USB FIRMWARE USER'S GUIDE

USB Firmware User's Guide

Table of Contents

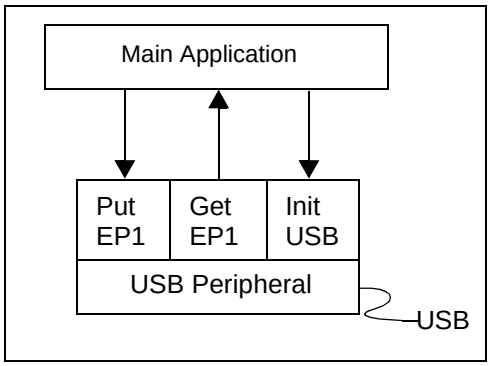
USB Firmware User's Guide

Chapter 1. General Information

1.1 Introduction

The USB Support Firmware provided by Microchip Technology Inc., implements the functionality defined in Chapter 9 of the USB Specification (version 1.1) and the Human Interface Device Class Definition. By using a pre-defined library of Application Programming Interfaces (APIs), the Support Firmware allows developers with little or no USB experience to quickly implement their applications on PIC16C745/765 microcontrollers. Routine tasks such as initialization and communication, as well as other specialized USB tasks, are handled by nine APIs and two user-definable functions. This allows the developer to concentrate on application-specific details such as device descriptors, and not worry about re-creating common utilities over and over again.

Figure 1.1:



V2.00 marks a fairly substantial change in architecture from the V1.xx versions. While V1.xx handled the USB completely from the Internet Service Routine (ISR), V2.00 shifts most of the processing to a function called from the main program loop. There are several advantages to this. First, it eliminates the many levels of the stack that currently must be reserved for the ISR. Second, it minimizes the length of the ISR which is desirable from a "good software engineering practice" standpoint. Finally, it gives control over, when the USB is serviced, to the software engineer writing the application. This is particularly important for timing critical systems where a few hundred instruction cycles to process the interrupt at the wrong time could potentially damage the system.

V2.00 requires rev. A2 or newer devices. See Errata DS80114C and DS80143A for more details.

1.1.1 About This Guide

The USB support firmware is available in two programming languages: Microchip Assembly, and Hi-Tech C. Each version will have its own section in this manual, where appropriate.

1.1.2 Firmware Updates

Microchip is constantly providing new USB application examples. Please refer to Microchip's webpage, www.microchip.com, for updates and new USB examples.

1.2 USB Basics

There are certain basic USB concepts that apply to the PIC16C745/765. This section will clear up any confusion regarding the capabilities of this device.

1.2.1 Endpoints

Endpoints are buffers where data waits to be put onto the USB bus, or where data is removed from the bus. Endpoints consist of a number and a direction. All USB devices use EP0 for administrative communication between the device and the host. This means EP0 is bi-directional and is actually two endpoints, EP0 IN and EP0 OUT.

The Buffer Descriptor Table for the PIC16C745/765 has 40 bytes, or five (eight byte each) buffers, set aside for endpoint buffers. EP0 IN and OUT need dedicated buffers since a setup transaction can never be NAKed. That leaves three buffers for four possible endpoints. Of these, only two can be used because the USB specification only allows low speed devices to have two endpoints (USB V1.1 paragraph 5.3.1.2.) other than EP0.

The default configuration allocates individual buffers to EP0 OUT, EP0 IN, EP1 OUT, and EP1 IN. The last buffer is shared between EP2 IN and EP2 OUT. Again, the spec says low speed devices can only use two endpoints beyond EP0. This configuration supports most of the possible combinations of endpoints (EP1 OUT and EP1 IN, EP1 OUT and EP2 IN, EP1 OUT and EP2 OUT, EP1 IN and EP2 OUT, EP1 IN and EP2 IN). The only combination that is not supported by this configuration is EP2 IN and EP2 OUT. If an application needs both EP2 IN and EP2 OUT, the Set_Configuration function will need to be edited to give each of these endpoints dedicated buffers at the expense of EP1. Using one of the other combinations should be sufficient, however.

