AN58726

USB HID Intermediate with PSoC® 3 and PSoC 5LP

Author: Robert Murphy

Associated Project: Yes

Associated Part Family: All PSoC® 3 and PSoC 5LP parts

Software Version: PSoC Creator™ 3.3 SP1 and higher

Related Application Notes: See Related Resources

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/an58726

AN58726 is a continuation of basic-level AN57473. It describes additional features of the USB Human Interface Device (HID) protocol, including input and output transactions and composite devices, using PSoC® 3 and PSoC 5LP and the PSoC Creator™ USBFS Component. A variety of HID devices, including a keyboard with LEDs and a composite device, are used as examples. This application note is a prerequisite for the advanced-level AN56377 and AN82072.

Contents

1 Introduction

USB is a complex protocol, and it can be difficult for beginners to get a USB-based application up and running quickly. However, some aspects of USB are easy to use, especially the Human Interface Device (HID) protocol. HID is designed for common PC interface devices such as a keyboard or mouse, but it can be adapted for many custom applications. Most PC operating systems, including Windows, Mac, and Linux, include HID drivers. This means that you do not have to write a driver; instead, you can focus on developing your application firmware.

AN58726 is a continuation of AN57473. It shows you how to do more complex data transfers with PSoC 3 and PSoC 5LP using USB HID. The PSoC devices include a dedicated Full-Speed (FS) 12-Mbps USB 2.0 peripheral, which uses an internal oscillator—a crystal is not required. The PSoC Creator IDE and the USBFS Component make it possible to build an application quickly. Several devices, including a keyboard with LEDs and a composite device, are used as examples.

This application note assumes that you are familiar with developing applications using PSoC Creator for PSoC 3 or PSoC 5LP. If you are new to these products, refer to AN54181 – Getting Started with PSoC 3 and AN77759 – Getting Started with PSoC 5LP. If you are new to PSoC Creator, see the PSoC Creator home page.

If you are new to USB, AN57294 explains USB basic concepts, and AN57473 presents USB HID basics. If you are familiar with USB basics, you can find more advanced information in the Related Resources: Application Notes.

2 Standard HID Keyboard Report

To learn about HID input and output items, let us look at a standard HID keyboard report. Before diving into the details of the report descriptor, we will outline what information the keyboard sends to the host and what information the host sends to the keyboard.

2.1 Data Sent to the Host

A transfer that carries data to the host from the device is known as an input transfer. Table 1 lists the input data that the keyboard sends to the host.

Table 1. Keyboard Input Report Table

Byte	Information
0	Modifier keys
1	Reserved (for OEM use)
2	Key code 1
3	Key code 2
4	Key code 3
5	Key code 4
6	Key code 5
7	Key code 6

The eight bytes that are sent to the host are organized as modifier keys followed by general keys. A modifier key is a key that modifies the function of a general key. Examples include Ctrl, Alt, and Shift, as Table 2 shows. The GUI keys are the Windows, Apple, or Meta keys on a keyboard, which are reported in the same byte as the modifier keys. The name or icon printed on the keyboard depends on the operating system for which the keyboard is designed. All other keys are considered to be general keys.

While the information is sent in an 8-byte packet, there are only seven bytes of useful information. This is because of the reserved byte (byte 1) shown in Table 1. It is intended for OEM use and is not used in most applications. The reserved key may be used on a keyboard that contains a nonstandard key that performs a function that is specific to that PC. This is occasionally seen in laptops and tablet computers. In most keyboards purchased in the consumer market, the reserved byte remains a constant value of 0x00.

The last six bytes in the configuration are the general keys. As many as six key codes can be sent to the PC in a given transaction. This enables as many as six simultaneous key presses. The order of the key codes in the array is not significant.

In Table 2, note that each modifier key has a corresponding bit associated with it. This means that the modifier keys are stored in a bit field of information. The HID usage tables, located in the USB HID Usage Tables document, show the usage values for the modifier keys ranging from E0–E7. However, the usage values are not sent as array data. The modifier keys are sent as variable data, which means that each individual bit in the 8-bit value corresponds to one of the modifier keys. The usage minimums/maximums are then used to link the modifier key information in the bit field to the proper usage value with the usage minimum/maximum.

Table 2. Modifier Key Index

Bit	Key	Modifier Value
0	Left Ctrl	0000 0001
1	Left Shift	0000 0010
2	Left Alt	0000 0100
3	Left GUI (Win/Apple/Meta)	0000 1000
4	Right Ctrl	0001 0000
5	Right Shift	0010 0000
6	Right Alt	0100 0000

Bit	Key	Modifier Value
7	Right GUI (Win/Apple/Meta)	1000 0000

To clarify the difference between “array” and “variable” information, refer to the HID basic application note, AN57473, which states the following:

“Array versus Variable: *Array means only controls that are currently active are reported such as a button being pressed. Variable means that the data reported is the current state of every control regardless if a button is pressed or not.”*

The difference between “array” and “variable” becomes more relevant in this keyboard application. While a variable configuration is good for returning a single bit that represents the status of a single button or LED, an array is useful in returning an index that corresponds to a button being pressed. In the case of a keyboard, an array configuration is used to report the keyboard’s general key scan codes. If an out-of-range value is reported, the host views it as no buttons being asserted. Because each button press is not being represented by an individual bit, multiple fields are required when reporting multiple simultaneous button presses. The logical minimum/maximum and usage minimum/maximum are then used to determine the corresponding key code.

For example, to type a capital *A*, the user would press the right Shift key and the “A” key. The right Shift key is sent by setting the sixth bit in the modifier key byte, and the “A” key is sent by loading the first available array byte with the “A” key code, which is 0x04. You can view the individual key codes in the USB HID Device Class Specification. The input report for a capital *A* would look like Table 3.

Table 3. Capital *A* Input Report

Byte	Bit Field Value	Information
0	0010 0000	(Right Shift)
1	Reserved	(For OEM Use)
2	0000 0100	(“A”)
3	0000 0000	NA
4	0000 0000	NA
5	0000 0000	NA
6	0000 0000	NA
7	0000 0000	

2.2 Data Sent to the Keyboard

HID devices often have indicators that give the user status information about the host. With a keyboard, there may be LED indicators for the Num Lock, Caps Lock, or Function Lock keys.

The same variable bit field organization used for the modifier keys is also applied to the LEDs. According to the USB HID Device Class Specification, the usage values for the LEDs are as provided in Table 4.

LED information is an “absolute” Item, which means that the output report must include the state of each LED with ‘0’ meaning off and ‘1’ meaning on. If the LEDs were relative items, then a ‘0’ would represent no change and ‘1’ would represent a change in state.

When a user presses any of the corresponding buttons such as Caps Lock or Num Lock, the host manages the updating of the LED information. Additionally, LED states are set by sending a report to the keyboard device via a SET_REPORT(Output). Rather than output data being transferred through a dedicated OUT endpoint, SET_REPORTs are done through Endpoint 0, the control endpoint, and do not require a dedicated endpoint to be provided.

Table 4. LED Indicator Index

Bit	LED
0	Num Lock
1	Caps Locks
2	Scroll Lock
3	Compose
4	Kana
5 to 7	Constant

Because a bit field is used to represent each of the possible LED combinations, the output report consists of a single byte, as shown in Table 5.

Table 5. Keyboard Output Report

Byte	Bit Field	Information
0	000n nnnn	LED Indicators

Now that you know how the keyboard interacts with the host, the next section takes a look at how to create the report descriptor, which makes this communication possible.

3 Keyboard Report Descriptor

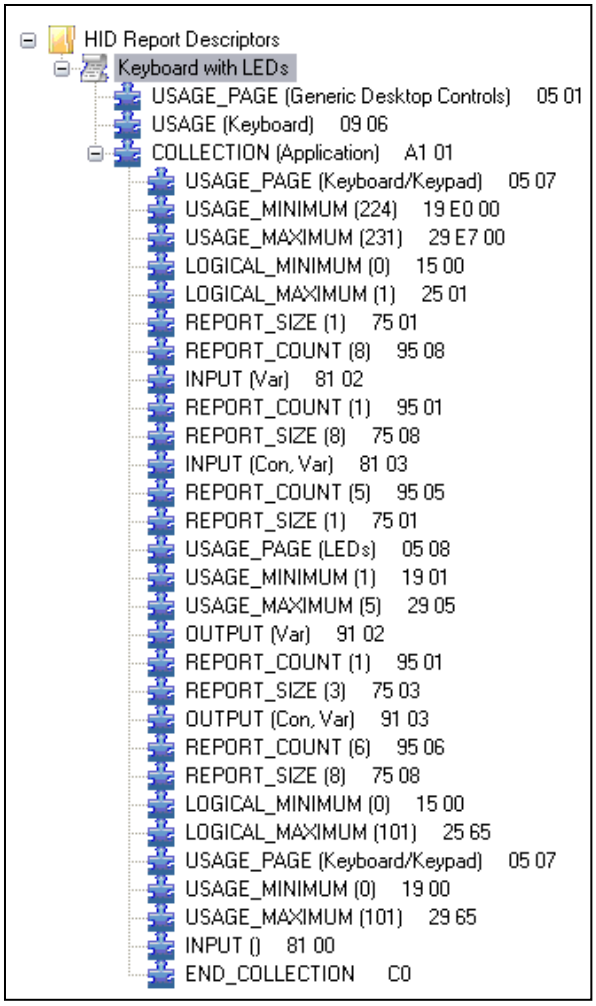
HIDs let you easily interface with and control a host computer. Almost all HIDs send information to the host, for example, a keyboard and mouse. However, the PC often needs to send information in the other direction—to the device, for example—to control an LED on a keyboard. This is done using an output report.

The result of this transfer can take the form of an LED on a keyboard or the force feedback on a joystick. Both of these devices require interaction from the host to provide them with information. Consider the example of a keyboard. A keyboard has input items for the keys that the user presses and an output item for the LEDs on the keyboard that display information (such as checking if the Caps Lock is enabled or not).

To receive an output report properly, output Items in the HID report descriptor must be configured properly. The configuration of output items is similar to the setup of input items discussed in AN57473.

To understand output items in greater detail, look at a report descriptor for a 104-key keyboard, as shown in Figure 1, where the organization of the report descriptor follows the required format for a Boot Interface. The report descriptor can be organized in various ways, however, and still function as a keyboard with LEDs.

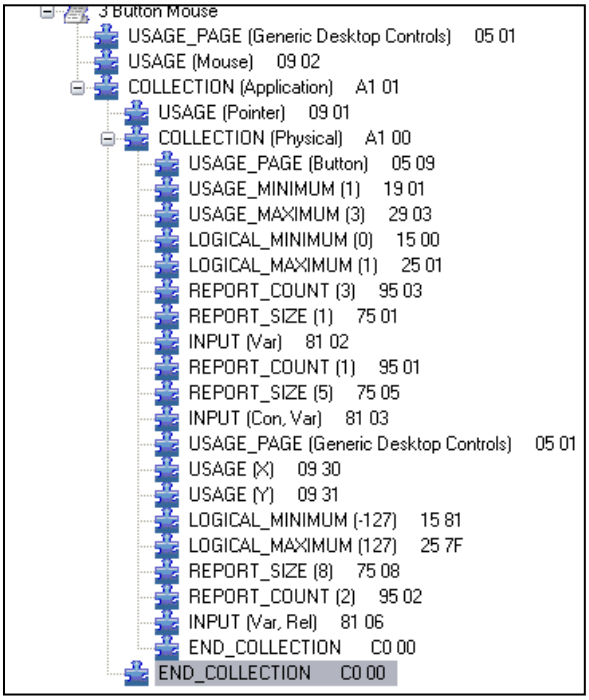
Figure 1. Keyboard Report Descriptor



4 Output Reports and Items

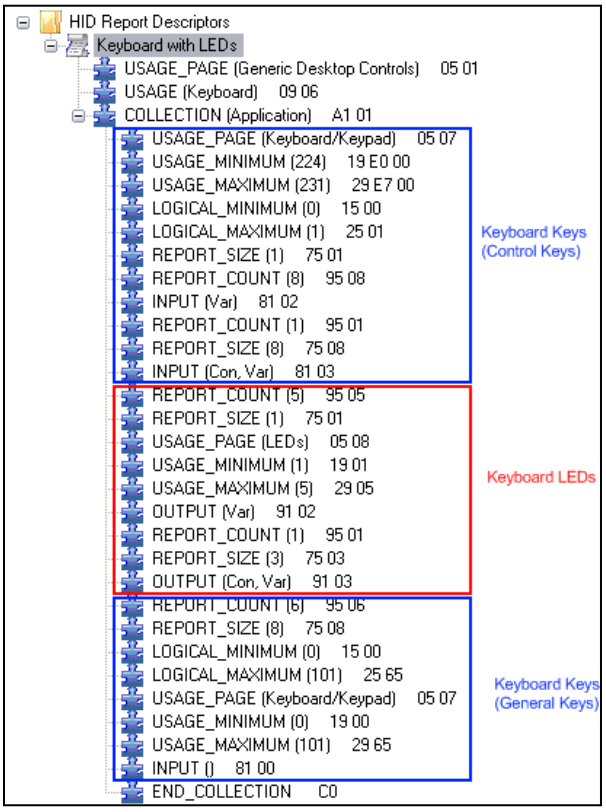
Many of the Items in the report descriptor are similar to those described in AN57473, where a report descriptor for a mouse is used as an example. The same descriptor is shown in Figure 2 for comparison. The only major difference is the addition of output items.

Figure 2. Mouse Report Descriptor from AN57473



When the descriptor is broken apart, each item's function becomes clearer (see Figure 3). The report descriptor has created a sandwich with the keyboard control keys on top, LEDs in the middle, and keyboard general keys at the bottom (input-output-input). While this organization may look odd, it follows the format required to implement a Boot Interface.

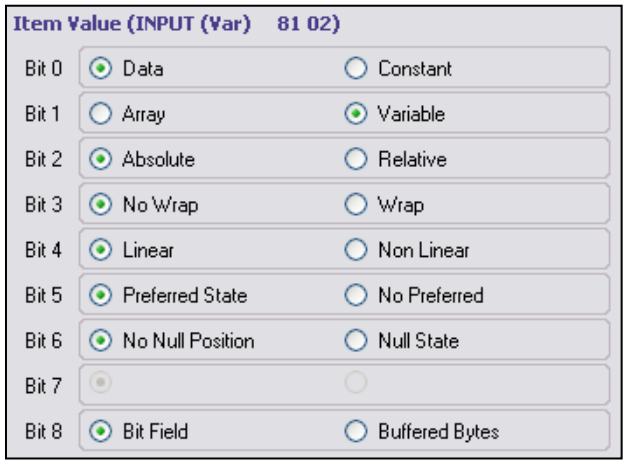
Figure 3. Sectioned Keyboard Report Descriptor



Look closely at the report descriptor and note that eight bytes are sent to the host as an input item and one byte is received from the host as an output item.

