Application Note

USB HID Keyboard Using the Z8F6482 MCU

AN041601-0418

Abstract

This application note demonstrates the creation of a USB HID keyboard application using the Z8F6482 Development Kit.

Note: The source code file associated with this application note, AN0416-SC01.zip, is available free for download from the Zilog website. The source code in these files has been tested with ZDS II version 5.2.2 for the Encore! XP Series of MCUs. Subsequent releases of ZDS II may require you to modify the code supplied with this application note.

Features

The Z8F6482 MCU's USB module provides USB full-speed device functionality with eight USB endpoints. Features include:

- Full-speed (12 Mbps) USB device
- IN endpoint 0 and OUT endpoint 0 control endpoints
- IN endpoints 1–3 and OUT endpoints 1–3 capable of bulk and interrupt transfers
- USB Suspend, host-initiated Resume, and device-initiated Resume (remote wake-up)
- USB clock of 48 MHz from internal PLL or external clock source
- 512 bytes of dedicated USB endpoint buffer memory; each endpoint buffer memory can be configured as 8, 16, 32, or 64 bytes
- Integrated full-speed USB PHY with integrated pull-up resistor
- Support for two DMA channels

Discussion

This application note modifies the Z8F6482 MCU USB Peripheral Application Note (AN0411) to create an HID keyboard application. To learn more about setting up and enabling use of the Z8F6482 MCU's USB peripheral, refer to AN0411.

Endpoints

This application uses four USB endpoints – IN0, OUT0, IN3, and OUT3. Endpoint0 is used for standard USB request processing, while Endpoint3 is used to send/receive USB key code information. Figure 1 illustrates the endpoint buffer memory arrangement set in this application note.