1.2.2 Bandwidth

The PIC16C745/765 is a low-speed USB device. The data rate for one endpoint is limited to 800 bytes/second. There are two endpoints (beyond EP0) and another 800 bytes/second is available on the other endpoint. This gives a uni-directional minimum guaranteed transfer rate of 1600 bytes/second. This transfer rate is uni-directional because, as mentioned above, an endpoint is defined as a number and a direction. If the desired direction of communication was device to host, for instance, endpoints EP1 IN and EP2 IN would be used.

The HID Class Specification makes available the Get_Report and Set_Report requests. These requests can increase the bandwidth of a device up to 10 times that available on EP1 or EP2. Get_Report and Set_Report do this because they allow for data tranfers via EP0. EP0 has a polling rate of 1ms, thus the increase in bandwidth: up to 8000 bytes/second in each direction.

1.2.3 Data Types

The USB specification defines four transfer types: control, interrupt, isochronous, and bulk. Of these, only two are available to low-speed devices: control and interrupt. Control transfers support enumeration and communicate via EP0. The host sends requests and setup information to the microcontroller over EP0 OUT. The microcontroller also responds to those requests via EP0 IN. Interrupt transfers guarantee maximum latency. The user specifies the maximum polling rate in the endpoint descriptor. For instance, if 10 ms is specified, this means that the host polls the particular endpoint at least once every 10 ms. 10 ms is the fasted interval that can be requested according to the USB Specification. For clarification, low-speed devices use EP0 control transfers while EP1 and EP2 use only interrupt transfers.

Chapter 2. Example Application

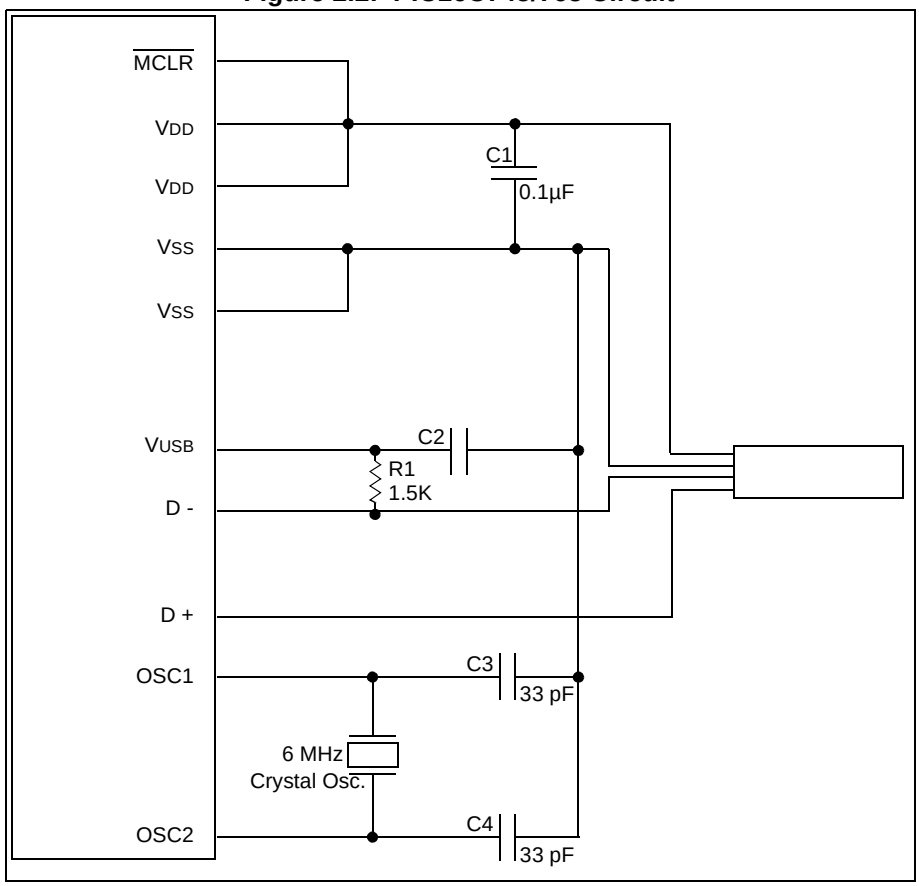
2.3 Cursor in a Circle Example

The USB Support Firmware includes an example application. This application enumerates as a mouse and has the effect of moving the cursor in a circle on the screen. The example provides known working software for the purpose of verifying hardware. It also provides properly setup descriptors, an example ISR, and an example of how enumeration is initiated.