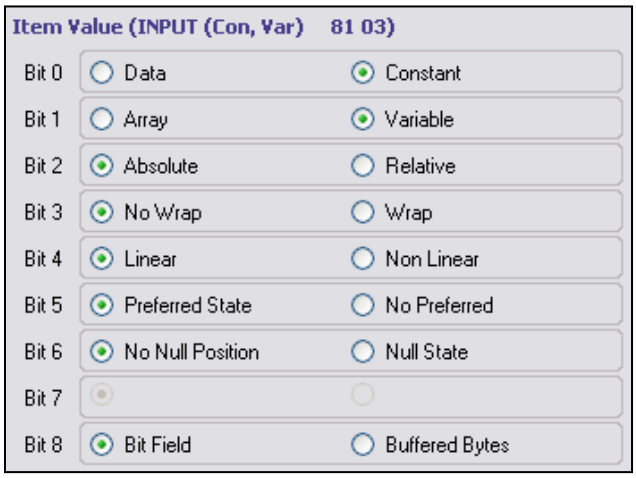
The first input item when referencing the report descriptor from the top down is for the modifier keys (Shift, Alt, and GUI). As stated in the Data Sent to the Host section, these keys are configured in a variable configuration to implement a bit field. Figure 4 shows the input item configuration. The usage IDs for the control keys, set by the usage minimum and maximum, are defined in the USB HID specification.

Figure 4. Input Item for Keyboard Modifier Keys



To reserve the second byte in the data structure, the entire byte is padded with zeros and configured as a constant. To declare this byte as a constant, bit 0 is set to a value of '1' (see Figure 5). This is done because each report is byte aligned.

Figure 5. Input Item for Reserving Second Byte



The next step in configuring the descriptor is to configure the output item for the LEDs on the keyboard, as Figure 6 shows. Note the following:

- The prefix value is 91, signifying an output item with one byte to follow.
- Bit 7 is set to '0', signifying that the bit is nonvolatile, which means the device does not change the value, and only the host can change the value.
- Bit 1 is set to '1', or "Variable," which configures the item as a bit field.

Figure 6. Output Item for LEDs

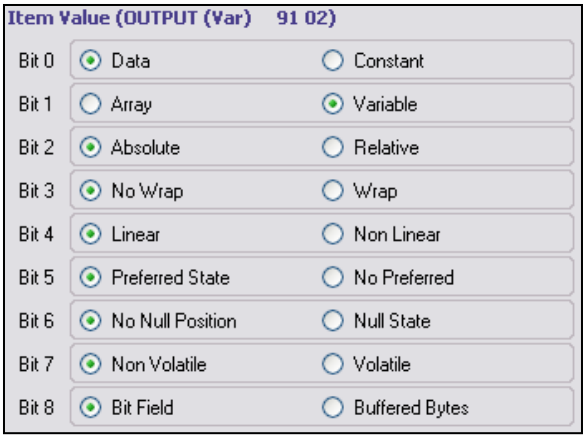
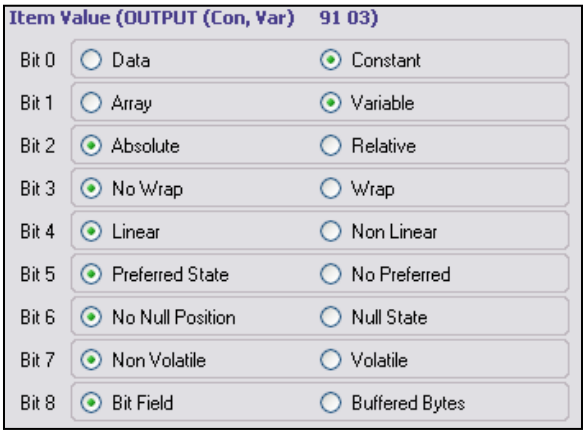


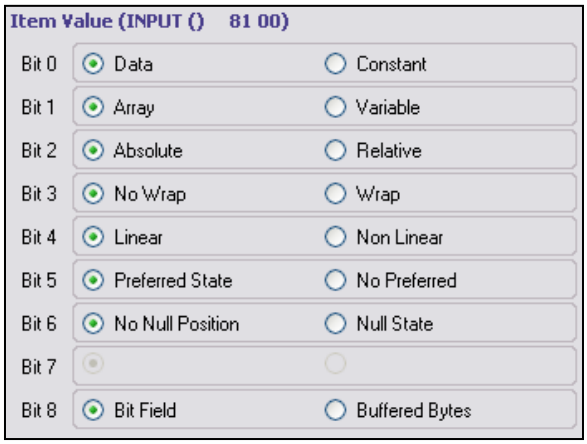
Figure 6 shows that only five of the total eight bits are used for LED information. Because all information is byte aligned, the remaining three bits are reserved by padding them with a value of '0' by setting those bits to remain constant. This is done by setting bit 0 in the output item to a value of '1', as Figure 7 shows.

Figure 7. Output Item for Padding LEDs



The final input item to configure is for the general (nonmodifier) keys. The HID specification requires these keys to be configured with an "Array" and "Absolute" configuration, as Figure 8 shows.

Figure 8. Input Item for General Keyboard Button



After you understand the functionality of each block of the report descriptor and how the information is interpreted, the next step is to study the report descriptor in detail and learn how each Item works together to create a fully functional keyboard. Figure 9 gives a fully commented version of the keyboard report descriptor. You can also view the commented report in the example project associated with this application note.

Figure 9. Commented Keyboard Report Descriptor

Section 8.3 of the USB HID specification contains an example of how the 8-byte array is configured when a user enters the Ctrl-Alt-Delete button combination (see Table 6).

Table 6. Ctrl-Alt-Delete Example

Step	Modifier Byte	Array Byte 2
Left Alt Down	00000100	0x00
Right Ctrl Down	00010100	0x00
Delete Down	00010100	0x4C
Delete Up	00010100	0x00
Right Ctrl Up	00000100	0x00
Left Alt Up	00000000	0x00

5 Boot Interface

Occasionally, a PC user may want to enter a PC's BIOS at bootup. In the BIOS, a dialog box is displayed that is navigated using a keyboard or mouse, which is different from using the keyboard and mouse with the OS. The USB HID specification defines protocols, called "boot protocols," for using the keyboard and mouse with the BIOS. The device must meet these protocol specifications to be a boot device.

The boot protocol supports a maximum of eight bytes of information. The BIOS ignores anything more than eight bytes.

Note that the BIOS does not read the report descriptor because of the predefined standard; it expects the information in a certain format. Thus, an HID device such as a keyboard or mouse can have two interfaces.

One of the interfaces, the boot interface, does not necessarily require a hard-coded report descriptor. Instead, it expects the host to recognize the device as a boot keyboard or a boot mouse and automatically understands how to interpret the information.

The second interface, the USB-aware interface, is loaded on entering the operating system via a "Change Protocol" command issued automatically after the PC loads the HID class driver.

The advantage of this method is that devices that have limited RAM can conserve memory. The second interface may or may not have the same configuration as the first interface. The developer makes this choice.

In this application note, the report descriptor used for the keyboard is the boot device format, although the example project is not configured to be a boot device. Project 1, the mouse project in AN57473, also follows this specification. For more information about boot devices, refer to Appendix A in the USB HID specification.

The subclass (bInterfaceSubClass) officially defines an interface as a boot interface. If the subclass has a value of '0' assigned to it, then there is no subclass. If a value of '1' is assigned, then the subclass has a boot interface. Values 2–255 are reserved by the HID specification and are not currently used.

After the subclass is configured as a boot interface, the configuration type needs to be selected. As mentioned previously, the BIOS does not read the report descriptor. To learn which type of device is connected, the protocol (bInterfaceProtocol) needs to be configured. If the protocol value is '1', then it is configured as a keyboard. When the protocol is set to '2', then it is configured as a mouse. Values 3–255 are reserved by the HID specification and are not currently used.

Both bInterfaceSubClass and bInterfaceProtocol are located in the interface descriptor. Because the host does not read the report descriptor, the array index value must be the same as the usage value for each key. If a boot interface is not used, the array index value and the usage value do not need to be equal to each other.

A boot keyboard can have 84, 101, or 104 keys. Table 12, Keyboard/Keypad Page, in the USB HID Usage Tables document shows the subset of scan codes that are supported with a boot interface for various computer operating systems. If a boot interface is not required or desired, the usage maximum in Figure 1 on page 5 can be changed to 255 to support the entire key code set.

6 Remote Wakeup

One feature that is not exclusive to HID but is commonly used with HID devices is remote wakeup. If you have ever walked up to a computer that was in a hibernate or suspend state, pressed a button on a keyboard or mouse, and observed the PC return to an active state, then you have used remote wakeup.

Remote wakeup is an optional USB feature in which the device is allowed to wake the host from a suspend mode by simply issuing a USB state on the bus. Support for remote wakeup is defined in a device's configuration descriptor. If you wish to use remote wakeup, select "Enabled" in the USBFS Configuration Wizard under the "Configuration Descriptor."

To issue a remote wakeup command, the device must first enter a suspend mode. When a host wants to enter suspend, it stops all USB traffic on the bus, including start-of-frame (SOF) packets. The USB specification states that after 3 ms of inactivity on the bus, you should begin entering a low power state and not draw more than 2 mA. Since an SOF occurs every 1 ms, you can create a timer with a 1-ms interrupt, as Figure 10 and Figure 11 show.

Figure 10. Timer for USB Suspend

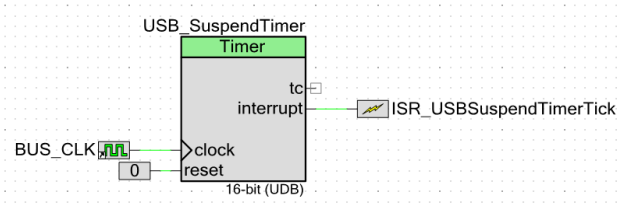
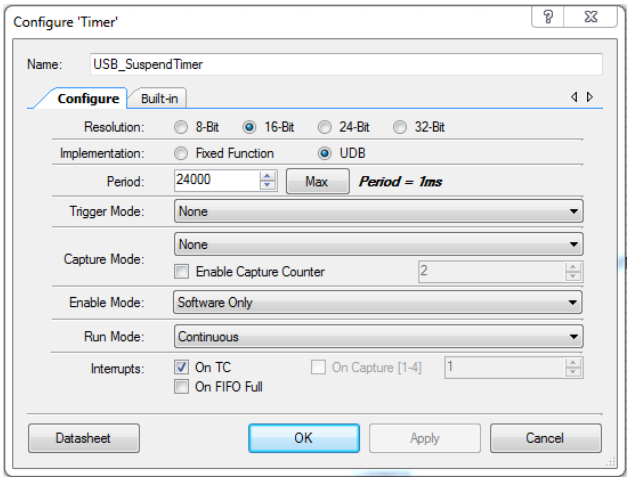


Figure 11. USB Suspend Timer Configuration



Every 1 ms, ISR_USBSuspendTimerTick triggers. At that point, you want to check if there has been any bus activity by using the PSoC Creator API function `USBFS_bCheckActivity()`. If the function returns a zero, meaning there was no bus activity, then you increment a counter. If it returns a non-zero value, indicating there was activity, then you reset the counter. When the counter becomes greater than 3, you put the USBFS Component and the PSoC device into sleep mode. On PSoC, make sure that the device is configured to wake from sleep on a PICU interrupt.

Once the device is in sleep mode, you can configure a digital input to be connected to a button with a PICU ISR. When the button is pressed, it wakes the PSoC device, which then checks to make sure that the host has enabled remote wakeup. If so, you can use Code 1 to wake the host from suspend.

Code 1. Remote Wakeup Code Example

```
/* Check if host enabled Remote Wakeup */
if(USBFS_RWUEnabled() != 0)
{
   /* Wait 20ms */
   CyDelay(20);
   /* Issue a K State */
   USBFS_Force(USBFS_FORCE_K);
   /* Wait 20ms */
   CyDelay(20);
   /* Remove K State from Bus */
   USBFS_Force(USBFS_FORCE_NONE);
   /* Wait 20ms */
   CyDelay(20);
}
```

7 Compound and Composite Devices

USB applications include two combined device types: compound devices and composite devices. Consider a keyboard that has an integrated trackpad in the enclosure. Only one USB cable is connected to the PC. For the two devices to communicate with the PC, either a compound or a composite device configuration is required.

With a compound device, each internal device has a separate USB controller connected to an internal hub within the device. The internal hub is connected to the PC.

A composite device has multiple logical interfaces with only one device address. Each interface in the composite device has its own device descriptors. A composite device also shares many of its resources with the various interfaces, for example Endpoint 0 (EP0). In the keyboard/trackpad example, the keyboard may use EP1 and EP2, and the trackpad may use EP3.

One important piece of information to be aware of when creating composite devices is the Interface Association Descriptor (IAD). The purpose of an IAD is to identify multiple interfaces that are associated with a particular function. Many devices define their function in the interface descriptor rather than the device descriptor. This is especially true of composite devices that contain different device classes such as an HID and Communication Class Device (CDC) or USB UART. When adding multiple different interfaces to a composite device, the IAD tells the host which interfaces are associated with which function. Figure 12 shows an example of an IAD for a CDC device.

Figure 12. IAD Descriptor Example

```
/*******************************************************************************
IAD Descriptor (CDC Communications Class Interface)
*******************************************************************************/
Descriptor size in bytes
/* bLength                                          */ 0x08u,
Constant IAD designator (0x0B)
/* bDescriptorType:IAD                              */ 0x0Bu,
Identifies the interface for the function
/* bFirstInterface                                  */ 0x00u,
Number of sequential interfaces associated with the function
/* bInterfaceCount                                  */ 0x02u,
Class Code (HID, CDC, etc).
/* bInterfaceClass : Communication i/f              */ 0x02u,
Subclass Code
/* bInterfaceSubClass : ACD                         */ 0x02u,
Protocol Code
/* bInterfaceProtocol : V.25ter                     */ 0x01u,
String descriptor index for the function
/* iInterface                                       */ 0x00u,
```

A CDC device consists of multiple interfaces, such as data interfaces and control interfaces. The IAD in Figure 12 shows how you can group a range of interface numbers together to define their function. Descriptor Tables for Composite Device (HID and CDC) gives a full example of an HID and CDC device descriptor. Notice that in the example, the bDeviceClass in the device descriptor contains a value of 0xEF. This signifies that the device contains an IAD.

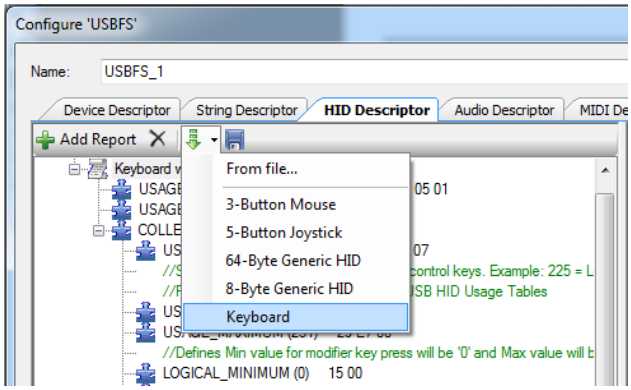
In addition, *.inf* file editing is required to link the multiple interfaces to the CDC function. However, that topic will be covered in another application note or example project. The report descriptor provided in Descriptor Tables for Composite Device (HID and CDC) is meant to explain the concept of an IAD in greater detail.

8 Project 1: Keyboard with LEDs

This project introduces an HID device that does both input and output transfers. The project emulates a PC keyboard—it types messages and displays the status of Num Lock, Caps Lock, and Scroll Lock. By following the steps in this section, you will gain an understanding of the project, Component setup, and code.

