Lab 6 Report

EE 4341

Spring 2016

Cole Nielsen niels538@umn.edu

Abstract

The enumuration process of the USB serial bus was ran through using a PICDEM FS USB board without using Microchip firmware by directly controlling the PIC's hardware SIE (Serial Interface Engine) through its registers. Debug information pertaining to the enumuration process and USB packet information was then coded on the PIC to transmit over a USART/RS232 interface to a computer, where the data was then displayed.

1 Introduction

This lab was focused on understanding USB (universal serial bus) and the SIE (serial interface engine) of the PIC microcontroller in order to perform a USB enumuration for the PIC board connected to a PC host. USB is a industry standard serial communication interface commonly used on computers to connect to peripheral devices. USB is a host controlled protocal, meaning the host (typically a computer) controls all transfers/transactions that occur. One important type of USB transaction event is enumeration, which is where the host solicits the connected device for descriptor information so the host then can load drivers to control the device. In this lab, part of an emuration process was performed using the PICDEM FS board, by controlling the SIE in the board's microcontroller. The SIE is a hardware peripheral of the PIC that automatically handles the transmission of packets, CRC generation/error detection, handshaking and so on for USB. The microcontroller simply has to tell the SIE what it wants to do, and the SIE takes care of all other details for performing USB transactions. Usage of the SIE in this lab was implemented using provided sample code, and then code was implemented to transmit debugging information about the enumeration process to a computer via RS232.

2 Implementation

Implementation for this lab was performed using a given help file, which implemented the SIE control for enumeration. Using this file, code was written to output debugging information related to the enumeration process.

The Provided Code

The provided code for this lab will be first discussed to give an understanding of how it is used to utilize the SIE in order to perform a USB enumeration process. First, using preprocessor directives, regions of ram are set up to be used by the various registers needed to control the SIE. This code is below. There are a couple types of registers used, one for each transmission direction: BDnSTAT_X is for status information, BDnADDRX_X is used to set the address for transmission, BDnCNT_X is used to set the transmission size in bytes. The SIE is able to access these RAM locations without processor intervention, so as soon as these RAM locations are written for a transaction by the processor, it can move on and the SIE will handle everything.

```
#pragma udata BDT = 0x400
volatile ram char BD0STAT_OUT;
volatile ram char BD0CNT_OUT;
volatile ram char BD0ADRL_OUT;
volatile ram char BD0ADRH_OUT;
volatile ram char BD0STAT_IN;
volatile ram char BD0CNT_IN;
volatile ram char BD0ADRL_IN;
volatile ram char BD0ADRH_IN;
#pragma code

#pragma udata PKTOUT = 0x500
volatile ram char Data_OUT[64];
#pragma code

#pragma udata PKTIN = 0x600
volatile ram char Data_IN[64];
#pragma code
```

