Leonid Shuster

CSE 121

Lab Section 1C

11/11/19

Lab Report 4

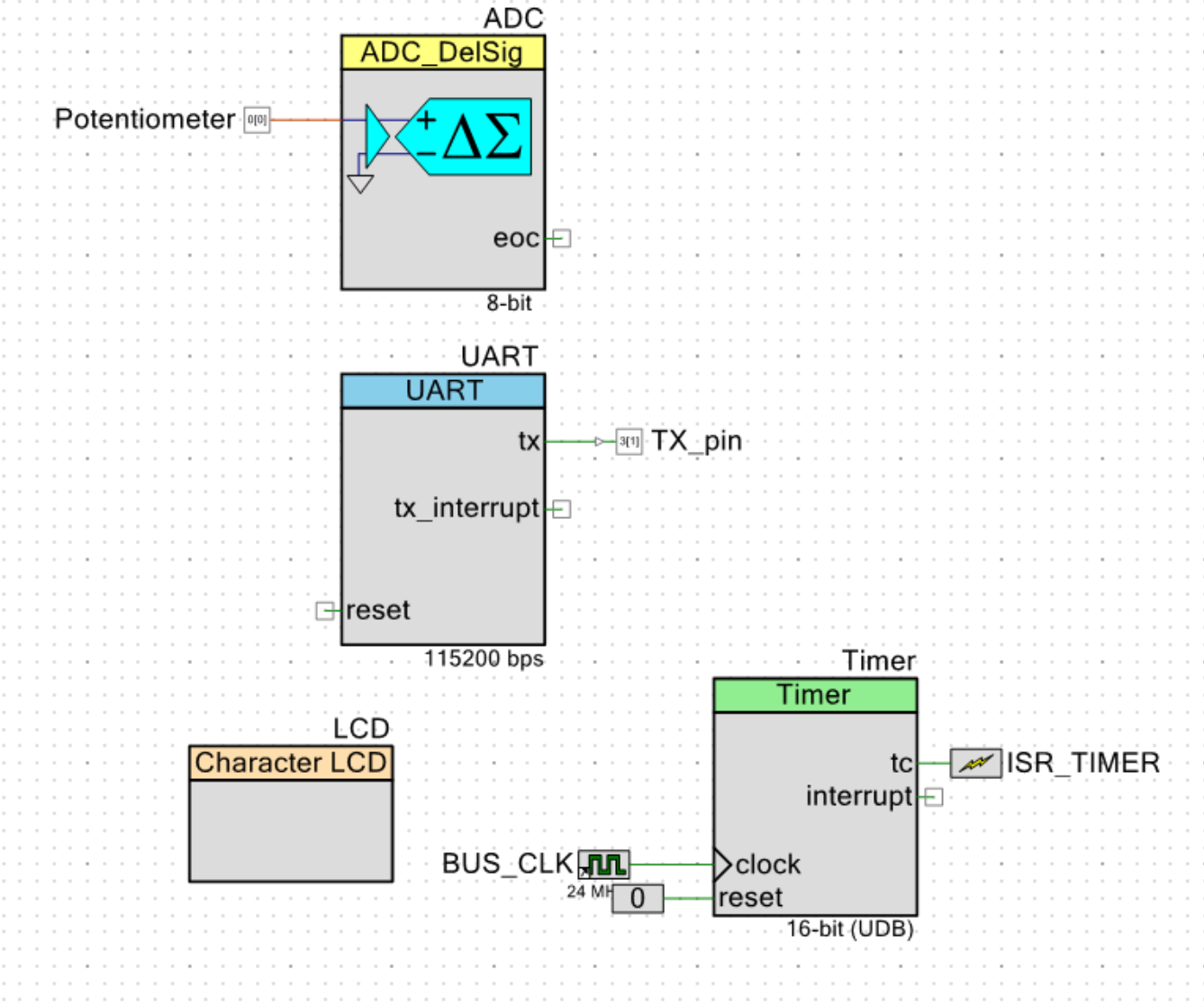
**Introduction**

In this lab, we were tasked with learning how to use the Raspberry Pi hardware and programming environment by connecting it to the PSoC 5 and performing various actions. In the first part, we connected a potentiometer to the PSoC which was used to control the brightness of an LED that was connected to the Raspberry Pi. And in the second part, we sampled an analog signal with the PSoC and sent it over to the Raspberry Pi which sent it back to the PSoC and reproduced the original signal.