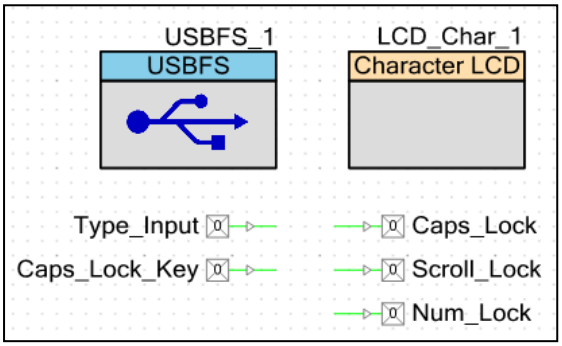
Note: This walkthrough is a step-by-step guide to creating an HID report from scratch. PSoC Creator also provides templates for common HID reports, which can be imported into the Component using the import drop-down menu in the **HID Descriptor** tab, as shown in Figure 13.

Figure 13. Importing an HID Report Template



1. Open PSoC Creator and create an empty project named “Project_1_Keyboard.” Then place the following Components into the schematic entry page (*TopDesign.cysch*), as Figure 14 shows.
   - Character LCD
   - USBFS
   - (2) Digital Input Pins
   - (3) Digital Output Pins

Figure 14. PSoC Creator Components for Keyboard

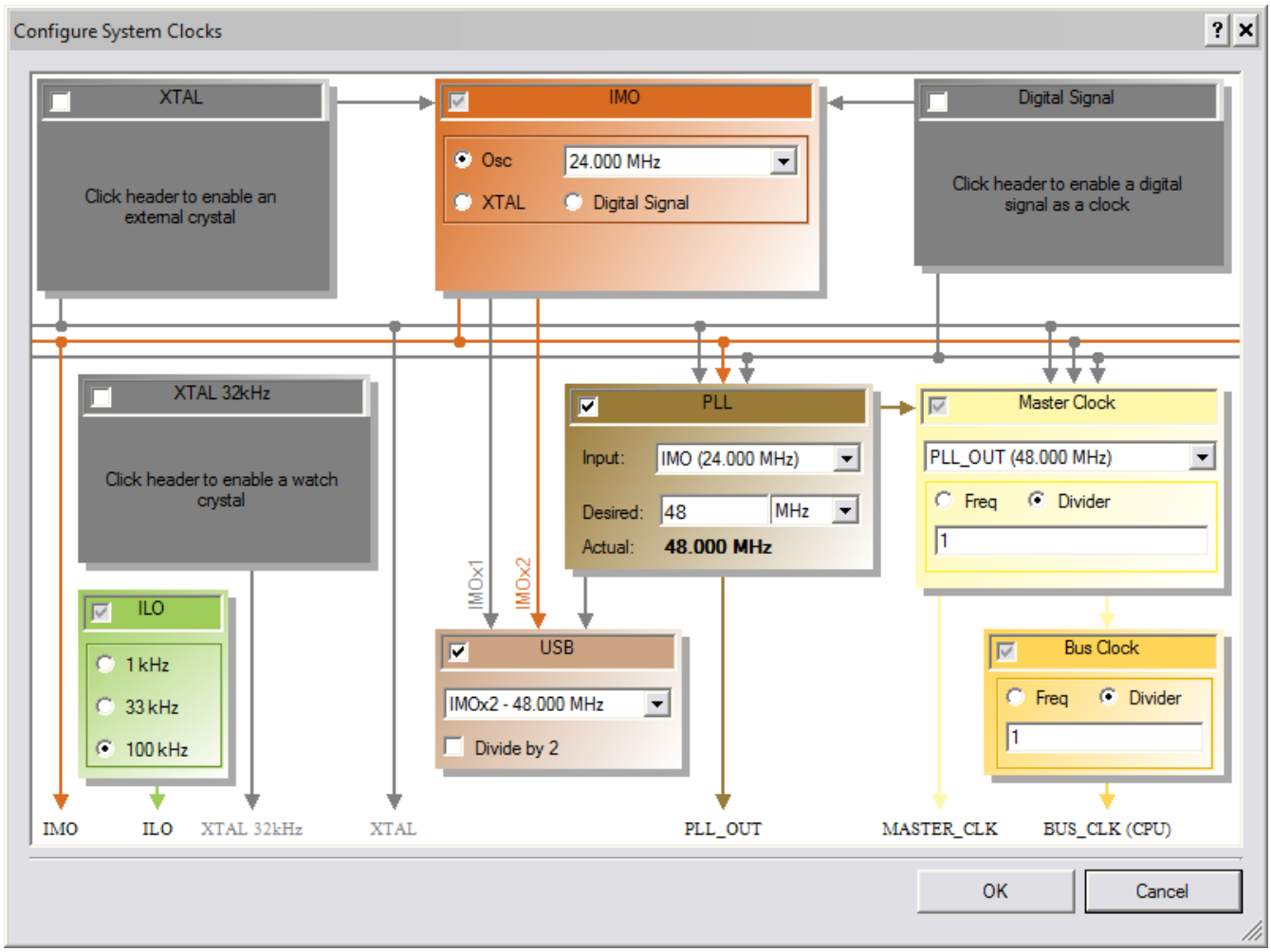


2. The Pin Components require a few configuration changes. For each Digital Input Pin, open the configuration dialog box, deselect the “HW Connection” option, and change the drive mode to “Resistive Pull Down.” For each Digital Output Pin, open the configuration dialog box, deselect the “HW Connection” option, and change the drive mode to “Strong.”

3. Go to the Workspace Explorer window and double-click on *MyFirstKeyboardHID.cydwr*. Click the **Clocks** tab and then double-click on one of the clocks to open the clock configuration window. Make the following changes to the clocks, as Figure 15 shows.

- IMO: 24 MHz
- ILO: 100 kHz
- PLL: Input: IMO, Desired: 33 MHz
- Master Clock: PLL Out
- USB Clock: IMO x2

Figure 15. System Clock Configuration Window



4. Click on the “Pin” tab in the *cydwr* file and assign the pins to resemble Figure 16. The USB pins are always located at P15[6] and P15[7]; the other pins can be moved if desired. The pinout in Figure 16 is designed to work with the CY8CKIT-001 PSoC Development Kit (DVK) board.

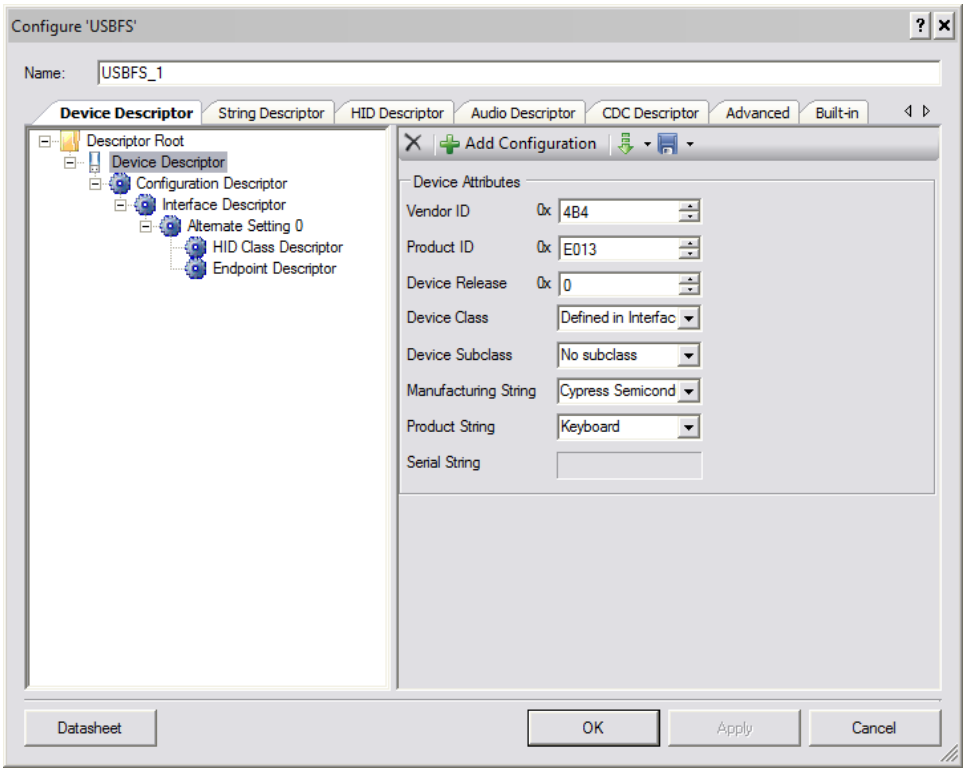
Figure 16. PSoC Creator Pin Configuration (Keyboard)

Alias	Name	Pin	Lock
	\USBFS_1:Dm\	P15[7]	☐
	\USBFS_1:Dp\	P15[6]	☐
	\LCD_Char_1:LCDPort\[6:0]	P2[6:0]	☑
	Caps_Lock	P1[6]	☑
	Scroll_Lock	P1[7]	☑
	Num_Lock	P1[5]	☑
	Type_Input	P0[7]	☑
	Caps_Lock_Key	P0[6]	☑

5. Configure the USBFS Component using the USBFS Configuration Wizard. This process follows the same steps as in AN57473 but with the addition of an OUT EP. Double-click on the USBFS Component to open the Configuration Wizard.

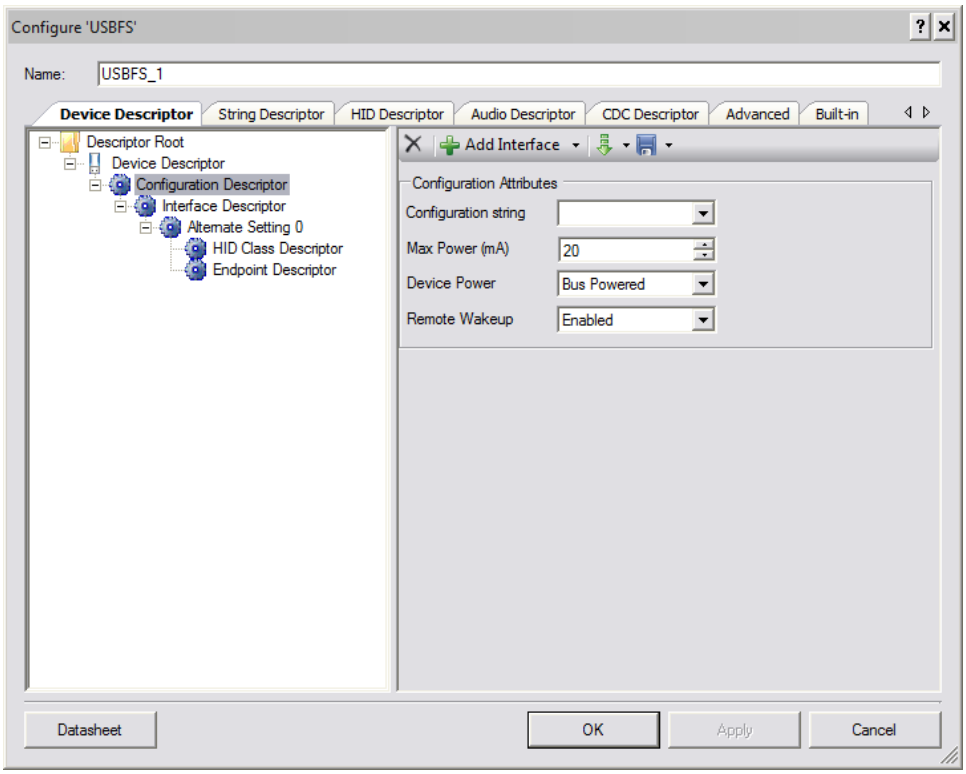
   Configure the “Vender ID” (VID) and “Product ID” (PID) to be 0x4B4 and 0xE013. Fill in the “Manufacturing String” and “Product String” too. You can use the strings shown in Figure 17 or choose others. You can also change the values for the VID and PID. Any value can be used for demonstration purposes, but if you are entering production or distributing the example project, your own VID must be assigned from the USB Implementers Forum.

Figure 17. USB Device Descriptor Setup



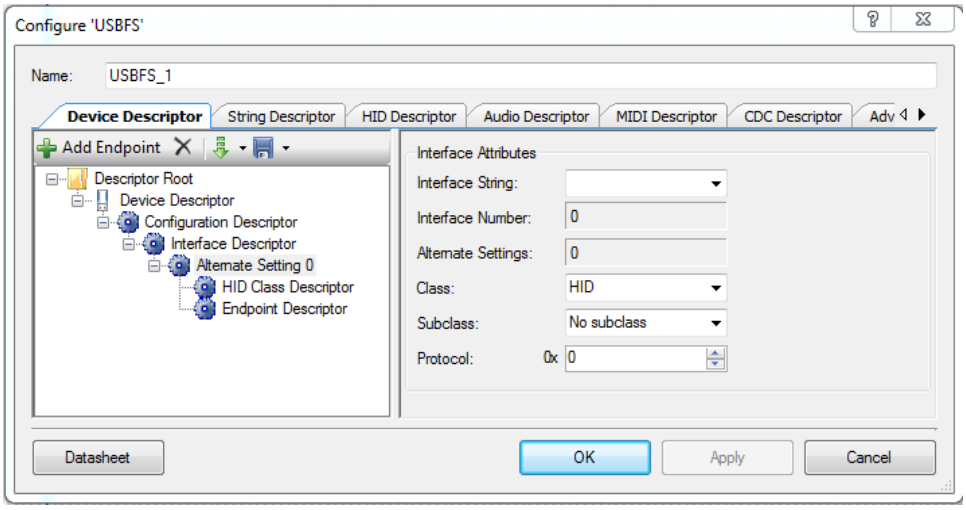
6. Select “Configuration Descriptor.” The window changes, as Figure 18 shows. Because the project is bus powered, limit the maximum current that can be supplied to the device. It is important to specify a value that is appropriate for the device. For this application, 20 mA is more than sufficient.

Figure 18. USB Configuration Descriptor Setup



7. Configure the interface descriptor by selecting “Alternate Setting 0.” In this menu, you only need to set the “Class” type to “HID.” This tells the host that the attached device is an HID, as Figure 19 shows. Note that “Subclass” is set to “No subclass.” If a boot interface is required, change the subclass to “Boot Interface.” This causes another drop-down menu to appear where you can choose between a mouse and a keyboard for the boot interface.

Figure 19. USB Interface Descriptor Setup



8. An HID report descriptor must be associated with the interface. To do so, an HID report descriptor must be created. Click the **HID Descriptor** tab in the dialog box. The window shown in Figure 20 opens.

   The format for the HID report descriptor is shown in Figure 1 on page 5. Make the required additions to the report descriptor so it resembles Figure 1 on page 5. A text version of the report descriptor is provided in Appendix B.

Upon completion of the report descriptor, return to the “HID Class Descriptor” menu shown in Figure 21. Set the “HID Report” to the name of the HID report that was just created.

