Getting Started with FX2LP™

About this document

Scope and purpose

AN65209 introduces you to the EZ-USB® FX2LP™ USB 2.0 device controller. This application note helps you build a project for FX2LP and explore its various development tools, and then guides you to the appropriate documentation to accelerate in-depth learning about FX2LP.

Associated part family

CY7C68013A/14A/15A/16A/53

Associated project

Yes

Software version

None

More code examples? We heard you.

To access a variety of FX2LP code examples, please visit our **USB High-Speed Code Examples webpage**.

To access USB 3.0 product family, please visit our **USB 3.0 Product Family webpage**.

Table of contents

About this document **1**
Table of contents **1**
1 Introduction **3**
1.1 Bulkloop_FX2LP Project 3
1.1.1 VS_Control Center 3
1.1.2 Bulkloop_VCS 3
1.1.3 Windows Driver and Library 3
2 USB 2.0 ..... **4**
3 FX2LP Introduction **5**
3.1 FX2LP Features 5
3.1.1 USB Interface 5
3.1.2 Parallel interfaces 6
3.1.2.1 Slave FIFO 6
3.1.2.2 General Programmable Interface 7
3.2 Serial interfaces 8
3.3 CPU and Memory 8
3.3.1 8051 8
3.3.2 Boot Options 8
3.3.2.1 Package Choices 8

Table of contents

3.4 Example Applications of FX2LP 10
3.4.1 Interfacing FPGA/ASIC using Slave FIFO 10
3.4.2 Further Reading 10
3.4.3 Booting an FPGA from FX2LP 11
3.4.4 Further Reading 11
4 Cypress Design Resources 12
5 FX2LP Development Tools 14
5.1 FX2LP Development Board 15
5.2 Running the Bulkloop Demo 16
5.2.1 Using USB Control Center 16
5.2.2 Using BULKLOOP_VCS 19
5.3 FX2LP Firmware Development 20
5.3.1 Keil uVision2 20
5.3.2 Cypress USB Firmware Frameworks 20
5.3.2.1 TD_Init 22
5.3.2.2 TD_Poll 22
5.3.2.3 Interrupt Service Routines 23
5.3.2.4 Handling USB Dual Speeds 23
5.3.3 Building the Bulkloop Project 24
5.4 GPIF Designer 25
6 The Windows Side 27
6.1 Cypress USB Driver 27
6.1.1 Driver/Library Alternatives 27
6.2 Cypress Libraries 28
7 Summary 30
8 Appendix A: FX2LP Development Kit (DVK) 31
8.1 Firmware Example Projects 31
9 Appendix B: Additional USB Hi-Speed Devices from Cypress 34
10 Appendix C: Third-Party Development Kits and SDKs 35
10.1 Third-party SDKs 35
11 Appendix D: Application Notes and Reference Designs 36
11.1 Application Notes 36
12 Appendix E: Adding Custom VID and PID to the *.inf* File 39
Revision history 41

1 Introduction

The Cypress EZ-USB FX2LP (hereafter abbreviated as FX2LP) is a flexible USB 2.0 PERIPHERAL controller designed to handle maximum USB 2.0 bandwidth. To take full advantage of the USB 2.0 480 Mbps signaling rate, FX2LP contains specialized hardware to buffer the USB data and connect seamlessly to a variety of high-bandwidth external devices such as MCUs, ASICs, and FPGAs. After a brief introduction to USB 2.0, this application note describes FX2LP features that contribute to its high throughput.

Any IC is only as good as the tools that support it, so the remaining sections of this application note introduce the **FX2LP Development Kit** (DVK) and the FX2LP firmware and software development tools. A companion zip file contains folders with code samples and an application is described in detail in this application note. Briefly, they are as follows:

1.1 Bulkloop_FX2LP Project

The Bulkloop project attached to this application note shows how to structure the FX2LP firmware to create a USB device. Much of the USB request handling is done by a Cypress-provided USB Firmware Framework, with the user code required only for the specific application requirements. The *bulkloop.c* file contains a full USB device template that can serve as the basis for a custom application.

Cypress tools include a free evaluation version of Keil uVision2 for the FX2LP CPU (8051). The Keil uVision2 Integrated Development Environment (IDE) compiles the user code and USB Firmware Framework to produce the hex file. This hex file needs to be loaded by the provided **USB Control Center** application and exercised by the provided **Bulkloop_VCS** application.

As part of the firmware tools discussion, the GPIF Designer program is introduced to show how custom interfaces are created using graphical waveform entry (see **GPIF Designer**).

1.1.1 VS_Control Center

This is a Microsoft Visual Studio solution, written in Visual C#, that creates an application called **USB Control Center**. This application is used to download the FX2LP code (a hex file) into the FX2LP Development Kit. The executable is provided for instant use and the source code is provided for reference. It uses the Cypress .NET library, demonstrating the use of many of its high-level functions.

1.1.2 Bulkloop_VCS

This is a simple Windows Visual Studio application written in Visual C#. It features continuous data looping and byte counting that can be used to measure USB transfer bandwidths with various USB controller cards and computers. As with the USB Control Center, both the source code and executable are provided.

1.1.3 Windows Driver and Library

Cypress provides a Windows driver to support FX2LP-based designs. It supports all USB transfer types, and is available in binary form for use with customer products.

The USB Control Center and Bulkloop_VCS applications use **CyUSB.dll**, a managed Microsoft .NET class library provided by Cypress. This makes it compatible with the Visual Basic.NET, Visual C#, Visual C++ (Win Forms), and Visual J# languages. This library provides high-level calls that hide driver complexity and present a very simple USB programming model.

2 USB 2.0

USB standardizes the connection of computer peripherals, such as keyboards, mice, printers, thumb drives, hard disks, and portable media players. USB provides communication and power to peripheral devices. USB has become the dominant connectivity solution for PCs and consumer devices. Its popularity is largely due to its ease of use, achieved by its standardized and robust underlying structure.

The USB 1.0 specification, released in 1996, defined two transfer speeds to address common device types of the time. Low-speed devices operate at 1.5 Mbps to support devices such as keyboards and joysticks. Full-speed devices operate at 12 Mbps to support higher bandwidth devices, such as printers and disk drives. A minor specification revision 1.1, released in 1998, mostly addressed hub issues and became the widely adopted first-generation specification.

The USB 2.0 specification was released in 2000. It increased the signaling rate to 480 Mbps, naming it Hi-Speed. The 2.0 specification is compatible with the previous transfer rates.

The USB 3.0 specification, released in 2008, further increased the signaling rate to 5 Gbps, calling it SuperSpeed. The Cypress FX3 family supports this speed.

The subject of this application note is the Cypress FX2LP, which operates at Hi-Speed and Full-speed. **Appendix B** lists other Hi-Speed devices available from Cypress. All USB controllers from Cypress are listed **here**. For more details of USB 2.0, refer to **USB 101: An Introduction to Universal Serial Bus 2.0**.

3 FX2LP Introduction

FX2LP integrates a USB 2.0 transceiver, a smart serial interface engine (SIE), large data buffers, an enhanced 8051 microcontroller, and a programmable peripheral interface in a single chip. A simplified FX2LP block diagram is shown in **Figure 1**.

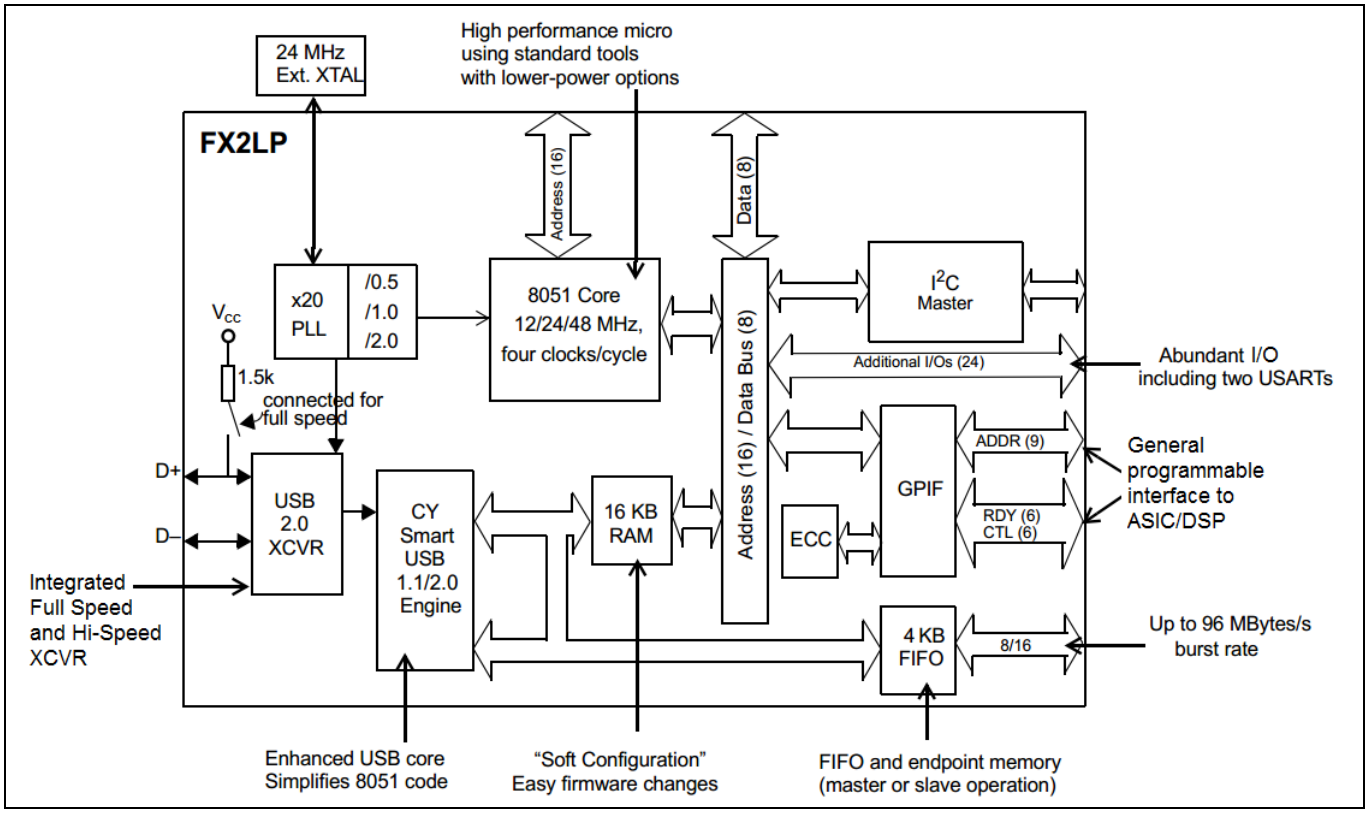


Figure 1 FX2LP Block Diagram

The function of FX2LP is to transfer data between a USB host and a peripheral device. FX2LP provides this connectivity using parallel and serial interfaces. In systems where the FX2LP CPU is not required to modify the data before sending it to the USB host, the FX2LP firmware only needs to initialize its hardware transfer units, allowing high-bandwidth USB transfers to proceed without CPU intervention.

FX2LP can be programmed to enumerate as any USB device type, conforming to a standard USB class, such as HID or mass storage, or a custom device.

3.1 FX2LP Features

This section briefly describes the key features of FX2LP.

3.1.1 USB Interface

Hi-Speed USB transfers can be visualized as supporting two ends of a pipe. On one end, the pipe receives and sends data at 480 Mbps over the USB interface, as shown in **Figure 2.** The other end of the pipe is described in **Parallel interfaces**.

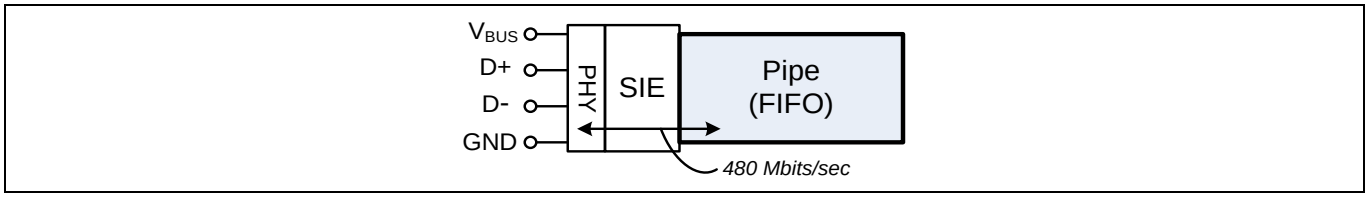


Figure 2 USB Connection to a Data Pipe

To handle the USB host side, FX2LP includes:

- A physical bus interface (PHY) containing a USB 2.0 transceiver that supports Hi-Speed and Full Speed USB transfers.
- A Smart serial interface engine (SIE). The SIE translates PHY signals into bytes. All Cypress USB SIEs add logic to handle low-level USB details, such as error correction and Packet ID (PID) synchronization, relieving the programmer of these tasks. This Smart SIE delivers bytes to an endpoint FIFO memory. The Smart SIE also contains logic to enumerate FX2LP as a full-fledged USB device and also capable of loading code into its internal RAM.
- Seven endpoints that support all four USB transfer types: CONTROL, BULK, INTERRUPT, and ISOCHRONOUS. An endpoint is a transmitter or receiver of USB data.
- Large endpoint buffers (FIFOs) with multiple buffering. Double, triple, or quad buffering allows USB transfers to be pipelined with a peripheral, increasing the throughput.
- Low-power operation (the “LP” in FX2LP). The USB VBUS wire supplies 5 V, which provides limited power to peripheral devices. The low-power consumption of FX2LP enables you to create bus-powered applications. For example, the FX2LP Development Kit is bus-powered, eliminating the need for an external power unit. FX2LP can also be used in self-powered designs in which the peripheral device supplies its own power.

