OVERVIEW

This Technical Brief describes a USB combination device consisting of a mouse and a gamepad implemented on a PIC16C765. Previous technical briefs describe the translation of a PS/2 mouse to a USB mouse (TB055) and the translation of a gameport gamepad to a USB gamepad (TB054). These two devices were combined to form the USB combination device described in this brief. Details about the translation and implementation of the mouse and the gamepad are not covered. Refer to the earlier briefs for this information as well as their circuit diagrams. This brief focuses on setting up the descriptors appropriately for a combination device.

DESCRIPTORS

A combination device, from the USB standpoint, is a device with multiple interfaces. An interface is an endpoint or a group of endpoints that perform(s) a certain function. Since only two endpoints are available in low speed USB devices, low speed combination devices will always consist of two interfaces, one endpoint each. Two interfaces require two interface descriptors. In this case, interface one is a mouse and interface two is a gamepad. Figure 1 shows the descriptors needed for the implementation of the combination mouse and gamepad USB device. Both the mouse and gamepad are part of the Human Interface Device (HID) class - this means each device has a report and HID descriptor associated with it.

Note 1: See the complete descriptor listing in file descript.asm

FIGURE 1: USB COMBINATION DEVICE DESCRIPTORS
The descriptors are sent to the host in response to the following requests:

- Get_Device_Descriptor
- Get_Configuration_Descriptor
- Get_Report_Descriptor
- Get_String_Descriptor

The DEVICE descriptor is the first descriptor to be sent to the host during enumeration. It is sent in response to the Get_Device_Descriptor request. The CONFIGURATION, INTERFACE, HID, and ENDPOINT descriptors are all sent to the host in response to the Get_Configuration_Descriptor request. They are sent in the following order (refer to Figure 1):

- CONFIGURATION
- INTERFACE1
- HID1
- ENDPOINT1
- INTERFACE2
- HID2
- ENDPOINT2.

Report descriptors are sent in response to the Get_Report_Descriptor request and finally STRING descriptors in response to the Get_String_Descriptor request. Depending on the operating system, the string descriptors may be requested before the report descriptors.

How report descriptors are associated with endpoints is not entirely intuitive or easily understood from the HID Class Definitions. Report descriptors are not like other descriptors in that they do not have a field that identifies them as a report descriptor. Nor do report descriptors have an index number that distinguishes one from another. Report descriptors are composed entirely of report format data. The host associates a report descriptor with the appropriate endpoint using the following information:

1. The order report descriptors are mentioned by HID descriptors
2. The descriptor tree (contains the descriptors found on a particular branch)

From Figure 1 it is seen that HID1 is associated with ENDPOINT1 because it is on the same branch of the descriptor tree as ENDPOINT1. Also seen in the figure, HID1 specifies that it has a report descriptor, REPORT1. The host assigns this report descriptor an index value of 0 (this is a zero based array) simply because HID1 is the first descriptor sent to the host that specifies a report descriptor. Similarly, the host assigns the report descriptor specified by HID2 and index value of 1. The host keeps track of which descriptors branch off a particular interface. Therefore, the host is able to deduce that report descriptors with indexes 0 and 1 are describing communication that will take place over the endpoints described by ENDPOINT1 and ENDPOINT2, respectively. The host will obtain these report descriptors from the device with the Get_Report_Descriptor request. As part of this request, the host specifies the index number of the descriptor it desires. It is up to the developer to make sure that REPORT1 is sent to the host when index 0 is specified in the request and REPORT2 is sent to the host when index 1 is specified. The REPORT_DESCRIPTOR_INDEX, in file descript.asm, makes this process possible. In this index, the labels for the report descriptors are listed in the order that the corresponding HID descriptors are sent to the host. Example 1 is a listing of the report descriptor index for the USB mouse/gamepad combination device:

**EXAMPLE 1: REPORT DESCRIPTOR INDEX**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report_desc_index</td>
<td>; this function is called with the index number of the report descriptor in W</td>
</tr>
<tr>
<td>movwf temp</td>
<td></td>
</tr>
<tr>
<td>movlw HIGH RDI_start</td>
<td></td>
</tr>
<tr>
<td>movwf PCLATH</td>
<td></td>
</tr>
<tr>
<td>movlw low RDI_start</td>
<td></td>
</tr>
<tr>
<td>addwf temp,w</td>
<td></td>
</tr>
<tr>
<td>btfsc STATUS,C</td>
<td></td>
</tr>
<tr>
<td>incf PCLATH,f</td>
<td></td>
</tr>
<tr>
<td>movwf PCL</td>
<td></td>
</tr>
<tr>
<td>RDI_start</td>
<td>; this table calculates the offsets for each report</td>
</tr>
<tr>
<td>retlw low ReportDescriptorLen1</td>
<td></td>
</tr>
<tr>
<td>retlw high ReportDescriptorLen1</td>
<td></td>
</tr>
<tr>
<td>retlw low ReportDescriptorLen2</td>
<td></td>
</tr>
</tbody>
</table>
The REPORT DESCRIPTOR INDEX is called in order for the firmware to obtain the starting address of a specific report descriptor. Another jump table is responsible for feeding the descriptor to the host once it has this information.

| Similar descriptor indexes are included in Microchip’s USB firmware for multiple string and configuration descriptors. See file descript.asm for their listings |

Table 1 specifies the fields within descriptors that are affected by a combination device.

### TABLE 1: AFFECTED FIELDS WITHIN DESCRIPTORS

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Offset</th>
<th>Field</th>
<th>Why Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>4</td>
<td>bNumInterfaces</td>
<td>Specifies number of interface descriptors to follow.</td>
</tr>
<tr>
<td>Interface</td>
<td>2</td>
<td>bInterfaceNumber</td>
<td>Multiple interfaces mean interface descriptors with different interface numbers (IDs).</td>
</tr>
<tr>
<td>Interface</td>
<td>4</td>
<td>bNumEndpoints</td>
<td>There will always be one endpoint per interface in a low speed combination device.</td>
</tr>
<tr>
<td>Endpoint</td>
<td>2</td>
<td>bEndpointAddress</td>
<td>This is the endpoint number and direction. Two endpoints must be used for a low speed combination device.</td>
</tr>
</tbody>
</table>

### MEMORY USAGE

In the PIC16C765, the following memory was used:

- Data Memory: 50 bytes
- Program Memory: 2K bytes

### REFERENCES

1. USB Specification, Version 1.1: Chapter 9 (located at www.usb.org)
2. Device Class Definition for Human Interface Devices (located at www.usb.org)
3. HID Usage Tables (located at www.usb.org)
5. USB Complete, Second Edition, Jan Axelson; Lakeview Research, 2001 (www.lvr.com)
7. TB054: An Introduction to USB Descriptors with a Gameport to USB Gamepad Translator
8. TB055: PS/2 to USB Mouse Translator
9. TB056: Demonstrating the Set_Report Request with a PS/2 to USB Keyboard Translator Example
10. TB058: Demonstrating the Soft Detach Function with a PS/2 to USB Translator Example
APPENDIX A: SOURCE CODE

Due to the length of the source code for the USB Combination Mouse/Gamepad example, the source code is available separately. The complete source code is available as a single WinZip archive file, tb057sc.zip, which may be downloaded from the Microchip corporate Web site at:

www.microchip.com
Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable".
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELoQ, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PROMATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microID, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.