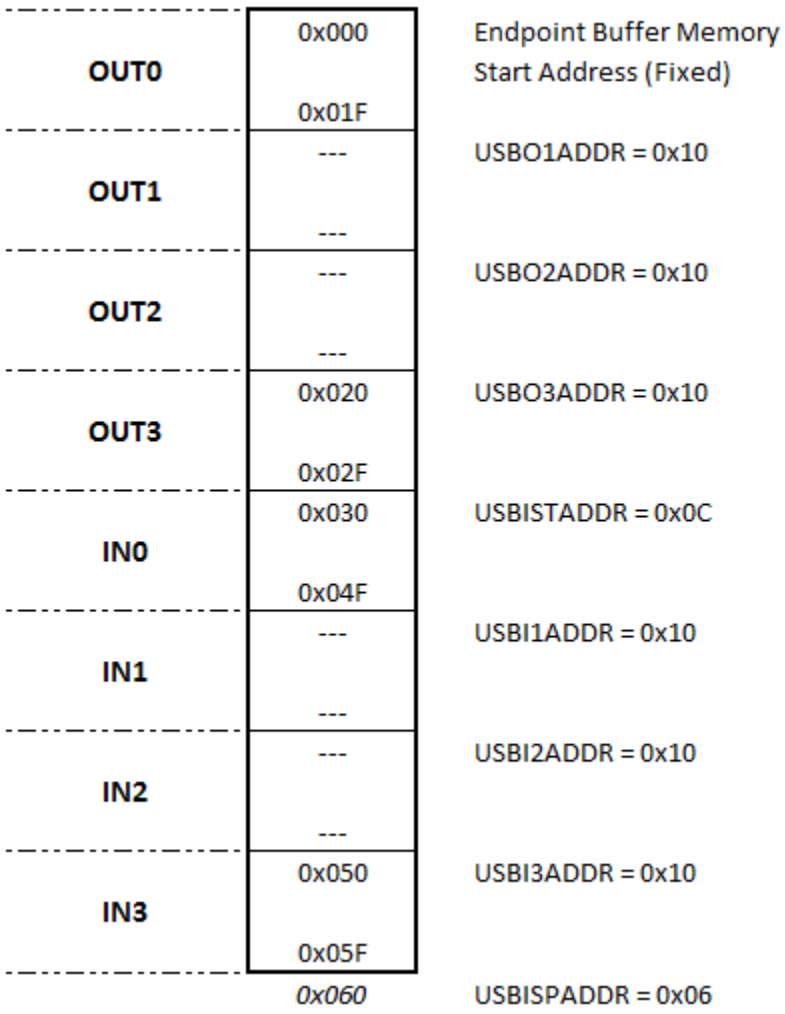

Figure 1. Example Endpoint Buffer Memory Allocation

HID Class Configuration Descriptor

Similar to other USB devices, the HID class requires a set of information (descriptors) to describe general information about the device. Additionally, the HID class includes information to describe the exact type of device and its features. Figure 2 shows the structure and organization of the configuration descriptors used in this application note.

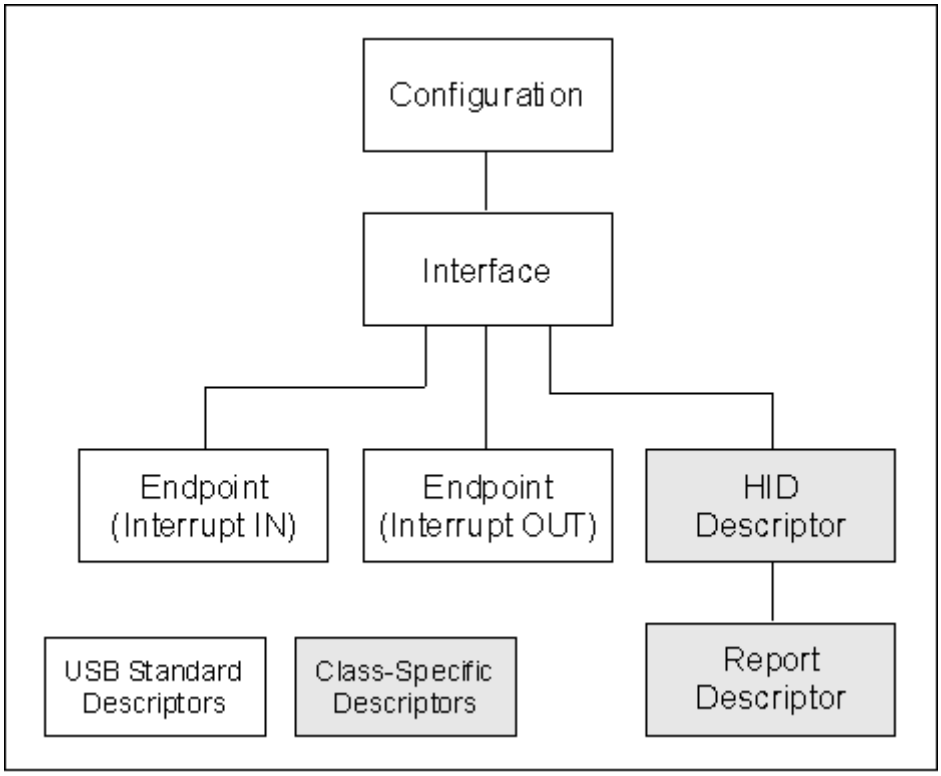

Figure 2. Human Interface Device Class Configuration

In defining these descriptors, HID specifications require that the HID descriptor must be inserted between the interface and endpoint descriptors. Therefore, the order should be:

```
Configuration Descriptor
    Interface Descriptor
        HID Descriptor
            Endpoint Descriptor (Interrupt IN)
            Endpoint Descriptor (Interrupt OUT)
```

HID Descriptor and HID Report Descriptor

The HID descriptor provides an overview of the number and type of descriptors included in the device's configuration. This keyboard application note includes the report descriptor in the device's configuration. The HID report descriptor defines the device's data packets that will be exchanged to/from the host during communication. It includes the number of bytes in a data packet, bit assignments, and byte arrangement.