3.1.2 Parallel interfaces

The USB 2.0 specification and generous FX2LP buffering take care of transporting bytes in and out of endpoint FIFOs at high speed over the USB interface, but that is only half of the job. The other end of the pipe must transport the FIFO data on and off chip at speeds matching USB transfer rates. FX2LP contains two hardware interfaces specifically designed for this purpose: Slave FIFO and general programmable interface (GPIF), as shown in **Figure 1**.

Note: In addition to the high-speed transfer logic, the FX2LP 8051 has random access to the endpoint FIFOs for applications that require interpreting or modifying the data that is moving between the USB and peripheral interfaces.

3.1.2.1 Slave FIFO

FX2LP provides a Slave FIFO interface for use by external devices containing a FIFO controller, such as an MCU, FPGA, or ASIC (see **Figure 3**).

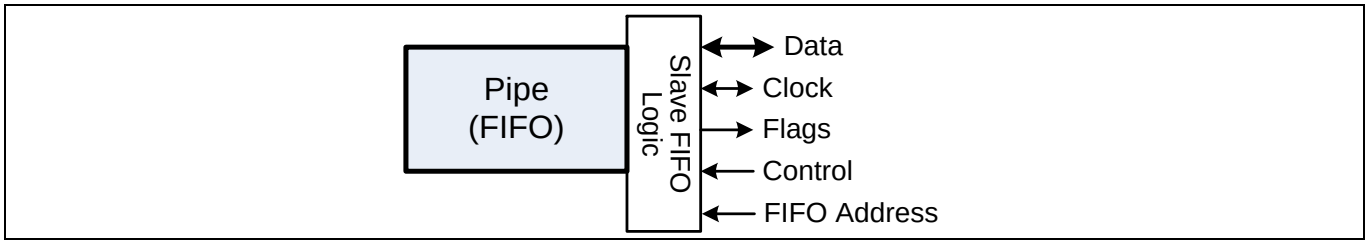


Figure 3 FX2LP Slave FIFO Interface

The Slave FIFO interface:

- Provides a data bus that is 8 or 16 bits wide.
- Operates either asynchronously (no clock) or synchronously (with clock).
- Accepts an external clock or uses an internal FX2LP 30-/48-MHz clock (in synchronous mode), which it makes available on the CLKOUT pin. There is no need of crystal for the external peripheral device if it can work based on the clock provided on the CLKOUT pin.
- Has output flags indicating FIFO status, such as full and empty.
- Has control inputs OE#, RD#, and WR#.
- Has two address lines to select one of four FX2LP FIFOs.
- Automatically launches USB transfers when the FIFO fills or empties.
- Has a control signal (PKETND) that the interface device can use to launch a “short packet” (FIFO has data but is not full). Short packets routinely occur as the final packet of a long USB transfer.

In systems where data processing is not required in FX2LP, the FX2LP Slave FIFO interface requires minimal firmware just to select the Slave FIFO interface and to configure the flags to indicate the status of the FIFOs. Waveforms and timing values for the Slave FIFO interface are given in the **FX2LP datasheet**.

3.1.2.2 General Programmable Interface

Not all external controllers are designed to connect to a FIFO. Therefore, FX2LP provides a high-speed interface, GPIF (shown in **Figure 4**), which provides direct connection to common interfaces, such as disk drives, FPGAs, and ASICs, without requiring additional glue logic. The GPIF’s core is a state machine—you must develop waveforms using **GPIF Designer** to control the state machine. The GPIF is driven by one of the four Waveform Descriptors, which are data structures containing all the waveform information. GPIF Designer relieves the designer of understanding the descriptor formats. It uses the graphical waveform entry to create a C-language source file, which can be included in an FX2LP project.

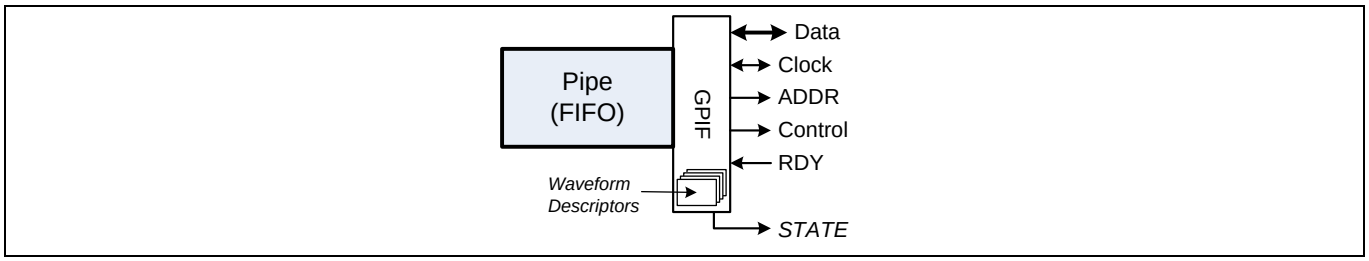


Figure 4 FX2LP GPIF Interface

The GPIF (see the **GPIF Designer** section):

- Provides a data bus that is 8 or 16 bits wide.
- Operates synchronously (with clock).
- Accepts an external clock or uses an internal FX2LP 30-/48-MHz clock (in synchronous mode), which it makes available on a device pin.
- Provides nine address outputs. Addresses can be initialized and incremented on a clock-by-clock basis.
- Provides six control (CTL) and ready (RDY) signals. Six control outputs are also programmable on a clock-by-clock basis. As a simple example, RD# and WR# strobes can be created with programmable durations and polarities using these control signals. Six ready inputs can be tested in a GPIF state machine to read status and perform interface synchronization. Three state outputs (GSTATE [2:0]) indicate the GPIF state, useful for debugging with a logic analyzer.

3.2 Serial interfaces

- In addition to the USB and peripheral interfaces, FX2LP contains:
- An I^2C master (only) which operates at 100 or 400 kHz.
- Two standard 8051 USARTs. These are standard UARTs with a synchronous option. The USART interface pins are available on separate I/O pins, not multiplexed with GPIO pins.
- Up to 40 GPIOs. These pins are multi-purpose, serving as GPIO pins or pins that support the Slave FIFO or GPIF interfaces. The **Package Choices** section illustrates these interface options.

3.3 CPU and Memory

3.3.1 8051

- FX2LP has an 8051 core with two USARTs, three counter/timers, and an enhanced interrupt system. The core can use a 48 MHz, 24 MHz, or 12 MHz clock. The CPU is supported by 16KB of on-chip code/data RAM.
- The enhanced interrupt system uses an “Autovector” mechanism to automatically call one of the 27 USB interrupt service routines (ISRs), depending on the USB activity that requires service. Automatically incrementing pointer hardware (“Autopointers”) speed up block transfers.

3.3.2 Boot Options

- FX2LP uses RAM for program storage. FX2LP has the following boot options:
  - USB boot.
  - I^2C boot.
  - Boot from external parallel memory.
- When connected to USB, its Smart SIE enumerates as a USB bootloader capable of loading program code into its internal RAM. After the code is loaded, FX2LP electrically disconnects itself from USB and immediately reconnects as the device defined by the downloaded code. This process is called Re-Numeration™.
- The FX2LP program RAM also can be loaded at power-on from an external serial EEPROM. Boot options are detailed in **AN50963 - EZ-USB® FX1™/FX2LP™ Boot Options**.

3.3.2.1 Package Choices

FX2LP is available in three packages, as **Figure 5** shows:

- 56-pin SSOP, QFN, and VFBGA.
- 100-pin TQFP package.
- 128-pin TQFP package.

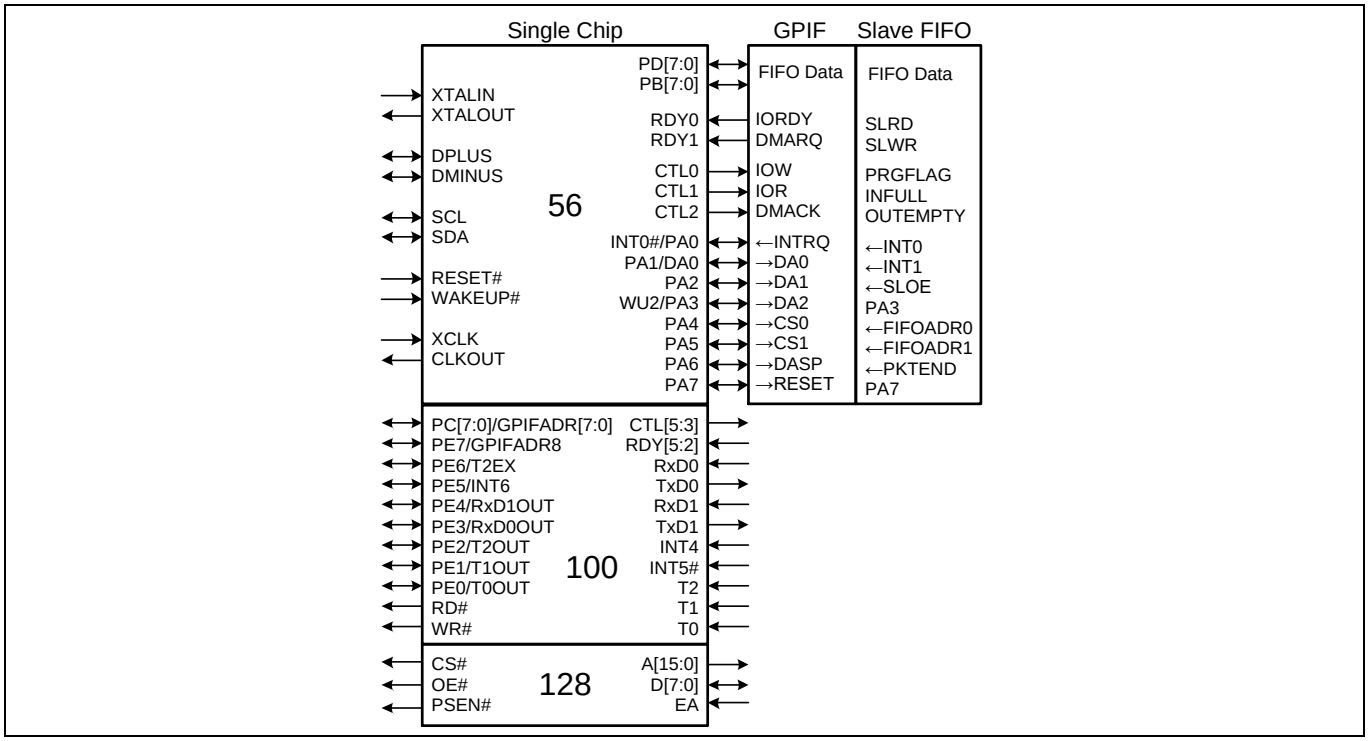


Figure 5 Three FX2LP Package Choices

The 56-pin package is the lowest-cost version of FX2LP. The signals on the left edge of the 56-pin block (in **Figure 5**) are common to all package versions in the family. Three modes are available in all package versions: single-chip, GPIF, and Slave FIFO. These modes define the signals shown on the right edge of the 56-pin block (in **Figure 5**). The 8051 selects the interface mode using an internal register. Single-chip mode is the power-on default configuration.

The 100-pin package adds functionality to the 56-pin package by adding 44 pins:

- Two additional 8-bit I/O ports, PORTC and PORTE.
- Seven additional GPIF controls (CTL) and ready (RDY) signals.
- Nine non-multiplexed control signals (two UARTs, three timer inputs, INT4, and INT5#).
- Eight additional control signals multiplexed on to PORTE.
- Nine GPIF address lines, multiplexed on to PORTC(8) and PORTE(1),
- RD# and WR# signals, which may be used as read and write strobes for PORTC.

The 128-pin package adds 8051 address and data buses and their control signals. These added pins allow FX2LP to operate with an external 8051 memory. This package is used in the FX2LP Development Board.

3.4 Example Applications of FX2LP

3.4.1 Interfacing FPGA/ASIC using Slave FIFO

In **Figure 6**, an FPGA or ASIC contains a FIFO controller that connects directly to the FX2LP Slave FIFO pins. The FPGA/ASIC also connects to hardware that is specific to the application, such as a data logger or image sensor. Although a synchronous FIFO is shown using the IFCLK signal, FX2LP also supports an asynchronous (no clock) FIFO interface.

