AN76348

Differences in Implementation of EZ-USB® FX2LP™ and EZ-USB FX3™ Applications

Author: Rama Sai Krishna V
Associated Part Family: CY7C6801XA, CYUSB301X
Related Application Note`s: For a complete list of the application notes, click here.

With the release of the USB 3.0 specification, USB controller chips required rearchitecting to handle the ten-fold increase in USB bandwidth over the USB 2.0 specification. Cypress offers USB controllers for applications based on USB 2.0 (EZ-USB® FX2LP™) and USB 3.0 (EZ-USB FX3™). This application note describes the implementation differences between the two controllers. Example applications highlight those differences at the architectural, hardware, and firmware framework levels. While this application note emphasizes the new FX3 features, it provides links to FX2LP background materials. For the complete list of SuperSpeed code examples, visit http://www.cypress.com/?rID=101781 and for the complete list of Hi-Speed code examples, visit http://www.cypress.com/?rID=101782.

Contents

1 Introduction

Cypress EZ-USB FX3 is a USB 3.0 peripheral controller with highly integrated and flexible features that add USB 3.0 functionality to any system.

FX3 has a fully configurable, parallel general programmable interface called GPIF II, which can connect to an external processor, ASIC, or FPGA. The GPIF II is an enhanced version of the GPIF in FX2LP, Cypress's flagship USB 2.0 product. GPIF II provides glueless connectivity to popular interfaces such as asynchronous SRAM, asynchronous and synchronous address data multiplexed interface, and others. To accommodate the USB 3.0 SuperSpeed signaling rates, FX3 offers architectural enhancements over FX2LP such as a RISC processor and DMA system. This application note explains the architectural differences and introduces the FX3 RTOS-based firmware frameworks.

EZ-USB FX2LP-based designs are not directly portable to FX3 due to the architectural differences. This application note uses a simple LED blinking example to introduce the FX3 firmware frameworks. It then uses a simple example, bulkloop, to explain the differences in the firmware frameworks and to provide guidelines on how an FX2LP application can be modified to work on FX3. The bulkloop example loops data over two USB BULK endpoints under the control of a Windows based application.

Note: This application note is intended for users who have experience working with the FX2LP device.

The following application notes can help you get started working with FX2LP or FX3:

- AN65209 – Getting Started with FX2LP™ gives you background information about USB 2.0 along with the hardware, firmware, and software aspects of working with the FX2LP.
- AN75705 – Getting Started with EZ-USB® FX3™ contains background information about USB 3.0 and comparisons with USB 2.0. It details the hardware, firmware, and software aspects of working with the FX3.

2 Architectural Differences

Figure 1 and Figure 2 show the FX2LP and FX3 block diagrams, respectively. Table 1 summarizes the feature differences.

