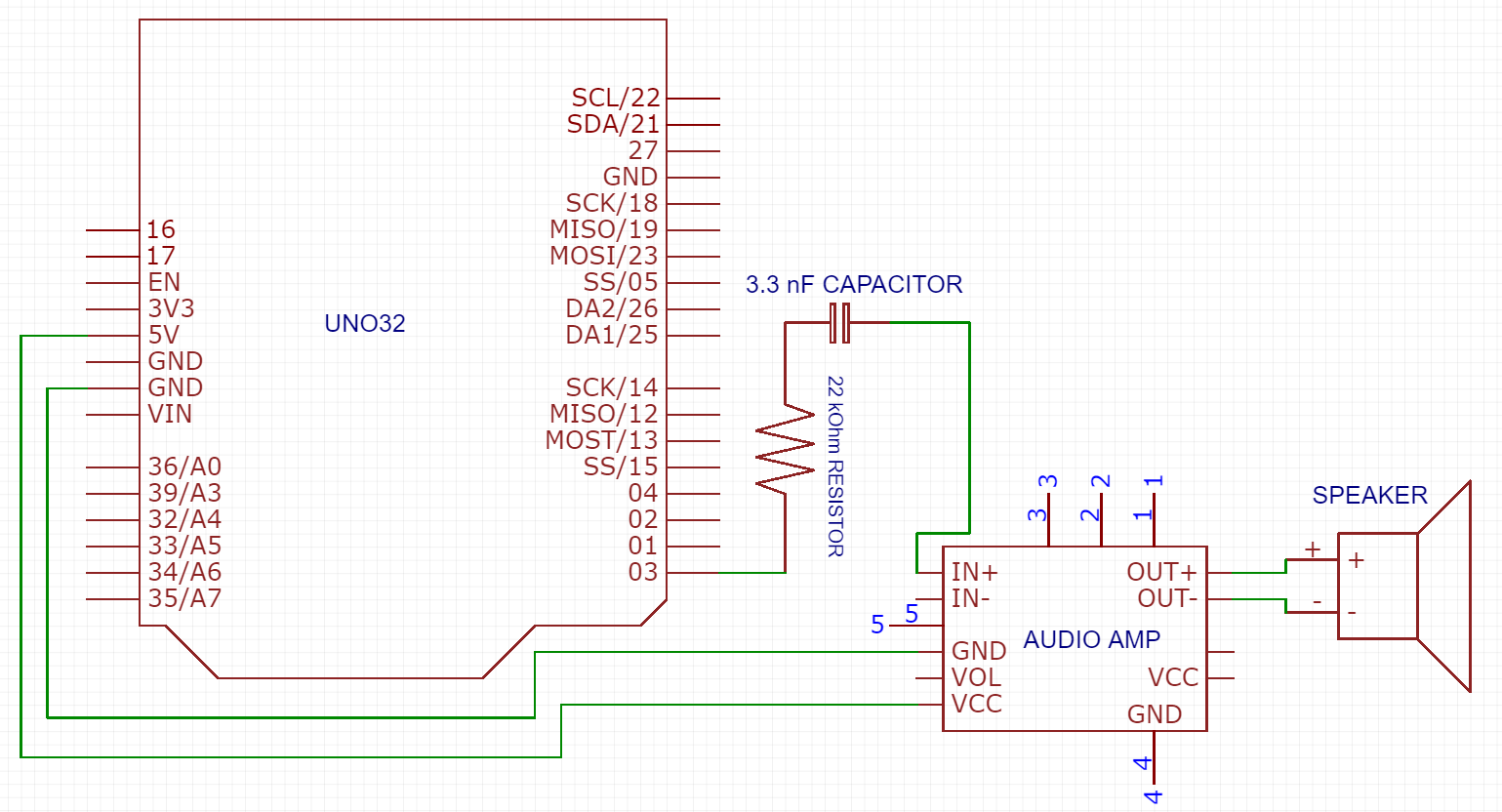
In the next part of the lab, we were tasked with using the Uno32’s onboard potentiometer to change the tone coming from the Uno32 into the amplifier and speaker, as well use software filtering to make it sound better. The external setup was the same as the previous part.

