CMPE 121L: Microprocessor System Design Lab

Fall 2016

Lab Exercise 4

Check-off Due in lab section, week of October 31, 2016

Report due: Within 3 days after checkoff due

The purpose of this exercise is to learn to use the USB interface of the PSoC 5 board to communicate with a PC and transfer blocks of data between them. The PSoC 5 acts as a USB device and the PC as a USB host. We will use the USB interface as a virtual UART for transmitting and receiving data. The code that you develop will be handy later for debugging your designs by using the PC as a display terminal.

The USB 2.0 Specifications define many communication classes: The one we use is called the Communications Device Class (CDC). This is a general service that is used for transferring data between a USB device and a USB host.

The USB 2.0 Specification supports three speed classes: Low Speed (1.5 Mbits/second), Full Speed (12 Mbits/second), and High Speed (480 Mbits/second). The Cypress USBFS component uses the Full Speed option.

Preparation:

Before starting this exercise, open the example project USB_UART.

- Set up the board for 3.3V operation.
- Build the project and program the board.
- Remove the USB cable from the Program/Debug connector and connect to the USB connector marked USB COM.
- If the PC does not have a USB driver installed for the USB UART, you will need to provide a path for it to install the driver. The path is normally C:\Program Files\ Cypress\PSoC Creator\3.1\PSoC Creator\examples\sampleprojects\ USB_UART\.
- Open Device Manager from the Control Panel to find the COM port number assigned to the USB port.
- Bring up the terminal application on the PC. Supply the COM port number to open the terminal. Set the UART parameters to 38400 baud, 8-bit data, odd parity, 1 stop bit, RTS/CTS flow control.
- If you type any text on the keyboard, the application will send it to PSoC-5 board over the USB link, and the echoed text will appear in the terminal window.
- Click on the USBFS component in the schematic, open its datasheet and save it for easy reference.
- Study the C code of the example.

Initialization of USB component:

Before your program can transfer data through the USB link, the USB component must be initialized using the API call **USBUART_CDC_Init().** The program must then wait for the host to enumerate the device and load its driver. This is achieved by the following code sequence:

```
USBUART_Start(0u, USBUART_1_3V_OPERATION);
while (!USBUART_GetConfiguration()){};
USBUART_CDC_Init();
```

Part 1: Transmission and Reception of data using programmed IO

- For this part, you can use the configuration of USBFS in the example with no changes.
- Define two 64-byte arrays to serve as receive buffers: **RxBuffer0** and **RxBuffer1**. These are to be used for storing the received data in ping-pong fashion.
- Your program should initialize the USB interface and periodically poll for any data received from the PC host. If any data is received, the bytes should be stored consecutively in the receive buffers, starting with RxBuffer0 and switching to RxBuffer1 when it becomes full, and returning back to RxBuffer0 when the latter becomes full (thus cyclically switching back and forth between the buffers).
- When any of the buffers becomes full, the program should transmit its entire contents (64 bytes) as a single block to the PC host.
- You may find it easier to test the code with a small buffer size (say 4) and increase it to 64 after it is fully functional.
- You should use the API function **USBUART_DataIsReady()** to poll the receive side and **USBUART_GetAll()** to read the RX data. Similarly, you should use **USBUART_ CDCIsReady()** to check if the transmit side is ready and then use **USBUART_PutData()** to transmit the block of data.

Note: USB protocols format the data being transferred as sequences of bytes, called packets. The size of each packet can vary from 0 through 64 bytes. When you use **USBUART_PutData(),** the data specified in the call is sent as a single packet on the USB link. A complete transfer may involve more than one packet. The host will interpret the end of a transfer when it receives any packet of size less than the maximum size (64 bytes). This means if you call **USBUART_PutData()** with a length argument of 64, the host will receive the entire block of data in a single maximum-size packet, and will wait to receive more packets. Thus, when sending a 64-byte block, you will need to follow a call to the **USBUART_PutData()** function with another call specifying zero as the length to complete the transfer.

CHECKOFFS:

1. Demonstrate the operation of your program using the HyperTerminal application on the PC. The program should not echo any data until 64 bytes are accumulated in the RX buffer, and should then send the 64-byte block at once to the terminal. Cut and paste a block of characters (more than 128 bytes) into the terminal window to verify that the entire data is echoed back with no losses.
2. The program should not have any blocking code: For example, if the transmitter is not ready when 64 bytes have been accumulated to send, the program should still continue to receive any new data arriving from the host.

Part 2: Transmission and Reception of data using DMA

In this part, you will modify the code from Part 1 to use DMA for both the inbound and outbound transfer of data. A DMA controller is built into the USB component, so you do not need to use external DMA components. Because the USB UART does not support interrupts, you will need to use polling to check for the completion of DMA transfers.

- Change the configuration of the USBFS component to use DMA: Under the Device Descriptor tab, select "DMA with Manual Memory Management."
- Under the CDC Descriptor tab, find the two Endpoint Descriptors for CDC Interface 2. These identify the transmit and receive directions. Note the Endpoint Numbers for the IN and OUT descriptors. The IN direction indicates the data transfer from the PSoC-5 device to the PC host, and the OUT direction describes the transfer from the host to the PSoC-5 device. Note the Endpoint numbers for each direction (these are usually 2 and 3).
- Set up a DMA for the receive side (from the USB UART to one of the RX buffers) using the API call

  USBUART_ReadOutEP(epNumber, buffer, length);

  Where buffer is one of the RX buffers, and length should be set to 64. This API function is non-blocking, so you will need to check the status of the DMA later using the function USBUART_GetEPState() and switch to the other RX buffer when DMA is complete.

- When 64 bytes are available in one of the RX buffers, start a DMA on the transmit side using the API call

  USBUART_LoadInEP(epNumber, buffer, length);

  where buffer is one of the RX buffers, and length should be set to 64. This API function returns only after the entire data block has been transferred to the USB component, so there is no need to check the status of the DMA.

CHECKOFFS:

1. Demonstrate the operation of your program using the terminal application on the PC. The program should not echo any data until 64 bytes are accumulated in the RX buffer, and should then send the 64-byte block at once to the terminal.
2. The program should not result in any data loss. For example, while the DMA in the transmit direction is transferring a 64-byte block from one of the RX buffers, the DMA on the receive side must be enabled to transfer incoming data to the other RX buffer. Cut and paste a block of characters (more than 128 bytes) into the terminal window to verify that the entire data is echoed back with no losses.

What to submit?

- Schematics and code for all parts
- Descriptions of your designs
- Discussion of any problems encountered
- The built-in DMA controller of the USB component allows only data to be transferred from/to the main memory. This is sufficient for most applications, but there are cases where data needs to be transferred from the USB interface to another peripheral (for example, a DAC). State how can you implement such a transfer without the software performing a byte-by-byte copy from memory to the peripheral (only a description of the idea is needed, you don't need to implement the idea).