Creating a FX1/FX2LP™ Composite HID Device

Author: Anand Srinivasan Asokan

Associated Project: Yes

Associated Part Family: FX1/FX2LP™

Software Version: Keil uVision2

More code examples? We heard you.

For a consolidated list of USB Hi-Speed Code Examples, visit this page.

This application note describes how to implement a composite human interface device (HID) using FX1/FX2LP™. The example firmware explained here is a two-button mouse and a two-button keyboard function, using the four buttons on the FX1/FX2LP development board. The example firmware also has an interface which implements a loopback over bulk endpoint using Endpoints 6 and 8 to emphasize the flexibility and bandwidth capability of the device. This document assumes that the reader is familiar with the HID specification, HID usage tables, and FX1/FX2LP. In this application note, the term 'EZ-USB' refers to FX1 and FX2LP, except where noted.

Contents

1 Introduction

The EZ-USB family of chips is usually used in medium to high-end applications. However, in some cases, developers may still choose to create a HID device based on the EZ-USB family because of its ease of programming and 'soft' RAM architecture. Moreover, driver development is not needed because most operating systems have a native HID driver. Due to these factors, an EZ-USB device can be used under the HID class.

Developers can choose the HID class for their device if it is able to behave within the parameters of the HID specification.

The first major check-off item to help determine whether a device fits in the HID class is to make sure that the USB bandwidth required at full-speed is no greater than 64 KB/s. Any device that requires some form of human control and falls under the class's limits is a candidate for HID. Some examples of applications are mice, keyboards, video display controls, speakers, barcode readers, thermometers, and voltmeters.

2 Firmware Basics

The device's firmware must meet the following basic requirements to function as a HID:

- The device's descriptor table must indicate that the device is using the HID interface
- The firmware must support at a minimum, an interrupt IN endpoint along with the default control pipe (endpoint zero) for data transfers

- The descriptor table must also contain report descriptors that define a structure for the transmission and reception of data. The report descriptors are important because they describe the size and contents of the data being transferred (contained in reports) by a HID and what the recipient should do with this data. Devices may support more than one report, which is the case in this example

To send data to the host, the firmware must support the GET_REPORT (01h) HID class request and Interrupt IN transfers. To receive data, the firmware must support the SET_REPORT (09h) HID class request and/or an Interrupt OUT endpoint. All HID class requests go through endpoint zero, just like standard USB requests.

Reports can be one of the following 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 represents configuration data flowing in either direction.

3 Firmware

The firmware is based on the EZ-USB firmware framework files (*fw.c*, *EZUSB.LIB*, *fx2regs.h*, and so on). After installing CY3684 EZ-USB FX2LP Development Kit, these can be found in C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Target\ properly sub-divided in folders (the location varies based on installation). See *EZ-USB® Development Kit User Guide.pdf,* found in C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation (the location varies based on installation), for details on firmware frameworks. The firmware attached with this application note is for a two-button mouse/two-button keyboard function using the four buttons on the EZ-USB development board plus an interface which performs loopback over bulk endpoint.

The following sections describe modifications to the firmware frameworks for creating the firmware.

3.1 DSCR.A51

An HID device is no different from any other device as the Host still needs to learn about its capabilities. However, an HID device sends an HID Descriptor to let the Host know that the connected device is of HID class. One or more report descriptors are sent to define the format and use of the data being sent. In this example, the data reports represent mouse button clicks and keyboard keystrokes.

An HID device that complies with the HID 1.1 specification uses the following order in the descriptor table:

- Device descriptor
- Configuration descriptor
- Interface descriptor (the class field of this descriptor defines the device as an HID class device)
- HID descriptor (one for each HID interface)
- Endpoint descriptors
- Report descriptor/physical descriptor (this example does not use a physical descriptor)

3.1.1 Device Descriptor

The Vendor ID (VID)/Product ID (PID) in the device descriptor in this example is 0x04B4/0x1004 (the same as that used in the firmware framework).

3.1.2 Configuration Descriptor

The value in the *bNumInterfaces* field determines whether Windows recognizes the device as a **composite device**. Here, the *bNumInterfaces* field is changed to ‘3’ because there are three interfaces.

By indicating that there is more than one interface, Windows detects a composite device (it loads the USB generic parent driver, *usbccgp.sys,* and exposes the three interfaces as three different devices), and then proceeds to load the driver that talks to each interface. In this case, there will be two instances of the HID driver and one instance of *CyUSB.sys*. The latest version of the *CyUSB.sys* driver can be obtained from FX3 SDK. For details on inf file modification, see AN61465 - Working With inf File of a Device Using *CyUSB.sys*.