In Lab0\_Part4Main.c, I added the analog pin corresponding to the potentiometer just like in Part 2. In a while loop, I read the value of the potentiometer from the analog pin just like before, and gave the value of it as an argument to ToneGeneration\_SetFrequency() so that changing the value of the potentiometer directly produced a different frequency. However, changing the frequency with the potentiometer sounded scratchy and bad because the potentiometer’s value kept minorly changing due to debouncing when not moving. In order to fix this, I implemented software filtering by only changing the frequency being produced by the Uno32 if the potentiometer’s value had changed by more than a value of 2. By doing so, the frequency stayed the same until the potentiometer had actually had a significant change in value rather than from debouncing.

**Part 5: Basic filtering of output (single pole RC to smooth tone generation)**

In the next part of the lab, we were tasked with again using the Uno32’s onboard potentiometer to change the tone coming from the Uno32 into the amplifier and speaker, except we had to use a low-pass filter to make the tone sound better instead. The speaker could not play frequencies higher than around 1000 Hz, so I chose the cutoff frequency to be around 2000 Hz so that it would be a little higher than the actual desired cut off frequency and less than the maximum frequency of 36 kHz. Using the formula for cutoff frequency, where fc is the cutoff frequency, R is the resistor value and C is the capacitor value, I found R to be 22 kOhm and C to be 3.3 nF.

On the outside of the Uno32, the setup was the same as the previous two parts with the addition of a low-pass filter which I implemented by adding the 22 kOhm resistor and 3.3 nF capacitor in between the Uno32’s pin 3 and amplifier’s In + pin (see Figure 2).



*Part 5 External Schematic*

In Lab0\_Part5Main.c, I used the same code from Part 4 to produce a tone based on the value of the Uno32’s potentiometer.

With the low-pass filter, the sound of the speaker sounded cleaner and more clear, as well as quieter. When observing the square wave being produced, I noticed that the low-pass filter caused less distortion compared to without it.

**Part 6: Make some music**

In the last part of the lab, we had to put everything together by using the Uno32’s potentiometer and buttons, the amplifier and speaker, and a low-pass filter to make some music. Each one of the four buttons on the Uno32 produced a certain tone, and depending on the value of the potentiometer, that tone was offset by that amount. The external setup was the same as the previous part.

In Lab0\_Part6Main.c, I added the analog pin corresponding to the potentiometer just like before, and in a while loop, I read its value from the analog pin. Next I checked which button was being pressed down by using the function BUTTON\_STATES() from the board library. BUTTON\_STATES() returns a four bit number with each bit representing the state of each button, so to check if a button was being pressed down, I ANDED the value of four different numbers with the four bit number to see which button was being pressed. If a certain button was pressed, a tone was produced at a certain frequency plus the offset from the potentiometer. If none of the buttons were being pressed, no tone was produced.

One of the problems I ran into was that I was trying to AND BUTTON\_STATES() with the four different buttons found in BOARD.h: BTN1, BTN2, BTN3, and BTN4. However, when I tried this, only my first button was working, while the others did not do anything. I found the problem to be something with PORT D, but could not figure out how to solve it. What I did instead was just AND BUTTON\_STATES() with a number representing the same thing: 0x01, 0x02, 0x04, and 0x08.

**Conclusion**

After having gone through the lab, I feel like I fundamentally understand how to work with the Uno32 microcontroller and its potentiometer, PWM, and buttons, as well as how to produce tones with the Uno32 and an amplifier and speaker. From printing “Hello World” in the first part, reading the value of the Uno32’s potentiometer in the second part, generating a tone in the third part, changing the value of the tone with the potentiometer and filtering it using software in the fourth part, filtering the same tone using a low-pass filter in the fifth part, and finally combining it all together to produce music in the sixth part, I feel confident that I can effectively use the Uno32 to produce clean desired tones. If I were to do this lab again, I would further play around with how the sound changes when adding software filtering and a low-pass filter.