The main function does very little other than print out a header line via USART showing "USB FIRMWARE LAB" and to instaniate the USB initialization function. After this, there is an empty loop that basically just uses up spare processor cycles. The initialization function configures both the USART and USB to function in th intended mode for this lab. In particular, interrupts are set up for the USB, as all USB operations are handled via interrupts in the code. The initial addresses, transmission size and statuses of the USB data registers are also set, and the device descriptor is written in the Data_IN RAM buffer. There is also a function ascii() implemented that converts a decimal number into its corresponding ASCII character so the data can be properly read through the serial terminal read out. The important piece of code is the ISR, which handles the USB events. The ISR is set to handle all interrupt events in the one ISR. Essentially the ISR works by unsetting the USB and interrupt flags, and then goes through a number of if-else statements and Case statements, using the status register, USTAT register and Data buffer to determinine the PID type/event type. If a setup packet is detected, it prints out "STP", this is the first step of the enumeration, and then the Data buffer is check to see if the received data corresponds to a request, which "GETDEV" is printed out for. The number of bytes pertaining to transmission length is also written (in this case 40 Bytes). If instead an OUT token is detected by the case, rather than SETUP, "OUT" is printed. An enumeration process consists first of a setup token, so it is expected the first string of text read via RS-232 should be "STP-GETDEV-40". Another important event this ISR detects (relevant to this lab) is the reset detect, which checks the USB interruput reset flag, and if it is set, "RST" will be printed indicating a reset. The USB bus resets numerous times in the enumeration process, so this is important to see.

```
#pragma interrupt high_ISR
void high_ISR(void)
```

```
{
   UCONbits.PKTDIS = 0;
   PIR2bits.USBIF = 0;

   if(UIRbits.TRNIF) {
   // while (!TXSTAbits.TRMT); TXREG = 'T';
   // while (!TXSTAbits.TRMT); TXREG = 'E';
      //SCI_newline();

      if(!(USTAT & 0x04)) //OUT transaction?
      {
         switch((BD0STAT_OUT>>2)&0x0F) //token PID
         {
            case SETUP:
               while (!TXSTAbits.TRMT); TXREG = 'S';
               while (!TXSTAbits.TRMT); TXREG = 'T';
               while (!TXSTAbits.TRMT); TXREG = 'P';
               while (!TXSTAbits.TRMT); TXREG = '-';
               set_device_descriptor();
               if(Data_OUT[1] == GETDEV) //SETUP request type == GETDEV?
               {
                  while (!TXSTAbits.TRMT); TXREG = 'G';
                  while (!TXSTAbits.TRMT); TXREG = 'E';
                  while (!TXSTAbits.TRMT); TXREG = 'T';
                  while (!TXSTAbits.TRMT); TXREG = 'D';
                  while (!TXSTAbits.TRMT); TXREG = 'E';
                  while (!TXSTAbits.TRMT); TXREG = 'V';
                  while (!TXSTAbits.TRMT); TXREG = '-';
               }
               while (!TXSTAbits.TRMT); TXREG = ascii(Data_OUT[6]>>4); //
                   DATA_OUT[6]=number of bytes requested
               while (!TXSTAbits.TRMT); TXREG = ascii(Data_OUT[6]&0x0F);
               SCI_newline();
               break;
            case OUTTOKEN:
               while (!TXSTAbits.TRMT); TXREG = 'O';
               while (!TXSTAbits.TRMT); TXREG = 'U';
               while (!TXSTAbits.TRMT); TXREG = 'T';
               SCI_newline();
               break;
            default:
               while (!TXSTAbits.TRMT); TXREG = ascii((BD0STAT_OUT>>2)&0x0F);
         }
         dump_info(); //print out everything
         while(BD0STAT_OUT&0x08);
         BD0STAT_OUT = 0x80; //UOWN= 1, control back to SIE
      } else { //in token
         switch((BD0STAT_IN>>2)&0x0F)
         {
            case INTOKEN:
         //    while (!TXSTAbits.TRMT); TXREG = 'I';
         //    while (!TXSTAbits.TRMT); TXREG = 'N';
               //SCI_newline();
               break;
         }
         while(BD0STAT_IN&0x08);
         BD0STAT_IN = 0x80;
      }

      UIRbits.TRNIF = 0;
   }
   if (UIRbits.URSTIF) {

      while (!TXSTAbits.TRMT); TXREG = 'R';
      while (!TXSTAbits.TRMT); TXREG = 'S';
      while (!TXSTAbits.TRMT); TXREG = 'T';
      while (!TXSTAbits.TRMT); TXREG = 10;
      while (!TXSTAbits.TRMT); TXREG = 13;
      UIRbits.URSTIF = 0;

   }
   if(UIRbits.SOFIF) {
      UIRbits.SOFIF = 0;
   //while (!TXSTAbits.TRMT); TXREG = 'F';
   }
   if(UIRbits.UERRIF) {
      while (!TXSTAbits.TRMT); TXREG = 'E';
      UIRbits.UERRIF = 0;
   }
}
```

