SILICON LABORATORIES

AN167

USB INTERRUPT EXAMPLE FIRMWARE- REFERENCE MANUAL

Relevant Devices

This application note applies to the following devices:
C8051F320, C8051F321.

Introduction

This application note covers the implementation of a simple USB application using the interrupt transfer type. This includes support for device enumeration, control and interrupt transactions, and definitions of descriptor data. The purpose of this software is to give a simple working example of an interrupt transfer application; it does not include support for multiple configurations, or other transfer types.

Overview by File Name

The firmware includes three header files and four source files. USB_MAIN.h consists of program constants, definitions, and function prototypes. The firmware can run in either low or full-speed, depending on whether _USB_LOW_SPEED_ is defined in this header file. USB_DESCRIPTOR.h contains the various structures used for the different types of descriptors. USB_REGISTER.h contains addresses for the USB core registers, and register access macros used throughout the software.

USB_MAIN.c - This file contains the main routine and some initialization subroutines. The main routine calls all initialization routines, then starts an infinite loop of updating the OUT packet with information received from the host, and the IN packet with information from the development boards peripherals. This file contains two interrupt service routines (ISR). The first ISR is called when timer two overflows and checks to see if a switch has been pressed or released. The second ISR is called when the analog to digital converter (ADC) finishes a conversion. This ISR switches the ADC connection between the potentiometer wheel and the temperature sensor, allowing for each to updated on every second conversion.

USB_DESCRIPTOR.c - This file contains the definitions of the actual descriptors used by the application. This includes a single device descriptor, configuration descriptor and interface descriptor. Since this firmware uses different endpoints for input and output transactions, there are two endpoint descriptors defined, one for each direction. There are also three string descriptors defining the string language, manufacturer, and product type respectively.

USB_ISR.c - The top-level USB ISR is located here. This is called when any USB type event occurs, and calls the appropriate event handler between reset (USBReset), incoming setup packet (HandleSetup), incoming data packet (HandleIN1), and out packet sent (HandleOUT2). The last three routines are responsible for actually reading and writing from the different endpoints. The USB_State variable used in this file contains the current USB state of the device, and the EpStatus array contains the current state of the 3 endpoints used.

USB_STD_REQ.c - All USB device requests, as defined in chapter 9 of USB specification, are handled by routines in this file. All of these have been made as simple as possible, and return responses specifically for this software implementation and which uses only one interface and one configuration. These routines all begin by checking the setup packet and verifying the instruction matches **exactly** as specified by the USB specification, otherwise the software returns a stall packet to the host.

Making Modifications

Although this software has been provided strictly as an example of a USB type application, it can be used in to some extent in a variety of applications.

To use this software for a similar application using the interrupt transfer type, very few changes are necessary. The main routine in USB_MAIN.c will need to be updated according to the type of data being used. Also, the packet size definitions in USB_MAIN.h should be modified for your particular application, the sample application uses 64 and 8 byte endpoint zero packet sizes in full and low speed, and 8 byte packets on endpoints one and two. The polling interval on the packets can be changed using the bInterval field in the endpoint descriptors in USB_DESCIPTOR.c.

Using this software for other transfer types requires two in addition to those described above. First, the endpoint descriptors in USB_DESCIPTOR.c will need to be updated to reflect the transfer type you choose. This will depend on the number and type of endpoints required. Also, you need to write IN and OUT routines specifically for your application. These will be similar to the routines given in USB_ISR.c.

Multiple configurations and multiple interfaces can be added to this software by adding the necessary descriptors to USB_DESCRIPTOR.c and possibly USB_DESCRIPTOR.h. After adding the necessary descriptors, you should update the interface and configuration routines in USB_STD_REQ.c as they currently only support one configuration and interface.

To reduce the current size of the application, there are a few areas that can be cut to easily reduce program size. First, the string descriptors for iManufacturer and iProduct can be removed in most cases. Also, the standard request routines currently check all fields of the setup packet for validity, while it is possible to check only some of these fields and still be considered compliant to the USB Specification. Most of these checks could be removed to reduce code size while still passing the current revision of the compliance software, however this is not guaranteed for future revisions.

Notes:

Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: productinfo@silabs.com
Internet: www.silabs.com