Leonid Shuster

CSE 121

Lab Section 1C

10/28/19

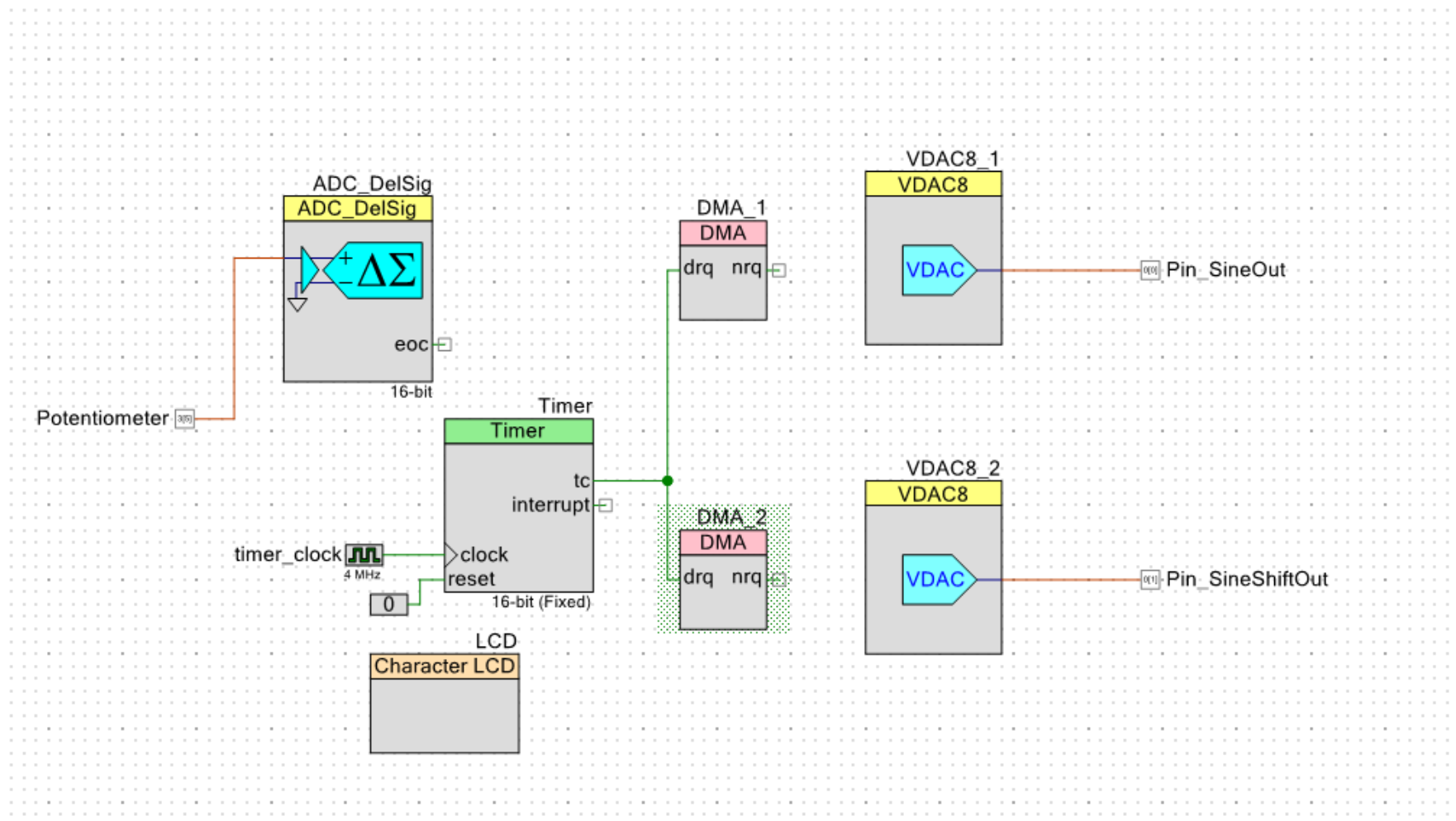
Lab Report 2

**Introduction**

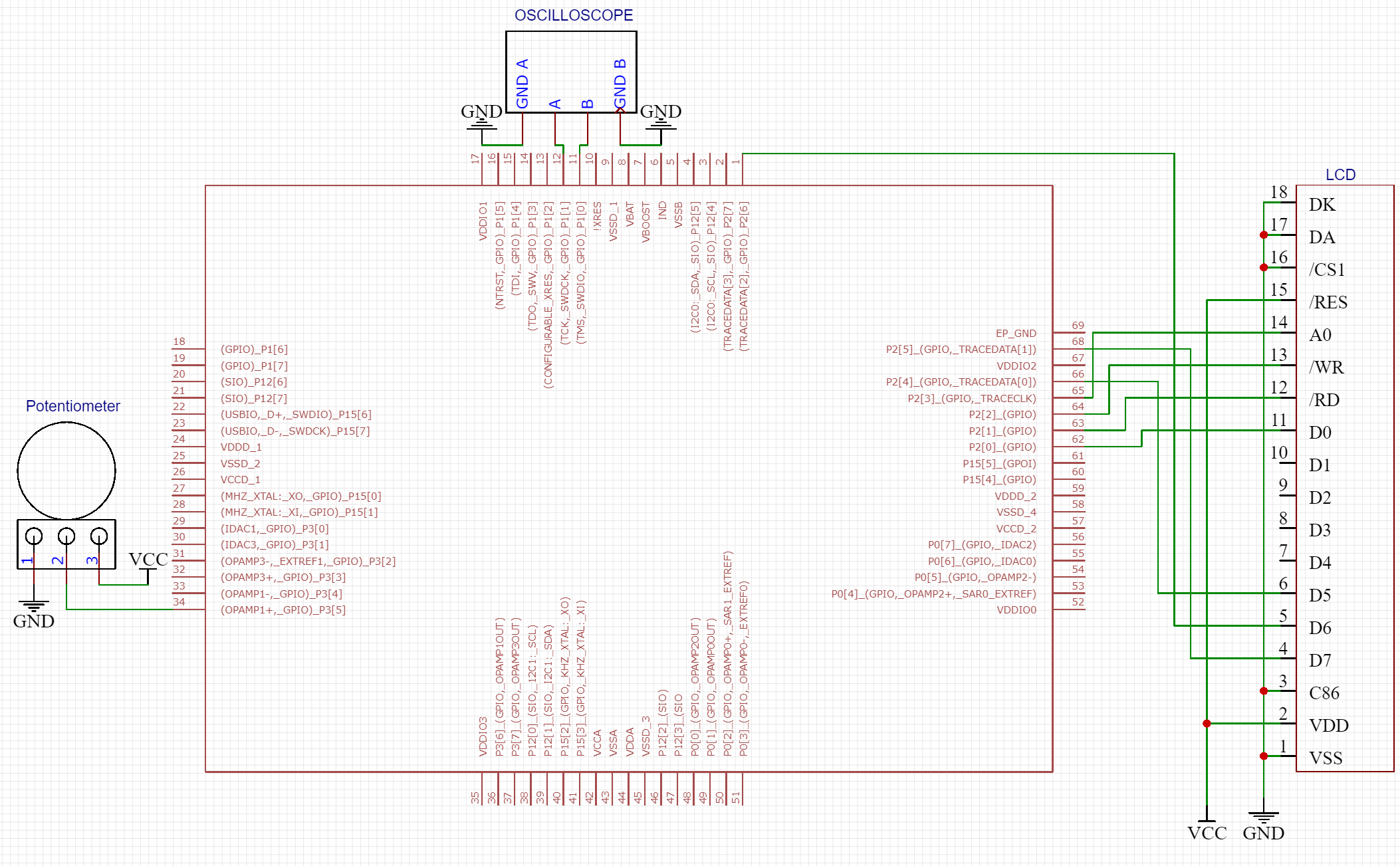
In this lab, we were tasked with learning how to use the DMA hub in the PSoC 5. In the first part, we created a dual-channel waveform generator in which one of the waves had a phase shift that was controlled by a potentiometer. And in the second part of the lab, we used the DMA hub to perform a memory-to-memory block transfer, which we compared with a software loop that carried out the same function.

