AN75705

Getting Started with EZ-USB® FX3™

Author: Sonia Gandhi
Associated Project: No
Associated Part Family: EZ-USB® FX3™
Software Version: SDK 1.3
Related Application Notes: For a complete list of application notes, click here

Abstract

USB enters its third generation with version 3.0, featuring 5 gigabits per second signaling (known as SuperSpeed) and compatibility with previous USB generations. AN75705 gets you started with the Cypress EZ-USB® FX3™ USB 3.0 device controller. This note highlights key FX3 features and applications while providing signposts along the way to various design resources to help with FX3 development. It also walks you through the steps to get started with FX3 firmware development using free Cypress tools featuring the Eclipse integrated development environment (IDE). Finally, it provides an overview of FX3 software and hardware development kits.

Contents

Introduction

Cypress's EZ-USB FX3 (hereafter abbreviated to FX3) is a powerful USB 3.0 peripheral controller, providing integrated and flexible features. FX3 enables developers to add USB 3.0 functionality to their systems.

AN75705 helps you get started with FX3. It highlights the key uses, applications, and features of FX3. A comprehensive list of design resources available from Cypress is also provided. This application note walks you through the steps to get started with USB transfers using the FX3 development kit (DVK) and software development kit (SDK).

Appendix A provides an overview of USB 3.0. Appendix B explains how to install the FX3 DVK Windows driver if not already installed.

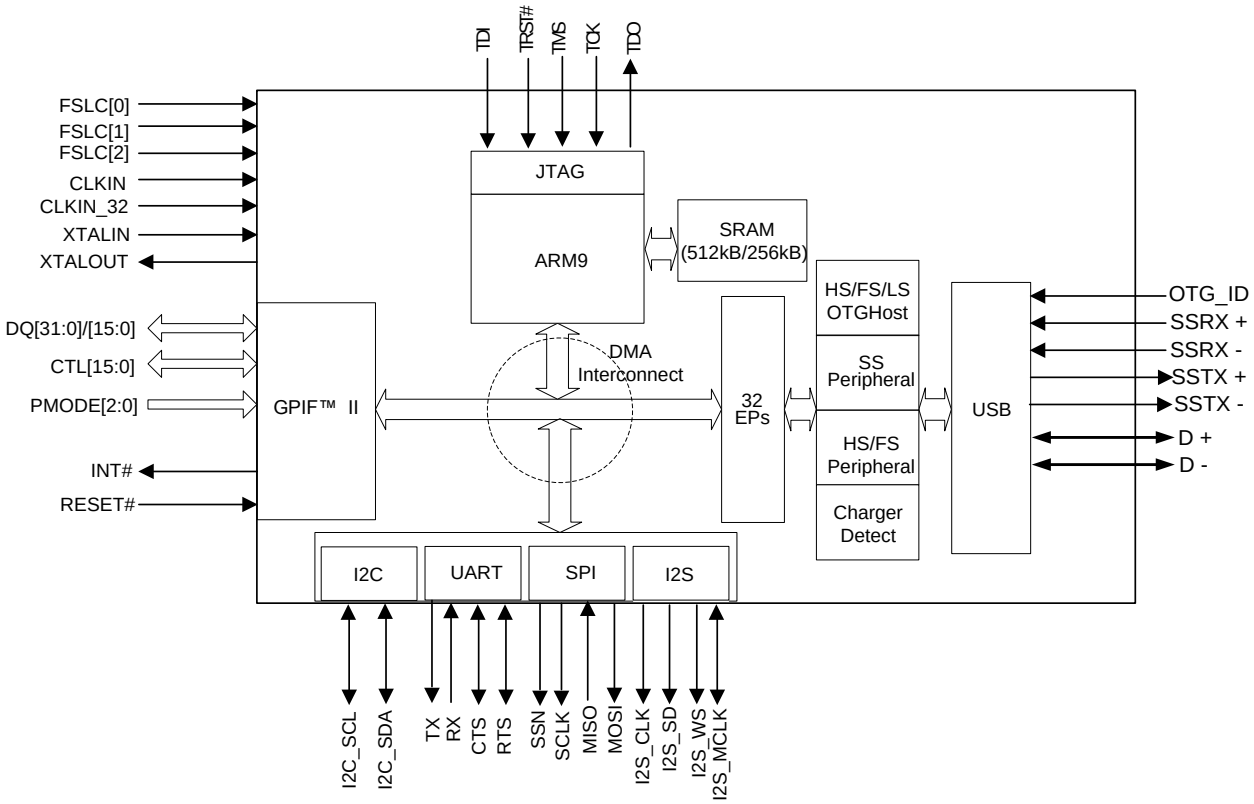
What is FX3?

FX3 is a USB 3.0 peripheral controller with an integrated ARM9 processor. Parallel and serial interfaces provide high-speed connectivity with other devices in the user system.

The main function of the FX3 device in a system is to transfer high-bandwidth data between a USB host and a peripheral device, such as a camera or scanner. The presence of a powerful ARM9 processor on-chip also allows FX3 to access the data stream and efficiently process data. In systems where FX3 is not required to perform data processing, the ARM9 firmware only initializes and manages data transfers between two interfaces—USB and a data consuming/providing device. .

FX3 has a highly flexible, programmable interface known as the General Programmable Interface Generation 2 (GPIF II) in addition to I^2C, SPI, UART, and I2S serial interfaces. GPIF II programmability allows FX3 to be connected to various types of devices including FPGAs, image sensors, ADCs, and application processors. This makes FX3 a good controller choice in a wide range of high-performance USB 3.0 applications.

Figure 1. FX3 Block Diagram



FX3 Features

This section briefly describes the key features of FX3.

USB Interface

FX3's USB interface supports the following features:

- USB SuperSpeed and Hi-Speed peripheral functionality compliant with the USB 3.0 Specification, Revision 1.0. Devices designed to this specification, such as FX3, are backward compatible with the USB 2.0 Specification.
- Compliant with the OTG Supplement, Revision 2.0. FX3 supports Hi-Speed, Full-Speed, and Low-Speed OTG dual-role device capability. As a peripheral, FX3 is capable of super-speed, high-speed, and full-speed functionality. As a host, it is capable of high-speed, full-speed, and low-speed functionality.
- Carkit Pass-through UART functionality on USB D+/D– lines based on the CEA-936A Specification.
- Up to 16 IN and 16 OUT endpoints.

GPIF II

The high-performance GPIF II (a part of the processor interface block (PIB)) enables functionality similar to, but more advanced than, FX2LP's GPIF and Slave FIFO interfaces. GPIF II is a programmable state machine that enables a flexible interface running on its own high-speed clock, autonomous to the ARM9. GPIF II may either function as a master or slave in industry-standard or proprietary interfaces. Both parallel and serial interfaces may be implemented with GPIF II.

The key features of GPIF II are:

- Functions as master or slave.
- Provides 256 programmable states.
- Supports 8-bit, 16-bit, 24-bit, and 32-bit parallel data bus.
- Supports interface frequencies up to 100 MHz.
- Supports 14 configurable I/O pins (to function as control signals) when a 32- bit data bus is used. Control pins can be input, output, or bidirectional.
- Supports 16 control I/O pins when a 16/8 data bus is used. Control pins can be input, output, or bi-directional.

Cypress's GPIF II Designer Tool enables fast development of GPIF II state machines and includes examples for common interfaces. The GPIF II Designer Tool is available with the EZ-USB FX3 SDK installation.

A popular implementation of GPIF II is a synchronous Slave FIFO interface, used in many FPGA interfaces. For details on the synchronous Slave FIFO interface, refer to AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface.

CPU

FX3 has an on-chip 32-bit, 200-MHz ARM926EJ-S core CPU. The core has direct access to 16 KB of instruction tightly coupled memory (TCM) and 8 KB of data TCM. The ARM926EJ-S core provides a JTAG interface for firmware debugging.

FX3 integrates 512 KB or 256 KB of embedded SRAM (depending on the part number selected) and supports four methods for booting the code—USB, GPIF II, I^2C, or SPI.

FX3 enables efficient and flexible DMA connectivity between its various peripherals (such as, USB, GPIF II, I^2S, SPI, and UART). After the FX3 firmware configures data accesses between peripherals, the DMA fabric manages transfers without involving the ARM9 core. The example FX3 firmware is available with the EZ-USB FX3 SDK installation.

JTAG Interface

FX3's JTAG interface has a standard five-pin interface to connect to a JTAG debugger to debug firmware through the CPU-core's on-chip-debug circuitry. Industry-standard debugging tools for the ARM926EJ-S core can be used for the FX3 application development.

UART Interface

FX3's UART interface supports full-duplex communication and consists of the TX, RX, CTS, and RTS signals. The UART is capable of generating a range of baud rates, from 300 bps to 4608 Kbps, selectable by the FX3 firmware.

I^2C Interface

FX3's I^2C interface operates as an I^2C master enabling communication with I^2C slave devices. For example, FX3 may boot from an EEPROM connected to the I^2C interface. FX3's I^2C master controller supports multi-master functionality and allows for I^2C clock stretch. The I^2C controller supports bus frequencies of 100 kHz, 400 kHz, and 1 MHz.

I^2S Interface

FX3 has an I^2S transmitter to support external audio codecs and other I^2S receivers. The sampling frequencies supported by the I^2S interface are 32 kHz, 44.1 kHz, and 48 kHz.

SPI Interface

FX3 supports an SPI master interface at a maximum operation frequency of 33 MHz. The SPI controller supports four modes of SPI communication with transaction sizes ranging from 4 bits to 32 bits.

Boot Options

FX3 can load boot images from the following sources:

USB, I^2C, SPI, GPIF II (over the synchronous ADMux, asynchronous ADMux, or asynchronous SRAM interface supported by the booloader).

AN76405 - EZ-USB FX3 Boot Options contains details for deciding on the appropriate boot method for your design.

Clocking

FX3 allows either a crystal or an external clock to be connected. The crystal frequency supported is 19.2 MHz, while the external clock frequencies supported are 19.2, 26, 38.4, and 52 MHz.

Clock inputs to FX3 must meet the phase noise and jitter requirements specified in the EZ-USB FX3 Datasheet.

Voltage Domains

FX3 has independent voltage supply domains for each of the functional blocks (GPIF II, UART/SPI/I2S, I^2C, JTAG, USB, Clock, and Core).

Voltage inputs to FX3 must meet the requirements specified in the EZ-USB FX3 Datasheet.

Refer to AN70707 - EZ-USB FX3/FX3S Hardware Design Guidelines and Schematic Checklist when designing the FX3 hardware.

Application Development with FX3

This section highlights some typical applications of FX3 in a system.

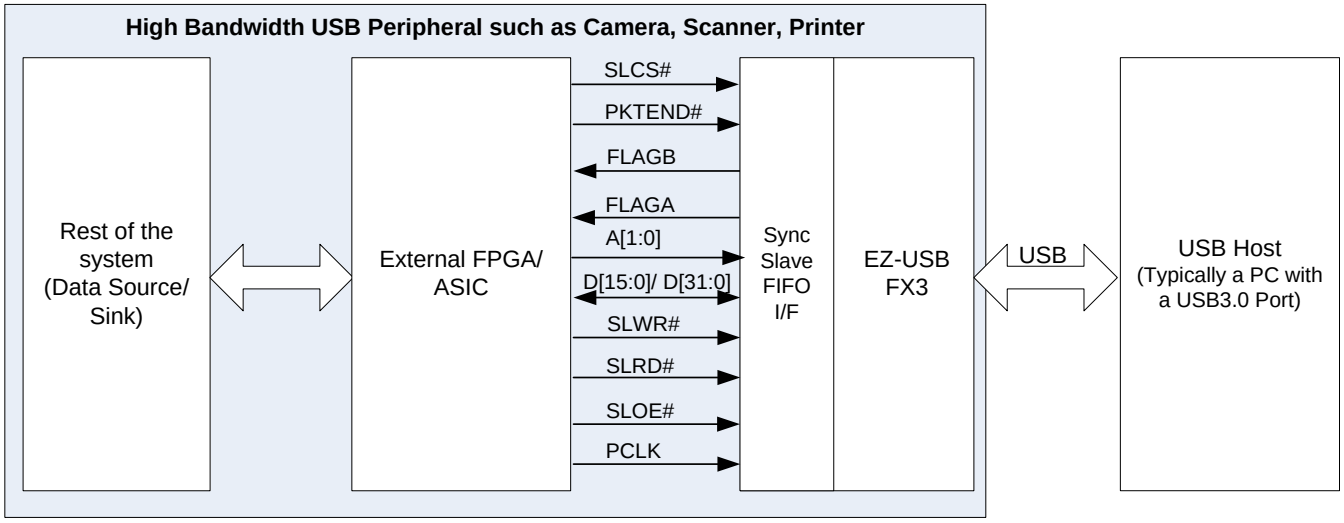
A complete FX3 design involves software, firmware, and hardware development. Cypress provides design resources for each of these aspects. FX3 design resources are listed in Table 1.

FPGA/ASIC Interfaced to FX3

Several applications connect an FPGA or ASIC to FX3 over a high-speed parallel GPIF II interface. Another device, which is the source or sink of data, is connected to the FPGA or ASIC. Examples of such applications are data acquisition devices, printers, scanners, and imaging devices.

In such applications, FX3 serves as a fast data pipe to and from a USB host. For these applications, the FX3-to-FPGA/ASIC interface is typically the synchronous Slave FIFO interface (Figure 2).

Figure 2. FPGA/ASIC Interfaced to FX3



For a detailed description of the Slave FIFO interface and an example of a complete design in which an FPGA connects to FX3, refer to AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface.

Another popular interface that the GPIF II can implement is the Synchronous Address Data Multiplexed interface.

Configuring an FPGA via FX3

In applications where FX3 connects to an FPGA, it may be preferred to load the FPGA bit file through FX3. In this case, the FPGA configuration file can be transferred from a USB host through FX3 to the FPGA. This saves the cost of an FPGA boot memory and enables FPGA reconfiguration (including updates) from the PC. For an example implementation, refer to AN84868 - Configuring an FPGA over USB Using Cypress EZ-USB FX3.

