Cypress EZ-USB™FX3™ SDK

Quick Start Guide

Version 1.2.1

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
http://www.cypress.com

Copyrights

The information in this document is subject to change without notice and should not be construed as a commitment by Cypress. While reasonable precautions have been taken, Cypress assumes no responsibility for any errors that may appear in this document. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Cypress. Made in the U.S.A.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

License Agreement

Please read the license agreement during installation.

Contents

1 FX3 SDK

1.1 FX3 Overview

Cypress EZ-USB FX3 is the next generation USB3.0 peripheral controller which is highly integrated, flexible and enables system designers to add USB 3.0 capability to any system.

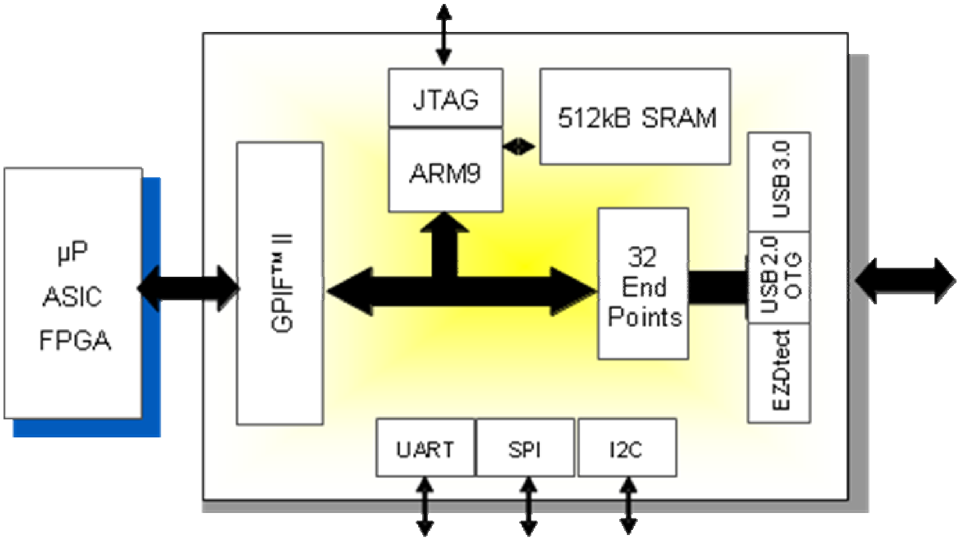


Figure 1: EZ USB FX3 System Diagram

It has a fully configurable, parallel, General Programmable Interface called GPIF II, which can connect to any processor, ASIC, DSP, or FPGA. It has an integrated USB Phy and controller along with a 32-bit microcontroller (ARM926EJ-S) for powerful data processing and for building custom applications. There is 512 KB of on-chip SRAM for code and data. It has an inter-port DMA architecture which enables data transfers of **> 400 MBps**.

The FX3 is fully compliant to the USB 3.0 v1.0 and USB 2.0 specifications. An integrated USB 2.0 OTG controller enables applications that need dual role usage scenarios. The device is also compliant with the USB Battery Charging Specification v1.1.

There are also serial peripherals such as UART, SPI, I2C, and I2S for communicating to on board peripherals (for example the I2C interface is typically connected to an EEPROM).

The General Programmable Interface GPIF II is an enhanced version of the GPIF in FX2LP, Cypress's flagship USB2.0 product. The GPIF II Controller drives the behavior and timing of the GPIF II Interface based on one or more user configurable state machines. It provides an easy and glueless interface to popular interfaces such as asynchronous SRAM, synchronous SRAM, Address Data Multiplexed interface, parallel ATA, and can be programmed to implement most other parallel communication protocols.

1.2 FX3 SDK Overview

Cypress delivers the complete software and firmware stack for FX3 in order to easily integrate all USB applications in the embedded system environment. The software development kit comes with application examples which help accelerate application development.

The components of the FX3 software development kit are shown in Figure 2.

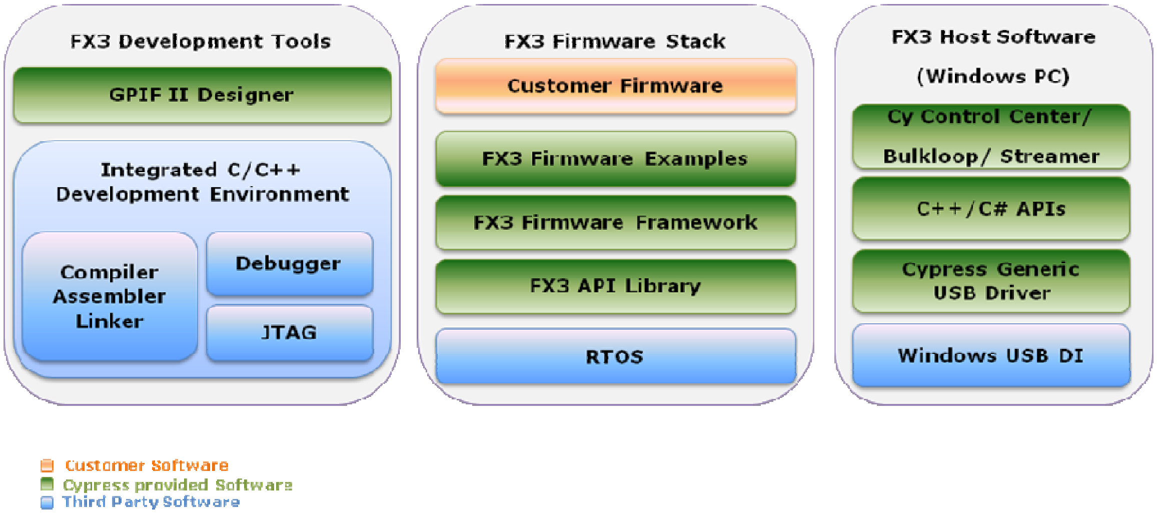


Figure 2: FX3 Software Solution (SDK)

On the device side, there is the FX3 firmware stack that comprises of a full fledged API library and a comprehensive firmware framework. A well documented library of APIs enable users to access the hardware functions of FX3. The SDK also includes application implementation examples in source form articulating different usage models. Users can also develop their own application using this firmware framework and modifying it as applicable. There is complete flexibility for users to develop custom applications using the firmware framework. An RTOS library is integrated with the firmware stack, allowing users to implement complex applications requiring multiple threads of firmware execution.

