AN58726

PSoC® 3 / PSoC 5 USB HID Intermediate (with Keyboard and Composite Device)

Author: Robert Murphy
Associated Project: Yes
Associated Part Family: CY8C38xxx/CY8C55xxx
Software Version: PSoC® Creator™ 2.0
Related Application Notes: AN57294, AN57473

Abstract

AN58726 covers the next level of human interface devices (HID) USB development on PSoC® 3 / PSoC 5 discussing OUTPUT items using a keyboard and a composite device as example projects. Using PSoC's full speed USB interface, this application note will take the basics of USB HID development learned in AN57473, and builds upon that knowledge by teaching users how to incorporate OUTPUT items to receive information from a host device using the status LEDs on a keyboard as an example, while also sending keyboard information as an INPUT to type a predefined string of text into a text editor. Users of this application note will also learn how to create a composite device by combining a numeric keypad along with a PC volume controller using a quadrature decoder into a single device with a separate interface. For additional USB design resources, see the 'Related Application Notes and Example Projects' at the end of this application note.

Contents

Introduction 1
Output Reports and Items 2
Boot Interface 6
Compound and Composite Devices 6
Project 1: Keyboard with LEDs 7
Project 2: USB Composite Device 10
Summary 14
Related Resources 14
  Additional Information: 14
About the Author 14
Appendix 1: *main.c* Code for Project 1 15
Appendix 2: Descriptor Tables for Project 1 19
Appendix 3: *main.*c Code for Project 2 22
Appendix 4: Descriptor Tables for Project 2 26
Appendix 5: Descriptor Tables for project 2 30
Document History 35

Introduction

The USB market is constantly growing as manufacturers develop new and more inventive devices. Simple keyboards with a few LEDs have now evolved into keyboards that have LCDs and dynamically changing graphical keys. Joysticks are no longer simple buttons and thumb sticks but also contain force feedback. To add any of these features to a device, information must be received from the host. By receiving this information, LEDs glow and joysticks rumble. To achieve this, an Output Report from the PC is required and it is easy to configure the USB report descriptor for a human interface device.

Many USB HID features are common in the industry and are included in many of the devices consumers purchase for their PC. A consumer might buy a keyboard that has a track pad or a track ball integrated in the same enclosure. To do so, a composite device is created to connect both devices through a single USB connection. When consumers enter their PC's BIOS on boot up, the keyboard and mouse require a special configuration called a Boot Interface.

This application note discusses these topics and how to implement them in USB applications. For more information

on USB HID fundamentals, refer to the application note AN57473 at www.cypress.com.

Output Reports and Items

HID enable consumers to easily interface with and control their PCs. Almost all HIDs send information to the host. Examples include keyboards and mice. However, the PC often needs to send information to the peripheral. Receiving information from the host is accomplished with an output report.

To receive this output report properly, output Items in the HID report descriptor must be configured properly. Configuring output Items is similar to the setup of Input items discussed in AN57473.The result of this transfer can take the form of a LED on a keyboard or the force feedback on a joystick. Both these devices require interaction from the host to provide them with information. Take the example of a keyboard. A keyboard has input items for the keys that the user presses and also has an output Item for the LEDs on the keyboard that display information (such as checking if the caps lock is enabled or not).

To understand the output Items in greater detail, let us look at a report descriptor for a 104-key keyboard. See Figure 1. The chosen organization of the report descriptor follows the required format for a boot interface, which is discussed in the Boot Interface section on page 5. However, the report descriptor can be organized in various ways and still function as a keyboard with LEDs.

Figure 1. Keyboard Report Descriptor

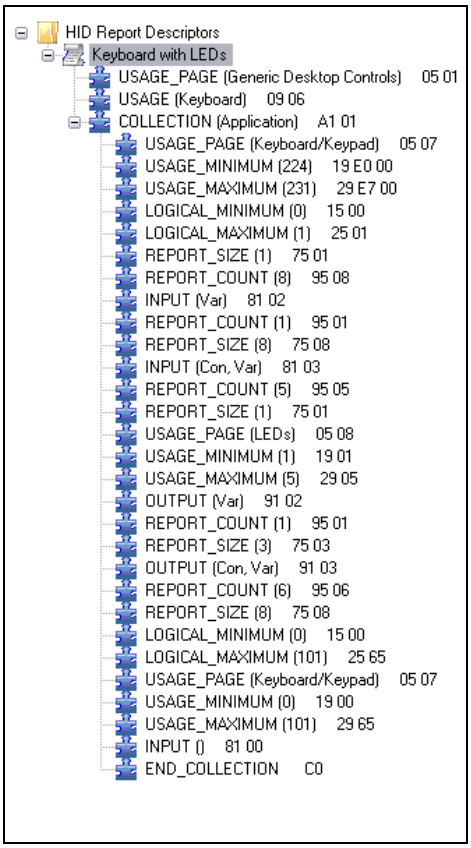
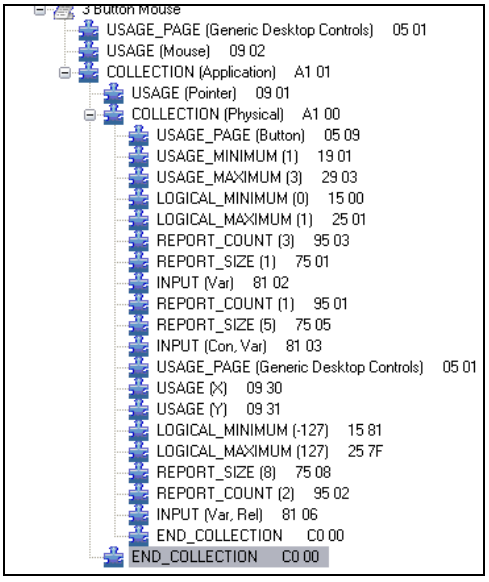


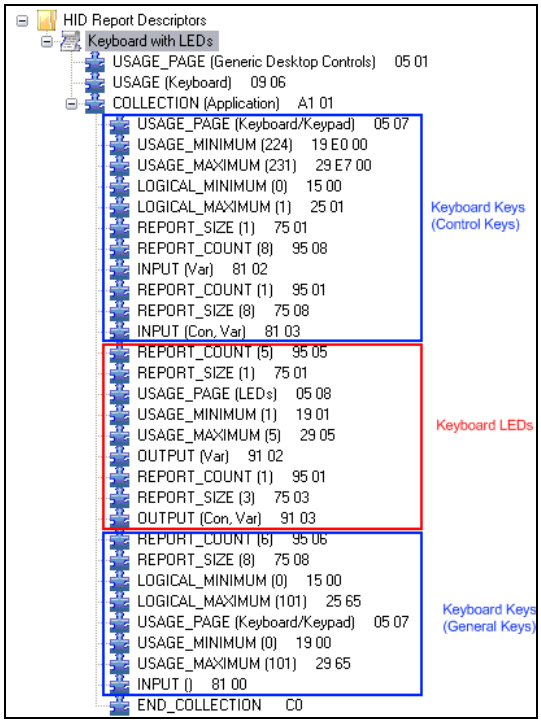
Figure 2. Mouse Report Descriptor from AN57473



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

difference is the addition of output Items. 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 keys on top, LEDs in the middle, and keyboard keys at the bottom (Input-Output-Input). While this organization may look odd, this format is organized 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 eight bytes that are sent to the host are organized with modifier keys first, followed by general keys as shown in Table 1. A modifier key is a key that modifies the function of a general key to perform an alternative function. Modifier keys include the shift, alt, and GUI keys on the keyboard. The GUI keys are the Win, Apple, or Meta keys on a keyboard. The name or icon is printed on the keyboard depends on the operating system for which the keyboard is designed. All other keys are considered general keys. The format in which this information is transferred to the host is shown in Table 1 on page 3.

Table 1. Keyboard Input Report Table

Byte	Information
0	Modifier Keys
1	Reserved (For OEM Use)
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

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 non-standard 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 remaining bytes in the configuration are the general keys. Six available bytes of information indicate that up to six key codes can be sent to the PC in a given transaction. This enables up to six simultaneous key presses. The order of the key codes in the array does not have significance. Determining the order and sorting out the key codes in the array is the responsibility of the host. The sorting is accomplished by comparing the newly received report to the previously received report.