**Part 1a: Dual-Channel Waveform Generator**

In the first part of the lab, we were told to use a lookup table stored in flash memory to create two sine waves on two output pins with one of the sine waves’ phase shift being controlled by a potentiometer which would be displayed in degrees on an LCD display. To begin, I looked at Example 3 in the Cypress Application Note *Introduction to DMA* which explained what type of transfer we needed to do, an array-to-point transfer, and used the schematic and source code provided in the example as the basis for my design. For my top design schematic, I had an analog pin for the potentiometer connected to an ADC which converted the potentiometer’s position and value from an analog signal to an internal digital one. I also added two DMAs that were used to transfer the value from the potentiometer to two VDACs, which then outputted the two sine waves on two output pins. Finally, I had a timer connected to both DMAs that could be used to change the frequency of the sine waves, and an LCD to display the value of the phase shift (see Figure 1).



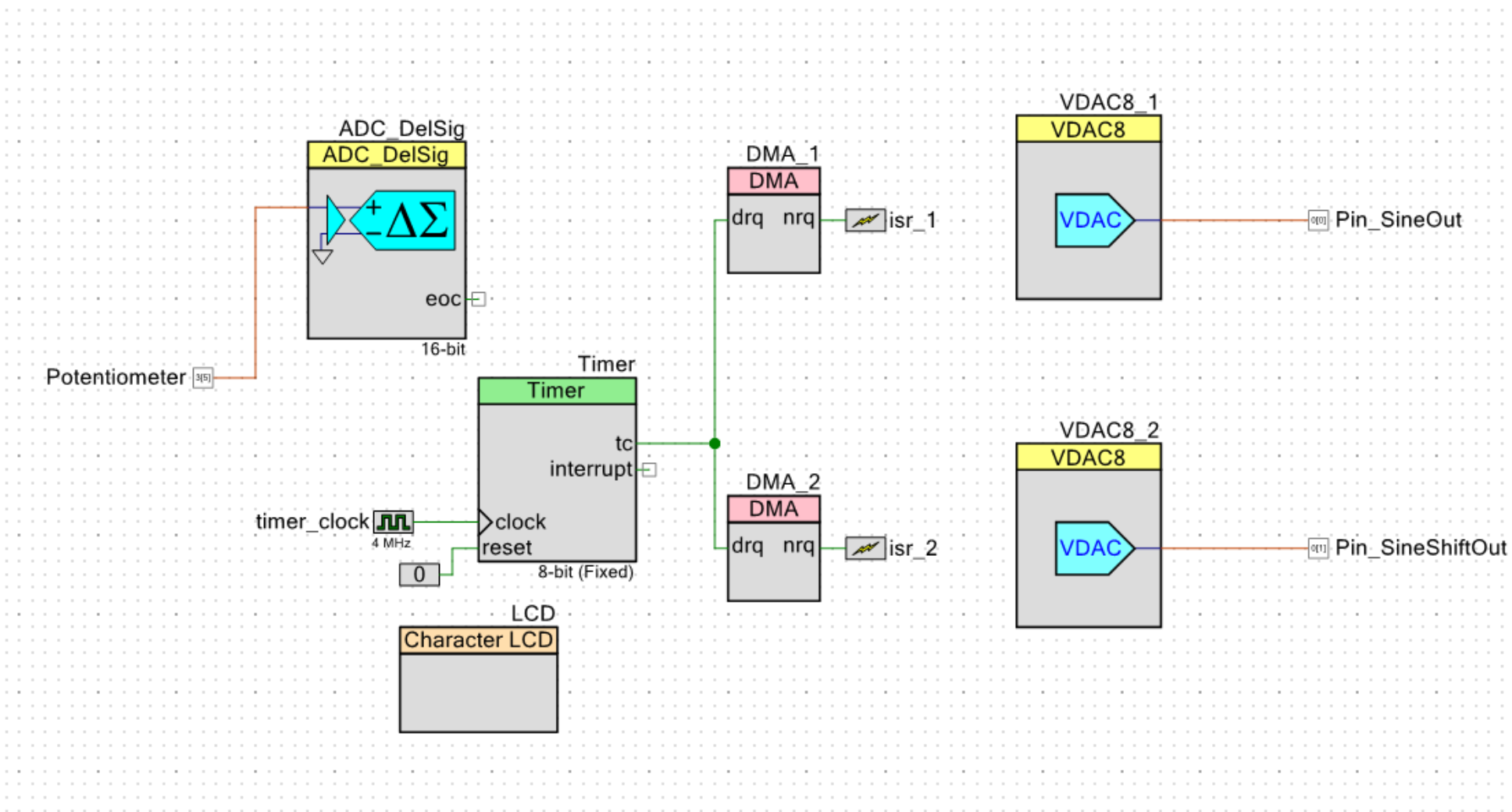
*Figure 1: Part 1a Top Design Schematic*