A set of development tools is provided with the SDK, which includes the GPIF II Designer and third party toolchain and IDE.

The firmware development environment will help the customer develop, build and debug firmware applications for FX3. The third party ARM® software development tool provides an integrated development environment with compiler, linker, assembler and JTAG debugger. A recent build of the free GNU tool-chain for ARM processors and an Eclipse based IDE is provided by Cypress.

The USB host side (Microsoft Windows) stack for the FX3 contains:

- Cypress generic USB 3.0 driver (WDF) on Windows 7 (32/64 bit) and Windows Vista (32/64 bit) and Windows XP (32 bit)

- Convenience APIs that expose generic USB driver APIs through C++ and C# interfaces

- USB control center, a Windows utility that provides interfaces to interact with the device at low levels such as configuring end points, data transfers etc

- Bulkloop application, a user application to perform data loopback on the Bulk endpoints.

- Streamer application, a user application to perform data streaming over Isochronous or Bulk endpoints.

1.3 FX3 DVK Board Overview

Cypress provides a FX3 DVK board that can be used for firmware and system development using the FX3 device. The DVK board provides the means to connect an external processor or device to the GPIF interface and to connect slave devices to the I2C, SPI and I2S interfaces.

The FX3 DVK Board can be used to execute the firmware examples in the SDK and also for firmware development based on the SDK. The hardware connection and programming sections below assume the use of the FX3 DVK board.

Please refer to the FX3 DVK User Guide for additional information regarding the DVK board.

2 SDK Installation

2.1 Components of the FX3 SDK

The FX3 SDK is provided in the form of multiple MSIs providing the following functionalities:

1. FX3 Firmware MSI – This contains the FX3 Firmware libraries, Header files, Example code and firmware conversion utility
2. ARM GCC MSI – This contains the ARM GNU Toolchain
3. Eclipse MSI – The eclipse IDE with the required plug-ins and the Java runtime.
4. USB Suite MSI – The windows host driver, C++ & C# API libraries, and the Control center, Bulkloop & Streamer. There are 2 MSIs – one for 32 bit platforms (Windows XP, Vista and Windows 7) and one for one for 64 bit platforms (Windows Vista and Windows 7)

Run each of the MSIs and follow the instructions to complete the installation. The default installation path for the ARM GCC and Eclipse IDE is C:\Program Files\Cypress . The default installation path for the FX3 Firmware and USB Suite is C:\Cypress .

The FX3 SDK also includes the following documentation:

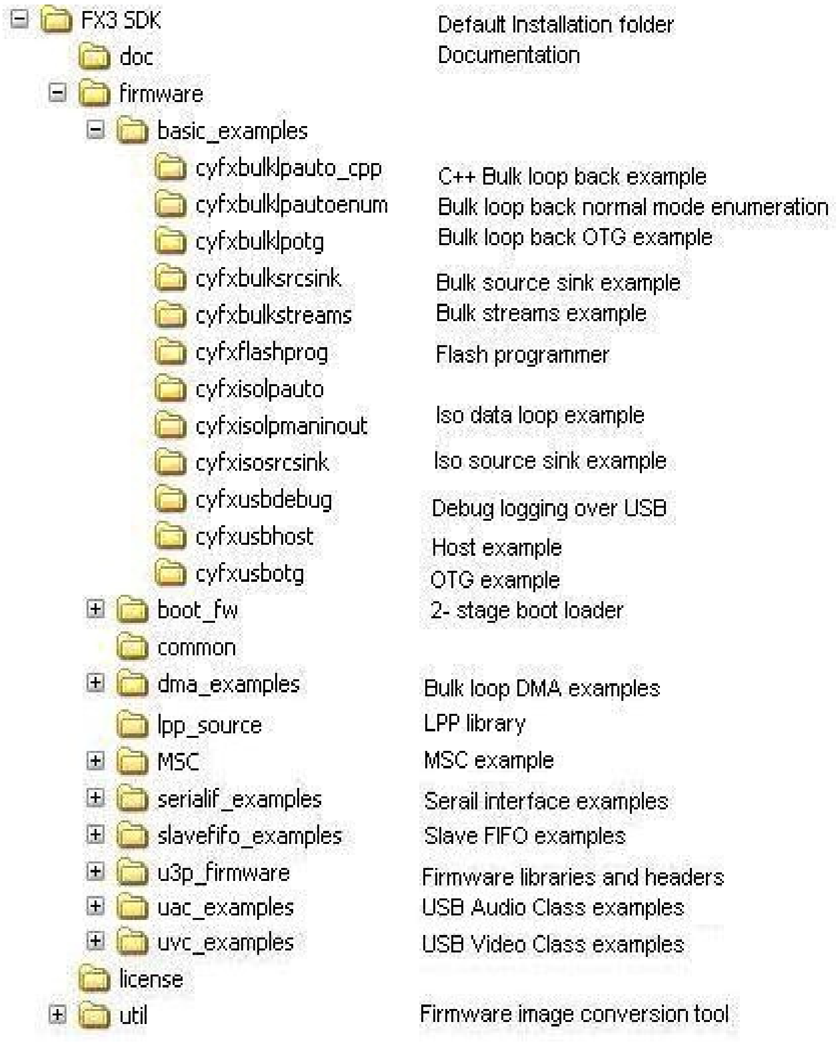
1. FX3 Programmer's Manual
2. FX3 API guide
3. FX3 Release Notes

