USB HID Keyboard Firmware Using EZ-USB® FX2LP™

EP65575

Project Name: HID_KB
Programming Language: C
Associated Part Families: CY7C68013A,CY7C68014A
Software Version: Keil uVision2
Related Hardware:CY3684(FX2LP) DVK Board
Author: Narayana Murthy

Project Objective

This example project describes the implementation of a USB HID Class keyboard using the FX2LP™ development kit board. The project uses the four push buttons available on the development board to demonstrate the keyboard functionality.

Overview

This example project documents the USB HID keyboard (hid_kb) firmware example available along with the CY3684 (FX2LP) development kit software. The HID class specification supports Interrupt endpoint transfers because of guaranteed bandwidth, error-free transmission and the flexibility to change the latency to access the data from a HID device. The firmware in this project uses Interrupt endpoints EP1IN and EP1OUT for data transfers.

System Requirements

Software

- Keil uVision2 to edit, compile or debug the firmware project

Firmware

- The **hid_kb** firmware example part of the CY3684 development kit software is used here.

Hardware

- CY3684 FX2LP development kit board is used to demonstrate the example project
    - USB (A-B) cable to connect FX2LP board to PC or laptop
    - PC or laptop hosting Windows OS with ability to detect HID keyboard device using an in-built native driver (*usbhid.sys*).

Operation

Initialization

The initialization of the FX2LP, that is, all components involved in the HID keyboard implementation, is done in the *TD_Init()* function, which is the first function called in *main()*. For the HID class devices like keyboard, Mouse, the USB bandwidth requirements is typically 64 KB/sec. Most of the HID devices are either low-speed or full-speed devices. Due to this low data rate requirements of the device, only the endpoint EP1(64 byte buffer) is selected for both IN and OUT interrupt transfers. The high-speed data endpoints EP2, EP4, EP6, and EP8 are disabled as shown in the following code snippet:

```
EP1OUTCFG = 0xB0;        // valid, interrupt OUT, 64 bytes, Single buffered

EP1INCFG = 0XB0;         // valid, interrupt IN, 64 bytes, Single buffered

EP2CFG = EP4CFG = EP6CFG = EP8CFG = 0;    // disable unused endpoints
```

HID Keyboard Implementation

The HID class firmware is based on cypress EZ-USB® firmware framework. Additional components like functions, code, descriptors related to HID were added to this basic framework to convert it into a HID class keyboard firmware. The firmware can be classified into three essential parts to describe its functionality as a HID class keyboard. The HID class keyboard related implementation in the firmware is also highlighted as a part of the description.

Descriptor Information

The Host PC during enumeration requires a set of standard descriptor data to properly identify the type of the device with which it is communicating. The firmware in this case should also contain HID class descriptors. The entire descriptor information for the device is stored in *dscr.a51* file. The important parts in each descriptor and specifically, the HID data associated with each of them are highlighted as follows in a sequential order:

- **Device Descriptor:** The host enumerates the device with a VID/PID value of 0x4B4/0x1005 defined in this descriptor
- **Configuration Descriptor:** The descriptor indicates the number of interfaces and the configuration number of the device to the Host PC. In the firmware, both the parameters are set to 1.
- **Interface Descriptor:** There are several USB HID Class devices like mouse, keyboard, game controller and alphanumeric displays. Following is the 9-byte format of this descriptor data to indicate a HID class keyboard device to Host PC.

```
db    DSCR_INTRFC_LEN       ;; Descriptor length
db    DSCR_INTRFC           ;; Descriptor type
db    0                     ;; Zero-based index of this interface
db    0                     ;; Alternate setting
db    2                     ;; Number of end points
db    03H             ;; Interface class: HID
db    00H                 ;; Interface sub class
db    00H                 ;; Interface sub sub class
db    0               ;; Interface descriptor string index
```

- **HID Descriptor:** This descriptor is specific to the HID class devices' HID class specification. The device adheres to the length of the Report descriptor of a HID device. The important fileds in the HID descriptor are highlighted.

