Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 001-63788

Spec Title: CYUSB.SYS DRIVER FOR EZ-USB(R) - AN63788

Sunset Owner: GAYATHRI VASUDEVAN (GAYA)

Replaced by: NONE

CyUSB.sys Driver for EZ-USB®

Author: Anand Srinivasan
Associated Project: No
Associated Part Family: EZ-USB® (AN21xx/FX/FX1/FX2/FX2LP)
Software Version: None
Related Application Notes: AN61465, AN50963

Driver development for EZ-USB can be divided into two parts: the OS-specific part and the EZ-USB (hardware)-specific part. AN63788 describes CyUSB.sys such that it can be used as a reference while developing a driver for EZ-USB.

Introduction

A USB driver exposes an interface to higher-level software through which a host can communicate with a particular USB device. The functionality and features of the driver depend on both the OS and the device. The development of a driver such as CyUSB.sys also involves hardware-specific support to enable specific device functionality (that is, the USB firmware download mechanism of EZ-USB). The OS-specific dependencies involve how the interface between the native OS USB stack and driver code is defined.

Overview

This application note provides a high-level view of the OS-specific and EZ-USB-specific portions of CyUSB.sys so that you can successfully develop a driver similar to CyUSB.sys.

The question that arises is why CyUSB.sys cannot just be ported to another OS. Developing a non-windows driver is not as simple as porting CyUSB.sys code from one OS to another. The driver-level interfaces vary with each OS depending on the level of support for USB native to the OS. In the case of Windows, too, there are different driver models and a change from one model to another demands code change. The process of porting is neither straightforward nor simple.

OS-specific Part

The OS-specific part of CyUSB.sys is discussed in terms of windows. This can be used as reference to understand the possible implementation methodologies in the target OS.

Loading the Driver

Binding Device with Driver

In Windows the *inf* file is used to bind the driver to the device. For more details on the *inf* file, refer to the application note, Working With inf File of a Device Using CyUSB.sys - AN61465.

Passing Data

In Windows, OS data is stored in the registry. CyUSB.sys and the host application extract certain data from the registry. For example, in the case of script-loading, the *inf* file registers the location of the script file in the registry and CyUSB.sys extracts this and loads it on the device.

Driver Control Interface

CyUSB.sys exposes non-standard function calls through an input/output control (IOCTL) interface. The application layer sends IOCTLs using the Windows API DeviceIoControl() to communicate with CyUSB.sys (See the Windows SDK documentation for details about DeviceIoControl). For more information on specific IOCTLs implemented in CyUSB.sys, refer to '*CyUSB.pdf*' located at *C:\Program Files\Cypress\Cypress Suite USB 3.4.2\Driver* (the location may vary based on installation). This comes as part of SuiteUSB 3.4 - USB Development tools for Visual Studio.

Driver Communication

Communication with OS

The USB driver communicates with the Plug 'n Play (PnP) manager, power manager, I/O manager, and host controller driver. The communication happens through I/O request packets (IRPs) and USB request blocks (URBs). While URBs are used to communicate with the host controller driver, IRPs are used to communicate with the OS. Not all IRPs can be completely handled by CyUSB.sys. In certain cases, IRP handling depends on the lower level driver as well. In these cases the CyUSB.sys handles part of the IRP and then passes it down to the lower level driver. In certain cases, the lower-level driver handles its part of the response and passes the response to the CyUSB.sys. In these cases a callback function is specified when the IRP is passed down. The callback function handles the completion of the request when the IRP moves back up.

Data Management

The host controller, OS, and device information are required for the proper working of the USB driver. Therefore, the CyUSB.sys creates appropriate buffers and stores these data (speed of enumeration, descriptor, host controller version, OS version, and so on). The buffer for data transfer is allocated in the host controller driver. The buffers for IRP and other parameters passing are not always required. Therefore, CyUSB.sys creates and destroys them if required. Note that because these buffers come from a common pool of resources, the driver has to lock access to avoid corruption issues.

PnP Manager

The communication with PnP manager involves:

- IRPs that query the readiness of the driver to handle device removal (software: ejecting of device as well as hardware: unplugging of device)
- IRPs that notify the driver about the removal of the device.

I/O Manager

The communication with the I/O manager primarily involves communication from the host application. Here, the driver parses the parameters sent through IOCTLs and converts them to appropriate URB parameters in order to send the request to the host controller driver. In certain cases CyUSB.sys has the data stored in its buffer (for example, the speed of enumeration may be stored in CyUSB.sys) and it responds directly. In certain special cases like remote wakeup CyUSB.sys generates the IRP and sends it to the I/O manager.

Power Manager

Power management states are defined based on the amount of power consumed. The device power states are denoted with a prefix 'D' followed by a number and system with a prefix 'S' followed by a number. The power consumption is inversely proportional to the number used to denote the state. Windows usually transitions to the lowest power state supported by all the enabled wake-up devices (devices that are capable of signaling wake-up and are enabled to do so by the OS). Windows broadcasts requesting permission from all the applications for power state transition. If none of the applications denies the request, then it broadcasts the transition to drivers. If any driver denies the request, then the next highest sleep state is requested. This is done until a power state is entered successfully or there are no more available sleep states. This mainly involves storing certain variables and turning off certain services based on the power state transition. The power management implementation methodology and the abstraction level (driver, lower level driver, bus driver etc) at which this is handled will depend on the target OS. Handling this properly in Windows is critical as it affects the power state of the entire system. CyUSB.sys handles this properly and has support till power state S4 (hibernation).