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
7	Right GUI (Win/Apple/Meta)	1000 0000

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 show the Usage values for the modifier keys to range from E0-E7. However, the usage values are not sent as array data. The modifier keys are sent as variable data that means each individual bit in the 8-bit value corresponds to one of the modifier keys. The Usage Minimum/Maximums are then used to link the modifier key

information in the bit field to the proper Usage value with the Usage Minimum/Maximum.

To clear any confusion between Array and Variable information, refer to the HID Fundamentals 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 this 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.

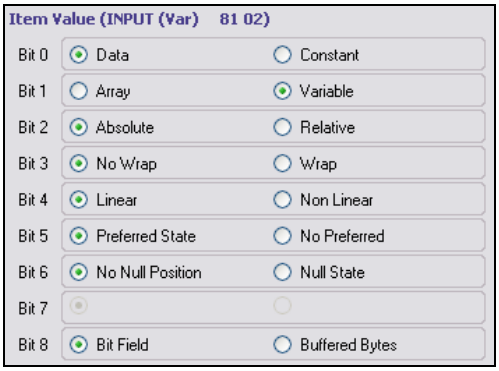
The same variable bit field organization used for the modifier keys is also applied to the LEDs. According to the USB HID Usage table, the usage values for the LEDs are provided in Table 3. LED information is an Absolute Item, which means that the Output Report must include the state of each LED with '0' meaning off and '1' meaning on. The LEDs are not relative items, which means that a '0' represents no change and '1' represents a change in state. When a user presses any of the corresponding buttons such as Caps Lock or Num Clock, the host manages 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 3. LED Indicator Index

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

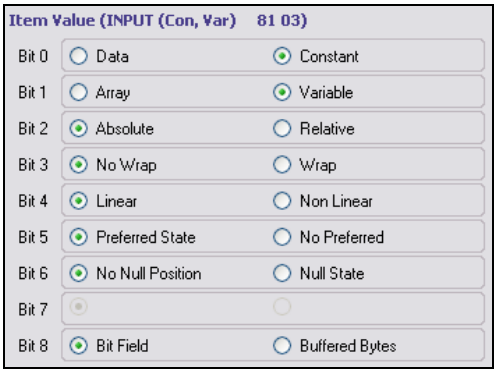
The first Input item when referencing the Report Descriptor from top down is for the modifier keys (Shift, Alt, and GUI). As stated earlier, these keys are configured in a Variable configuration to implement a bit field. The Input Item configuration is shown in Figure 4.

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. Declaring this byte as a constant is done by setting bit 0 to a value to '1' (see the following figure. This is done because each report is byte aligned.

Figure 5. Input Item for Reserving Second Byte

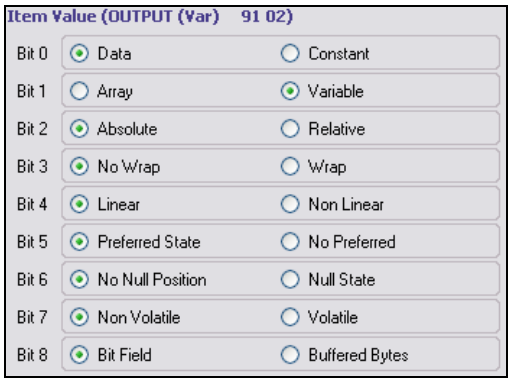


The next step in configuring the Descriptor is to configure the Output Item for the LEDs on the keyboard. Note two things here:

- Because the Item is an Output Item, note that the prefix value is '91' signifying an Output Item with one byte to follow.
- Bit 7 can be modified which is not the case with any Input Item. This is because this value only applies to Output and Feature Items. Bit 7 is set to '0' signifying that the bit is nonvolatile which means the device only alters the value with host interaction.

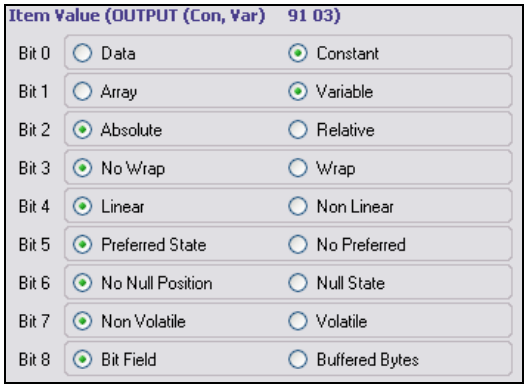
Note that bit 1 is set to be variable. This again configures the Item for a bit field configuration.

Figure 6. Output Item for LEDs



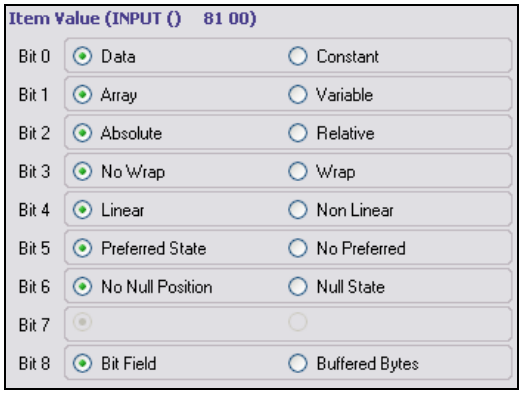
As shown in Figure 6, only five bits out of the total eight bits are used for LED information. Because all information is byte aligned, the remaining three bits are reserved by padding the bits with a value of '0' by setting those bits to remain constant. This is done by setting bit 0 in the Output Item to a value of '1'.

Figure 7. Output Item for Padding LEDs



The final Input Item to configure is for the general (non-modifier) keys. The HID specification requires these keys to be configured with an Array and Absolute configuration.

Figure 8. Input Item for General Keyboard Button



After you understand each block of the Report Descriptor for its functionality and how the information is interpreted, the next step is to study the Report Descriptor in detail and understand how each item works together to create a fully functional keyboard. A fully commented version of the keyboard report descriptor is given in the following figure.

Figure 9. Commented Keyboard Report Descriptor



The USB HID Specification contains a good example of how the 8-byte array is configured when a user is entering the ALT+CTRL+DEL button combination (see Table 4) and can also be found in Section 8.3 of the HID Specification document.

Table 4. ALT+CTRL+DEL Example

Step	Modifier Byte	Array Byte
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

Boot Interface

Occasionally, a PC user desires to enter a PC's BIOS upon boot up to edit it. To accomplish this, the USB HID Specification defines a boot protocol for both the mouse and keyboard. These protocols are predefined and the device is required to conform to the specifications to be a boot device. The boot protocol supports up to eight bytes of information. The BIOS ignores anything over eight bytes. Note that the BIOS does not actually read the report descriptor because of the predefined standard; the BIOS expects the information in a certain format. Because of this, a 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. This is done by a 'Change Protocol' command issued automatically after the HID class driver is loaded by the PC. The advantage in this method is 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 needs to make this choice. In this application note, the report descriptor used for the keyboard is the format that a boot device has, 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.

What officially defines an interface to be a Boot Interface is the subclass (*bInterfaceSubClass*). 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 to be a boot interface, the configuration type needs to be selected. As mentioned earlier in this section, the BIOS does not read the report descriptor. To know what 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* can be found in the Interface Descriptor. Because the host does not actually read the report descriptor, it is required that the array index value is the same as the usage value for each key. If a boot interface is not used, the array index value and the Usage values 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, shows a list of all the scan codes that are supported with a boot interface. This table also shows which keys are supported by various computer operating systems. If a boot interface is not required or desired, the Usage Maximum in Figure 1on page 2 can be changed to '255' to support the entire key code set.

Compound and Composite Devices

In USB applications, there are two types of combined device types. The two types are compound devices and composite devices. Consider a keyboard that has an integrated track pad 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 composite device configuration is required.