2.3.1 Hardware

The circuit required to run the firmware is shown in Figure 2.2. The PICDEM™ USB Demonstration Board is a great investment as it provides the circuit and a generous prototyping area. It includes status LEDs for viewing the enumeration progress of the USB support firmware, connectors for USB, RS232, gameport and PS/2 connectors, and sockets for both the PIC16C745 and PIC16C765.

Figure 2.2: PIC16C745/765 Circuit



2.3.2 Running the Example

The following sections provide instructions on how to run the USB firmware on a PIC16C745, or on the emulator. Emulation tips are also given.

2.3.2.1 Programming a PIC16C745

1. Unzip usbxxxx.zip to a project folder.
2. Program a PIC16C745. Make sure the configuration bits are set as follows:
   - Oscillator: H4
   - Watchdog Timer: Off
   - Power-up Timer: Off
   - Code Protect: Off
3. Apply the PIC16C745 in the circuit described in the previous section.
4. Plug in the USB cable. On machines with Windows 98, second edition or newer, or Mac OS X, the operating system will detect a new device and install the necessary drivers automatically. After this occurs, the cursor will rotate in a small circle on the screen. To stop the cursor from rotating, detach the USB cable.

2.3.2.2 Using the Emulator

1. Unzip usbxxxx.zip to a project folder.
2. Make sure the emulator is setup as follows in the development mode dialog. (Setup will differ for a third party emulator.)
   - Tools: MPLAB-ICE Emulator
   - Clock - Desired Frequency: 24 MHz
   - Configuration - Watch Dog Timer None
   - Power - Processor Power: From Target Board
3. Attach the appropriate processor module and device adapter to the ICE. Then attach the device adapter to the circuit described in the previous section. Refer to Microchip's Product Line Card for the identity of the processor module and device adapter.
4. Plug in the USB cable and run the emulator. On machines with Windows 98, second edition or newer, or Mac OS X, the operating system will detect a new device and install the necessary drivers automatically. After this occurs, the cursor will rotate in a small circle on the screen. To stop the cursor from rotating, press F5.

2.3.2.3 MPLAB-ICE Emulation Tips

1. The emulator is getting its power from the target board, so the board must be powered. This is done by plugging in the USB cable. (On the PICDEM USB board make sure the J3 jumper is set to “bus powered”.)
2. Make sure the emulator is turned on.
3. Make sure the emulator development mode is selected:
   - Go to the “Option” menu and click “Development Mode”.
   - Select the “MPLAB-ICE Emulator” radio button in the Development Mode dialog.
   - Click “OK”. The emulator will initialize.
4. Make sure the proper processor module and device adapter are installed. Refer to Microchip’s Product Line Card for this information.
5. Should there be a problem at this point, refer to the MPLAB-ICE User’s Guide.

2.4 File Packaging

The file packaging differs between the Assembly and C versions of the USB Support Firmware. The following paragraphs detail the file packaging for each version.

2.4.1 Assembly Version

The Assembly version is comprised of seven files necessary to build the Hex file that is with the project. These files are:

- movecurs.pjt
- 16C745.lkr
- usb_defs.inc
- usb_main.asm
- usb_ch9.asm
- hidclass.asm
- descript.asm

The last five files pertain to implementing USB on the PIC16C745/765. A brief description of these files are as follows:

- usb_defs.inc - Defines USB variables and macros used in the following assembly files.
- usb_main.asm - Implements the Cursor in a Circle example. The proper way of implementing an Interrupt Service Routine (ISR), initializing USB communication, polling the ServiceUSB routine, and using the Put interface are components of the example that are useful to the user.
- usb_ch9.asm - Implements the functions found in Chapter 9 of the USB Specification, version 1.1(1). These functions are discussed in Section 3.6.
- hidclass.asm - Implements the functions found in the Human Interface Device Class Definition(2). As some users will not be implementing a HID device, details on removing HID related code from the firmware is given in Section 4.14.2. Section 3.6.3 describes the HID functions in detail.
- descript.asm - Contains the descriptors for the Cursor in a Circle example. The user's own descriptors will replace those found in this file. Details on creating descriptors are covered in Section 4.10.