There are two types of reports:

Input Reports. Input reports are used to send data packets from the device to the host

Output Reports. Output reports are used to send data from the host to the device

Figures 3 and 4 describe the format for Input and Output reports respectively.

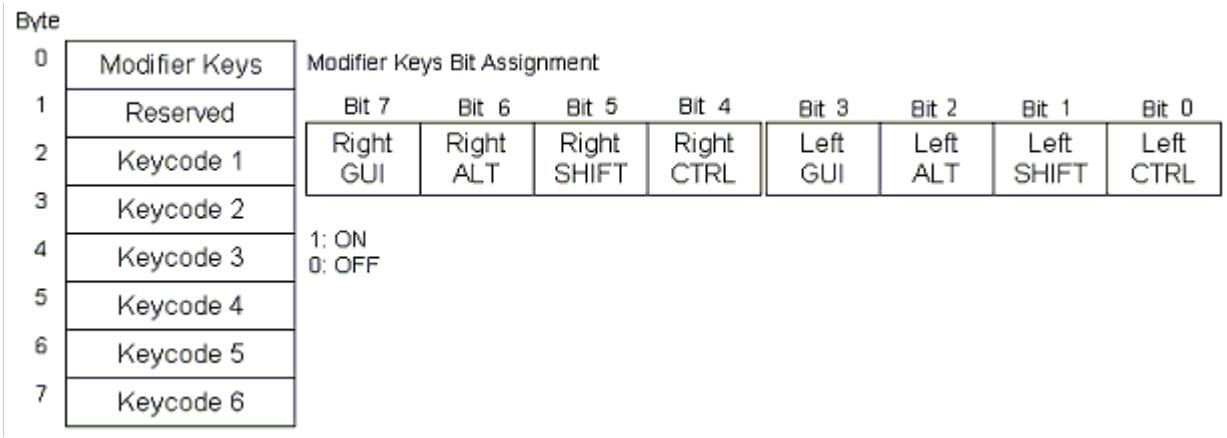

Figure 3. Input Report Format

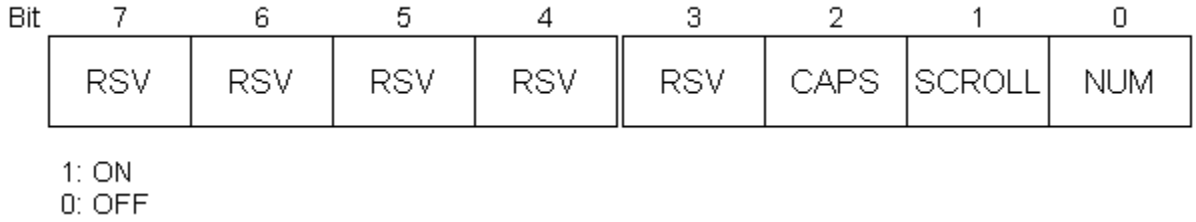

Figure 4. Output Report Format

Hardware Design

A 4 x 5 keypad matrix is used to simulate the alphanumeric keys in a keyboard, in addition to the switches and LEDs provided in the Z8F6482 development board.

Figure 5 illustrates how the keypad is connected to the Z8F6482 development board. Each button on this keypad is assigned a specific key code to be sent to the host when pressed.