With a compound device, the keyboard and mouse with separate USB controllers are attached to an internal hub within the device. It is through that internal hub that the two devices are attached to the PC. In case of a composite device, the device has multiple interfaces that are controlled independent of each other. Unlike a compound device where the keyboard and track pad has its own device address, a composite device has only one device address. Each interface in the composite device has its own descriptors. A composite device also shares many of its resources with the various interfaces such as Endpoint0 (EP0). With the example mentioned above, the keyboard may use EP1 and EP2, while the mouse 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 in composite devices that contain of different device classes such as a Human Interface Device (HID) and Communication Class Device (CDC or USB UART). When adding multiple different interfaces to a composite device, the IAD will inform the host of which interfaces are associated with what function. See the following figure for an example of an IAD for a CDC device.

A CDC device consists of multiple interfaces. There are data interfaces and control interfaces. The IAD in the figure above shows how we can group a range of interface

numbers together to define their function. Appendix 5 shows a full example of a 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. There is also .inf file editing that is required to link the multiple interfaces to the CDC function. However, that will be covered in another application note or example project. The Report Descriptor provided in Appendix 5: Descriptor Tables for is meant to explain the concept of an IAD in greater detail.

Code 1. IAD Descriptor Example

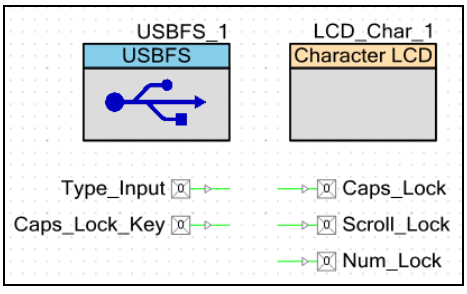
```
/*******************************************************************************
IAD Descriptor (CDC Communications Class Interface)
*******************************************************************************/
Descriptor size in bytes
/*  bLength                                 */ 0x08u,
Constant 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,
```

Project 1: Keyboard with LEDs

This project introduces the user to a HID device that consists of both an Input and Output transfer. The project acts as a keyboard that types messages and displays the status of num lock, caps lock, and scroll lock.

1. For this project, start by opening PSoC® Creator™ and create an empty project named 'Project_1_Keyboard'. After the project has been created and PSoC Creator is completely loaded, place the following components into the schematic entry page (*TopDesign.cysch*).
Character LCD
USBFS
(2) Digital Input Pins
- (3) Digital Output Pins

Figure 10. PSoC Creator Components for Keyboard

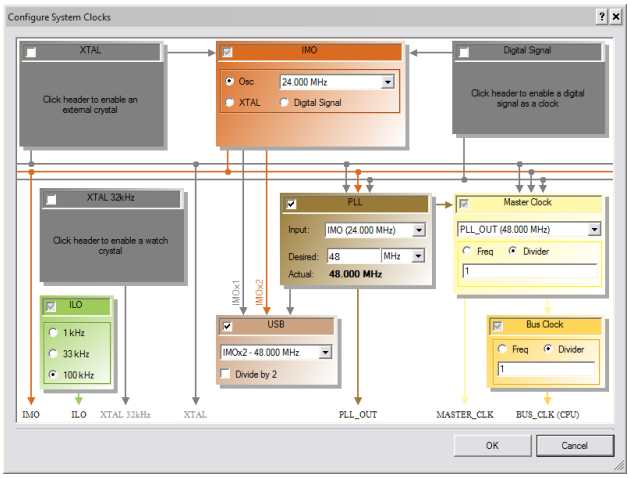


2. The pins require a few setting changes. For the Input Pins, open the pin configuration customizer and be sure to uncheck the 'HW Connection' box and change their drive mode to 'Resistive Pull Down' that is done by clicking on the 'General' tab in the pin customizer menu. For the Digital Output Pins, open the configuration menu, uncheck the 'HW Connection' box, and change their drive mode to 'Strong'.

3. After the components are placed, go to the Workspace Explorer window and double click on *MyFirstKeyboardHID.cydwr*. Click on the 'Clocks' tab and then double click on one of the clocks to open the GUI clock configuration window. Make the following changes to the clock as shown in Figure 11

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

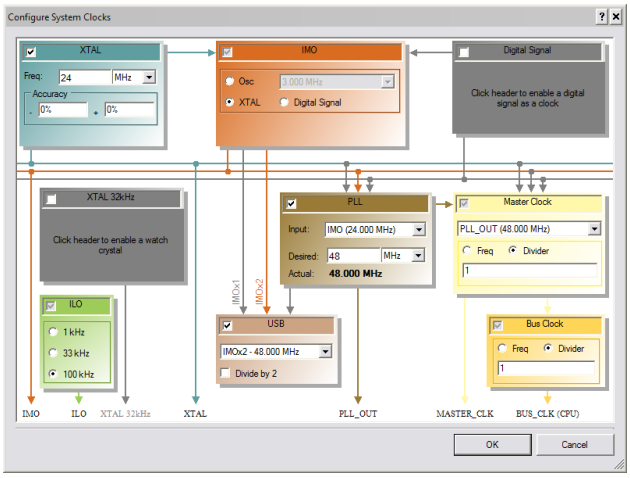
Figure 11. System Clock Configuration Window (PSoC 3)



3b. In the instance of PSoC 5, it is required to use an external 24MHz oscillator for USB operation. As a result, the configuration of the clock tree looks a little different than it does on PSoC 3 as shown in Figure 12 and listed below.

XTAL → Enable, Frequency: 24MHz
IMO → XTAL
ILO → 100 kHz
PLL → Input: IMO, Desired: 33 MHz
Master Clock → PLL Out
USB Clock → IMO x2

Figure 12. System Clock Configuration Window (PSoC 5)



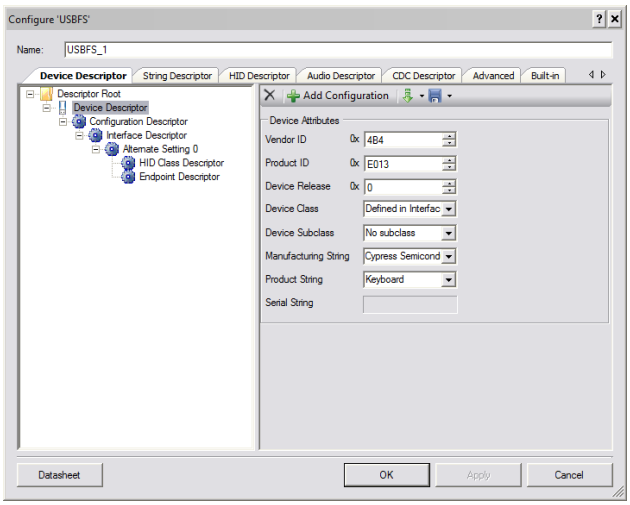
4. The next step is to configure the pin configuration for the project. To do so, click on the 'Pins' tab located in *Project 1 – Keyboard.cydwr* and change the pins to resemble Figure 12 on page 8. Remember the USB pins are always located at P15[6] and P15[7] but the other pins can be moved if desired. The following pinout is configured to work with the project provided with this application note on the DVK board.

Figure 12. 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]	☑

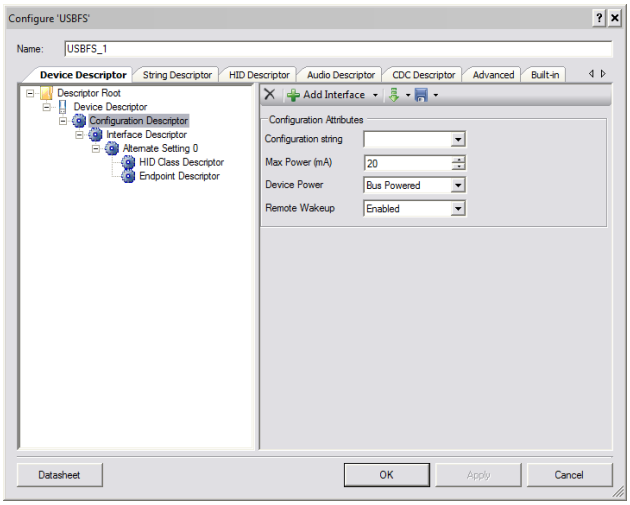
5. Next, the USB component needs to be configured using the USB Wizard. This process follows the same steps as in AN57473 but with the addition of an OUT EP. Double click on the USBFS component to open the Configuration Wizard. Configure the Vender ID (VID) and Product ID (PID) to be 0x4B4 and 0xE013. Fill in the Manufacture and Product strings too. The strings shown in Figure 13 can be used or other strings can be chosen. The values for the VID and PID can also be changed. Any value can be used for demonstration purposes but if entering production or distributing the example project, your own VID must be assigned from the USB Implementers Forum.