Figure 20. USB HID Report Descriptor Setup

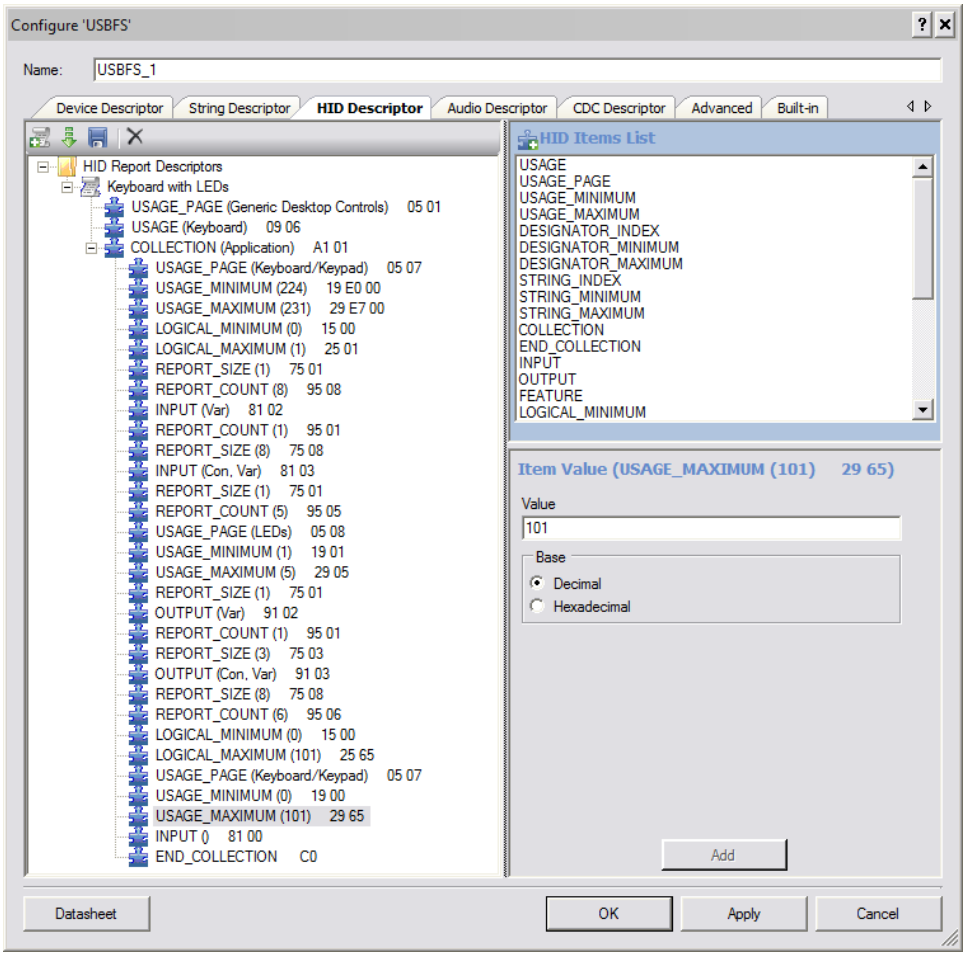
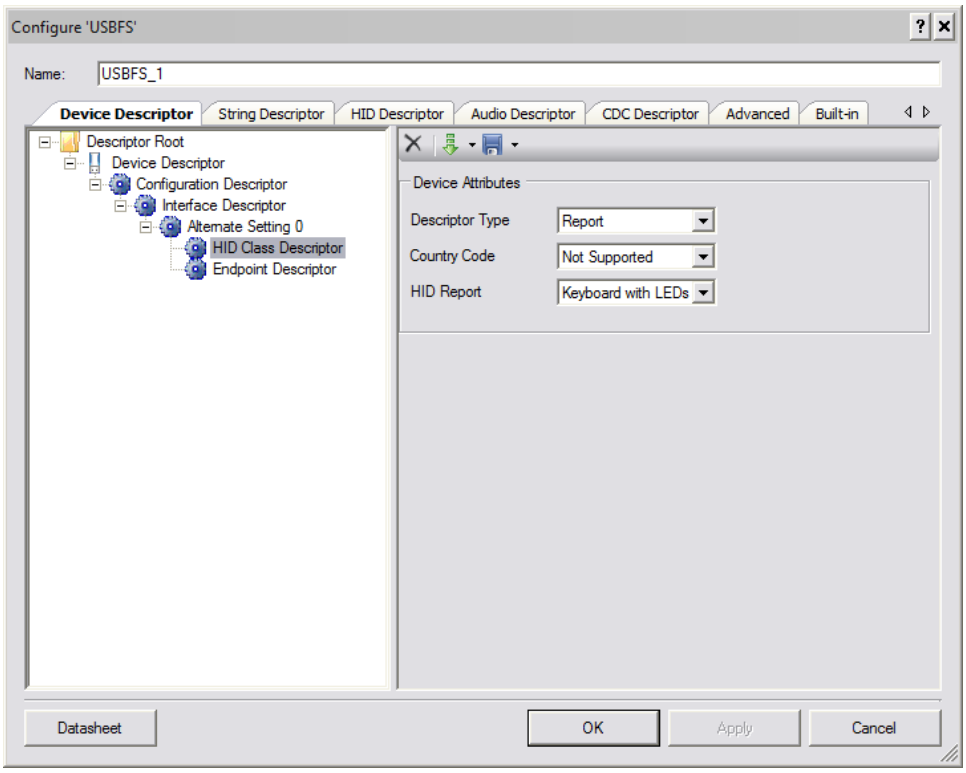


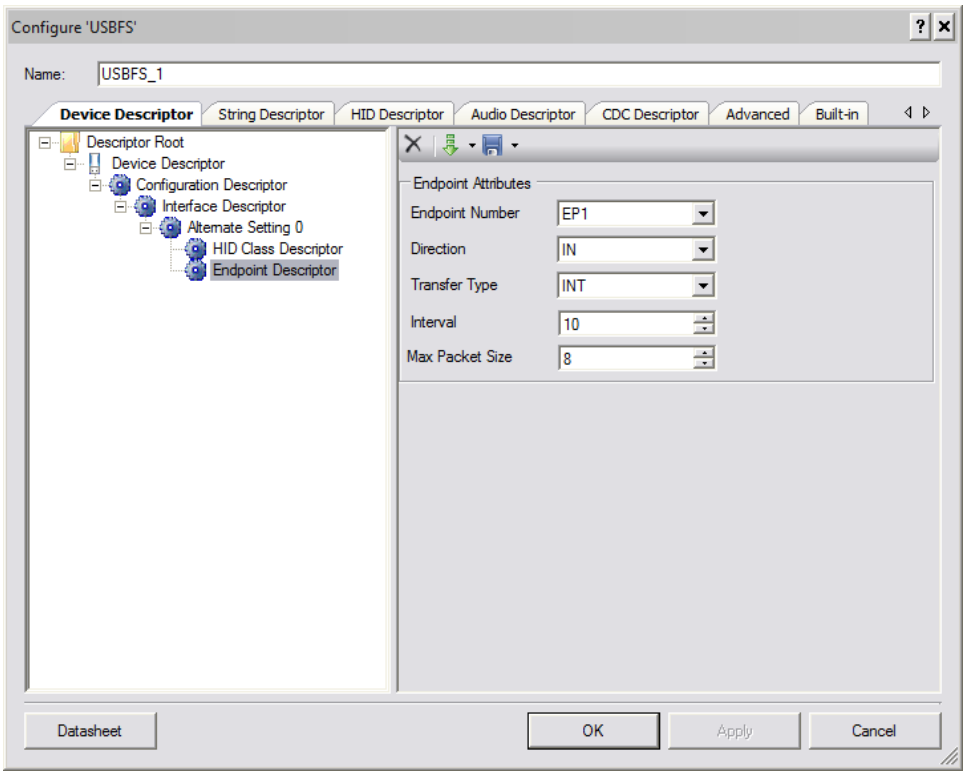
Figure 21. USB HID Class Descriptor Setup



9. The final step of configuration is to set the endpoints. As mentioned in Data Sent to the Keyboard, LED data is sent through EP0 via a SET_REPORT. As a result, only an IN endpoint is required. Select “Endpoint Descriptor.” Under “Endpoint Attributes,” make the following selections, as shown in Figure 22.

   - Endpoint Number: EP1
   - Direction: IN
   - Transfer Type: INT
   - Interval: 10
   - Max Packet Size: 8

Figure 22. USB Endpoint Setup (IN)



10. Locate the *main.c* file in the Workspace Explorer and open the file to edit it.

    The following code shows how easy it is to initialize USB communication:

```
/*Start USBFS Operation for Device 0
  with 5V operation*/
USBFS_1_Start(0,
   USBFS_1_DWR_VDDD_OPERATION);
/* Enables OUT EP */
USBFS_1_EnableOutEP(2);
/* Waits for USB to enumerate */
while(!USBFS_1_bGetConfiguration());
/* Begins USB Traffic. Prime the Pump */
USBFS_1_LoadInEP(1, Keyboard_Data, 8);
```

The following code does the OUT transfer that occurs over the control endpoint. It is done using the SET_REPORT command.

```
/* Reads the OUT Report data */
Out_Data[0] = USBFS_1_DEVICE0_
   CONFIGURATION0_INTERFACE0_ALTERNATE0_
   HID_OUT_BUF[0];
```

The USBFS Component firmware stores the control data in the HID_OUT_BUF[] element shown above on its own. Note that this applies only when using an EP0 with a SET_REPORT. This does not apply when using one of the general-purpose endpoints (EP1 through EP8). More details are covered in AN56377.

The following code shows how simple it is to do an INPUT transfer:

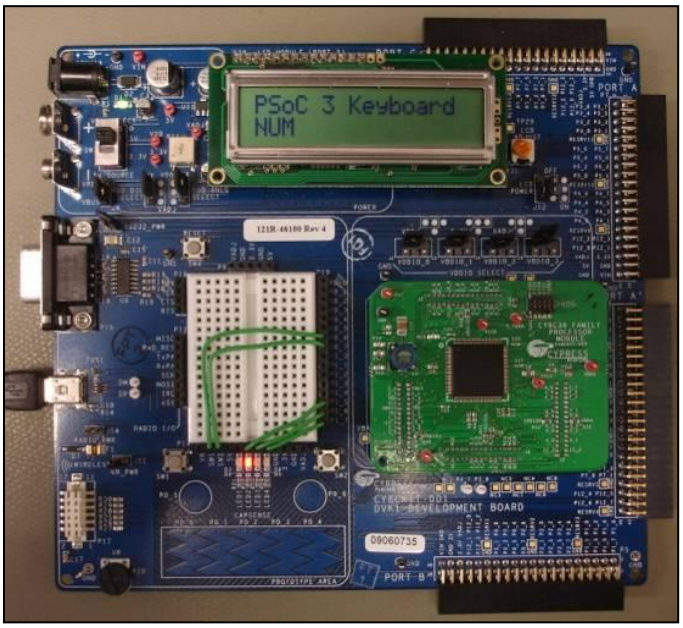
```
/* Loads EP1 for a IN transfer to PSoC
   Creator */
USBFS_1_LoadInEP(1, Keyboard_Data, 8);
/* Waits for ACK from PSoC Creator */
while(!USBFS_1_bGetEPAckState(1));
```

A completed “Project1 – Keyboard” project is associated with this application note. Replace the code in the *main.c* file of your project with the code in the *main.c* file of the associated project.

After the code has been placed in the project, build the project and program it into the PSoC device.

11. Use jumper wires to make the following connections on the CY8CKIT-001, as shown in Figure 23. If you decide to change the pin configuration from Figure 16 on page 16, be sure to change the wiring configuration accordingly.
    - SW1 → P0[6] →Caps Lock Key
    - SW2 → P0[7] → Type Constant String
    - LED2 → P1[5] → Num Lock LED
    - LED3 → P1[6] → Caps Lock LED
    - LED4 → P1[7] → Scroll Lock LED
    - LCD → P2[6:0] → Status Display

Figure 23. Project 1 on CY8CKIT-001



12. After the device is programmed, connect a USB cable from the PC to the DVK. After the device has enumerated, press the [**Caps Lock**], [**Num Lock**], and [**Scroll Lock**] keys on your keyboard and observe the LEDs and LCD change accordingly. Press SW1 on the DVK and observe the Caps Lock toggle. Next, open a text editor on the PC and press SW2. Observe a string of text transmitted to the PC and displayed in the text editor.

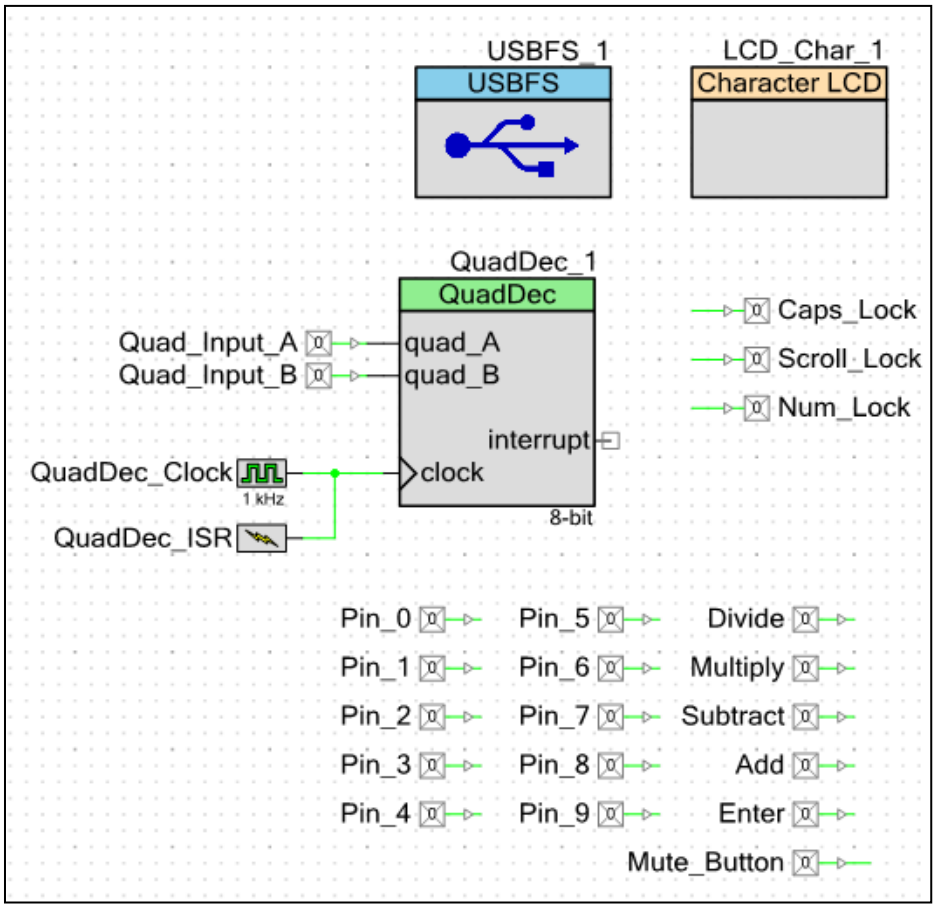
9 Project 2: USB Composite Device

Now that you understand the input and output items, this project will explore other aspects of HID. As stated in Compound and Composite Devices, a composite device lets you create multiple device interfaces in one PSoC device and control each interface independently. This example uses a 10-key pad along with a device that falls into the Consumer Devices HID class. This device implements a volume control dial using a quadrature rotary encoder. You change the volume by turning the dial and mute the volume by pushing a button.