Figure 1. FX2LP Block Diagram

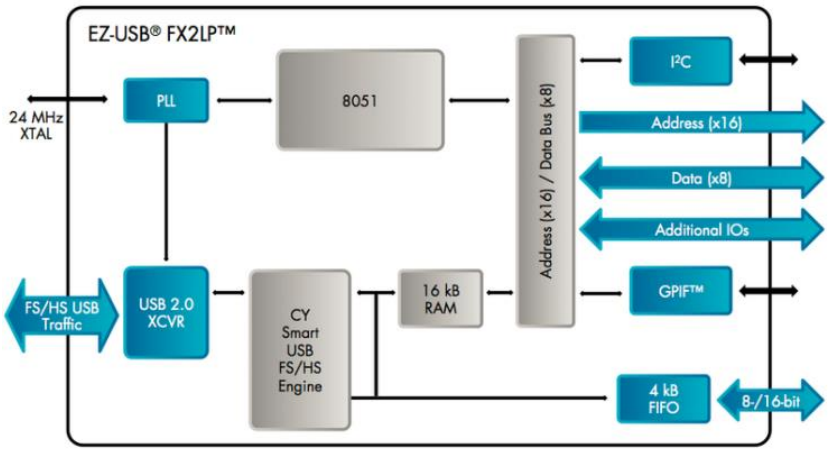


Figure 2. FX3 Block Diagram

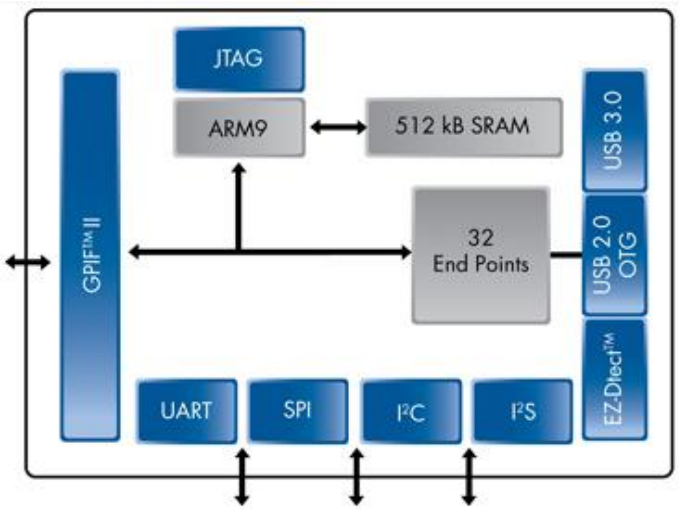


Table 1. Feature Differences Between FX2LP and FX3

Feature	FX2LP	FX3
Core	8051	ARM926EJ-S
CPU speed	48 MHz	200 MHz
RAM	16 KB	512 KB
Endpoints	7	32
Serial interfaces supported	I^2C, UART	I^2C, UART, I^2S, SPI
Flexible programmable interfaces	GPIF, 48 MHz, 8 – and 16-bit interface	GPIF II, 100 MHz, 8-, 16-, and 32-bit interface
USB	USB 2.0 device	USB 3.0 device (includes USB 2.0 device support), USB 2.0 OTG
Speeds supported	High Speed, Full Speed	SuperSpeed, High Speed, Full Speed
GPIOs	Up to 40	Up to 60
JTAG debugger	Not available	Supported
Support for battery charging spec 1.1	No	Yes
Package options	56-pin QFN (8 x 8 mm) 100-pin TQFP (14 x 20 x 1.4 mm) 128-pin TQFP (14 x 20 x 1.4 mm) 56-pin VFBGA (5 x 5 x 1.0 mm) 56-pin SSOP	121-ball FBGA (10 x 10 x 1.2 mm)

3 Serial Interfaces

Table 2 lists the details on the serial interfaces supported by FX2LP and FX3.

Table 2. Serial Interfaces Supported by FX2LP and FX3

Serial Interface	FX2LP	FX3
I^2C	Master only at 100 and 400 kHz	Master only at 100 kHz, 400 kHz and 1 MHz
UART	Baud rates from 2.4 Kbps to 230.4 Kbps	Range of baud rates from 300 bps to 4608 Kbps
I^2S	Not supported	I^2S master as transmitter only; sampling frequencies supported by the I^2S interface are 32 kHz, 44.1 kHz, and 48 kHz
SPI	Not supported; SPI master interface can be implemented by bit-banging GPIOs	SPI master; maximum frequency of operation is 33 MHz

4 GPIF Versus GPIF II

FX3 offers a high-performance general programmable interface, GPIF II. This interface enables functionality similar to but more advanced than the FX2LP's GPIF and Slave FIFO interfaces.

The GPIF II is a programmable state machine that enables a flexible interface that functions either as a master or a slave in industry-standard or proprietary interfaces. Both parallel and serial interfaces can be implemented with GPIF II.

The features of the GPIF II are as follows:

- Functions as master or slave.

- Provides 256 firmware programmable states.
- Supports 8-bit, 16-bit, 24–bit, and 32-bit parallel data bus.
- Enables interface frequencies up to 100 MHz
- Supports 14 configurable control pins when a 32-bit data bus is used. All control pins can be either input/output or bidirectional.
- Supports 16 configurable control pins when a 16-bit or 8-bit data bus is used. All control pins can be either input/output or bidirectional.

Table 3 lists the main difference between the FX2LP GPIF interface and the FX3 GPIF II interface.

Table 3. Differences Between GPIF and GPIF II

Feature	GPIF	GPIF II
Interface clock	48 MHz	100 MHz
Data bus width	8-bit and 16-bit	8-bit, 16-bit, 24-bit, and 32-bit
Address lines	9	32 when data bus is 8-bit
Control/status lines	CTL[5:0] and RDY[5:0]	14 when 32-bit data bus is used 16 when 16- or 8-bit data bus is used
Number of states	8 (including IDLE)	256
Software tool	GPIF designer	GPIF II designer

5 Hardware Differences

The following application notes describes the hardware design guidelines for FX2LP and FX3, including the recommended pad sizes.

- Hardware design guidelines for FX2LP: AN15456 – Guide to Successful EZ-USB® FX2LP™ Hardware Design
- Hardware design guidelines for FX3: AN70707 – EZ-USB® FX3™/FX3S™ Hardware Design Guidelines and Schematic Checklist

5.1 Power Supply Configurations and Decoupling Capacitance

FX2LP requires a supply voltage of 3.3 V. FX3 requires multiple power supplies for its various internal blocks. Table 4 shows the different power domains and the voltage settings on each of these domains for FX3.

Table 4. FX3 Power Domains

Parameter	Description	Min (V)	Max (V)	Typical (V)
V_{DD}	Core voltage supply	1.15	1.25	1.2
A_{VDD}	Analog voltage supply	1.15	1.25	1.2
V_{IO1}	GPIF II I/O power domain	1.7	3.6	1.8, 2.5, and 3.3
V_{IO2}	IO2 power domain	1.7	3.6	1.8, 2.5, and 3.3
V_{IO3}	IO3 power domain	1.7	3.6	1.8, 2.5, and 3.3
V_{IO4}	UART/SPI/I^2S power domain	1.7	3.6	1.8, 2.5, and 3.3
V_{IO5}	I^2C and JTAG supply domain	1.15	3.6	1.2, 1.8, 2.5, and 3.3
V_{BATT}	USB voltage supply	3.2	6	3.7
V_{BUS}	USB voltage supply	4.1	6	5
C_{VDDQ}	Clock voltage supply	1.7	3.6	1.8, 3.3
$U3TX_{VDDQ}$	USB 3.0 1.2-V supply	1.15	1.25	1.2
$U3RX_{VDDQ}$	USB 3.0 1.2-V supply	1.15	1.25	1.2

Table 5. shows the I/O voltage comparison for FX2LP and FX3.