3.1.3 Interface Descriptor

This example has three interfaces, each with a corresponding interface descriptor. The three interfaces are as below:

- Mouse Interface
- Keyboard Interface
- Vendor Interface/ Loopback over bulk Interface

The following sections describe the various interface descriptors for each of these interfaces.

3.1.3.1 Interface/HID/Endpoint Descriptors (Mouse Interface)

Code 1 (taken from the project in this application note) depicts the interface, HID, and endpoint descriptors for the mouse interface.

Code 1. Interface/HID/Endpoint Descriptors (Mouse Interface)

```
HID1IntrfcDscr:
db HID1IntrfcDscrEnd-HID1IntrfcDscr;; Descriptor length
db DSCR_INTRFC ;; Descriptor type
db 00H ;; bInterfaceNumber
db 0 ;; Alternate setting
db 1 ;; Number of end points
db 03H ;; Interface class (HID)
db 01H ;; Boot Interface sub class
db 02H ;; Interface sub sub class (Mouse)
db 0 ;; Interface descriptor string index
HID1IntrfcDscrEnd:

HID1Dscr:
db 09h ;; length
db 21h ;; type: HID
db 10h,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 (HID1ReportDscrEnd-HID1ReportDscr) ;; length of HID
;; descriptor
db 00h
HID1DscrEnd:

HID1EpInDscr:
db HID1EpInDscrEnd-HID1EpInDscr;; Descriptor length
db DSCR_ENDPNT ;; Descriptor type
db 82H ;; Endpoint number, and direction
db ET_INT ;; Endpoint type
db 40H ;; Maximum packet size (LSB)
db 00H ;; Max packet size (MSB)
db 10 ;; Polling interval
HID1EpInDscrEnd:
```

Note that in the interface descriptor for the mouse, the *Interface class* field, *bInterfaceClass*, is of type HID (03H). The next two fields, *bInterfaceSubClass* and *bInterfaceProtocol*, allows the BIOS, which support the mouse boot protocol, to communicate with this device (optional). Also of significance is the *bInterfaceNumber* field, which tells the Host that the number of this particular mouse interface is'0'. This value is what the firmware uses to determine which particular HID and report descriptor to load.

The endpoint descriptor for the mouse interface shows that endpoint '2 IN' is used as the interrupt pipe. The polling interval of 10 ms is arbitrary for this example.

3.1.3.1.1 *Report Descriptor (Mouse)*

The report descriptor for the mouse interface is taken from the HID specification (version 1.1) available for download from http://www.usb.org/developers/hidpage/. This descriptor follows the protocol format shown in Table 1.

Code 2. Report Descriptor (Mouse)

```
HID1ReportDscr:
db 05h, 01h ; Usage Page (Generic Desktop),
db 09h, 02h ; Usage (Mouse),
db 0A1h, 01h ; Collection (Application),
db 09h, 01h ; Usage (Pointer),
db 0A1h, 00h ; Collection (Physical),
db 95h, 03h ; Report Count (3),
db 75h, 01h ; Report Size (1),
db 05h, 09h ; Usage Page (Buttons),
db 19h, 01h ; Usage minimum (1)
db 29h, 03h ; Usage maximum (3)
db 15h, 00h ; Logical minimum (0),
db 25h, 01h ; Logical maximum (1),
db 81h, 02h ; Input (Data, Variable, Absolute), (3 button bits)
db 95h, 01h ; Report Count (1),
db 75h, 05h ; Report Size (5),
db 81h, 01h ; Input (Constant)
db 75h, 08h ; Report Size (8)
db 95h, 02h ; Report Count (2)
db 05h, 01h ; Usage Page (Generic Desktop),
db 09h, 30h ; Usage (X),
db 09h, 31h ; Usage (Y),
db 15h, 81h ; Logical Minimum (-127),
db 25h, 7Fh ; Logical Maximum (+127),
db 81h, 06h ; Input (Data, Variable, Relative), (2 position
; bytes - X & Y)
db 0C0h ; End Collection
db 0C0h ; End Collection
HID1ReportDscrEnd:
```

Table 1. Report Format for Mouse Interface

Byte	Bits	Description
0	0 1 2 4 to 7	Button 1 Button 2 Button 3 (not used in this example) Device-specific (not used in this example. Assigned as zero)
1	0 to 7	X displacement (not used in this example. Assigned as zero)
2	0 to 7	Y displacement (not used in this example. Assigned as zero)
3 to n	0 to 7	Device-specific (optional) (not used in this example)

In this example, only buttons 1 and 2 (corresponding to the buttons f1 and f2 in CY3684 DVK) are used (that is, left-click and right-click). However, the firmware can be easily modified to implement button 3 and X-Y displacements.