1. Open PSoC Creator and create an empty project named "Project_2_Composite." Place the following Components in the schematic entry page (*TopDesign.cysch*), as shown in Figure 24.

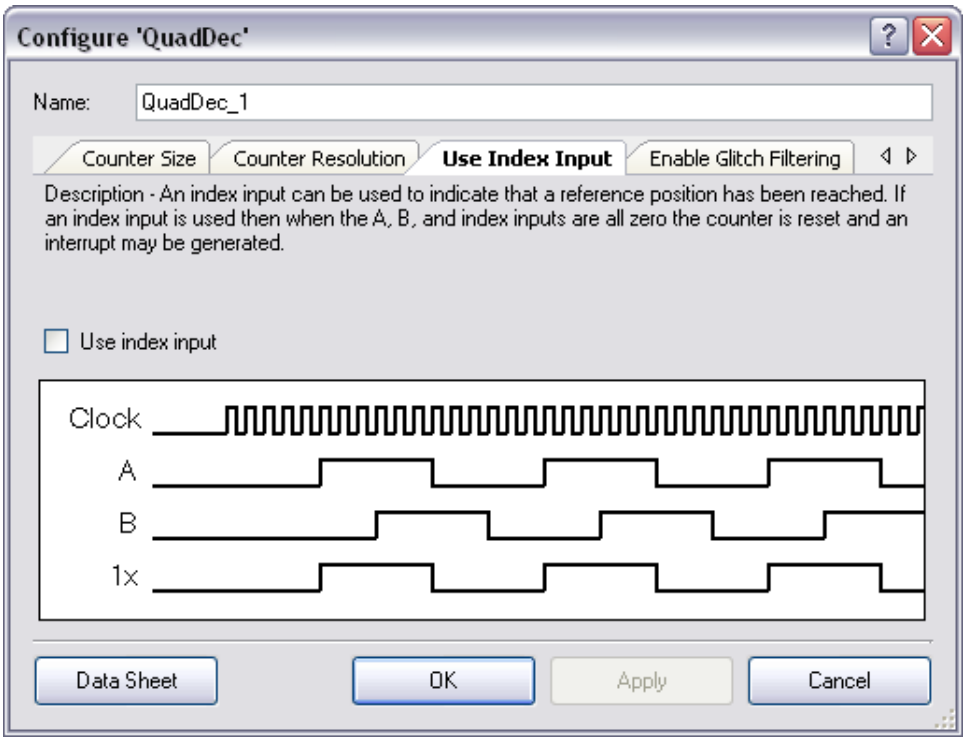
   - Character LCD
   - USBFS
   - Quadrature Decoder
   - Clock
   - ISR
   - (18) Digital Input Pins
   - (3) Digital Output Pins

Figure 24. PSoC Creator Components for Composite Device



The only change that must be made to the QuadDec Component is to remove the index input that is enabled by default. To remove this connection, open the configuration customizer dialog box and deselect the "Use index input" option, as Figure 25 shows.

Figure 25. QuadDec Index Input Settings



2. The pin configuration for this project is similar to that of Project 1. All pins have their “HW Connection” disabled. All output pins are configured to “Strong” drive. All input pins are configured for “Resistive Pull Down.”

   It is better to configure the pins to “Pulled Down” for the “Power-On Reset” state. To do this, for each pin click the **Reset** tab in the customizer and change the “Power-On Reset” selection from the drop-down menu.

3. The clocking selections used in this project and Project 1 are identical. Referencing Figure 15 on page 15, change the clock settings as needed.

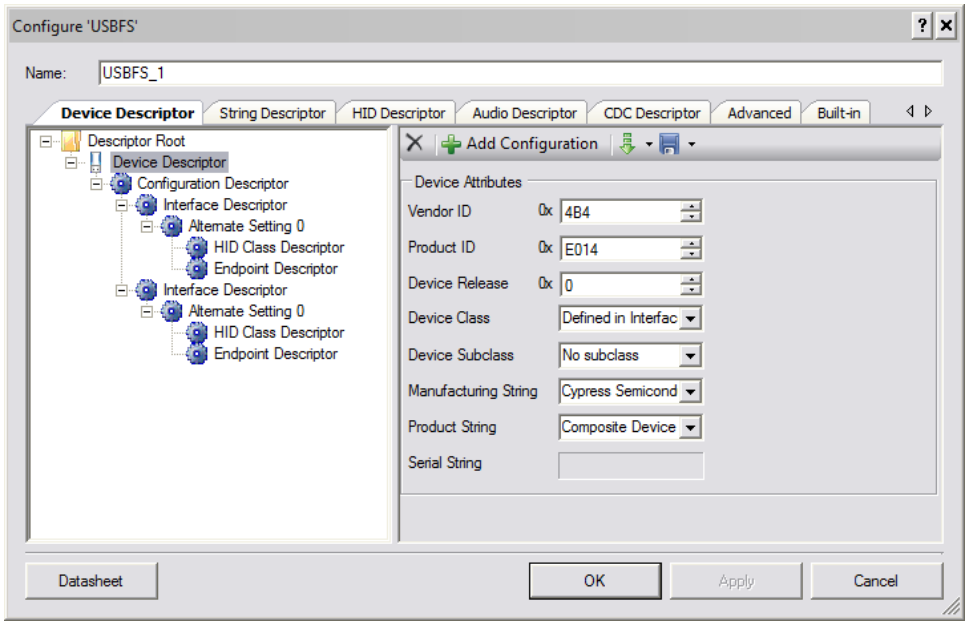
4. Adjust the pin locations as shown in Figure 26.

Figure 26. PSoC Creator Pin Configuration (Composite)

Alias	Name	Pin	Lock
	\LCD_Char_1:LCDPort\[6:0]	P2[6:0]	☑
	\USBFS_1:Dm\	P15[7]	☑
	\USBFS_1:Dp\	P15[6]	☑
	Add	P4[3]	☑
	Caps_Lock	P1[6]	☑
	Divide	P4[0]	☑
	Enter	P4[4]	☑
	Multiply	P4[1]	☑
	Mute_Button	P1[4]	☑
	Num_Lock	P1[5]	☑
	Pin_0	P5[0]	☑
	Pin_1	P5[1]	☑
	Pin_2	P5[2]	☑
	Pin_3	P5[3]	☑
	Pin_4	P5[4]	☑
	Pin_5	P5[5]	☑
	Pin_6	P5[6]	☑
	Pin_7	P5[7]	☑
	Pin_8	P3[0]	☑
	Pin_9	P3[1]	☑
	Quad_Input_A	P1[0]	☑
	Quad_Input_B	P1[1]	☑
	Scroll_Lock	P1[7]	☑
	Subtract	P4[2]	☑

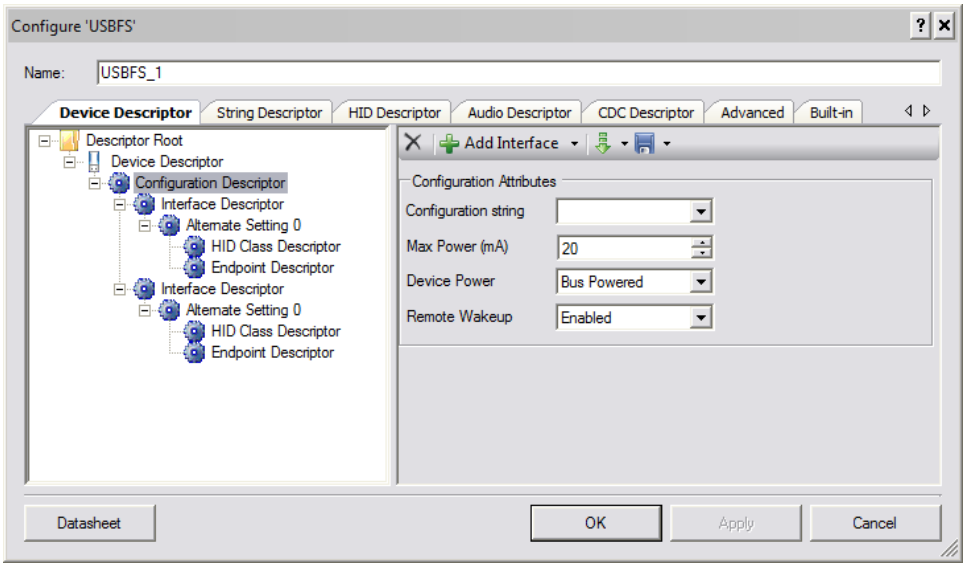
4. Configure the USBFS Component. Because this is an addition to the keyboard from Project 1, most of the USB settings remain the same. Start by changing the “Product ID” and the “Manufacturing String” and “Product String,” as Figure 27 shows. The attached example project uses a PID of 0xE014.

Figure 27. USB Device Descriptor Setup



5. Because this project is a composite device, another interface must be added. Select the “Configuration Descriptor.” Once it is selected, a button to “Add Interface” appears. Click the button to add the second interface, shown in Figure 28. Note that the power settings are the same as those in Project 1.

Figure 28. USB Configuration Descriptor Setup



6. Select “Alternative Setting 0” for each interface and configure them as shown in Figure 29 and Figure 30. To be consistent, make the first interface the keyboard and make the second interface the volume control. Note that an “Interface String” was given to each interface to distinguish them during enumeration and in the system properties in the operating system where they are enumerated.

Figure 29. USB Interface Descriptor Setup (Keyboard)

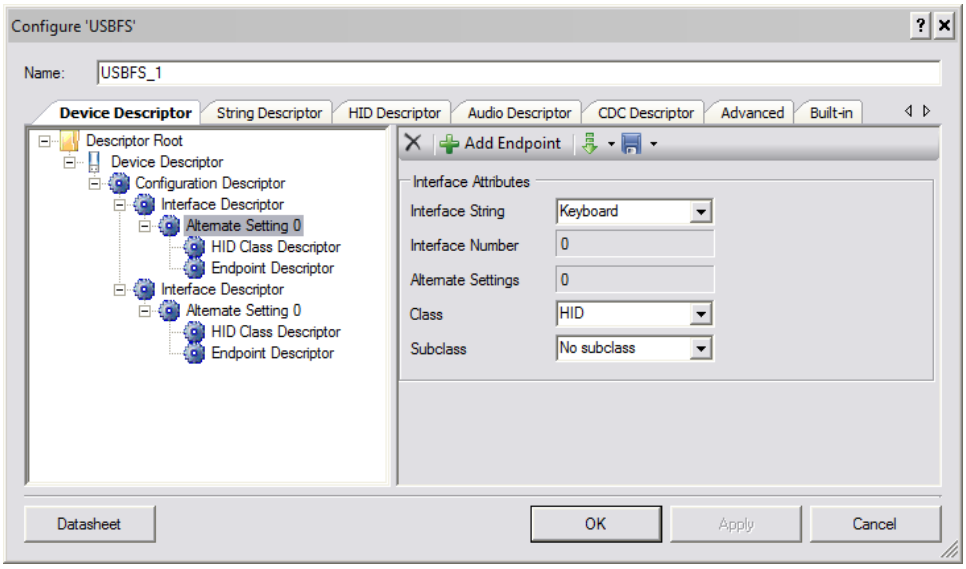
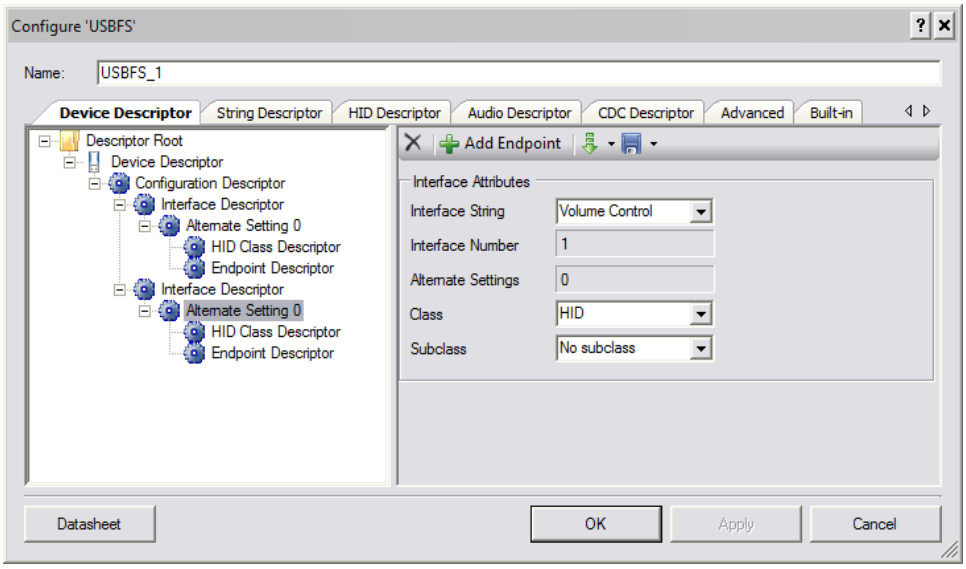


Figure 30. USB Interface Descriptor Setup (Volume)



7. Because there are two independent devices, each device needs its own report descriptor. The report descriptor for the keyboard remains the same as in Project 1. A report descriptor for the volume controller must be added. Refer to Figure 31 and Figure 32 to see how to set up the descriptor. Note that only controls for Volume Up, Volume Down, and Mute are implemented in this instance. A text version of the report descriptor is located in Descriptor Tables for Project 2.