HIDDscr:

```
db    09h      ; length
db    21h      ; type: HID
db    010h,01h      ; release: HID class rev 1.1
db    00h      ; country code (none)
db    01h      ; number of HID class descriptors to follow
db    22h      ; report descriptor type (HID)
db    (HIDReportDscrEnd - HIDReportDscr) ; length of HID descriptor
db    00h
```

HIDDscrEnd:

- **Endpoint Descriptor:** To transfer data to the host, the firmware needs to support the Interrupt IN transfers. To receive data, the firmware must support an Interrupt OUT endpoint. EP1IN and EP1OUT are used as Interrupt endpoints in the firmware. The Host requests data from the device on both the endpoints in a periodical time duration as mentioned in the device endpoint descriptors. The firmware sets this value to 10 m/sec that is highlighted in the following section in the EP1IN and EP1OUT descriptors.

;; Endpoint Descriptor

```
db    DSCR_ENDPNT_LEN     ;; Descriptor length
db    DSCR_ENDPNT         ;; Descriptor type
db    81H                 ;; Endpoint number, and direction
db    ET_INT              ;; Endpoint type
db    40H                 ;; Maximum packet size (LSB)
db    00H                 ;; Max packet size (MSB)
db    0AH                 ;; Polling interval
```

;; Endpoint Descriptor

```
db    DSCR_ENDPNT_LEN      ;; Descriptor length
db    DSCR_ENDPNT          ;; Descriptor type
db    01H                  ;; Endpoint number, and direction
db    ET_INT               ;; Endpoint type
db   40H                 ;; Maximum packet size (LSB)
db   00H                 ;; Max packet size (MSB)
db   0AH                 ;; Polling interval
```

- **Report Descriptor:** This descriptor provides definition of data provided by each control of the HID device. Reports can be of one of three types: *Input, Output, and Feature*. Data sent to the host is carried in an Input report, data received by the device is contained in an Output report, and a Feature report may represent data flowing in either direction. Following is the Report descriptor for the HID keyboard taken from page 79 of the HID class spec version 1.1.

  HIDReportDscr:

```
db 05h, 01h     ; Usage Page (Generic Desktop)
db 81h, 02h      ;      Input (data, variable, absolute)
db 95h, 01h      ;      Report count (1)
db 75h, 08h      ;      Report size (8)
db 81h, 01h      ;      Input (constant)
db 95h, 05h      ;      Report count (5)
db 75h, 01h      ;      Report size (1)
db 05h, 08h      ;      Usage Page (LED)
db 19h, 01h      ;      Usage minimum (1)
db 29h, 05h      ;      Usage maximum (5)
db 91h, 02h      ;      Output (data, variable, absolute)
db 95h, 01h      ;      Report count (1)
db 75h, 03h      ;      Report size (3)
db 91h, 01h      ;      Output (constant)
db 95h, 03h      ;      Report count (3)
db 75h, 08h      ;      Report size (8)
db 15h, 00h      ;      Logical minimum (0)
db 25h, 65h      ;      Logical maximum (101)
db 05h, 07h      ;      Usage page (key codes)
db 19h, 00h      ;      Usage minimum (0)
db 29h, 65h      ;      Usage maximum (101)
db 81h, 00h      ;      Input (data, array)
db 0C0h          ; End Collection
```

  HIDReportDscrEnd:

HID Class Request Support

The HID and Report descriptors mentioned here are sent to the HOST PC by the device using two HID class specific requests: GET_HID_DESRIPTOR and GET_REPORT_DESCRIPTOR. Both these requests are serviced using the **DR_GetDescriptor** () function in the firmware *periph.c* source file. Following is the function definition that handles both these requests on Control Endpoint (EP0).

```
BOOL DR_GetDescriptor(void)
{
      BYTE length,i;

      pHIDDscr = (WORD)&HIDDscr;
      pReportDscr = (WORD)&HIDReportDscr;
      pReportDscrEnd = (WORD)&HIDReportDscrEnd;

      switch (SETUPDAT[3])
      {
            case GD_HID:                    //HID Descriptor
            SUDPTRH = MSB(pHIDDscr);
            SUDPTRL = LSB(pHIDDscr);
            return (FALSE);
            case GD_REPORT:                 //Report Descriptor
            length = pReportDscrEnd - pReportDscr;

              AUTOPTR1H = MSB(pReportDscr);
              AUTOPTR1L = LSB(pReportDscr);

              for(i=0;i<length;i++)
                 EP0BUF[i]=XAUTODAT1;
```

```
                        EP0BCL = length;
                        return (FALSE);

                      default:
                      return(TRUE);
                }
        }
```

HID Data Reports

For a typical HID device, the data related to events like button press, key strokes, mouse clicks are transferred to Host in the form of Input Reports using an Interrupt IN endpoint. Similarly, the Reports can be requested by Host PC using control endpoint or an Interrupt OUT endpoint. The firmware sets EP1IN and EP1OUT as Interrupt endpoints for data transfers. The following table summarizes the mapping of Push buttons on the FX2LP development board to keyboard buttons.