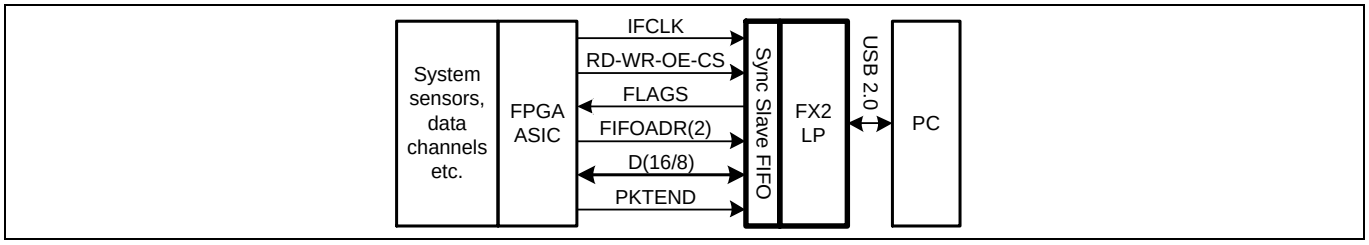


Figure 6 FPGA/ASIC Sees FX2LP as a FIFO

Further Reading

For more information about the FX2LP Slave FIFO interface, refer **AN61345 - Designing with EZ-USB FX2LP™ Slave FIFO Interface using FPGA**, which presents a detailed design example.

Refer to **KBA222479 USB2.0 Camera Interface Using FX2LP™ and Lattice CrossLink FPGA** for an example project that implements a UVC framework to interface an image sensor with the Host PC/mobile phone using the FX2LP device.

Some interface chips may provide the necessary FIFO interface signals without modification. For example, an MPEG decoder can map its signals to the FX2LP FIFO as follows:

Table 1 MPEG Decoder Connections to FX2LP

MPEG Decoder Signals	FX2LP Signals
MPEG_CLK	IFCLK
MPEG_SYNC	PKTEND#
MPEG_VALID	SLWR#
D[7:0]	FD[7:0]
External Tuner Control	I^2C bus

3.4.2 Further Reading

Cypress provides a reference design for a “TV Dongle”, documented at **FX2LP DMB-T / H TV Dongle Reference Design**.

If an external chip does not exactly map to the FX2LP Slave FIFO signals, as in **Table 1**, the FX2LP GPIF can be programmed to match the required signals without additional external logic.

3.4.3 Booting an FPGA from FX2LP

FX2LP is all about efficient system integration. **Figure 7** and **Figure 8** illustrate another way FX2LP can save external logic.

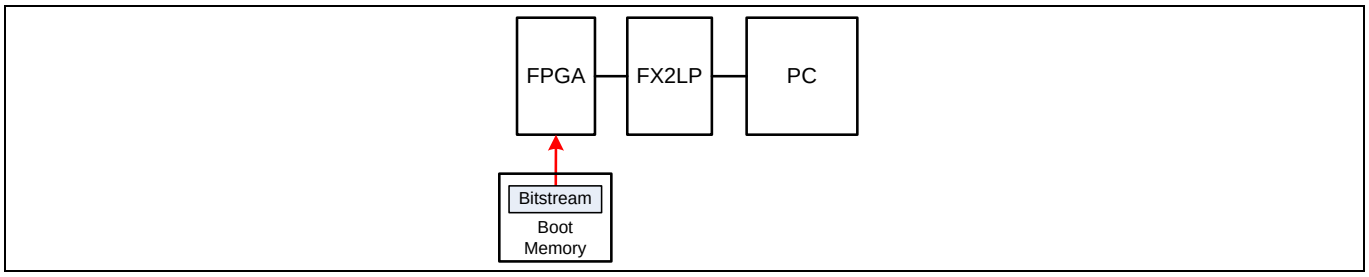


Figure 7 FPGA Boots from External Memory

At power-on, an FPGA boots its configuration bitstream using an external memory (**Figure 7**).

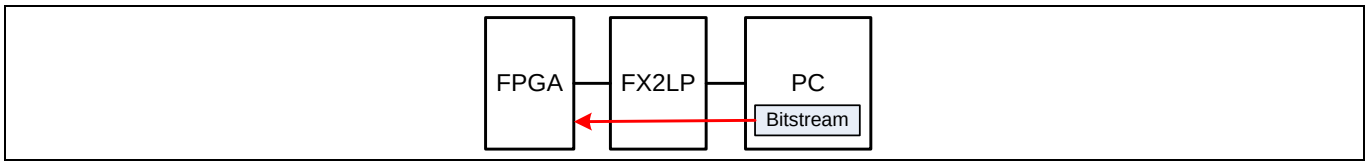


Figure 8 FPGA Boots through FX2LP

FX2LP can be configured to load the FPGA bitstream file from the PC as in **Figure 8**. This eliminates the need for a boot memory and enables FPGA updates from the PC.

3.4.4 Further Reading

For an example implementation using a Xilinx Spartan-3E FPGA, refer **AN63620 - Configuring a Xilinx Spartan-3E FPGA Over USB Using EZ-USB FX2LP™**.

4 Cypress Design Resources

Cypress FX2LP design resources include datasheets, application notes, evaluation kits, reference designs, firmware examples, and software tools. The resources are summarized in **Table 2**.

Table 2 FX2LP Resource Summary

Design	Available Resources	Where To Find Resources
Hardware	Development Board – Schematic, Board files and documentation	Development Kit (DVK) Schematic Board files available with **FX2LP DVK** installation **DVK User Guide** **DVK Quick Start Guide**
	Hardware design guidelines including recommendations for crystals, decoupling capacitors for power supplies and PCB layout	Application note – **AN15456**
	IBIS model	**http://www.cypress.com/?id=193&rtID=114**
FX2LP Firmware	Free version of Keil IDE (up to 4KB of code size)	Available with **FX2LPDVK** installation
	Firmware examples	
	Sync Slave FIFO firmware example in a complete design with FPGA	Application note - **AN61345**
Firmware Debug	Setting Up, Using, and Troubleshooting the Keil Debugger Environment	Application note - **AN42499**
	Serial (UART) Port Debugging of FX1/FX2LP Firmware	Application note – **AN58009**
Host PC Software	USB2.0 driver – cyusb.sys	Available with **Suite USB** installation. This Suite USB installation file (.exe) is also available with the **FX2LP DVK** installation. **Suite USB for Mac OS** is also available. Use **FX3 SDK for Linux** to get the host application similar to Control Center for Linux platform.
	Host application examples – Control Center and Streamer applications	
GPIF Interface Design	GPIF Designer Tool that enables you to design a GPIF waveform and generate code to be integrated into firmware	Available with **GPIF Designer** installation. The GPIF Designer installation file (.exe) is also available with the **FX2LP DVK** installation.
	Examples of popular GPIF implementations	Application notes - **AN57322 – Interfacing SRAM with FX2LP over GPIF** **AN66806 – EZ-USB® FX2LP™ GPIF Design Guide** **AN63787 – EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples using 8-bit Asynchronous Interface**
	Documentation on GPIF and instructions for using the tool	GPIF Designer's User Guide – available with GPIF Designer Tool
Other Collateral		
FX2LP Datasheet		**http://www.cypress.com/?rID=38801**

Cypress Design Resources

Design	Available Resources	Where To Find Resources
FX2LP Technical Reference Manual		**http://www.cypress.com/?rID=38232**
Application Notes		**http://www.cypress.com/?id=193&rtID=76**
Reference Designs		**http://www.cypress.com/?id=193&rtID=201**
Knowledge Base Articles		**http://www.cypress.com/?id=193&rtID=118**
Material on USB 2.0		**http://www.beyondlogic.org/usbnutshell/usb1.shtml** **AN57294 - USB 101: An Introduction to Universal Serial Bus 2.0**
Third-party Development Kits		**http://www.ztex.de/usb-fpga-1** **http://www.opalkelly.com/products/xem6010/** Features of these development kits are given in **Appendix C: Third-Party Development Kits and SDKs**.

5 FX2LP Development Tools

The steps involved in developing and testing the FX2LP firmware are shown in **Figure 9**.

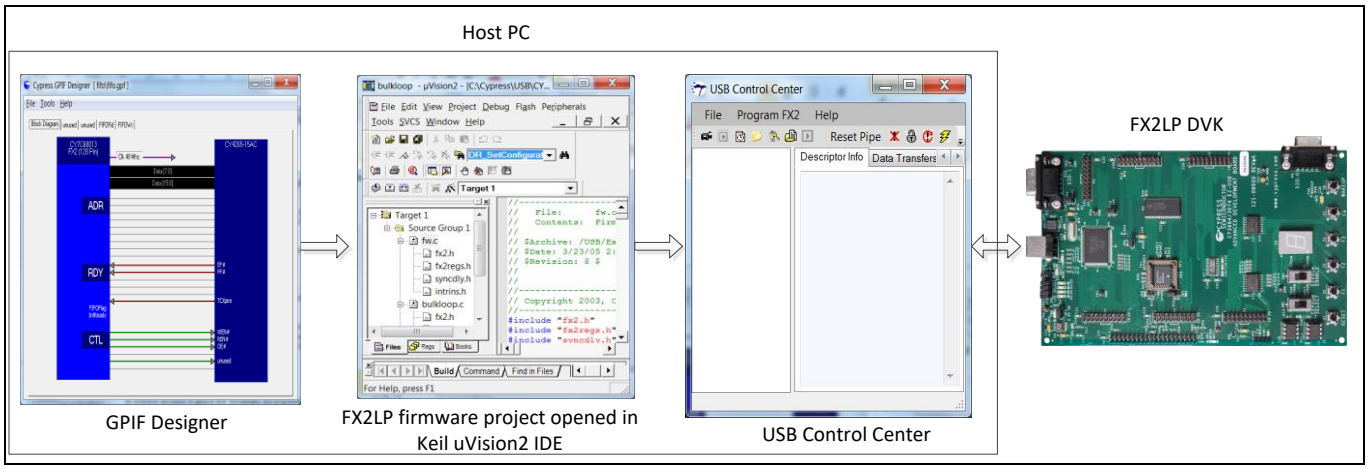


Figure 9 Steps for Developing and Testing Firmware

1. Develop the interface waveforms using GPIF Designer to communicate with the peripheral device connected to the GPIF of FX2LP. Export the *.c* file and integrate the *.c* file into the FX2LP firmware project. This step is not required if you are not using the GPIF interface. More details on using the GPIF Designer are described in the **GPIF Designer** section.
2. Develop the FX2LP firmware with the help of the firmware framework provided by Cypress. Integrate the .c file that was exported from step 1. Build the firmware project using the Keil uVision2 IDE. This step generates *.hex* and *.iic* file. The *.hex* file is for programming the RAM of FX2LP and the *.iic* file is for programming the EEPROM that is connected to FX2LP. The FX2LP firmware framework and the steps to build the FX2LP firmware are described in the **FX2LP Firmware Development** section.
3. Use the Control Center application to program the RAM of FX2LP or the EEPROM that is connected to FX2LP. Usage of the Control Center application is shown in the **Using USB Control Center** section.
4. Verify the functionality of the firmware with the help of the FX2LP DVK. More details of the FX2LP DVK are described in section **5.1** and **Appendix A: FX2LP Development Kit (DVK)**.

The zip file accompanying this application note contains the following directory structure:

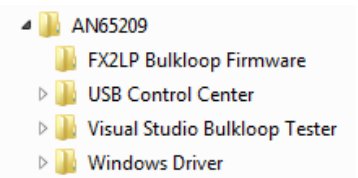


Figure 10 Application Note Folders

*Note: The unzipped application note folder contains copies of various Cypress code samples and development tools available for download. For convenience, they are collected in one folder. For further development, check the **Cypress web site** for the most current file versions.*

This section contains directions for creating and loading an FX2LP firmware example called *bulkloop* into an FX2LP Development Kit. Along the way, it introduces various tools that ease design of any USB peripheral device using FX2LP. If you do not have the FX2LP DVK, the instructions still provide an overview of the FX2LP

chip and tools available. This section shows how to create and run Windows applications to test the *bulkloop* design.

5.1 FX2LP Development Board

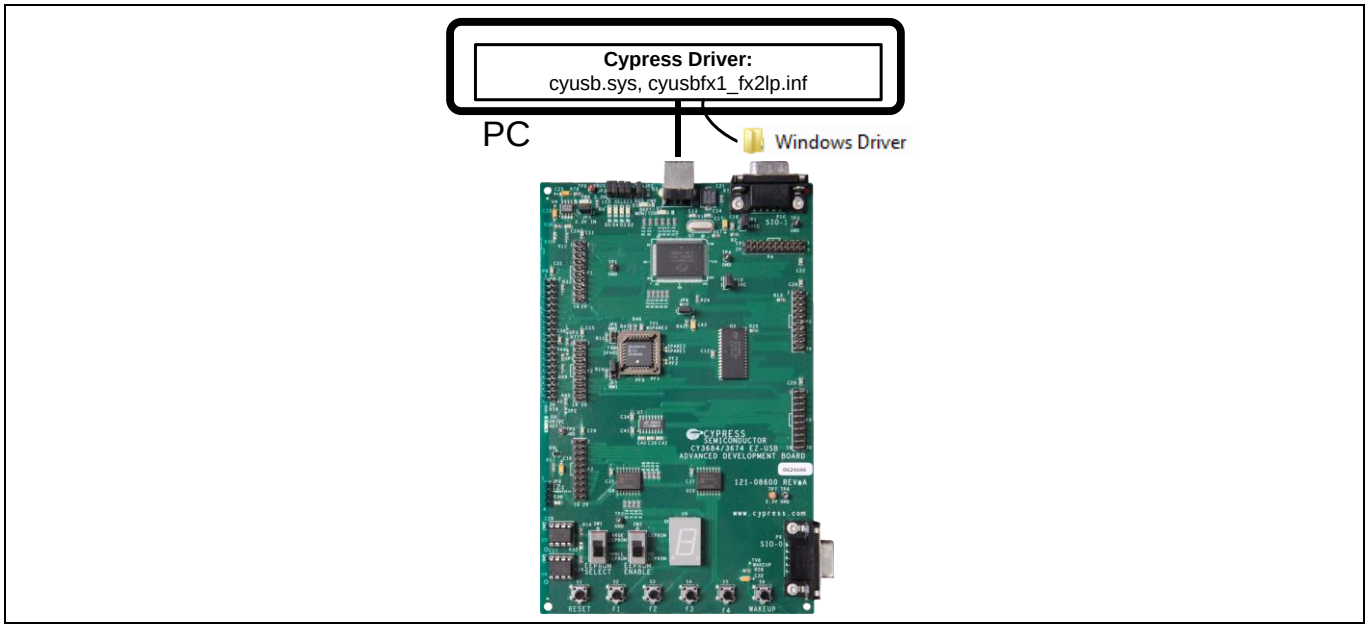


Figure 11 FX2LP Development Kit and Windows Driver

The FX2LP Development Kit is available at: **http://www.cypress.com/?rID=14321**.