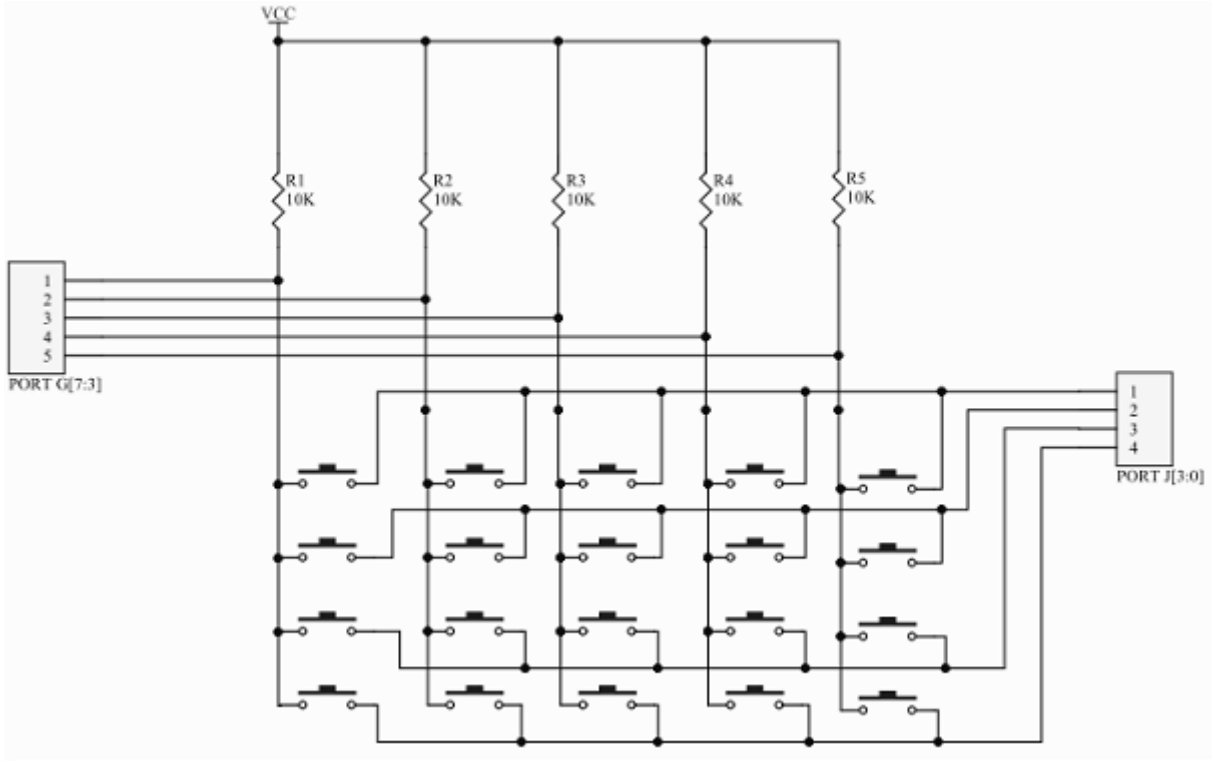

Figure 5. 4 x 5 Keypad Matrix Interface

Notes:

1. The keypad switches are a sample of the keys that appear on a full QWERTY keyboard and are used to simulate key press events on the keyboard's alphanumeric keys.
2. PJ3-PJ0 goes to row0-row3, in this order.
3. PG7-PG3 goes to col0-col4, in this order.

Table 1 lists the key code assignments for each switch in the keypad matrix.

Table 1. Keypad Row-Column Key Code Assignment

	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	CAPS Lock	SCROLL Lock	NUM Lock	ENTER	Backspace
Row 1	A	B	C	D	E
Row 2	F	G	H	I	J
Row 3	K	L	M	N	O

The switches provided with the development board are used to simulate key press events on the keyboard's modifier keys. The modifier keys are the Shift, Ctrl, and Alt keys. For this demonstration, only the Shift and Ctrl keys are assigned. Table 2 lists key assignments for the on-board switches.

Table 2. Modifier Keys Switch Assignment

Label	Port	Key Assigned
SW 2	PA6	Shift Key
SW 3	PA7	Ctrl Key

The LEDs are used to display the LED states sent by the host. Table 3 lists the LED assignments.

Table 3. LED Assignments

Label	Port	LED Assigned
D4	PB7	Caps Lock
D5	PC4	Scroll Lock
D6	PC5	Num Lock

Firmware Design

This application note uses the source code from the Z8F6482 MCU USB Peripheral Application Note (AN0411) as its base code. In AN0411, the USB firmware is divided into two modules – USB driver and Device Class Driver. The same USB driver module is used in this application note; to learn more about it, refer to AN0411. The device class driver is modified such that the USB device will work as an HID device instead of a CDC device. The following sections describe this HID Device Class Driver.