Figure 3. Configuring an FPGA via FX3

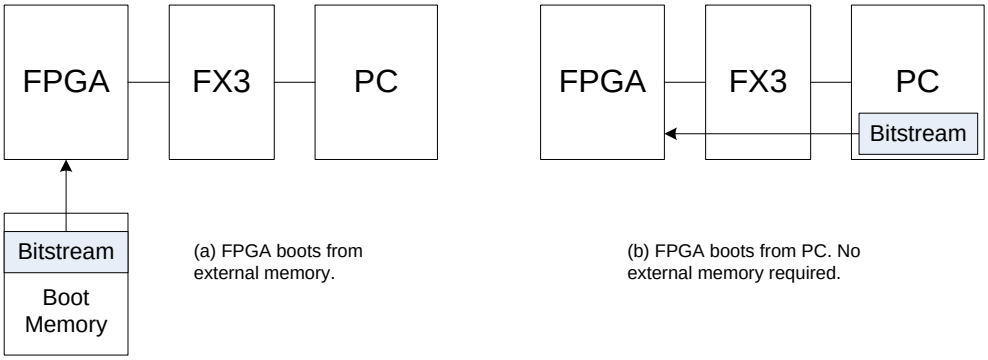
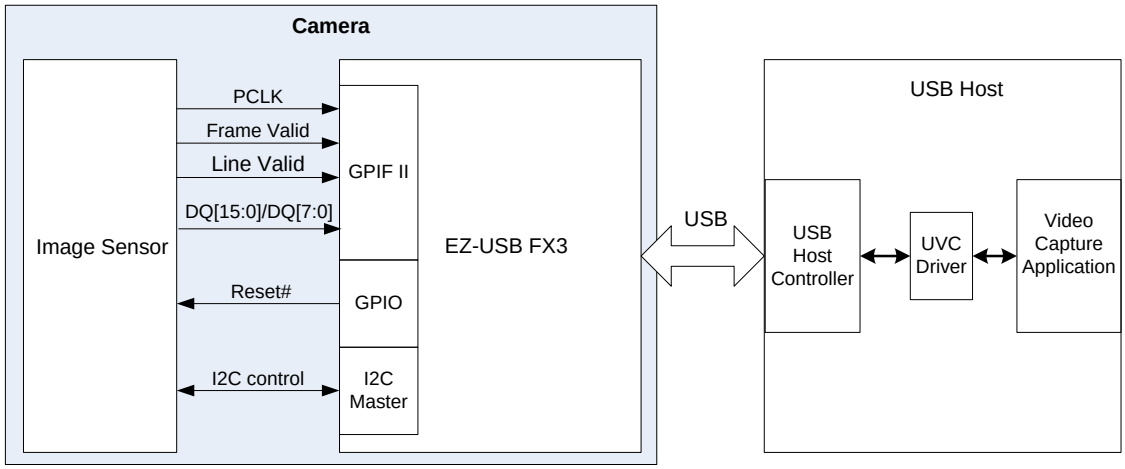


Image Sensor Interfaced to FX3

In imaging applications, an image sensor is connected directly to FX3 over a parallel GPIF II interface and video is streamed from the sensor to a USB host through FX3.

The system can be made compatible with the USB Video Class by programming the FX3 with the appropriate firmware. For an example of such an application, refer to AN75779 - How to Implement an Image Sensor Interface with EZ-USB FX3 in a USB Video Class (UVC) Framework.

Figure 4. Image Sensor Interfaced to FX3



Designing FX3 Hardware

Because of the speed and power of USB 3.0, pay careful attention to PC design and layout. Refer to AN70707 - EZ-USB FX3/FX3S Hardware Design Guidelines and Schematic Checklist when designing the FX3 hardware. For an example schematic design, you may also refer to the FX3 DVK Schematic.

Upgrading an FX2LP Design to FX3

If you are upgrading an existing FX2LP design to use FX3, refer to AN76348 - Differences in Implementation of EZ-USB FX2LP and EZ-USB FX3 Applications.

Design Resources

Table 1 lists the various resources available from Cypress to help with development using FX3.

Table 1. FX3 Design Resources

Design	Resources	Where to Find Resources
FX3 Firmware	Eclipse IDE installation with GCC compiler	Included with EZ-USB FX3 SDK installation
	APIs for performing various functions	
	Firmware examples	
	Documentation on using the SDK	FX3 Programmer's Manual
	Documentation on all APIs provided in the SDK	FX3 SDK API Guide
	Sync Slave FIFO firmware example in a complete design with FPGA	AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface
	UVC compatible firmware example	AN75779 - How to Implement an Image Sensor Interface with EZ-USB FX3 in a USB Video Class (UVC) Framework
Host Software	USB3.0 driver – cyusb3.sys	Available with EZ-USB FX3 SDK installation
	Host application examples - Control Center and Streamer applications	
	Cypress USBSuite Application Development - Quick Start Guide	Cypress USBSuite Application Development - Quick Start Guide
GPIF II Interface Design	Using the GPIF II Designer Tool, you can design a custom GPIF II interface using state machine entry. The tool then generates the necessary code to be integrated into your FX3 firmware.	Included with EZ-USB FX3 SDK installation
	Examples of popular GPIFII implementations including Slave FIFO, SRAM slave, and ADMux slave	GPIFII Designer Tool - included with EZ-USB FX3 SDK installation
	GPIF II documentation, including instructions for using the tool	GPIFII Design Guide - available with GPIFII Designer Tool (EZ-USB FX3 SDK installation)
Hardware Design	Development board – Schematic, board files, and documentation	FX3 DVK Design (Schematics, CAD, Gerber files) DVK User Guide
	Hardware design guidelines including recommendations for crystals, power supply decoupling capacitors, and PCB layout	AN70707 - EZ-USB FX3/FX3S Hardware Design Guidelines and Schematic Checklist
	IBIS model	FX3 IBIS Model
Firmware Debug	Setting up and using the JTAG debugger	Chapter 12 of FX3 Programmer's Manual
Other Collateral		
Application Notes	EZ-USB FX3 Application Notes	
Training Videos	EZ-USB FX3 Architecture Overview (available on the same web page as this application note)	
	Designing an image sensor interface with the GPIF™ II Designer - Part 1	
	Designing an image sensor interface with the GPIF™ II Designer - Part 2	
	Designing an image sensor interface with the GPIF™ II Designer - Part 3	
Knowledge Base Articles	EZ-USB FX3 Knowledge Base Articles	

FX3 Terminology

To understand the data transfer in and out of FX3, it is important to know the following terms:

- Socket
- DMA Descriptor
- DMA buffer
- GPIF thread

A **socket** is a point of connection between a peripheral hardware block and the FX3 RAM. Each peripheral hardware block on FX3, such as USB, GPIF, UART, and SPI, has a fixed number of sockets associated with it. The number of independent data flows through a peripheral is equal to the number of its sockets. The socket implementation includes a set of registers that point to the active DMA descriptor and enable or flag interrupts associated with the socket.

A **DMA Descriptor** is a set of registers allocated in the FX3 RAM. It holds information about the address and size of a DMA buffer as well as pointers to the next DMA Descriptor. These pointers create DMA Descriptor chains.

A **DMA buffer** is a section of RAM used for intermediate storage of data transferred through the FX3 device. DMA buffers are allocated from the RAM by the FX3 firmware, and their addresses are stored as part of DMA Descriptors.

A **GPIF thread** is a dedicated data path in the GPIF II block that connects the external data pins to a socket.

Sockets can directly signal each other through events or they can signal the FX3 CPU via interrupts. The firmware configures this signaling. Take, for example, a data stream from the GPIF II block to the USB block. The GPIF socket can tell the USB socket that it has filled data in a DMA buffer, and the USB socket can tell the GPIF socket that a DMA buffer is empty. This implementation is called an automatic DMA channel. The automatic DMA channel implementation is typically used when the FX3 CPU does not have to modify any data in a data stream.

Alternatively, the GPIF socket can send an interrupt to the FX3 CPU to notify it that the GPIF socket filled a DMA buffer. The FX3 CPU can relay this information to the USB socket. The USB socket can send an interrupt to the FX3 CPU to notify it that the USB socket emptied a DMA buffer. Then, the FX3 CPU can relay this information back to the GPIF socket. This is called the manual DMA channel implementation. This implementation is typically used when the FX3 CPU has to add, remove, or modify data in a data stream. The firmware example of the UVC application note (AN75779) uses the manual DMA channel implementation because the firmware needs to add the UVC video data header.

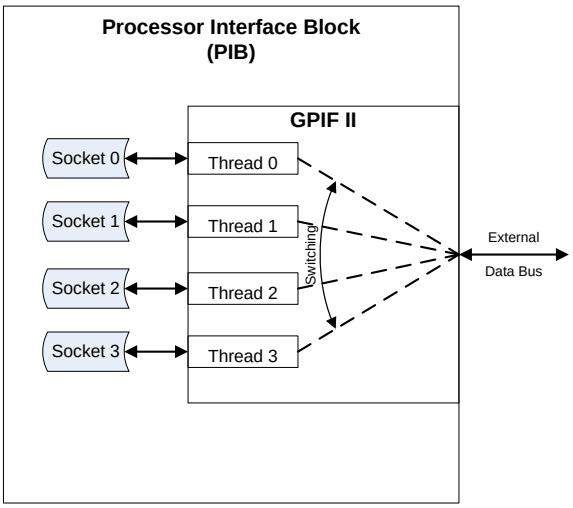
A socket that writes data to a DMA buffer is called a producer socket. A socket that reads data from a DMA buffer is called a consumer socket. A socket uses the values of the DMA buffer address, DMA buffer size, and DMA Descriptor chain stored in a DMA Descriptor for data management. A socket takes a finite amount of time (up to a few microseconds) to switch from one DMA Descriptor to another after it fills or empties a DMA buffer. The socket cannot transfer any data while this switch is in progress. This latency can be a problem for interfaces that have no flow control. One such example is an image sensor interface.

This issue is addressed in the GPIF II block using multiple GPIF threads. The GPIF II block implements four GPIF threads. Only one GPIF thread can transfer data at a time. The GPIF II state machine must select an active GPIF thread to transfer data.

The GPIF thread selection mechanism is like a mux. The GPIF II state machine uses internal control signals or external inputs (address lines A1, A0 in case of Slave FIFO 2-bit application note) to select the active GPIF thread. Switching the active GPIF thread switches the active socket for the data transfer, thereby changing the DMA buffer used for data transfers. The GPIF thread switch has no latency.

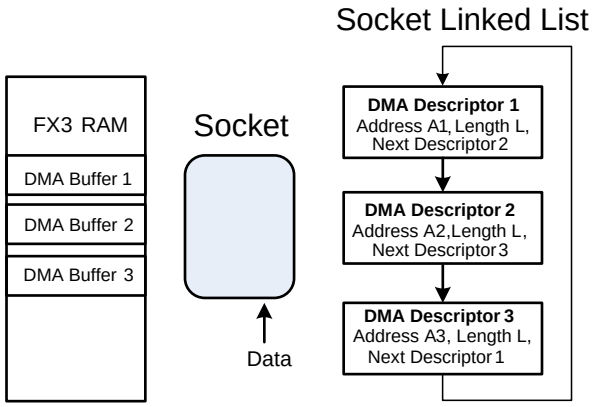
The default mapping of the sockets and GPIF threads is shown in Figure 5 — Socket 0 to GPIF thread 0, Socket 1 to GPIF thread 1, Socket 2 to GPIF thread 2, and Socket 3 to GPIF thread 3.

Figure 5. Default GPIF II Socket/Thread Mapping



To understand DMA transfers, the concept of a socket is illustrated in the following four figures.

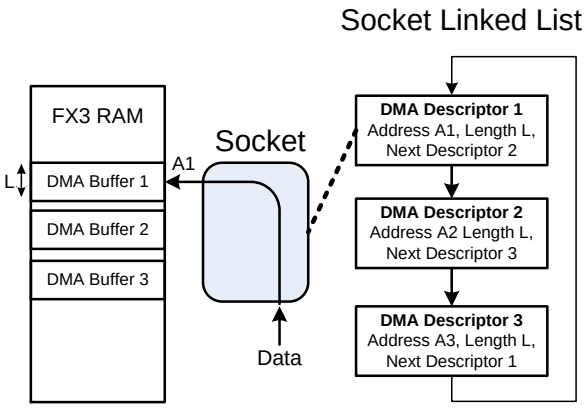
Figure 6. Socket Routes Data According to a List of DMA Descriptors



The socket linked list is a set of data structures in the main memory called DMA descriptors. Each descriptor specifies a DMA buffer address and length as well as a pointer to the next DMA descriptor. As the socket operates, it retrieves the DMA descriptors individually, routing the data to the DMA buffer specified by the descriptor address and length. When L bytes are transferred, the socket retrieves the next descriptor and continues transferring bytes to a different DMA buffer.

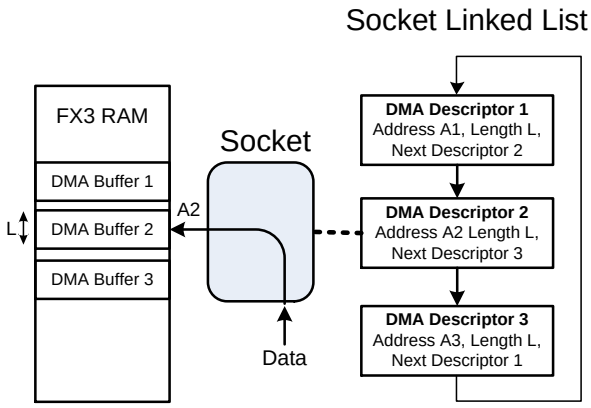
This structure makes a socket versatile because any number of DMA buffers can be created anywhere in memory and can be automatically chained together. For example, the socket in Figure 6 retrieves DMA descriptors in a repeating loop.

Figure 7. Socket Operating with DMA Descriptor 1



In Figure 7, the socket has loaded DMA Descriptor 1, which tells it to transfer bytes starting at A1 until it has transferred L bytes, at which time it retrieves DMA Descriptor 2. Then, it continues with its address and length settings A2 and L (Figure 8).

Figure 8. Socket Operating with DMA Descriptor 2



In Figure 9 the socket retrieves the third DMA descriptor and transfers data, starting at A3. When it has transferred L bytes, the sequence repeats with DMA Descriptor 1.