2.4.2 HI-TECH C Version

The HI-TECH C version is composed of five files. These files are:

- usb.pjt
- usb.h
- usb_defs.h
- usb_main.c
- usb_ch9.c

The last four files pertain to implementing USB on the PIC16C745/765. A brief description of these files are as follows:

- usb.h - Header file for usb_main.c. It declares the function from usb_ch9.c that is used in usb_main.c.
- usb_defs.h - Defines the USB variables and macros used in following files.
- usb_main.c - Implements the Cursor in a Circle example. The proper way of initializing USB, polling the ServiceUSB() function, and using the Put interface are components of the example that are useful to the user.
- usb_ch9.c - Implements the functions found in the chapter 9 of the USB specification version 1.1 (1) and the Human Interface Device Class Definition (2). The descriptors for the Cursor in a Circle example are also in this file. Details about this file are covered in sections 6.1 and 8.5.

NOTES:

Chapter 3. Processor Resources

3.5 Microcontroller Resource Usage

Processor resources are always a concern in microcontroller firmware development. These resources include ROM, RAM, Common RAM, Stack Levels and processor cycles. This chapter attempts to quantify the impact on each of these resources, and shows ways to avoid conflicts.

3.5.1 Stack Levels

The hardware stack on the PICmicro microcontroller® is only eight levels deep. So the worst case call between the application and ISR can only be eight levels. The ISR only takes two stack levels, leaving six for the application code. This can easily be reduced to one level, if necessary. The ServiceUSB() function requires five levels, therefore, the ISR should never be greater than three stack levels deep.

3.5.2 ROM

The code required to support the USB interrupt, including the chapter 9 interface calls, but not including the descriptor tables, is approximately 1.5 kBytes. The descriptor and string descriptor tables can each take up to an additional 256 bytes.

3.5.3 RAM

With the exception of Common RAM discussed below, servicing the USB interrupt requires approximately 40 bytes of RAM in Bank 2. This leaves all the General Purpose RAM in Banks 0 and 1, plus half of Bank 2, available to use.

3.5.4 Common RAM Usage

The PIC16C745/765 has 16 bytes of common RAM. These are the last 16 addresses in each bank, and all refer to the same 16 bytes of memory without regard to which bank is currently addressed by the RP0, RP1 and IRP bits. These are particularly useful when responding to interrupts. When an interrupt occurs, the ISR doesn't immediately know which bank is addressed. With devices that don't support common RAM, the W register must be provided for in each bank. The PIC16C745/765 can save the appropriate registers in common RAM and not waste a byte in each bank for the W register.

3.6 Function Call Reference

The USB support firmware functions range from the most basic to advanced. The basic functions initialize USB communication, get data from the bus, or put data on the bus. More advanced functions deal with power-save modes and reinitializing enumeration. This section also discusses common user defined functions.

Note: The assembly function call will be followed by the Hi-Tech C function call in {}. Any Hi-Tech C specific notes will also be in {}.

3.6.1 Basic Functions

- InitUSB {InitUSB()} - This function should be called by the main routine after power-up. Be sure to precede the call with a 16 us delay to give the USB peripheral time to reset before beginning enumeration. InitUSB enables the USB peripheral by setting the DEV_ATT bit, thus prompting the host to enumerate the device. The USB Reset interrupt is enabled as well as the general USB interrupt. Global and peripheral interrupts are also enabled.
- ConfiguredUSB {ConfiguredUSB} - A macro in both the Assembly and Hi-Tech C versions of the firmware, ConfiguredUSB polls the enumeration status bits to see if the device has been configured by the host. ConfiguredUSB returns a 1 in the Z flag if the device is configured, or a 0 if it is not. This macro should be used after the call to InitUSB in order to determine whether Endpoints 1 and 2 can be used (via GetEP*n* or PutEP*n*). Endpoints 1 and 2 can only be used if the device is configured.
- DeinitUSB {DeinitUSB()} - Causes the host to ignore the device. The USB serial interface engine (SIE) is placed in suspend to conserve power and the device essentially detaches itself from the bus. USB interrupts are disabled.
- ServiceUSB {ServiceUSB()} - Call this function from the main program loop to process USB transactions. Transactions from the host signal a TokenDone. The enumeration process works off of commands specified in chapter 9 of the USB specification(1). The transactions are parsed and supply the appropriate response. See Section 4.12 for information on how often to call this function.
- PutEP*n* {PutEP*n*(bytes, buffer[])} - Sends data to the host via endpoint *n*. The number of bytes being sent must be specified in W {the 'bytes' argument}. The FSR and the IRP bits must point to the block of data that will be sent. {The buffer to be sent should be entered as the second argument, 'buffer[]'.} The Carry flag is set {the function returns a one} when the UOWNs bit (BDndST:<7>) is set appropriately signifying the data will be sent. The Carry flag is cleared {a zero is returned} if the function fails to send the data.