Also included with this kit (and available for free download) is the development software package including a free version of the Keil uVision2 IDE (limited in object size to 4 kilobytes) and C compiler. More details about the DVK are in **Appendix A**.

To attach the board to a PC for the first time, follow these steps:

1. Prepare FX2LP board jumpers as shown in **Table 3**.

Table 3 EZ-USB FX2LP Board Jumper Settings

JP	State	Purpose
6,7	OUT	Memory is configured for development
2	IN	Power the board from USB connector
1,5,10	IN	Local 3.3V power source
3	IN	All 4 jumpers IN—activate 4 LEDs D2-D5
8	Either	Not used (for Remote Wakeup testing)

2. In the lower left corner of the board, move the EEPROM ENABLE slide switch to the “NO EEPROM” (down) position. This enables the FX2LP chip to enumerate itself as a code loader. The other slide switch (EEPROM SELECT) can be in either position.
3. Plug the FX2LP board into a PC USB port. If this is the first time, you should see a popup message to install a USB driver. Navigate to the application note driver folder and select the sub-folder corresponding to your Windows OS.

You can confirm a successful driver install by viewing the Windows Device Manager:

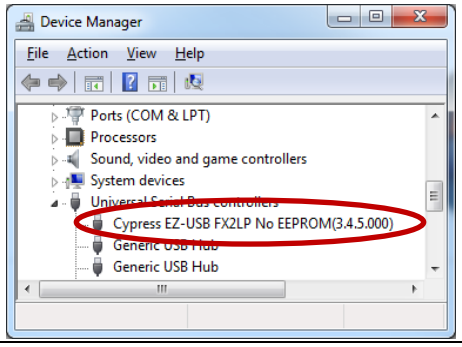


Figure 12 FX2LP Board Driver (Bootloader) is Installed

After this driver is installed, any Windows apps that communicate with the FX2LP Development Board will recognize it.

Before showing how to create the **bulkloop.hex** file, the next two sections illustrate how to use two Windows apps to load and test the FX2LP firmware.

5.2 Running the Bulkloop Demo

5.2.1 Using USB Control Center

The **Error! Reference source not found.** folder contains a Microsoft Visual C# solution to create a Windows a pplication called *USB Control Center.* This application is used to download the FX2LP firmware (a hex file) into the FX2LP chip. After the firmware loads, the FX2LP board automatically detaches itself from USB and then re-attaches as the device defined by the loaded hex file (ReNumeration™). If the loaded hex file has custom Vendor ID (VID) and Product ID (PID) then you need to add these values to the Cypress .inf file.

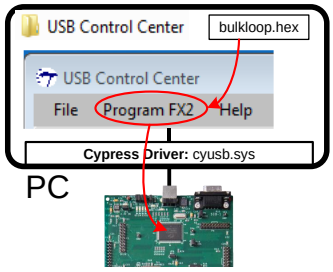


Figure 13 USB Control Center Loads Hex Files into the FX2LP Development Board.

Further Reading

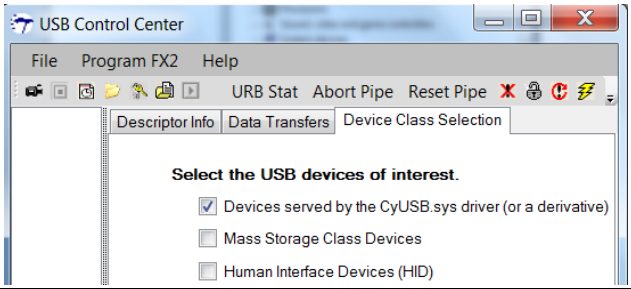
For more information about adding custom VID and PID to the .inf file, refer to **Appendix E: Adding Custom VID and PID to the .inf File**.

The USB Control Center also has a “Data Transfer” tab that allows you to initiate USB transfers into and out of the FX2LP DVK to test the *bulkloop* application or any other firmware you write.

The attached zip file contains a Visual Studio solution (**AN65209\USB Control Center**) and its compiled binary. To try it out, follow these steps:

1. Double click on the **CyControl.exe** file in either the Debug or Release folder (**AN65209\USB Control Center\bin**).

You should see the FX2LP board along with other connected USB devices listed in the left panel. To simplify this display to show only the FX2LP board, click the **Device Class Selection tab** in the right panel and uncheck everything except the **Devices served by the CyUSB.sys driver (or a derivative)** item.



Then the left panel should look like **Figure 14**.

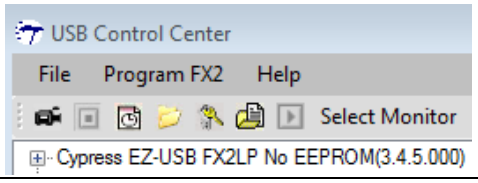


Figure 14 USB Control Center Finds the FX2LP Board.

2. Now you are ready to load the FX2LP firmware ***bulkloop.hex*** (**AN65209\FX2LP Bulkloop Firmware**) on to the FX2LP DVK. Click on the Cypress device entry to highlight it, and then select **Program FX2 > RAM**. Open the **FX2LP Bulkloop Firmware** folder and select the ***bulkloop.hex*** file. After the code loads, the FX2LP Development Kit disconnects itself from USB and reconnects as the device created by the loaded firmware ***bulkloop.hex*** (**Figure 15**). This is Cypress ReNumeration™ in action—FX2LP has enumerated twice, first as a code loader and then as the loaded device.

The LED D5 should blink 8 times per second for a Hi-Speed USB attachment, and once per second for a Full Speed attachment. You can observe this by looking at the toggling frequency of the LED D5 present on the FX2LP DVK. For Hi-Speed USB attachment, you'll see the LED blink slowly for a second or so, and then start blinking quickly. This is because a USB device initially attaches at Full Speed, then negotiates the Hi-Speed connection with the host. This is one of the many firmware details handled by the Cypress USB frameworks described in the **FX2LP Firmware Development** section.

The 7-segment readout present on the FX2LP DVK should light up with the number zero, indicating the number of packets ready to be transferred back to the host—none have been received and "looped back" yet.

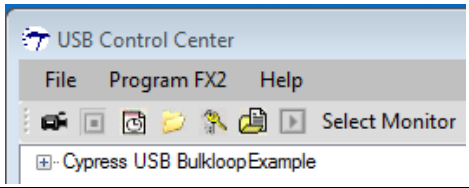


Figure 15 The FX2LP Board Re-appears

*Note: Before you load any hex file into the FX2LP DVK, you must first press the Reset button (**Error! Reference source not found.**, lower left corner) by keeping the "EEPROM ENABLE" switch in "NO EEPROM" position to reset FX2LP and thereby enable the FX2LP boot loader.*

Figure 16 shows the interaction between USB Control Center and the FX2LP Development Board loaded with the ***bulkloop.hex*** file. The *bulkloop* firmware enumerates as a Full Speed or Hi-Speed device depending on the PC connection and copies (loops) BULK data it receives from OUT Endpoint 2 into an IN Endpoint 6 buffer for transmission back to the PC. As the diagram indicates, the endpoints are double-buffered, a feature demonstrated in this section.

Note: ***Because*** *USB directions are host-centric, OUT means host to device, and IN means device to host.*

The *bulkloop* firmware also contains the code for useful operations, such as controlling the LEDS and 7-segment readout on the FX2LP Development Board, and using endpoint interrupts. This code is examined in detail in the **FX2LP Firmware Development** section.

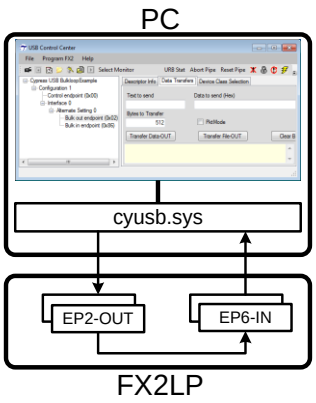


Figure 16 USB Control Center Tests the Bulkloop Firmware

1. Expand the Bulkloop Example tree view in the Control Center to reveal the implemented BULK endpoints (**Figure 17**).

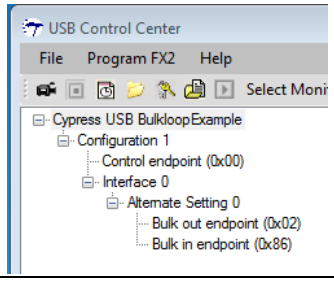


Figure 17 Bulkloop Example Device Internals

2. Select the **Data Transfer** tab. Click on the **Bulk out endpoint (0x02)** entry in the left-hand panel, and notice that the Transfer Data button is **Transfer Data-OUT**. Clicking this button leads to the following:

- 512 bytes (with zero default values) transfer from the PC to the FX2LP DVK.
- LED D3 flickers to indicate the OUT transfer.
- The 7-segment readout increments to 1, indicating one packet has been received over EP2-OUT and loaded into the EP6-IN endpoint FIFO, ready for transfer to the host.

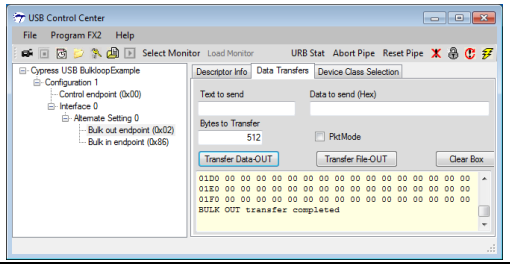


Figure 18 Successful BULK-OUT Transfer

1. Click the **Transfer Data-OUT** button again. The PC dispatches a second packet to FX2LP, and the 7-segment readout increments to 2.
2. Highlight the **Bulk in endpoint (0x86)** entry. The Transfer button label now indicates **-IN**. Clicking this button leads to the following:

- 512 bytes transfer from FX2LP to the host, which it displays as hexadecimal values.
- LED D2 flickers to indicate the IN transfer.
- The 7-segment readout decrements to 1.

3. Click the **Transfer Data-IN** button again. The second queued packet transfers to the host, and the 7-segment readout indicates 0 packets waiting. This sequence confirms the double-buffered operation of the two endpoints.
4. Select the **Bulk out** endpoint again. Then, position the mouse cursor inside the **Text to send** box and type **1234**. The hex values display in the **Data to send** box and the **Bytes to Transfer** box increments for every digit typed. Click the **Transfer Data-OUT** button.
5. In the **Text to send** box, type **abcdefg**, then click the **Transfer Data-OUT** button.
6. Select the Bulk in endpoint, and leave the default **Bytes to Transfer** value of 512 bytes. Click the **Transfer Data-IN** button twice to read the two packets of data that was transferred.

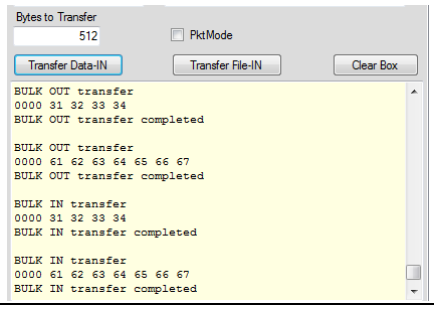


Figure 19 Two Packets OUT, Two Packets IN

As **Figure 19** demonstrates, a feature of USB is that a USB peripheral always sends the smaller of bytes-requested (512, in this example) and bytes-available (4 or 7).

5.2.2 Using BULKLOOP_VCS

A second Visual Studio project called **Bulkloop_VCS** performs streamlined transfers in and out of the FX2LP Development Board. It continuously sends and receives BULK data while keeping a running total of bytes transferred. Like the USB Control Center, it displays the device characteristics in the left panel for information. Unlike the USB Control Center, it automatically sets up and conducts continuous transfers when the **Start** button is pressed.

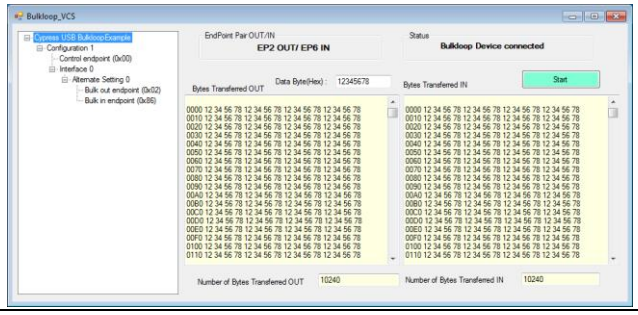


Figure 20 Bulkloop_VCS Window

Before using this application, use the USB Control Center to load the ***bulkloop.hex*** file as described in the previous section. Be sure to press the FX2LP Development Kit RESET button first to enable the FX2LP bootloader by keeping the “EEPROM ENABLE” switch in “NO EEPROM” position.