In main.c, I followed the source code from the example to setup the DMAs. First, I created a lookup table with points located in flash memory that would be used for the sine waves. I then initialized a DMA channel for each DMA, and allocated transcription descriptors (TDs) for both of them. Next, I configured the TDs to transfer a certain amount of bytes and point back to itself, and then set the source address and destination address of each DMA as the lookup table and the two VDACs. Finally, I mapped the TDs to the DMA channels and enabled them. In an infinite loop, I continuously updated the source address of one of the DMAs with a value from the lookup table depending on the position of the potentiometer, resulting in a varying sine wave. On the outside of the PSoC, I connected a potentiometer to the pin being used as an analog input pin, two wires connected to an oscilloscope to the pins outputting the two sine waves, and an LCD to their proper pins as stated in the lab manual (see Figure 2).

*Figure 2: PSoC 5 connected to potentiometer, oscilloscope, and LCD*

**Part 1b: Dual-Channel Waveform Generator with Controlled Phase Change**

In the next part, we were tasked with creating a design similar to the previous part, except that the phase of one of the sine waves would only change at its zero-crossing points. I couldn’t figure out how to detect zero-crossing points, so I wasn’t able to finish the lab. My top design schematic was the same as the one from the previous part, except I connected interrupts to the DMAs in an attempt to detect when the DMA was finished transferring its data to the VDACs (see Figure 3).