Figure 9. Socket Operating with DMA Descriptor 3

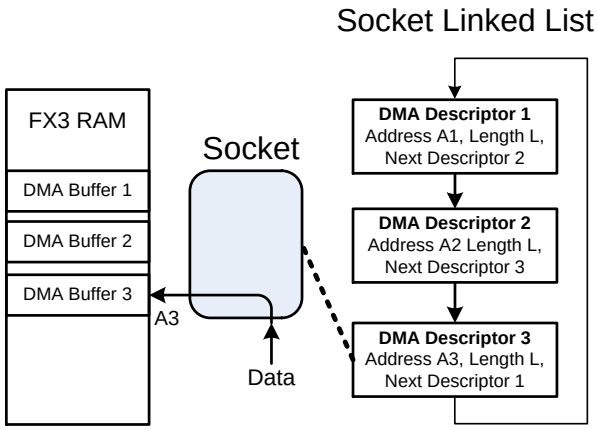
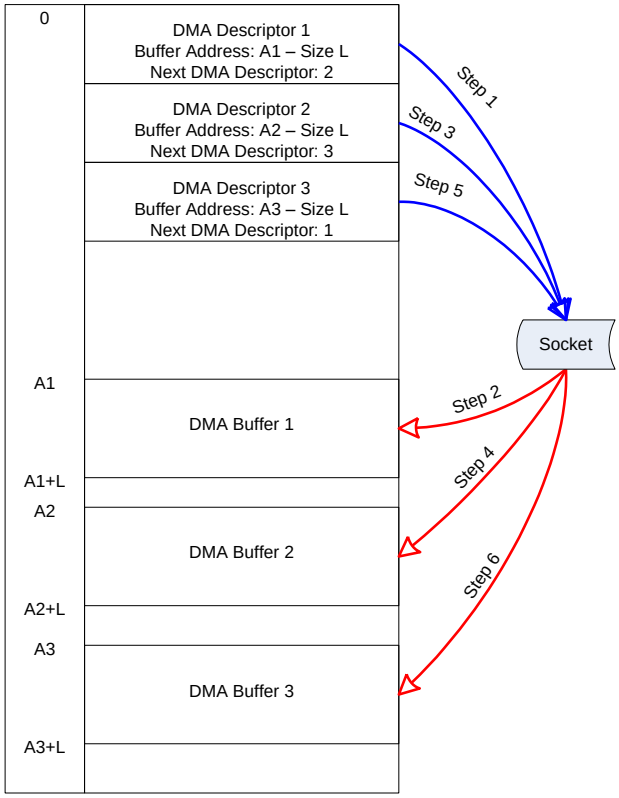


Figure 10 shows a DMA data transfer in detail. Take, for example, an application using three DMA buffers of length L chained in a circular loop. The FX3 memory addresses are on the left. The blue arrows show the socket loading the socket linked list descriptors from the memory. The red arrows show the resulting data paths. The following steps show the socket sequence as the data is moved to the internal DMA buffers.

Step 1: Load DMA Descriptor 1 from the memory into the socket. Get the DMA buffer location (A1), DMA buffer size (L), and the next descriptor (DMA Descriptor 2) information. Go to step 2.

Step 2: Transfer data to the DMA buffer location starting at A1. After transferring DMA buffer size L amount of data, go to step 3.

Figure 10. DMA Transfer Example



Step 3: Load DMA Descriptor 2 as pointed to by the current DMA Descriptor 1. Get the DMA buffer location (A2), DMA buffer size (L), and the next descriptor (DMA Descriptor 3) information.

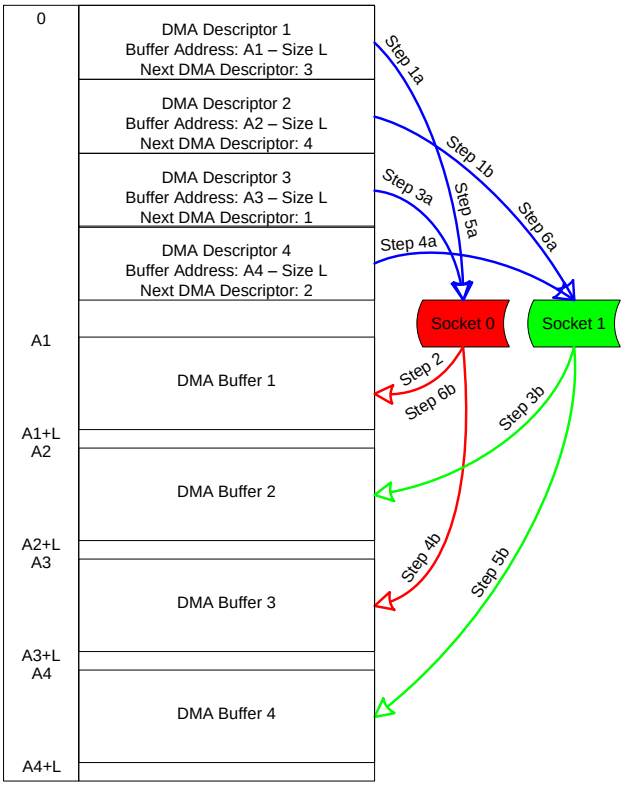
Step 4: Transfer data to the DMA buffer location starting at A2. After transferring DMA buffer size L amount of data, go to step 5.

Step 5: Load DMA Descriptor 3 as pointed to by the current DMA Descriptor 2. Get the DMA buffer location (A3), DMA buffer size (L), and next Descriptor (DMA Descriptor 1) information.

Step 6: Transfer data to the DMA buffer location starting at A3. After transferring DMA buffer size L amount of data, go to step 1.

This simple scheme causes data loss if the external peripheral sends data when the socket retrieves the next DMA descriptor from the memory, typically 1 microsecond. A better solution is to take advantage of the fact that the sockets can be switched without latency—in one clock cycle. Therefore, it makes sense to use two sockets if the external peripheral does not have any flow control mechanism. Data transfer using dual sockets is described in Figure 11, with numbered execution steps. The access of Socket0 and Socket1 to the DMA buffers is differentiated by red and green arrows (data paths for individual sockets), respectively. The ‘a’ and ‘b’ parts of each step occur simultaneously. This parallel operation of the hardware eliminates DMA descriptor retrieval dead time and allows the GPIF II to stream data continuously into internal memory.

Figure 11. Dual Sockets Yield Seamless Transfers



Step 1: At initialization of the sockets, Socket 0 and Socket 1, load the DMA Descriptor 1 and DMA Descriptor 2, respectively.

Step 2: As soon as the data is available, Socket 0 transfers the data to DMA Buffer 1. The transfer length is L. At the end of this transfer, go to step 3.

Step 3: GPIF II switches the GPIF thread and, therefore, the socket for data transfer. Socket 1 starts to transfer data to DMA Buffer 2, and, at the same time, Socket 0 loads the DMA Descriptor 3. By the time Socket 1 finishes transferring L amount of data, Socket 0 is ready to transfer data to DMA Buffer 3.

Step 4: GPIF II now switches back to the original GPIF thread. Socket 0 now transfers the data of length L into DMA Buffer 3. At the same time, Socket 1 loads the DMA Descriptor 4, making it ready to transfer data to DMA Buffer 4. After Socket 0 finishes transferring the data of length L, go to step 5.

Step 5: GPIF II routes the Socket 1 data into DMA Buffer 4. At the same time, Socket 0 loads DMA Descriptor 1 to prepare to transfer data into DMA Buffer 1. Note that Step 5a is the same as Step 1a except that Socket 1 is not initializing but, rather, transferring data simultaneously.

Step 6: GPIF II switches sockets again and Socket 0 starts to transfer data of length L into DMA Buffer 1. It is assumed that by now, the DMA buffer is empty, having been depleted by the UIB consumer socket. At the same time, Socket 1 loads DMA Descriptor 2 and is ready to transfer data into DMA Buffer 2. The cycle now goes to Step 3 in the execution path.

GPIF II sockets can transfer data only if the consuming side (USB) empties and releases the DMA buffers in time to receive the next chunk of data from GPIF II. If the consumer is not fast enough, the sockets drop data because their DMA buffer writes are ignored.

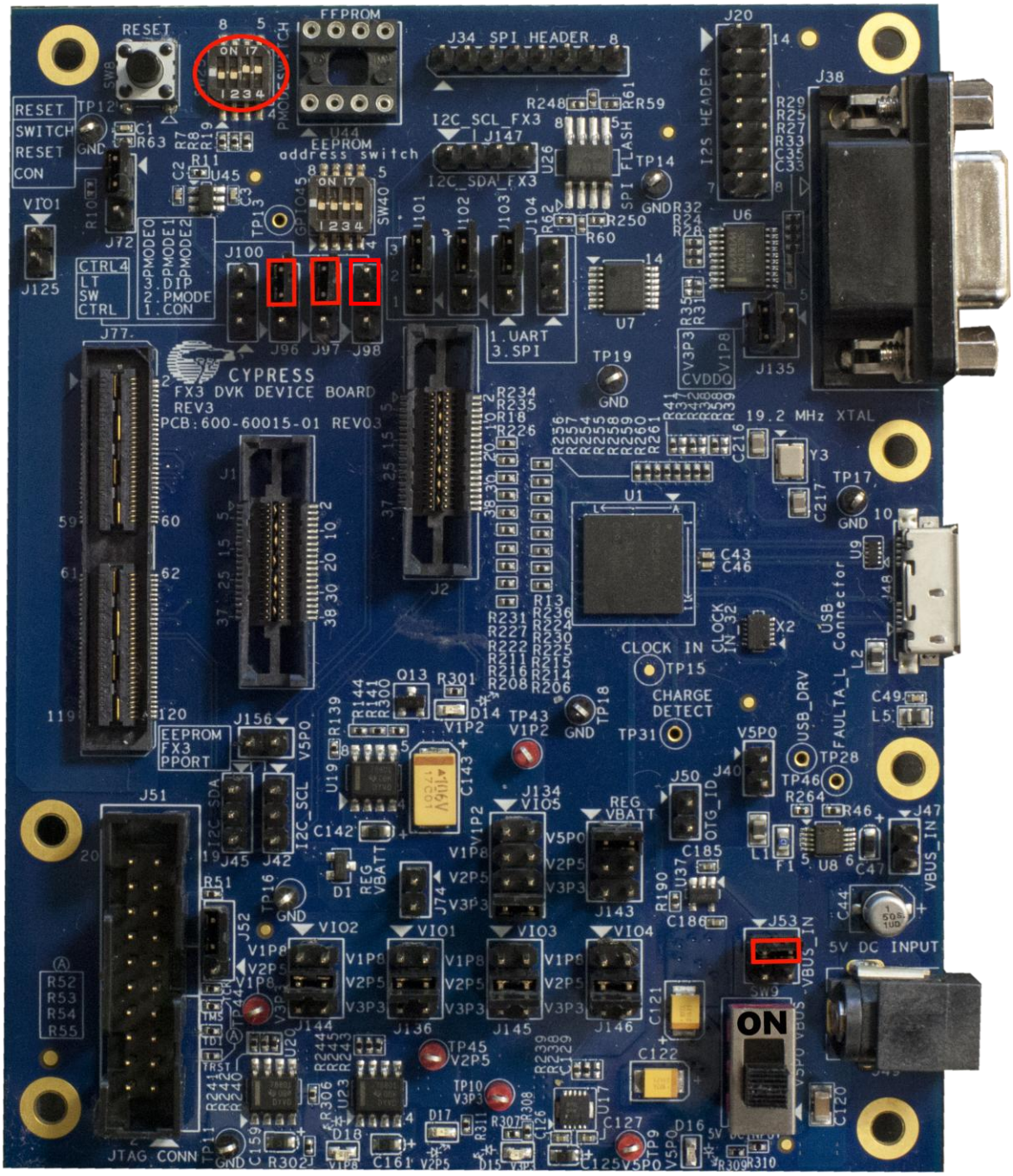
My First USB 3.0 Transfer using FX3

This section walks you through building and running a simple firmware example, which enables you to perform BULK OUT and BULK IN transfers from a USB host to the FX3 device.

Before starting this exercise, do the following:

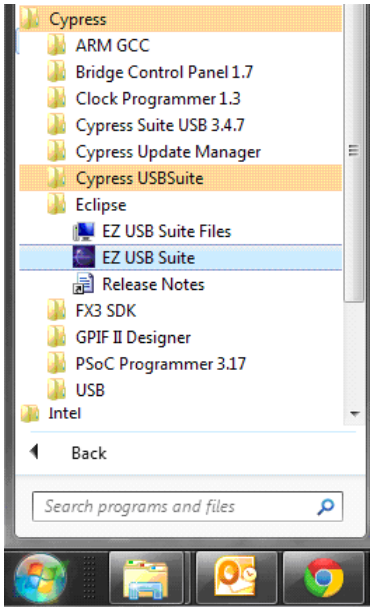
1. Obtain a EZ-USB FX3 Development Kit. In this exercise, the FX3 firmware is downloaded from the USB host, so the FX3 DVK board must be configured for USB boot. To select USB boot, make sure the three jumpers and four switches are set according to Figure 12. These jumper settings are highlighted with a red box or circle. For details on all the other DVK jumper settings, refer to the DVK User Guide. The FX3 DVK board can be powered from an external power supply (self-powered) or from the USB cable (bus-powered). For convenience, this example is bus-powered, requiring only the USB cable to be connected to a USB host. The lower right jumper, shown in Figure 12, selects this mode. The other jumpers and switches configure the board for FX3 code download over USB. The power switch (lower left) is a convenient way to disconnect and reconnect the DVK board quickly from the host PC.