**Part 1: Remote Control of LED brightness with PWM**

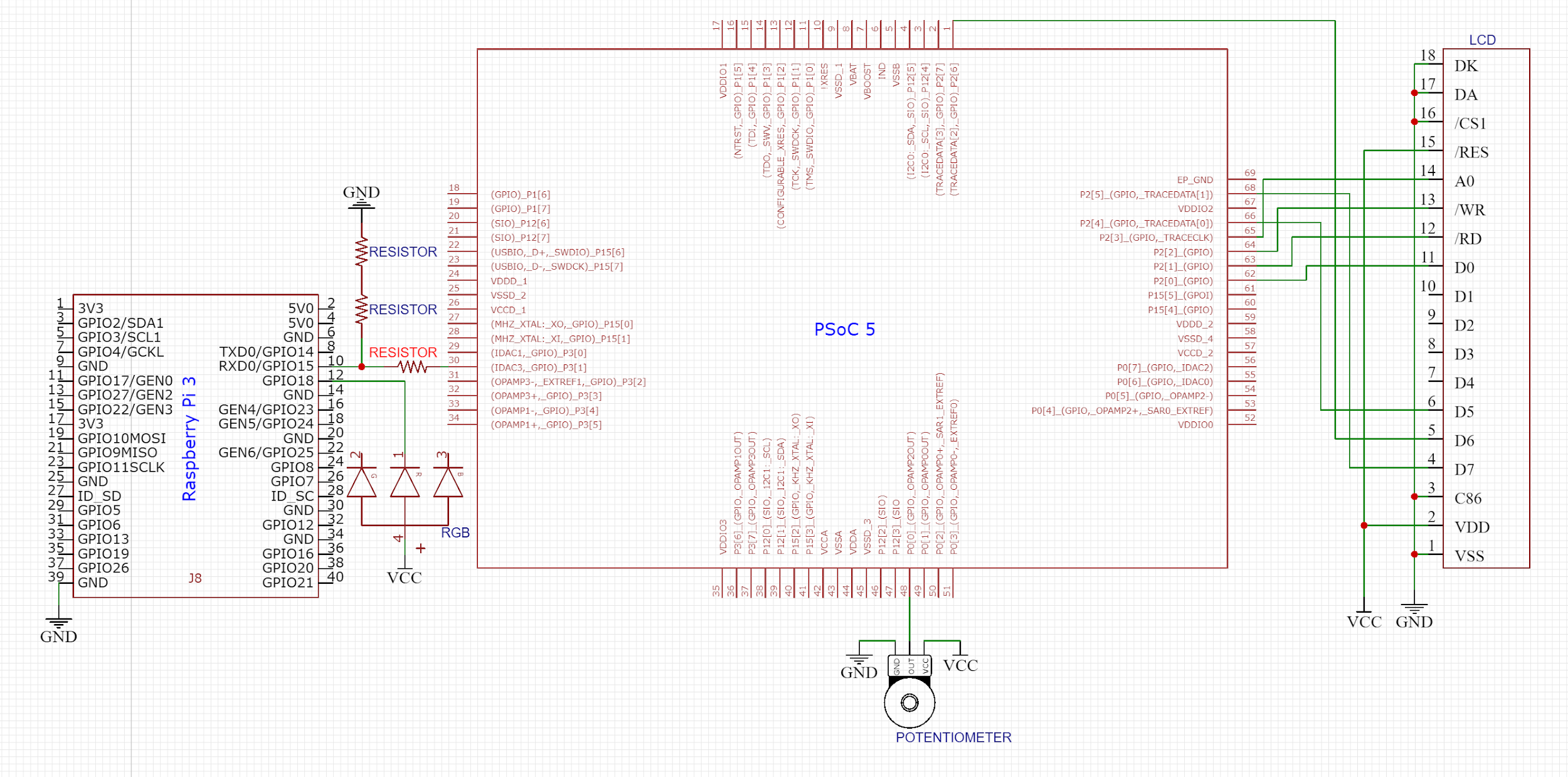
In the first part of the lab, we were told to use a potentiometer to control the brightness of an LED connected to the Raspberry Pi. However, because the Raspberry Pi does not have any built-in analog blocks and could not receive any analog input, we had to use the PSoC to read in data from the potentiometer, convert it to a digital value using an ADC, and use a UART to send the data over to the Raspberry Pi which would then use its own UART to read in the data and a PWM to control the brightness of the LED.

I worked on the PSoC side first, and in my top design, I had an ADC connected to an analog input pin, a UART with its TX output connected to an output pin, a timer that went off every millisecond and triggered an interrupt that would be used to sample the ADC and sent its data to the UART TX FIFO, and an LCD (see Figure 1).