Table 5. I/O Voltage Comparison

Parameter	Description	Min (V)		Max (V)		Conditions	
		FX2LP	FX3	FX2LP	FX3	FX2LP	FX3
V_{IH}	Input HIGH voltage	2	0.625 x V_{CC}	5.25	V_{CC} + 0.3	–	2.0 V ≤ V_{CC} ≤ 3.6 V*
			V_{CC} – 0.4		V_{CC} + 0.3		1.7 V ≤ V_{CC} ≤ 2.0 V*
V_{IL}	Input LOW voltage	–0.5	–0.3	0.8	0.25x V_{CC}	–	–
V_{OH}	Output HIGH voltage	2.4	0.9 x V_{CC}	–	–	I_{OUT} = 4 mA	I_{OH} (max) = –100 µA
V_{OL}	Output LOW voltage	–	–	0.4	0.1 x V_{CC}	I_{OUT} = –0.4mA	I_{OH} (min) = 100 µA

*Except the USB port; V_{CC} is the corresponding I/O voltage supply.

Refer to the FX2LP and FX3 datasheets for more details on I/O voltages.

FX2LP designs require 0.1-µF ceramic decoupling capacitors on the device power pins. Table 6 shows FX3 power supply decoupling recommendations.

Table 6. Power Domain Decoupling Requirements

Cap Value (µF)	Number of Caps	Pin Name
0.01, 0.1, 22	4 of 0.01 µF, 3 of 0.1 µF, 1 of 22 µF	VDD
0.1, 2.2	1 of each	AVDD
0.1, 22	1 of each	U3TXVDDQ
0.1, 22	1 of each	U3RXVDDQ
0.1, 0.01	1 of each	CVDDQ
0.1, 0.01	1 of each per supply	VIO1-5
0.1	1	VBUS

5.2 Booting Options

FX2LP boots from USB or from an I^2C EEPROM. FX3 can load boot images from various sources, selected by the configuration of the PMODE pins. The boot options for FX3 are:

- Boot from USB
- Boot from I^2C (ATMEL and Microchip EEPROMs are supported)
- Boot from SPI (SPI devices supported are M25P16 (16 Mb), M25P80 (8 Mb), M25P40 (4 Mb), and their equivalents
- Boot from GPIF II Async ADMUX mode
- Boot from GPIF II Sync ADMUX mode
- Boot from GPIF II Async SRAM mode

See AN50963 for more details on FX2LP boot options and AN76405 for more details on FX3 boot options.

Table 7 shows the levels of the PMODE[2:0] signals required for the different booting options.

Table 7. PMODE Signals Setting

PMODE[2:0]	Boot From
F00	Sync ADMUX (16-bit)
F01	Async ADMUX (16-bit)
F11	USB boot
F0F	Async SRAM (16-bit)
F1F	I^2C, on Failure, USB boot
1FF	I^2C only
0F1	SPI, on Failure, USB boot

F = Float. The PMODE pin floats by leaving it unconnected.

If an external EEPROM is used on the I^2C bus for firmware image booting, then the SCL and SDA lines should be pulled high using 2 kΩ to VIO5 of FX3.

Cypress recommends adding pull-up and pull-down options on the PMODE [2:0] signals and loading the combination needed for the preferred booting option. Adding the options gives the flexibility to debug the system during early development.

5.3 Crystal/Clock

FX2LP and FX3 support an external clock input along with the crystal input. FX2LP has a CLKOUT pin that can supply a 12-, 24-, or 48-MHz clock. FX3 does not have the ability to provide a system clock to the external world. This system clock is different from the interface clock provided by GPIF or GPIF II.

Table 8 lists the details of the clock or crystal inputs that these two devices accept.

Table 8. Clock/Crystal Supported by FX2LP and FX3

	FX2LP	FX3
External clock	24 MHz	19.2, 26, 38.4, and 52 MHz
Crystal	24 MHz	19.2 MHz
CLKOUT	Available	Not available

Three Frequency Select pins FSLC[2:0] determine FX3 clocking. These pins should be tied to ground or CVDDQ through weak pull-up resistors (10 kΩ). Table 9 shows the values of FSLC[2:0] for different clocking options.

Table 9. Frequency Select Configuration

FSLC[2]	FSLC[1]	FSLC[0]	Crystal/Clock
0	0	0	19.2-MHz crystal
1	0	0	19.2-MHz input clock
1	0	1	26-MHz input clock
1	1	0	38.4-MHz input clock
1	1	1	52-MHz input clock

6 Software Differences

6.1 Development Tools

The FX2LP firmware framework used in the FX2LP Development Kit (DVK) uses the Keil µVision2 IDE and a firmware framework based on an event loop.

Firmware development for FX2LP devices can also be done using Eclipse IDE and SDCC compiler on Linux and Windows systems. Refer to Appendix. A Compiling FX2LP Project on Linux. Please note that Cypress uses Keil C51 C compiler for verifying all FX2LP examples and associated projects.

FX3 development tools include an Eclipse-based IDE and an RTOS called “ThreadX.” The IDE includes a compiler, linker, assembler, and JTAG debugger. You can download the free FX3 Software Development Kit (SDK).

6.2 USB Host-Side Applications

Control Center: Cypress includes a PC-based Control Center application in the FX3 SDK. You can use this application to program both FX2LP and FX3, either to download code into RAM or to program an external boot EEPROM.

Streamer: The streamer application for FX3 is similar to the one in FX2LP. Using this application, you measure the throughput numbers for the ISO and BULK streams.