Implemented Debug Code

In order to display debug information about the enumeration process, code was added in the dump_info() function (code below)to write out the data transaction information (data, byte count, status) when that type of transaction occurs. The dump_info() function is called everytime the ISR statements pertaining to data transactions are executed. This function is used entirely for transmitting USB register/buffer values over USART. First the RAM buffer values for IN/OUT are printed out on two lines using corresponding functions. These values printed represent the latest received/transmitted values, and for the enumeration should contain the device descriptor in the IN buffer and the host and SETUP packet information in the DATA RAM buffer. The values of UCON (USB control register) are then printed out in ASCII, preceded by a label "UCON ". Then the USTAT (USB status register) is printed out with its label "USTAT ". Finally, UEP0 (USB endpoint register) is printed with its label. These register values provide valuable information regarding type of previous transmission, endpoint and configuration. Next, the DATA IN/OUT (BDnSTAT_X) status register information is written out in the same manner, as well as the data length registers (BDnCNT_X). These give information regarding length of the latest packets and status details.

```
void dump_info() {

   int i;

   SCI_newline();

   //Display BuffRX here
   //name first
   display_data_rx();
   SCI_newline();

   //Display BuffTX
   //name first
   display_data_tx();
   SCI_newline();

   //Display UCON
   //name first
    while (!TXSTAbits.TRMT); TXREG = 'U';
    while (!TXSTAbits.TRMT); TXREG = 'C';
    while (!TXSTAbits.TRMT); TXREG = 'O';
    while (!TXSTAbits.TRMT); TXREG = 'N';
    while (!TXSTAbits.TRMT); TXREG = ' ';

   while (!TXSTAbits.TRMT); TXREG = ascii(UCON>>4);
   while (!TXSTAbits.TRMT); TXREG = ascii(UCON & 0x0F);

   while (!TXSTAbits.TRMT); TXREG = ' ';

   // Display USTAT (similar method to UCON)
   //name first
   //implement here
    while (!TXSTAbits.TRMT); TXREG = 'U';
    while (!TXSTAbits.TRMT); TXREG = 'S';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = 'A';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = ' ';
   while (!TXSTAbits.TRMT); TXREG = ascii(USTAT>>4);
   while (!TXSTAbits.TRMT); TXREG = ascii(USTAT & 0x0F);
    while (!TXSTAbits.TRMT); TXREG = ' ';

   //Display UEP0
   //name first
    while (!TXSTAbits.TRMT); TXREG = 'U';
    while (!TXSTAbits.TRMT); TXREG = 'E';
    while (!TXSTAbits.TRMT); TXREG = 'P';
    while (!TXSTAbits.TRMT); TXREG = '0';
    while (!TXSTAbits.TRMT); TXREG = ' ';

   while (!TXSTAbits.TRMT); TXREG = ascii(UEP0>>4);
   while (!TXSTAbits.TRMT); TXREG = ascii(UEP0 & 0x0F);
   SCI_newline();

    while (!TXSTAbits.TRMT); TXREG = 'B';
    while (!TXSTAbits.TRMT); TXREG = 'D';
    while (!TXSTAbits.TRMT); TXREG = '0';
    while (!TXSTAbits.TRMT); TXREG = 'S';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = 'A';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = '_';
    while (!TXSTAbits.TRMT); TXREG = 'O';
    while (!TXSTAbits.TRMT); TXREG = 'U';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = ' ';

   //Display BD0STAT_OUT
   //name first

   while (!TXSTAbits.TRMT); TXREG = ascii(BD0STAT_OUT>>4);
   while (!TXSTAbits.TRMT); TXREG = ascii(BD0STAT_OUT & 0x0F);
   SCI_newline();

    while (!TXSTAbits.TRMT); TXREG = 'B';
    while (!TXSTAbits.TRMT); TXREG = 'D';
    while (!TXSTAbits.TRMT); TXREG = '0';
    while (!TXSTAbits.TRMT); TXREG = 'C';
    while (!TXSTAbits.TRMT); TXREG = 'N';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = '_';
    while (!TXSTAbits.TRMT); TXREG = 'O';
    while (!TXSTAbits.TRMT); TXREG = 'U';
    while (!TXSTAbits.TRMT); TXREG = 'T';
    while (!TXSTAbits.TRMT); TXREG = ' ';
```

```
  //Display BD0CNT_OUT
  //name first
  //implement here (similar to BD0STAT_OUT)

  while (!TXSTAbits.TRMT); TXREG = ascii(BD0CNT_OUT>>4);
  while (!TXSTAbits.TRMT); TXREG = ascii(BD0CNT_OUT & 0x0F);
  SCI_newline();


   while (!TXSTAbits.TRMT); TXREG = 'B';
   while (!TXSTAbits.TRMT); TXREG = 'D';
   while (!TXSTAbits.TRMT); TXREG = '0';
   while (!TXSTAbits.TRMT); TXREG = 'S';
   while (!TXSTAbits.TRMT); TXREG = 'T';
   while (!TXSTAbits.TRMT); TXREG = 'A';
   while (!TXSTAbits.TRMT); TXREG = 'T';
   while (!TXSTAbits.TRMT); TXREG = '_';
   while (!TXSTAbits.TRMT); TXREG = 'I';
   while (!TXSTAbits.TRMT); TXREG = 'N';
   while (!TXSTAbits.TRMT); TXREG = ' ';
  //Display BD0STAT_IN
  //name
  //implement (similar to BD0STAT_OUT)
  while (!TXSTAbits.TRMT); TXREG = ascii(BD0STAT_IN>>4);
  while (!TXSTAbits.TRMT); TXREG = ascii(BD0STAT_IN & 0x0F);
  SCI_newline();

   while (!TXSTAbits.TRMT); TXREG = 'B';
   while (!TXSTAbits.TRMT); TXREG = 'D';
   while (!TXSTAbits.TRMT); TXREG = '0';
   while (!TXSTAbits.TRMT); TXREG = 'C';
   while (!TXSTAbits.TRMT); TXREG = 'N';
   while (!TXSTAbits.TRMT); TXREG = 'T';
   while (!TXSTAbits.TRMT); TXREG = '_';
   while (!TXSTAbits.TRMT); TXREG = 'I';
   while (!TXSTAbits.TRMT); TXREG = 'N';
   while (!TXSTAbits.TRMT); TXREG = ' ';
  //BD0CNT_IN
  //implement similar here as BD0CNT_out
  while (!TXSTAbits.TRMT); TXREG = ascii(BD0CNT_IN>>4);
  while (!TXSTAbits.TRMT); TXREG = ascii(BD0CNT_IN & 0x0F);
  SCI_newline();

}
```

3 Summary

The outcome of this lab was a function code for the debugging of USB events. The code was shown to be functional as when the board was connected to a USB Host, several reset events were observed through its output (as expected), followed by the proper setup sequence and buffer data print out. This code is of value as a Windows host PC provides very little debug information for USB, so a code like this can be invaluable to have when developing USB-based devices for figuring exactly what the bus is doing.