5.3 FX2LP Firmware Development

This section explains the steps involved in generating the ***bulkloop.hex*** file.

5.3.1 Keil uVision2

The Cypress FX2LP DVK includes a free demonstration version of the Keil uVision2 tools for the 8051. This IDE is fully featured, but limited in code size to 4096 bytes. Cypress demo projects, such as **bulkloop.uv2,** fit under this limit so you can study and modify the code. Larger projects require purchasing the full uVision2 toolset from Keil.

5.3.2 Cypress USB Firmware Frameworks

In conjunction with the Keil tools, Cypress provides a set of files called the USB Firmware Frameworks to handle low-level USB details. This approach allows you to concentrate development time on your specific application code.

When you open the **bulkloop.uv2** project, you see the following project files:

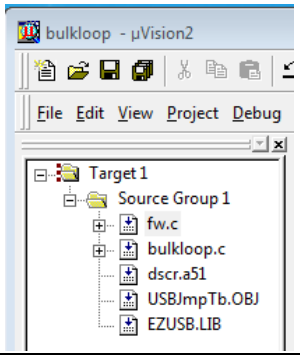


Figure 21 bulkloop.uv2 Project Files

- **fw.c** is the Cypress USB Firmware Frameworks.

For more information on firmware frameworks, refer to Chapter 5 in **FX2LP Development Kit User Guide.pdf**.

- **bulkloop.c** is your application. Other Cypress examples may call this code module **peripheral.c**; it is a good idea to rename it for your specific project name.

- **dscr.a51** is an assembly language file containing the data necessary to enumerate the USB device. The file comprises table data in the form of **.db** (define byte) statements. You edit this file to customize items such as VID/PID and descriptive strings for your design.
- **USBJmpTb.OBJ** is a jump table required by the FX2LP architecture, never requiring modification.
- **EZUSB.LIB** contains a library of useful FX2LP functions, mostly dealing with I^2C communication. The source code is included with the FX2LP DVK in the **Target\Lib\LP** subfolder.

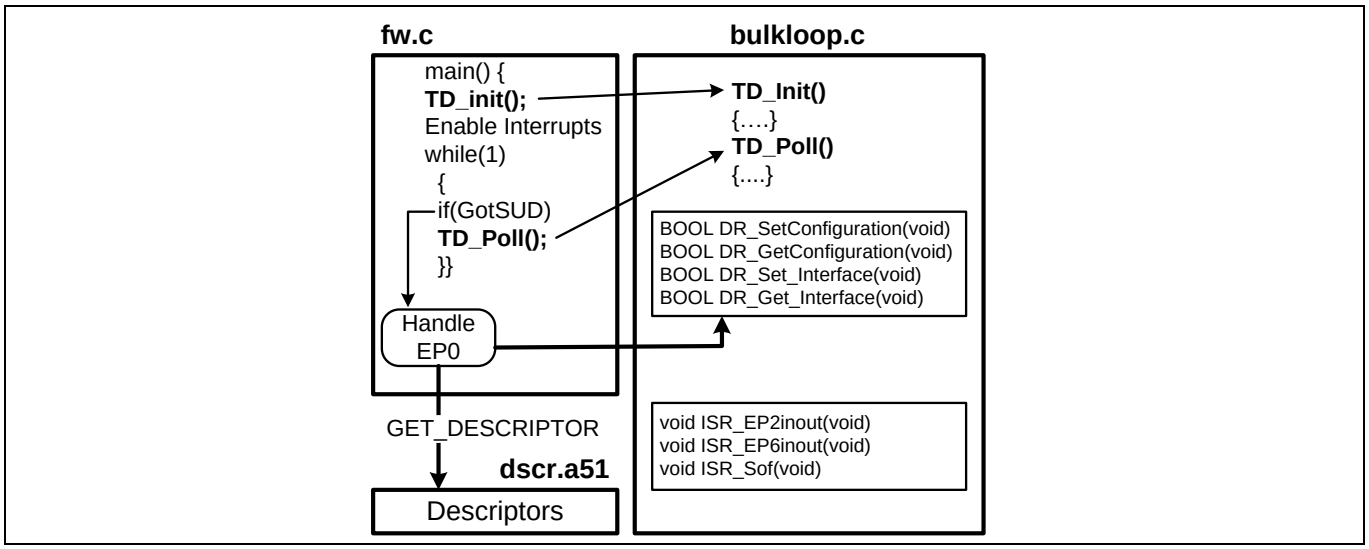


Figure 22 USB Firmware Framework Structure

Figure 22 shows how the code modules fit together.

Fw.c contains the **main** function. It performs much of the USB maintenance, such as enumeration, and it calls specifically-named external functions in the application code (**bulkloop.c**) whenever customization is required. **Fw.c** mostly does not require your modification. After performing various housekeeping steps, it calls an external function called **TD_init**, which you provide in **bulkloop.c**. (The prefix TD stands for “Task Dispatcher”.) Then it enters an endless loop that checks for arrival of SETUP packets over CONTROL endpoint 0. The loop also checks for the USB suspend event, but this is not used by the *bulkloop* application. Every time through the loop, it calls the external **TD_Poll** function which you provide in ***bulkloop.c*.** In this application the **TD_Poll** function does the work of looping PC endpoint data through FX2LP.

Every USB peripheral receives two types of requests over its CONTROL endpoint: enumeration and operational.

Enumeration

When a USB device is attached, the host PC sends multiple **GET_DESCRIPTOR** requests to discover the device type and its requirements as part of a process called enumeration. The **fw.c** code intercepts these requests and handles them using the values stored in the **dscr.a51** file.

An advantage of using the USB Frameworks is that the code has been tested and verified to pass USB “Chapter 9” requirements. Chapter 9 refers to the chapter in the USB Specification that deals with device requests (over EP0) and their proper responses.

Operational

Wherever the user code is needed, **fw.c** calls a specifically-named external function with the DR prefix (Device Request) that you provide in the **bulkloop.c** file. For a simple application such as *bulkloop*, there is only one configuration and one interface, so the two **DR_Set-Get** function pairs in **Error! Reference source not found.** s

imply store the Set values sent by the host and echo them back when the host issues the Get requests. For more complex configurations, you can use these DR calls (“hooks”) to change camera resolutions or route requests to two different interfaces.

Because **bulkloop.c** contains a complete template for any USB device, you can use it as the basis for your custom implementation. The remainder of this section describes the three portions of this file that require user code to implement the bulkloop application.

5.3.2.1 TD_Init

This function does the following:

- Sets the 8051 clock to 48 MHz.
- Turns off the development board’s four LEDs. LEDs are turned on and off by reading specific memory locations. This method controls the LEDs without consuming any I/O pins.
- Configures EP2 as a BULK-OUT endpoint and EP6 as a BULK-IN endpoint. Both are double-buffered and use 512 byte FIFOs (**Error! Reference source not found.**).
- Enables (“arms”) EP2-OUT to receive two packets. The OUT endpoint is armed by writing any value into its byte count register whose MSB is set. Setting the MSB (called the “SKIP bit”) skips the packet.
- Enables the FX2LP dual auto pointers. These hardware pointers auto-increment for efficient memory-to-memory byte transfers from the EP2-OUT buffer to the EP6-IN buffer.
- Enables three interrupts: SOF, EP2, and EP6 endpoint interrupts.

5.3.2.2 TD_Poll

TD_Poll is called in an infinite loop residing in **fw.c** (**Error! Reference source not found.**). For the *bulkloop* application, only two tasks are required:

1. Update the 7-segment readout with the number of packets waiting for transmission to the host. The FX2LP register EP6CS (Endpoint 6 Control and Status) provides this number in bits 6-4.

Further Reading

For more information on registers, refer to Chapter 15 of the **FX2LP Technical Reference Manual**.

2. Check the endpoint FIFO flags to determine when it is time to transfer an OUT packet to an IN buffer. When it is time, move the packet data from the EP2-OUT buffer to the EP6-IN buffer using auto pointers.

To understand how the item 2 transfer decision is made, it is important to understand two points regarding the FX2LP endpoint FIFO flags:

a) When multiple buffering is used, the FULL and EMPTY flags (of EP2468STAT register) reflect all the buffers, not just one. Therefore, in the double-buffered case for this example, if one OUT packet is received, the FULL flag remains unasserted because the second buffer is still available for an OUT transfer. The FULL flag asserts only when a second packet arrives. Similarly, an IN endpoint EMPTY flag asserts only when both buffers are empty, ready for the 8051 to fill them with new data.

b) FX2LP updates FIFO flags (of the EP2468STAT register) only after successful receipt or transmission of a USB packet.

Therefore, a looping transfer occurs when the following two conditions are satisfied:

- EP2-OUT is not empty, AND
- EP6-IN is not full.

In other words, EP2 has a packet, and EP6 has room for a packet. Doing the test this way handles any packet size and takes the double-buffering into account.

5.3.2.3 Interrupt Service Routines

The **bulkloop.c** file contains ISR functions for every USB interrupt source. A few ISRs set flags for the **fw.c** code, and others perform USB enumeration functions. You only need to fill in code for the ISRs that your application uses.

The following ISR shows how to clear an FX2LP USB interrupt request:

```
// Setup Token Interrupt Handler
voidISR_Sutok(void) interrupt 0
{
   EZUSB_IRQ_CLEAR();
   USBIRQ = bmSUTOK;      // Clear SUTOK IRQ
}
```

The “0” after the interrupt keyword is the ID for all USB interrupt requests. Two interrupt request flags are cleared in a particular order: first, the general USB interrupt flag and then the individual USB source flag; in this example, the “Setup Token Arrived” flag. This ISR is an example of a code “hook”; you can take any action when a SETUP packet arrives by inserting code into this ISR.

The *bulkloop* application requires four customized ISRs:

Set_Configuration ISR

The host sets the FX2LP configuration as the last step of its enumeration process. This is a good time to initialize application hardware. The I^2C unit that drives the 7-segment readout is initialized here.

EP2INOUT/EP6INOUT ISR

These IRQs fire when a packet is dispatched from EP6-IN or arrives at EP2-OUT. The ISR code turns on an LED on the FX2LP Development Board, then sets an **inblink** (EP6-IN) or **outblink** (EP2-OUT) variable (in **bulkloop.c**) to control how long the LED stays on.

SOF ISR

The SOF ISR serves as a convenient timer, firing every millisecond at Full Speed and every 125 microseconds at Hi-Speed. The ISR code toggles an LED on the FX2LP Development Board every 500 times through the ISR, which equates to once per second at Full Speed and 8 times per second at Hi-Speed. The ISR code also decrements **inblink** and **outblink** variables that were set when IN and OUT packets arrived, turning off indicator LEDs when the counters hit zero.

5.3.2.4 Handling USB Dual Speeds

When a USB 2.0 device comes out of reset it operates at Full Speed. The host then negotiates with the device using low-level bus signaling to determine its operating speed. FX2LP provides an interrupt to indicate that it has just switched to Hi-Speed operation as the result of the host-device speed negotiation.

A USB 2.0 device must operate at Full Speed and Hi-Speed. A Hi-Speed USB device provides two sets of descriptors, one for each speed. Two ISRs take care of designating the proper descriptors depending on speed:

- The **ISR_Ures**(USB Reset) code designates the Full Speed descriptor as the *current speed* descriptor, and the Hi-Speed descriptor as the *other speed* descriptor. If plugged into a full-speed port, no further action is required.
- The **ISR_Highspeed** interrupt service code swaps the *current/other* descriptor designations—Hi-Speed is now the *current speed* and full-speed is the *other speed*. This dual designation allows Windows to put up this message if it detects a Hi-Speed device plugged into a Full Speed port:

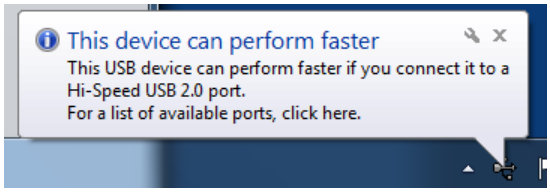


Figure 23 Windows Knows When a High-Speed Device Can Perform Better

This nontrivial bit of USB housekeeping is an example of what the Firmware Frameworks does for you. The dual descriptors and descriptor swapping code is written for you. All you need to do is fill in the descriptor fields unique to your application.

5.3.3 Building the Bulkloop Project

In the FX2LP Bulkloop Firmware folder, double-click the **bulkloop.uv2** file. This opens the uVision2 IDE and loads the bulkloop project. To compile and link the project, click the **Rebuild All Target Files** button.

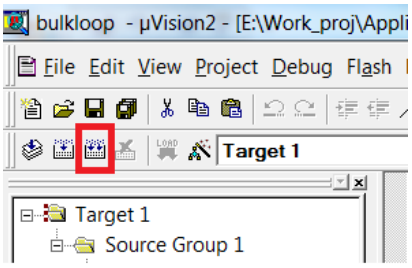


Figure 24 uVision2 IDE

This creates the *bulkloop.hex* file you downloaded in previous sections.