Bulkloop: The FX3 SDK includes a bulkloop application to test the bulkloop example. The application gives you the option to send different types of data to run this bulkloop test.

You can find all these applications in the following path once you install the FX3 SDK in the default location:

C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\application

Table 10 lists the differences in the software resources available for FX2LP and FX3.

Table 10. Software Resources for FX2LP and FX3

Software Tools	FX2LP	FX3
Compiler	Keil C51 C Compiler	Arm® GCC C Compiler
Assembler	Keil A51 Assembler	Arm GCC Assembler
IDE	Keil	Eclipse-based IDE
Driver	*CyUSB3.sys*† or *CyUSB.sys*	*CyUSB3.sys*
Applications	Control Center	Control Center
Tool to develop GPIF interface	GPIF Designer	GPIF II Designer*

† *CyUSB3.sys* is recommended for new designs. Cypress recommends that you migrate existing designs to use *CyUSB3.sys*. Cypress no longer supports *CyUSB.sys*.

* FX3 GPIF II Designer is not compatible with FX2LP.

7 Programmer’s View of FX3

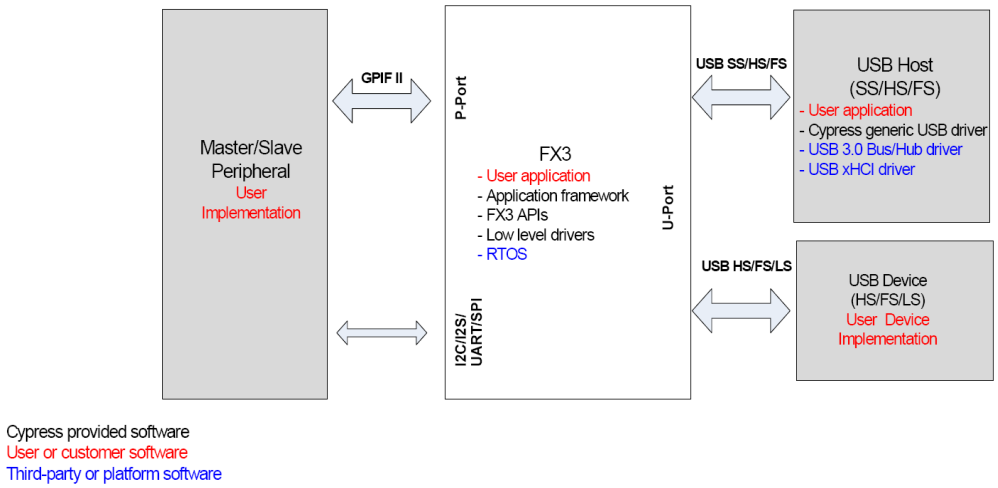
The FX3 comes with the easy-to-use EZ-USB tools, giving you a complete solution for fast application development (see Figure 3). Use the FX3 device to:

- Configure and manage USB functionality, such as charger detection, USB device/host detection, and endpoint configuration
- Connect to different master and slave peripherals on the GPIF interface
- Connect to serial peripherals (UART, SPI, GPIO, I^2C, I^2S)
- Set up, control, and monitor data flows between the peripherals (USB, GPIF, and serial peripherals)
- Perform necessary operations, such as data inspection, data modification, addition/deletion of packet header and footer information.

Two other important entities are external to the FX3:

- USB host/device
  - When the FX3 is connected to a USB host, it functions as a USB device. The FX3 enumerates as a SuperSpeed, high-speed, or full-speed USB peripheral corresponding to the host type.
  - When a USB device is connected, the FX3 plays the role of the corresponding high-speed, full-speed, or low-speed USB host.
- GPIF II master/slave
  - GPIF II is a fully configurable interface and can use any application-specific protocol. You can connect any processor, ASIC, DSP, or FPGA to the FX3. The FX3 bootloader or firmware configures GPIF II to support the corresponding interface.

Figure 3. Programming View of FX3



8 FX2LP and FX3 Firmware Framework Differences

8.1 FX2LP Firmware Framework

FX2LP code development uses a Cypress firmware framework that implements a control loop and low-level routines to handle USB events. Cypress firmware examples are framework based, so the best way to start is to use one of the examples as a reference and make the necessary modifications. These examples are available when you install the FX2LP DVK.

You can understand the differences in the firmware framework for FX2LP and FX3 by studying the bulkloop example.

8.2 Bulkloop Example on FX2LP

The bulkloop example is in the directory *Cypress\USB\Examples\FX2LP\Bulkloop* after you install the FX2LP DVK.

If you look at the FX2LP firmware framework, you can see the following files, which interact as shown in Figure 4.

- *fw.c:* This is the main framework source file. It contains `main()`, the task dispatcher, and the SETUP command handler. For most firmware projects, there is no need to modify this file. Four dispatcher functions are called in `main()`: `TD_Init()`, `TD_Poll()`, `TD_Suspend()`, and `TD_Resume()`.