3.1.3.2 Interface/HID/Endpoint Descriptors (Keyboard Interface)

Code 3 (taken from the project in this application note) depicts the interface, HID, and endpoint descriptors for the keyboard interface. They are similar to those of the mouse interface.

Code 3. Interface/HID/Endpoint Descriptors (Keyboard Interface)

```
HID2IntrfcDscr:
db HID2IntrfcDscrEnd-HID2IntrfcDscr;; Descriptor length
db DSCR_INTRFC ;; Descriptor type
db 01H ;; bInterface Number
db 0 ;; Alternate setting
db 1 ;; 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
HID2IntrfcDscrEnd:


HID2Dscr:
db 09h ;; length
db 21h ;; type: HID
db 10h,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 (HID2ReportDscrEnd-HID2ReportDscr) ;; length of HID
;; descriptor
db 00h
HID2DscrEnd:

HID2EpInDscr:
db HID2EpInDscrEnd-HID2EpInDscr;; 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 10 ;; Polling interval
HID2EpInDscrEnd:
```

The *bInterfaceNumber* field of this keyboard interface is 01H (remember that the interface number for the mouse is '0'). The firmware uses this value to decide when to load the HID and report descriptors for the keyboard. Endpoint 1 IN is used as Interrupt IN pipe.

3.1.3.2.1 Report Descriptor (Keyboard)

The report descriptor for the keyboard interface is taken from the HID specification, version 1.1 (only Input report used in this example). This descriptor follows the protocol format shown in

Table 2.

Code 4. Report Descriptor (Keyboard)

```
HID2ReportDscr:
db 05h, 01h ; Usage Page (Generic Desktop)
db 09h, 06h ; Usage (Keyboard)
db 0A1h, 01h ; Collection (Application)
db 05h, 07h ; Usage Page (Key codes)
db 19h, 0E0h ; Usage minimum (234)
db 29h, 0E7h ; Usage maximum (231)
db 15h, 00h ; Logical minimum (0)
db 25h, 01h ; Logical maximum (1)
db 75h, 01h ; Report size (1)
db 95h, 08h ; Report count (8)
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
HID2ReportDscrEnd:
```

Table 2. Report Format for Keyboard Interface

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

3.1.3.3 Interface/Endpoint Descriptors (Loopback Over Bulk Interface)

Code 5 (taken from the project in this application note) depicts the interface and endpoint descriptor of the loopback over bulk endpoint interface.

Code 5. Interface/Endpoint Descriptors (Loopback Over Bulk Interface)

```

;; Interface Descriptor
db DSCR_INTRFC_LEN      ;; Descriptor length
db DSCR_INTRFC          ;; Descriptor type
db 02H ;; Zero-based index of this interface
db 0                ;; Alternate setting
db 2                ;; Number of end points
db 0ffH             ;; Interface class
db 00H               ;; Interface sub class
db 00H            ;; Interface sub sub class
db 0    ;; Interface descriptor string index

;; Endpoint Descriptor
db DSCR_ENDPNT_LEN      ;; Descriptor length
db DSCR_ENDPNT          ;; Descriptor type
db 06H    ;; Endpoint number, and direction
db ET_BULK             ;; Endpoint type
db 00H        ;; Maximum packet size (LSB)
db 02H         ;; Max packet size (MSB)
db 00H               ;; Polling interval

;; Endpoint Descriptor
```

```
db DSCR_ENDPNT_LEN      ;; Descriptor length
db DSCR_ENDPNT          ;; Descriptor type
db 88H    ;; Endpoint number, and direction
db ET_BULK             ;; Endpoint type
db 00H         ;; Maximum packet size (LSB)
db 02H          ;; Max packet size (MSB)
db 00H               ;; Polling interval
```

3.2 FW.C

An additional hook is added to void the SetupCommand() function in FW.C to check for class requests. This ensures a cleaner code body by handling HID class requests in the function in *hid.C*, DR_ClassRequest(). Class requests are not used in this example, but firmware that handles the class requests like GET_REPORT,and SET_REPORT should be placed in this function. In this firmware, all the class requests are stalled.