Note the following when installing Keil uVision2:

- Sometimes, files unzipped from a web download install as read-only. Right-click on the folder, select “Properties”, and uncheck the “Read-only” box if selected.
- The final Keil build step is automatically to run a program called **hex2bix.exe,** which converts the Keil output into a iic file. The location of this file is coded into the project, so moving or installing the Keil tools may break the path. A simple remedy is to include the **hex2bix.exe** file in the Keil project folder, and invoke the utility without specifying a path. (This is done in the companion code to this application note.) You can locate the path by right-clicking the top file in the Keil project window (default name is “Target 1”), selecting “Options…”, and then selecting the “Output” tab. The path should look like this:

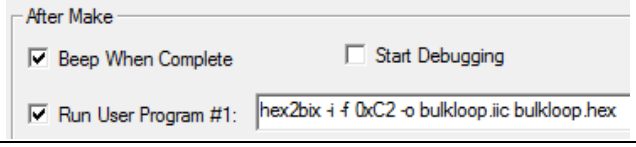


Figure 25 Path to the hex2bix Utility

5.4 GPIF Designer

The *bulkloop* example handles everything on-chip, where the 8051 checks flags, moves bytes between endpoint FIFOs, and updates the development kit status indicators (LEDs and 7-segment readout). Most FX2LP designs do more than this, transporting data between FX2LP and outside peripherals, such as disk drives or image sensors. For this purpose, the GPIF provides a fast and flexible interface to outside systems. This section highlights **AN66806 - Getting Started with EZ-USB FX2LP GPIF**, which gives step-by-step instructions for using the Cypress GPIF Designer tool.

The first step in a GPIF design is to fill in a block diagram with the interface signals, as **Figure 26** shows.

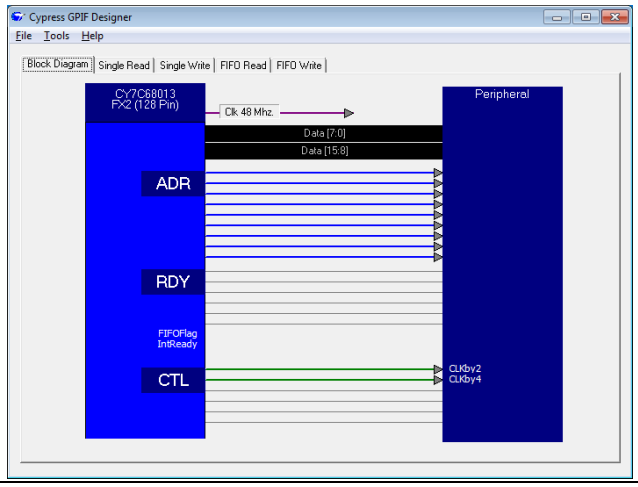


Figure 26 GPIF Designer Block Diagram Tab

The GPIF has six Control (CTL) outputs and six Ready (RDY) inputs that you can rename in the block diagram by right-clicking in the RDY or CTL text blocks. Your names propagate through GPIF Designer, appearing as choices for state machine decision points and waveform names.

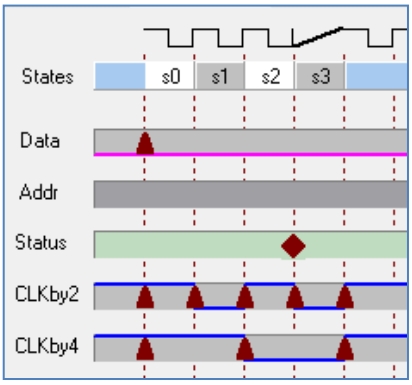


Figure 27 GPIF Waveform Entry Creates States

Figure 27 shows a GPIF Designer waveform entry screen. In this simple example, the internal 48-MHz clock is divided by 2 and 4, and output on the renamed CTL0 (CLKby2) and CTL1 (CLKby4) pins. The vertical dotted lines show the clock edges on which transitions can occur; these can be set to rising or falling edges. To create a transition, you click in the desired waveform on a clock edge. This toggles the waveform and inserts a red triangle to mark the transition point. You can delete these triangles or drag them to different clock edges, and set logic levels directly to 1 or 0 with mouse clicks.

As you define waveform transitions, GPIF Designer automatically creates states in the top line—in this example, the four states S0 through S3. This waveform is designed to repeat endlessly as it is designed to produce the clock outputs. This is accomplished by branching from state 3 back to state 0. Branching to a state is done by inserting a *decision point* in the Status line—click on the dotted clock line at the beginning of the decision state. GPIF Designer inserts a diamond to signify the decision state. Right-clicking this diamond brings up a dialog box that allows you to create a logic equation using two variables and three logical operators (AND, OR, XOR) for the branch condition. An unconditional branch is created by selecting the same state as destination for *if* condition check and for *else* condition.

Further Reading

For more information on GPIF, refer to Chapter 10 of the **FX2LP Technical Reference Manual**.

For information on interfacing RAM to GPIF of FX2LP, refer to **Interfacing SRAM with FX2LP over GPIF - AN57322**.

6 The Windows Side

Running the Bulkloop Demo shows how to use the **USB Control Center** and **BULKLOOP_VCS** Windows applications to test the bulkloop hex file created by the Keil tools. This section gives an overview of tools from Cypress and others that can be used to create Windows applications that communicate with FX2LP-based devices.

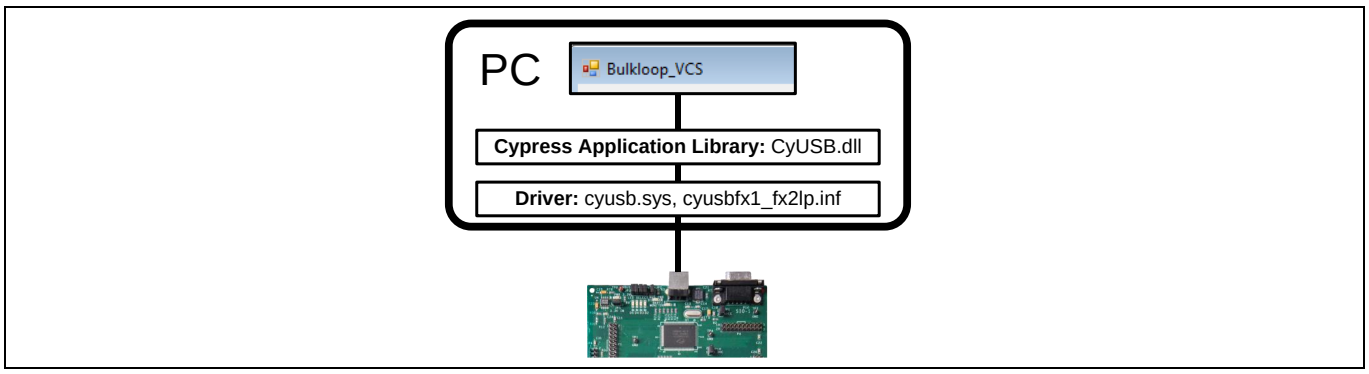


Figure 28 Windows .NET Apps Communicate with FX2LP using a Cypress Driver and Library

6.1 Cypress USB Driver

The Cypress USB driver (**CyUSB.sys**) is a robust high-performance Windows driver used for host communication with a USB-connected target. This driver is installed on the host PC as part of the EZ-USB FX2LP DVK installation, and in the accompanying zip file in the **Windows Driver** folder. The driver comes in binary form and can be distributed with your FX2LP-based devices.

- **CyUSB.sys** is a kernel mode USB function driver, which is capable of communicating with any USB 2.0 compliant devices. The driver is general-purpose, compatible with primitive USB commands, but does not implement higher-level commands specific to USB device classes.
- The driver is ideal for communicating with a vendor-specific device from a custom USB application or for sending low-level USB requests to any USB device for experimental or diagnostic applications.
- Operating systems/platforms supported by the driver are Windows XP, Windows Vista, and Windows 7, on both 32- and 64-bit versions.

If you modify the *.inf* file, for example, to include your company's VID and/or PID, or to include custom-descriptive text strings, Cypress recommends that you obtain "WHQL" certification from Microsoft (WHQL stands for "Windows Hardware Quality Labs".) With this Windows Logo certification, you do not get a warning message when the driver installs. **Appendix E: Adding Custom VID and PID to the .inf File** shows how to add a custom VID/PID to the *inf* file.

Further Reading

For more information on **CyUSB.sys**, refer to *CyUSB.pdf* present in the **Cypress Suite USB\Driver** folder (after Cypress Suite USB installation).

6.1.1 Driver/Library Alternatives

Other Windows drivers and libraries are available. WinUSB from Microsoft (details at the **msdn** site) comprises a kernel-mode driver (**Winusb.sys**) and a user-mode dynamic link library (**Winusb.dll**) that exposes WinUSB functions. By using these high-level functions, you can manage USB devices with user-mode software. **Winusb.sys** is supported on Windows XP, Windows Vista, Windows 7, and Windows 8. Like the Cypress driver, it requires an accompanying *.inf* file for installation.

Note: ***WinUSB.sys*** *does not support ISOCHRONOUS transfers, while the Cypress driver supports all USB transfer types.*

Note: ***Linux and*** *Mac users can investigate LIBUSB, a suite of open source user-mode USB routines. Details can be found in the* ***Libusb-1.0 API Reference****.*

In Linux machines, the CyUSB Suite for Linux software (provided as part of FX3 SDK for Linux and can be used for FX2) enables you to download firmware images to FX2/FX3 devices and test the various interfaces on the device. See *cyusb_linux_user_guide.pdf* available in the **FX3 SDK for Linux** installation folder: *fx3_sdk_v1.3_linux\cyusb_linux_1.0.4\cyusb_linux_1.0.4\docs*. This document describes how to install the software, download firmware to FX2/FX3, test Vendor Extensions, BULK OUT/IN transfers, and ISOCHRONOUS OUT/IN transfers. The CyUSB Suite for Linux - Programmers Reference Manual (*cyusb_linux_programmers_guide.pdf* available in the same folder), describes the cyusb library for Linux and how to build and integrate user-written applications with the library”

FX2LP can also be programmed to be compliant with a standard Windows class, for example Human Interface Device (HID), Communications (COMM) device or Mass-Storage (MSC) device. The Cypress web site has example FX2LP programs for these common classes. The advantage of conforming to a Windows class is that no driver installation is required when the device is first plugged in—the drivers are part of Windows.

6.2 Cypress Libraries

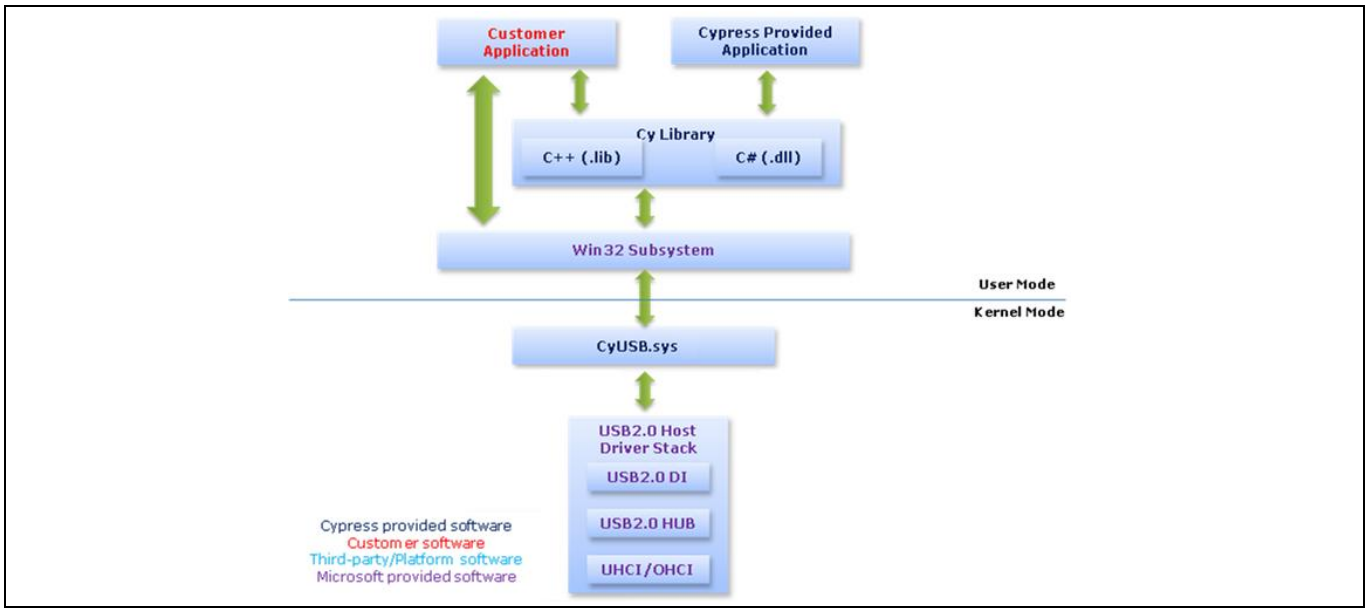


Figure 29 Detailed View of How Cypress Libraries Communicate with Applications

The **Bulkloop_VCS** application is an example of the software architecture used for any Windows-to-FX2LP communication using the Cypress .NET library. A C# language program makes library calls that access the Cypress driver to communicate with FX2LP (see **Figure 28**). The **CyUSB.dll** library supports the Microsoft .NET Languages. Cypress also provides a C++ class library called **CyAPI.lib**. Both libraries are available with the **Cypress Suite USB** installation.