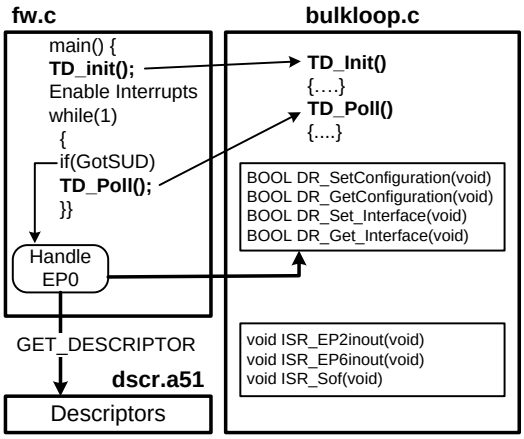
  `TD_Init()` is called once during the initialization of the framework. `TD_Poll()` is called repeatedly during device operation. `TD_Poll()` contains the logic that implements your peripheral function.
- *bulkloop.c:* This source file contains initialization and task dispatch function definitions that are called from *fw.c*. This is where you customize the frameworks for your specific device, in this case, bulkloop transfers.
- *dscr.a51:* This is the assembly file that contains your device's custom descriptors.
- *USBJmpTb.OBJ:* This object code contains the ISR jump table for USB and GPIF interrupts.

8.2.1 FX2LP Firmware API Library

EZUSB.LIB: The EZ-USB library is an 8051 *.LIB* file that implements functions that are common to many firmware projects. Typically, there is no reason to modify these functions, so they are provided in library form. However, the kit includes the library source code if you need to modify a function or if you simply want to know how something is done.

The `TD_Poll()` function in *bulkloop.c* implements the bulkloop application code. Bulkloop is a simple application that can be tested using the USB Control Center included in the FX2LP DVK. Using the USB Control Center, you download the bulkloop code (a *.hex* file) into FX2LP RAM. Refer to AN50963 for the details on different ways of downloading code into FX2LP. Then you can test the application using Control Center to send and receive BULK transfers over FX2LP BULK endpoints. Data sent to EP2-OUT loops back over EP6-IN, and data sent to EP4-OUT loops back over EP8-IN.

Figure 4. Structure of an FX2LP Application

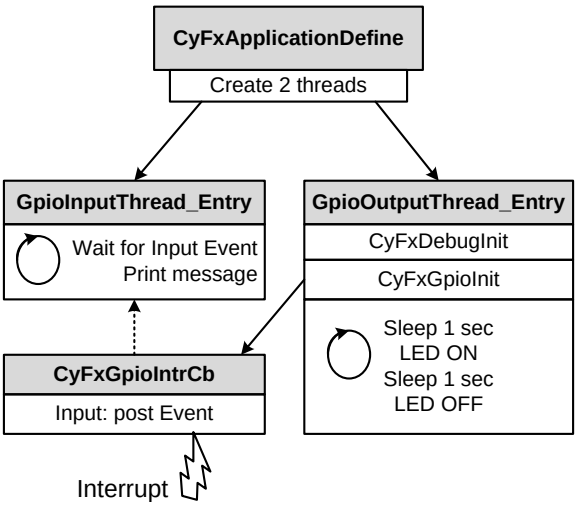


8.3 FX3 Firmware Framework

FX3 firmware uses a different approach than the single event loop of FX2LP firmware. Instead, the FX3 development tools use an RTOS, ThreadX, to launch and run concurrent application threads.

Before comparing the bulkloop example with the FX2LP implementation, a simple example that reads a push button and blinks an LED illustrates the basic structure of the FX3 application code. Figure 5 shows this structure.

Figure 5. Structure of an FX3 Application



This simple application is provided in the FX3 SuperSpeed Explorer Kit webpage under the title “First FX3 App.” The FX3 SuperSpeed Explorer Kit User Guide shows different ways to download and debug FX3 code:

- Download and run code
- Download and run code with serial debug
- Download and debug over JTAG

The `CyFxApplicationDefine` function creates two threads. Threads run concurrently on a time-shared basis as the application executes. This example creates two threads, one for output (the LED) and the other for input (the push button).

The Output thread initializes the serial debugger and creates an interrupt callback link to the function `CyFxGpioIntrCb`. Then it runs a continuous loop that blinks the LED. Timing is accomplished by telling the thread to sleep for a programmed number of milliseconds, 1000 ms in this example. Putting threads to sleep until work is a good way to save power and maximize CPU utilization.

Part of the GPIO initialization is to cause an interrupt on any change of the push button state. A state change calls the `CyFxGpioIntrCb` ISR. Serial debug messages cannot be printed from the GPIO callback as it runs in the interrupt context. All interrupts compete for CPU attention, and there may be more important tasks at hand. ThreadX handles this by providing a messaging system whereby the ISR posts a message for another noninterrupt function to execute.

The Input thread is the recipient of the message to print a debug message. It runs a continuous loop that checks for the message posted by the ISR and prints the debug message over the serial port.

You can build powerful and flexible applications using FX3 firmware examples and API libraries provided with the FX3 SDK installation. The firmware (or application) framework contains the startup and initialization code. The firmware library contains the individual drivers for the USB, GPIF II, and serial interface blocks. The framework does the following:

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

8.3.1 Firmware API Library

The FX3 API library provides a comprehensive set of APIs to control and communicate with the FX3 hardware. These APIs provide a complete programmatic view of the FX3 hardware.

8.3.2 *cyfxapi.a, cyu3lpp.a, cyu3threadx.a*

The FX3 SDK includes a full-fledged API library. This API library is similar to *EZUSB.LIB* in the FX2LP toolset. You need to manually link these libraries to your project.

