CMPE121L

Xingyu Zhou

Anujan Varma

2019/5/12

**Lab4 Report**

**Introduction:**

The purpose of this lab was to taught us the basics of the UART on the raspberry Pi as well as how to utilize the GPIO pins for data transmission. To start we needed to read the documentation on raspberry Pi as well as the example code given to set up the Pi. This lab is consisted of two parts. The first part quite similar to our lab1, but instead of using PSoC’s PWM control the brightness of the LED, we would be using Pi’s PWM to control it. The second part was similar to our lab2, which was generating two sine waves. This time instead of looping the data inside of the PSoC alone, we would need to send the data through PSoC’s UART to Pi’s UART, and receive the data back to display the waveform.

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

**A close up of a map

Description generated with high confidence**

**Figure 1: Part 1 Top Design**

In this part we need to control the LED brightness using the Pi’s PWM. First thing to do was setting up the PSoC side. It was pretty easy because I’ve done that before. I used the code from lab 1 and mapped my potentiometer reading between 0-255. Then inside of the infinite for loop I just sent this reading through Tx every 1 millisecond. Instead of using a timer, I just use CyDelay(1). Then I started working on the Pi side. The Pi side, to be honest, was completely based on the example code. The set up part was identical and the only thing different was the data receiving part. I notice that my potentiometer reading can only go up to around 691 and the LED won’t turn completely off. Then I learned that the potentiometer voltage was not very accurate and there could be some delays between PSoC and the Pi. The way I fixed it was to scale up the value and treat 691 as 1023 by multiplying the reading by 1.48. For the hardware connection, I used a jumper cable to connect the Tx of my PSoC to the Rx of the Pi, and another jumper cable to connect the LED to the Pi.

**Results for part1:**

I was able to toggle the LED from completely off to completely on.

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

A close up of a device

Description generated with high confidence

**Figure 2: Part2 Top Design**

In this part the set up process for the PSoC side was quite similar to part1 of lab2. According to the lab manual, we needed two DMA channels and each channel should be consisted of two TDs so that they could form a Ping Pong buffer. The DMA request was also enabled and connected to a clock. The sampling rate I got was 10.47 kHz, and this data was acquired by using the BAUDRATE of the UART divided by the bytes to be transfer—in this lab it was 11. 115200/11 = 10.47k. In order to switch between the two buffers, I declared a variable called channel. Every time my first DMA ISR was triggered, this channel would be NOT with itself, a flag was also set to TRUE indicating the DMA transfer was completed. Then in the infinite for loop, I transmitted the data through UART based on which channel was filled. The Rx side was a little bit different. Since we needed to fill the Rx buffer first and then send the data to VDAC, it was a bad idea to keep the DMA running at all times. Here’s what I did: First I start filling my first buffer inside of my RxISR, when it reached the array size(64), the counter would be reset to 0, the DMA2 would be enabled and it would start filling the second buffer (channel switching method same as Tx). When the DMA was done sending data to the VDAC, the interrupt would be triggered and the second DMA would be disabled inside of the ISR. By looping the Rx side back to by Tx, I was able to observe a clear sine wave on the oscilloscope. Then I moved onto the Pi side. The set up process was again following the example code. Inside of the infinite for loop I was constantly receiving and sending out the data. I had set the Tx to only transmit data when there was something inside my Rx buffer to avoid any data loss. When I first connected PSoC and Pi together, I was having a lot of data loss, eventually I had to lower the sampling rate of the DMA to 8 kHz to produce a smooth sinewave.

**Result for part 2:**

I was able to generate a smooth sinewave on the oscilloscope and the program can run many seconds without generating any errors or data loss.

**Questions:**

The sinewave started to break above 1 kHz and the data loss was increasing as the frequency went higher. Eventually at 2.3 kHz my code failed to generate a sinewave. One of the problem would be the hardware connection, there would be various delays between each wire and they would cause the data loss at higher frequency. I would increase the BAUDRATE of the UART and see if it helps. Or I could get a better wire.

**Conclusion:**

This lab was more like a refresher to me since it is a combination of all previous labs. Now I feel more comfortable manipulating memory with UART and DMA but I still need more time to get used to the raspberry Pi. In general it was a fun lab and I look forward to lab5 and the final project.

A screenshot of a cell phone

Description generated with high confidence

**Figure 3: External Connection between PSoC and Pi**