These documents are located in

<Installation folder>\FX3 SDK\doc\

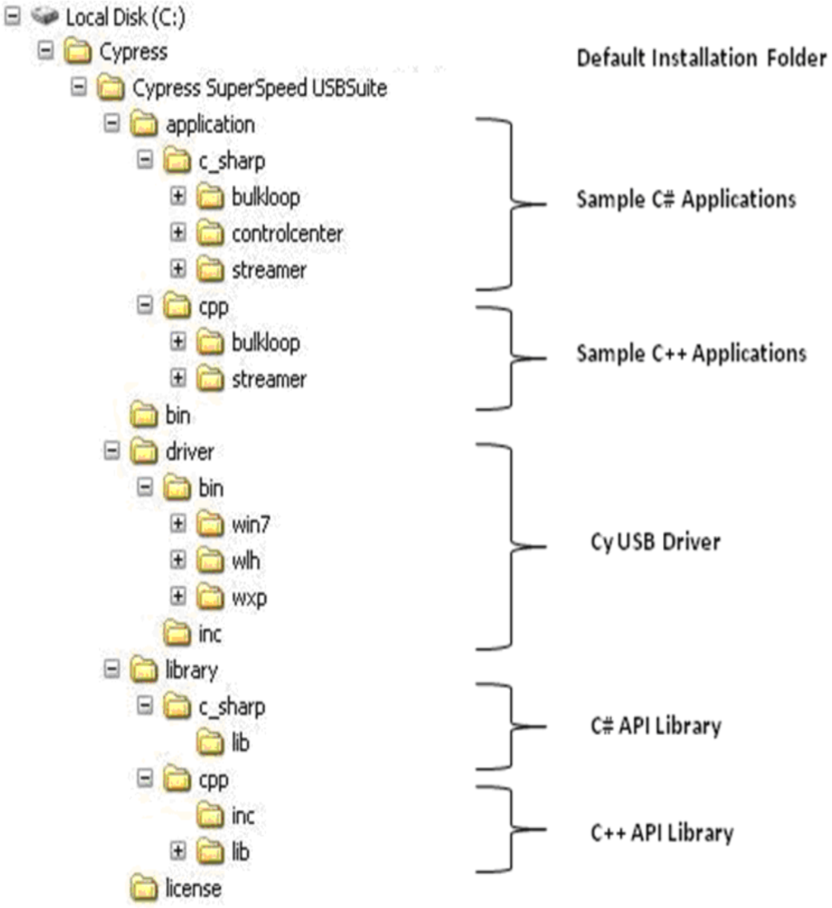
2.2 Installed Directory Structure

The installed FX3 SDK directory structure contains the following:



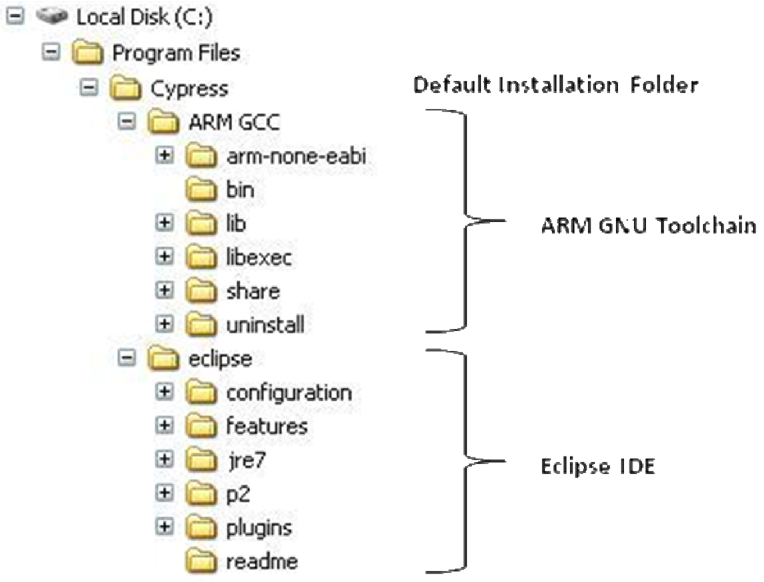
⊟ FX3 SDK	Default Installation folder
doc	Documentation
⊟ firmware	
⊟ basic_examples	
cyfxbulklpauto_cpp	C++ Bulk loop back example
cyfxbulklpautoenum	Bulk loop back normal mode enumeration
cyfxbulklpotg	Bulk loop back OTG example
cyfxbulksrcsink	Bulk source sink example
cyfxbulkstreams	Bulk streams example
cyfxflashprog	Flash programmer
cyfxisolpauto	
cyfxisolpmaninout	Iso data loop example
cyfxisosrcsink	Iso source sink example
cyfxusbdebug	Debug logging over USB
cyfxusbhost	Host example
cyfxusbotg	OTG example
⊞ boot_fw	2- stage boot loader
common	
⊞ dma_examples	Bulk loop DMA examples
lpp_source	LPP library
⊞ MSC	MSC example
⊞ serialif_examples	Serail interface examples
⊞ slavefifo_examples	Slave FIFO examples
⊞ u3p_firmware	Firmware libraries and headers
⊞ uac_examples	USB Audio Class examples
⊞ uvc_examples	USB Video Class examples
license	
⊞ util	Firmware image conversion tool