Note: Among the class requests, GET_REPORT request handling is mandated by the HID specification. Similarly, GET_PROTOCOL and SET_PROTOCOL requests are mandated for HID devices with boot support.

3.3 hid.c

3.3.1 Initialization

TD_Init() takes care of EZ-USB initialization. The two interrupt IN endpoints and the two bulk endpoints are initialized using Code 6.

Code 6. EZ-USB Initialization

```
EP1INCFG = 0xB0;
SYNCDELAY;         // see TRM
EP2CFG = 0xF0;
SYNCDELAY;
EP6CFG = 0xA2;
SYNCDELAY;
EP8CFG = 0xE2;
SYNCDELAY;
EP6BCL = 0x80; // arm EP6OUT by writing byte count w/skip.
SYNCDELAY;
EP6BCL = 0x80;
SYNCDELAY;
```

The bulk endpoint EP6 has to be ARMed for proper functioning by writing to EP6BCL register with the skip bit (bit 7) ON because it is an OUT endpoint. Here, it is double-buffered so it is ARMed twice.

3.3.2 Loading HID/ Report Descriptor

The loading of the HID and report descriptors is handled in the DR_GetDescriptor() function using Code 7:

Code 7. Loading HID/ Report Descriptor

```
switch (SETUPDAT[3])
{
    case GD_HID: //HID Descriptor
    switch (SETUPDAT[4])
```

```
     {
        case GD_IF0:
            SUDPTRH = MSB(pHID1Dscr);
            SUDPTRL = LSB(pHID1Dscr);
        break;
        case GD_IF1:
            SUDPTRH = MSB(pHID2Dscr);
            SUDPTRL = LSB(pHID2Dscr);
        break;
        default:
            EZUSB_STALL_EP0();
    }
    return (FALSE);
    break;
    case GD_REPORT:  //Report Descriptor
    switch (SETUPDAT[4])
    {
        case GD_IF0:
            HID1length = pHID1ReportDscrEnd - pHID1ReportDscr;
            AUTOPTR1H = MSB(pHID1ReportDscr);
            AUTOPTR1L = LSB(pHID1ReportDscr);
            for(i=0;i<HID1length;i++)
                EP0BUF[i]=XAUTODAT1;
                EP0BCL = HID1length;
        break;
        case GD_IF1:
            HID2length = pHID2ReportDscrEnd - pHID2ReportDscr;
            AUTOPTR1H = MSB(pHID2ReportDscr);
            AUTOPTR1L = LSB(pHID2ReportDscr);
            for(i=0;i<HID2length;i++)
                EP0BUF[i]=XAUTODAT1;
                EP0BCL = HID2length;
        break;
        default:
            EZUSB_STALL_EP0();
     }
     return (FALSE);
     break;
     default:
     return(TRUE);
```

The first switch statement, *switch (SETUPDAT[3])*, determines the type of GET_DESCRIPTOR request made (that is, HID or report). The next switch statement, *switch (SETUPDAT[4])*, then determines the HID or report descriptor to load depending on the interface number, 00H or 01H (00H corresponding to the mouse interface, 01H corresponding to the keyboard interface) returned by the GET_DESCRIPTOR request. See the GET_DESCRIPTOR request description of the *HID Class Specification (version. 1.1)* for more details on how the switch-case statements are indexed.

3.3.3 Reading Button States

The read_buttons() function reads the state of the development board buttons (the buttons are mapped to an I^2C Philips I/O expander), PCF8574 using Code 8. See *EZ-USB(R) Development Kit User Guide.pdf* (available at C:\Cypress\USB\CY3684_EZ-USB_FX2LP_DVK\1.0\Documentation after DVK installation) for more details on I/O expanders.

Code 8. Reading Button States

```
BYTE read_buttons (void)
{
    BYTE d;
    while (I2CS & 0x40);     //Wait for stop to be done
    I2CS = 0x80;             //Set start condition
    I2DAT = BTN_ADDR;                //Write button address
    while (!(I2CS & 0x01));//Wait for done
    I2CS = 0x20;             //Set last read
    d = I2DAT;               //Dummy read
    while (!(I2CS & 0x01));//Wait for done
    I2CS = 0x40;             //Set stop bit
    return(I2DAT);           //Read the data
}
```

Here, 0x80 is written to I2CS register to set start of the I^2C transaction. Next, BTN_ADDR (0x41) is written to I2DAT register to send the I^2C address of the Philips I/O expander with the read bit set to signal an I^2C read. Because this is just a single -byte read, 0x20 is written to I2CS to indicate this. Next, I2DAT is read to trigger a read on the I^2C bus; the data returned now is not the actual data that was read from the Philips I/O expander. To read the actual data, 0x40 is written to I2CS to signal a stop condition and then I2DAT is read. Here, I2DAT is read during the generation of the 'stop' condition to prevent generation of another read on the bus due to the read to I2DAT.

