Wilson Au-Yeung

CMPE 121 L

Anujan Varma

November 5, 2016

Lab Report 4

Introduction:

This lab is comprised of two sections, each of which were focused on teaching us the basics of using and getting familiar with the USB interface driver. The USBUART component establishes communication with PC as a virtual COM port. We would use the Psoc 5 board as a USB device to simulate a UART for transmitting and receiving data. In lab 3, we used the UART to transfer data from one location (array) to another (array) within the Psoc Board. In this particular lab, we were told to transmit data from the PC host to the Psoc device and receive data back into the PC host. In the example code, we were given all the API functions used to read in data and write data into the console. Initially all it did was receive data, and then it would echo whatever that was put in back into the console. What we were told to was to receive all characters and echo only in packets of 64. Left overs would be stored until another set of 64 was reached. In the second part of the lab is to implement the same idea as part one but instead use the DMA for both transfer and receiving data.

Procedure:

Part 1: Transmission and Reception of data using Programmed IO

To set up this part of the lab, we opened up the USB_UART example code and got all the API functions needed to read in data and write in data into the console. We did not change to preset settings for configuration of the USBFS in the example. Instead of echoing every character received in the console back to the computer, we had to echo in packets of 64. In incorporate this, we had two receiving buffers store the transmitted data and write in a ping pong fashion. I first created a large array which received all transmitted data into it.

```
uint8 TempStore[USBUART_TEMP_BUFFER_SIZE];
uint8 remainder[USBUART_BUFFER_SIZE];
uint8 RxBuffer0[USBUART_BUFFER_SIZE];
uint8 RxBuffer1[USBUART_BUFFER_SIZE];
```

Figure 1: Creating Arrays for Temporary Storage and RxBuffers

I divided the number of bytes received by 64 to know how many times to write data into the PC. I then took the 64 bytes and stored them into the first RxBuffer0. And then I wrote the data into the PC. The next 64 bytes would be stored into RxBuffer1 and would be printed out into the PC. And it would switch between RxBuffer0 and RxBuffer1. We started off having a smaller buffer size of 4 so we could test it easier.

```
USBUART_PutData(RxBuffer0, USBUART_BUFFER_SIZE);

total = USBUART_GetAll(TempStore);
```

Figure 2: Functions used for Receiving and Transmitting Data

We then had to account for the remaining data that could have been left over by the last USBUART_GetAll() function. Then it would keep receiving data and finally print out when it finally receives at least 64 bytes. When it sends 64 bytes, we needed to send a specifying 0 to complete the transfer.

```
total = USBUART_GetAll(TempStore);
numtoRun = (total + leftOver) / 64;
leftOver = total % 4;
```

Figure 3: Keeping Track # Times to Run and Leftover Data

Part 2: Transmission and Reception of data using DMA

In this section, we were told to use the same code we created in Part 1, but this time use the DMA for both receiving and transmitting data. The USB component already has a built in DMA controller, so all we had to do is select it in the configuration options. The USB component also does not have interrupts, so we had to have a lot of if statements checking the status of the transfers.

```
elements = USBUART_GetEPCount(3);
total = USBUART_ReadOutEP(3, TempStore, elements);

USBUART_LoadInEP(2, RxBuffer0, USBUART_BUFFER_SIZE);
```

Figure 4: Functions used for Receiving and Transmitting Data (DMA)

We had to configure certain API functions used for the transfer since it is a DMA now. We had to find out the Endpoint Descriptors for the CDC interface so we found out that the In Direction describes a similar function to PUTDATA() function used in part 1, and the Out directions was similar to the READALL function. The only difference was that when trying to read out data from the PC, we had to specify the amount of bytes that was being read. But I just used another built in function USBUART_GetEPCount(3); and I was able to get that information.

Conclusion

The purpose of the lab was to understand how the USB_UART works and how to use its configuration to manipulate memory and where to transfer it. We also learned how to incorporate the same idea to transfer memory from one location to another using the built in DMA in the USBUART. The example code gave us a lot to get started. I really appreciated that it gave all the API functions needed to do the lab which saved a lot of time. Even though we read the datasheet thoroughly, it is hard to gauge which function would be the best to use for the lab.