The API library and the corresponding header files provide all the APIs required for programming the different blocks of FX3. The APIs provide for the following:

- Programming each of the individual blocks of the FX3 device: GPIF II, USB, and serial interfaces
- Programming the DMA engine and setting up data flows between these blocks
- The overall framework for application development, including system boot and initialization, OS entry, and application initialization.
- ThreadX OS calls as required by the application
- Power management features
- Logging capability for debug support

8.3.3 Embedded Real-Time OS

Because the FX3 firmware framework is based on an RTOS, the drivers for various peripheral blocks in the platform are typically implemented as separate threads. Standard OS services such as semaphores, message queues, mutexes, and timers are used for interthread communication and task synchronization and are available through the library.

The framework provides hooks for the application logic to configure the device behavior and to perform data transfers through it. The application logic can be implemented across multiple threads and use all the OS services that are used by the Cypress provided drivers.

The embedded RTOS in the FX3 device uses the ThreadX operating system from Express Logic. The application logic makes available for use all the functionality supported by the ThreadX OS. Some constraints on their use help ensure smooth functioning of all the drivers.

The ThreadX services are not directly exposed by the firmware framework. This is to ensure that the application logic is independent of the OS used and need not be changed to accommodate any future changes in the embedded OS. The OS services are made available through a set of platform-specific wrappers.

8.4 Bulkloop Example on FX3

This section revisits the bulkloop application to compare FX3 development with that of FX2LP.

When the FX3 SDK is downloaded and installed, the bulkloop example is located in the following directory:

C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\dma_examples\cyfxbulklpauto

The bulkloop example consists of these files:

- *cyfx_gcc_startup.S*: This file contains the FX3 startup code.
- *cyfxbulklpauto.h:* This file contains the defines used in *cyfxbulklpdscr.c.*
- *cyfxbulklpdscr.c*: This file contains the USB descriptors. It is similar to *dscr.a51* in the FX2LP toolset.
- *cyfxtx.c*: This file defines the porting required for the ThreadX RTOS. It is provided in source form and must be compiled with the application source code.
- *cyfxbulklpauto.c*: This file contains the main application logic of the bulkloop example.

8.4.1 Firmware Entry

The entry point for the FX3 firmware is the `CyU3PFirmwareEntry()` function. This function is defined in the FX3 API library and is not visible to the user. As part of the linker options, the entry point is specified as the `CyU3PFirmwareEntry()` function.

Note: Refer to *FX3APIGuide.pdf* located at *C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc* to learn the definition of FX3 APIs.

The firmware entry function performs these actions:

- Invalidates the caches (which were used by the bootloader)
- Initialize the memory management unit (MMU) and the caches
- Initializes the SYS, FIQ, IRQ, and SVC stack modes
- Transfers execution to the tool chain initialization (`CyU3PToolChainInit()`) function.

8.4.2 Tool Chain Initialization

The next step in the initialization sequence is a tool-specific initialization of the stacks and C library. Since all required stack initialization is performed by the firmware entry function, the tool chain initialization is overridden, so the stacks are not reinitialized.

The tool chain initialization function is written for the GNU GCC compiler for Arm processors. You can find this part of code in *cyfx_gcc_startup.S*. There is no need to modify this file.

In this function, only two actions are performed:

- The BSS area (a part of the data segment containing statically-allocated variables) is cleared.
- Control is transferred to the `main()`.

8.4.3 Device Initialization

The function `main()` is the C programming language entry for the FX3 firmware. This function performs the following consecutive actions.

1. Device initialization: This is the first step in the `main()`.

```
status = CyU3PDeviceInit (NULL);
```

As part of the device initialization, The CPU clock is set up. A NULL is passed as an argument for `CyU3PDeviceInit()` to select the default clock configuration.

2. Device cache configuration: The second step configures the device caches. FX3 has an 8-KB data cache and an 8-KB instruction cache. In this example instruction & data cache are enabled.

```
status = CyU3PDeviceCacheControl (CyTrue, CyTrue, CyTrue);
```

3. I/O matrix configuration: The third step configures the required I/Os. This includes the GPIF II and the serial interfaces (SPI, I²C, I²S, GPIO, and UART).

```
io_cfg.isDQ32Bit = CyFalse;
io_cfg.useUart   = CyTrue;
io_cfg.useI2C    = CyFalse;
io_cfg.useI2S    = CyFalse;
io_cfg.useSpi    = CyFalse;
io_cfg.lppMode = CY_U3P_IO_MATRIX_LPP_UART_ONLY;
/* No GPIOs are enabled. */
io_cfg.gpioSimpleEn[0]  = 0;
io_cfg.gpioSimpleEn[1]  = 0;
io_cfg.gpioComplexEn[0] = 0;
io_cfg.gpioComplexEn[1] = 0;
status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

In this bulkloop example:

a. GPIF II interface is not used

b. GPIO, I²C, I²S, and SPI interfaces are not used

c. Only the UART interface is used

The I/O matrix configuration data structure is initialized and the `CyU3PDeviceConfigureIOMatrix` function (in the library) is invoked.

4. The final step in the `main()` invokes the OS scheduler by issuing a call to the `CyU3PKernelEntry()` function. This function is defined in the library and is a nonreturning call. It is a wrapper to the actual ThreadX OS entry call. This function:

a. Initializes the OS

b. Sets up the OS timer used for scheduling

8.4.4 Application Definition

The FX3 library calls the function `CyFxApplicationDefine()` after the OS is invoked. In this function, you create application-specific threads. This function is similar to the `TD_Init()` function in FX2LP firmware, since it is called only once.

In the bulkloop example, only one thread is created in the application define function.