Figure 12. FX3 DVK Board Setting for USB Boot



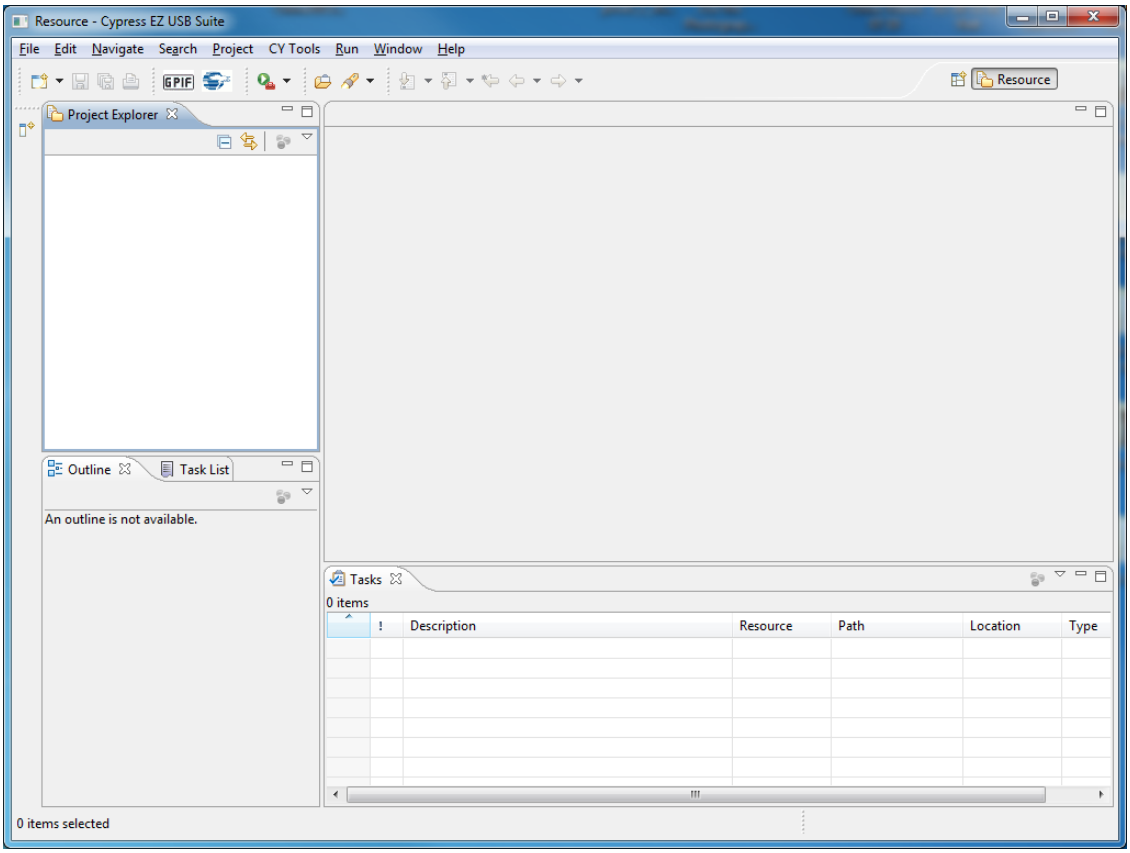
2. Install the EZ-USB FX3 SDK. When asked to choose the **Installation Type**, select the default **Typical** setting. After installing the FX3 SDK, import the firmware examples into the Eclipse IDE workspace (an Eclipse IDE installation is provided as part of the SDK). To import all of the firmware examples into the IDE, follow these steps:
   a. Open the Eclipse IDE by going to your Start menu. Select **All Programs > Cypress > Eclipse > EZ USB Suite.** Accept the default workspace folder; however, you can define your own location later.

Figure 13. Open Eclipse IDE from Start Menu



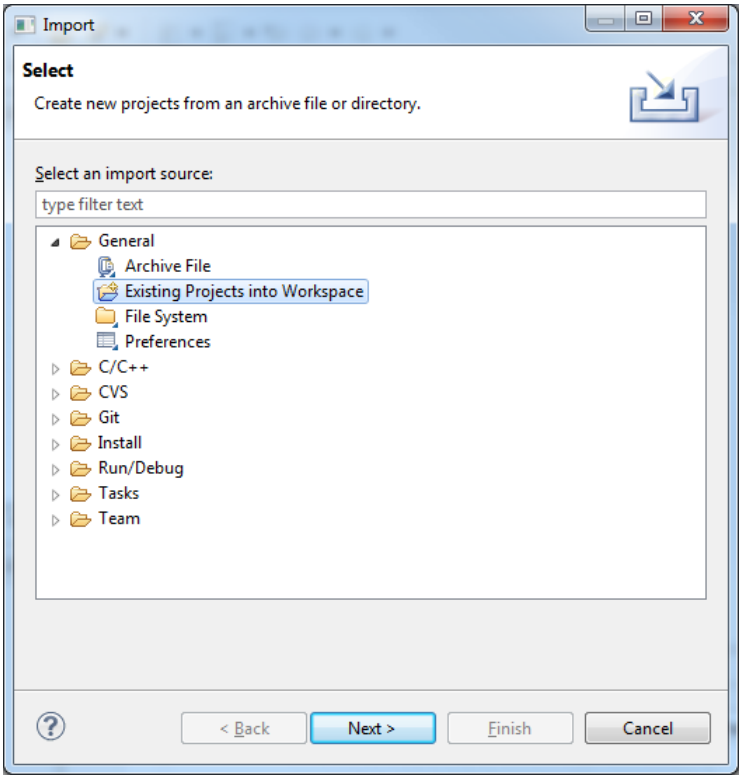
b. This starts an empty Eclipse IDE with no projects in the Project Explorer tab. The next step is to import the Cypress example projects into the Project Explorer.

Figure 14. Eclipse IDE before Importing Projects



c. Select **File > Import.**
d. Select **General > Existing Projects into Workspace** and click **Next.**

Figure 15. Select Existing Projects into Workspace



e. Browse to the firmware directory in your SDK installation. Click the **Browse…** button and navigate to the Cypress SDK **firmware** folder. For a typical Windows installation, this folder is located at `C:\Program Files (Program Files (x86 on 64-bit machines)\Cypress\EZ-USB FX3 SDK\firmware`.

Figure 16. Select Root Directory

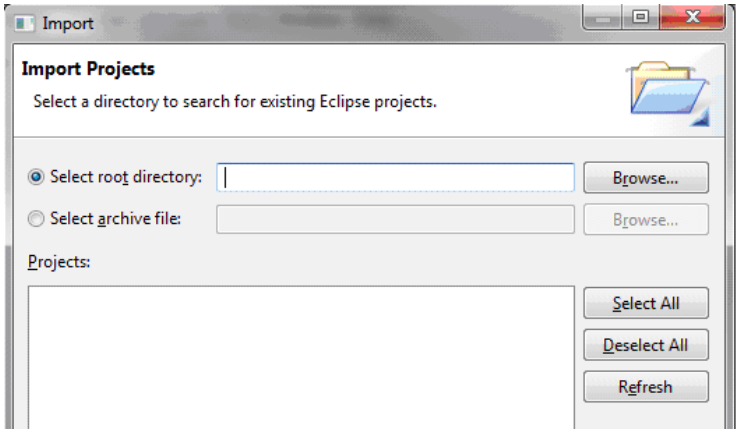
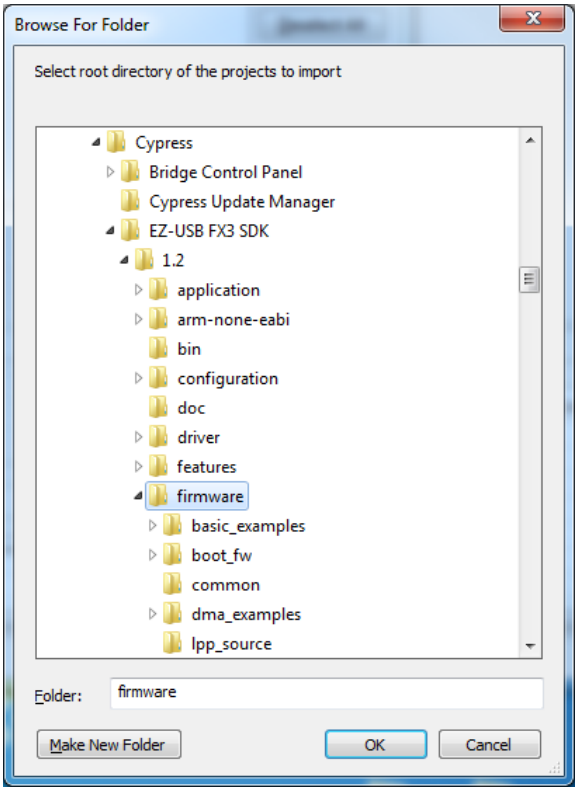
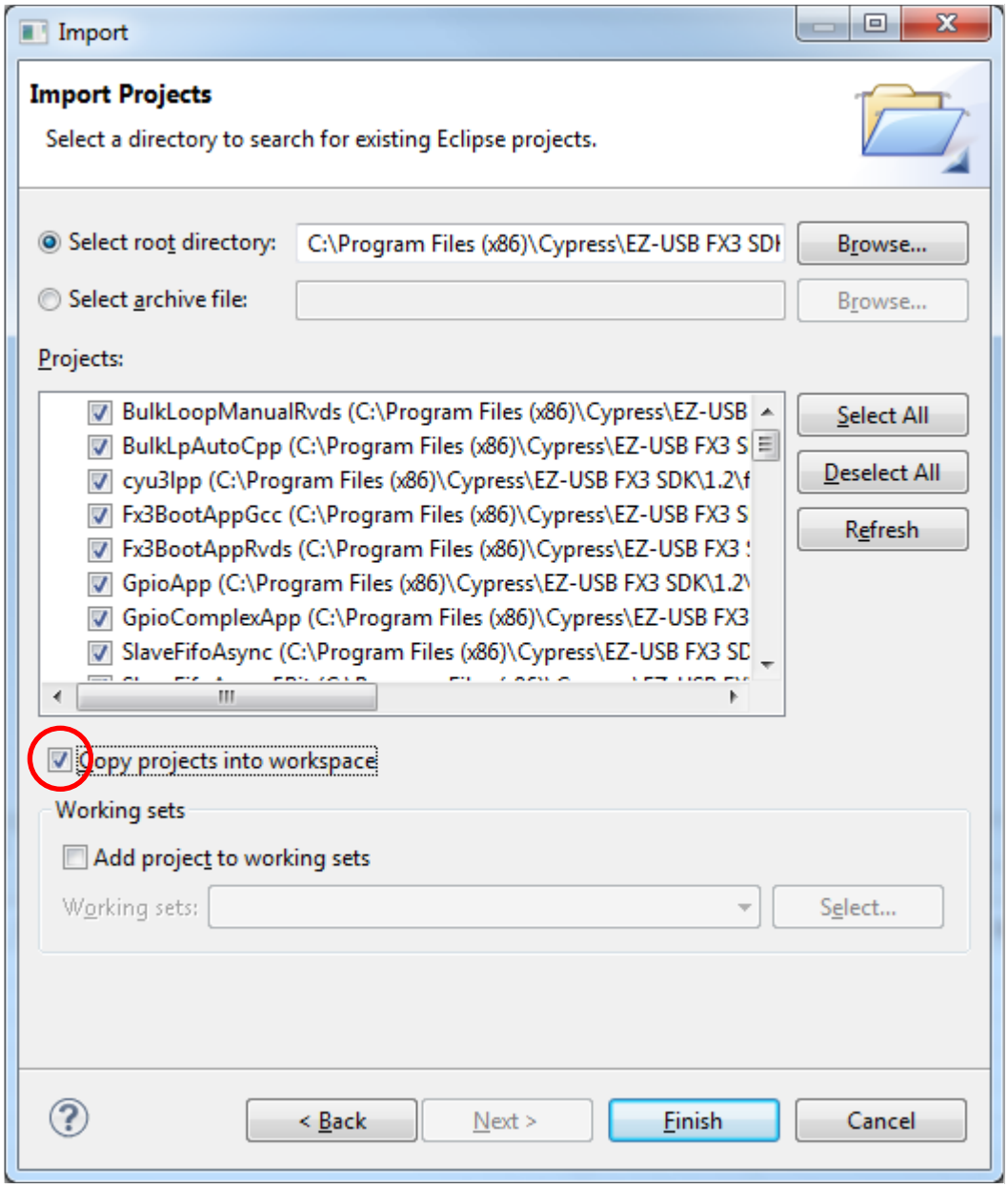


Figure 17. Browse to “firmware” Directory in SDK Installation



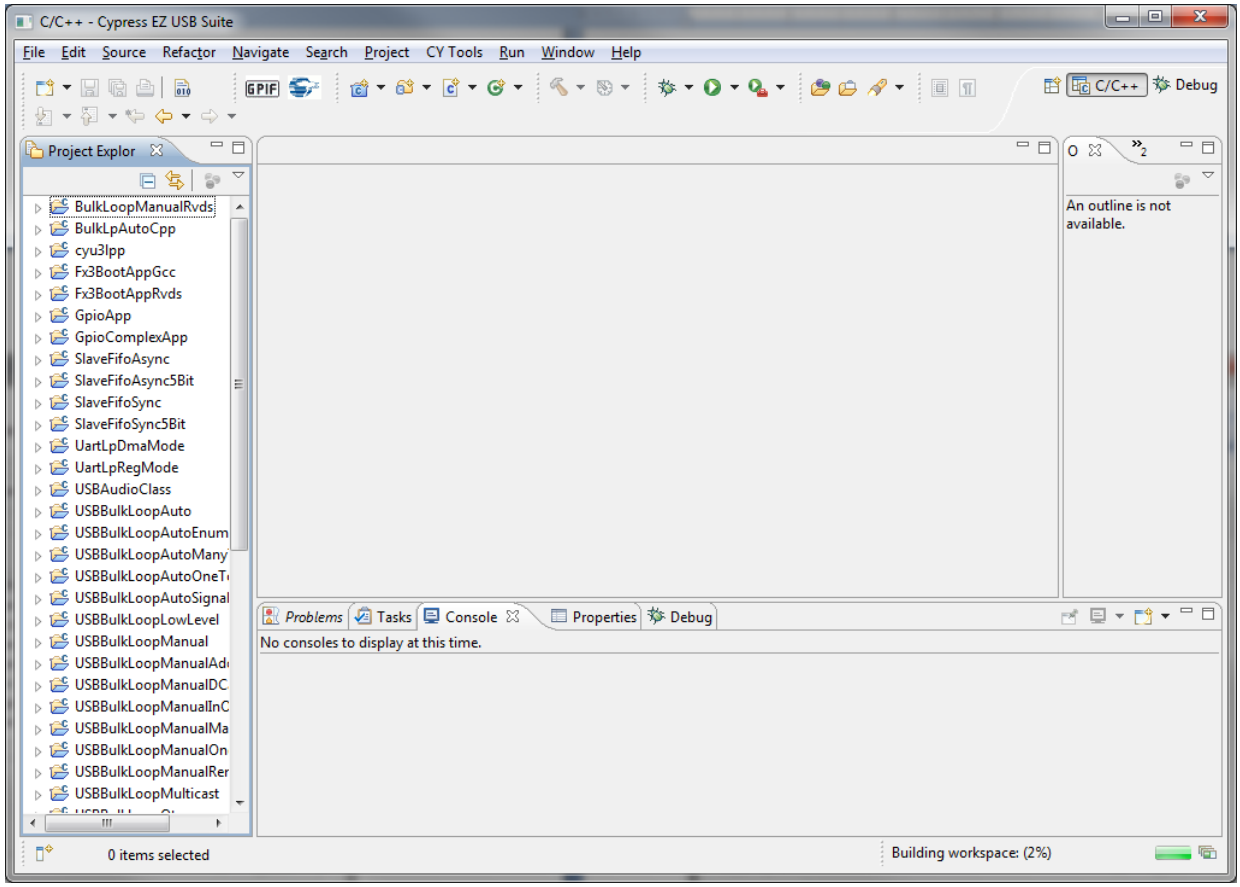
f. All the Cypress example projects should be selected with check marks. Otherwise, press the **Select All** button. Also, check **Copy projects into workspace**, then click **Finish**.