Note: The routines included in the HID device class driver module can be found in the `hid.c` file. The HID device descriptor, HID configuration descriptor, and HID report descriptor functions can be found in the `hid_conf.c` file.

HID Device Class Driver

The HID device class driver module includes basic endpoint handler routines to implement an HID keyboard application. Table 4 lists routines included in this module.

Table 4. HID APIs

Function Name	Returns	Description
HID_StdRequest	void	Handles HID-specific USB standard requests
HID_ClassRequest	void	Handles HID class-specific requests
HID_SetConfig	void	Handles application-specific endpoint configuration
HID_Out3Hdlr	void	Processes received data

Device Descriptor

Similar to the CDC application, the device descriptor provides general information about the USB device, including the product ID, vendor ID, USB specification release number, and device class code. The host requests this information during the enumeration process when it sends a `GET_DESCRIPTOR` request with a `DEVICE` descriptor type.

The device descriptor used in this application note specifies HID as the device class.

```
const USBDVC_DESCR DeviceDescr =
{
    sizeof(USBDVC_DESCR),              // bLength
    USBDESCR_DEVICE,                   // bDescriptorType = 0x01
    SWAP(USB_SPECIFICATION),           // bcdUSB = 0x0200
    HID_DEVICE_CLASS,                  // bDeviceClass = 0x03
    HID_SUBCLASS_BOOT,                 // bDeviceSubClass = 0x01
    HID_PROTOCOL_KEYBOARD,             // bDeviceProtocol = 0x01
    EP0_PACKETSIZE,                    // bMaxPacketSize0 = 0x20
    SWAP(USB_VENDOR_ID),               // idVendor = 0xFFFF
    SWAP(USB_PRODUCT_ID),              // idProduct = 0xFFFF
    SWAP(USB_RELEASE_NUMBER),          // bcdDevice = 0x0001
```

```
    USB_MANUFACTURER_IDX,               // iManufacturer
    USB_PRODUCT_IDX,                    // iProduct
    USB_SERIALNUMBER_IDX,               // iSerialNumber
    1                                   // bNumConfigurations
};
```

Configuration Descriptor

The configuration descriptor provides information about the device's configuration and interface settings, plus alternate settings and their endpoints. The host requests this information during the enumeration process when it sends a `GET_DESCRIPTOR` request with a `CONFIGURATION` descriptor type.

