Anthony White
Oct 18, 2021
CSE 121

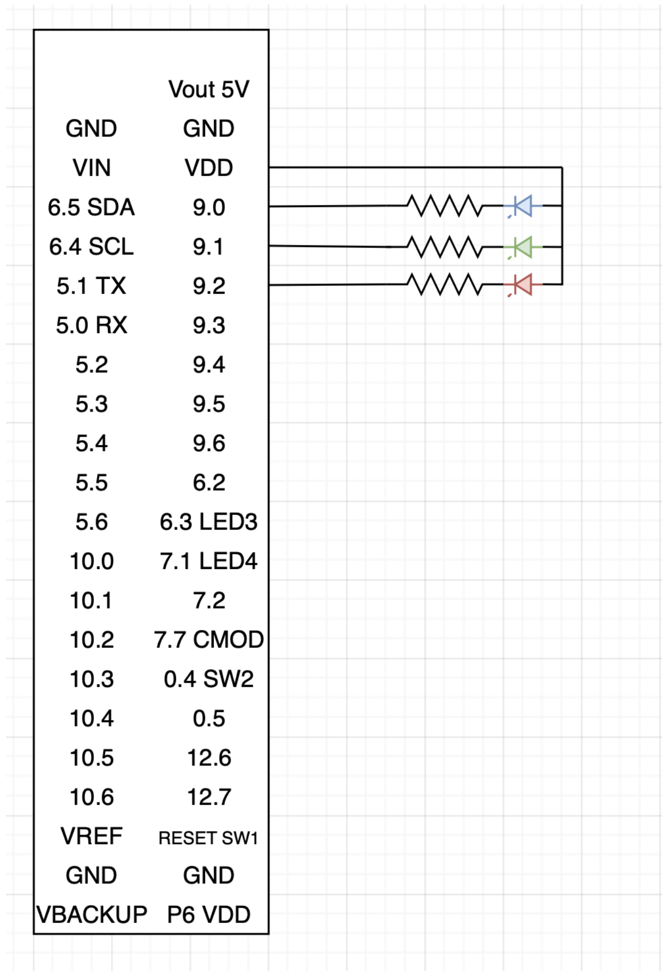
Lab 1 Report

Introduction:

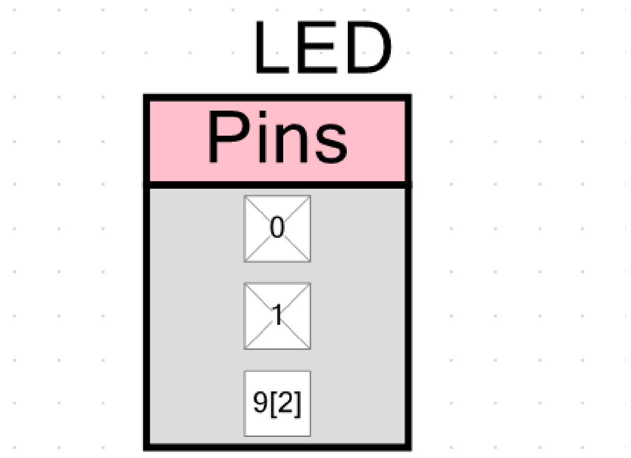
The purpose of this lab was to familiarize ourselves with the PSoC 6 environment, and with the hardware components provided to us for lab use. Part 1 focused on testing the performance of each core, and Control Register vs. Component-based API to turn an LED on and off as fast as possible. Part 2 aimed at using a Potentiometer to control the pulse width to control the brightness of our LED. Part 3's focus was on using a Proximity Sensor to measure the distance of an object. This was used to Trigger an LED given an object was in a certain range. Part 4 was aimed at using a Wave Generator to generate different frequency waves and an LCD to display our calculated values for that frequency using two different methods for calculation.