Figure 18. Select All Projects in Firmware Directory



g. All the firmware examples are imported into the workspace and become visible in the Eclipse Project Explorer. Because the projects are built automatically on importing, it may take several minutes for the import operation to load and build them all. When loaded, you can run them as they are or modify and rebuild them.

Figure 19. All Projects Imported into Workspace



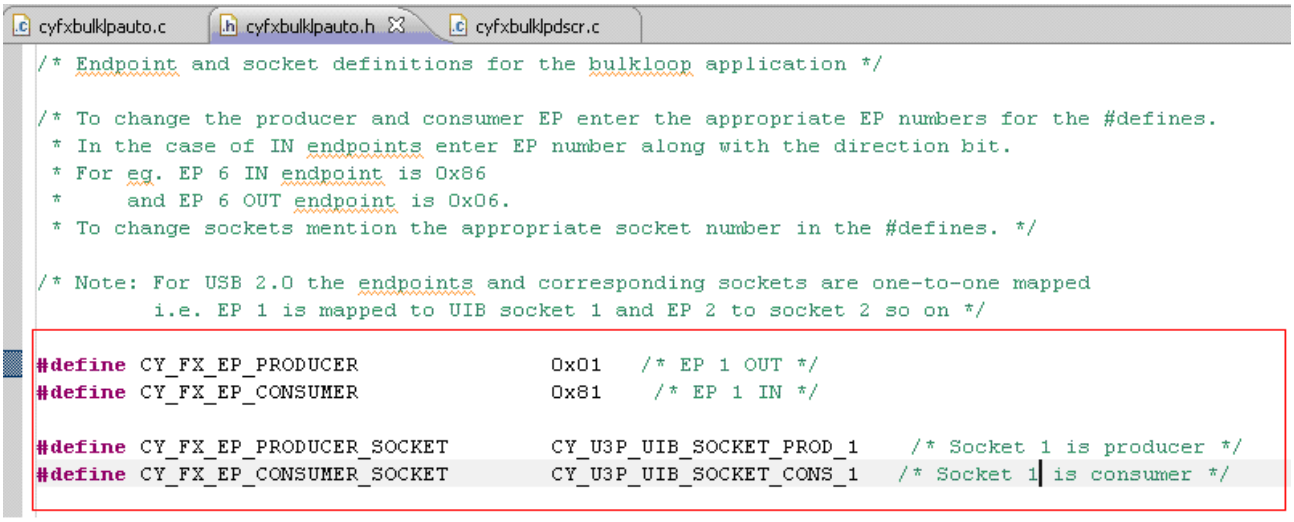
Building and Running the USBBulkLoopAuto Firmware

The next part of the exercise is to build and run a basic firmware example.

The USBBulkLoopAuto firmware example provided in the FX3 SDK configures BULK endpoint 1 IN and BULK endpoint 1 OUT by default. The data sent by the USB host to the OUT endpoint is looped back to the IN endpoint by the FX3 firmware. The same data can be received back by the USB host from the IN endpoint.

1. Open the USBBulkLoopAuto firmware in the Eclipse IDE by double-clicking its name or clicking on the little expansion arrow to the left of its name. You can build this firmware with its default settings or modify it bys changing the endpoint numbers as explained in the following steps. To run this firmware without making any changes, jump to step 5. To make a minor modification and then run, continue on to step 2. The modification steps are recommended to give a more detailed look at the Eclipse tool chain.
2. In the list of files, double-click **cyfxbulkloopauto.h** to bring it up in the editor window. The endpoint numbers and their associated sockets are defined in *cyfxbulkloopauto.h*, using identifiers CY_FX_EP_PRODUCER, CY_FX_EP_CONSUMER, CY_FX_EP_PRODUCER_SOCKET, and CY_FX_EP_CONSUMER_SOCKET near the end of the *.h* file.

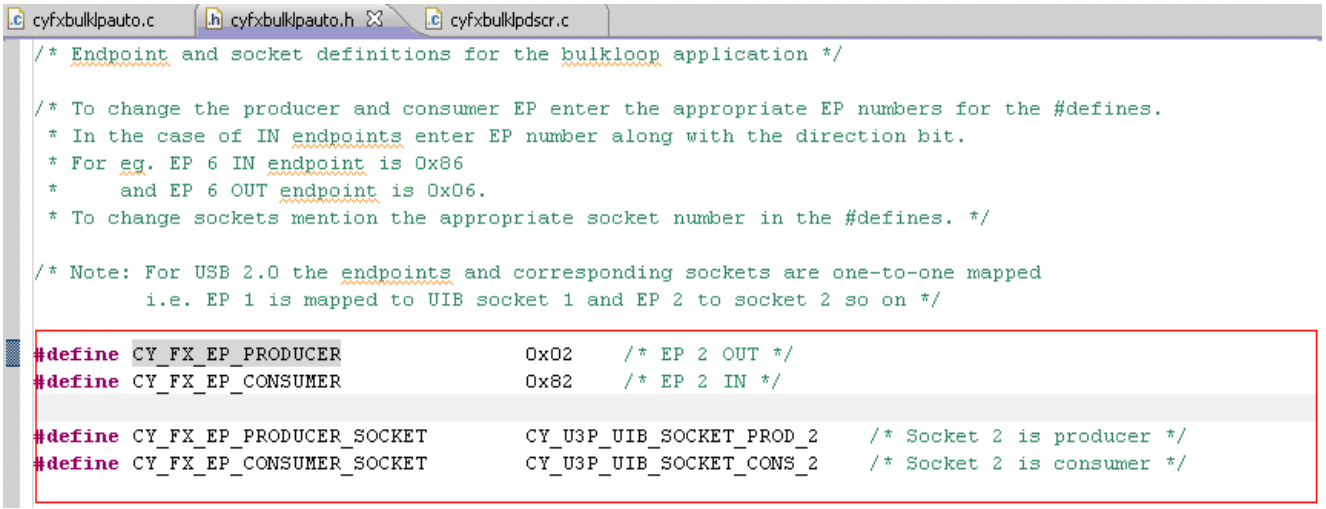
Figure 20. Endpoint and Socket Definitions in cyfxbulklpauto.h



3. Using the identifiers highlighted above, the endpoint numbers and associated sockets are changed from EP1 to EP2, as shown in Figure 21. Now EP2 IN is the BULK IN and EP2 OUT is BULK OUT in the modified firmware.

   Note In USB, an endpoint is identified by a byte comprising bit 7 as the direction (1=in, 0=out) and a 7-bit address in bits 6:0. An FX3 "socket" is equivalent to an endpoint in the context of USB transfers. The socket in the USB domain interfaces with a socket in another peripheral domain (for example, the GPIF II or even the CPU) for data transfers.

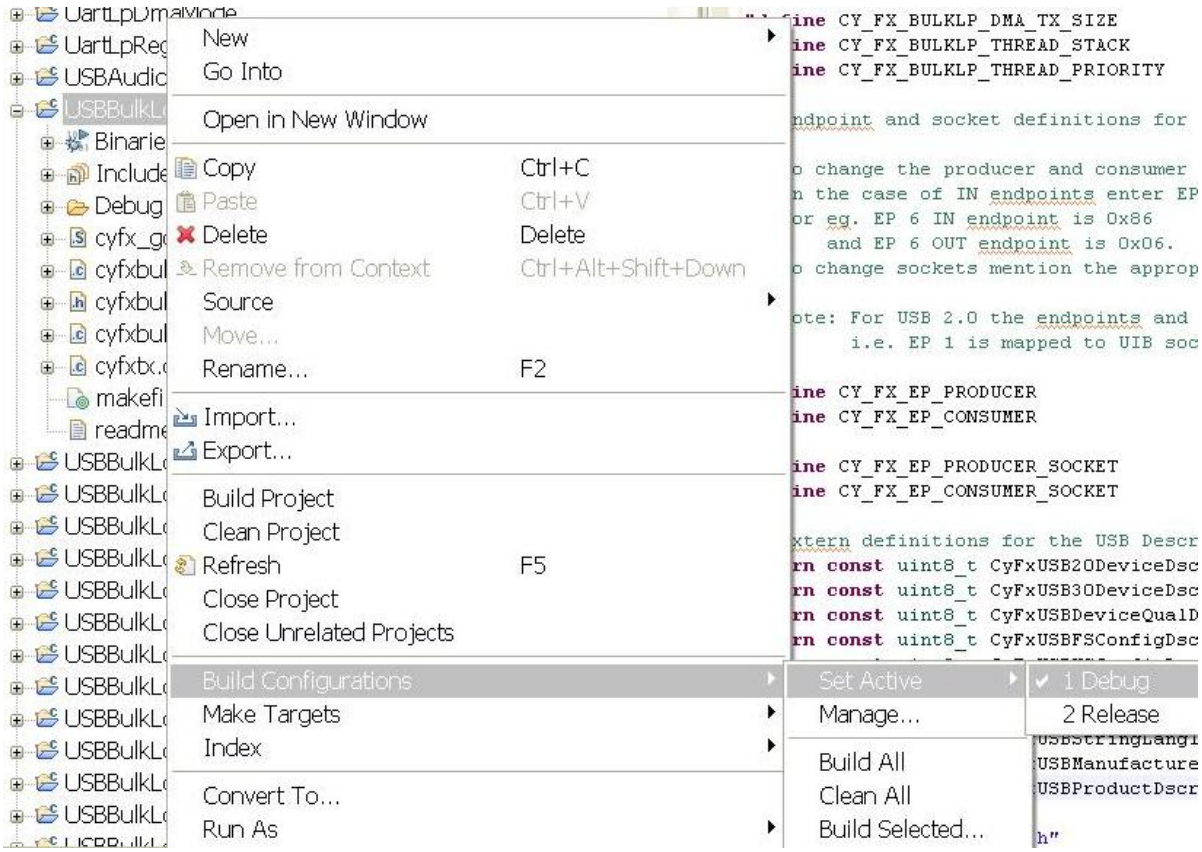
Figure 21. Endpoint Number and Associated Socket Modified from EP1 to EP2



4. Make sure you save the edited *cyfxbulklpauto.h* file (**File > Save**). The Eclipse build process does not automatically save freshly edited files. Build the project by ensuring that the USBBulkLoopAuto project is expanded in the Project Explorer pane, and then selecting **Project > Build Project**.

   You can also select **Build Configurations** before building the project. Right-click on the project name in the Project Explorer pane; then, select **Build Configurations > Set Active > Debug/Release** (see Figure 22). .

Figure 22. Selecting Build Configuration in Eclipse IDE

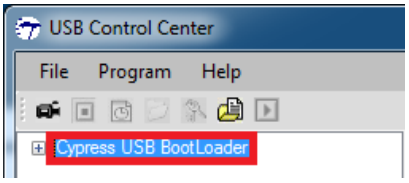


Note The firmware image size in **Release** mode is smaller compared to the **Debug** mode image. The Debug mode image contains additional debug symbols to allow the user to debug using JTAG.

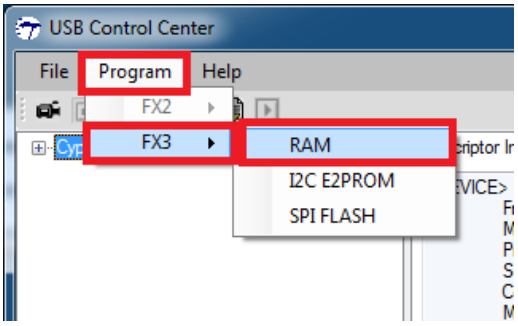
Loading USBBulkLoopAuto Firmware into FX3 RAM

An application called the Cypress Control Center loads the FX3 code into the FX3 DVK. The Control Center application installs as part of the FX3 SDK.

5. Open the Cypress Control Center application by going to your Start menu. Select **Start > All Programs > Cypress > Cypress USBSuite > Control Center**
6. Using the cable supplied in the DVK, connect one end to the DVK USB connector (center-right) and the other end to your development PC using either a SuperSpeed or Hi-Speed USB port. If you previously installed the DVK and driver software, the USB Control Center should show the “Cypress USB Bootloader” as a connected device. However, if you get a “no drive found” message, refer to Appendix B to install the DVK Windows driver.



7. To load the firmware in the FX3 RAM, select **Program > FX3 > RAM**.



8. Navigate to the firmware image file to load into FX3. This file is the *USBBulkLoopAuto.img* file that Eclipse generated as the last step of its build process. For the default workspace, this file is located at `C:\Users\antioch\Work\USBBulkLoopAuto\Debug`. Click **Open** after selecting the file or double-click the file name.

After the USB Control Center downloads the new FX3 code, Windows discovers a new USB device called Cypress USB BulkLoopExample. This is FX3 re-enumeration. FX3 initially appears to Windows as a bootloader, but when it loads the new code, it electrically disconnects from the USB and then reconnects as a new device defined by the downloaded code—the BULK looper. Note that the FX3 'Cypress USB Bootloader' disappears and a new USB peripheral appears in the Control Center menu as 'Cypress USB BulkloopExample'. FX3 has re-enumerated with the new Vendor ID/Product ID combination of 0x04B4 and 0x00F0.

Note When you are ready to load a new project, cycle power on the DVK board to re-install the bootloader. The Power Toggle switch on the board is a convenient way to cycle power without disconnecting the DVK board from the PC.