Figure 13. USB Device Descriptor Setup



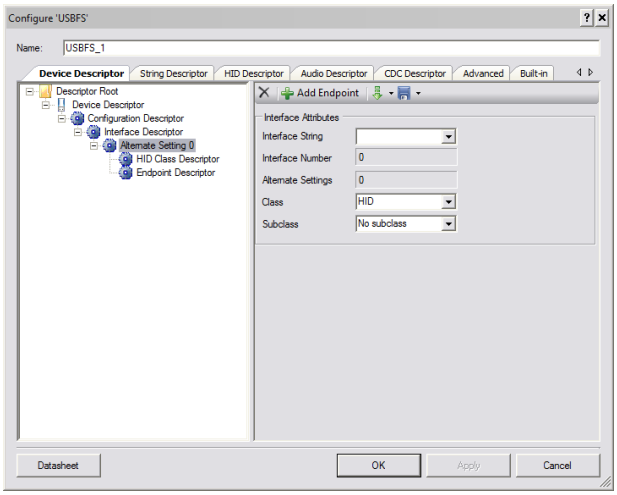
6. Next, select 'Configuration Descriptor' and the window changes as shown in Figure 14.Because the project is bus powered, limit the maximum current that can be supplied to the device. It is important to specify a value that is appropriate for the device. For this application, 20 mA is more than sufficient.

Figure 14. USB Device Descriptor Setup



7. Next, configure the interface descriptor. To do this, click on 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 a HID. The dialog is shown in Figure 18 Note that 'Subclass' is set for 'No SubClass'. If a Boot Interface is required, the user must change the subclass to 'Boot Interface'. This causes another drop down menu to appear where the user chooses between a mouse and a keyboard for the boot interface.

Figure 15. USB Interface Descriptor Setup



8. A HID report descriptor must be associated with the interface. To do so, the HID Report Descriptor must be created. This is done by clicking on the HID descriptor tab in the dialog. The following window (see Figure 16) opens. The format for the HID report descriptor is shown in Figure 1 on page 2. Make the required additions to the Report Descriptor so it resembles Figure 1 on page 2. A text version of the Report Descriptor is provided in Appendix 2. Upon completion of the report descriptor, return to the HID Class Descriptor menu shown in Figure 17. Set the HID Report to the name of the HID Report that was just created.

Figure 16. USB HID Report Descriptor Setup

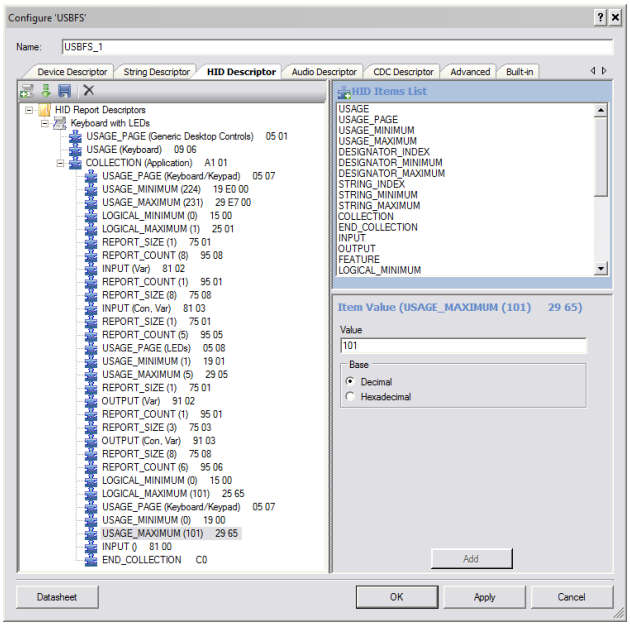
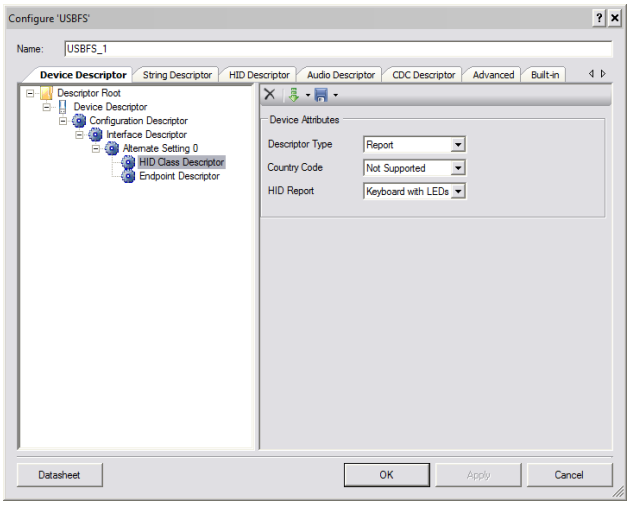
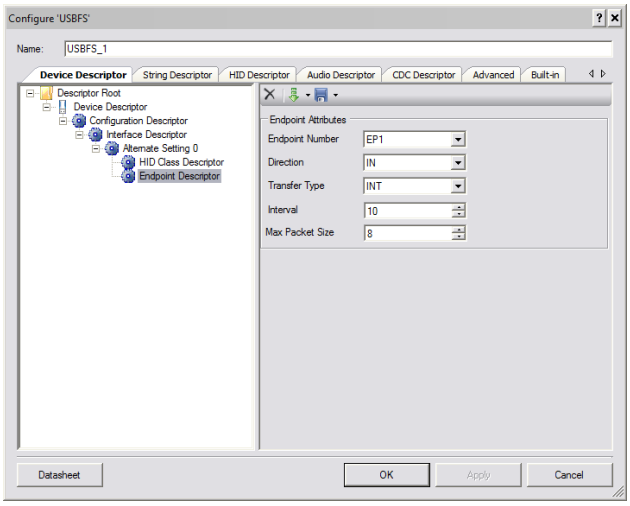


Figure 17. USB HID Class Descriptor Setup



9. The final step of configuration is to set the Endpoints. As mentioned earlier, LED data will be sent through EP0 via a SET_REPORT. As a result, only an IN endpoint is required. Select the 'Endpoint Descriptor' and under 'Endpoint Attributes', select the Endpoint number to be EP1, the direction to be 'IN', and the transfer type to be 'INT'. Also set the interval to '10' and the maximum packet size to '8'. This is shown in Figure 18.

Figure 18. USB Endpoint Setup (IN)



10. Locate the *main.c* file in the Workspace Explorer and open the file to edit it. Delete the existing code in *main.c* and place the C code that is shown in Appendix 1 into the file. After the code has been placed in the project, Build/Compile the project and program it into the PSoC device.

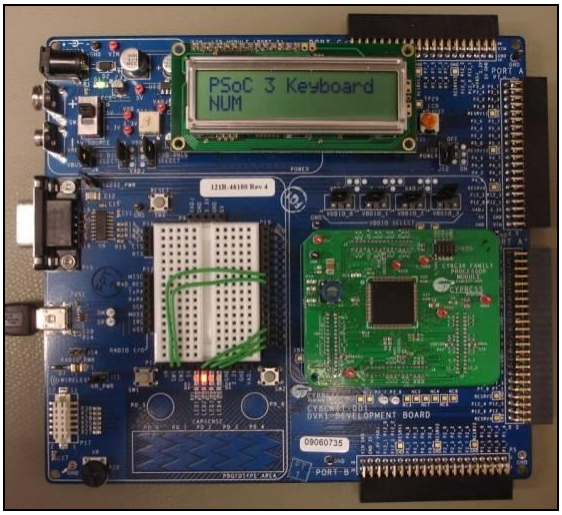
11. Use the jumper wires to make the following connections on the PSoC CY8CKIT-001 as shown in Figure 19. If you decide to change the pin

configuration from Figure 12 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

12. After the device is programmed, connect a USB cable from the PC to the DVK. After the device has enumerated, press the Caps, Num, 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 PC and displayed in the text editor. Feel free to edit the string in the *main.c* file to type anything. The Descriptor tables for this project are shown in Appendix 2.