Figure 31. Volume Control Report Descriptor

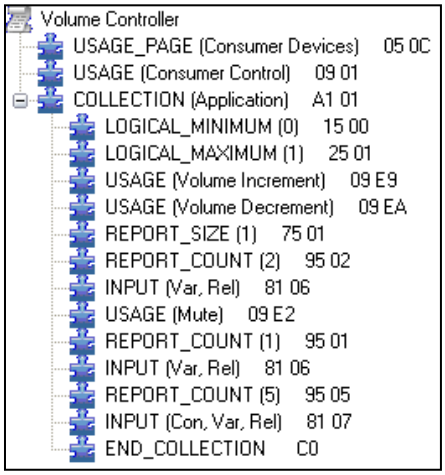
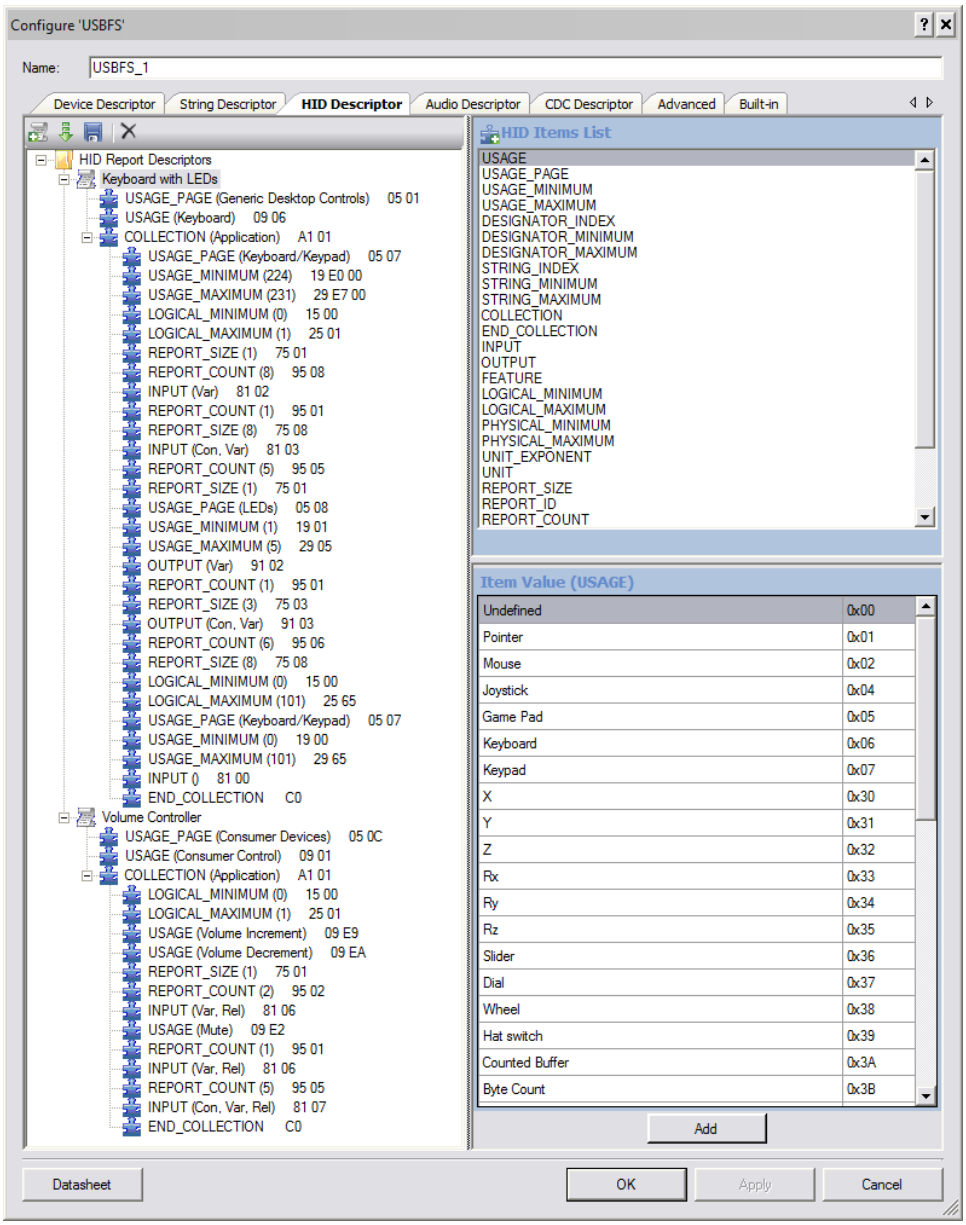


Figure 32. USB HID Report Descriptor Setup



8. After the report descriptor for each device has been created, you need to assign the proper report descriptor to each interface, as shown in Figure 33 and Figure 34.

Figure 33. USB HID Class Descriptor Setup (Keyboard)

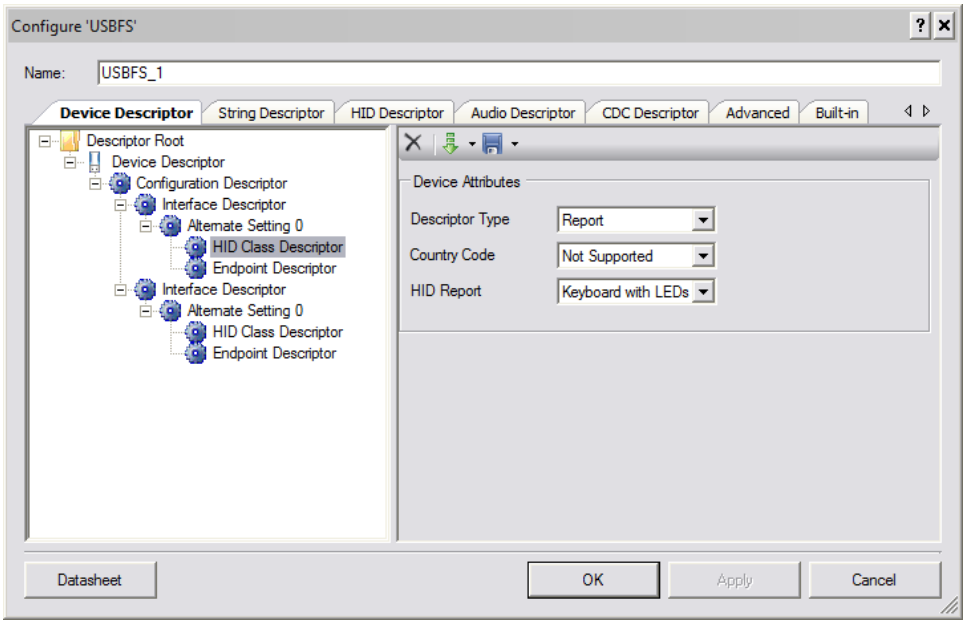
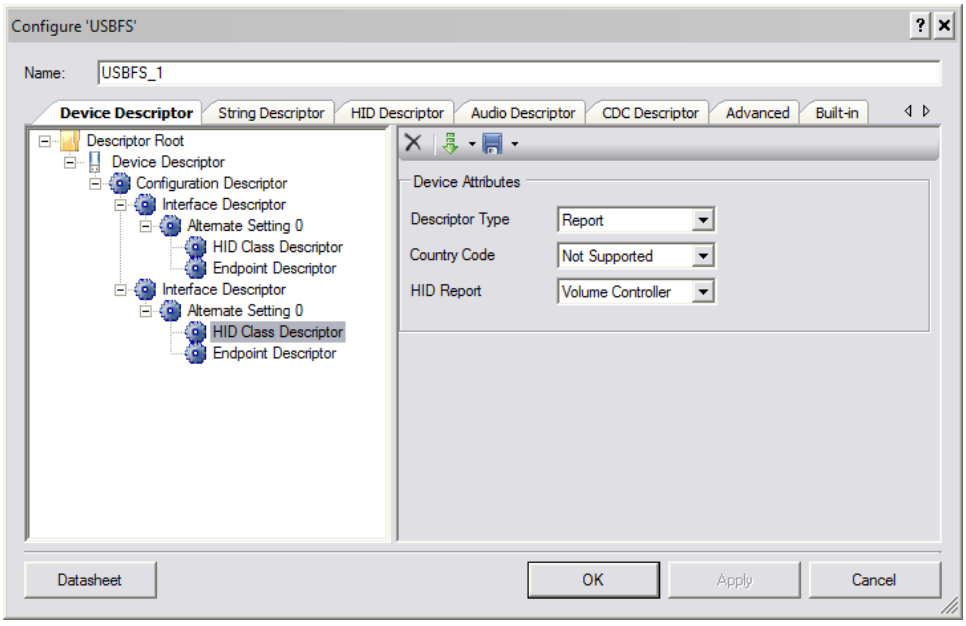


Figure 34. USB HID Class Descriptor Setup (Volume)



9. The final step in the configuration of the USBFS Component is the endpoints. The same endpoint configuration for Project 1 is used in this project. Because another endpoint is required for the volume control, this endpoint is EP3. This endpoint is configured for an INT transfer type and has a max packet size of 1. Figure 35 and Figure 36 show the endpoint configurations.

Figure 35. USB Endpoint Setup (Keyboard IN)

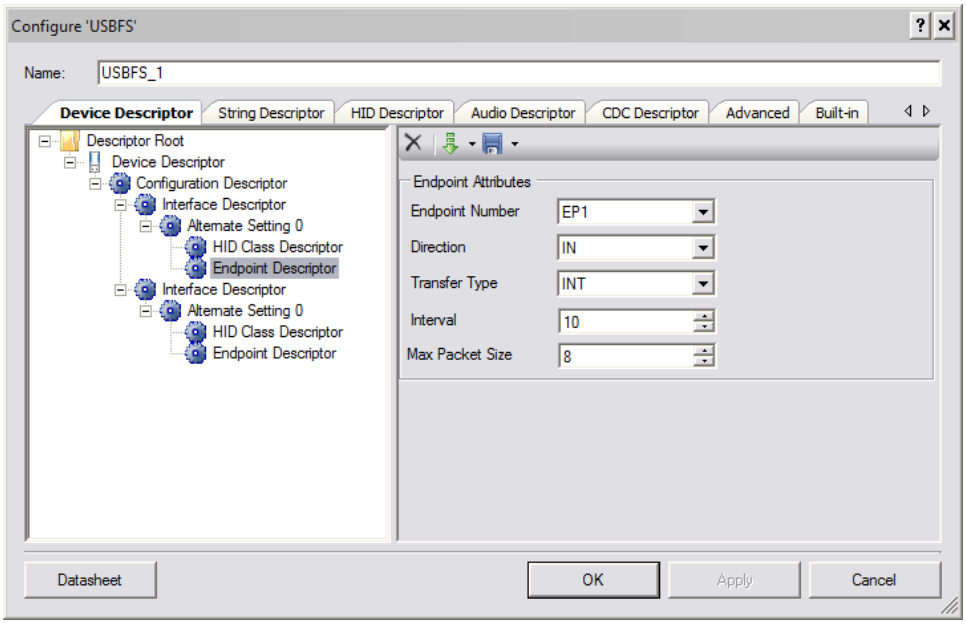
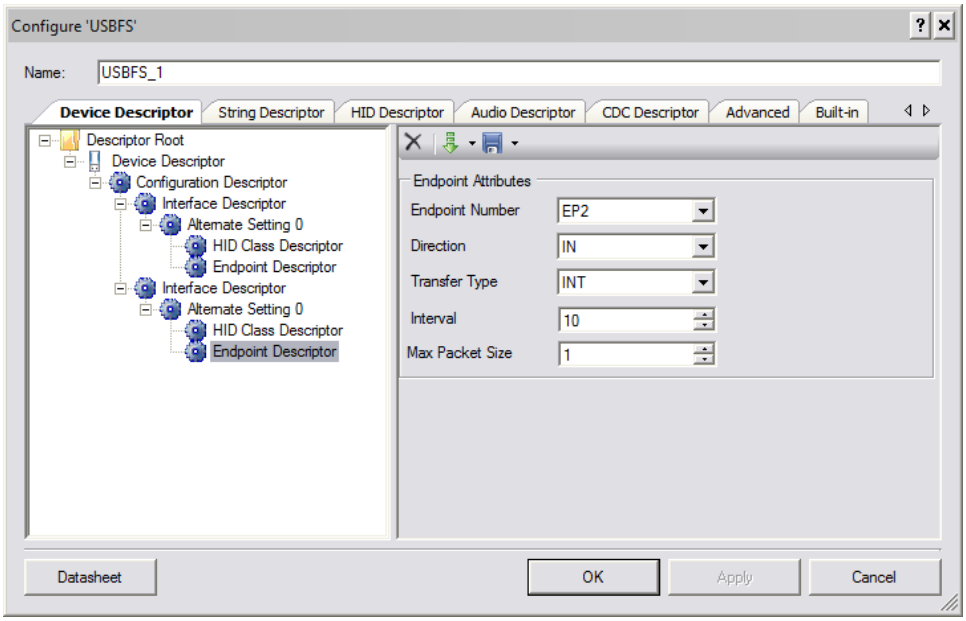


Figure 36. USB Endpoint Setup (Volume Control IN)



10. Locate the *main.c* file in the Workspace Explorer and open the file to edit it.

    The following code shows how easy it is to send data for both devices:

```
/* Loads IN data for Keyboard */
USBFS_1_LoadInEP(1,
   USB_IN_Keyboard_Data, 8);
/* Loads IN data for Volume
   Controller */
USBFS_1_LoadInEP(2,
   USB_IN_Volume_Data, 1);
```

    A completed “Project2 – Composite” project is associated with this application note. Replace the code in the *main.c* file of your project with the code in the *main.c* file of the associated project.

11. In addition, you need to place the following code in *QuadDec_ISR.c*, under the custom includes/defines and the custom ISR code sections. After the code is placed in the project, the next step is to build the project. The descriptor tables for this project are located in Appendix 2 for reference.

```
/***************************************
  Place your includes, defines and code
  here
***************************************/
/* `#START QuadDec_ISR_intc` */
#include <device.h>
extern int8 Volume_Data;
/* `#END` */
```

    AND

```
CY_ISR(QuadDec_ISR_Interrupt)
{
  /* Place your Interrupt code here. */
  /* `#START QuadDec_ISR_Interrupt` */
  Volume_Data = QuadDec_1_GetCounter();
  /* `#END` */
}
```

12. For this project, two demonstration boards are created. One is a small keypad to simulate the number pad on a full-size keyboard. The other board consists of a mechanical quadrature rotary encoder. The schematics for the connections of the encoder and the keypad are shown in Figure 37. While Figure 38 shows that there are two encoders, only the left one is used in this project.

    If a rotary encoder is not available, the project can easily be modified to use the onboard DVK potentiometer and an ADC on the PSoC device or by using multiple pushbuttons. With the encoder and switches connected to the PSoC device, program and reset the device.

Figure 37. Schematics of Rotary Encoder

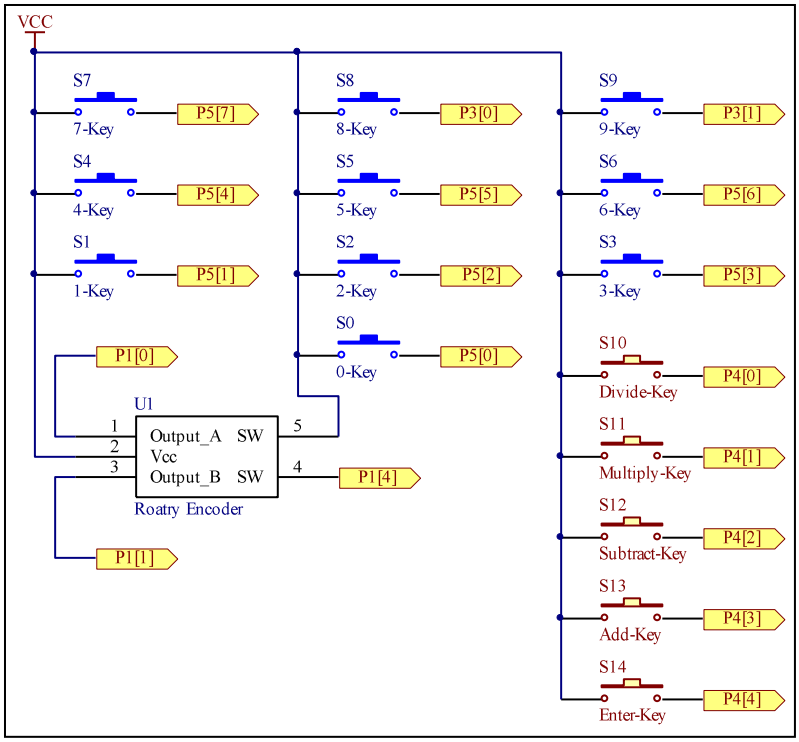
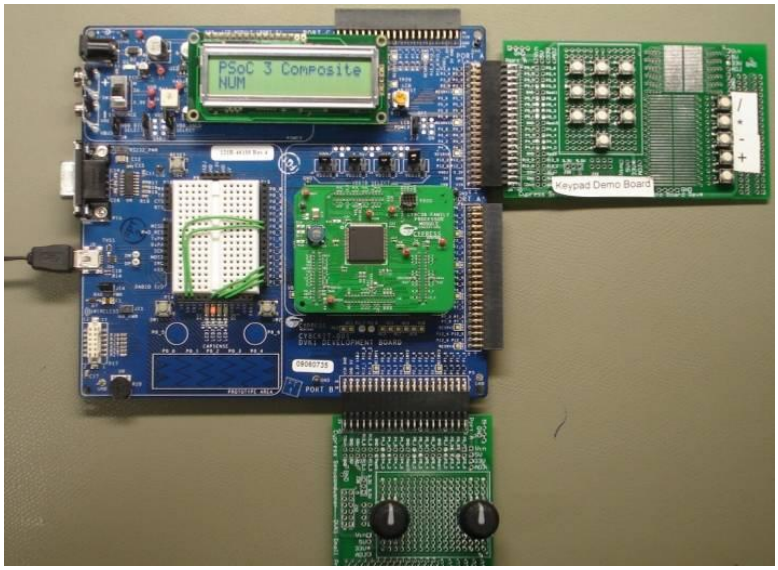


Figure 38. Photograph of Project on PSoC DVK



During the enumeration process, the “Found New Hardware” bubble pops up in the Windows taskbar, informing you that a composite device is detected, followed by the interface string assigned to each interface. Open a calculator application and observe the functionality of the keypad. Play some music and turn the encoder. As you turn the encoder, the volume changes according to the direction in which it is turned.

