Leonid Shuster

CSE 121

Lab Section 1C

10/21/19

Lab Report 1

**Introduction**

In this lab, we were tasked with learning the basics of the PSoC-5 kit. This included using the GPIO pins, controlling the inputs/outputs of a program, and learning how to use a pulse-width modulator (PWM), timer, counter, and power management. The lab consisted of three parts. In the first part, we learned how to toggle certain pins to light up an LED, and controlled the brightness of an LED with a potentiometer. In the second part, we implemented a proximity detector that lit up an LED at a certain distance, and did the same thing for a second design that used less power. And in the final part of this lab, we created a frequency meter in which the output frequency of a PWM was regulated by a potentiometer that displayed its result on an LCD display.

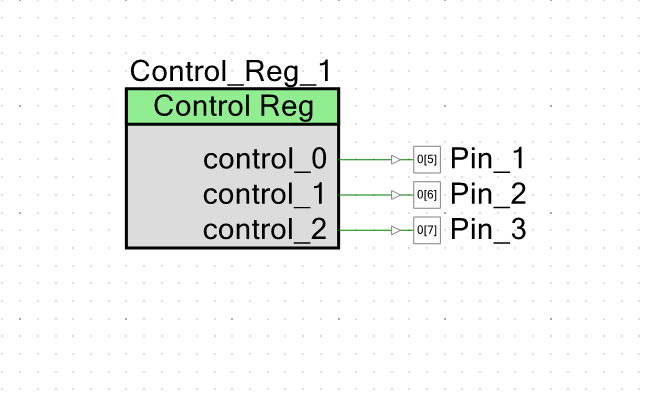
**Part 1(a): GPIO Pin Toggling**

In the first part of the lab, we were told to control the state of three output pins to flash an LED three different ways: using Per-Pin APIs, using Component APIs, or using a Control Register. For the Per-Pin API method, I looked in the datasheet on how to implement it. To do so, I added three separate pins to the top design schematic from the component catalog (see Figure 1).