Part 1:

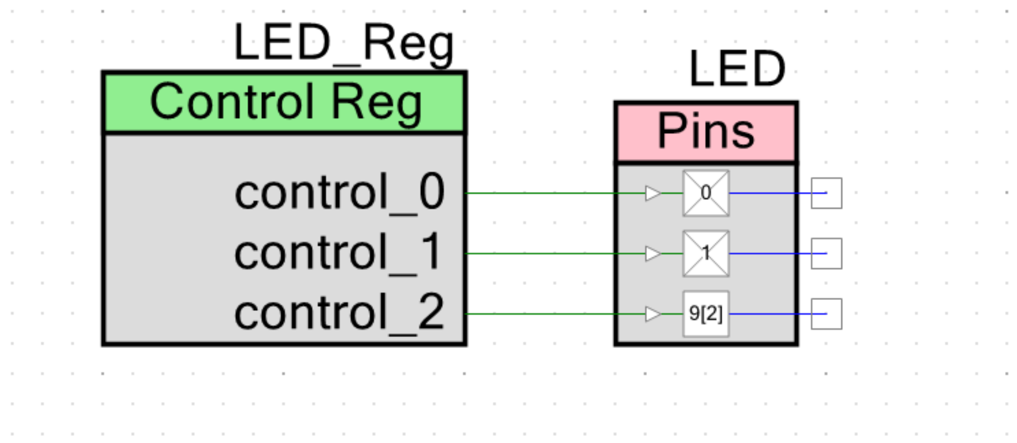
External Schematic



Component Based API Top Design:



Control Register Top Design:

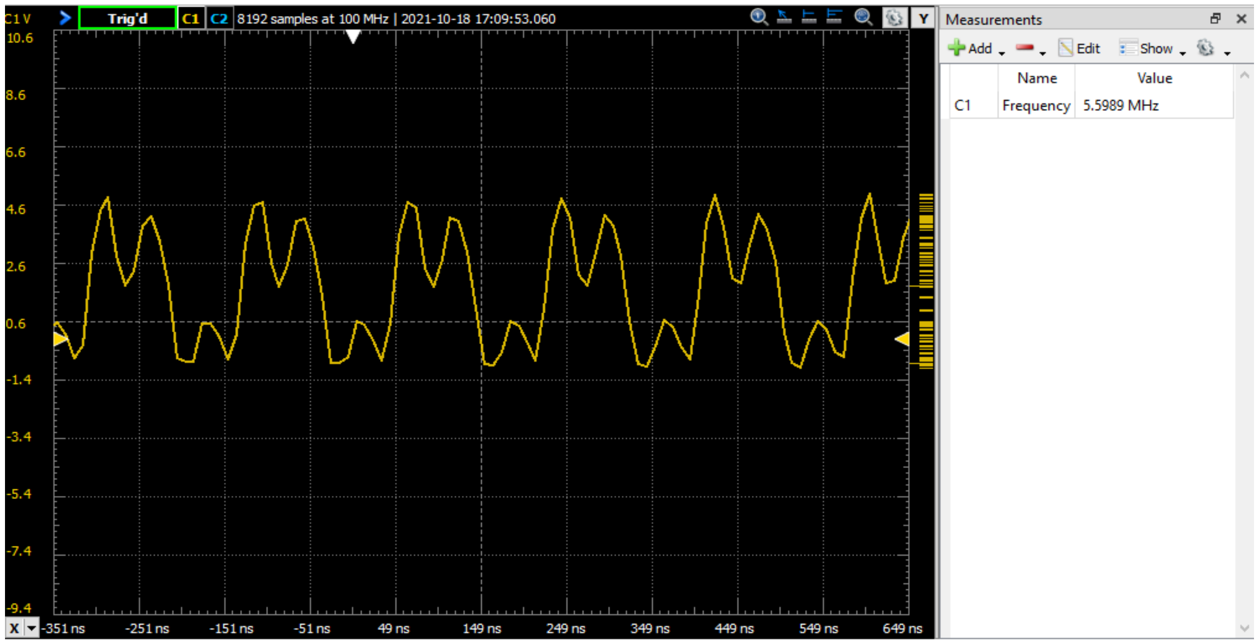


The Design for this part was straight forward. I connected the LED to Power, and three pins using 1k resistors. For the Component-based API implementation of this part, one of the LED pins was called with a write function directly and switched on and off continuously for both Core 0 and Core 1. For the Control Register implementation, a Control Register was created to control the LED pins. The initial values were set to 1, because I want the initial state of the LED to be off and the LED's are active low. If they were set to 1, the LED would initialize to on and would not blink on and off. This control register was then written to using a function to turn one pin on the LED on and off continuously.