10 Summary

This application note enables you to feel comfortable enough with USB and HID to create almost any basic USB HID application. While many more features and capabilities of USB are detailed in the USB specification, this application note describes the most central elements of any USB design. To build on these projects, implement the ability to control bass and treble to expand the volume control on a PC to build on Project 2. Another suggestion is to implement a keyboard and mouse composite device.

This application note covered only a small number of the vast possibilities and potential ways to process HID data. One of the many benefits of USB in general is the seemingly endless number of ways to handle and process data.

11 Related Resources

11.1 Application Notes

- AN57294 – USB 101: An Introduction to Universal Serial Bus 2.0
- AN57473 – USB HID Basics with PSoC 3 and PSoC 5LP
- AN56377 – PSoC 3 and PSoC 5LP – Introduction to Implementing USB Data Transfers
- AN82072 – PSoC 3 and PSoC 5LP USB General Data Transfer with Standard HID Drivers
- AN73503 – USB HID Bootloader for PSoC 3 and PSoC 5LP

11.2 Additional Information

- USB HID Device Class Specification
- Cypress PSoC USB Connectivity landing page
- Official USB 3.1 Specification
- *USB Complete*, by Jan Axelson

About the Author

Name:	Robert Murphy
Title:	Application Engineer Sr.
Background:	Robert Murphy graduated from Purdue University with a Bachelors degree in Electrical Engineering Technology.

A Descriptor Tables for Composite Device (HID and CDC)

```
/*******************************************************************
 Device Descriptors
 *******************************************************************/
uint8 USBUART_CODE USBUART_DEVICE0_DESCR[] = {
/* Descriptor Length                        */ 0x12u,
/* DescriptorType: DEVICE                   */ 0x01u,
/* bcdUSB (ver 2.0)                         */ 0x00u, 0x02u,
/* bDeviceClass:                            */ 0xEFu,
/* bDeviceSubClass                          */ 0x02u,
/* bDeviceProtocol                          */ 0x01u,
/* bMaxPacketSize0                          */ 0x08u,
/* idVendor                                 */ 0x34u, 0x12u,
/* idProduct                                */ 0x33u, 0x44u,
/* bcdDevice                                */ 0x00u, 0x00u,
/* iManufacturer                            */ 0x00u,
/* iProduct                                 */ 0x00u,
/* iSerialNumber                            */ 0x00u,
/* bNumConfigurations                       */ 0x01u
};

/*******************************************************************
 Config Descriptor
 *******************************************************************/
uint8 USBUART_CODE USBUART_DEVICE0_CONFIGURATION0_DESCR[] = {
/*  Config Descriptor Length                */ 0x09u,
/*  DescriptorType: CONFIG                  */ 0x02u,
/*  wTotalLength: "Hand Computed"  67B      */ 0x64u, 0x00u,
/*  bNumInterfaces                          */ 0x03u,
/*  bConfigurationValue                     */ 0x01u,
/*  iConfiguration                          */ 0x00u,
/*  bmAttributes                            */ 0x80u,
/*  bMaxPower                               */ 0x31u,
/*******************************************************************
Interface Association Descriptor (CDC)
 *******************************************************************/
/*  bLength                                 */ 0x08u,
/*  bDescriptorType: IAD                    */ 0x0Bu,
/*  bFirstInterface                         */ 0x00u,
/*  bInterfaceCount                         */ 0x02u,
/*  bFunctionClass                          */ 0x02u,
/*  bFunctionSubClass                       */ 0x02u,
/*  bFunctionProtocol                       */ 0x01u,
/*  iFunction                               */ 0x00u,

/*******************************************************************
 Interface Descriptor (CDC Communications Class Interface)
 *******************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x00u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x01u,
/*  bInterfaceClass : Communication i/f     */ 0x02u,
/*  bInterfaceSubClass : ACM                */ 0x02u,
/*  bInterfaceProtocol : ATcmds             */ 0x01u,
/*  iInterface                              */ 0x00u,
```

```
/*************************************************************************
 CDC Class-specific Descriptors
 Header Functional Descriptor
 *************************************************************************/
/*  Descriptor Length                       */ 0x05u,
/*  DescriptorType: CS_INTERFACE            */ 0x24u,
/*  DescriptorSubType: Header Functional    */ 0x00u,
/*  bcdCDC: CDC Release Number              */ 0x10u, 0x01u,

/*************************************************************************
 CDC Class-specific Descriptors
 Call Management Functional Descriptor
 *************************************************************************/
/*  Descriptor Length                       */ 0x05u,
/*  DescriptorType: CS_INTERFACE            */ 0x24u,
/*  DescriptorSubType: Call Management      */ 0x01u,
/*  bmCapabilities:                         */ 0x01u,
/*  bInterface: i/f # of Data Class i/f     */ 0x01u,

/*************************************************************************
 CDC Class-specific Descriptors
 Abstract Control Management Functional Descriptor
 *************************************************************************/
/*  Descriptor Length                       */ 0x04u,
/*  DescriptorType: CS_INTERFACE            */ 0x24u,
/*  DescriptorSubType: ACM                  */ 0x02u,
/*  bmCapabilities :                        */ 0x06u,

/*************************************************************************
 CDC Class-specific Descriptors
 Union Functional Descriptor
 *************************************************************************/
/*  Descriptor Length                       */ 0x05u,
/*  DescriptorType: CS_INTERFACE            */ 0x24u,
/*  DescriptorSubType: Union Functional     */ 0x06u,
/*  bMasterInteface                         */ 0x00u,
/*  bSlaveInterface                         */ 0x01u,

/*************************************************************************
 Endpoint Descriptor (CDC)
 *************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x81u,
/*  bmAttributes                            */ 0x03u,
/*  wMaxPacketSize                          */ 0x08u, 0x00u,
/*  bInterval                               */ 0x1Au,
/*************************************************************************
 Interface Descriptor (CDC Data Interface)
 *************************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x01u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x02u,
/*  bInterfaceClass : Data Interface        */ 0x0Au,
/*  bInterfaceSubClass                      */ 0x00u,
/*  bInterfaceProtocol                      */ 0x00u,
/*  iInterface                              */ 0x00u,
/*************************************************************************
 Endpoint Descriptor (CDC)
```

```
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x82u,
/*  bmAttributes                            */ 0x02u,
/*  wMaxPacketSize                          */ 0x40u, 0x00u,
/*  bInterval                               */ 0x00u,
/***************************************************************************
 Endpoint Descriptor (CDC)
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x03u,
/*  bmAttributes                            */ 0x02u,
/*  wMaxPacketSize                          */ 0x40u, 0x00u,
/*  bInterval                               */ 0x00u,
/***************************************************************************
 Interface Descriptor (HID)
 ***************************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x02u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x01u,
/*  bInterfaceClass                         */ 0x03u,
/*  bInterfaceSubClass                      */ 0x00u,
/*  bInterfaceProtocol                      */ 0x00u,
/*  iInterface                              */ 0x00u,
/***************************************************************************
 HID Class Descriptor (HID)
 ***************************************************************************/
/*  HID Class Descriptor Length             */ 0x09u,
/*  DescriptorType: HID_CLASS               */ 0x21u,
/*  bcdHID                                  */ 0x11u, 0x01u,
/*  bCountryCode                            */ 0x00u,
/*  bNumDescriptors                         */ 0x01u,
/*  bDescriptorType                         */ 0x22u,
/*  wDescriptorLength (LSB)                 */ USBUART_HID_RPT_1_SIZE_LSB,
/*  wDescriptorLength (MSB)                 */ USBUART_HID_RPT_1_SIZE_MSB,
/***************************************************************************
 Endpoint Descriptor (HID)
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x84u,
/*  bmAttributes                            */ 0x03u,
/*  wMaxPacketSize     kmc-was3,0           */ 0x08u, 0x00u,
/*  bInterval                               */ 0x1Au
};

/***************************************************************************
 String Descriptor Table
 ***************************************************************************/
uint8 USBUART_CODE USBUART_STRING_DESCRIPTORS[] = {
/***************************************************************************
 Language ID Descriptor
 ***************************************************************************/
/* Descriptor Length                        */ 0x04u,
/* DescriptorType: STRING                   */ 0x03u,
/* Language Id                              */ 0x09u, 0x04u,
/***************************************************************************
```

```
 String Descriptor: "Cypress Semiconductor"
 *************************************************************/
 /* Descriptor Length                      */ 0x2Cu,
/* DescriptorType: STRING                  */ 0x03u,
 'C', 0,'y', 0,'p', 0,'r', 0,'e', 0,'s', 0,'s', 0,' ', 0,'S', 0,'e', 0
,'m', 0,'i', 0,'c', 0,'o', 0,'n', 0,'d', 0,'u', 0,'c', 0,'t', 0,'o', 0
,'r', 0,
/*************************************************************
 String Descriptor: "Square Mouse"
 *************************************************************/
 /* Descriptor Length                      */ 0x1Au,
/* DescriptorType: STRING                  */ 0x03u,
 'S', 0,'q', 0,'u', 0,'a', 0,'r', 0,'e', 0,' ', 0,'M', 0,'o', 0,'u', 0
,'s', 0,'e', 0,
/*************************************************************/
/* Marks the end of the list.              */ 0x00u};
/*************************************************************/
/*************************************************************
 Serial Number String Descriptor
 *************************************************************/
uint8 USBUART_CODE USBUART_SN_STRING_DESCRIPTOR[] = {
/* Descriptor Length                      */ 0x02u,
/* DescriptorType: STRING                  */ 0x03u
};
/*************************************************************
 HID Report Descriptor: 3-Button Mouse
 *************************************************************/
uint8 USBUART_CODE USBUART_HIDREPORT_DESCRIPTOR1[] = {
/*  Descriptor Size                       */ USBUART_HID_RPT_1_SIZE_LSB,
                                              USBUART_HID_RPT_1_SIZE_MSB,
/* USAGE_PAGE                             */ 0x05u, 0x01u,
/* USAGE                                  */ 0x09u, 0x02u,
/* COLLECTION                             */ 0xA1u, 0x01u,
/* USAGE                                  */ 0x09u, 0x01u,
/* COLLECTION                             */ 0xA1u, 0x00u,
/* USAGE_PAGE                             */ 0x05u, 0x09u,
/* USAGE_MINIMUM                          */ 0x19u, 0x01u,
/* USAGE_MAXIMUM                          */ 0x29u, 0x03u,
/* LOGICAL_MINIMUM                        */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                        */ 0x25u, 0x01u,
/* REPORT_COUNT                           */ 0x95u, 0x03u,
/* REPORT_SIZE                            */ 0x75u, 0x01u,
/* INPUT                                  */ 0x81u, 0x02u,
/* REPORT_COUNT                           */ 0x95u, 0x01u,
/* REPORT_SIZE                            */ 0x75u, 0x05u,
/* INPUT                                  */ 0x81u, 0x03u,
/* USAGE_PAGE                             */ 0x05u, 0x01u,
/* USAGE                                  */ 0x09u, 0x30u,
/* USAGE                                  */ 0x09u, 0x31u,
/* LOGICAL_MINIMUM                        */ 0x15u, 0x80u,
/* LOGICAL_MAXIMUM                        */ 0x25u, 0x7Fu,
/* REPORT_SIZE                            */ 0x75u, 0x08u,
/* REPORT_COUNT                           */ 0x95u, 0x02u,
/* INPUT                                  */ 0x81u, 0x06u,
/* END_COLLECTION                         */ 0xC0u,
/* END_COLLECTION                         */ 0xC0u,
/*************************************************************/
/* End of the HID Report Descriptor       */ 0x00u, 0x00u};
/*************************************************************
*/
```