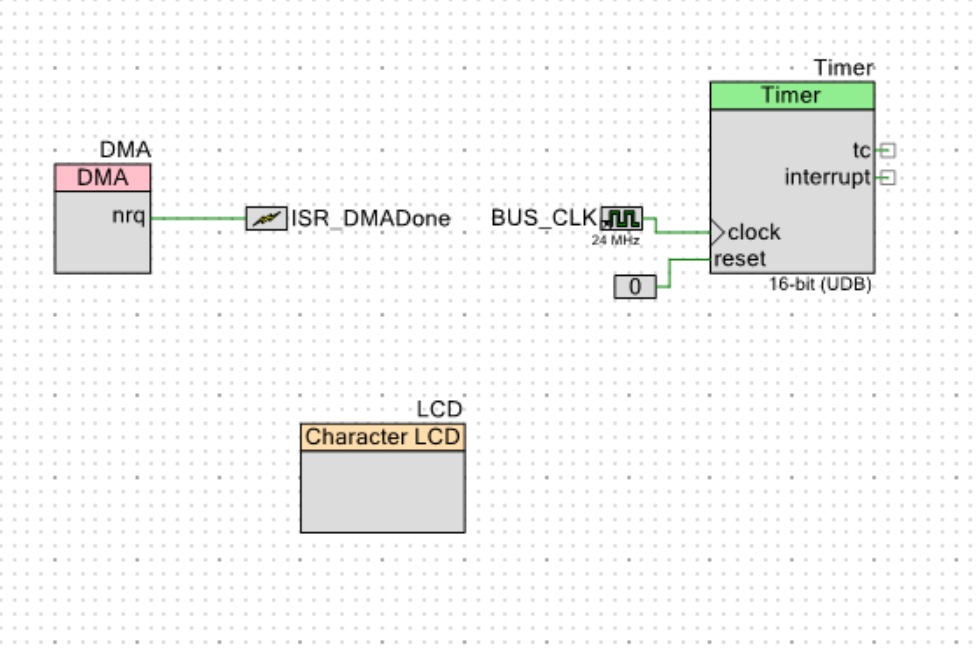


*Figure 3: Part 1b Top Design Schematic*

My main.c was the same as from the previous part, except that I tried to configure the interrupts to stop the VDAC from outputting its updated value, but wasn’t successful in doing so. On the outside of the PSoC, my setup was also the same as from the previous part.

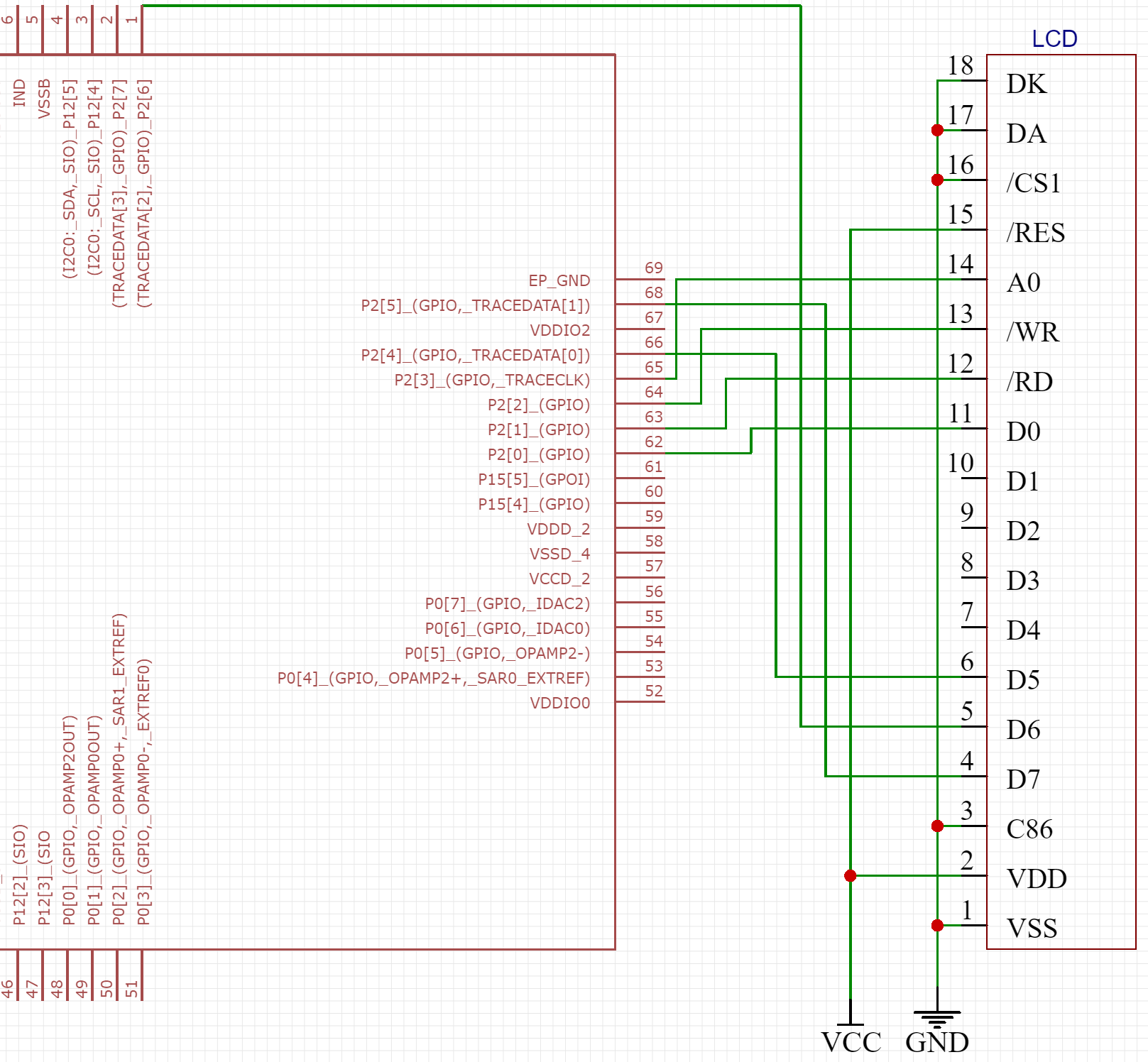
**Part 2: Memory-to-Memory Block Transfer**