The communication with the power manager involves

- IRPs that query the power capabilities and current power state
- IRPs that notify the driver of a power state change (device and system).

Communication with Host Controller Driver

CyUSB.sys supports all four types of USB transfers and can generate certain bus conditions such as suspend, bus reset, and so on. The parameters and electrical conditions of these are dictated by the USB protocol. In Windows, URBs with appropriate parameters are called to implement these. The execution flow primarily consists of getting parameters from the host application and translating them to URB parameters, and then calling appropriate URBs to communicate this to the host controller driver.

Error Handling

Because the driver uses a common pool of resources, it constantly validates the parameters/buffers that have been passed to it before it actually processes the request. This part comes at the top of the execution flow.

Script File

The script file contains the USB transfer parameters and data encoded in a Cypress-specific format. While developing for the target OS, the format (of the analogy) of script file depends on the driver because the driver decodes the script file to send appropriate USB transfers. Therefore, knowledge of this Cypress-specific format is not required and a custom encoding/decoding algorithm can be implemented.

EZ-USB Specific Part

The enumeration sequence and descriptor of a USB device are specified by the USB specification. Therefore, CyUSB.sys as a generic USB driver is able to work with any generic USB device. However, for the application/hardware specific functionalities that are built over USB protocol, support might be required from the

driver. Script loading is one such hardware specific functionality in the case of EZ-USB bound to CyUSB.sys.

Script Loading

EZ-USB supports an A0 vendor request (bRequest 0xA0) through which the internal RAM of EZ-USB can be accessed. Here, the address to be written to is specified by wValue. This vendor command is valid only while the 8051 is held in reset. The 8051 is reset by sending the A0 vendor command to the CPU control and status (CPUCS) register (0xE600 in the case of FX1/FX2/FX2LP and 0x7F92 in the case of AN21xx/FX) with data 0x01. The 8051 is brought out of reset by sending the A0 vendor command to the CPUCS register with data 0x00. This enables downloading firmware to EZ-USB through the USB interface and, as a result, the possibility to store firmware in the host and bootload EZ-USB from the host. The implementation methodology of this functionality is called script loading. For details on generation and use of script for downloading firmware refer to AN50963.

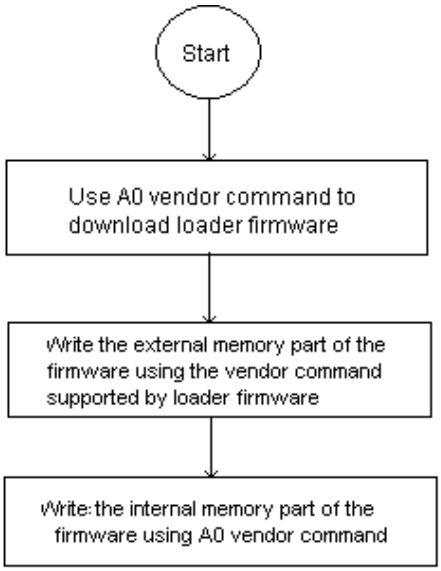
Note The location of CPUCS register is hardcoded as 0xE600 in CyConsole and, therefore, it cannot be used to generate script for AN21xx/FX. In those cases CyScript is found in C:\Cypress\USB\bin\ (location may vary based on installation) after installing CY3684 EZ-USB FX2LP Development Kit. After opening CyScript, pressing Ctrl+P opens a pop-up window through which the CPUCS register and maximum internal memory address of the target device can be specified.

In cases where part of the firmware resides in the external memory, the host does not have a method to directly access external memory. The following section describes the method used to overcome this restriction and Figure 1 illustrates the flow chart.

Firmware Download Algorithm

1. Download a loader firmware to the internal memory which can process a vendor request to access external memory.
2. Write the part of firmware that resides in external memory by sending vendor requests to the loader firmware
3. Overwrite the loader firmware with the internal memory part of the firmware.

Figure 1. Firmware Download Algorithm Flow Chart



The algorithm to overcome this restriction can be handled in the script file or the driver. It can be handled in the driver in the following ways:

1. Driver stores the loader firmware as part of its code.
2. It extracts the firmware from the script file.
3. Separate the internal and external memory part of the firmware.
4. Then follow the firmware download algorithm.

It can be handled in the script file in the following ways:

1. The script file stores data in the sequence in which they are to be sent when following the firmware download algorithm.
2. The driver decodes the script file and sends USB requests accordingly.

The script file has a method of storing USB transfer parameters and data. Therefore, handling this in the script file adds few more USB transfer parameters and data. Handling in the script file is faster and less complex. In CyUSB.sys, the algorithm is handled in the script file. The loader firmware used can be vend_ax or A3load example available in C:\Cypress\USB\Examples\Device\vend_ax and C:\Cypress\USB\Examples\Device\A3load respectively. The 0xA3 vendor command is used to access the external memory of EZ-USB. The device is EZ-USB (does not have A3load) or FX or FX1 or FX2 or FX2LP based on the target EZ-USB device.

Summary

This document describes the OS-specific and EZ-USB specific parts of the EZ-USB driver and how to implement them successfully.

About the Author

Name: Anand Srinivasan
Title: Applications Engineer Sr
Contact: aasi@cypress.com

Document History

Document Title: CyUSB.sys Driver for EZ-USB® - AN63788

Document Number: 001-63788

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3015497	AASI	08/25/2010	New application note.
*A	3711250	GAYA	08/13/2012	Updated template.
*B	4460861	RSKV	07/30/2014	Obsolete document. Completing Sunset Review.

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

Technical Support

cypress.com/go/support

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

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