The installed USB Suite (Host Drivers and libraries) contains the following:



```
Local Disk (C:)
  Cypress
    Cypress SuperSpeed USBSuite          Default Installation Folder
      application
        c_sharp
          bulkloop
          controlcenter                  Sample C# Applications
          streamer
        cpp
          bulkloop                       Sample C++ Applications
          streamer
      bin
      driver
        bin
          win7
          wlh                            Cy USB Driver
          wxp
        inc
      library
        c_sharp
          lib                            C# API Library
        cpp
          inc                            C++ API Library
          lib
      license
```

The installed Toolchain and IDE contain the following:



```
⊟ Local Disk (C:)
    ⊟ Program Files
        ⊟ Cypress
            ⊟ ARM GCC
                ⊞ arm-none-eabi
                  bin
                ⊞ lib
                ⊞ libexec
                ⊞ share
                ⊞ uninstall
            ⊟ eclipse
                ⊞ configuration
                ⊞ features
                ⊞ jre7
                ⊞ p2
                ⊞ plugins
                  readme
```

Default Installation Folder

ARM GNU Toolchain

Eclipse IDE

3 Working with the SDK

3.1 Programming the FX3 device

The FX3 examples can be built and run on the FX3 board. The following steps describe the setup and boot process:

1. Install the components of the SDK (Firmware, Toolchain, IDE and Host drivers) on the host machine.
2. Connect the FX3 board to the host machine and power the board up.
3. Bind the driver for the FX3 device.
4. Launch the IDE, import the example projects and build them
5. Launch the CyControl Center utility (available from USB Suite software)
6. Download the boot image to the FX3 device using the Control Center “Download Firmware” feature.
7. The FX3 device will reboot using this new firmware.

These steps are explained in more detail in the following sections.

The UsbBulkLoopAuto example is used here to explain the steps. This example implements a vendor specific USB 3.0 peripheral with two bulk endpoints. The firmware application loops back any data that is sent on the OUT endpoint on the IN endpoint.

Section 8 of the FX3 Programmers Manual describes the overall application structure.

It is assumed that the FX3 DVK board is used to run the firmware.

Note: Since the host side tools for FX3 (USB Suite) are only supported on Windows, a Windows based host is needed to perform these steps.

3.2 Building the firmware

Each of the firmware examples in the FX3 SDK has an Eclipse project associated with it, which can be used for building and debugging the application. The instructions for building and running the Firmware examples using the Eclipse IDE and GNU toolchain are provided in detail in Section 11.2.2 of the FX3 Programmers Manual. The section details how to import the projects, configure settings for the projects, debug and run the example using GNU debugger.

Import all of the firmware examples in the SDK into the Eclipse IDE. The example applications are automatically built when they are imported, and generate ELF binaries. These binaries are converted by the Eclipse projects to binary image format using the elf2img utility. This binary image format can be programmed onto the FX3 part. The elf2img binary conversion utility is part of the FX3 SDK (<Installation folder>\FX3 SDK\util\elf2img). The eclipse project post build step uses the following command for the conversion. Refer to the readme.txt under (<Installation folder>\FX3 SDK\util\elf2img) for more information on the conversion tool. Also refer to application note AN68914 - EZ-USB® FX3 I2C Boot Option for I2C boot and AN70193 - EZ-USB® FX3 SPI Boot Option for SPI boot.

```
elf2img.exe –i BulkLoopAuto.elf –o BulkLoopAuto.img
```

3.2.1 Serial Peripheral Drivers and APIs

The drivers and APIs for the FX3 serial peripheral interfaces (I2C, I2S, SPI, UART and GPIO) are provided with the FX3 SDK in source form. Full register documentation for these blocks is part of the FX3 Programmer's Manual.

The drivers and APIs for the serial peripherals are built into a separate ARM EABI library (cyu3lpp.a). The source code as well as the Eclipse IDE projects for building this library can be found in the SDK under the <Installation Folder>\FX3 SDK\firmware\lpp_source folder.

All of the FX3 application examples link with this library in addition to the FX3 API library consisting of the drivers and APIs for the device, USB and GPIF blocks (cyfxapi.a).

3.3 Setting up the FX3 DVK Board

The FX3 device can be configured to boot from a USB host by enumerating as a custom device. The boot mode for the device is specified through a set of three PMODE pins that can be controlled through some jumpers and switches on the FX3 DVK board.

Please refer to section 1.1.6.1 of the FX3 DVK User Guide for the settings for the PMODE pins.

3.4 Host driver binding

The steps for installation and binding of the host drivers for different Windows platforms are described in section 3 of the host driver help file, CyUSB.pdf, installed as part of the USB Suite.

Once the board has been connected to the host PC, the device will be seen enumerating and asking for driver selection. For booting, the FX3 enumerates as a USB 2.0 device with VID=0x04B4 and PID=0x00F3.

Select the "Install from a specific location" and point to the *<Installation Root>/Cypress SuperSpeed USBSuite/driver/bin/<os>/<arch>/cyusb3.inf* file to bind the device with the Cypress CyUsb3.sys driver.

The corresponding VID/PID entries must be made to cyusb3.inf for the appropriate VID/PID of the device connected.

3.5 Firmware Download

The CyControl center utility, which can be used to communicate with the FX3 device, is described in CyControlCenter.pdf, installed as part of the USB Suite.

Once the driver binding has been completed, open the CyControl Center (Start->Programs->Cypress->Cypress SuperSpeed USBSuite->Control Center). The Cypress FX3 Device should be visible through the tool and you can look through the USB descriptors reported by the device.