- GetEP*n* {GetEP*n*(buffer[])} - Retrieves data from the host via endpoint *n*. The desired location for the data must be loaded into FSR and the IRP bits. {Specify the desired block of ROM the data is to be placed in 'buffer[]'.} The Carry flag is set {a one is returned} when the UOWNs bit (BDndST:<7>) is set appropriately and the data is retrieved successfully. The Carry flag is cleared {a zero is returned} when the function fails to retrieve the data.

3.6.2 Advanced Functions

- SoftDetachUSB {SoftDetachUSB()} - Clears the DEV_ATT bit electrically disconnecting the device from the bus, then reconnecting so it can be re-enumerated by the host. This process takes approximately 50 ms to ensure that the host has seen the device disconnect and re-attach to the bus. The benefit of this function is that it allows a device to re-enumerate as a new device on-the-fly. An example of such application is a PS/2 dongle that enumerates a USB mouse if a PS/2 mouse is plugged into it, or as a USB keyboard if a PS/2 keyboard is plugged into it.
- RemoteWakeup {RemoteWakeup()} - Prompts the host to reinitiate communication with the PICmicro Microcontroller. If the host has enabled the Remote Wake-up feature, the device can send a resume signaling command to the host in order to bring the particular bus segment back into the active condition (see Section 7.1.7.5 in the USB Specification 1.1.) RemoteWakup sends the resume signaling command.
- StallUSBEP/UnstallUSBEP {StallUSBEP(endpoint), UnstallUSBEP(endpoint)} - Sets or clears the stall bit in the endpoint control register. Enter these functions with the desired endpoint in the W register {in the 'endpoint' argument}. The stall bit indicates to the host that user intervention is required, and until such intervention is made, further attempts to communicate with the endpoint will not be successful. Once the user intervention has been made, UnstallUSBEP will clear the bit allowing communications to take place. These calls are useful to signal to the host that user intervention is required. An example of this might be a printer out of paper.

3.6.3 User Defined Functions (Application Specific)

- HID_Get_Report - Services HID Get_Report requests. These requests make it possible for HID devices to send data to the host via EP0. Should developers choose to send data via EP0, they must write the body of this function in order to handle the data. This function is located in hidclass.asm {usb_ch9.c}. Writing this function requires understanding Section 7.2.1 of the HID Class Definition, version 1.1.
- HID_Set_Report - Services HID Set_Report requests. These requests make it possible for HID devices to receive data from the host via EP0. Should developers choose to receive data via EP0, they must write the body of this function in order to handle the data. This function is located in usb_ch9.asm {usb_ch9.c}. Writing this function requires understanding Section 7.2.2 of the HID Class Definition, version 1.1.

3.7 Behind the Scenes

The PIC16C745/765 powers up with no interrupts enabled. The first thing that happens is the main routine waits 16 us for the USB Serial Interface Engine (SIE) to reset. Once this happens, InitUSB is executed. InitUSB clears the error counters and enables the 3.3V regulator (UCTRL:<DET_ATT>) and the USB Reset interrupt. This implements the requirement that USB devices cannot respond to commands until the device has been reset.

When the 3.3V regulator is enabled, the host sees the device and recognizes it as a low-speed device due to the pull-up resistor from the regulator output (Vusb) to the D-line. The host resets the device to begin the enumeration process. During the Reset process, the firmware initializes Endpoint0's Buffer Descriptor Table (BDT) and clears out the USTAT FIFO.