Figure 19. Project 1 on CY8CKIT-001



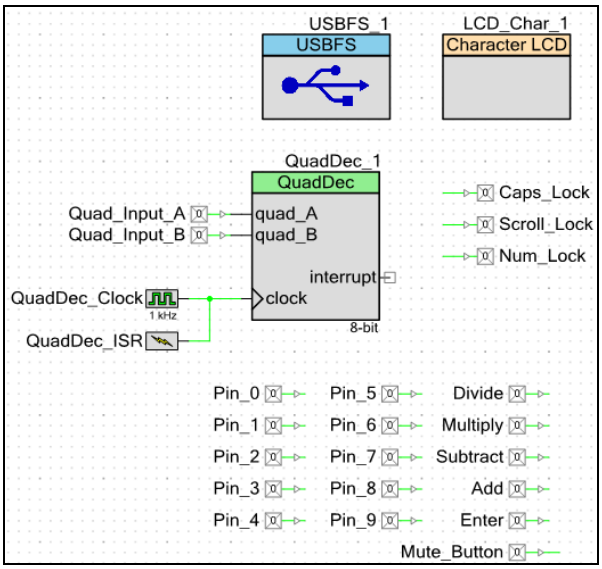
Project 2: USB Composite Device

Now that both Input and Output items are understood, other aspects of HID can be explored. As stated earlier, a composite device gives users the ability to create multiple device interfaces with one PSoC device and control each interface independently. In this example, a 10-key number pad with simple mathematical keys is used with the addition of a device that falls under the Consumer Devices class of HID. This device also provides capability to add a volume control dial using a quadrature rotary encoder that enables users to both increase and decrease the volume by turning the dial and muting the volume by pushing a button.

1. For this project, start by opening PSoC Creator and create an empty project named 'Project_2_Composite'. After the project has been created, place the following components in the schematic entry page (*TopDesign.cysch*) as shown in Figure 20.

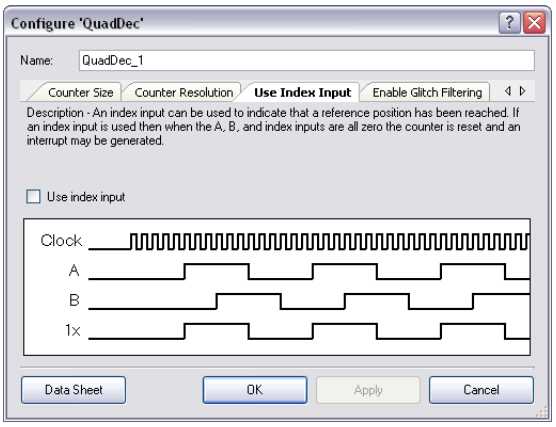
- Character LCD
- USBFS
- Quadrature Decoder
- Clock
- ISR
- (18) Digital Input Pins
- (3) Digital Output Pins

Figure 20. PSoC Creator Components for Composite Device



The only change that must be made to the QuadDec component is removal of the Index Input that is enabled by default. To remove this connection, open the configuration customizer, select the 'Use Index Input', and uncheck the box. This customizer option is shown in the following figure.

Figure 21. 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 in the project are configured for 'Resistive Pull Down'. It is better to configure the pins to 'Pulled Down' for 'Power-On Reset' state. To do this, click on the 'Reset' tab in the customizer for each pin and change the Power-On Reset selection from the drop down menu.

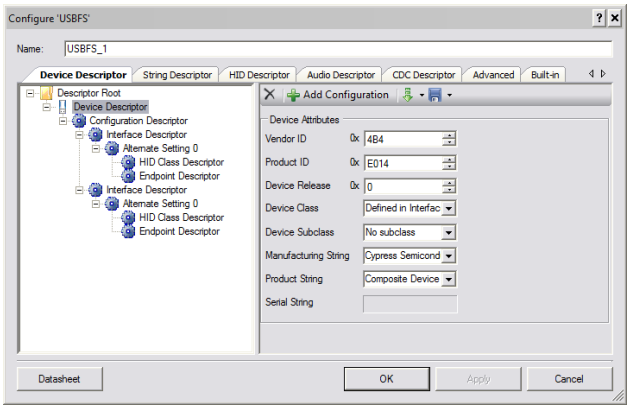
3. The clocking selections used in this project and Project 1 are identical. Referencing Figure 11, change the clock settings as needed. After the clocks are configured, enter the Pins menu and adjust the pin locations as shown in Figure 22.

Figure 22. 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]	☑

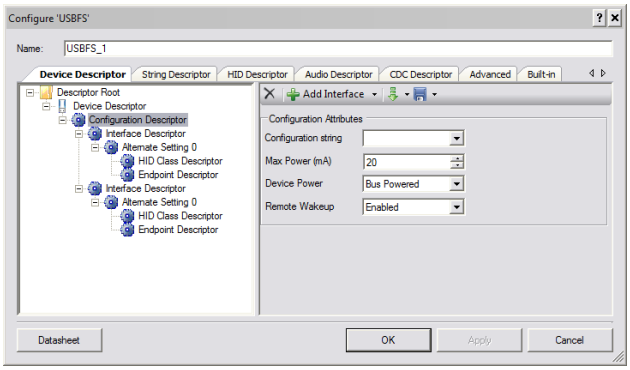
4. The next step is to configure the USB component. Because this is an addition to the keyboard from Project 1, most of the USB settings remain the same. Start by changing the PID and the Manufacture/Product String. The attached example project uses a PID of 0xE014. Because this project is a Composite Device, another interface must be added.

Figure 23. USB Device Descriptor Setup



5. This is done selecting the Configuration Descriptor. Once selected, a button to 'Add Interface' is seen. Click on the button to add the second Interface. This is shown in Figure 23 on page 11. Note that the power settings are the same as they are in Project 1.

Figure 23. USB Configuration Descriptor Setup



6. Next, select the 'Alternative Setting' for each interface and configure them as shown in Figure 24 and Figure 25. 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 24. USB Interface Descriptor Setup (Keyboard)

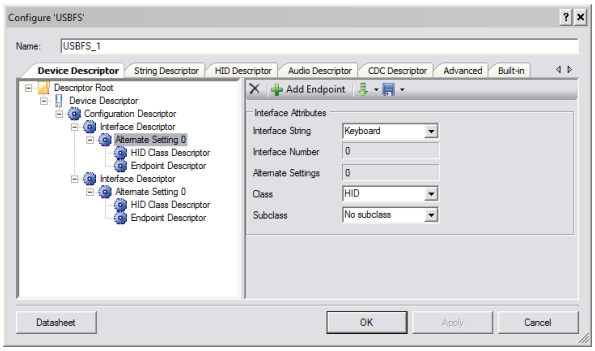
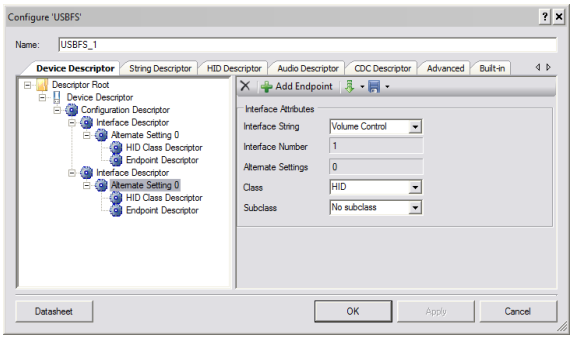


Figure 25. USB Interface Descriptor Setup (Volume)



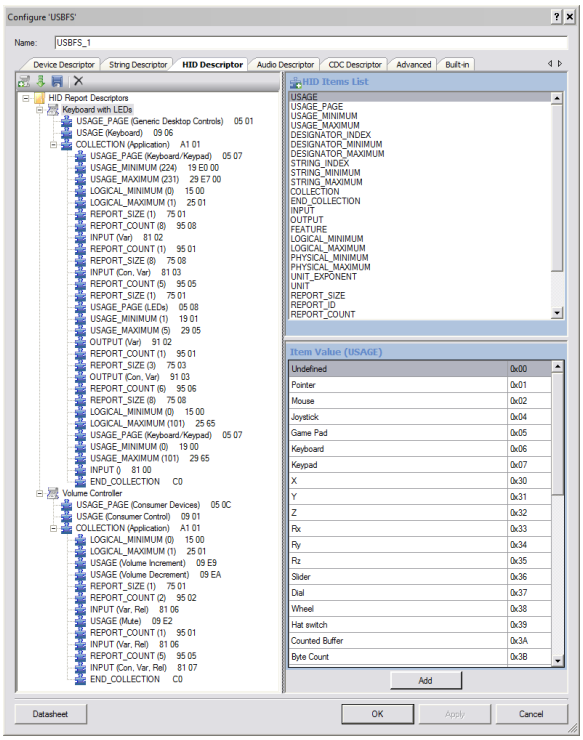
7. Because there are two independent devices, each device needs its own Report Descriptor. The Report Descriptor for the keyboard remains the same as in Project 1. A Report Descriptor for the volume controller must be added. Refer to Figure 26 and Figure 27 to see how to set up the Descriptor. Note that only controls for Volume Up, Volume Down, and Mute are implemented in this instance. A text version of the Report Descriptor can also be found in Appendix 4.