Use the Program->FX3 option on the tool to program the previously generated BulkLoopAuto.img binary file onto the device. This download is performed directly onto the RAM on the FX3 device.

3.6 Testing the application

Once the download is complete, a new USB device can be seen enumerating and asking for driver selection. Repeat the driver selection process performed during initial boot and associate the same CyUsb3.sys driver with the new device as well.

The data transfer feature of the CyControl Center can be used to verify the data loop-back functionality.

4 Firmware Example Overview

The firmware examples included in the FX3 SDK are listed below. These examples are provided as separate Eclipse projects.

4.1 USB Bulk data loopback examples

These examples illustrate a loopback mechanism between two/three USB Bulk Endpoints. The example comprises of Vendor Class USB enumeration descriptors with 2/3 Bulk Endpoints. The DMA multichannel examples use 3 endpoints for the loopback.

Following are the different types of Bulk data loopbacks provided. These examples are provided as Eclipse projects.

1. cyfxbulklpauto: This example makes use of the DMA AUTO Channel for the loopback between the endpoints.
2. cyfxbulklpautosig: This example makes use of the DMA AUTO Channel with Signaling for the loopback between the endpoints.
3. cyfxbulklpmanual: This example makes use of the DMA MANUAL Channel for the loopback between the endpoints.
4. cyfxbulklpmaninout: This example makes use of the DMA MANUAL IN + DMA MANUAL OUT Channel for the loopback between the endpoints.
5. cyfxbulklpautomanytoone: This example makes use of the Multichannel DMA AUTO MANY TO ONE for the loopback between endpoints.
6. cyfxbulklpautoonetomany: This example makes use of the Multichannel DMA AUTO ONE TO MANY for the loopback between endpoints.
7. cyfxbulklpmanonetomany: This example makes use of the Multichannel DMA MANUAL ONE TO MANY for the loopback between endpoints.
8. cyfxbulklpmanmanytoone: This example makes use of the Multichannel DMA MANUAL MANY TO ONE for the loopback between endpoints.
9. cyfxbulklpmulticast: This example makes use of the Multichannel DMA MULTICAST for the loopback between endpoints.
10. cyfxbulklpman_removal: This example demonstrates the use of DMA MANUAL channels where a header and footer get removed from the data before sending

out. The data received from EP1 OUT is looped back to EP1 IN after removing the header and footer. The removal of header and footer does not require the copy of data.

11. cyfxbulklplowlevel: The DMA channel is a helpful construct that allows for simple data transfer. The low level DMA descriptor and DMA socket APIs allow for finer constructs. This example uses these APIs to implement a simple bulkloop back example where a buffer of data received from EP1 OUT is looped back to EP1 IN.
12. cyfxbulklpmandcache: FX3 device has the data cache disabled by default. The data cache is useful when there is large amount of data modifications done by the CPU. But enabling D-cache adds additional constraints for managing the data cache. This example demonstrates how DMA transfers can be done with the data cache enabled.

All the above examples are implemented to work at USB 2.0 and USB 3.0 speeds.

4.2 USB Isochronous data loopback examples

These examples illustrate a loopback mechanism between two USB Isochronous Endpoints. The example comprises of Vendor Class USB enumeration descriptors with 2 Isochronous Endpoints.

Following are the different types of Isochronous data loopbacks provided. These examples are provided as Eclipse projects.

1. cyfxisolpauto: This example makes use of the DMA AUTO Channel for the loopback between the endpoints.
2. cyfxisolpmaninout: This example makes use of the DMA MANUAL IN + DMA MANUAL OUT Channel for the loopback between the endpoints.

The above examples are implemented to work at USB 2.0 and USB 3.0 speeds.

4.3 USB debug example

cyfxusbdebug: This example demonstrates the use of USB interrupt endpoint to log the debug data from the FX3 device. The default debug logging in all other examples are done through the UART. This example shows how any consumer socket can be used to log FX3 debug data.

4.4 USB Video Class example

1. cyfxuvcinmem: This example implements a USB Video Class Driver with the help of the appropriate USB enumeration descriptors. With these descriptors the FX3 device enumerates as a USB Video Class device on the USB host. The video streaming is accomplished with the help of a DMA Manual Out channel. Video frames are stored in contiguous memory locations as a constant array. The streaming of these frames continuously from the device results in a video like appearance on the USB host.

2. cyfxuvcinmem_bulk: This example demonstrates the USB video class device stack implementation for FX3. The example is similar to the UVC example, but uses Bulk endpoints instead of isochronous endpoints.

These examples are implemented to work at USB 2.0 and USB 3.0 speeds.

4.5 Slave FIFO Application examples

The Slave FIFO application example demonstrates data loopback between the USB Host and the PCI host. The example comprises of two data pipes between the USB Host and the PCI host. Data is looped back at the PCI host. The loopback is observed on the USB host. GPIF™ - II interface is configured to implement the Slave FIFO protocol for Sync and Async modes (16 and 32 bit configurations).

1. cyfxslaveasync: Asynchronous mode Slave FIFO example using 2 bit FIFO address.
2. cyfxslavesync: Synchronous mode Slave FIFO example using 2 bit FIFO address.
3. cyfxslaveasync5bit: Asynchronous mode Slave FIFO example using 5 bit FIFO address.
4. Cyfxslavesync5bit: Synchronous mode Slave FIFO example using 5 bit FIFO address.

The above examples are implemented to work at USB 2.0 and USB 3.0 speeds.

4.6 Serial Interface examples

The serial interface examples demonstrate data accesses to the Serial IOs, I2C, SPI and UART.

