CMPE121L

Xingyu Zhou

Anujan Varma

04/28/2019

Lab report 2

**Introduction:**

This lab is composed of 3 parts and each of them were for us to get familiar the functionality of the DMA hub(Direct memory access)and how to manipulate it. All 3 parts required the manipulation of the DMA to transfer the data from memory to memory. In part 1a of the lab, we were asked to generate two sin waveforms and shift the second waveform with respect to the first one. Also we would need to shift it from 0 to 360 degrees and display the result on the LCD screen. In part 1b, we need to modify our part 1a so that the waveform should stay consistent during the zero-crossing point and continue to shift once the potentiometer changed. For part 2, we just need to do array to array memory transfer.

**Part 1a: dual-channel waveform generator**

**A screenshot of a cell phone

Description generated with high confidence**

**Figure 1: part 1a top design**

The first part was fairly easy since there was already an example code that can generate the first sine wave. All I did is copy the example code and set up the second DMA channel so that I can have two waveforms at a time. The size of the original look up table was 128 and I doubled it to 256. The reason being that when I was shifting the second sine wave, the operation was + 128 from the current position. For example, if I started on 20, then 20 + 128 will be 148, which exceeds the size of my array and the program will read in garbage values. That was the reason why my second waveform was glitchy when I started to shift. Setting up the potentiometer was the same as lab1. Also we needed to transfer the look up table from FLASH to SRAM at the beginning of our code to avoid the automatic shifting at higher frequencies. In order to set the range of the frequency between 100Hz and 100k Hz, I used the clockdivider function. How to determine the value of the divider? I simply set my clock to random frequencies and checked the frequency of my sine wave on the oscilloscope. Once I found the correct clock frequency, I compared them with the Master Clock (24MHz) and got the divider using the clock tap.

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

**A screenshot of a social media post

Description generated with very high confidence**

**Figure 2: part 1b top design**

In this part we were asked to modify our code for part1 so that the wave would only shift at zero crossing points. This was the part which I didn’t finish. Basically my idea was separating the second TD into 2 TDs with one having 127 bytes and the one have 1. When the transfer count hits 127 bytes, it would trigger the ISR, and while the program was in the ISR, the second DMA channel would be disabled, slightly delayed, and reenable again. That should allow me to see the waveform in the lab manual. However my first waveform was all glitchy when I tried to separate my second TD into 2. I couldn’t solve it before the check off.

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

**A close up of a piece of paper

Description generated with high confidence**

**Figure 3: Part2 top design**

In this part of the lab, we were asked to use the DMA to transfer data from one place in RAM to another place. The total bytes to transfer was 16384 but each TD could only handle 4096 bytes maximum. in order to deal with it, I set up 5 TDs with 4 of them having 4092 bytes and one of them having 16 bytes. Nearly all the codes outside of my infinite for loop were generated by DMA wizard, and we were told to use two method to transfer the data.

The first one was using the hardware. When the transfer is finished and ISR will be triggered. Then in the ISR, the timer (used to measure the time taken to transfer data) will be stopped and the value will be transfer to microsecond. Then the program will return from the ISR.

The second one was to use the software. Since we were told to change from little-endian to big-endian, what I did was to copy the 4th element from source array and store it to the first place in my destination array. I did this by having two new variables, column and row. When a byte was transferred, I would increase my column variable so that I can move to the next byte. When the column reached 4, it would be reset and the row variable would be increase by 1, advancing to the next row. The timer would start before the transfer and would end after the code exited the for loop.

The error checking method was basically the same as my software controlled method, except I was checking for mismatches instead of copying the value. At the end of the error checking the number of mismatches would be displayed on the LCD screen.

Below is the time both methods took to transfer the data

|  |  |  |
| --- | --- | --- |
| DMA | Software Loop | Slow Down |
| 1380ms | 760ms | 0.5X |

Technically speaking the DMA method should be way faster than the software loop but somehow it was the other way around. I think the reason why DMA should be faster is that DMA can move the data directly without involving the CPU. The CPU needs to handle a lot of things at the same time so it won’t be able to transfer the data at full power, and thus the software loop is slower. I might mess up somewhere in my code and result in a situation which software loop is faster than DMA.

**Conclusion:**

The purpose of this lab was for us to understand how the DMA works and how to use TDs configuration to manipulate memory. I also learned that DMA is much more efficient than using a software loop for transferring data(learned by researching on Internet, not by experiment). I enjoyed doing this lab since playing with the waveforms was fun and the lab was easier than the last one. I also enjoyed how some of the parts were built on top of each other and that the feeling of gradually progressing.

A close up of a map

Description generated with very high confidence

**Figure 4: External Diagram**