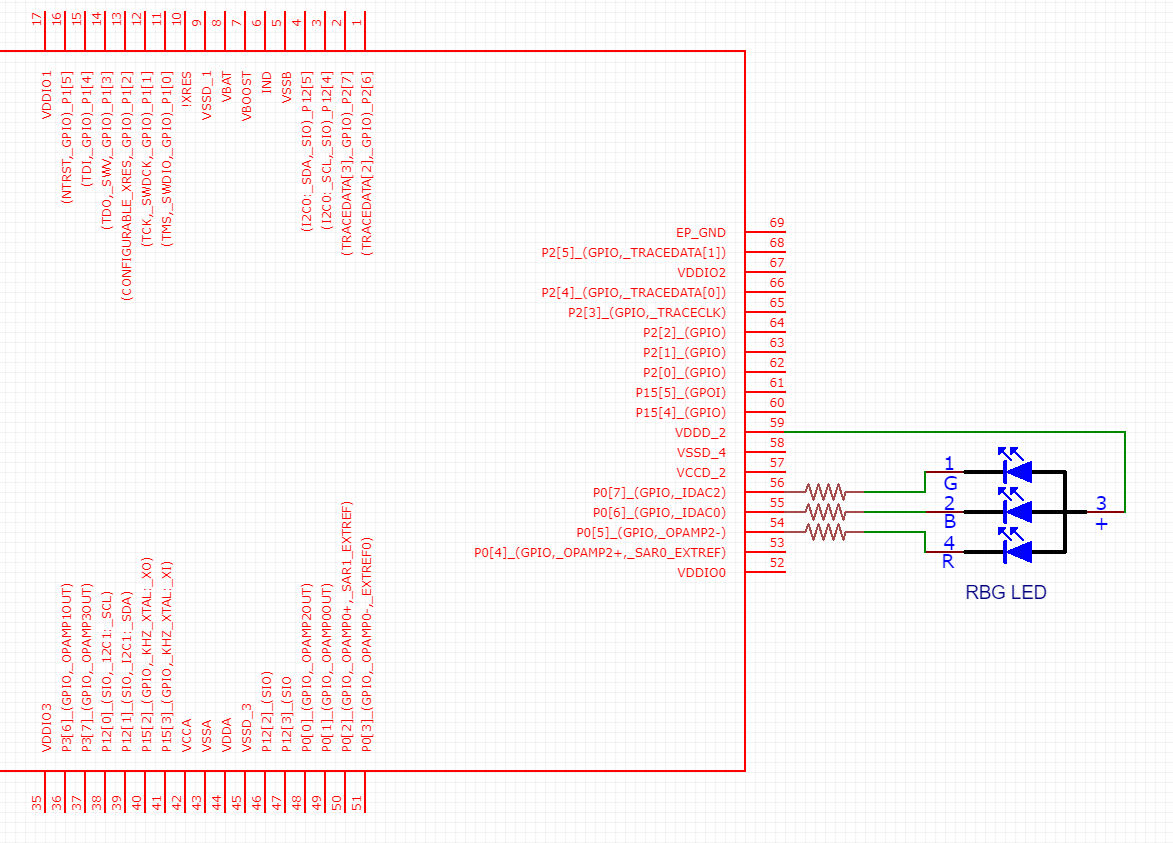
*Figure 1*

I then configured them to be digital outputs and assigned them to pins that were located on the PSoC. In main.c, I used the *CyPins\_SetPin()* and *CyPins\_ClearPin()* functions individually to toggle each of the pins. The same was done for the Component API method, including having the same top design schematic and routing of pins, except that in main.c I used the functions *Pin\_Write()* along with *Pin\_Read()* to toggle the pins all at once. For the Control Register method, we had to make use of a control register, so in my top design schematic, I added a control register with its outputs connected to three digital pins, which were routed to the same output pins as before (see Figure 2).



*Figure 2*

In main.c, I added values to the control register using *Control\_Reg\_Write()* that in turn toggled the output pins. In all three methods, each of the output pins of the PSoC were connected to a single lead of the RGB LED through a resistor, with one of the leads of the LED being connected to VDD (see Figure 3).

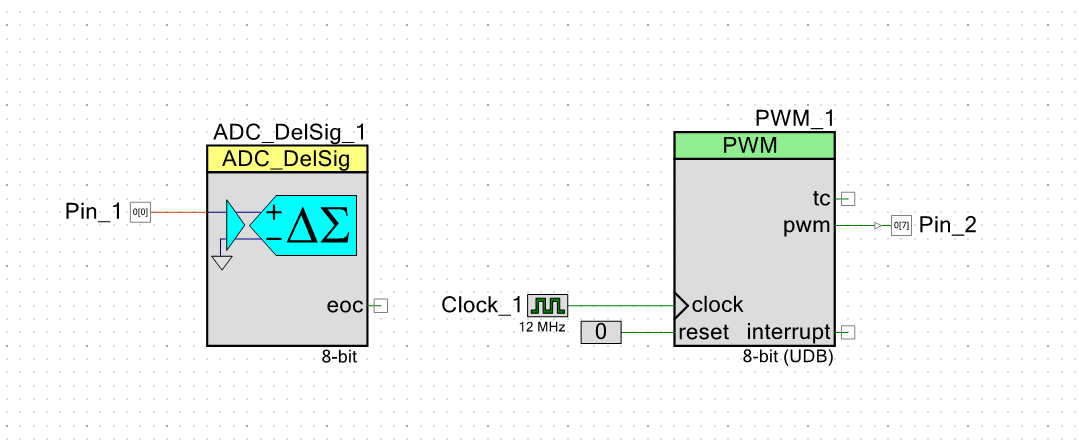


*Figure 3*

I then toggled one of the pins between 0 and 1 as fast as possible in a loop using all three different approaches, and used an oscilloscope to measure their frequencies. I saw that through the Per-Pin API method the frequency was around 704 KHz, with the Component API method the frequency was about 153 KHz, and with the Control Register method the frequency was the highest, around 1.33 MHz. I think they vary because of how fast they access data and move it, as well as how long it takes to perform a certain function call. A pro to using Per-Pin APIs is that you have more control over which pins are toggled, while some cons can be that if there were a lot of pins to control, it would take a lot of time to toggle them all. For Component APIs, a pro can be that you can control multiple pins at once, while a con is that it requires that all pins share the same output pin. Finally, for Control Registers, a pro can be that you can give pins different values in a single function call, while a con can be that it is more taxing on the system since you are doing it through one additional component.