The USB specification defines a set of standard configuration descriptors whereas the HID specification defines additional class-specific descriptors. Figure 2 shows the sequence of a configuration descriptor for an HID keyboard device. This configuration descriptor is a combination of the USB and HID specifications.

```
// Configuration Descriptor
const USBDVC_CONFDESCR ConfigDescr =
{
    sizeof(USBDVC_CONFDESCR),           // bLength
    USBDESCR_CONFIG,                    // bDescriptorType
    SWAP(CONFIG_TOTAL_LEN),             // wTotalLength
    1,                                  // bNumInterfaces
    1,                                  // bConfigurationValue
    0,                                  // iConfiguration
    0x80 | (USB_SELF_POWER << 6) | (USB_REMOTE_WU << 5),
                                       // bmAttributes
    50                                  // bMaxPower
};

// Interface Descriptor
const USBDVC_IFDESCR InterfaceDescr =
{
    sizeof(USBDVC_IFDESCR),             // bLength
    USBDESCR_INTERFACE,                 // bDescriptorType
    HID_KEYBOARD_INTERFACE,             // bInterfaceNumber
    0,                                  // bAlternateSetting
    2,                                  // bNumEndpoints
    HID_DEVICE_CLASS,                   // bInterfaceClass
    HID_SUBCLASS_BOOT,                  // bInterfaceSubClass
    HID_PROTOCOL_KEYBOARD,              // bInterfaceProtocol
    0                                   // iInterface
};

// Functional Descriptors
const UINT8 HidKeyboardDescr[] =
{
    9,                                  // bLength
    HID_DESCRTYPE_HID,                  // bDescriptorType
    GET_HIGHBYTE(HID_VERSION),          // bcdHID
```

```
    GET_LOWBYTE(HID_VERSION),
    0,                                  // bCountryCode
    1,                                  // bNumDescriptors
    HID_DESCRTYPE_REPORT,               // bDescriptorType
    GET_HIGHBYTE(0x003F),               // wDescriptorLength
    GET_LOWBYTE(0x003F)
};

// Endpoint Descriptor
const USBDVC_EPDESCR EpIn3Descr =
{
    sizeof(USBDVC_EPDESCR),             // bLength
    USBDESCR_ENDPOINT,                  // bDescriptorType
    0x83,                               // bEndpointAddress
    0x03,                              // bmAttributes(0x03=intr)
    SWAP(EP3_PACKETSIZE),               // wMaxPacketSize
    100                                 // bInterval
};

// Endpoint Descriptor
const USBDVC_EPDESCR EpOut3Descr =
{
    sizeof(USBDVC_EPDESCR),             // bLength
    USBDESCR_ENDPOINT,                  // bDescriptorType
    0x03,                               // bEndpointAddress
    0x03,                              // bmAttributes(0x03=intr)
    SWAP(EP3_PACKETSIZE),               // wMaxPacketSize
    100                                 // bInterval
};
```

HID Report Descriptor

Input and output report formats vary widely across HID applications. The host is able to identify these formats through the HID report descriptor. The input and output report formats shown in Figures 3 and 4 are a result of the HID report descriptor below.

```
// Keyboard Protocol 1, HID 1.11 spec, Appendix B, page 59-60
const UINT8 HidKeyboardReportDescr[HID_KYBRD_REPORTDESCR_SIZE] = {
        0x05, 0x01,         // Usage Page (Generic Desktop),
        0x09, 0x06,         // Usage (Keyboard),
        0xA1, 0x01,         // Collection (Application),
        0x75, 0x01,         // Report Size (1),
        0x95, 0x08,         // Report Count (8),

        0x05, 0x07,         // Usage Page (Key Codes),
        0x19, 0xE0,         // Usage Minimum (224),
        0x29, 0xE7,         // Usage Maximum (231),
        0x15, 0x00,         // Logical Minimum (0),
        0x25, 0x01,         // Logical Maximum (1),

        0x81, 0x02,         // Input (Data, Variable, Absolute),
  ;Modifier byte
        0x95, 0x01,         // Report Count (1),
        0x75, 0x08,         // Report Size (8),
        0x81, 0x03,         // Input (Constant),
  ;Reserved byte
        0x95, 0x05,         // Report Count (5),

        0x75, 0x01,         // Report Size (1),
        0x05, 0x08,         // Usage Page (LEDs),
        0x19, 0x01,         // Usage Minimum (1),
        0x29, 0x05,         // Usage Maximum (5),
        0x91, 0x02,         // Output (Data, Variable, Absolute),
  ;LED report

        0x95, 0x01,         // Report Count (1),
        0x75, 0x03,         // Report Size (3),
        0x91, 0x03,         // Output (Constant), ;LED report
                            // padding
0x95, 0x06,                 // Report Count (6),
        0x75, 0x08,         // Report Size (8),

        0x15, 0x00,         // Logical Minimum (0),
        0x25, 0x68,         // Logical Maximum(104),
        0x05, 0x07,         // Usage Page (Key Codes),
        0x19, 0x00,         // Usage Minimum (0),
        0x29, 0x68,         // Usage Maximum (104),

        0x81, 0x00,         // Input (Data, Array),
        0xc0                // End Collection
};
```

Equipment Used

The following equipment is used to build and test this application:

- Z8 Encore! XP Z8F6482 Development Board
- Zilog USB or Ethernet SmartCable
- USB A (Male) to Mini-B USB Cable
- 4 x 5 keypad matrix
- RS232 to 6-pin circuit adapter (optional)

Testing and Demonstrating the Application

This section describes the process of downloading and running this application.