Based on the button pressed, the input report is sent to the host to signify a mouse click or a keystroke. Table 3 summarizes the mapping of the development board buttons with the mouse/keyboard buttons.

Table 3. Function Mapping of Development Board Buttons

Development Board Button	Function
f1	Left-click
f2	Right-click
f3	Send 'b'
f4	Send 'c'

3.3.4 TD_Poll()

The code that actually sends the input reports of the mouse/keyboard function and the loopback over bulk endpoint resides in TD_Poll(). The mouse interface uses Endpoint 2 IN to send either a 'left-click' or a 'right-click' event to the host, whereas the keyboard interface uses Endpoint 1 IN to send the characters 'b' or 'c' to the host (viewable in a text editor like Notepad or Microsoft Word) and the loopback over bulk endpoint interface transfers packets from Endpoint 6 to Endpoint 8.

The portion of the code that checks whether EP2 is available and fills it with the corresponding data is the mouse portion of the code. If Endpoint 2 (IN) is as in Code 9 below, the status of the buttons is read using the read_buttons() function. In between button change states, a 0x00 is sent to the Host to indicate that no event has occurred. This clears the button state as seen by the Host. If a change in button state is detected, the button pressed on the development board is identified. Then, according to the report format summarized in Table 1, the corresponding byte information representing a left-click or a right-click event is sent to the Host. Note that it is important to send the X and Y displacement information even though it is not being used in this example. This is because the Host expects these fields to appear in the report because they were defined in the report descriptor.

Code 9. Mouse Interface

```
//mouse interface
if(!(EP2468STAT & bmEP2FULL)) //Is the EP2 available,
{
    HID1buttons = read_buttons();
    if (HID1buttons == read_buttons())   // Debounce
    {
        HID1buttons &= 0x0F;
        EP2FIFOBUF[0] = 0x00; //clear button state as
                              // seen by the host
        if ((HID1oldbuttons-HID1buttons)!= 0)
        //Change in button state
        {
            if ( !(HID1buttons & 1) )   //left click
            {
                EP2FIFOBUF[0] |= 0x01;
            }
            if ( !(HID1buttons & 2) )   //right click
            {
                EP2FIFOBUF[0] |= 0x02;
            }
            EP2FIFOBUF[1] = 0x00;
            EP2FIFOBUF[2] = 0x00;
            EP2BCH = 0;
            SYNCDELAY;
            EP2BCL = 3;
        }
        HID1oldbuttons = HID1buttons;
    }
}
```

The keyboard portion of Code 10 checks whether EP1 is available and fills it with the corresponding data. The status of the buttons is read if Endpoint 1 IN is available. If a change in button state is detected, the button pressed on the development board is identified and the corresponding keycode for ‘b’ or ‘c’, 05H or 06H respectively, is sent. If a button is not pressed, a value of 0x00 is sent instead to the host to indicate that no event has occurred. The keycodes are obtained from the *HID Usage Tables* document available for download from http://www.usb.org/developers/hidpage/.

Code 10. Keyboard Interface

```
//keyboard interface
if(!(EP1INCS & bmEPBUSY)) // Is the EP1IN  available,
{
    HID2buttons = read_buttons();
    if (HID2buttons == read_buttons())   //Debounce
    {
        HID2buttons &= 0x0F;
        if ((HID2oldbuttons-HID2buttons)!= 0
        //Change in button state
        {
            if (HID2buttons & 4)    // b
                EP1INBUF[3] = 0x00;
            else
                EP1INBUF[3] = 0x05;
            if (HID2buttons & 8)    // c
                EP1INBUF[4] = 0x00;
            else
                EP1INBUF[4] = 0x06;
            EP1INBUF[0] = 0x00;
            EP1INBUF[1] = 0x00;
            EP1INBUF[2] = 0x00;
            EP1INBC = 5;
        }
        HID2oldbuttons = HID2buttons;
     }
        }
```

The ‘loopback over bulk interface’ portion of Code 11 checks whether EP6 has packets in it and transfers them to EP8, if EP8 has free buffers,. It uses Autopointer (Auto-incrementing pointer) for the transfer of data between the endpoints.