*Figure 1: Part 1 Top Design*

On the outside of the PSoC and Raspberry Pi, I connected a potentiometer to the PSoC input pin going to the ADC and an external wire from the PSoC UART TX output pin to the Raspberry Pi UART RX input pin. Because the PSoC outputs 5V while the maximum the Pi can work under is 3.3V, I had to add a voltage divider to connect the PSoC and Pi. Finally, I had an LED connected to the PWM output pin of the Pi and an LCD to the PSoC (see Figure 2).



*Figure 2: Part 1 External Schematic*

In main.c of the PSoC, I set up the timer interrupt to read a 16-bit value from the ADC into a 16-bit integer, checked if the number was less than 0 or greater than 255 and kept it in those bounds, and then sent the value to the UART TX FIFO.

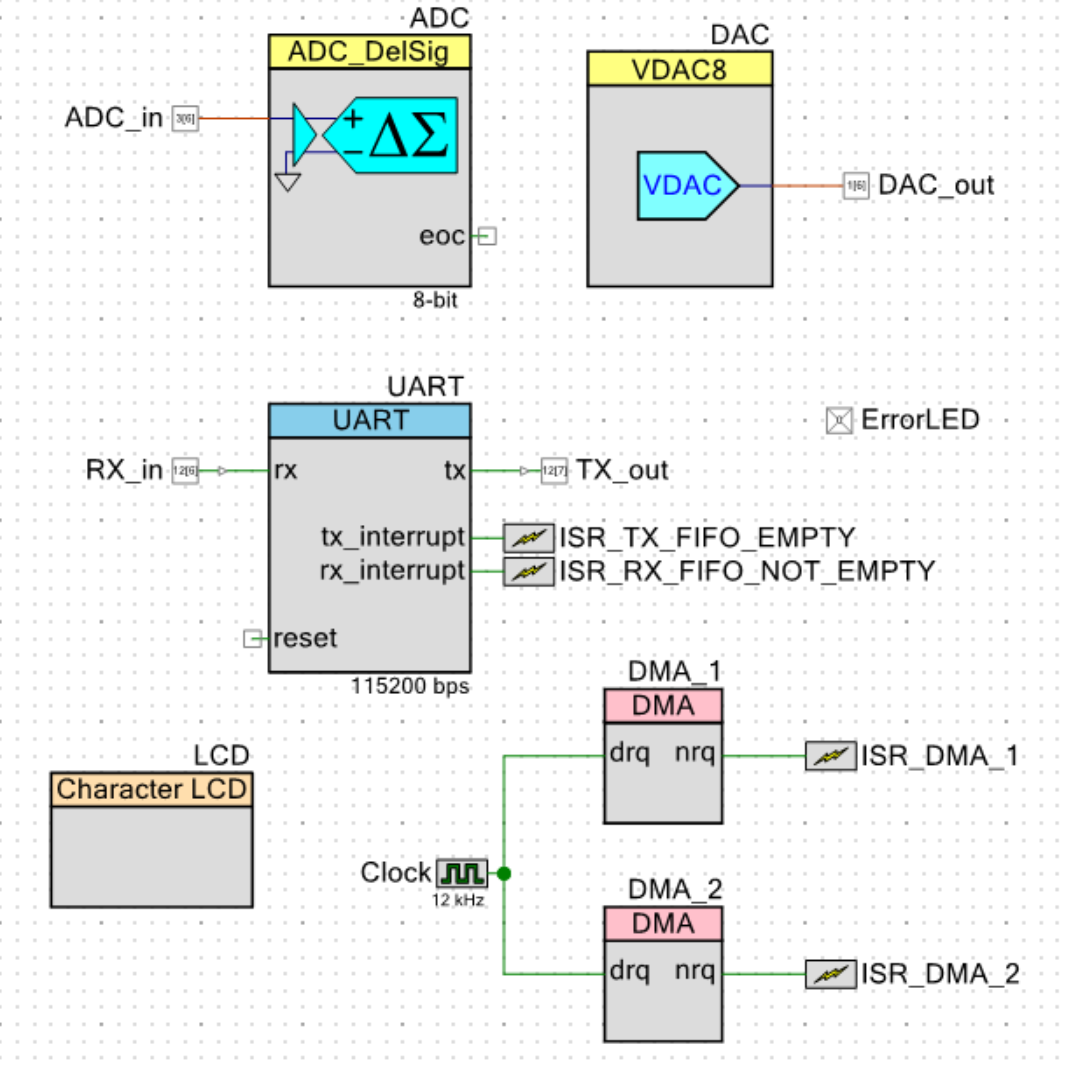
Meanwhile, in main.c of the Raspberry Pi, I used the example code provided by the professor to set up the Pi’s UART and wiring and pin configuration, and used an endless for loop to continuously read a value from the RX FIFO, scale the value, and write it to the Pi’s PWM.