The results in this part are as follows.

Lab 1: Part 1 Results	
Control Register Implementation	
Core 0 Frequency	2.28 MHz
Core 1 Frequency	5.59 MHz
Component-based API Implementation	
Core 0 Frequency	3.50 MHz
Core 1 Frequency	8.30 MHz

Part 1 Control Register Core 1 Waveform



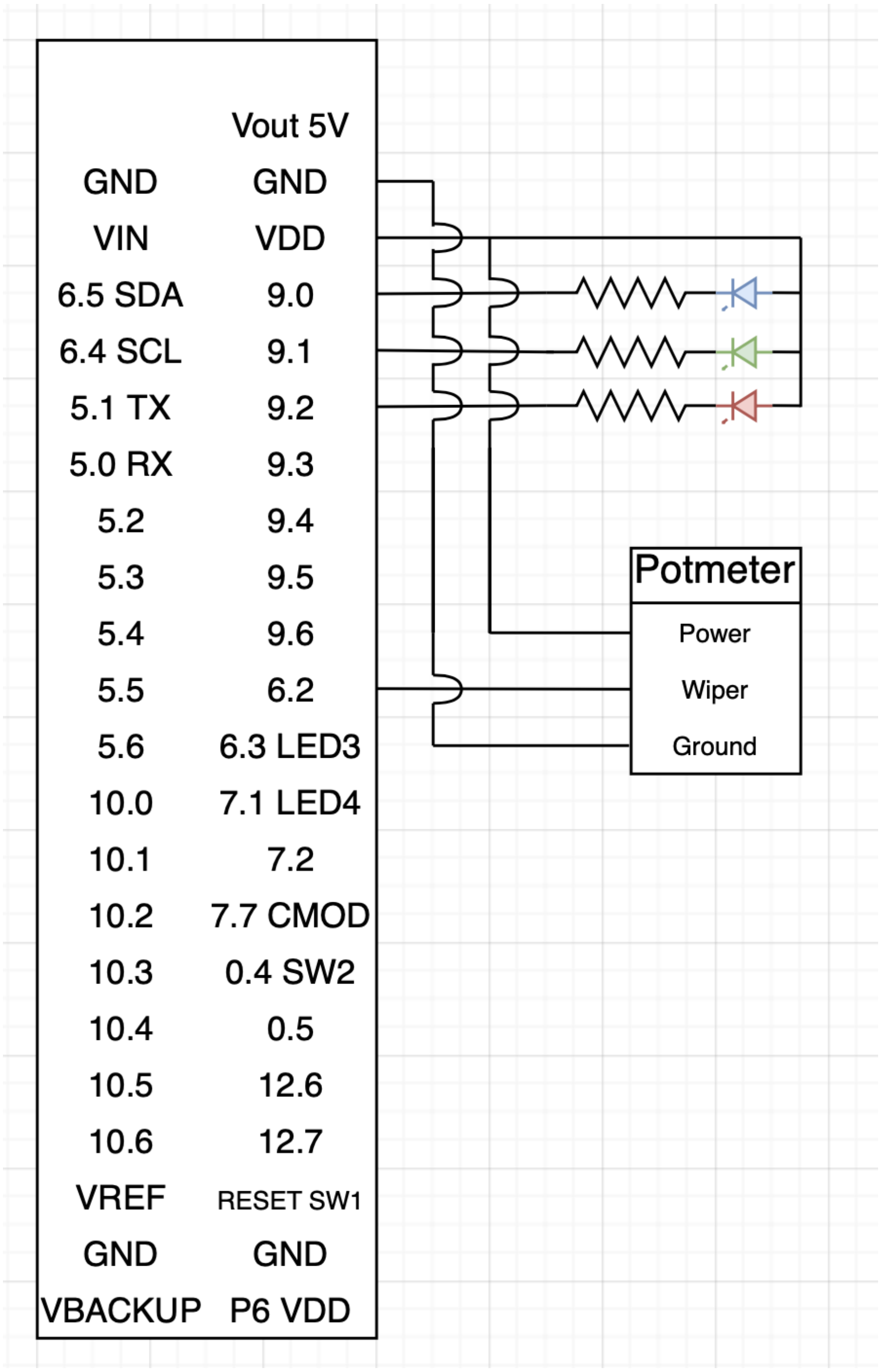
Measurements

	Name	Value
C1	Frequency	5.5989 MHz

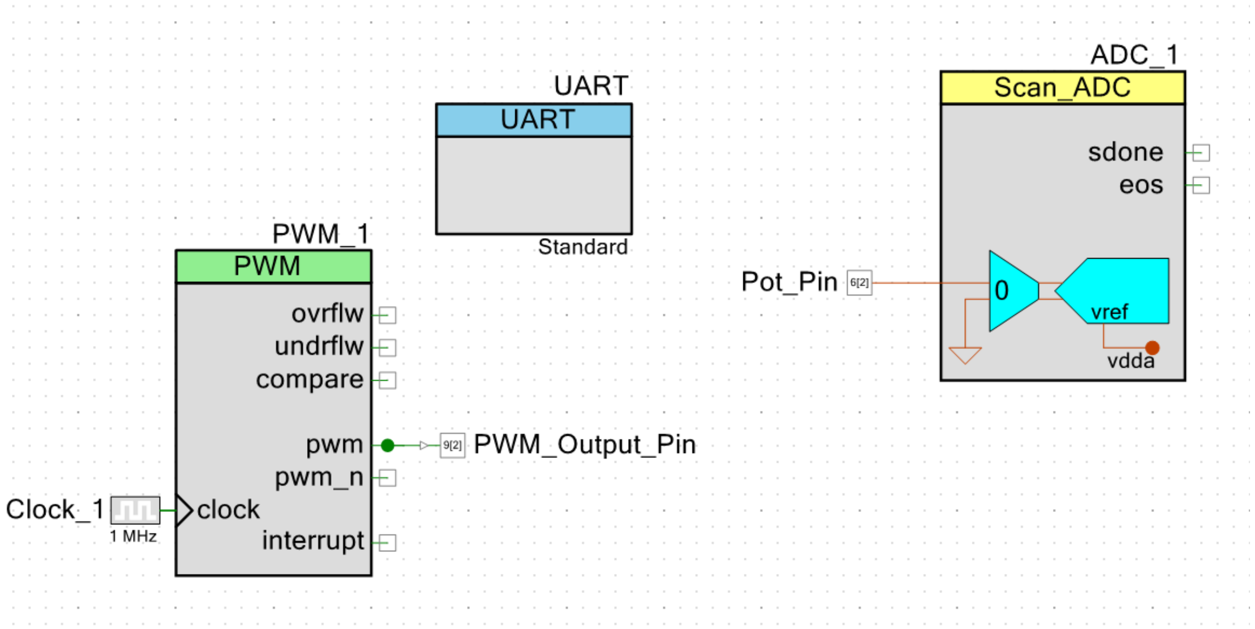
Unfortunately, I was not able to capture the Waveform for the Component-based API Implementation.
As seen above the frequencies are not the same, the Component-based API approach resulted in a faster frequency because it did not have an extra layer of routing, and the pin was called directly using GPIO functions. Core 1 processes at 100 MHz as opposed to Core 0 that runs at 50 MHz, but the data shown above does not see a 2x performance increase from Core 0 to Core 1. This is because both Cores are still subject to global constraints that are fundamental to using this microcontroller. Each approach to blinking an LED has a use. On one hand, the Component-based API Implementation is faster and would result in a more efficient way to control an LED, but scaling this method would be difficult. The advantage to using a Control Register is its scalability and usability. You can control a whole bus of LED's using one control register and control them all using one function.