Code 11. Loopback Over Bulk Endpoint Interface

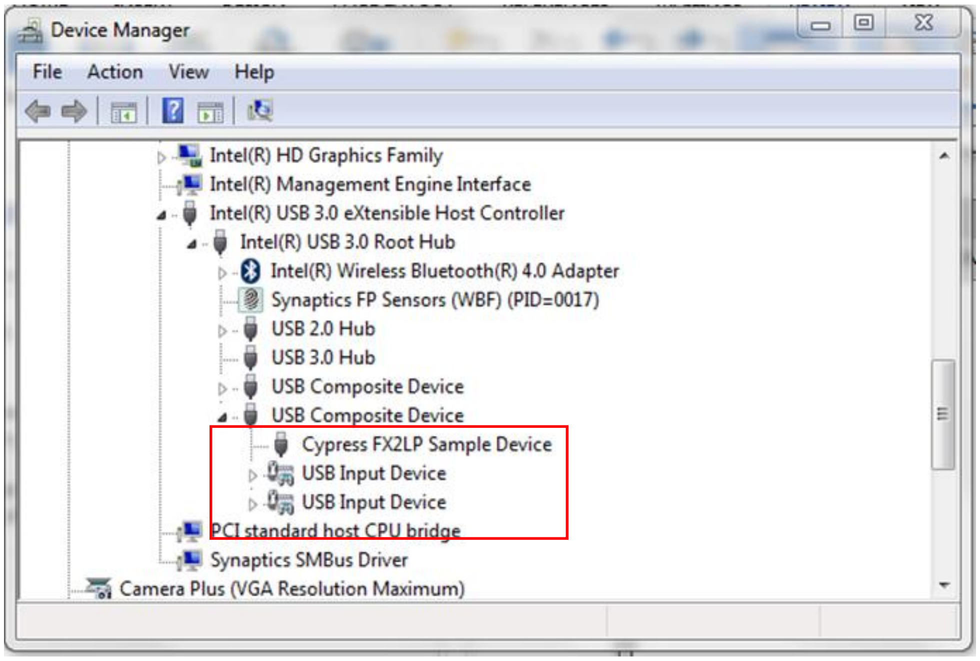
```
//loopback over bulk endpoint interface

if(!(EP2468STAT & bmEP6EMPTY))

{ // check EP6 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is
empty

       if(!(EP2468STAT & bmEP8FULL))

       { // check EP8 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when
       FIFO is

         //full

              APTR1H = MSB( &EP6FIFOBUF );

              APTR1L = LSB( &EP6FIFOBUF );

              AUTOPTRH2 = MSB( &EP8FIFOBUF );

              AUTOPTRL2 = LSB( &EP8FIFOBUF );

              count = (EP6BCH << 8) + EP6BCL;


              // loop EP6OUT buffer data to EP8IN

              for(i = 0x0000;i < count; i++)

              {

                     // setup to transfer EP6OUT buffer to EP8IN buffer using
              AUTOPOINTER(s)

                     EXTAUTODAT2=EXTAUTODAT1;

              }

              EP8BCH = EP6BCH;

              SYNCDELAY;

              EP8BCL = EP6BCL;   // arm EP8IN

              SYNCDELAY;

              EP6BCL = 0x80; //re(arm) EP6OUT

       }

}
```

4 How to Test the Application

1. Install FX3 SDK. This installs the Control Centre utility.
2. In CY3684 FX2LP DVK, select the “EEPROM ENABLE” switch to the ‘No EEPROM’ position. Connect the (FX2LP DVK) board to a PC. The DVK enumerates with the default internal descriptor. Use the *CyUSB.inf* file to bind to the *CyUSB.sys* driver available with FX3 SDK. For details on inf file modification, see AN61465 - Working With inf File of a Device Using *CyUSB.sys*.

3. Open the Control Center application (Start > All programs > Cypress > EZ-USB FX3 SDK > Cypress USBSuite > Control Center). Download the *ezcombo.hex* file from the Firmware folder (of the attachment along with this application note) to the FX2LP RAM using the Control Centre utility (Program FX2 > RAM). The FX2LP device then re-enumerates running the application firmware.
4. If Windows pops up asking you to bind the driver, repeat the steps for binding the driver. Note that the device will enumerate as a composite driver. Therefore, it must be bound to composite driver for the HID devices to come up. The vendor interface must be bound to *CyUSB.sys* as explained in Step 2.
5. Once the driver is bound successfully for all three interfaces, there must be three instances in the device manager as shown in Figure 1. The figure shows the view using the "Devices by Connection" option in the device manager.