B Descriptor Tables for Project 1

```
/*******************************************************************
 Device Descriptors
 *******************************************************************/
/* Descriptor Length                        */ 0x12u,
/* DescriptorType: DEVICE                   */ 0x01u,
/* bcdUSB (ver 2.0)                         */ 0x00u, 0x02u,
/* bDeviceClass                             */ 0x00u,
/* bDeviceSubClass                          */ 0x00u,
/* bDeviceProtocol                          */ 0x00u,
/* bMaxPacketSize0                          */ 0x08u,
/* idVendor                                 */ 0xB4u, 0x04u,
/* idProduct                                */ 0x13u, 0xE0u,
/* bcdDevice                                */ 0x00u, 0x00u,
/* iManufacturer                            */ 0x01u,
/* iProduct                                 */ 0x02u,
/* iSerialNumber                            */ 0x00u,
/* bNumConfigurations                       */ 0x01u

/*******************************************************************
 Config Descriptor
 *******************************************************************/
/*  Config Descriptor Length                */ 0x09u,
/*  DescriptorType: CONFIG                  */ 0x02u,
/*  wTotalLength                            */ 0x22u, 0x00u,
/*  bNumInterfaces                          */ 0x01u,
/*  bConfigurationValue                     */ 0x01u,
/*  iConfiguration                          */ 0x00u,
/*  bmAttributes                            */ 0xA0u,
/*  bMaxPower                               */ 0x0Au,

/*******************************************************************
 Interface Descriptor
 *******************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x00u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x01u,
/*  bInterfaceClass                         */ 0x03u,
/*  bInterfaceSubClass                      */ 0x00u,
/*  bInterfaceProtocol                      */ 0x00u,
/*  iInterface                              */ 0x00u,

/*******************************************************************
 HID Class Descriptor
 *******************************************************************/
/*  HID Class Descriptor Length             */ 0x09u,
/*  DescriptorType: HID_CLASS               */ 0x21u,
/*  bcdHID                                  */ 0x11u, 0x01u,
/*  bCountryCode                            */ 0x00u,
/*  bNumDescriptors                         */ 0x01u,
/*  bDescriptorType                         */ 0x22u,
/*  wDescriptorLength (LSB)                 */ USBFS_1_HID_RPT_1_SIZE_LSB,
/*  wDescriptorLength (MSB)                 */ USBFS_1_HID_RPT_1_SIZE_MSB,

/*******************************************************************
 Endpoint Descriptor
```

```
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x81u,
/*  bmAttributes                            */ 0x03u,
/*  wMaxPacketSize                          */ 0x08u, 0x00u,
/*  bInterval                               */ 0x0Au


/*******************************************************************
 String Descriptor Table
 *******************************************************************/

/*******************************************************************
 Language ID Descriptor
 *******************************************************************/
/* Descriptor Length                        */ 0x04u,
/* DescriptorType: STRING                   */ 0x03u,
/* Language Id                              */ 0x09u, 0x04u,

/*******************************************************************
 String Descriptor: "Cypress Semiconductor"
 *******************************************************************/
/* Descriptor Length                        */ 0x2Cu,
/* DescriptorType: STRING                   */ 0x03u,
 'C', 0,'y', 0,'p', 0,'r', 0,'e', 0,'s', 0,'s', 0,' ', 0,'S', 0,'e', 0
,'m', 0,'i', 0,'c', 0,'o', 0,'n', 0,'d', 0,'u', 0,'c', 0,'t', 0,'o', 0
,'r', 0,

/*******************************************************************
 String Descriptor: "Keyboard"
 *******************************************************************/
/* Descriptor Length                        */ 0x12u,
/* DescriptorType: STRING                   */ 0x03u,
 'K', 0,'e', 0,'y', 0,'b', 0,'o', 0,'a', 0,'r', 0,'d', 0,

/*******************************************************************
 HID Report Descriptor: Keyboard with LEDs
 *******************************************************************/
/* Descriptor Size (Not part of descriptor)*/ USBFS_1_HID_RPT_1_SIZE_LSB,
                                               USBFS_1_HID_RPT_1_SIZE_MSB,
/* USAGE_PAGE                               */ 0x05u, 0x01u,
/* USAGE                                    */ 0x09u, 0x06u,
/* COLLECTION                               */ 0xA1u, 0x01u,
/* USAGE_PAGE                               */ 0x05u, 0x07u,
/* USAGE_MINIMUM                            */ 0x19u, 0xE0u,
/* USAGE_MAXIMUM                            */ 0x29u, 0xE7u,
/* LOGICAL_MINIMUM                          */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                          */ 0x25u, 0x01u,
/* REPORT_SIZE                              */ 0x75u, 0x01u,
/* REPORT_COUNT                             */ 0x95u, 0x08u,
/* INPUT                                    */ 0x81u, 0x02u,
/* REPORT_COUNT                             */ 0x95u, 0x01u,
/* REPORT_SIZE                              */ 0x75u, 0x08u,
/* INPUT                                    */ 0x81u, 0x03u,
/* REPORT_COUNT                             */ 0x95u, 0x05u,
/* REPORT_SIZE                              */ 0x75u, 0x01u,
/* USAGE_PAGE                               */ 0x05u, 0x08u,
/* USAGE_MINIMUM                            */ 0x19u, 0x01u,
/* USAGE_MAXIMUM                            */ 0x29u, 0x05u,
/* OUTPUT                                   */ 0x91u, 0x02u,
```

```
/* REPORT_COUNT                    */ 0x95u, 0x01u,
/* REPORT_SIZE                     */ 0x75u, 0x03u,
/* OUTPUT                          */ 0x91u, 0x03u,
/* REPORT_COUNT                    */ 0x95u, 0x06u,
/* REPORT_SIZE                     */ 0x75u, 0x08u,
/* LOGICAL_MINIMUM                 */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                 */ 0x25u, 0x65u,
/* USAGE_PAGE                      */ 0x05u, 0x07u,
/* USAGE_MINIMUM                   */ 0x19u, 0x00u,
/* USAGE_MAXIMUM                   */ 0x29u, 0x65u,
/* INPUT                           */ 0x81u, 0x00u,
/* END_COLLECTION                  */ 0xC0u,
```

C Descriptor Tables for Project 2

```
/*******************************************************************
 Device Descriptors
 *******************************************************************/
/* Descriptor Length                        */ 0x12u,
/* DescriptorType: DEVICE                   */ 0x01u,
/* bcdUSB (ver 2.0)                         */ 0x00u, 0x02u,
/* bDeviceClass                             */ 0x00u,
/* bDeviceSubClass                          */ 0x00u,
/* bDeviceProtocol                          */ 0x00u,
/* bMaxPacketSize0                          */ 0x08u,
/* idVendor                                 */ 0xB4u, 0x04u,
/* idProduct                                */ 0x14u, 0xE0u,
/* bcdDevice                                */ 0x00u, 0x00u,
/* iManufacturer                            */ 0x01u,
/* iProduct                                 */ 0x03u,
/* iSerialNumber                            */ 0x00u,
/* bNumConfigurations                       */ 0x01u

/*******************************************************************
 Config Descriptor
 *******************************************************************/
/*  Config Descriptor Length                */ 0x09u,
/*  DescriptorType: CONFIG                  */ 0x02u,
/*  wTotalLength                            */ 0x3Bu, 0x00u,
/*  bNumInterfaces                          */ 0x02u,
/*  bConfigurationValue                     */ 0x01u,
/*  iConfiguration                          */ 0x00u,
/*  bmAttributes                            */ 0xA0u,
/*  bMaxPower                               */ 0x0Au,

/*******************************************************************
 Interface Descriptor
 *******************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x00u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x01u,
/*  bInterfaceClass                         */ 0x03u,
/*  bInterfaceSubClass                      */ 0x00u,
/*  bInterfaceProtocol                      */ 0x00u,
/*  iInterface                              */ 0x02u,

/*******************************************************************
 HID Class Descriptor
 *******************************************************************/
/*  HID Class Descriptor Length             */ 0x09u,
/*  DescriptorType: HID_CLASS               */ 0x21u,
/*  bcdHID                                  */ 0x11u, 0x01u,
/*  bCountryCode                            */ 0x00u,
/*  bNumDescriptors                         */ 0x01u,
/*  bDescriptorType                         */ 0x22u,
/*  wDescriptorLength (LSB)                 */ USBFS_1_HID_RPT_1_SIZE_LSB,
/*  wDescriptorLength (MSB)                 */ USBFS_1_HID_RPT_1_SIZE_MSB,

/*******************************************************************
 Endpoint Descriptor
```

```
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x81u,
/*  bmAttributes                            */ 0x03u,
/*  wMaxPacketSize                          */ 0x08u, 0x00u,
/*  bInterval                               */ 0x0Au,


/*******************************************************************
 Interface Descriptor
 ***************************************************************************/
/*  Interface Descriptor Length             */ 0x09u,
/*  DescriptorType: INTERFACE               */ 0x04u,
/*  bInterfaceNumber                        */ 0x01u,
/*  bAlternateSetting                       */ 0x00u,
/*  bNumEndpoints                           */ 0x01u,
/*  bInterfaceClass                         */ 0x03u,
/*  bInterfaceSubClass                      */ 0x00u,
/*  bInterfaceProtocol                      */ 0x00u,
/*  iInterface                              */ 0x04u,

/*******************************************************************
 HID Class Descriptor
 ***************************************************************************/
/*  HID Class Descriptor Length             */ 0x09u,
/*  DescriptorType: HID_CLASS               */ 0x21u,
/*  bcdHID                                  */ 0x11u, 0x01u,
/*  bCountryCode                            */ 0x00u,
/*  bNumDescriptors                         */ 0x01u,
/*  bDescriptorType                         */ 0x22u,
/*  wDescriptorLength (LSB)                 */ USBFS_1_HID_RPT_2_SIZE_LSB,
/*  wDescriptorLength (MSB)                 */ USBFS_1_HID_RPT_2_SIZE_MSB,

/*******************************************************************
 Endpoint Descriptor
 ***************************************************************************/
/*  Endpoint Descriptor Length              */ 0x07u,
/*  DescriptorType: ENDPOINT                */ 0x05u,
/*  bEndpointAddress                        */ 0x82u,
/*  bmAttributes                            */ 0x03u,
/*  wMaxPacketSize                          */ 0x01u, 0x00u,
/*  bInterval                               */ 0x0Au

/*******************************************************************
 String Descriptor Table
 ***************************************************************************/

/*******************************************************************
 Language ID Descriptor
 ***************************************************************************/
/* Descriptor Length                        */ 0x04u,
/* DescriptorType: STRING                   */ 0x03u,
/* Language Id                              */ 0x09u, 0x04u,

/*******************************************************************
 String Descriptor: "Cypress Semiconductor"
 ***************************************************************************/
/* Descriptor Length                        */ 0x2Cu,
/* DescriptorType: STRING                   */ 0x03u,
 'C', 0,'y', 0,'p', 0,'r', 0,'e', 0,'s', 0,'s', 0,' ', 0,'S', 0,'e', 0
```

```
,'m', 0,'i', 0,'c', 0,'o', 0,'n', 0,'d', 0,'u', 0,'c', 0,'t', 0,'o', 0
,'r', 0,

/*******************************************************************
 String Descriptor: "Keyboard"
 *******************************************************************/
/* Descriptor Length                        */ 0x12u,
/* DescriptorType: STRING                   */ 0x03u,
 'K', 0,'e', 0,'y', 0,'b', 0,'o', 0,'a', 0,'r', 0,'d', 0,

/*******************************************************************
 String Descriptor: "Composite Device Demo"
 *******************************************************************/
/* Descriptor Length                        */ 0x2Cu,
/* DescriptorType: STRING                   */ 0x03u,
 'C', 0,'o', 0,'m', 0,'p', 0,'o', 0,'s', 0,'i', 0,'t', 0,'e', 0,' ', 0
,'D', 0,'e', 0,'v', 0,'i', 0,'c', 0,'e', 0,' ', 0,'D', 0,'e', 0,'m', 0
,'o', 0,

/*******************************************************************
 String Descriptor: "Volume Control"
 *******************************************************************/
/* Descriptor Length                        */ 0x1Eu,
/* DescriptorType: STRING                   */ 0x03u,
 'V', 0,'o', 0,'l', 0,'u', 0,'m', 0,'e', 0,' ', 0,'C', 0,'o', 0,'n', 0
,'t', 0,'r', 0,'o', 0,'l', 0,



/*******************************************************************
 Serial Number String Descriptor
 *******************************************************************/
/* Descriptor Length                        */ 0x02u,
/* DescriptorType: STRING                   */ 0x03u

/*******************************************************************
 HID Report Descriptor: Keyboard with LEDs
 *******************************************************************/
/*  Descriptor Size (Not part of descriptor)*/ USBFS_1_HID_RPT_1_SIZE_LSB,
                                               USBFS_1_HID_RPT_1_SIZE_MSB,
/* USAGE_PAGE                               */ 0x05u, 0x01u,
/* USAGE                                    */ 0x09u, 0x06u,
/* COLLECTION                               */ 0xA1u, 0x01u,
/* USAGE_PAGE                               */ 0x05u, 0x07u,
/* USAGE_MINIMUM                            */ 0x19u, 0xE0u,
/* USAGE_MAXIMUM                            */ 0x29u, 0xE7u,
/* LOGICAL_MINIMUM                          */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                          */ 0x25u, 0x01u,
/* REPORT_SIZE                              */ 0x75u, 0x01u,
/* REPORT_COUNT                             */ 0x95u, 0x08u,
/* INPUT                                    */ 0x81u, 0x02u,
/* REPORT_COUNT                             */ 0x95u, 0x01u,
/* REPORT_SIZE                              */ 0x75u, 0x08u,
/* INPUT                                    */ 0x81u, 0x03u,
/* REPORT_COUNT                             */ 0x95u, 0x05u,
/* REPORT_SIZE                              */ 0x75u, 0x01u,
/* USAGE_PAGE                               */ 0x05u, 0x08u,
/* USAGE_MINIMUM                            */ 0x19u, 0x01u,
/* USAGE_MAXIMUM                            */ 0x29u, 0x05u,
/* OUTPUT                                   */ 0x91u, 0x02u,
/* REPORT_COUNT                             */ 0x95u, 0x01u,
```

```
/* REPORT_SIZE                                  */ 0x75u, 0x03u,
/* OUTPUT                                       */ 0x91u, 0x03u,
/* REPORT_COUNT                                 */ 0x95u, 0x06u,
/* REPORT_SIZE                                  */ 0x75u, 0x08u,
/* LOGICAL_MINIMUM                              */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                              */ 0x25u, 0x65u,
/* USAGE_PAGE                                   */ 0x05u, 0x07u,
/* USAGE_MINIMUM                                */ 0x19u, 0x00u,
/* USAGE_MAXIMUM                                */ 0x29u, 0x65u,
/* INPUT                                        */ 0x81u, 0x00u,
/* END_COLLECTION                               */ 0xC0u,

/*****************************************************************
 HID Report Descriptor: Volume Controller
 *****************************************************************/
/*  Descriptor Size (Not part of descriptor)*/ USBFS_1_HID_RPT_2_SIZE_LSB,
                                                USBFS_1_HID_RPT_2_SIZE_MSB,
/* USAGE_PAGE                                   */ 0x05u, 0x0Cu,
/* USAGE                                        */ 0x09u, 0x01u,
/* COLLECTION                                   */ 0xA1u, 0x01u,
/* LOGICAL_MINIMUM                              */ 0x15u, 0x00u,
/* LOGICAL_MAXIMUM                              */ 0x25u, 0x01u,
/* USAGE                                        */ 0x09u, 0xE9u,
/* USAGE                                        */ 0x09u, 0xEAu,
/* REPORT_SIZE                                  */ 0x75u, 0x01u,
/* REPORT_COUNT                                 */ 0x95u, 0x02u,
/* INPUT                                        */ 0x81u, 0x06u,
/* USAGE                                        */ 0x09u, 0xE2u,
/* REPORT_COUNT                                 */ 0x95u, 0x01u,
/* INPUT                                        */ 0x81u, 0x06u,
/* REPORT_COUNT                                 */ 0x95u, 0x05u,
/* INPUT                                        */ 0x81u, 0x07u,
/* END_COLLECTION                               */ 0xC0u,
```

Document History

Document Title: AN58726 - USB HID Intermediate with PSoC® 3 and PSoC 5LP

Document Number: 001-58726

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2846059	YJI/RLRM	01/11/2010	New application note
*A	2991550	SRIH	07/22/2010	Fixed branding discrepancies
*B	3128709	RLRM	01/05/2011	Updated associated project file
*C	3159035	RLRM	02/21/2011	Significant updates to the project files. Updated the title and abstract. Added additional information in Report Descriptor section. Added Additional Resources section. Added Appendix 5.
*D	3204826	RLRM	03/24/2011	Updated Abstract (changed AN57473 as AN58726 in the very first sentence).
*E	3452272	RLRM	12/02/2011	Template update Updated project files to Creator 2.0 Minor content change to account for changes made to project.
*F	3809380	DASG	11/12/2012	Updated for PSoC 5LP
*G	4029948	MKEA	06/15/2013	Changed Title, Abstract, Introduction, updated keywords in document properties, updated related application notes section. Deleted code appendices 1 and 3, and replaced with references to code in associated projects. Renumbered remaining appendices. Miscellaneous formatting and readability improvements.
*H	4646232	KLMZ	01/30/2015	Reorganized and added clarifying text Removed obsolete application notes from related resources Added information on HID templates Corrected appendix references Sunset review
*I	5220040	RLRM	04/13/2016	Update to document template and project files. Updated template
*J	5700273	AESATP12	04/28/2017	Updated logo and copyright

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