4.6.1 UART examples

In the UART examples, the data is looped from the PC host back to the PC host through the FX3 device. The loopback can be observed on the PC host. The examples are provided for both DMA mode and Register mode of data transfers for the UART.

1. cyfxuartlpdmamode: This example makes use of the DMA MANUAL Channel for the loopback to the PC host over UART. The DMA buffer is 32 bytes.
2. cyfxuartlpregmode: This example loops back data one byte at a time.

4.6.2 I2C examples

In the I2C examples, a USB I2C data loopback is achieved over 2 USB Isochronous endpoints through an I2C slave device. A DMA MANUAL IN and a DMA MANUAL OUT channels are used for the Isochronous endpoints. The data is looped back one I2C page at a time which is 64 bytes. The examples are implemented for USB 2.0 speeds.

1. cyfxusbi2cdmamode: This example makes use of the DMA MANUAL IN and a DMA MANUAL OUT channel for the I2C DMA access.
2. cyfxusbi2cregmode: This example reads/writes one page to the I2C device using the register mode accesses.

4.6.3 SPI examples

In the SPI examples, a USB SPI data loopback is achieved over 2 USB Isochronous endpoints through a SPI slave device. A DMA MANUAL IN and a DMA MANUAL OUT channels are used for the Isochronous endpoints. The data is looped back one page at a time which is 16 bytes. The examples are implemented for USB 2.0 speeds.

1. cyfxusbspidmamode: This example makes use of the DMA MANUAL IN and a DMA MANUAL OUT channel for the SPI DMA access.
2. cyfxusbspiregmode: This example reads/writes one page to the SPI device using the register mode accesses.
3. cyfxusbspigpiomode: This example demonstrates the use of GPIO to build an SPI master. The example read / writes data to an SPI Flash attached to the FX3 device using FX3 GPIOs.

4.6.4 I2S example

cyfxusbi2sdmamode: This example demonstrates the use of I2S APIs. The example sends the data received on EP1 OUT to the left channel and EP2 OUT to the right channel.

4.6.5 GPIO examples

This is a simple GPIO application example illustrating the usage of GPIO operations with the help of an Input and an Output GPIO.

1. cyfxgpioapp: This example implements simple Set operation on the Output GPIO and Get operation on the Input GPIO. It also implements GPIO interrupt for the Input GPIO.
2. Cyfxgpiocomplexapp: This example implements the following features:
   a) A PWM output.
   b) Input line which measures low time of the input signal.
   c) A counter which increments on the negative edge of input signal.

4.7 USB Bulk/Isochronous data source sink examples

These examples illustrate data source and data sink mechanism with two USB Bulk/Isochronous Endpoints. The example comprises of Vendor Class USB enumeration descriptors with 2 Bulk/Isochrounous Endpoints. The examples are implemented to work for USB 2.0 and USB 3.0 speeds.

1. cyfxbulksrcsink: This example makes use of the DMA MANUAL IN channel for sinking the data received from the Bulk OUT endpoint and DMA MANUAL OUT Channel for the sourcing the data to the Bulk IN endpoint.
2. cyfxisosrcsink: This example makes use of the DMA MANUAL IN channel for sinking the data received from the Isochronous OUT endpoint and DMA MANUAL OUT Channel for the sourcing the data to the Isochronous IN endpoint.

4.8 USB Bulk Streams example

This example illustrates data source and data sink mechanism with two USB Bulk Endpoints using the Bulk Streams. A set of Bulk Streams are defined for each of the endpoints (OUT and IN). Data source and data sink happen through these streams. Streams are applicable to USB 3.0 speeds only. The example works similar to Bulk source sink example (cyfxbulksrcsink) when connected to USB 2.0.

1. cyfxbulkstreams: This example makes use of the DMA MANUAL IN channel for sinking the data received from the streams of the Bulk OUT endpoint and DMA MANUAL OUT Channel for the sourcing the data to the streams of the Bulk IN endpoint.

4.9 USB enumeration example

The enumeration example is an implementation of a USB bulk loop with normal mode enumeration in FX3.

1. cyfxbulklpautoenum: The example illustrates the normal mode enumeration mechanism provided for FX3 where all setup requests are handled by the application. The example implements a simple bulk loop.

4.10 Flash Programmer example

This example illustrates the programming of I2C EEPROMS and SPI flash devices from USB. The read / write operations are done using pre-defined vendor commands. The utility can be used to flash the boot images to these devices.

4.11 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.

4.12 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. Since the audio class does not require high bandwidth, this example works only at USB 2.0 speeds.

4.13 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 be used for Real View Tool chain is also provided.

4.14 USB host and OTG examples

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

1. cyfxusbhost: Mouse and MSC driver for FX3 USB Host. This example demonstrates the use of FX3 as a USB 2.0 single port host. The example supports simple HID mouse class and simple MSC class devices.
2. cyfxusbotg: FX 3 as an OTG Device. This example demonstrates the use of FX3 as an OTG device which when connected to a USB host is capable of doing a bulkloop back using DMA AUTO channel. When connected to a USB mouse, it can detect and use the mouse to track the three button states, X, Y, and scroll changes.
3. cyfxbulklpotg: FX3 Connected to FX3 as OTG Device. This example demonstrates the full OTG capability of the FX3 device. When connected to a USB PC host, it acts a bulkloop device. When connected to another FX3 in device mode running the same the firmware, both can demonstrate session request protocol (SRP) and host negotiation protocol (HNP).

5 FX3 Programming Guidelines

This section provides a few basic guidelines for developing applications using the FX3 SDK.