Figure 26. Volume Control Report Descriptor

8.

```
Volume Controller
    USAGE_PAGE (Consumer Devices)    05 0C
    USAGE (Consumer Control)    09 01
    COLLECTION (Application)    A1 01
        LOGICAL_MINIMUM (0)    15 00
        LOGICAL_MAXIMUM (1)    25 01
        USAGE (Volume Increment)    09 E9
        USAGE (Volume Decrement)    09 EA
        REPORT_SIZE (1)    75 01
        REPORT_COUNT (2)    95 02
        INPUT (Var, Rel)    81 06
        USAGE (Mute)    09 E2
        REPORT_COUNT (1)    95 01
        INPUT (Var, Rel)    81 06
        REPORT_COUNT (5)    95 05
        INPUT (Con, Var, Rel)    81 07
        END_COLLECTION    C0
```

Figure 27. USB HID Report Descriptor Setup



9. After the Report Descriptor for each device has been created, the next step is to assign the proper Report Descriptor to each interface as shown in Figure 28 and Figure 29.

Figure 28. USB HID Class Descriptor Setup (Keyboard)

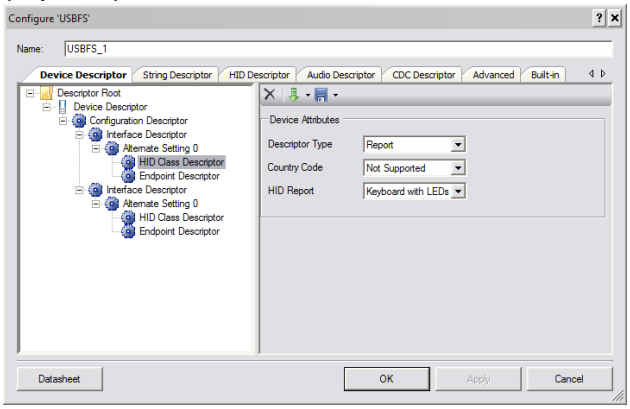
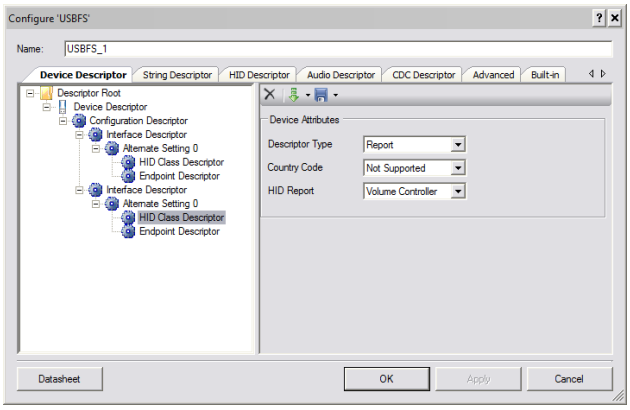


Figure 29. USB HID Class Descriptor Setup (Volume)



10. The final step in the configuration of the USB 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 30 and Figure 31 show the endpoint configurations.

Figure 30. USB Endpoint Setup (Keyboard IN)

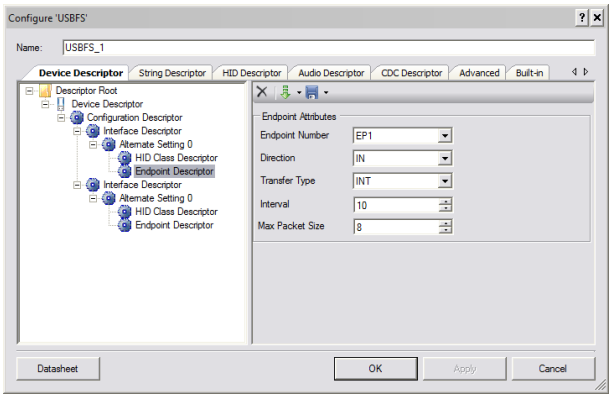
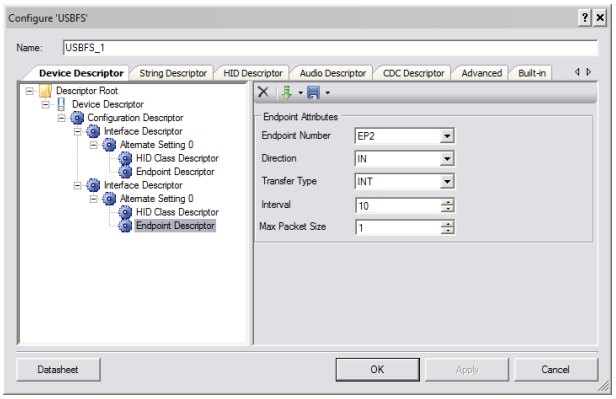


Figure 31. USB Endpoint Setup (Volume Control IN)



11. Locate the *main.c* file in the Workspace Explorer and open the file to edit it. Delete the existing code in *main.c* and place the C code shown in Appendix 3 in the file.

In addition, the following code needs to be placed in the QuadDec_ISR.c under the custom user includes/defines and the customer ISR code sections. After the code is placed in the project, the next step is to Build/Compile the project. The Descriptor tables for this project are located in Appendix 4 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 schematic for the connections of the encoder and the key pad are shown in Figure 32. While it can be seen in Figure 33 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 or by using multiple push buttons. With the encoder and switches connected to the PSoC, program and reset the device.

Figure 32. Schematic of Rotary Encoder

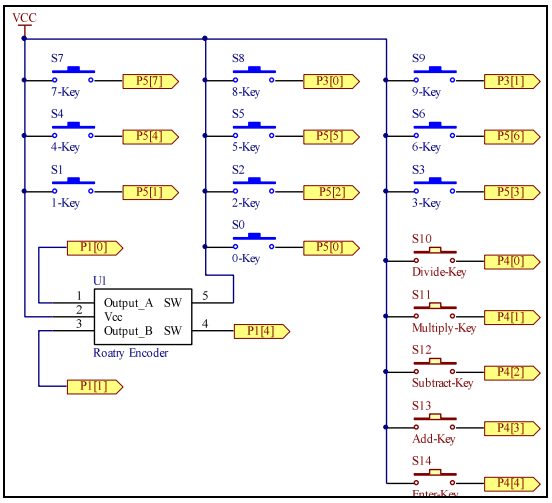


Figure 33. Photograph of Project on a PSoC DVK



During the enumeration process, you see the 'Found New Hardware' bubble pop up in the Windows taskbar informing you that a composite device is detected. This is then 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.

Summary

This application note enables the user to feel comfortable enough with USB and HID to create almost any basic USB HID application. While there are many more features and capabilities of USB which are detailed in the USB Specification, this application note describes the most central elements of any USB design. To build upon 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 only covered a small number of the vast possibilities and potneital ways to process HID data. One of the many benefits to USB in general is the seemingly endless number of ways to handle and process data. For then can be covered in a single application note.