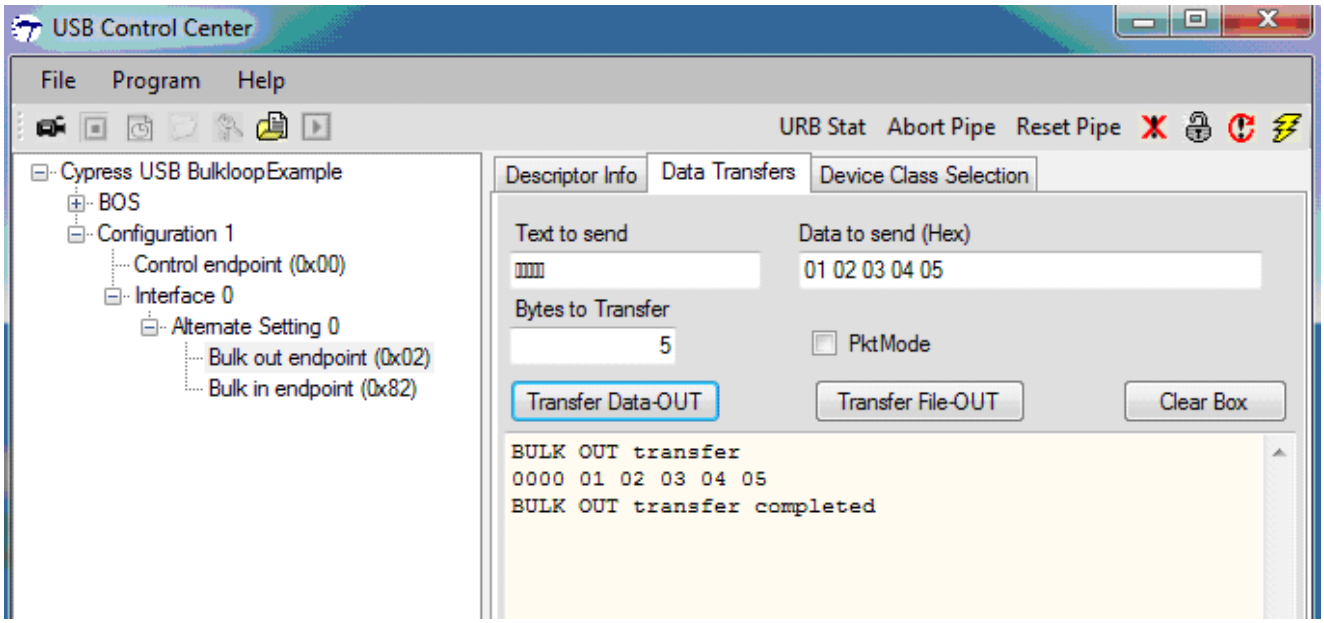
Performing BULK Transfers

1. Expand all the sub-fields under the 'Cypress USB Bulkloop Example' entry. If you modified and built the firmware as explained in steps 1 to 6, then you will see BULK EP2 OUT and BULK EP 2 IN appear inside Interface 0, Alternate Setting 0 (Figure 23).

   Note If you did not modify the firmware, you will see the default 'BULK out endpoint (0x01)' and 'BULK in endpoint (0x81)' in the Control Center.

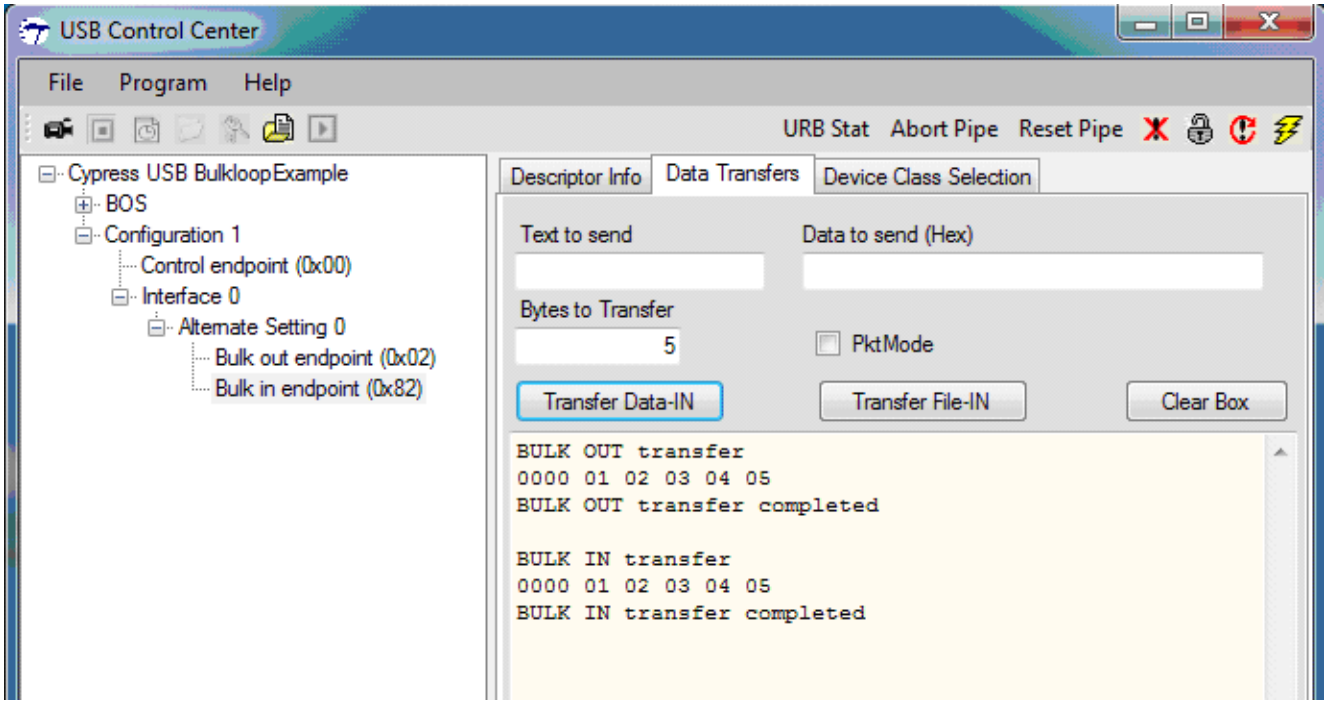
2. To perform a BULK OUT transfer, click the **Data Transfers** tab, then select the **BULK out endpoint (0x02)** in the left pane. Enter the number of bytes to transfer in the **Bytes to Transfer** field and the data to be transferred in the **Data to send (Hex)** field. The Control Center automatically inserts spaces between pairs of hex digits. Click on **Transfer Data-OUT**. Figure 23 shows the transfer of 5 bytes to the BULK OUT endpoint (0x02).

Figure 23. BULK OUT Transfer



3. The BULK Loop program sends back any data it receives over the OUT endpoint using the correspondingly-numbered IN endpoint. To perform the BULK IN transfer, select the **BULK in endpoint (0x82)** in the left pane. Enter the number of bytes to transfer in the **Bytes to Transfer** field. Click on **Transfer Data-IN.** Figure 24 shows the same 5 bytes being received back on IN endpoint (0x82).

Figure 24. BULK IN Transfer



Development Tools

Now that you have performed basic operations with the FX3 device, this section describes further details of the FX3 DVK and SDK.

Introduction to FX3 DVK

Cypress's FX3 DVK provides the hardware that you need to get started. The PCB provides the necessary clocks and voltages for the FX3 as well as configurable I/O voltages. The DVK has high-speed connectors for interfacing with external devices. The DVK also implements various boot modes. An I^2C EEPROM socket can be used to program and test booting from an I^2C EEPROM. An on-board SPI flash chip allows programming and booting over the SPI bus. Cypress example projects supplied with the SDK contain firmware to program these devices using the DVK board. If you do not already have a DVK, you can find one here.

Two important pieces of hardware, external to the FX3 DVK board, are the USB 3.0 host (typically the PC) and the external device connected to the GPIF II interface (such as an image sensor or FPGA). For details on using the FX3 DVK, refer to the DVK User Guide. Figure 25 shows a picture of the FX3 DVK board with the key areas called out.

Figure 25. FX3 DVK Board

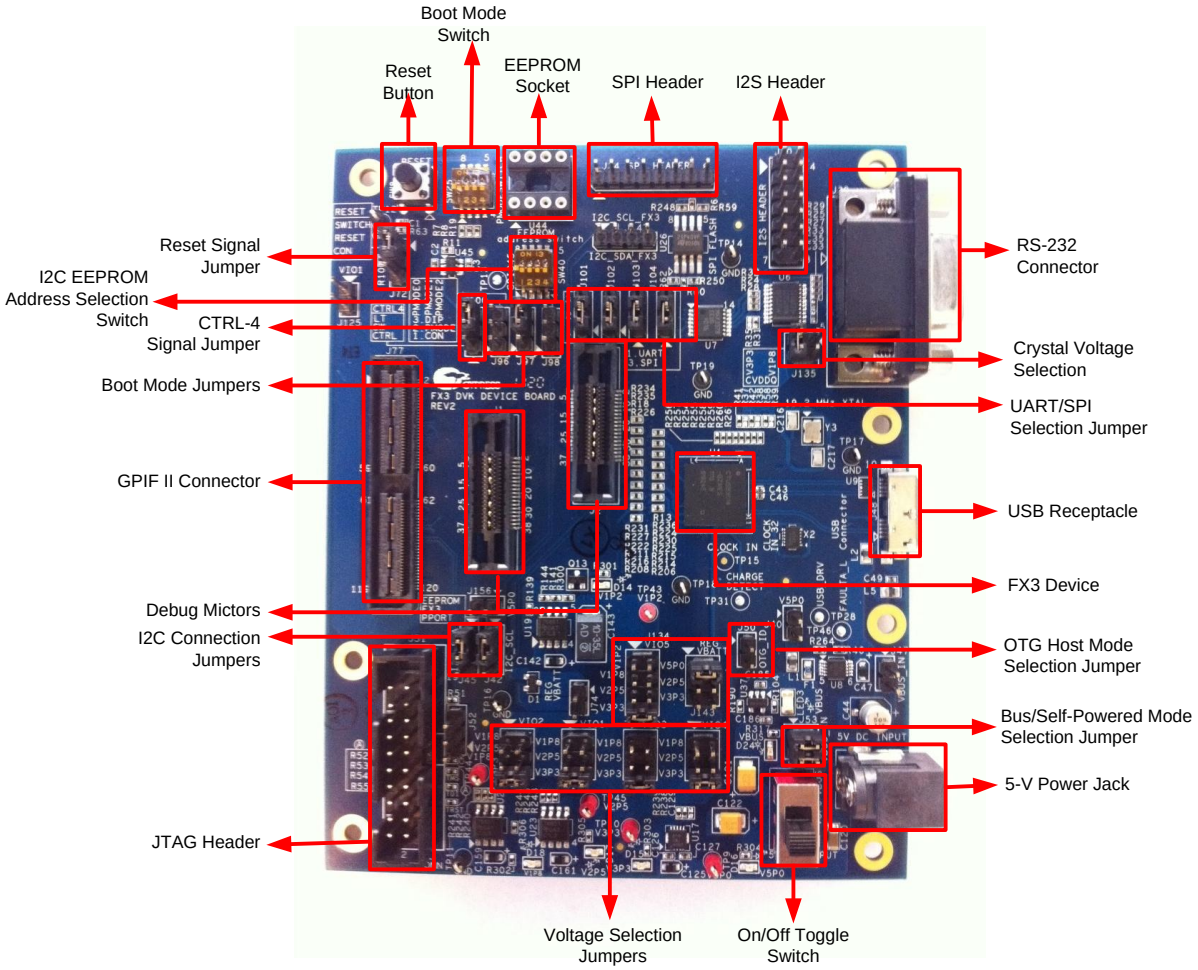


Table 2 shows the key jumpers and their default settings on the DVK board.

Table 2. FX3 DVK Board Jumpers

Jumper/Switch	Pins Shorted using Jumpers (Default Setting)	Function (Default Setting)
J101	1 and 2	GPIO_46=UART_RTS
J102	1 and 2	GPIO_47=UART_CTS
J103	1 and 2	GPIO_48=UART_TX
J104	1 and 2	GPIO_49=UART_RX
J136	3 and 4	VIO1(3.3V)
J144	3 and 4	VIO2(3.3V)
J145	3 and 4	VIO3(3.3V)
J146	3 and 4	VIO4(3.3V)
J134	4 and 5	VIO5(3.3V)
J135	2 and 3	CVDDQ(3.3V)
J143	1 and 6	VBATT(2.5V)
J96 & SW25	2 and 3	a).PMODE0 Pin state(ON/OFF) selection using SW25. b). SW25.1=OFF
J97 & SW25	2 and 3	a). PMODE0 Pin state(ON/OFF) selection using SW25. b). SW25.2=OFF
J98	1 and 2	PMODE2 Pin Floating
J72	1 and 2	RESET
J42	Not Installed	GPIO_58=I2C_SCL
J45	Not Installed	GPIO_59=I2C_SDA
J100	1 and 2	GPIO_21=CTL4

Introduction to FX3 SDK

Cypress provides a complete software and firmware stack for FX3 to integrate USB applications in the embedded system environment. The software development kit (SDK) comes with application examples that accelerate application development. The SDK is available for download from the Cypress website. Figure 26 shows the directory structure created when the SDK is installed.

Figure 26. Directory Structure

Firmware Stack and APIs

Cypress provides a powerful API library to make developing firmware easier even for complex designs. Here are some of the advantages of the FX3 SDK:

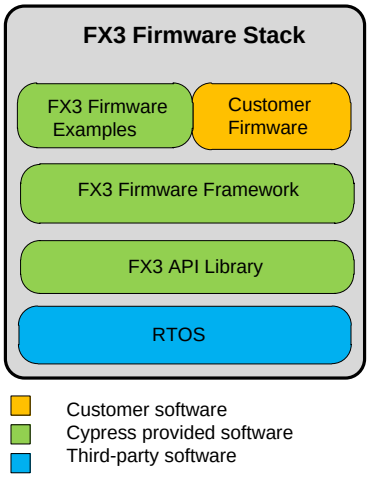
- RTOS

  The included ThreadX Real Time OS (RTOS) simplifies the firmware development process. With the RTOS, multiple threads can easily be created to simplify the firmware flow.

- Modular approach

  An API-based approach means that the developer does not require knowledge of FX3's registers and bits, but can focus on the firmware logic and flow. This approach is easy to use, debug, modify, and support.