Just as the Cypress USB Firmware Frameworks simplifies code development at the FX2LP level, these libraries simplify coding at the Windows level. The software model (based on **CyUSB.dll**) employs the following three elements:

The Windows Side

- The **USBDeviceList** class creates a list of all attached USB devices. The constructor can filter the list for a single device class, for example HID or MSC, or to devices served by the Cypress driver.
- Device List members are **CyUSBDevice** class instances.
- **CyUSBDevice** classes expose one or more **CyUSBEndPoints**, through which data transfers are performed.

Communicating with USB devices using this high-level model is a great improvement over multiple Win32 API calls such as **DeviceIOControl**.

An excellent way to gain familiarity with the Cypress library is to open the **Bulkloop_VCS** solution using Microsoft Visual C# Express edition, and inspect the code.

Further Reading

For more information on developing host applications, refer to the application note, **AN70983 - Designing a Bulk Transfer Host Application for EZ-USB® FX2LP™/FX3™**.

For more information on .NET dll library, refer to *CyUSB.NET.pdf* present in the folder **Cypress Suite USB\CyUSB.NET** (after Cypress Suite USB installation). For more information on **CyAPI.lib**, refer *CyAPI.pdf* present in the folder **Cypress Suite USB\CyAPI.**

7 Summary

The Cypress FX2LP is designed to meet your USB 2.0 Hi-Speed design requirements. Cypress provides an extensive suite of support collateral to help with each step of your design cycle. This application note first introduced FX2LP, then introduced Cypress hardware, firmware, and software tools by stepping through the design, creation and testing of a BULK transfer example. This example can serve as a basis for your custom application.

About the Author

Name: Rama Sai Krishna

Title: Application Engineer Senior

8 Appendix A: FX2LP Development Kit (DVK)

The **CY3684EZ-USB FX2LP Development Kit** is a complete development resource. It provides a platform to develop and test custom projects. The development kit contains collateral materials for the firmware, hardware, and software aspects of a design. **Error! Reference source not found.** shows the FX2LP DVK Development Board components.

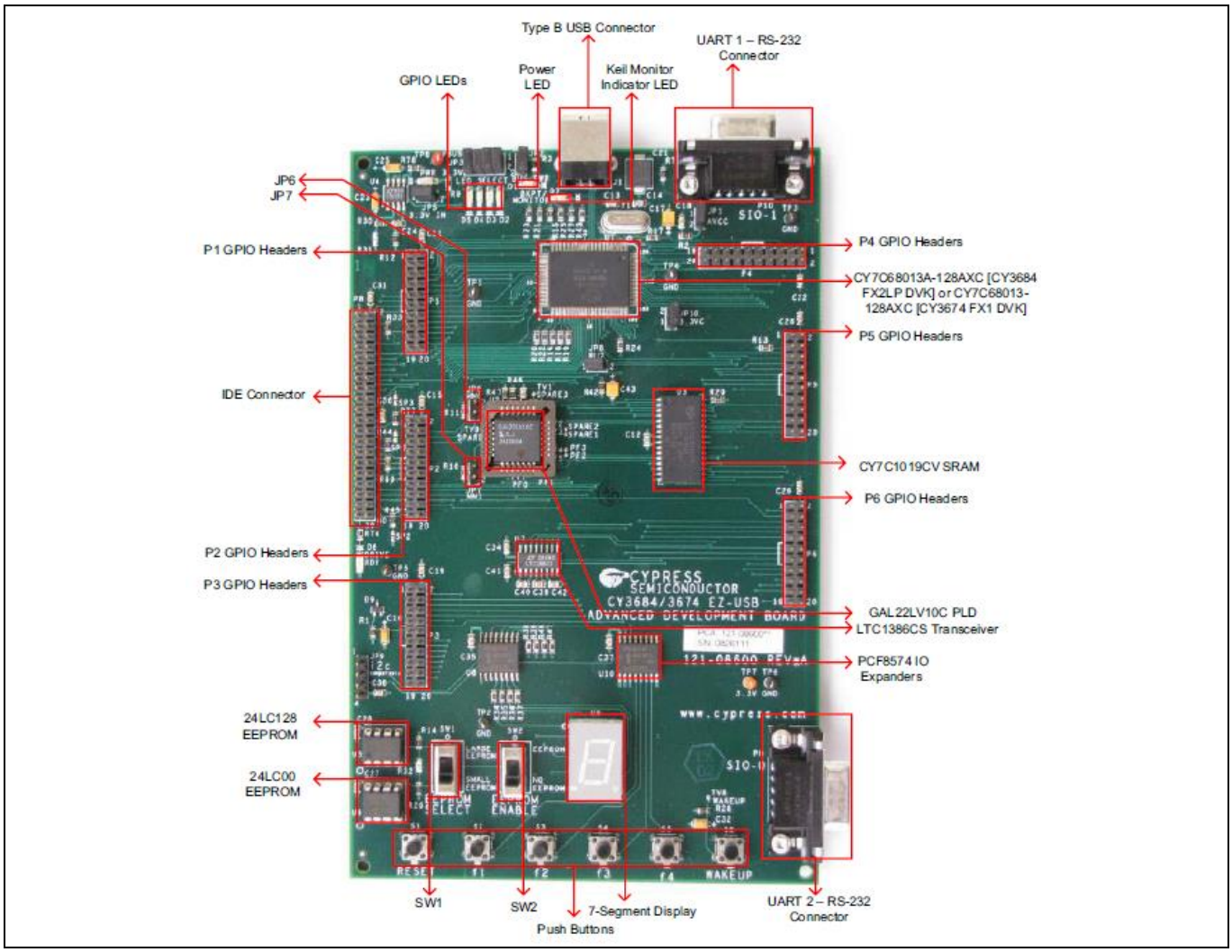


Figure 30 FX2LP DVK Development Board Components

The FX2LP DVK is an excellent debugging platform for FX2LP. The board can load code images from USB (as in this application note) or the code can be programmed into an onboard EEPROM.

When developing hardware for USB 2.0 Hi-Speed devices, board layout and design are critical to the success of the project. To help developers avoid common layout errors, Cypress provides several resources to help design a new board. Application notes **AN1168** and **AN15456** give design guidelines for FX2LP. In addition, the hardware subdirectory of the FX2LP DVK installation contains the FX2LP development kit schematic, the BOM for the development board, and the development board layout and design files.

8.1 Firmware Example Projects

As explained in the **FX2LP Firmware Development** section, the FX2LP DVK provides a USB Firmware Frameworks library that satisfies the Chapter 9 compliance requirements of the USB 2.0 specification for high-

speed devices. The USB Frameworks simplify and accelerates custom firmware development by using Cypress code for common operations, such as FX2LP chip initialization, USB standard device request handling, and USB suspend-resume power management. USB Firmware Frameworks also provides function hooks and firmware examples, easing the firmware development process. You can write the USB descriptor table and code to implement the desired functionality without worrying about low-level USB details.

After installing the **CY3684EZ-USB FX2LP Development Kit** the firmware directory contains the examples shown in **Table 4**. These examples can be used as a reference or used as the basis for custom FX2LP-based products.

Table 4 Description of FX2LP Firmware Examples

S.No	Firmware Example	Description
1	hid_kb	Emulates a HID-class keyboard using the buttons and 7-segment display on the DVK board.
2	Bulkloop	A bulk loopback test that exercises the EZ-USB bulk endpoints. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN, and updates Development Board indicators.
3	Bulkext	A bulk loopback test that uses external RAM. Data is copied from an OUT endpoint buffer to external RAM on the DVK board and then back to the IN endpoint buffer. It loops back EP2OUT to EP6IN and EP4OUT to EP8IN.
4	Bulksrc	Endless providers and consumers of BULK data for testing. It can be driven using the CyConsole or CyBulk. EP2OUT and EP4OUT accept all BULK packets. EP6IN always returns a 512-byte packet when operating at Hi-Speed and 64 bytes when operating at Full Speed. Based on buffer availability in EP8IN, the most recent packet of EP4OUT is written to EP8IN.
5	dev_io	Source files to build simple development board I/O sample. This software demonstrates how to use the buttons and LEDs on the EZ-USB development kit.
6	EP_Interrupts	Bulk loopback firmware using endpoint interrupts.
7	extr_intr	External interrupt handling using INT0, INT1, INT4, INT5, and INT6.
8	ibn	Bulk loopback of EP2OUT to EP6IN and EP4OUT to EP8IN using the IBN (In Bulk NAK) interrupt to initiate the transfer.
9	LEDCycle	Use of the general-purpose indicator LEDs (D2, D3, D4, D5) on the DVK.
10	Pingnak	Bulk loopback of EP2OUT to EP6IN andEP4OUT to EP8IN using the PING NAK interrupt to initiate the transfer.
11	iMemtest	Tests on-chip RAM.
12	vend_ax	Shows how to implement vendor-specific commands.

USB Firmware Frameworks uses the EZ-USB library (EZUSB.LIB), which implements functions that are common to many firmware projects. These functions need not be modified and are, therefore, provided in library form. However, the kit includes the source code for the library in the event that you need to modify a function or if you just want to know how something is done. Detailed information about the EZ-USB library (section 5.4) and firmware framework (chapter 5) is available in the '**CY3684 DVK Kit_Guide**'. This library is included in DVK under the 'Target\Lib\LP' folder. To help customers design their applications faster with a wide range of Cypress Hi-Speed products, a comprehensive list of all **USB Hi-Speed Code Examples** are available. Details on other USB High-Speed Devices, Kits, SDKs, Application Notes and Reference Designs are available below.

Appendix A: FX2LP Development Kit (DVK)

Note: *These software resources are continuously improved. Therefore, Cypress recommends downloading the latest software from the **Cypress website**.*

9 Appendix B: Additional USB Hi-Speed Devices from Cypress

The **Cypress USB Hi-Speed peripherals** family includes the following additional devices:

- **AT2LP:** Cypress's EZ-USB® AT2LP™ (CY7C68300C/301C/320C) implements a fixed-function bridge between one USB port and one or two ATA- or ATAPI-based mass storage device ports. The PATA interface on AT2LP enables the use of hard disk drives (HDD), compact flash, and solid state drives (SSD) in your design. The AT2LP is perfect for mass storage type applications and enables quick time to market without custom firmware. AT2LP supports all ATA/ATAPI-6 compliant mass storage devices.
- **NX2LP-Flex:** Cypress's EZ-USB NX2LP-Flex™ (CY7C68033/34) is a low-power programmable USB to SLC NAND controller. Its programmability allows designers to include special features in the controller along with NAND device support, an advantage over fixed controllers. The hardware ECC engine present in NX2LP-Flex supports 1-bit error correction and 2-bit error detection.
- **SX2:** The EZ-USB SX2 (CY7C68001) is a programmable device designed to work with any external master, such as standard microprocessors, DSPs, ASICs, and FPGAs to enable USB 2.0 support for any peripheral design. SX2 has a built-in USB transceiver and serial interface engine (SIE), along with a command decoder to send and receive USB data. The controller has four endpoints that share a 4KB FIFO space for maximum flexibility and throughput. SX2 has three address pins and a selectable 8- or 16- bit data bus for command and data input or output.
- **FX2LP18:** MoBL-USB FX2LP18 (CY7C68053) operates at 1.8 V, making it suitable for use in low-power handheld devices. **AN6076** lists the differences between FX2LP and FX2LP18.

10 Appendix C: Third-Party Development Kits and SDKs

1. **FPGA + FX2LP board from Opal Kelly:**

More details of this board can be found in the following location:

http://www.opalkelly.com/products/xem6010/

Features:

- Hi-Speed USB 2.0 interface (Cypress FX2LP - CY68013A) for downloading and control
- Xilinx Spartan-6 (XC6SLX45-2FGG or XC6SLX150-2FGG)
- 32-Mib serial flash (Numonyx M32P25)
- 128-MiByte DDR2 (Micron MT47H64M16HR)
- Small form-factor—smaller than a credit card at 75 mm x 50 mm x 15.9 mm (2.95" x 1.97" x 0.63")
- Self-powered by external DC source
- Multi-PLL, multi-output clock generator (Cypress CY22393).

2. **FPGA + FX2LP board from ZTEX:**

More details of these boards can be found in the following location:

http://www.ztex.de/usb-fpga-1/

Features of one such board from ZTEX:

- Cypress CY7C68013A EZ-USB FX2LP Microcontroller
- Hi-Speed (480 Mbps) USB interface
- Xilinx Spartan 3 XC3S400 FPGA
- 60 general purpose I/Os (GPIO)
- 20 special I/Os (SIO)
- 128-kb EEPROM (for example, for firmware)
- Flash memory (optionally)

10.1 Third-party SDKs

ZTEX provides a SDK, which works with FX2LP-based boards and also provides JAVA-based APIs to help development of the host software. For more details, visit **http://www.ztex.de/firmware-kit/**.

11 Appendix D: Application Notes and Reference Designs

11.1 Application Notes

- **AN15456 - Guide to Successful EZ-USB® FX2LP™ and EZ-USB FX1™ Hardware Design and Debug**

This application note identifies possible USB hardware design issues, especially when operating at high-speed. It also facilitates the process of catching potential problems before building a board and assists in the debugging when getting a board up and running.