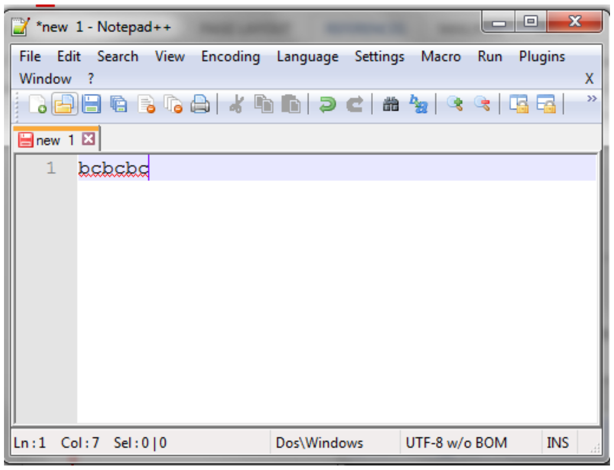
Figure 1. FX2LP Enumerated as Composite HID Device



Intel(R) HD Graphics Family
Intel(R) Management Engine Interface
Intel(R) USB 3.0 eXtensible Host Controller
Intel(R) USB 3.0 Root Hub
Intel(R) Wireless Bluetooth(R) 4.0 Adapter
Synaptics FP Sensors (WBF) (PID=0017)
USB 2.0 Hub
USB 3.0 Hub
USB Composite Device
USB Composite Device
Cypress FX2LP Sample Device
USB Input Device
USB Input Device
PCI standard host CPU bridge
Synaptics SMBus Driver
Camera Plus (VGA Resolution Maximum)

6. Open any text editor. Buttons 'f3' and 'f4' functions as the keyboard keys 'b' and 'c', respectively. Each clicking of the buttons 'f3' and 'f4' of FX2LP DVK causes the letters 'b' and 'c' to be printed, respectively as in Figure 2.

Figure 2. Upon Clicking Button 'f3', 'f4' of CY3684 Alternatively

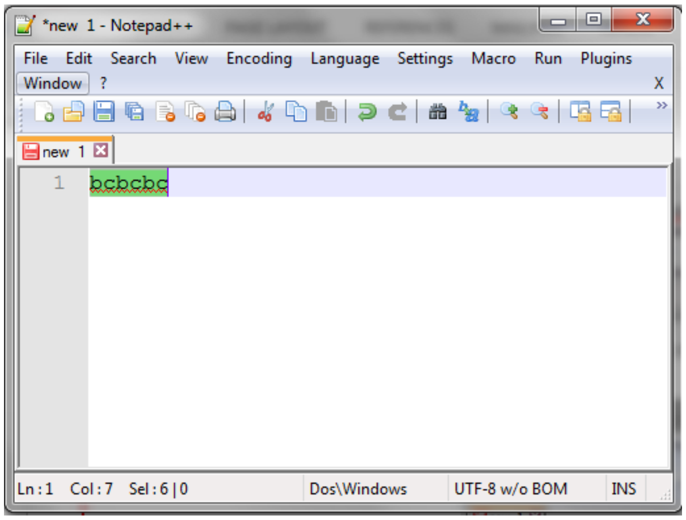

*new 1 - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins
Window ?
new 1

```
bcbcbc
```