Figure 27. Structure of Firmware SDK



The APIs enable:

- Programming each individual block of the FX3 device, the GPIF II, USB, and serial interfaces
- Programming the DMA engine and setting up of data flow between these blocks
- ThreadX OS calls as required by the application
- Debug capability
- USB host mode of operation
- Power management features

Framework API

The firmware (or application) framework contains startup and initialization codes. Also included are the individual drivers for the USB, GPIF II, and serial interface blocks. The framework performs the following functions:

- Defines the program entry point
- Performs the stack setup
- Performs kernel initialization
- Provides placeholders for application thread startup code

FX3 Firmware Examples

The SDK includes many firmware examples as seen in the Eclipse Installation above. This section details several of the more common example firmware projects. A complete list of currently available firmware examples is in the SDK release notes. The examples work at both USB 2.0 and USB 3.0 speeds.

Note Example firmware is provided in the form of individual Eclipse projects. This allows your complete example firmware inventory to be seen at a glance in the Eclipse Project Explorer.

USB BULK Data Loopback Examples

These illustrate a simple loopback mechanism between USB BULK endpoints. All standard setup requests from the USB host PC are handled by the FX3 application example. The examples implement the loop back using DMA AUTO or MANUAL channels.

The DMA multichannel examples use three endpoints for the loopback.

USB ISOCHRONOUS Data Loopback Examples

These examples illustrate a loopback mechanism between USB ISOCHRONOUS endpoints. This is similar to the BULK loopback examples, except that the endpoints used here are isochronous instead of BULK.

Slave FIFO Application Examples

The Slave FIFO application example demonstrates data transfer between the USB host and an external FIFO controller. The example consists of two unidirectional data pipes between the USB host and the external master. The GPIF II interface can be configured for either synchronous or asynchronous Slave FIFO transfers, using a 16-bit or 32-bit bus.

Serial Interface Examples

These examples demonstrate data accesses to the GPIOs, I^2C, SPI, and UART.

USB BULK/ISOCHRONOUS Data Source Sink Examples

These examples illustrate data source and data sink mechanism with two USB BULK/ISOCHRONOUS endpoints.

Flash Programmer Example

This example illustrates the programming of I^2C EEPROMS and SPI flash devices from USB. The read or write operations are done using pre-defined vendor commands. The utility can be used to flash the boot images to these devices.

Mass Storage Class Example

This example illustrates the implementation of a USB mass storage class (BULK Only Transport) device using a small section of the FX3 device RAM as the storage device. The example shows how mass storage commands can be parsed and handled in the FX3 firmware.

USB Audio Class Example

This example creates a USB Audio Class compliant microphone device, which streams PCM audio data stored on the SPI flash memory to the USB host. This example works only at USB 2.0 speeds.

Two Stage Booter Example

A simple set of APIs have been provided as a separate library to implement two-stage booting. This example demonstrates the use of these APIs. Configuration files that can also be used for Real View Tool chain are provided.

USB Host and OTG Examples

These examples demonstrate the host mode and OTG mode operation of the FX3 USB port.

FX3 Firmware Development Tools

Eclipse IDE

The Eclipse IDE for C/C++ Developer is provided as part of the FX3 SDK. This IDE comprises the base Eclipse platform and the CPP feature. The IDE includes plug-ins required for development.

- GNU ARM C/C++ Development support
- Zylin Embedded CDT

  This is a generic plug-in for the Eclipe IDE, which enables debugging of the FX3 firmware using the GNU debugger
- Java(TM) Platform, Standard Edition Runtime Environment Version 7 (JRE)

GNU Tool Chain

The GNU tool chain provided as part of the FX3 SDK comprises the following:

- GCC compiler (gcc)
- GNU Linker (ld)
- GNU Assembler (as)
- GNU Debugger (gdb)

The Eclipse IDE invokes these executables.

GPIF II Designer

The GPIF II Interface Design Tool is a Windows application that FX3 customers receive as part of the FX3 SDK. The tool provides a graphical user interface to specify the necessary interface for the target device. The user designs an interface using state machine entry, and the tool translates this into a C header file to be included in the Eclipse project.

JTAG Debugger

The Segger J-Link probe is the preferred JTAG probe for the FX3 SDK. This probe, along with the Segger J-Link ARM GDB Server, is used for debug. The Eclipse IDE connects to the J-link GDB server to debug your firmware. To get Eclipse working with the GDB server, you need to create a debug configuration for the J-link. Details are in Chapter 12 of the FX3 Programmer's Manual.

To debug, you may also use other JTAG probes such as the Olimex ARM-USB-OCD probe. In this case, the OpenOCD tools can be used for the GDB connection. Details on how to do this are in Chapter 12 of the FX3 Programmer's Manual.

Windows Software Overview

Cypress delivers device driver and interface APIs with libraries to develop FX3 USB applications in Windows.

Windows USB Device Driver

cyusb3.sys is a USB device driver for 32-bit Windows XP, 32/64-bit Windows Vista, 32/64-bit Windows 7, and 32/64-bit Windows 8. The driver is capable of communicating with any device that complies with the USB 2.0 and USB 3.0 specifications. The driver is general-purpose and understands primitive USB commands. However, it does not implement USB device-class protocols. For example, the driver does not directly interface a USB mass storage device to the Windows file system. The application firmware must implement this logic.

The generic Cypress driver is signed for a VID of 0x04B4 and PIDS of 0x00F0, 0x00F1, and 0x00F3. If you want to use your own VID/PID pair, you must get the driver signed by Microsoft WHQL. During the development phase, you can use an unsigned driver by disabling driver signature enforcement on your computer. The driver is ideal for communicating with a vendor-specific device from a custom USB application. Alternatively, the driver might be used to send low-level USB requests to any USB device for experimental or diagnostic applications. To use the driver to communicate with a device, Windows must match the device to the driver. The class library, CyAPI.lib and Cyusb.dll, provides a high-level programming interface to the driver.

Features

- Windows Driver Foundation (WDF)-compliant
- Compatible with any USB 2.0-compliant device
- Compatible with Cypress USB 3.0-compliant device
- Supports basic USB 3.0 features
- Supports Windows plug-and-play and power management
- Supports USB remote wake-up
- Supports CONTROL, BULK, INTERRUPT, and ISOCHRONOUS endpoints
- Supports multiple USB devices connected simultaneously
- Supports customizable driver GUID without rebuilding the driver
- Supports high-bandwidth data transfers passing multiple packets per frame

You are not required to use the Cypress-supplied driver. FX3 can be programmed to implement standard USB class devices. When implementing such a device, the USB class driver should be used instead. For example, if an FX3 is implementing the USB Video Class (UVC) it does not need the Cypress generic driver and instead uses the OS-provided UVC driver. This makes Windows, Linux, and Mac support simple because certain standard drivers are already provided by the OS.

Application Interface

CyAPI.lib

CyAPI.lib provides a simple and powerful C++ programming interface to the USB devices. A C++ class library provides a high-level programming interface to the cyusb3.sys device driver. The library can communicate only with USB devices served by this driver.

For further details on CyAPI.lib, refer to the Programmer's Reference C++ Library. This is available in the USB Suite Help documentation as part of the SDK.

CyUSB.dll

CyUSB.dll is a managed Microsoft .NET class library that provides a high-level programming interface to the USB devices. Rather than communicate with the USB device drivers directly via low-level Win32 API calls, applications can access USB devices through library methods and properties. Because CyUSB.dll is a managed .NET library, its classes and methods can be accessed from any of the Microsoft Visual Stuido.NET managed languages, such as Visual Basic.NET, C#, Visual J#, and managed C++. To use the library, add a reference to CyUSB.dll to your project's **References** folder. Then, any source file that accesses the CyUSB namespace must include a line to add the namespace in the appropriate syntax.

For further details on CyUSB.dll, refer to the Programmer's Reference C# Library. This is distributed in the USB Suite Help documentation as part of the SDK.

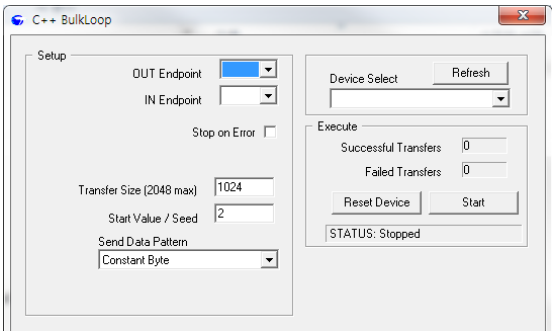
Refer to the Cypress USBSuite Application Development - Quick Start Guide for details on using the CyAPI.lib and CyUSB.dll.

Window Software Examples

BULKLoop example

The application BULKLoop is used to test the loopback of data transfer through BULK endpoints.

Figure 28. Cypress C++ BULKloop Application



Streamer Example

The application streamer is used to test the data transfer on the BULK/ISOCHRONOUS endpoints.

Figure 29. Cypress C++ Streamer Application

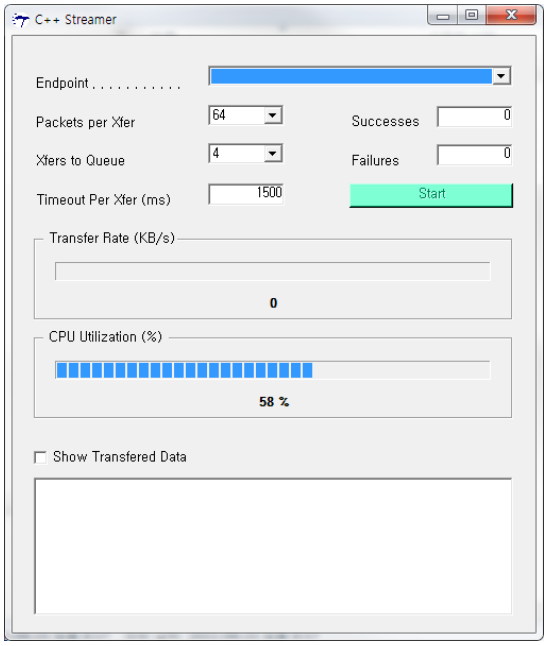
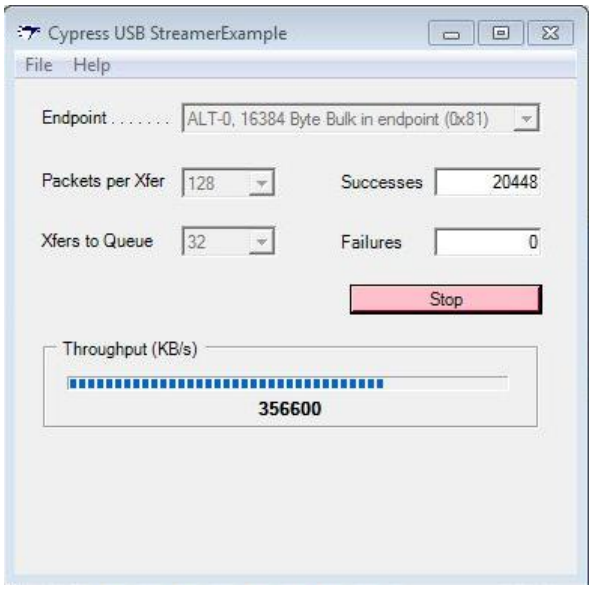


Figure 30. Cypress C# Streamer Application (running traffic in a Slave FIFO application)



On installing the FX3 SDK, both C++ and C# implementations of the BULKloop and Streamer applications are available in the 'applications' directory.

FX3 SDK and Software for Linux

EZ-USB FX3 SDK for Linux

The FX3 SDK supports firmware development with Eclipse IDE and debug using the J-Link JTAG debugger probe on a Linux platform.

The EZ-USB FX3 SDK for Linux contains the following:

- The FX3 firmware library and examples
- Sourcery ARM GNU toolchain
- Eclipse IDE for 32-bit Linux OS installations
- Eclipse IDE for 64-bit Linux OS installations
- The CyUSB Suite for Linux software

The installation procedure involves extraction of these archives and setting of environment variables. Refer to *FX3_SDK_Linux_Support.pdf* available in the following extracted folder of the FX3 SDK for Linux installation: `fx3_sdk_v1.3_linux\FX3_Firmware\cyfx3sdk\doc .`

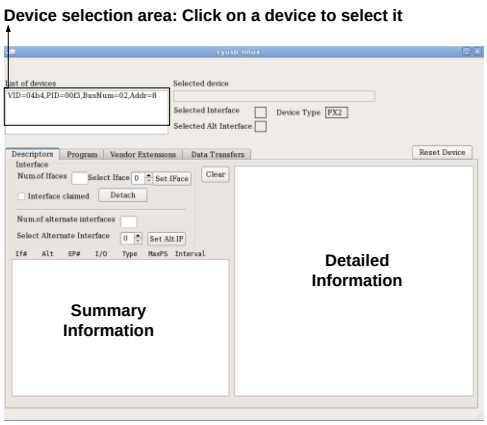
CyUSB Suite for Linux

The CyUSB Suite for Linux software enables you to download firmware images to FX3 devices, and test the various interfaces on the device. Refer to *cyusb_linux_user_guide.pdf* available in the following extracted folder of the FX3 SDK for Linux installation: `fx3_sdk_v1.3_linux\cyusb_linux_1.0.4\cyusb_linux_1.0.4\docs`. This document describes how to install the CyUSB Suite for Linux software and how to use the software to download firmware to FX3, to 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. The CyUSB Suite for Linux software allows you to do the following:

- View the device, configuration, interface, alternate-interface, and endpoint descriptors of attached devices.
- Select a specific interface and alternative interface for communication.
- Program the device (download firmware) to the FX3 device, downloading into RAM, I^2C-based EEPROM, or SPI-based flash.
- Test your own commands (Vendor Extensions) after downloading specific firmware that implements your commands.
- Test the BULK OUT endpoints by sending either constant or random or incrementing data patterns and also testing the BULK IN endpoints by looping back data after sending the OUT data.
- Test the ISOCHRONOUS OUT and IN endpoints and measure the data transfer rates.