Part 2:

External Schematic



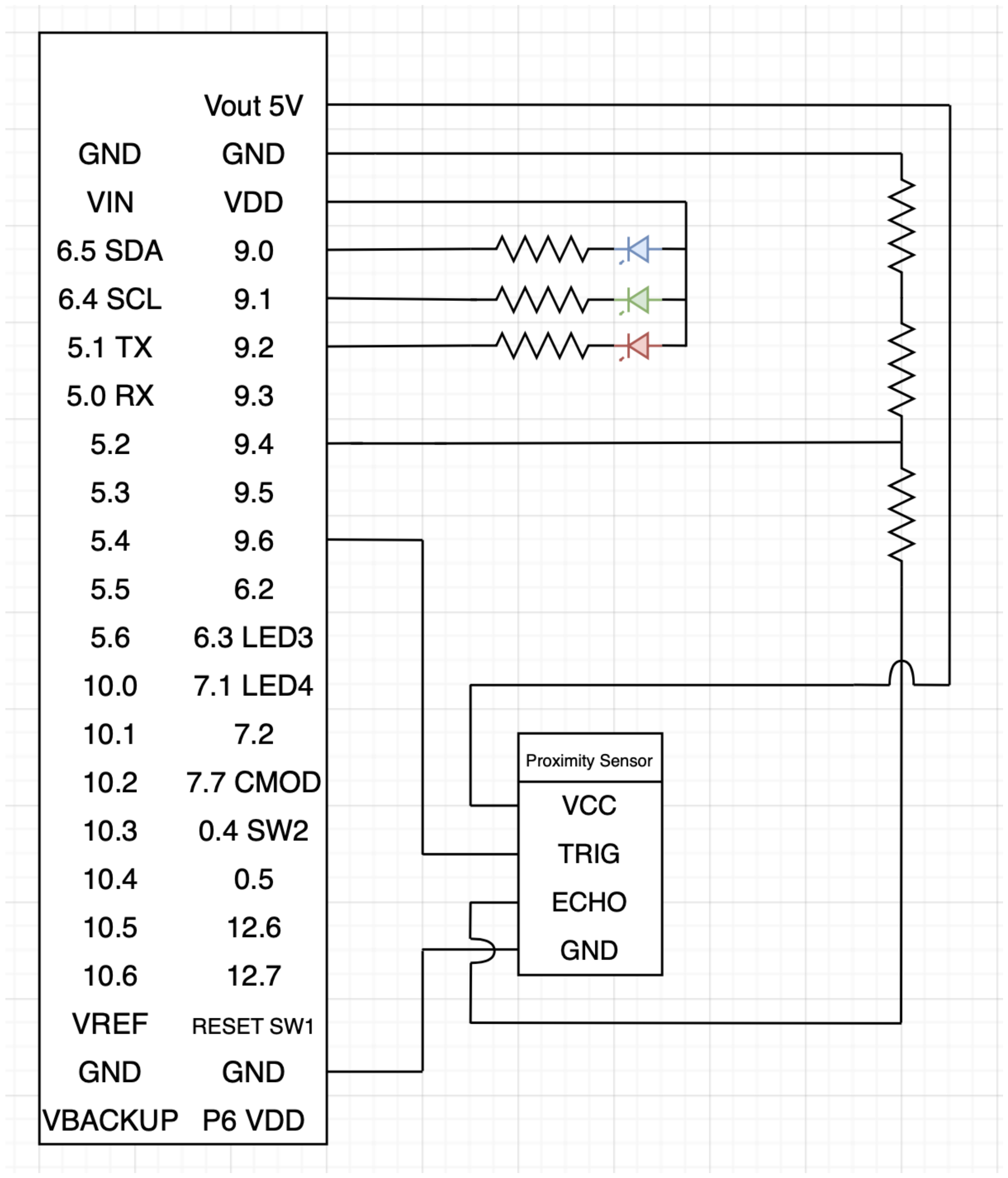
Top Design



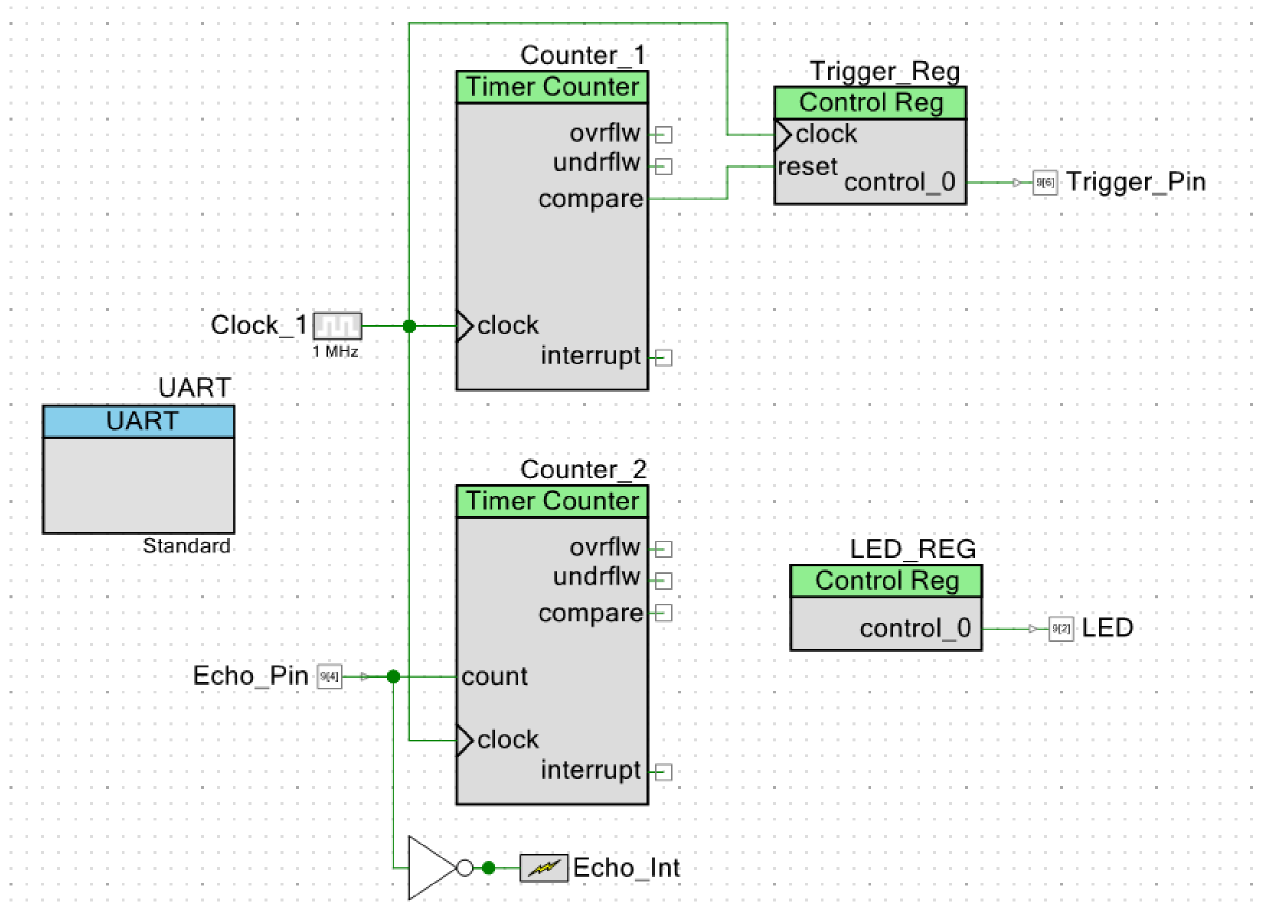
The Design for this part was the same as Part 1, only I also connected a Potentiometer to, Vdd, Gnd, and the Wiper to Pin 6.2. The software design for this part was simple, I connected the ADC to the Potentiometer Wiper Pin, and stored the value from the ADC. I then applied a bitmask to this value to mask out the upper 5 bits. I then divided this value by 2 to get the value in the appropriate range for full dimming on one end of the Potentiometer, and full brightness on the other. I then Set the PWM using this value to control the LED brightness. To test my design, I started by just reading the values from the Potentiometer and sending them to the stdout that was configured through the UART module. I used these outputs to test the bitmask and division operations, before connecting it to the LED. Once the Potentiometer was connected to the LED, the values were configured correctly and the system worked properly.