```
/* Allocate the memory for the threads */
    ptr = CyU3PMemAlloc (CY_FX_BULKLP_THREAD_STACK);

/* Create the thread for the application */
    retThrdCreate = CyU3PThreadCreate (&BulkLpAppThread,                    /* Bulk
loop App Thread structure */
                          "21:Bulk_loop_AUTO",
/* Thread ID and Thread name */
                          BulkLpAppThread_Entry,
/* Bulk loop App Thread Entry function */
                          0,                    /* No input parameter to thread */
                          ptr,                   /* Pointer to the allocated thread
stack */
                          CY_FX_BULKLP_THREAD_STACK,                        /* Bulk loop
App Thread stack size */
                          CY_FX_BULKLP_THREAD_PRIORITY,               /* Bulk loop App
Thread priority */
                          CY_FX_BULKLP_THREAD_PRIORITY,                  /* Bulk loop
App Thread priority */
                          CYU3P_NO_TIME_SLICE,                      /* No time slice for
the application thread */
CYU3P_AUTO_START                          /* Start the Thread immediately */
                          );
```

Note that more threads (as required by the user application) can be created in the application define function. All other FX3 specific programming must be done only in the user threads.

8.4.5 Application Code

In the bulkloop example, one Auto DMA channel is created after setting up the Producer (OUT) and Consumer (IN) endpoints. This DMA channel connects the two sockets of the USB port. Two endpoints, 1 IN and 1 OUT, are configured as bulk endpoints. The endpoint maxPacketSize is updated based on the speed.

```
CyU3PUSBSpeed_t usbSpeed = CyU3PUsbGetSpeed();

/* First identify the usb speed. Once that is identified, create a DMA channel and
start the transfer on this. */

/* Based on the Bus Speed configure the endpoint packet size */
switch (usbSpeed)
    {
        case CY_U3P_FULL_SPEED:
        size = 64;
        break;

        case CY_U3P_HIGH_SPEED:
        size = 512;
        break;

        case CY_U3P_SUPER_SPEED:
        size = 1024;
        break;
   }
CyU3PMemSet ((uint8_t *)&epCfg, 0, sizeof (epCfg));
epCfg.enable = CyTrue;
```

```
epCfg.epType = CY_U3P_USB_EP_BULK;
epCfg.burstLen = 1;
epCfg.streams = 0;
epCfg.pcktSize = size;

/* Producer endpoint configuration */
apiRetStatus = CyU3PSetEpConfig(CY_FX_EP_PRODUCER, &epCfg);

/* Consumer endpoint configuration */
apiRetStatus = CyU3PSetEpConfig(CY_FX_EP_CONSUMER, &epCfg);
/* Create a DMA Auto Channel between two sockets of the U port.
 * DMA size is set based on the USB speed. */
    dmaCfg.size = size;
    dmaCfg.count = CY_FX_BULKLP_DMA_BUF_COUNT;
    dmaCfg.prodSckId = CY_FX_EP_PRODUCER_SOCKET;
    dmaCfg.consSckId = CY_FX_EP_CONSUMER_SOCKET;
    dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
    dmaCfg.notification = 0;
    dmaCfg.cb = NULL;
    dmaCfg.prodHeader = 0;
    dmaCfg.prodFooter = 0;
    dmaCfg.consHeader = 0;
    dmaCfg.prodAvailCount = 0;
apiRetStatus = CyU3PDmaChannelCreate (&glChHandleBulkLp,
      CY_U3P_DMA_TYPE_AUTO, &dmaCfg);
```

8.4.6 Application Thread

The Application entry point for the bulkloop example is the `BulkLpAppThread_Entry ()` function. This function is similar to `TD_Poll ()` in the FX2LP firmware, where you write the application logic.

```
/* Entry function for the BulkLpAppThread. */
void
BulkLpAppThread_Entry (uint32_t input)
{
    /* Initialize the debug module */
    CyFxBulkLpApplnDebugInit();

    /* Initialize the bulk loop application */
    CyFxBulkLpApplnInit();

    for (;;)
    {
        CyU3PThreadSleep (1000);
    }
}
```

The main actions performed in this thread are the following:

1. Initialize the debug mechanism.
2. Initialize the main bulkloop application.

The following sections explain these steps.

8.4.7 Debug Initialization

The debug module uses the FX3 UART to output debug messages. The UART must be configured before the debug mechanism is initialized. This is done by invoking the UART init function.

```
/* Initialize the UART for printing debug messages */
apiRetStatus = CyU3PUartInit();
```

The next step is to configure the UART. The UART data structure is first filled in and then passed to the UART configuration function.

```
/* Set UART Configuration */
uartConfig.baudRate = CY_U3P_UART_BAUDRATE_115200;
uartConfig.stopBit = CY_U3P_UART_ONE_STOP_BIT;
uartConfig.parity = CY_U3P_UART_NO_PARITY;
uartConfig.txEnable = CyTrue;
uartConfig.rxEnable = CyFalse;
uartConfig.flowCtrl = CyFalse;
uartConfig.isDma = CyTrue;
apiRetStatus = CyU3PUartSetConfig (&uartConfig, NULL);
```

The UART transfer size is set to maximum so debug messages are not size-limited.