5.1 Device Initialization

5.1.1 Clock Settings

The first step involved in initializing the FX3 device is setting up the frequencies for various internal clocks. There are two classes of clocks on the FX3 device:

1. Some of the clocks such as the clock for the ARM CPU, the memory mapped register access and system DMA are expected to run continuously at all times.
2. Other clocks such as those for the USB block, the GPIF block, and the serial peripherals are only enabled when required; i.e., when the corresponding block init function has been called.

All of the clocks on the FX3 device are derived from a master clock which runs at approximately 400 MHz. If the FX3 device is being clocked using a 19.2 MHz crystal or using a 38.4 MHz clock input; the master clock frequency is set to 384 MHz by default. If the FX3 device is being clocked using a 26 MHz or 52 MHz input clock, the master clock frequency is set to 416 MHz.

The clock frequency for the ARM CPU is set to one half of the master clock frequency. The clock frequency for the system DMA and register access is set to one-fourth of the master clock frequency. These frequencies can be reduced further using the CyU3PDeviceInit() API.

If the system design requires the FX3 device to receive data on a 32 bit wide GPIF interface running at 100 MHz (yielding a maximum data rate of 400 MBps on the GPIF interface); a master clock setting of 384 MHz is insufficient for the system DMA to handle the incoming data. In such a case, the master clock needs to be changed to a value greater than 400 MHz.

This change is done through the CyU3PDeviceInit() API call. The clkCfg->setSysClk400 parameter passed to this function needs to be set to CyTrue to enable this frequency change. If this parameter is set to CyTrue, the firmware causes the master clock frequency to be changed to 403.2 MHz.

Note: It has been noted that changing the master clock frequency causes a transient instability on the device interfaces, which may cause an ongoing JTAG debug session to break. To minimize the impact of this instability, this frequency change is performed by firmware before any of the external interfaces on the device are initialized.

5.1.2 Setting up the IO Matrix

The next step in the device initialization is setting up the functionality for various IO pins on the device. Almost all of the device IOs can serve multiple functions, and the actual function to be used is selected through the CyU3PDeviceConfigureIOMatrix API call.

Please note that some constraints apply to the possible IO configurations. For example, it is not possible to use the SPI interface on the FX3 device if the GPIF is running in a 32 bit wide configuration.

Any of the pins on the device can be overridden to function as a GPIO instead of serving as part of another interface such as GPIF, UART, I2C etc. The CyU3PDeviceConfigureIOMatrix() makes some sanity checks to ensure that a pin that is otherwise in use, cannot be overridden as a GPIO pin.

e.g., if the UART interface is enabled using the useUart parameter; the API does not allow any of the UART pins (TX, RX, RTS and CTS) to be used as GPIOs.

It is possible that the above checks in the API constrain the user. For example, if the user does not plan to use flow control for the UART, there is no reason to prevent the RTS and CTS pins from being used as GPIOs.

In such a case, the CyU3PDeviceGpioOverride() API can be used to forcibly override the pin functionality to serve as a GPIO.

5.1.3 Using the Caches

The FX3 device implements 8 KB of instruction and data caches that can be used to speed up instruction and data access from the ARM CPU. It is recommended that the instruction cache be kept enabled in all applications for optimal functioning of the firmware.

If the firmware application makes use of any CPU bound data copy actions, turning the data cache on will improve the performance significantly.

The instruction and data caches are enabled/disabled using the CyU3PDeviceCacheControl() API.

Whenever the data cache is turned on, care needs to be exercised to prevent data corruption due to unexpected cache line evictions. It is required that the isDmaHandleDCache parameter to the CyU3PDeviceCacheControl() API be set to CyTrue, whenever the isDCacheEnable parameter is set to CyTrue.

5.1.4 Initializing other interface blocks

This section applies to the initialization sequence for interface blocks like USB, GPIF, I2C, UART etc. All of these blocks will be held in reset at the time when firmware execution starts.

The procedure for turning on any of these blocks involves:

1. Turning on the clock for the block
2. Bringing the block out of reset.
3. All of the interface blocks on the FX3 device (except the GPIOs) have a set of DMA sockets associated with them in addition to the core interface logic. It is required that these sockets be reset and initialized with clean default values when the corresponding block is being turned on.

These steps are performed by the init function associated with these blocks (CyU3PUsbStart, CyU3PPibInit, CyU3PUartInit etc.).

Since the DMA sockets associated with a block are reset during the block initialization; it is expected that the blocks are initialized before any DMA channels using these sockets are created by the application.

e.g., the PIB (GPIF) block needs to be initialized before any DMA channels using the PIB sockets are created.

The user also needs to ensure that the block is not repeatedly initialized at a later stage, thereby affecting the DMA channel functionality.

5.2 USB Device Handling

5.2.1 USB Device Enumeration

The FX3 firmware API supports two models of handling USB device enumeration.

In the fast enumeration model, the user registers a set of USB descriptors with the API; and the USB driver handles the control requests from the USB host using this data. The advantage of this model is that the control request handling in the driver has been tested to pass all USB compliance test requirements; and the user does not need to implement all of the negative condition checks required to pass these tests. The driver will continue to forward control request callbacks for any unknown requests (vendor or class specific, as well as requests for unregistered strings) to the user application.

In the application enumeration model, all control requests will trigger a callback to the user application; and the application can handle the request as suitable. The advantage of this model is that allows the application to implement any number of configurations and provides greater flexibility.

The model of enumeration is selected using the CyU3PUsbRegisterSetupCallback() API.

5.2.2 Handling Control Requests

On receiving a USB control request through the setup callback, the user application can perform one of four actions:

1. Call CyU3PUsbAckSetup() to complete the status handshake part of a control request with no data phase.
2. Call CyU3PUsbStall(0, CyTrue, CyFalse) to stall endpoint 0 so as to fail the control request.
3. Call CyU3PUsbSendEP0Data() to send the data associated with a control request with an IN data phase.
4. Call CyU3PUsbGetEP0Data() to receive the data associated with a control request with an OUT data phase.