Part 3:

External Schematic



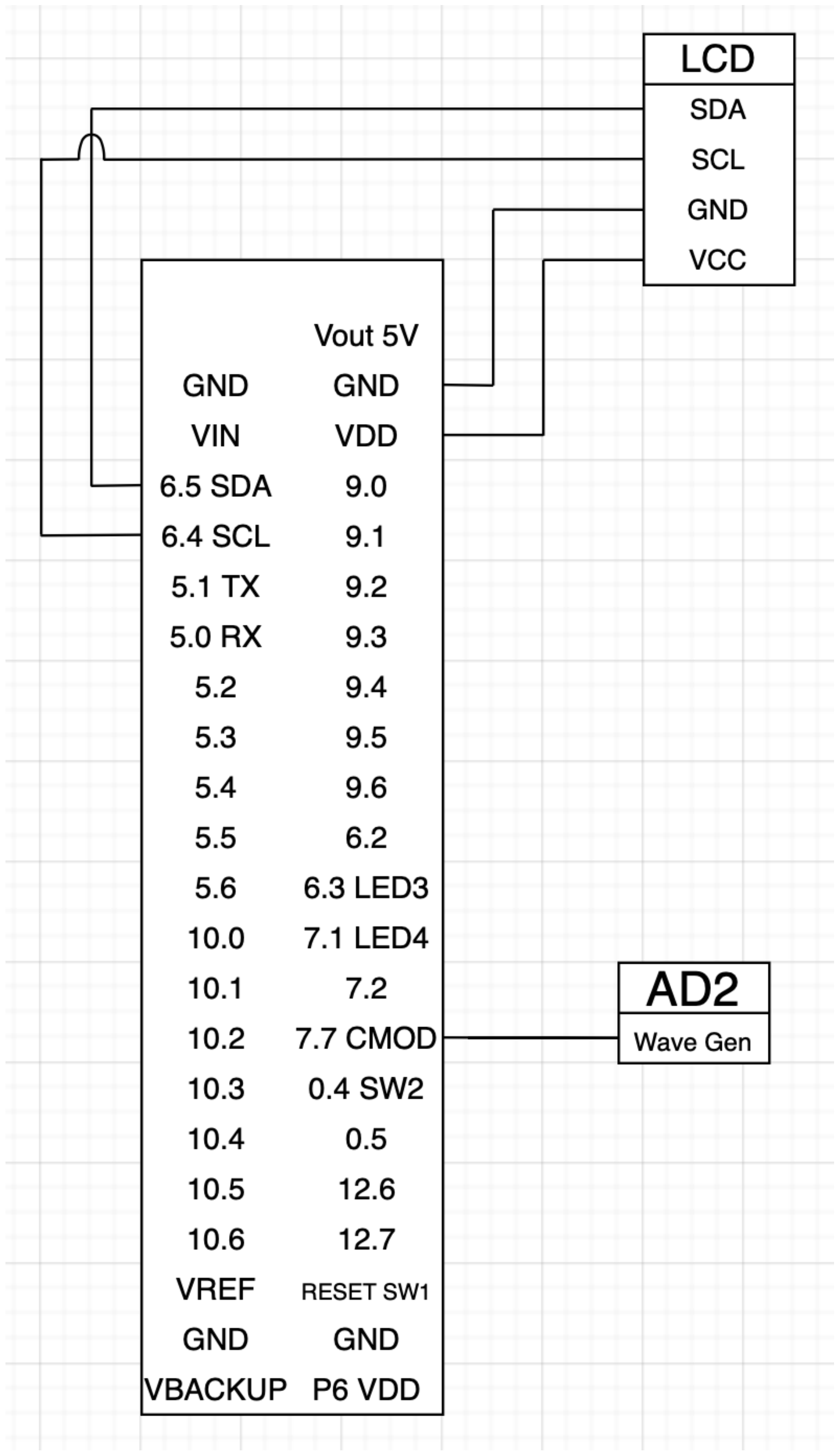
Top Design



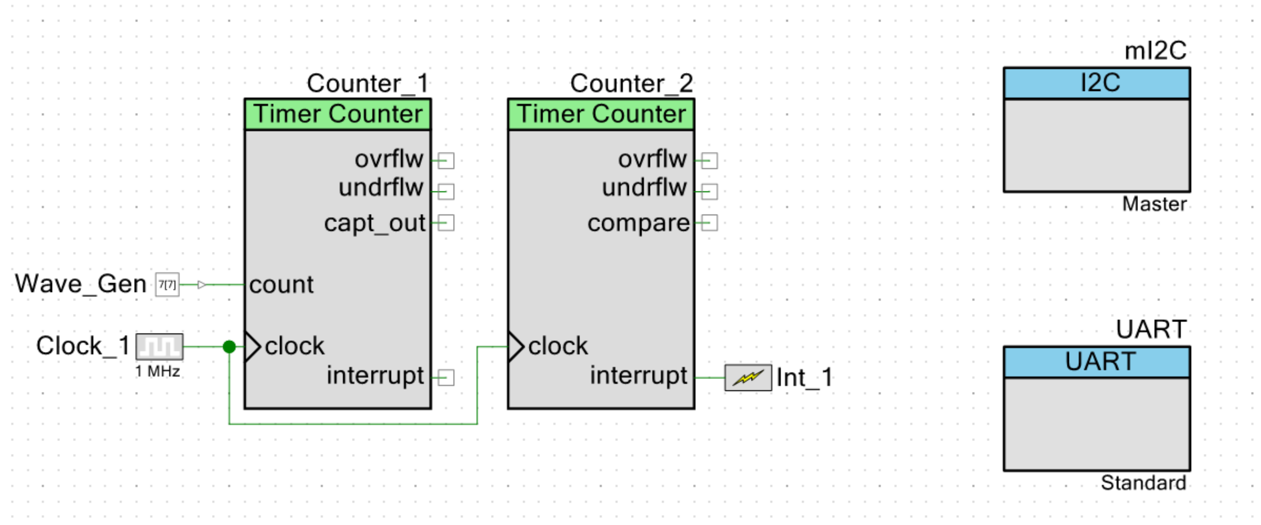
The Design for part 3 consisted of a Proximity Sensor and an LED. The Proximity Sensor needed a 5V power supply, but in order to preserve the microcontroller, the Echo Pin was routed through a voltage divider to limit voltage to 3.3V coming back into the board. The software design used an Interrupt to gather the value from the Echo Pin, the interrupt was then cleared and the program returned to the main loop. This value was used to calculate the distance by multiplying it by the speed of sound. Some basic logic was then used to keep the LED on for 5s when an object was within 100cm of the proximity sensor. A timeCtr variable is initialized to 10 when an object is within range, and the LED is turned on if the timeCtr variable is greater than 0. A 500ms delay is added to the end of the main for loop and the timeCtr variable is decremented each run through the main loop. Testing for this program started by first generating a counter that triggered the Interrupt. Then the Value from the Echo Pin was gathered and outputted to the stdout configured by the UART. The correct units of distance were then calculated to get a value in cm of distance read by the proximity sensor. The logic for turning the light on and off was then developed. I tested the system by putting an object at varying distances from the sensor and testing the amount of time it took the LED to turn on and stay on, putting an object in front of the sensor during and after the LED was on.

Part 4:

External Schematic



Top Design