- **AN50963 - EZ-USB® FX1™/FX2LP™ Boot Options**

This application note discusses the various methods to download firmware in to FX1/FX2LP.

- **AN66806 - EZ-USB® FX2LP™ GPIF Design Guide**

This application note describes the steps necessary to develop GPIF waveforms using the GPIF Designer.

- **AN61345 - Implementing an FX2LP™- FPGA Interface**

This application note provides a sample project to interface an FX2LP with an FPGA. The interface implements Hi-Speed USB connectivity for FPGA-based applications such as data acquisition, industrial control and monitoring, and image processing. FX2LP acts in Slave-FIFO mode and the FPGA acts as the master. This application note also gives a sample FX2LP firmware for Slave-FIFO implementation and a sample VHDL and Verilog project for FPGA implementation.

- **AN57322 - Interfacing SRAM with FX2LP over GPIF**

This application note discusses how to connect the Cypress CY7C1399B SRAM to FX2LP using the General Programmable Interface (GPIF). It describes how to create read and write waveforms using GPIF Designer. This application note is also useful as a reference to connect FX2LP to other SRAMs.

- **AN58009 - Serial (UART) Port Debugging of FX1/FX2LP Firmware**

This application note describes the code needed in the FX2LP firmware for serial debugging. This code enables the developer to print debug messages and real time values of variables in a PC terminal program or to capture data in a file using the UART engine in FX2LP.

- **AN42499 - Setting Up, Using, and Troubleshooting the Keil Debugger Environment**

This application note is a step-by-step beginner's guide to using the Keil Debugger. This guide covers the serial cable connection from PC to SIO-1/0, the monitor code download, and required project settings. Additionally, it gives guidelines to start and stop a debug session, set break points, step through code, and solve potential problems.

- **AN4053 - Streaming Data through Isochronous/Bulk Endpoints on EZ-USBR FX2 and EZUSB FX2LP**

This application note provides background information for a streaming application using the EZ-USB FX2 or the EZ-USB FX2LP part. It provides information on streaming data through BULK endpoints, ISOCHRONOUS endpoints, and high bandwidth ISOCHRONOUS endpoints along with design issues to consider when using the FX2/FX2LP in high-bandwidth applications.

- **AN58069 - Implementing an 8-Bit Parallel MPEG2-TS Interface Using Slave FIFO Mode in FX2LP**

This application note explains how to implement an 8-bit parallel MPEG2-TS interface using the Slave FIFO mode. The example code uses the EZ-USB FX2LP at the receiver end and a data generator as the source for the data stream. Hardware connections and example code are included.

- **AN58170 - Code/Memory Banking Using EZ-USB**

The EZ-USBFX2 family of chips contains an 8051 core. The 8051 core has 16-bit address lines and is able to access 64KB of memory. However, some applications require more than 64KB. This application note describes methods of overcoming this 64KB boundary.

- **AN1193 - Using Timer Interrupt in Cypress EZ-USB FX2LP Based Applications**

This application note helps EZ-USBR FX2LP firmware developers to use timer interrupts in their applications.

- **AN63787 - EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples using 8-bit Asynchronous Interface**

This application note discusses how to configure the General Programmable Interface (GPIF) and slave FIFOs in EZ-USB FX2LP in both manual mode and auto mode to implement an 8-bit asynchronous parallel interface. This application note is tested with two FX2LP development kits connected back-to-back; the first one operating in master mode and the second operating in slave mode.

- **AN61244 - Firmware Optimization in EZ-USB**

This application note describes firmware optimization methods in EZ-USB. Some of these methods are common to any processor and some are specific to the 8051 core of EZ-USB FX2LP.

- **AN74505 – EZ USB FX2LP - Developing USB Application on MAC OS X using LIBUSB**

This application note describes a host application built on the MAC OS platform that uses libusb. The host application (Cocoa application) communicates with the BULK IN and BULK OUT endpoints ofFX2LP, using the interfaces provided by the APIs of libusb. This host application implements the transfer with devices that pass the particular VID/PID (=0x04B4/0x1004) identification.

- **AN58764 - Implementing a Virtual COM Port in FX2LP**

This application note explains how to implement a virtual COM port device using the standard Windows driver in FX2LP. This information helps to migrate from UART to USB.

- **AN45471 - Vendor Command Design Guide for FX2LP**

This application note demonstrates how to code USB vendor commands to perform specific product. In addition, the note explains how to use the Cypress CyConsole utility to issue vendor commands.

Reference Designs

Several reference designs of FX2LP for popular applications are available. The reference designs include demonstration source code, reference schematics, and a BOM, where appropriate, for the design.

The reference designs available on the Cypress website are:

- **CY4661 - External USB Hard Disk Drives (HDD) with Fingerprint Authentication Security**

  The CY4661 reference design kit from Cypress and UPEK provides customers with a turnkey solution for an external USB hard disk drive (HDD), with fingerprint authentication, and security to protect and authenticate data. The reference design uses UPEK's Touch Strip Fingerprint Authentication Solution (TCS3 swipe fingerprint sensor and TCD42 security ASIC).

- **FX2LP DMB-T/H TV Dongle reference design**

Appendix D: Application Notes and Reference Designs

This reference design kit is based on the Cypress FX2LP and Legend Silicon's chipset. A captured and demodulated RF signal converted to an MPEG2 TS stream by the Legend Silicon chipset is sent to the PC through an FX2LP.The PC plays these streams using a media player. This is a complete design, including all required files.

12 Appendix E: Adding Custom VID and PID to the *.inf* File

The Bulkloop firmware example uses VID 0x4B4 and PID 0x1004. These values are included in the **cyusbfx1_fx2lp.inf** file that accompanies this application note. If you change the VID and PID values in your FX2LP firmware project and if they are not already listed in the *.inf* file, then you need to add those values to allow your device to be recognized by Cypress development tools such as the USB Control Center. The following steps show how to add custom VID and PID values to **cyusbfx1_fx2lp.inf**.

Open **cyusbfx1_fx2lp.inf** file. Because this is a text file, you can use any text editor such as WordPad. Add your custom VID and PID as shown in **Figure 31** and **Figure 32**. The steps shown in these figures assume the custom VID is 0x4B4 and PID is 0x1005. Note that these steps are shown for a Windows XP, 32-bit platform.

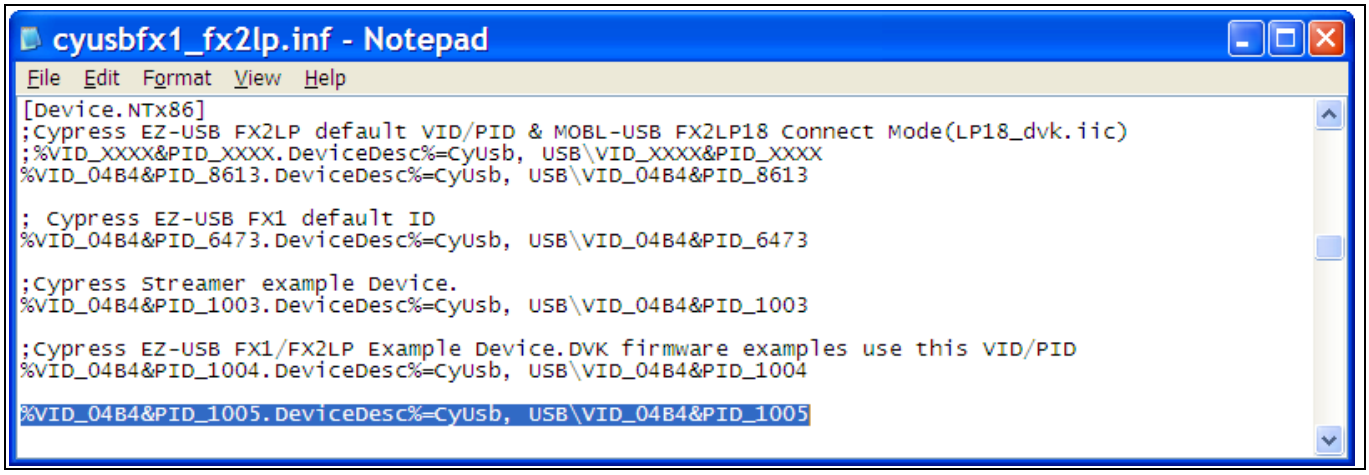


Figure 31 Adding Custom VID and PID to Cyusb.inf

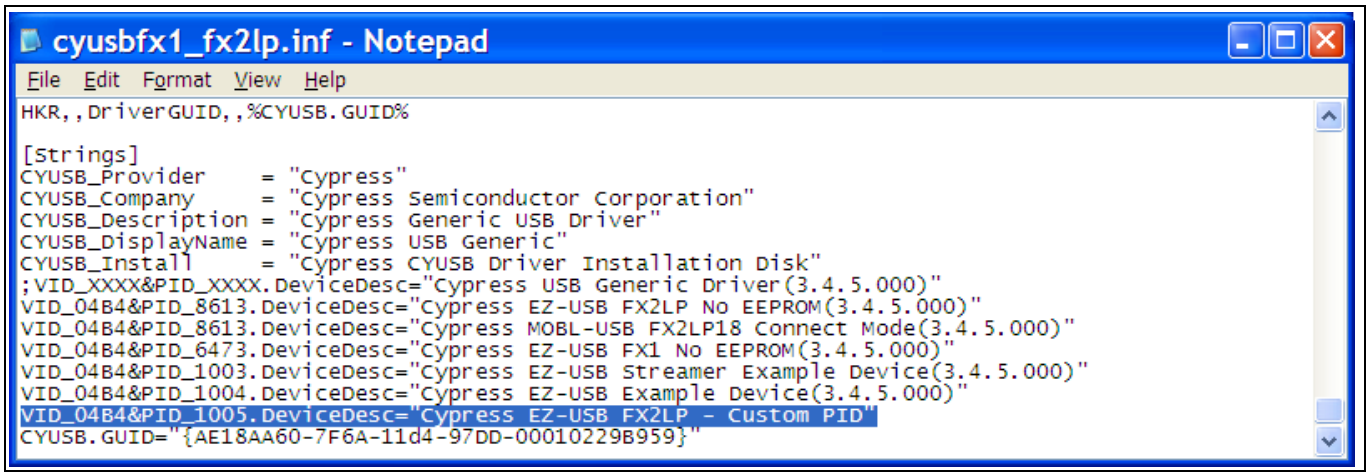


Figure 32 Adding custom VID and PID to Cyusb.inf

After making these changes, connect the FX2LP Development Board to a PC and download your firmware image (.hex file), which has VID 0x4B4 and PID 0x1005. When the driver installer asks for a driver location, you point it to the modified **cyusbfx1_fx2lp.inf** file, binding it to **CyUSB.sys**. After FX2LP is bound to **CyUSB.sys**, it will appear on the left panel of the Control Center as **Figure 33** shows. Now, you can perform data transfers using your custom VID and PID using the Control Center.

Appendix E: Adding Custom VID and PID to the .inf File

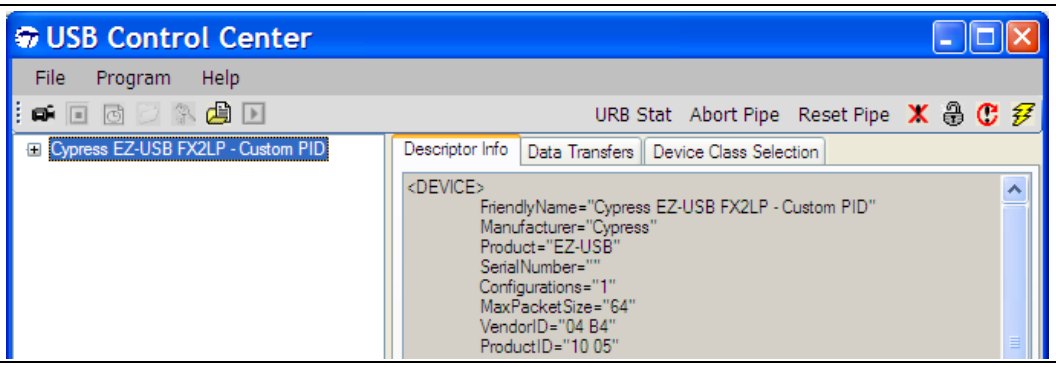


Figure 33 Custom FX2LP device listed in USB Control Center

Revision history

Document version	Date of release	Description of changes
**	2010-11-24	New application note.
*A	2011-04-12	Updated template according to current Cypress standards. Removed references to obsolete documents.
*B	2012-10-10	Re-written the application note.
*C	2013-04-22	Added Table 2 to list all the available design resources Added a section to list popular applications of FX2LP Added “Before You Start” and “About the Design” sections Restructured the entire application note Added Appendices A, B, C, D, and E.
*D	2013-12-11	Rewrote the entire application note for better clarity.
*E	2015-01-27	Included details on CyUSB Suite for Linux Removed reference to obsolete Specs Added link to USB Hi-Speed Code Examples Updated template Sunset review
*F	2017-04-19	Updated logo and copyright
*G	2017-07-19	Added the **More code examples** section.
*H	2018-08-17	Added reference to KBA222479 in the “Interfacing FPGA/ASIC using Slave FIFO” section. Updated Sales page.
*I	2021-03-17	Updated to Infineon template.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-03-19

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2021 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

001-65209 Rev.*I

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.