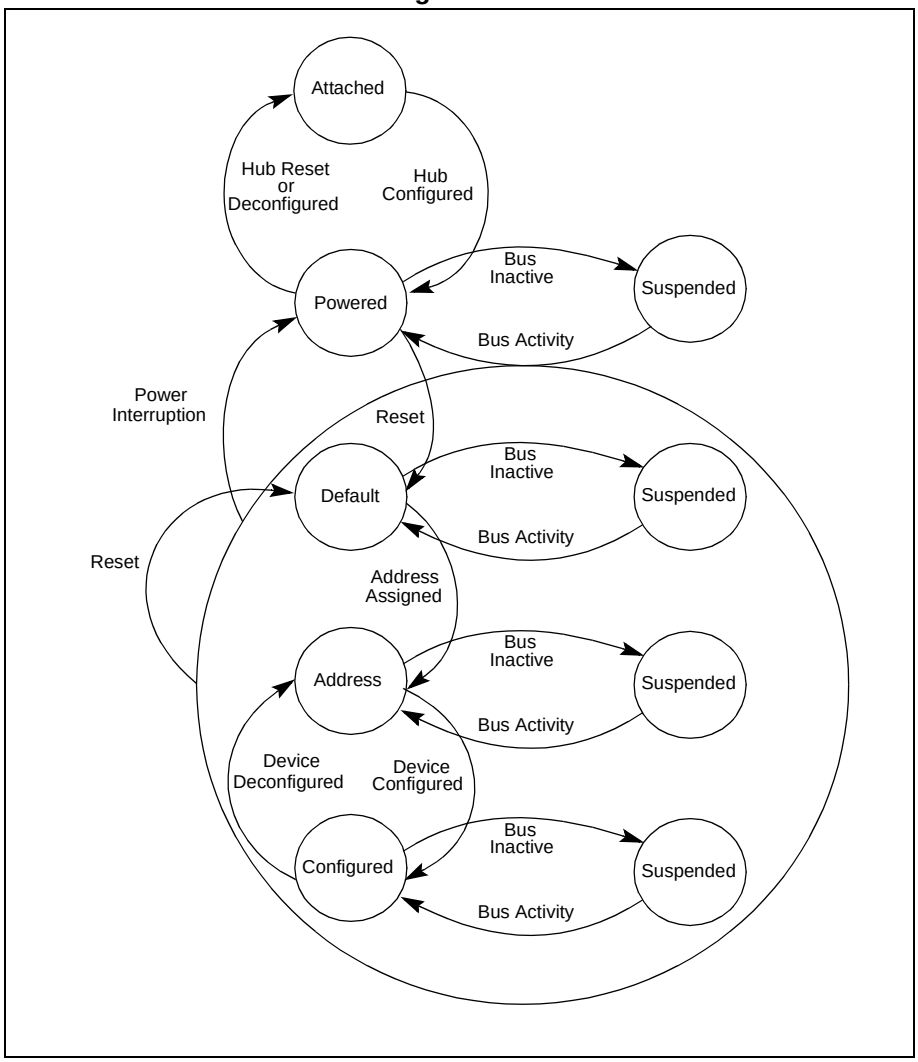
At this point, the host will start issuing Setup transactions containing Chapter 9 requests to enumerate the device. Typically, it will start with a Get Device Descriptor to find out what size packets the device will deal with. This information is located in the 8th byte of the device descriptor. These Setup transactions signal a TokenDone to the PICmicro microcontroller, which are then processed by the ServiceUSB call. (ServiceUSB must be polled on a regular basis after InitUSB is executed for enumeration to be successful. See Section 4.12 for polling interval requirements.)

When the host sends a Setup token requesting the device descriptor, the following occurs:

1. The USB Peripheral receives the Setup transaction, places the data portion in the EP0 out buffer, loads the USTAT register to indicate which endpoint received the data, and sets the Token Done (TOK_DNE) flag. The Chapter 9 command then interprets the Setup token and sets up the data to respond to the request in the EP0 In buffer. Then it sets the UOWN bit to tell the SIE there is data available to send to the host.
2. Next, the host sends an IN transaction to receive the data from the setup transaction. The SIE sends the data from the EP0 IN buffer and then sets the TOK_DONE flag to notify the microcontroller that the data has been sent. If there is additional data, the next buffer full is setup in the EP0 IN buffer.

This token processing sequence holds true for the entire enumeration sequence, which walks through the flow chart shown at the beginning of Chapter 9 of the USB spec (see Figure 3.3.) The device starts off in the powered state, transitions to RESET via the Reset interrupt, transitions to ADDRESSED via the Set Address request (this request typically follows the Get Device Descriptor request mentioned above), and transitions to CONFIGURED via a Set Configuration command.