Related Resources

Application Notes:

- **AN57473** - PSoC® 3 / PSoC 5 USB HID Fundamentals with Mouse and Joystick
- **AN58726** - PSoC® 3 / PSoC 5 USB HID Intermediate (with Keyboard and Composite Device)
- **AN56377** - PSoC® 3 / PSoC 5 USB Transfer Types
- **AN023** - USB Compliance Testing Overview
- **AN52970** - Windows Hardware Quality Lab (WHQL) Signing Procedure
- **AN14557** – Introduction to CYUSB.dll Based Application Development Using C#
- **AN61744** – Introduction to CYAPI.lib Based Application Development Using VC++

Additional Information:

Cypress PSoC USB Landing Page

Official USB 2.0 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.
Contact:	rlrm@cypress.com

Appendix 1: *main.c* Code for Project 1

```
#include <device.h>

#define LSHIFT 0x02
#define ENTER 0x28
#define CAPS 0x39

void In_EP (void);
void Out_EP (void);
void CapsLock (void);

/* External variable where OUT Report data is stored */
extern uint8
USBFS_1_DEVICE0_CONFIGURATION0_INTERFACE0_ALTERNATE0_HID_OUT_BUF[USBFS_1_DEVICE0_CONFIGURATIO
N0_INTERFACE0_ALTERNATE0_HID_OUT_BUF_SIZE];

/* Creates a Scan Code Look Up Table for the various ASCII values */
const uint8 aASCII_ToScanCode[] = {0x2C, 0x1E, 0x34, 0x20, 0x21, 0x22, 0x24, 0x34, 0x26,
0x27, 0x25, 0x2E, 0x36, 0x2D, 0x37, 0x38, 0x27, 0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x24,
0x25, 0x26, 0x33, 0x33, 0x36, 0x2E, 0x37, 0x38, 0x1F, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15 ,0x16, 0x17, 0x18,
0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x2F, 0x31, 0x30, 0x23, 0x2D, 0x35, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15 ,0x16,
0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x2F, 0x31, 0x30, 0x35, 0x4C};

/* Array of Keycode information to send to PC */
static unsigned char Keyboard_Data[8] = {0, 0, 0, 0, 0, 0, 0, 0};
/* Array of information pertaining to LEDS */
static unsigned char Out_Data[1] = {0};
/* String that will be printed when Type_Input (SW2) is pressed */
static const char StringStorage[] = {"Cypress PSoC USB HID: Stay Connected My Friends"};
static uint8 i;
static uint8 key;


void main()
{
   CYGlobalIntEnable;
   LCD_Char_1_Start();
   LCD_Char_1_Position(0,0);
   LCD_Char_1_PrintString("PSoC 3 Keyboard");
   /*Start USBFS Operation and Device 0 and 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*/
   USBFS_1_LoadInEP(1, Keyboard_Data, 8);


   for(;;)
   {
      /*Function to check for Caps Lock button press on DVK*/
      CapsLock();

      /*Checks for ACK from host*/
```

```
        if(USBFS_1_bGetEPAckState(1))
        {
           /*Function to Send Data to PC*/
           In_EP();

          /*Function to receive data from PC*/
          Out_EP();
        }
    }
}


void In_EP (void)
{
    /*Checks for Type_Input(SW1) to be pressed. This ensures that the text is only typed when
    button is pressed*/
    if(Type_Input_Read() == 0)
    {
        /*Waits for button to be released. This is a de-bounce implementation*/
        while(Type_Input_Read() == 0);
        /*Checks to be sure a null is not present*/
        while(StringStorage[i] != 0)
        {
            /*Ensures that all ASCII values in the string are in range*/
            if((StringStorage[i] >= 0x20) && (StringStorage[i] <= 0x7F))
            {
                /*Removes 0x20 offset to have key presses begin at 0. This is not required but
                used to make the example easier to understand.*/
                key = StringStorage[i] - 0x20;

                if(((key >= 1) && (key <= 6)) || ((key >= 8) && (key <=11)) || (key == 26) ||
                (key == 28) || ((key >= 30) && (key <=58)) || (key == 62) || (key == 63) ||
                ((key >= 91) && (key <= 94)))
                {
                    /*Applies Shift key is any keys that require the shift button is pressed.
                    The buttons include '!', '@', 'B', etc.*/
                    Keyboard_Data[0] = LSHIFT;
                }

                else
                {
                    /*If the shift is not required, then the modifier byte remains set to 0x00*/
                    Keyboard_Data[0] = 0x00;
                }

                /*Loads the converted Scan Code using LUT into the fist Key Code array
                location*/
                Keyboard_Data[2] = aASCII_ToScanCode[key];
            }

            /*Loads EP1 for a IN transfer to PC*/
            USBFS_1_LoadInEP(1, Keyboard_Data, 8);
            /*Waits for ACK from PC*/
            while(!USBFS_1_bGetEPAckState(1));
            /*Resets modifier byte to 0x00*/
            Keyboard_Data[0] = 0x00;
            /*Resets keycode 0 to 0x00*/
            Keyboard_Data[2] = 0x00;
            /*Loads EP1 for a IN transfer to PC. This simulates the buttons being released.*/
            USBFS_1_LoadInEP(1, Keyboard_Data, 8);
```

```
            /*Waits for ACK from PC*/
            while(!USBFS_1_bGetEPAckState(1));
            /*Advances to the next ASCII Character*/
            i++;
        }

        /*Loads 0x28 (Enter/Return) into keycode 0 to move to next line*/
        Keyboard_Data[2] = ENTER;
        /*Loads EP1 for a IN transfer to PC*/
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        /*Waits for ACK from PC*/
        while(!USBFS_1_bGetEPAckState(1));
        /*Resets keycode 0 to 0x00*/
        Keyboard_Data[2] = 0x00;
        /*Loads EP1 for a IN transfer to PC. This simulates the buttons being released.*/
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        /*Waits for ACK from PC*/
        while(!USBFS_1_bGetEPAckState(1));
        /*Resets string position counter to 0 to type string again*/
        i=0;
    }
}

void Out_EP (void)
{
    /*Reads the OUT Report data */
    Out_Data[0] = USBFS_1_DEVICE0_CONFIGURATION0_INTERFACE0_ALTERNATE0_HID_OUT_BUF[0];


    /*If the Num Lock is enabled, display on LCD and LED*/
    if ((Out_Data[0] & 0x01)!= 0)
    {
        LCD_Char_1_Position(1,0);
        LCD_Char_1_PrintString("NUM");
        Keyboard_Data[2] = 0x00;
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        Num_Lock_Write(1);
    }

    /*If the Num Lock not is enabled, clear the LCD and LED*/
    else
    {
        LCD_Char_1_Position(1,0);
        LCD_Char_1_PrintString("   ");
        Num_Lock_Write(0);
    }

    /*If the Caps Lock is enabled, display on LCD and LED*/
    if((Out_Data[0] & 0x02)!= 0)
    {
        LCD_Char_1_Position(1,6);
        LCD_Char_1_PrintString("CAPS");
        Keyboard_Data[2] = 0x00;
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        Caps_Lock_Write(1);
    }

    /*If the Caps Lock not is enabled, clear the LCD and LED*/
    else
    {
```

```
        LCD_Char_1_Position(1,6);
        LCD_Char_1_PrintString("    ");
        Caps_Lock_Write(0);
    }


    /*If the Scroll Lock is enabled, display on LCD and LED*/
    if((Out_Data[0] & 0x04)!= 0)
    {
        LCD_Char_1_Position(1,13);
        LCD_Char_1_PrintString("SCL");
        Keyboard_Data[2] = 0x00;
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        Scroll_Lock_Write(1);
    }

    /*If the Scroll Lock not is enabled, clear the LCD and LED*/
    else
    {
        LCD_Char_1_Position(1,13);
        LCD_Char_1_PrintString("   ");
        Scroll_Lock_Write(0);
    }
}

void CapsLock (void)
{
    /*Checks if Caps Lock (SW2) has been pressed. If so, loads the Caps Lock scan code into
    the key code array*/
    if(Caps_Lock_Key_Read() == 0)
    {
        while(Caps_Lock_Key_Read() == 0);
        Keyboard_Data[2] = CAPS;
        USBFS_1_LoadInEP(1, Keyboard_Data, 8);
        while(!USBFS_1_bGetConfiguration());
    }
}

/* End of File */
```