Figure 31. CyUSB Suite for Linux



Useful Debug Tools

USB 3.0 Protocol Analyzer

A USB 3.0 protocol analyzer is a useful debugging tool. It analyzes the traffic on the USB bus between the FX3 and the host. Software tools included with each analyzer then decode the data into USB transfer packets. By analyzing this data, issues can be easily identified, and performance can be maximized. Several USB 3.0 analyzers are available in the market today. Cypress does not recommend any specific analyzer but here are some options:

- Standalone USB 3.0 Protocol Analyzer
  - Ellisys USB Explorer 280
  - LeCory USB Voyager M3i
  - Beagle USB 5000 SuperSpeed Protocol Analyzer
- PC Software USB 3.0 Protocol Analyzer
  - SourceQuest SourceUSB
  - SysNucleus USBTrace

Logic Analyzer

Logic analyzers allow simple analysis of digital signals. They can be used to look at various signals between FX3 and other peripherals. There are two types of logic analyzers in the market:

- Standalone Type Logic Analyzer
  - Agilent 16800 Series Portable Logic Analyzer
- PC-Based Type Logic Analyzer
  - USBee Logic Analyzer
  - ZeroPlus Logic Analyzer

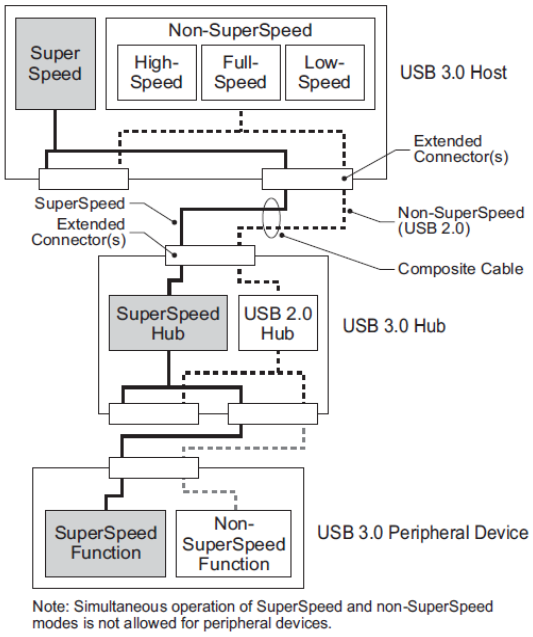
An important factor to consider is that the signal frequency range of the analyzer should be higher than the signals to be analyzed.

Appendix A

USB 3.0 Overview

USB 3.0 enables an increased data rate of 5 Gbps, reduces power consumption, and is backward-compatible with USB 2.0. The USB Specification, published by USB-IF, can be found here. Figure 32 shows the USB 3.0 architecture.

Figure 32. USB 3.0 Dual Bus Architecture



Courtesy: http://usb.org

The physical interface of USB 3.0 consists of two differential pairs and a ground for SuperSpeed transfers in addition to the USB 2.0 connections. This allows USB 3.0 to ensure backward compatibility with USB 2.0.

Electrical Interface

The USB 3.0 pinout is different from that of USB 2.0. In addition to the VBUS, D–, D+, and GND pins required for USB 2.0, USB 3.0 has five additional pins – two differential pairs and one ground (GND_DRAIN). The two differential pairs are for SuperSpeed data transfer, supporting dual simplex SuperSpeed signaling. The GND_DRAIN pin is for drain wire termination, management of signal integrity, and EMI performance. Table 3 shows a description of the nine pins.

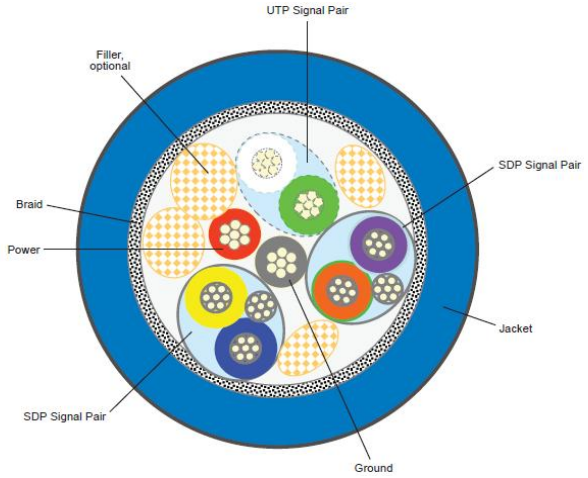
Table 3. USB 3.0 Pin Description

Pins Name	Description
VBUS	Power
D-	USB 2.0 differential pair
D+	
GND	Ground for power return
SSRX-	SuperSpeed receiver differential pair
SSRX+	
SSTX-	SuperSpeed transmit differential pair
SSTX+	
GND_DRAIN	Ground for signal return

Cables and Connectors

USB 3.0 has four additional data lines (SSRX+, SSRX-, SSTX+, SSTX-) for data transfer and one additional ground line for drain wire termination, signal integrity management, and EMI performance. Figure 33 shows the architecture of a USB 3.0 cable. Table 4 shows the description of these lines.

Figure 33. USB 3.0 Cable Architecture



Courtesy: http://usb.org

Table 4. USB 3.0 Cable Description

Name	Description	Color
PWR	VBUS	Red
UTP_D-	USB 2.0 D-	White
UTP_D+	USB 2.0 D+	Green
Ground	Ground for power drain	Black
SDP1-	Shielded differential pair 1	Blue
SDP1+		Yellow
SDP1_Drain	Drain line for SDP1	
SDP2-	Shielded differential pair 2	Purple
SDP2+		Orange
SDP2_Drain	Drain line for SDP2	

The USB 3.0 Specification also defines the following connectors:

- **USB 3.0 Standard-A Plug and Receptacle**

  The USB 3.0 Standard-A connector is defined in the SuperSpeed standard as the host connector. It is based on the design of the USB 2.0 Standard-A connector but has the additional SuperSpeed signals. A USB 3.0 Standard-A receptacle accepts either a USB 3.0 Standard-A plug or a USB 2.0 Standard-A plug. USB 3.0 capable Standard-A connectors use a unique color for easy identification. Figure 33 shows the color coding recommendation.

- **USB 3.0 Standard-B Plug and Receptacle**

  The USB 3.0 Standard-B connector is defined for large, stationary peripherals, such as external hard drives and printers. The USB 3.0 Standard-B receptacle accepts either a USB 3.0 Standard-B plug or a USB 2.0 Standard-B plug. You cannot insert a USB 3.0 Standard-B plug into a USB 2.0 Standard-B receptacle.

- **USB 3.0 Powered-B Plug and Receptacle**

  The USB 3.0 Powered-B connector enables a USB 3.0 device to provide power to a USB adapter without an external power supply. It is identical to the USB 3.0 Standard-B connector in form factor but has two more pins: one for power (DPWR) and one for ground (DGND).

- **USB 3.0 Micro-B Plug and Receptacle**

  The USB 3.0 Micro-B connector is defined for small handheld devices.

- **USB 3.0 Micro-AB and USB 3.0 Micro-A Connectors**

  The USB 3.0 Micro-AB receptacle is similar to the USB 3.0 Micro-B receptacle, except keying is different. It accepts a USB 3.0 Micro-A plug, a USB 3.0 Micro-B plug, a USB 2.0 Micro-A plug, or a USB 2.0 Micro-B plug. The USB 3.0 Micro-AB receptacle is allowed only on OTG products, which may function as either a host or a device. All other uses of the USB 3.0 Micro-AB receptacle are prohibited.

  The USB 3.0 Micro-A plug is similar to the USB 3.0 Micro-B plug, except for different keying and ID pin connections. The USB 3.0 Micro-A plug, the USB 3.0 Micro-AB receptacle, and the USB 3.0 Micro-B receptacle and plug, belong to the USB 3.0 Micro connector family. Their interfaces differ only in keying. Similar to the USB 2.0 Micro-A plug, the USB 3.0 Micro-A plug is defined only for OTG applications.

USB 3.0 versus 2.0

USB 3.0 has a dual-bus architecture that supports USB 2.0 and 3.0. The following table shows the main differences between USB 3.0 and USB 2.0.

Table 5. Differences Between USB 3.0 and USB 2.0

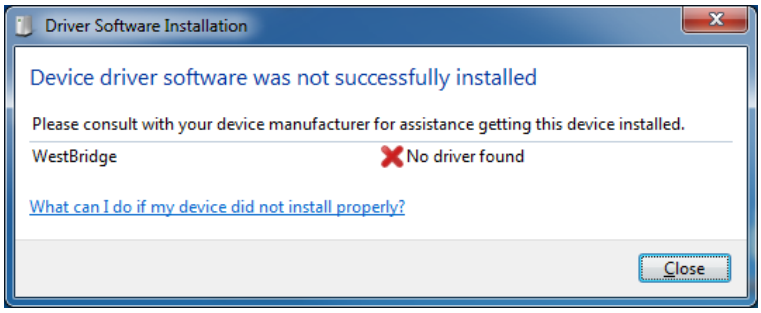
Feature	USB 2.0	USB 3.0
Data rate	480 Mbits/s (High Speed) 12 Mbits/s (Full Speed) 1.5 Mbits/s (Low Speed)	5.0 Gbits/s (SuperSpeed) 480 Mbits/s (High Speed) 12 Mbits/s (Full Speed) 1.5 Mbits/s (Low Speed)
Data interface	Half-duplex Two-wire differential signaling	Dual-simplex Four-wire differential signaling
Cable signal count	Four signals: - Two for USB 2.0 data (D, D–) - Two for VBUS and GND	Nine signals: - Four for SuperSpeed data - Two for USB 2.0 data (D, D–) - Three for VBUS and GND
Bus transaction protocol	Host directed Polled traffic flow Packets broadcast to all downstream devices No multiplexing of data streams	Host directed Asynchronous notifications Packets routed only to target device Multiple data streams possible for BULK transfers
Power management	Two modes - Active - Suspend	Four modes - Active (U0) - Idle, Fast (U1) - Idle, Slow (U2) - Suspend, Slow (U3)
Bus power	Low-power device : 100 mA High-power device : 500 mA	Low-power device : 150 mA High-power device : 900 mA
Port state	Port hardware detects connect events. System software uses port commands to transition the port into an enabled state.	Port hardware detects connect events and brings the port into operational state ready for SuperSpeed data communication
Maximum cable length	5 meters	Based on electrical specification. In practice 3 meters for 26 AWG copper
Data transfer types	Four data transfer types: CONTROL, BULK, INTERRUPT, and ISOCHRONOUS.	USB 2.0 types with SuperSpeed capabilities. BULK has streams capability.

Appendix B

Windows Install of FX3 DVK Driver

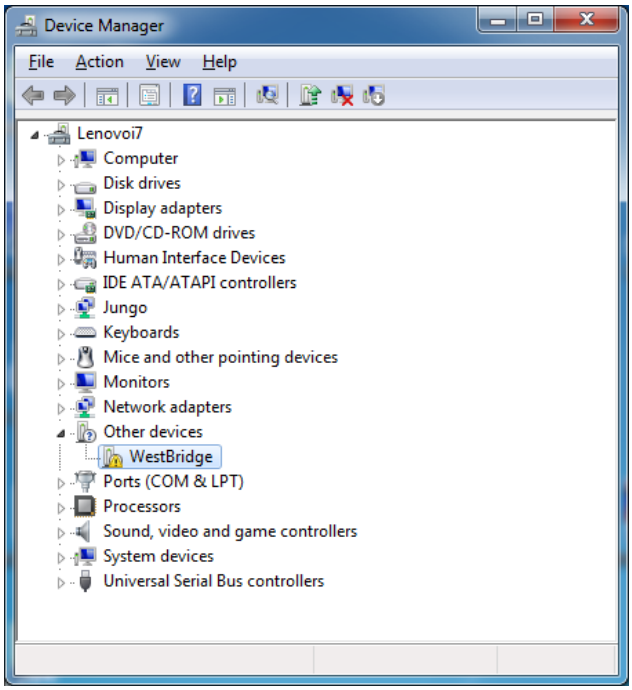
If you have not already installed the FX3 DVK kit on a Windows computer, the first time you connect the DVK to the computer you will see the following message.

Figure 34. Windows Message if Driver is Not Installed



Close the message box and navigate to Windows Device Manager. To do this, click the Windows **Start** button, right-click on **Computer** in the right-hand column, and select **Properties** to bring up System Information. Then, click **Device Manager** at the top of the left column.

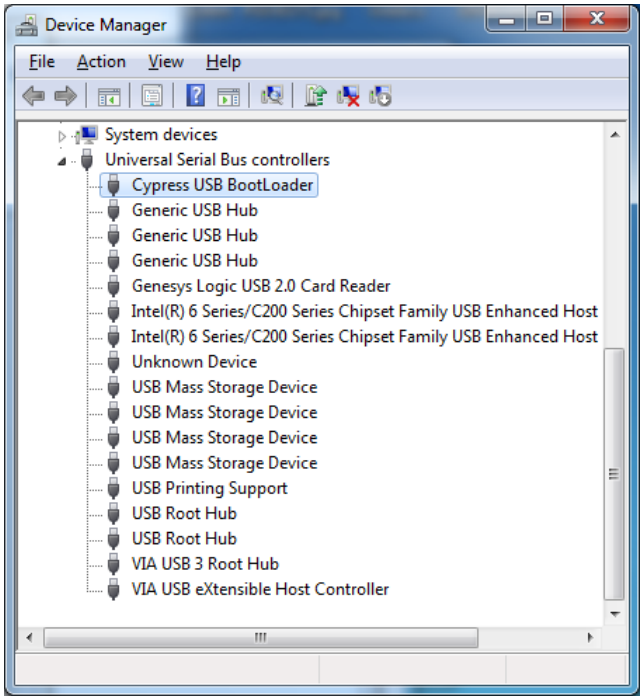
Figure 35. Windows Device Manager



Right-click on **WestBridge** and select **Browse my computer for driver software**. On a 64-bit Windows 7 machine, the driver is located at `C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.2\driver\win7\x64`. Your SDK version number may be higher than 1.2.

The Device Manager window should remove the WestBridge entry and identify the SDK board as the Cypress USB Bootloader (top entry).

Figure 36. Cypress USB BootLoader in Device Manager after Driver Installation



Document History

Document Title: Getting Started with EZ-USB® FX3™

Document Number: 001-75705

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3427934	ATAM	02/23/2012	New application note.
*A	3715875	ATAM/OSG	08/17/2012	Merged with AN75432 USB 3.0 EZ-USB® FX3™ Orientation Update application note for FX3 SDK 1.2
*B	3981810	OSG	04/25/2013	Restructured application note Added sections: What is FX3, Application Development with FX3, Design Resources Added FX3 Block Diagram Aligned with SDK 1.2.3
*C	4183163	RSKV	11/05/2013	Added FX3 Terminology section. Added the location of the FX3 SDK Linux support document and CyUSB Suite for the Linux user guide.

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 4 | PSoC 5LP

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