When I first ran the program, the value I was reading in the Pi from the PSoC was much higher than the value I was expecting, which was supposed to be somewhere between 0-255. This issue was fixed by reading in the value from the PSoC as an unsigned 8 bit integer. Also, the range of the PWM is 0-1023, but I made the mistake of only multiplying the value by 4 and got a range of 0-1020, and did not account for the last three values, which meant my LED did not completely turn off. If I correctly scaled the value to include the range of 0-1023, my LED would’ve completely turned off.

**Part 2: Analog Loopback through the Raspberry Pi**

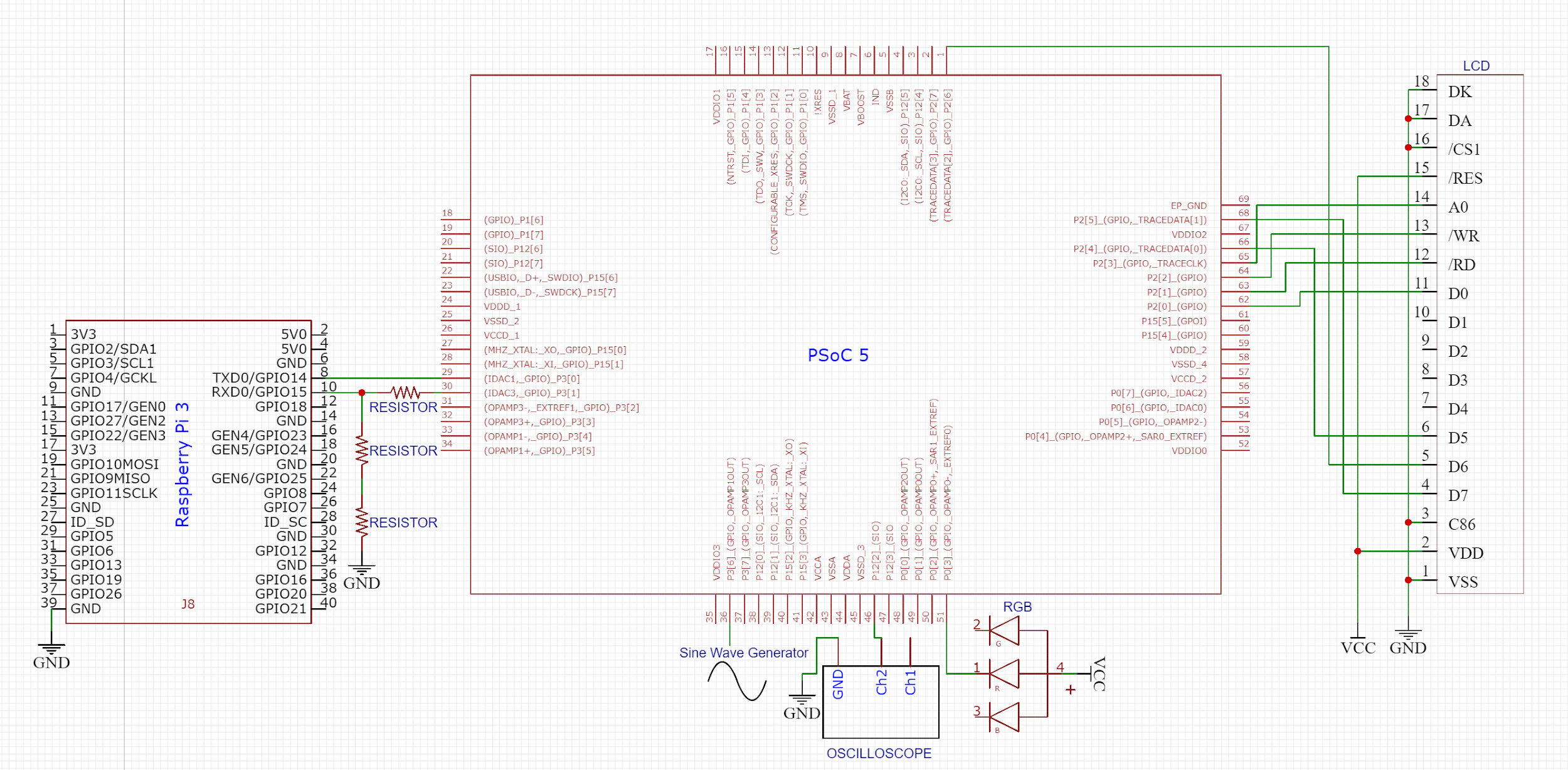
In the final part of the lab, we were told to sample an analog signal with the PSoC, convert the data to digital using an ADC, send it over to the Raspberry Pi, and then send it back to the PSoC, which would then convert the data back to analog using a DAC and display the original analog signal. Because we had to process, send, and receive a continuous stream of data, we also had to make use of ping-pong buffers and DMA transfers to ensure that no data was lost when transmitting and receiving the data.