Note: When running this application, the USB vendor ID and product ID are used to identify USB devices to a host. Each company is assigned a specific vendor ID. The IDs used in this application note are typical values for generic USB devices. If multiple applications with the same ID are run on a given device, the HID keyboard may not be automatically detected by Windows. If this occurs, change the vendor ID/product ID value so that at least one of the IDs is different per device. For example, in `hid.c`, change

```
#define USB_VENDOR_ID   (0xFFFF)
```

to

```
#define USB_VENDOR_ID   (0xFFFE)
```

Downloading Code to the Z8F6482 Development Board

1. Connect the USB SmartCable to the DBG terminal of the Development Board and connect the USB side to the development PC's USB port.
2. Connect the USB A (male) to Mini-B cable to P1 on the development board and connect the other end to a PC's USB port to apply power to the development board.
3. Download AN0416-SC01.zip from zilog.com and extract the source code files.
4. From the **File** menu, select **Open Project…** to open the ZDS II Z8 Encore v5.2.2 (or later) IDE.
5. Navigate to the extracted folder, select the `AN0416.zdsproj` project and click **Open** to open the project.
6. From the **Build** menu, select **Rebuild All** to rebuild the project.
7. From the **Debug** menu, select **Download Code** to load the program to the MCU.
8. After programming is complete, from the **Debug** menu, select **Stop Debugging**.
9. Remove the USB A (male) to Mini-B cable to remove power from the board.

10. Disconnect the USB Smartcable from the DBG terminal of the development board.

Running the Application

1. Ensure that code is programmed onto the development board by following the steps in the Downloading Code to the Z8F6482 Development Board section.
2. Connect a 4 x 5 keypad matrix to the development board as shown in Figure 5.
3. (Optional) Attach the RS232 to 6-pin circuit adapter to J20 of the development board to connect the board to a PC via serial port.
   - Open Realterm (or any other terminal application) and set its baud rate 57600 bps.
4. Connect the USB A (male) to Mini-B cable to P1 on the development board and connect the other end to a PC's USB port to apply power to the development board and provide a USB connection to the PC.

Note: After the Development Board with the USB cable is connected to the PC, wait for about 6–10 seconds before opening up the port in the RealTerm terminal.

5. Wait until Windows detects the new keyboard. If using Realterm, the message *HID keyboard ready!* appears, indicating that Windows is ready to receive inputs from the new keyboard.
6. Open a text-editing program such as Microsoft Notepad and begin typing by pressing a switch from the keypad matrix.
7. Observe how each character is typed into Notepad.

Note: To reset the application, prior to pressing the SW2 reset button on the Development Board,ensure that you close the RealTerm terminal port first by clicking the Open button on the Port tab and clearing the display. Then, press the SW2 reset button and wait 6–10 seconds before opening the RealTerm port again.

Summary

By modifying the descriptors presented in the Z8F6482 MCU USB Peripheral Application Note (AN0411), the Z8F6482 development board is able to function as an HID keyboard.

References

The following documents are associated with this Application Note:

- Z8 Encore XP F6482 Series Product Specification (PS0294)
- Z8 Encore XP F6482 Series Development Kit User Manual (UM0263)
- Universal Serial Bus Specification Rev 2.0 (http://www.usb.org/developers/docs/usb20_docs/)
- Device Class Definition for Human Interface Devices (HID), version 1.11 (http://www.usb.org/developers/hidpage/)
- HID Usage Tables version 1.12 (http://www.usb.org/developers/hidpage/)

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

 Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. The information contained within this document has been verified according to the general principles of electrical and mechanical engineering.

Z8 Encore! XP is a trademark or registered trademark of Zilog, Inc. All other product or service names are the property of their respective owners.