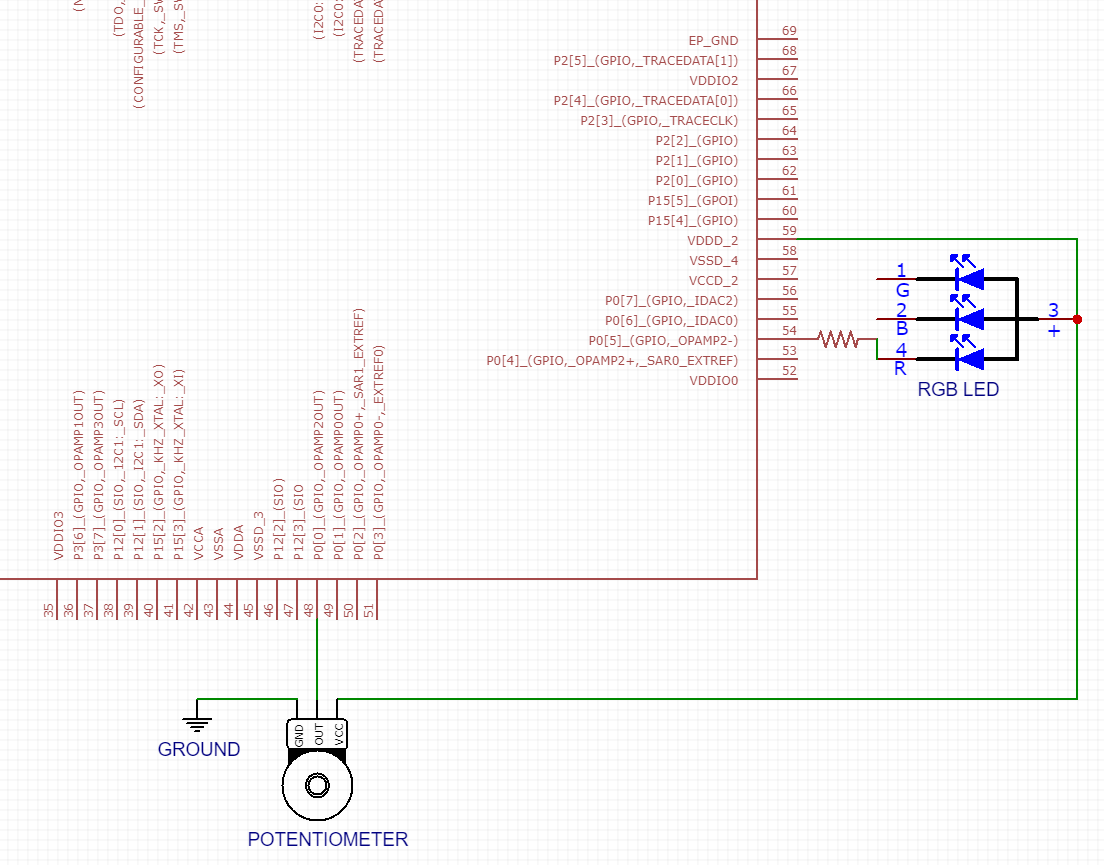
**Part 1(b): LED Brightness Control using PWM**

In the next part, we were tasked with changing the brightness of an LED depending on the position of a potentiometer by utilizing a PWM. The first thing we had to do was add an Analog-to-Digital Converter (ADC), which served to convert the analog input from the potentiometer into an internal digital signal for the system, and a PWM, which would receive the output from the ADC and output a signal that would go to the LED (see Figure 4).



*Figure 4*

I configured the ADC to be single-ended and have an output resolution of 16 bits and clock frequency of 1 MHz, while for the PWM, I changed the period of the PWM signal to 999, which resulted in a 1 kHz signal as its output. I then wrote a program in main.c that took the 32-bit result of the ADC conversion of the potentiometer’s position and gave it as a scaled input to the PWM. I had to account for the fact that the conversion could possibly lead to an overflow, so I had to set limits on the lowest and highest number to ensure the 32-bit number stayed within its proper bounds. Finally, I routed the digital pins of the potentiometer and LED onto the PSoC input/output pins with the potentiometer being connected to an input pin, and an output pin connected to an LED through a resistor, with one of the leads connected to VDD (see Figure 5).



*Figure 5*

When I read the potentiometer as a 16-bit number, the LED was changing from on to off or vice versa a lot quicker, since there was a smaller range of numbers as compared to a 32-bit number. I prefer the 32-bit number, since changing the potentiometer did not result in such a sudden change in the LED’s brightness.

**Conclusion**

Unfortunately, I was not able to complete the other parts in time. However, from the parts that I did finish, I learned the basics about how to work with the PSoC 5 and its GPIO output pins, ADC, and PWM. I was successfully able to toggle output pins that flashed an LED three different ways and control an LED’s brightness through a potentiometer. If I were to do this lab again, not only would I give myself much more time, but I would also look more into how each of the components work and interact with each other.