Andrew Mikhail

CMPE 121L: Microprocessor System Design Lab

Lab Exercise 4

4 November 2016

Section: Tuesday 10-12pm

Table of Contents:

Introduction **[3]**

Part 1: Transmission and Reception of Data Using Programmed IO **[4]**

Part 2: Transmission and Reception of Data Using DMA **[5]**

Conclusion **[6]**

Introduction:

The purpose of this lab exercise is to learn about USB interfacing with devices using the PSoC 5 board. For this lab, the PSoC 5 board acted as a USB device that was connected to a PC. A UART on the PSoC 5 board was used to transmit and receive data to the PC via a USB. The Communication Device Class (CDC) was used to service the transfer of data from our PSoC board to the PC. I also used the on-board DMA controller to receive and transfer data to the PC. The transfer of data was conducted using USB 2.0's full speed option that is 12Mbits per second.

Before starting the lab, I first had to perform a number of preparation steps and setup the board:

- I first arranged the board for 3.3V operation.
- I then built and programed the board with the USB_UART example project.
- Next, I connected the board to the PC via the USB COM connection.
- I determined what the COM port number is by checking the device manager on the PC.
- I then downloaded the terminal application "CoolTerm" to connect with our board.
- I finally setup the UART transfer parameters to be 38400 baud, 8-bit data, odd parity, 1 stop bit, RTS/CTS flow control.

Part 1: Transmission and Reception of Data Using Programmed IO

For this part of the lab, I made some slight adjustments to the USB_UART example project. Specifically, I implemented a program that would configure and initialize the USB interface then continuously poll for data in the form of keyboard presses from a host PC. Moreover, the data received would then be stored in buffers and transmitted back to the host PC once 64 bytes have been received.

I created two 64-byte arrays, RXBuffer0 and RXBuffer1, to behave as receive buffers. I then used the API functions USBUART_DataIsReady() to poll for keyboard presses from the host and USBUART_GetAll() to receive the data into the RX buffer. The USBUART_GetAll API function returns a 16-bit value that is the number of bytes received. Thus, I saved that value into a counter variable to track how many bytes have been received. Since data was continuously being received, I incremented the counter every iteration in the infinite for loop. I then compared the counter value with the buffer size, which was 64, to determine if the receive buffer was full. Finally, once the CDC was ready, the contents of the buffer were sent back to the PC using the USBUART_PutData() API function. Following that, I sent a NULL packet to the PC so the UART knows to stop waiting to receive more packets.

Furthermore, to ensure continuous data retrieval, the first receive buffer would switch to the next as soon as it gets full. I used a flag variable called bufferSwitch that allowed me to switch back and forth between the receive buffers. Once the first buffer became full, bufferSwitch would be set to switch to the second buffer. Then, bufferSwitch would be cleared after the second buffer became full to return back to the first RX buffer.

Part 2: Transmission and Reception of Data Using DMA

For this part of the lab, I simply made a few modifications to the code written for Part 1. First though, I changed the USB UART configuration to use DMA to transfer and receive data. I then determined that the Endpoint number for the IN descriptor is channel 2 and the Endpoint number for the OUT descriptor is channel 3. Next, I replaced all the instances of the USBUART_GetAll() function with the USBUART_ReadOutEP() API function. These two functions perform similar things. However, the ReadOutEP() function moves a specific number of bytes from Endpoint RAM to data RAM (Host PC to PSoC 5 board). Since I knew the Endpoint channel of the output is channel 3, I was able to read the number of bytes from the address of the buffer at each index. Likewise in Part 1, I saved the number of bytes read into a counter and incremented it every iteration in the infinite for loop.

I also replaced all the instances of the USBUART_PutData() function with the USBUART_LoadInEP() API function. Again, these functions perform similar actions. However, the LoadInEP() function transfers data RAM to Endpoint RAM (PSoC 5 board to host PC). Also, I knew the Endpoint channel of the input was channel 2. Therefore, I was able to transmit the contents of my receive buffers to the PC. Likewise, I loaded in a NULL packet after my data to indicate the end of the transfer. The same implementation of receive buffers from Part 1 was used here in Part 2. There were still two receive buffer of size 64 bytes that switched off receiving data.

The built-in DMA controller of the USB component allows only data to be transferred from/to the main memory. For applications requiring data to be transferred from the USB interface to another peripheral, I would simply use two DMA transactions.

One DMA transaction would be from the USB endpoint to the main memory and the second would be from the main memory to the peripheral.

Conclusion:

I thought this lab was effective in showing us the USB interface of the PSoC 5 board. This lab provided a complete introduction to using the USB interface as a virtual UART to transfer and receive data on the PSoC-5 microcontroller board. I also became familiar with implementing DMA transfers and receives over USB 2.0 using the DMA controller. This lab provided us an opportunity to become comfortable with USB interfacing using the COM port of the microcontroller. Overall, I thought this lab was well presented and quite informational on the topic. However, I think that this lab can be improved by informing students of a specific terminal application to use. I had quite a bit of trouble during my check off using the applications Hercules and HyperTerminal. I would suggest telling students to use CoolTerm instead since that worked most competently in my experience.