Wilson Au-Yeung

CMPE 121 / CMPE 121L

Anujan Varma

October 23, 2016

Lab Report 2

Introduction:

This lab is comprised of three sections, each of which were focused on teach us and help us understand the functionality of the DMA hub on the Cypress system. All of which required the use of the DMA hub in order to transfer data from memory or memory to memory. Part 1a consisted of creating two sinusoidal waveforms, one that stayed as a reference point, but the other would have a phase shift from 0 – 360 degrees depending on the value of the analog potentiometer. Both sinusoidal waveforms would use a look up table which would be stored in flash memory. For part 1b, we had to slightly alter what we had part 1a and only shift during the zero-crossing points in the waveform. If the potentiometer should change value and it was during the zero-crossing point, the line in the waveform should stay flat until the potentiometer stays constant (or with minimal change) and proceed like before. For Part 2, we had to transfer data from one RAM to the other. All parts in order familiarize ourselves on DMAs.

Method:

Part 1a: Dual-Channel Waveform Generator

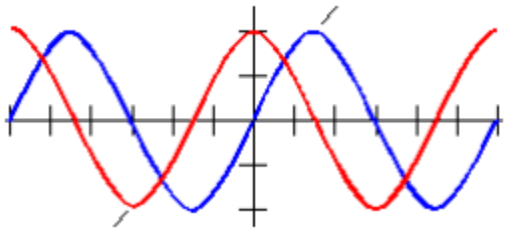

Figure 1: Two Sine Waves w/ Variable Phase

In this section of the lab, we were told to use a look up table which was held in flash memory. Using the flash table, we were told to construct two sinusoidal waves on two different output pins which had a variable phase shift (0-360) between them as shown in Figure 1. We started out using Example 3 in the DMA Datasheet. Example 3 gave us everything we needed to get started on this part of the lab. The lookup table was provided, as well as all the schematic and c code needed to access memory in the DMA. I made a duplicate of all the initializing code in order to create another sinusoidal wave so I could keep one constant.

Part 1b: Dual-Channel Waveform Generator with Controlled Phase Change

In this section of the lab, we were told to using the main concept in used in Part 1a and modify it so it would only change phases during zero crossing points in the sinusoidal wave. In order to do that we had to first create an event that would trigger if the potentiometer would start to move. We also had to take in the factor that we were using very cheap potentiometer and it would not stay at the exact value so we had to determine a specific threshold. We also had to create an event that would tell us when the wave was at zero-point.

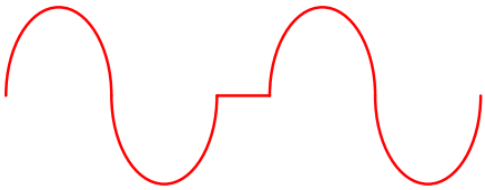

Figure 2: Phase Shifted during a Zero - Point

Part 2: Memory-to-Memory Block Transfer

In this last section of the lab, we were told to use the DMA to transfer from one part of RAM to another part. To set up the DMA transfer, we used the DMA wizard to take care of all the syntax it requires. Since the block is 16,384 bytes, and one TD only goes up 4096 and it has to be a multiple of the endian value, we had to use 5 TDs. 4092 bytes in the first four while the last one had 16 bytes. We connected the first 4 to the next TD so it would automatically configure to the next one when it runs out of memory. We first had to create a for loop placing in each spot an

increasing pattern of modulo 256, while in the destination array, we cleared. (Figure 3)

```
for(i = 0;i < BlockSize; i++){
        sourceArray[i] = i % 256;
}
for(j = 0;j < BlockSize; j++){
        destinationArray[j] = 0;
}
```

Figure 3: For Loop Storing Module 256

Procedure

Part 1a: Dual-Channel Waveform Generator

Like in the first lab, we have to define thresholds for the potentiometer so it does not go under 0x0000 and over 0xFFFF. We then had to determine the maximum and minimum phase change that the sinusoidal wave could go through and we had to do 360. We used the built in TD set -address function and pointed at different values of the look up table. It would point at a different location in the look up table depending on the value of the potentiometer.

Part 1b: Dual-Channel Waveform Generator with Controlled Phase Change

The hardest part for this part of the lab was figuring out how to determine whether the potentiometer has changed. So the way I implemented this was recording the old value of the potentiometer and comparing that to the new value. If the change was higher than 0x500 (which is usually higher than the fluctuating value when left alone) it would check if it was at the zero-point line. The other challenges involved setting the sine wave to 0 at the right time, so I decided to create a second lookup table containing 128s so it would continue sending a flat line until the potentiometer would stop changing and go back to where it left off in the original look up table.

Part 2: Memory-to-Memory Block Transfer

We then were asked to measure how long the data transfer was so we started the timer in the main function, started the transfer and on the top level schematic we sent an interrupt once the transfer was over. In the interrupt, we assigned the time to a variable and printed it out. (Figure 4). We also did an integrity check by forcing in multiple incorrect values inside the array and using a counter to display all the incorrect values.

```
CY_ISR (Interrupt){
    Timer_1_Stop();
    time = ((4294967296 - Timer_1_ReadCounter())/24);
    LCD_Position(1u,0u);
    LCD_PrintNumber(time);
    CyPins_SetPin(LED_1_0);
}
```

Figure 4: Interrupt Storing the Timer into Int

The last part of this section was to recreate all the transfers done by the DMA Hub but using a software loop instead. I then implemented the transfer using a for loop and tested it by forcing incorrect values like I did before. However, this method doesn't not require the use of and ISR because we could simply start the timer before the for loop and end it immediately after.

```
Timer_1_Start();
for(k = 0;k < BlockSize; k++){
        destinationArray[k] = sourceArray[(row*4)-count4-1];
        count4++;
        if(count4 == 4){
            count4 = 0;
            row++;
        }
}
Timer_1_Stop();
time = (42949672960-Timer_1_ReadCounter()/24);
```

Figure 5: Software Loop

DMA	Software Loop	Speed Up
4.608ms	21.048ms	4.6X

Conclusion

The purpose of this lab was for us to understand how the DMA works and how to use DMAs and TDs configuration to manipulate memory. I learned through this lab how using TDs and DMAs is a lot more efficient in moving data from memory than using a software loop and the CPU because it continually wastes CPU cycles. I figured out how thoroughly you have to read the datasheets as it would leave a lot of hints for you to progress on. I also enjoyed that we are finding more creative and innovative ways to design our schematics. I also really appreciated that it gradually built on what you learned previously and you had to really grasp the previous

section to really progress on. This lab was a really good refresher for me as it combined some aspects of CE 100, CE 13, and EE 101 in one lab.