Figure 3.3:



The USB peripheral detects several different errors and handles most of them internally. The USB_ERR flag notifies the PICmicro microcontroller that an error has occurred. No action is required by the PICmicro microcontroller when an error occurs. Instead, the errors are simply acknowledged and counted. There is no mechanism to pull the device off the bus if there are too many errors. If this behavior is desired, it must be implemented in the application.

The Activity interrupt is left disabled until the USB peripheral detects no bus activity for 3 ms. At this point, the UIDLE flag is set and USBSleep is executed. USBSleep suspends the USB peripheral and enables the activity interrupt. The Activity interrupt then reactivates the USB peripheral, when bus activity resumes, so processing may continue.

Chapter 4. USB Application Development

4.8 Recommended Tools

Several development tools are recommended when developing USB devices with the PIC16C745/765: the in-circuit emulator and a USB protocol analyzer. An in-circuit emulator will cut down on design time significantly over the "learn and burn" method. A USB protocol analyzer monitors all USB traffic on the bus. When USB communication is not behaving as desired, sometimes the only way to track the problem down in a timely manner is to step through the USB traffic generated when the problem occurred.

4.9 Deciding on Endpoint Usage

Before creating the descriptors, it is beneficial to spend some time deciding what endpoints are required for a specific application. Section 1.2.1 provides guidance as to what endpoint configurations are available to the user. As previously mentioned, only two endpoints (consisting of a number and direction) are allowed by the USB specification for low-speed devices. The following is a list of possible endpoint configurations:

1. One OUT endpoint
2. One IN endpoint
3. Two OUT endpoints
4. Two IN endpoints
5. One OUT endpoint and one IN endpoint

If configuration 5 is desired, it is suggested that EP2 IN and EP2 OUT not be chosen to satisfy this configuration. This will require re-allocating the endpoint buffers in Set_Configuration. Three physical configurations will satisfy this situational configuration and will ensure that this change is never necessary. These physical configurations are:

1. EP1 IN and EP1 OUT
2. EP1 IN and EP2 OUT
3. EP2 IN and EP1 OUT

4.10 Creating Descriptors

Creating the descriptors for a USB device can be one of the most challenging parts of USB development. Especially for HID type devices. HID type devices require the addition of a report descriptor to the standard descriptors outlined in Chapter 9 of the USB specification. Microchip provides a series of technical briefs that help with understanding descriptors (TB054, TB055, TB056, TB057, and TB058.)

Learning to write descriptors is best done by studying examples. Microchip provides numerous USB device examples and their respective descriptor sets on its webpage: www.microchip.com

Before developing the functionality for a device, Microchip recommends writing and testing the descriptors for that device. This is done by replacing the descriptors in the Cursor in a Circle example with a new set of descriptors and then testing the viability of these descriptors only. For instance, if a developer wants to create a USB joystick, the descriptors should be created first. Then the Cursor in a Circle descriptors should be replaced by the joystick descriptors. Next, the main routine of the Cursor in a Circle example should be modified to send dummy data for the joystick. The outcome of this will tell the developer whether the device enumerates and whether the joystick data is being communicated appropriately.

4.11 Assembly ISR Considerations

These considerations should be taken into account when writing an Interrupt Service Routine in Microchip Assembly: Save W, STATUS, FSR and PCLATH, which are the file registers that may be corrupted by servicing the USB interrupt. The 'Cursor in a Circle' example provides a skeleton ISR which does this, and includes tests for several other ISR bits.

4.12 Enumeration and Timing Considerations

As mentioned previously in Section 3.6.1, the InitUSB function prompts the host to begin the enumeration process. Also mentioned earlier, in order for the microcontroller to service the USB peripheral properly during enumeration, (and thereafter) ServiceUSB must be called at regular intervals. This brings to bear some timing considerations. The USB specification, section 9.2.6, gives several different timing requirements for processing requests. Nominally the PICmicro microcontroller has up to 50 ms to process most requests. Some commands are allowed to take longer, but 50 ms is a good compromise for polling ServiceUSB. Polling ServiceUSB faster than 50 ms will improve the speed of enumeration, but is not necessary.

Some requests must be serviced quickly, and thus these are processed via the interrupt. These requests are: Rest, Resume (Activity), and the last part of Set Address. The PICmicro microcontroller must quickly react to Reset and Activity flags, as Setup Transactions could follow immediately and the PICmicro microcontroller must be ready to accept and respond to these transactions. Similarly, the new address must be set within 2 ms of the ACK in response to the zero length packet.