Ln : 1 Col : 7 Sel : 0 | 0
Dos\Windows
UTF-8 w/o BOM
INS

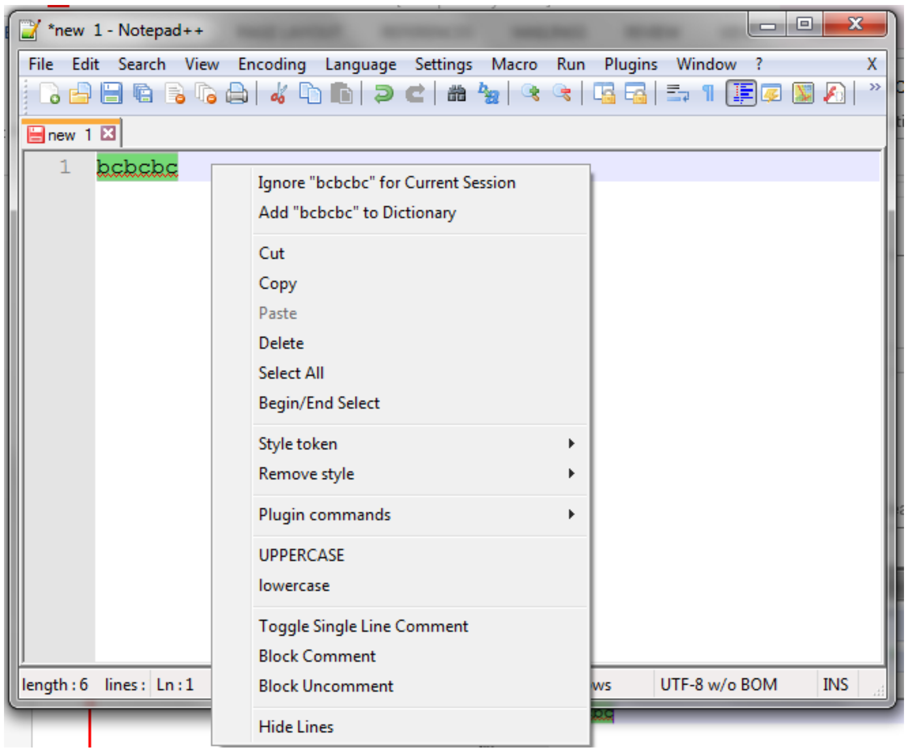
7. Button ‘f1’ of FX2LP DVK functions as the left-click of a mouse. Thus, double-clicking of the button ‘f1’ causes the text formed in Step 6 to be selected as in Figure 3.

Figure 3. Upon Double-Clicking button 'f1’ of CY3684


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
new 1
bcbcbc
Ln : 1 Col : 7 Sel : 6 | 0
Dos\Windows
UTF-8 w/o BOM
INS

8. Button ‘f2’ of FX2LP DVK functions as the right click button of a mouse. Thus clicking of the button ‘f2’ causes the context menu to open up as in Figure 4.

Figure 4. Upon Clicking Button 'f2’ of CY3684


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
new 1
bcbcbc
Ignore "bcbcbc" for Current Session
Add "bcbcbc" to Dictionary
Cut
Copy
Paste
Delete
Select All
Begin/End Select
Style token
Remove style
Plugin commands
UPPERCASE
lowercase
Toggle Single Line Comment
Block Comment
Block Uncomment
Hide Lines
length : 6 lines : Ln : 1
UTF-8 w/o BOM
INS

5 Summary

This application note shows how to implement a sample composite HID device using CY3684 EZ-USB FX2LP Development Kit. Not all developers choose EZ-USB as a peripheral controller for their HID device. However, since Cypress has developed very low-cost solutions for mice and keyboards based on the M8 core, this application note aims to jumpstart the development process if EZ-USB is the chip of choice.

6 References

1. “Device Class Definition for Human Interface Devices (HID) Version 1.1.” USB Implementers’ Forum, 07 April 1999.
2. “HID Usage Tables Version 1.1.” USB Implementers’ Forum, 04 Apr. 1999.
3. Axelson, Jan. USB Complete. Madison: Lakeview Research, 1999.
4. Hyde, John. USB Design by Example 2nd Ed. Intel Press, 2001.

About the Author

Name:	Anand Srinivasan Asokan
Title:	Applications Engineer Sr

Document History

Document Title: AN64020 - Creating a FX1/FX2LP™ Composite HID Device

Document Number: 001-64020

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3032410	AASI	09/16/2010	New application note.
*A	3124039	AASI	01/06/2011	Removed references to obsolete parts. Added a loopback over bulk interface to emphasize flexibility and bandwidth capability. Added driver files for Windows XP 32 and 64 bit environment for the loopback over bulk interface.
*B	4112388	GAYA	09/03/2013	Updated in new template. Completing Sunset Review.
*C	5177295	GAYA	06/02/2016	Removed driver files from project attachment; referencing to the drivers in FX3 SDK. Updated the project: fixed WORD ALIGNMENT of string descriptor, and descriptor handling for HID descriptor. Added How to Test section. Updated template
*D	5705455	AESATMP9	04/21/2017	Updated logo and copyright.
*E	5754794	GAYA	06/05/2017	Updated template

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709