Table 1. Function Mapping of Development Board Buttons

Development Board Button	Function
f1	Shift
f2	Send ‘a’
f3	Send ‘b’
f4	Send ‘c’

The function TD_poll() in the firmware (*periph.c*) is where the periodic checking of a new push button event is done. Following is the code snippet of this function.

```
        void TD_Poll(void)             // Called repeatedly while the device is
idle
        {
                if( !(EP1INCS & bmEPBUSY) )      // Is the EP1INBUF available,
               {
                EZUSB_ReadI2C(BTN_ADDR,0x01,&buttons); // Read button states
                     buttons &= 0x0F;
                     if ((oldbuttons - buttons) != 0)  //Change in button state
                     {

                               if (buttons & 1)  //Shift
                                  EP1INBUF[0] = 0;
                               else
                            EP1INBUF[0] = 2;

                         if (buttons & 2)  //a
                            EP1INBUF[2] = 0;
                         else
                            EP1INBUF[2] = 4;

                         if (buttons & 4)  //b
                            EP1INBUF[3] = 0;
                         else
                            EP1INBUF[3] = 5;

                         if (buttons & 8)  //c
                            EP1INBUF[4] = 0;
                         else
                            EP1INBUF[4] = 6;

                      EP1INBUF[1] = 0;
                      EP1INBC = 5;
               }
               oldbuttons = buttons;
               }
        ........................
        }
```

The code inside this function detects if the user presses any of the four push buttons mentioned in the Function Mapping table. The device sends them as equivalent keystroke data (as per the table) in a Report format. Following is the 8-byte input Report format for a HID keyboard from the HID spec document. The firmware uses first five bytes to send an Input report.

Table 2. Report Format of a Keyboard

Byte	Description
0	Modifier keys
1	Reserved
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

How to Test the Project

1. Download SuiteUSB 3.4.2 and install it. This installs a utility named as Control Centre
2. Connect the CY3684 (FX2LP DVK) board to the PC. It enumerates with the default internal descriptor. Use the *CyUSB.inf* file inside the driver's folder to bind with the device. If you have any doubt in binding the driver, refer to the Matching Driver To USB Device section in *CyUSB.chm* inside the Drivers folder
3. Download the *hid_kb.hex* file to the FX2LP board using the Control Centre utility. The *hid_kb.hex* file is present in firmware folder in the attached project
4. Open Device Manager Window by typing *devmgmt.msc* in Windows Start → Run box. The device will be shown as part of the HID devices list
5. Open a new notepad in windows and point the mouse to the text area of the notepad. Press button F2, F3, F4 sequentially and observe letters 'a, b, c' getting printed in the notepad. Press them along with F1 to get capital letters.

Note To detect the FX2LP device and download the firmware as mentioned in steps (2) and (3) Cypress drivers support is required along with a Windows based PC or laptop. After downloading, to test the firmware as a HID keyboard device any PC or laptop OS that has in-built native driver to detect the HID keyboard device should be sufficient.

Upgrade Information

This project was created in Windows XP and the driver works well in Windows XP. For other operating systems (32 bit or 64 bit), the *CyUSB.inf* and *CyUSB.sys* should be replaced with the files for appropriate OS. Drivers for other Windows OS versions are present in the Driver folder in the project attachment.

Summary

This example project configures FX2LP board such that it resembles a HID keyboard device and it implements keystroke functions using the four push buttons available on FX2LP development Board.

Document History

Document Title: USB HID Keyboard Firmware Using EZ-USB® FX2LP™

Document Number: 001-65575

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3096856	NMMA	12/16/2010	New Example Project

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com/