CMPE 121L

Xingyu Zhou

Anujan Varma

04/21/2019

Lab report 1

**Introduction:**

This lab is composed of 5 parts and the objectives of each of them were for us to get familiar with the PSoc-5LP microcontroller kit as well as some basics of the PSoc Creator. In order to get familiar with the microcontroller, we needed to study different data sheets to understand how to set up properly. In part 1 (a) of the lab, we were asked to toggle a single LED using 3 different methods. This was for us to get familiar with the GPIO pins. In part 1(b), we were asked to adjust the brightness of the LED using PWM and changing its duty cycle. In part 2 (a), we were asked to built a proximity sensor with a LED indicator while part 2(b) is basically the same but using a sleep timer. In the last part of the lab, we were asked to modify our design of part1 so that we could measure and output the frequency.

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

**A screenshot of a cell phone

Description automatically generated**

**Figure 1: Part1(a) Top Design**

In this part of the lab, we were told to control the state of a LED on the PSoc5 board. Three methods were introduced: Per-Pin, Component API and the Control Register. The first step was to add a digital output pin and assign to Pin that was connected to the LED on board. I have 3 pins since we need to display 3 colors simultaneously. Since Per-Pin method is fully controlled by software, there were no external input to my pins and I just need to write a C program to toggle them. The Component API method is basically the same as Per-Pin method, so I’ll talk about Control Register next. Instead of directly controlling the LED, we could also program the control register and let it do the toggling. When using control register, we need to enable HW connection option of the Pin and then connect them with the control register.(As it’s shown in Figure 1, Pin\_1). This part was fairly easy so I didn’t encounter any problem.

**Part1(b): LED Brightness Control Using PWM.**

This part of the lab asked us to change the brightness of the LED using one potentiometers. Two new component was introduced in this part—Delta Sigma ADC and PWM. Since we need to read off the value from potentiometer and it’s analog signals, we need to use ADC (Analog to Digital) to convert the signal. The signal was then fed into the PWM so that we could change the duty cycle and thus change the brightness of the LED. At first I’ve been using WritePeriod() but the LED was already off before the nob even passed half way. Then I learned that the WritePeriod() function was changing the PWM period all the time, but it should be fixed to 999. After using the diagram in the configuration tab, I noticed that all I needed to do is to compare the current period with the fixed period, and the function for that is WriteCompare(). After that my LED was working properly. The inverter was added so that the LED can be completely turned off. The Schematic is shown below.

A close up of text on a white background

Description automatically generated

**Figure 2: Part1(b) Top Design**

**Part2(a): Proximity Detector**

**A picture containing screenshot

Description automatically generated**

**Figure 3: Part2(a) Top Design**

In this part of the lab we were asked to design a proximity sensor using the ultrasonic module provided. A LED was used to indicate whether an object was within the range of 100cm. Two new modules were introduced in this lab—Counter and Timer. I started by implementing the trigger control. After the testing was completed, I switched the register to Sync mode (I didn’t exactly know why but Pulse mode is not working). Then I continued adding all the components by following the lab manual. In order to generate an interval of 10 microseconds, I chose a 1000kHz clock and set the period of my counter to be 10 (my friend told be 1 ms = 100kHz) and I set the clock input to be 5000kHz. (It must be at least 2 times greater than the clock of the count input). In order to determine the distance of the object, a timer is used. I set the timer to capture mode and it will capture each time when either edge (fall/rising) is detected. We were also asked to write a ISR, which to my understanding, is a place to handle the issue without using the too much CPU resources. Here’s what my ISR do:

1. Assign the current reading to the previous reading variable.
2. Read the current timer value and store it to the current reading variable
3. Return from ISR.

Here’s how I measure the width of the pulse:

1. I have two variable to store the current and previous reading of the timer.
2. Use the previous value to subtract the current value (since the timer is counting down) and their difference.
3. Plug in the difference in the formula provided d = 340\*result/20000. (Here I use 20000 instead of 2. The purpose is to convert the result to centimeters).

When I was testing my code in the debugger, I noticed that sometimes even the object was out of the range, my d value could be negative and thus turned on the LED. After rereading the datasheet of the timer, I noticed that when it was resetting itself my current value could be larger than my previous value, thus triggered the LED. To fix this, I add 65535 which was my maximum timer period to my previous value whenever it is less than the current value.

**Part2(b): Proximity Detector with Lower Power**

I put down all the codes that the lab manual provided as well as all the modules, but I didn’t know how to start so this part was unfinished.

**Part(3): Frequency Detector**

**A close up of text on a white background

Description automatically generated**

**Figure 4: Part3 Top Design**

In last part of the lab, we were told to generate a frequency between the range of 1kHz to 1Mhz. I copied pasted my design from part1(b) to here because they were quite similar. We were also asked to write two different ISRs to calculate the frequency and display it onto the LCD.

Method 1:

1. Read counter value and multiply it by 500 ( or /2ms)
2. Reset the counter.
3. Return from ISR.

My method 2 is the same one I use in Part1(b).

The expected frequency is calculated by dividing the clock frequency by the current period of PWM.

**Lab questions:**

**Part1(a):**

**Per-Pin API: 1.994MHz**

**Pros:**

1. Safely access individual pins without affecting the performance or taking more memory

**Cons:**

1. It’s unavailable when different components are connected to the same pin.

**Component API: 854.8kHz**

**Pros:**

1. More convenient to use since I can use a single call to access all the pins.

**Cons:**

1. All the pins need to be connected to a single port for this method to be effective.

**Control Register: 582kHz**

**Pros:**

1. Give us more control than writing into the pins directly

**Cons:**

1. Sometimes the control register is not fast enough and will generate bugs.
2. More expensive than the two methods above.

**Part1(b):**

The 16-bit option only gave me an value of 0x8000, while the 32-bit option gave me an reading of 0xFFFF. I would use the 32-bit option since this number is large enough to avoid the potential out of bound issue.

**Part3:**

|  |  |  |
| --- | --- | --- |
| Frequency | Track timer | Clear timer |
| 0x1200 | 0x1210(.347%) | 0x128A(2.76%) |
| 0x4600 | 0x460D(.16%) | 0x46D5(.64%) |

As it’s shown in the table above, the track timer method is more reliable since the error percentage is smaller than the clear timer method. Moreover, clearing the timer will add a little bit more time and thus the counter will count more and present an inaccurate reading. I did notice that the measurement changes a lot at the low end of range, but I’m not quite sure how to fix it.

**Conclusion:**

This lab teaches me some basics of the PSoc 5 boards and I learned a lot about the components by reading the data sheet. The difficult part for me was to determine the frequency of the timer/counter. I’m still kind of confused about how to get those data. Also I should be reading through the data sheet carefully before coding since most of the issue I encountered were addressed in the data sheet and I was too lazy to read through, thus took me a lot of time debugging. Other than that, the lab was quite fun. The proximity sensor reminds me of the camera in front of my house, which will only start recording if the sensor is triggered.

**A screenshot of a cell phone

Description generated with very high confidence**

**Figure 5: External Hardware Diagram**