I again started on the PSoC side, and in my top design, I had an ADC connected to an analog input pin and a UART with its TX output connected to an output pin as well as two interrupts when the TX FIFO was empty and the RX FIFO received a byte. I also had two DMAs that used the same clock for their hardware requests and their own interrupts when a transfer was completed, a DAC connected to an output pin, an output pin for an error LED, and an LCD (see Figure 3).



*Figure 3: Part 2 Top Design*

On the outside of the PSoC and Raspberry Pi, I connected the AD2 pattern generator to the PSoC input pin going to the ADC, an external wire from the PSoC UART TX output pin to the Raspberry Pi UART RX input pin with a voltage divider, another wire from the Pi TX output pin to the PSoC RX input pin, an LED to a PSoC output pin, an oscilloscope to another PSoC output pin, and the LCD to the PSoC (see Figure 4).



*Figure 4: Part 2 External Schematic*

In main.c of the PSoC, I started by just trying to transmit data through the UART TX correctly. The goal was to continuously process data from the ADC to the TX FIFO without losing any data, which could be done using a DMA and two ping-pong buffers. While one buffer was being filled with data from the ADC, the other would be used to send data to the TX FIFO. First, I created two 64-byte character arrays that would be used as ping-pong buffers, and in the main part of my program, I configured a DMA with two descriptors in a loop such that each descriptor would transfer 64 bytes to one of the buffers and then trigger an interrupt when it was done. Within the interrupt procedure, I had a flag that changed every time the interrupt was triggered in order to decide which array would be used to send data to the TX FIFO. Another interrupt would trigger when the TX FIFO was empty, and in the interrupt procedure, I checked what the flag was set to in order to decide the correct array to send data from, and then had a while loop that kept sending data from that array to the TX FIFO until it was full.

After verifying that the correct data was being transmitted, I set up the receiving end of the UART. Similarly to transmitting, I had two more ping-pong buffers, another DMA, and an interrupt from the UART RX FIFO when a byte was received. I configured the second DMA in the same way as the first one by having two descriptors in a loop, except now the DMA transferred data from one of the buffers to the DAC, and triggered another interrupt that set a different flag every time in order to decide which array would be used to receive data from the RX FIFO. When the RX FIFO received a byte and triggered an interrupt, it checked the flag to decide the correct array to send data to, and had a while loop to send data from the RX FIFO into the array until it was not empty.

I then connected the TX output and RX input and tested whether I could reproduce the analog signal from the AD2. At first, the DMA was not working properly and I was not getting any output from the DAC, but when I changed how my interrupts worked, I verified that I was reproducing the signal, although it was very noisy. The next step, which I didn’t get to, was to then make it so the PSoC would transmit data to the Raspberry Pi, and then the PSoC would receive it back from the Pi. If I had gotten a chance to do it, in main.c of the Pi I would follow a similar procedure as before where I would have two arrays that would transmit and receive data from the TX and RX FIFOs by checking the state of the FIFOs and having a while loop that would transmit or receive data.

**Conclusion**

After having gone through the lab, I feel like I fundamentally understand how to use the Raspberry Pi to transfer and receive data with another machine. From sampling a potentiometer’s value with the PSoC and sending its value to the Pi to light up an LED in the first part and reading in an analog signal with the PSoC, sending it to the Pi, and then back to the PSoC to reproduce it in the second part, I feel confident that I can effectively use the Pi to communicate with other machines. If I were to do this lab again, I would test whether the data the PSoC was transmitting was the same data it was receiving even more extensively, because even though the signal I was reproducing was similar to the initial analog signal, it was very noisy and had random glitches.