It is possible to make multiple SendEP0Data/GetEP0Data calls to complete the data phase of a transfer if the amount of data to be transferred is large. In such a case, the amount of data transferred using each API call should be an integral multiple of the maximum packet size for the control endpoint (64 bytes for USB 2.0, 512 bytes for USB 3.0).

Please note that the FX3 device architecture does not allow a control transfer to be stalled after the data phase has been completed. Therefore, the application should NOT stall the control endpoint if there is an error in handling a control transfer after the OUT data has been read using the CyU3PUsbGetEP0Data API.

e.g., Do not stall EP0 if the write to I2C slave fails after receiving data through the CyU3PUsbGetEP0Data() API call.

5.2.3 Endpoint Configuration

All of the FX3 application examples in the SDK configure the non-control endpoints while processing the SET_CONFIGURATION request. This allows the application to determine the USB connection speed, and then configure the endpoint accordingly.

It is also possible to configure the endpoints and DMA channels ahead of the USB device enumeration. As long as the endpoint and DMA channels are configured using the USB 3.0 parameters, they will continue to work fine at hi-speed and full speed as well.

There is one caveat that applies in this case.

If there is an OUT endpoint that has a maximum packet size of N bytes, and the DMA channel has been created with a buffer size of M * N bytes (where M is an integer greater than 1); the data received on the endpoint will be accessible to the consumer only after the buffer is filled up, or a short packet is received.

e.g., If the maximum packet size is 512 bytes and the DMA buffer size is 1024 bytes; the buffer can be read by the consumer only after the buffer is filled up (receives two full packets) or if a short packet is received. If only one full packet is received on the endpoint, the data will sit in the buffer and will not be accessible to the consumer.

If this behavior is undesirable, the endpoint behavior can be modified using the CyU3PSetEpPacketSize() or CyU3PUsbSetEpPktMode() APIs.

The CyU3PSetEpPacketSize() API allows the maximum packet size definition for the endpoint to be modified such that all incoming packets are treated as short packets, resulting in immediate buffer commit. For example, if this API is used to set the maximum packet size as 1024 bytes in the above case, all 512 byte packets will be treated as short packets; and made available to the consumer immediately.

The CyU3PUsbSetEpPktMode() API configures the endpoint in packet mode, where it commits each incoming data packet into one DMA buffer (independent of whether it is full or short). The downside of this approach is that it can limit data transfer performance in some cases. It is recommended that the packet mode not be used for any applications that use large DMA buffers for better performance.

5.2.4 USB Low Power Mode Handling

The USB 3.0 specification includes provision for link level low power modes which are used to save system power consumption when the link is idle. It has been noted that different host controllers (in some cases even the host controller drivers) use the low power modes differently. In some cases, the host is very aggressive and tends to push the link into U1/U2 modes whenever it is waiting for data from the device. In other cases, the host waits a little longer before trying to initiate a transition into a low power mode.

The FX3 device handles U1/U2 transitions passively in most cases, because the device cannot initiate a U0->Ux transition without firmware intervention. Due to this constraint, the normal power mode handling in FX3 involves the device accepting or rejecting low power mode requests from the host; and then initiating a transition back to the U0 mode when instructed by firmware.

By default, the device is placed in a state where it accepts or rejects low power mode requests automatically based on whether it has any data ready to send out or not. This behavior can be overridden to a state where it systematically rejects all attempts by the host to push the link into a low power mode. This change is requested through the CyU3PUsbLPMDisable () API.

The use of this API is recommended if the power mode transitions on the USB link are limiting the performance of the system to unacceptable levels.

The FX3 device does not automatically initiate a transition back to U0 (from U1/U2) when it has data ready to go out. In the case of control transfers, the USB driver in the FX3 firmware is aware that data is ready; and initiates a transition back to U0 at the appropriate time.

In the case of other endpoints, it is possible that the data transfer be blocked because the link is stuck in a low power mode. It is recommended that any applications that do not use the

CyU3PUsbLPMDisable() API, should call the CyU3PUsbSetLinkPowerState(CyU3PUsbLPM_U0) API periodically to ensure that the link does not get stuck in a low power mode.

5.3 Porting Applications from SDK 1.2 to SDK 1.2.1

There are no major changes to the FX3 API set between the 1.2 and 1.2.1 releases. However, the following points need to be kept in mind when migrating an existing SDK 1.2 based application to the SDK 1.2.1 version.

1. A new API called CyU3PUsbEnableEPPrefetch() has been added in the 1.2.1 SDK version. This API updates the DMA interface settings for the USB block to pre-fetch data from the sockets more aggressively. These settings were previously left enabled in all cases.

   If the firmware application makes use of multiple USB IN endpoints on a regular basis, this API should be called immediately after the CyU3PUsbStart() API. This is not required if the firmware application has only one IN endpoint which is accessed regularly.

2. The multicast DMA channel handlers have been kept disabled from regular application builds to reduce memory footprint. If the firmware application makes use of any multicast DMA channels, these handlers need to be enabled by calling CyU3PDmaEnableMulticast(). This API needs to be called before creating any multicast DMA channels.

3. A new USB event called CY_U3P_USB_EVENT_LNK_RECOVERY has been added to provide notification of cases where the device enters USB 3.0 link recovery. As this callback is provided in interrupt context, performing time consuming operations such as calling CyU3PDebugPrint is not allowed.

   If the USB event callback in the firmware application performs such actions in the default case (where the event type does not match other specified values), it will need to be updated to avoid these actions for this event type.