The Design for part 4 consisted of an LCD and the AD2 to generate a waveform. The LCD was connected to the specified ports for SCL and SDA. The waveform generator was connected to port 7.7. The software design consisted of an interrupt function that is triggered by a counter. This interrupt gathers the frequency two separate ways. One way was to just gather the counter value and reset it, the other way was to store the last timer value and subtract it from the new value in the next interrupt. The LCD is initialized and the calculated frequency is outputted to the LCD using its included functions. Testing of this program also included sending output to stdout configured by the UART to test the output frequency. I found that the frequency calculated by saving the last counter value and using it to calculate the frequency using the new counter value gave a slightly more accurate result. The difference in frequency was about 1 or 2 kHz between the two methods. I found that as the input frequency increased toward the upper values being tested the calculated frequency seemed to become slightly less than the input frequency. For example when my input frequency was 1MHz, the calculated frequency was 996 kHz.

Conclusion:

This Lab provided a well rounded introduction to the PSoC environment and a taste to what coding in this environment is like. The use of the different hardware (LED’s, LCD, Proximity Sensor, and Potentiometer) established a good basis for using these tools in the future. The Lab showed the uses for these hardware tools. In part 2, similar types of systems are widely used, for example a dimming light in your living room. For part 3, similar systems control motion detecting lights that help save energy. These real world applications show the potential of these technologies and give significance to the implementation of these systems.