4.13 Sending and Receiving Data

Sending data to and receiving data from the host computer is made simple with the GetEPn and PutEPn functions. These functions are described in Section 3.6.1. These functions setup/fill the registers in the Buffer Descriptor Table appropriately, according to the endpoint involved with a particular transfer (refer to section 10.6 in the PIC16C745/765 data sheet). As the Buffer Descriptor Table registers are initialized at the time the PICmicro microcontroller receives the Set_Configuration request, calling GetEPn or PutEPn before the device is configured could have disastrous results. Using the BDT for Endpoint 1 or 2 before they are initialized results in random RAM locations being overwritten. As a result, the ConfiguredUSB macro should be used to determine if the device is configured. When ConfiguredUSB returns a one, the device is configured and it is safe to use USB GetEPn and PutEPn.

4.14 Optimizing the Firmware

This firmware has been created to provide developers with ready-made USB functions so they don't have to create these functions for themselves. Most developers will not utilize all of the functions in the Ch9 firmware. In order to optimize the program memory, unused functions can be taken out of the firmware. The following guidelines are a good place to start the optimization.

4.14.1 Converting/Removing GetEP1, GetEP2, PutEP1 and PutEP2

In the assembly version of the firmware GetEP1, GetEP2, PutEP1, and PutEP2, all macros are defined in usb_defs.inc. Instances of each of these macros occur in usb_ch9.asm. If a developer does not utilize one or more of these functions, space can be saved by removing the instance(s) not needed from usb_ch9.asm.

In the Hi-Tech C version of the firmware, GetEP1 and PutEP1 are functions found in usb_ch9.c. If a developer does not utilize one or both of these functions, program memory can be saved by removing, or commenting out the function(s) not needed. A developer wanting to use GetEP2 and/or PutEP2 will need to create these functions for themselves. This is done by simply copying GetEP1 or PutEP1 and then replacing the Buffer Descriptor Table registers for EP1 with the corresponding Buffer Descriptor Table registers for EP2.

4.14.2 Optimizing the Code if Non-HID

The HID class is one of several classes suitable for low-speed USB. In addition to these other classes, a vendor-defined class can be specified. Should a developer use a class other than the HID class, any HID class specific code in the firmware would be wasting space. In the assembly version, the HID class specific code is found in the file hidclass.asm. In a case where the HID class is not being utilized by a developer, this file and any variables, or labels associated with it should be removed from the project. In the C version, the HID class specific code is found at the end of usb_ch9asm after the default mnemonic.

4.14.3 Removing Error Counting and LED Status Bits

Both versions of the firmware output the status of USB communication on Port B. This feature is intended for use with the PICDEM USB circuit board which drives an LED with each PORTB pin. The LEDs indicate the following USB status information:

- RB0 - powered
- RB1 - default
- RB2 - addressed
- RB3 - configured
- RB4 - sleeping
- RB5 - EP0 active
- RB6 - EP1 active
- RB7 - EP2 active.

These USB status indicators will probably not be used in a finished product by a developer although they are very helpful during development. All code associated with the USB status LEDs can be eliminated from the program memory by ensuring that SHOW_ENUM_STATUS is not defined at the top of usb_ch9.asm (usb.h in the Hi-Tech C version.)

Similar to the USB status LEDs, code exists in the firmware that counts various errors for debugging purposes. To eliminate this excess code from the program memory, simply make sure that COUNTERRORS is not defined at the top of usb_ch9.asm.

4.15 References

- USB Specification, version 1.1
- HID Class definition, version 1.1
- PIC16C745/765 Rev. A1 Errata
- PIC16C745/765 Rev. A2 Errata
- Microchip Technical Brief TB054
- Microchip Technical Brief TB055
- Microchip Technical Brief TB056
- Microchip Technical Brief TB057
- Microchip Technical Brief TB058
- **USB Complete**, by Jan Axelson (www.lvr.com)

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-4338

Atlanta
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1315, 13/F, Shenzhen Kerry Centre,
Renminnan Lu
Shenzhen 518001, China
Tel: 86-755-82350361 Fax: 86-755-82366086

China - Hong Kong SAR
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology (Barbados) Inc., Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

10/18/02