Appendix 2: 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,
```

Appendix 3: *main.c* Code for Project 2

```
#include <device.h>

void Scan_Keypad (void);
void Keyboard_LEDs (void);
void Volume_Control(void);

/* External variable where OUT Report data is stored */
extern uint8
USBFS_1_DEVICE0_CONFIGURATION0_INTERFACE0_ALTERNATE0_HID_OUT_BUF[USBFS_1_DEVICE0_CONFIGURATIO
N0_INTERFACE0_ALTERNATE0_HID_OUT_BUF_SIZE];
uint8 USB_IN_Volume_Data[1] = {0};
uint8 USB_IN_Keyboard_Data[8] = {0, 0, 0, 0, 0, 0, 0, 0};
uint8 USB_Out_Keyboard_Data[1] = {0};
int8 Volume_Data, Old_Volume_Data;

void main()
{
       CYGlobalIntEnable;
       QuadDec_1_Start();
       QuadDec_Clock_Start();
       QuadDec_ISR_Start();
       LCD_Char_1_Start();
       LCD_Char_1_Position(0,0);
       LCD_Char_1_PrintString("PSoC 3 Composite");
       /*Start USBFS Operation and Device 0 and with 5V operation*/
       USBFS_1_Start(0, USBFS_1_DWR_VDDD_OPERATION);
       /*Waits for USB to enumerate*/
       while(!USBFS_1_bGetConfiguration());
       /*Begins USB Traffic for Keyboard*/
       USBFS_1_LoadInEP(1, USB_IN_Keyboard_Data, 8);
       /*Begins USB Traffic for Volume Controller*/
       USBFS_1_LoadInEP(2, USB_IN_Volume_Data, 1);

    for(;;)
    {
       /*Checks for ACK from host for Keyboard Traffic*/
              if(USBFS_1_bGetEPAckState(1) != 0)
              {
                     /*Function to scan keypad board */
                     Scan_Keypad();

                     /*Function to receive LED data from PC out EP2*/
                     Keyboard_LEDs();
              }

              /*Checks for ACK from host for Volume Control Traffic*/
              if(USBFS_1_bGetEPAckState(2) != 0)
              {
                     /*Function to control volume based on encoder status */
                     Volume_Control();
                     /*Loads EP3 to transfer data to PC */
                     USBFS_1_LoadInEP(2, USB_IN_Volume_Data, 1);
              }
    }
}

void Scan_Keypad (void)
{
```

```
	/* Each button is scanned and if asserted, loads the proper scan code into
USB_IN_Keyboard_Data */
	if (Pin_0_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x27; /* 0 key */
		while(Pin_0_Read() != 0);
	}

	else if (Pin_1_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x1E; /* 1 key */
		while(Pin_1_Read() != 0);
	}

	else if (Pin_2_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x1F; /* 2 key */
		while(Pin_2_Read() != 0);
	}

	else if (Pin_3_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x20; /* 3 key */
		while(Pin_3_Read() != 0);
	}

	else if (Pin_4_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x21; /* 4 key */
		while(Pin_4_Read() != 0);
	}

	else if (Pin_5_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x22; /* 5 key */
		while(Pin_5_Read() != 0);
	}

	else if (Pin_6_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x23; /* 6 key */
		while(Pin_6_Read() != 0);
	}

	else if (Pin_7_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x24; /* 7 key */
		while(Pin_7_Read() != 0);
	}

	else if (Pin_8_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x25; /* 8 key */
		while(Pin_8_Read() != 0);
	}

	else if (Pin_9_Read() != 0)
	{
		USB_IN_Keyboard_Data[2] = 0x26; /* 9 key */
		while(Pin_9_Read() != 0);
```

```
        }

        else if (Divide_Read() != 0)
        {
                USB_IN_Keyboard_Data[2] = 0x54; /* '/' key */
                while(Divide_Read() != 0);
        }

        else if (Multiply_Read() != 0)
        {
                USB_IN_Keyboard_Data[2] = 0x55; /* '*' key */
                while(Multiply_Read() != 0);
        }
        else if (Subtract_Read() != 0)
        {
                USB_IN_Keyboard_Data[2] = 0x56; /* '-' key */
                while(Subtract_Read() != 0);
        }

        else if (Add_Read() != 0)
        {
                USB_IN_Keyboard_Data[2] = 0x57; /* '+' key */
                while(Add_Read() != 0);
        }

        else if (Enter_Read() != 0)
        {
                USB_IN_Keyboard_Data[2] = 0x58; /* 'Return' key */
                while(Enter_Read() != 0);
        }

        else
        {
                USB_IN_Keyboard_Data[2] = 0x00;
        }

        /*Loads EP1 to transfer data to PC */
        USBFS_1_LoadInEP(1, USB_IN_Keyboard_Data, 8);
}

void Keyboard_LEDs (void)
{
        /*Reads the OUT Report data */
        USB_Out_Keyboard_Data[0] =
USBFS_1_DEVICE0_CONFIGURATION0_INTERFACE0_ALTERNATE0_HID_OUT_BUF[0];

        /*If the Num Lock is enabled, display on LCD and LED*/
        if ((USB_Out_Keyboard_Data[0] & 0x01)!= 0)
        {
                LCD_Char_1_Position(1,0);
                LCD_Char_1_PrintString("NUM");
                Num_Lock_Write(1);
        }

        /*If the Num Lock not is enabled, clear the LCD and LED*/
        else
        {
                LCD_Char_1_Position(1,0);
                LCD_Char_1_PrintString("   ");
                Num_Lock_Write(0);
```

```
        }

        /*If the Caps Lock is enabled, display on LCD and LED*/
        if((USB_Out_Keyboard_Data[0] & 0x02)!= 0)
        {
                LCD_Char_1_Position(1,6);
                LCD_Char_1_PrintString("CAPS");
                Caps_Lock_Write(1);
        }
        /*If the Caps Lock not is enabled, clear the LCD and LED*/
        else
        {
                LCD_Char_1_Position(1,6);
                LCD_Char_1_PrintString("    ");
                Caps_Lock_Write(0);
        }

        /*If the Scroll Lock is enabled, display on LCD and LED*/

        if((USB_Out_Keyboard_Data[0] & 0x04)!= 0)
        {
                LCD_Char_1_Position(1,13);
                LCD_Char_1_PrintString("SCL");
                Scroll_Lock_Write(1);
        }

        /*If the Scroll Lock not is enabled, clear the LCD and LED*/
        else
        {
                LCD_Char_1_Position(1,13);
                LCD_Char_1_PrintString("   ");
                Scroll_Lock_Write(0);
        }
}

void Volume_Control (void)
{

        /*Checks to see if Mute button was pressed */
        if(Mute_Button_Read() != 0)
        {
                while(Mute_Button_Read() != 0);
                USB_IN_Volume_Data[0] |= 0x04;
        }

        /*Checks to see if encoder has moved in the positive direction*/
        else if(Volume_Data > Old_Volume_Data)
        /*Stores value to increase volume*/
        USB_IN_Volume_Data[0] |= 0x01;
        /*Checks to see if encoder has moved in the negative direction*/
        else if(Volume_Data < Old_Volume_Data)
        /*Stores value to decrease volume*/
        USB_IN_Volume_Data[0] |= 0x02;
        else
        USB_IN_Volume_Data[0] &= 0x00;

        /*Updates comparison variable with current data*/
        Old_Volume_Data = Volume_Data;
}
/* End of File */
```

Appendix 4: 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,
```

Appendix 5: Descriptor Tables for project 2

```
/*****************************************************************
 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_DESCRIPTORS = {&USBUART_HIDREPORT_DESCRIPTOR1[0]};
//#define USBUART_HIDREPORT_DESCRIPTOR1 USBUART_CODE USBUART_HIDREPORT_DESCRIPTORS

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};
/*******************************************************************/
```

Document History

Document Title: PSoC® 3 / PSoC 5 USB HID Intermediate (with Keyboard and Composite Device) – AN58726

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.

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

PSoC is a registered trademark of Cypress Semiconductor Corp. PSoC Creator is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2010-2011. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.