In the final part of the lab, we were tasked with using the DMA to transfer data from one part of the RAM to another, and display the amount of errors and the time the transfer took onto an LCD. To start, I looked at another example from *Introduction to DMA* that described how to perform a memory-to-memory block transfer, and used the schematic and source code as a starting point. For my top design schematic, I had a single DMA that was connected to an interrupt that triggered when the DMA was done transferring data. I also had a timer that counted how long the transfer took, as well as an LCD (see Figure 4).



*Figure 4: Part 2 Top Design Schematic*

In main.c, I created a source array with an increasing pattern of byte values and a cleared destination array that would be used by the DMA for transferring data. I setup the DMA similar to how I did in the first part, except that because we had to transfer 16,384 bytes and each TD could only transfer a maximum of 4095, I had to use 5 TDs, with each TD pointing to the next one, other than the last one which did not point to anything. Finally, I set the source address and destination address of the DMA to the source array and destination array. I then began the timer and started the DMA transfer, which triggered an interrupt when it finished. Within the interrupt, I stopped the timer and printed out the amount of time the transfer took. Afterwards, I ran a for loop that checked whether the values from the source array matched those in the destination array, and printed out the number of mismatches. We were then told to replicate the same process of transferring data but by means of a software loop instead of a DMA. To do so, I followed the same process of creating a source array with an increasing pattern of byte values and a cleared destination array, but I used a for loop instead of a DMA to transfer the data, with a timer that started and stopped before and after the for loop. When comparing the two methods, using a DMA took about 436 ms, whereas doing it through software took ten times as long with a time of 4036 ms. I believe it took much longer through software because the PSoC has dedicated hardware and channels for transferring of data, whereas doing it through software required that the CPU be involved. In terms of how the outside looked, I only had the PSoC connected to the LCD (see Figure 5).



*Figure 5: PSoC 5 connected to LCD*

**Conclusion**

After having gone through the lab, I feel like I fundamentally understand how DMAs work now. From the first part of the lab where we created a dual-channel waveform generator that utilized an array-to-point transfer to the second part in which we performed a memory-to-memory transfer, I feel confident that I can effectively use DMAs and integrate them into my future designs. If I were to do this lab again, I would go through more DMA examples provided by *Introduction to DMA* in order to become more familiar with DMAs and gain a stronger understanding of them.