```
/* Set the UART transfer */
apiRetStatus = CyU3PUartTxSetBlockXfer (0xFFFFFFFF);
```

Finally, the debug module is initialized. The two main parameters are as follows:

- The destination for debug prints, which is the UART socket
- The verbosity of the debug that is set to level 8, so all debug prints that are below this level (0 to 7) will be printed

```
/* Initialize the Debug application */
apiRetStatus = CyU3PDebugInit
(CY_U3P_LPP_SOCKET_UART_CONS, 8);
```

8.4.8 Application Initialization

The application initialization consists of these steps.

USB Initialization

1. The USB stack in the FX3 library is initialized. The initialization is done by invoking the USB Start function.

```
/* Start the USB functionality */
apiRetStatus = CyU3PUsbStart();
```

2. The next step is to register for callbacks. In this example, callbacks are registered for USB setup requests and USB events.

```
CyU3PUsbRegisterSetupCallback(CyFxBulkLpApplnUSBSetupCB, CyTrue);

    /* Setup the callback to handle the USB events. */
    CyU3PUsbRegisterEventCallback(CyFxBulkLpApplnUSBEventCB);
```

The callback functions and the callback handling are described in the later sections: USB Setup Callback and USB Event Callback.

3. The USB descriptors are set by invoking the USB Set Descriptor call for each descriptor.

```
/* Set the USB Enumeration descriptors */
/* Device Descriptor */
apiRetStatus = CyU3PUsbSetDesc(CY_U3P_USB_SET_HS_DEVICE_DESCR, NULL,
(uint8_t *)CyFxUSB20DeviceDscr);
```

The previous code snippet is for setting the Device Descriptor. The other descriptors set in the example are Device Qualifier, Other Speed, Configuration, BOS (for SuperSpeed), and String Descriptors.

4. The FX3 USB pins are connected to the bus. The FX3 USB device is visible to the host only after calling the `CyU3PConnectState` API.

   Therefore, it is important to complete all USB setup before connecting the USB pins.

```
/* Connect the USB Pins */
/* Enable Super Speed operation */
apiRetStatus = CyU3PConnectState(CyTrue, CyTrue);
```

8.4.9 USB Setup Callback

USB Standard requests are handled by the firmware library, and the vendor- and class-specific requests need to be serviced by the application. On successful processing of control requests, this function shall return true otherwise return false.

```
/* Callback to handle the USB setup requests. */
CyBool_t
CyFxBulkLpApplnUSBSetupCB (
        uint32_t setupdat0, /* SETUP Data 0 */
        uint32_t setupdat1  /* SETUP Data 1 */
    )
{
/* Only class and vendor requests are received by this function. */
    return CyFalse;
}
```

8.4.10 USB Event Callback

The USB events of interest are Set Configuration, Reset, and Disconnect. The bulkloop application starts on receiving a SETCONF event and stops with a USB reset or USB disconnect.

```
/* This is the callback function to handle the USB events. */
void
CyFxBulkLpApplnUSBEventCB (
    CyU3PUsbEventType_t evtype, /* Event type */
    uint16_t            evdata  /* Event data */
    )
{
    switch (evtype)
    {
    case CY_U3P_USB_EVENT_SETCONF:
 /* Stop the application before re-starting. */
    if (glIsApplnActive)
       {
        CyFxBulkLpApplnStop ();
       }
  /* Start the loop back function. */
        CyFxBulkLpApplnStart ();
        break;

case CY_U3P_USB_EVENT_RESET:
case CY_U3P_USB_EVENT_DISCONNECT:
/* Stop the loop back function. */
if (glIsApplnActive)
 {
 CyFxBulkLpApplnStop ();
 }
break;
```

```
default:
break;
    }
}
```

DMA Setup

The DMA channel transfer is enabled using the following piece of code.

```
/* Set DMA Channel transfer size */
apiRetStatus = CyU3PDmaChannelSetXfer (&glChHandleBulkLp, CY_FX_BULKLP_DMA_TX_SIZE);
```

Refer to AN75705 to learn the steps for importing and building an FX3 firmware project.

You can download and test the code by loading the code image *USBBulkLoopAuto.img* into FX3 using the USB Control Center. Refer to AN75705 for the steps.

For more details on the FX3 SDK, see the documents available in the path *C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc*. (`1.3` in this path is the version of the SDK, which may change in future).

9 Slave FIFO Interfaces of FX2LP and FX3

This section explains the differences between the synchronous Slave FIFO interfaces in FX2LP and FX3.

The synchronous Slave FIFO interface is suitable for applications in which an external processor or device needs to perform data read/write accesses to FX2LP or FX3 internal FIFO buffers. Register accesses are not done over the Slave FIFO interface.

The following two application notes provide details on the FX2LP and FX3 Slave FIFO interfaces.

- AN61345 – Describes the FX2LP Synchronous Slave FIFO interface. A sample design demonstrates an FPGA interface.
- AN65974 – Describes the FX3 Synchronous Slave FIFO interface. A design example demonstrates an FPGA interface.

Table 11 lists the differences in synchronous Slave FIFO interface signals available for FX2LP and FX3.

Table 11. Synchronous Slave FIFO Interface Signals

Signal Name		Signal Description
FX2LP	FX3	
SLCS#	SLCS#	The chip select signal for the Slave FIFO interface, which needs to be asserted to perform any access to the Slave FIFO interface.
SLWR#	SLWR#	The write strobe for the Slave FIFO interface. It must be asserted for performing write transfers to the Slave FIFO.
SLRD#	SLRD#	The read strobe for the Slave FIFO interface. It must be asserted for performing read transfers from the Slave FIFO.
SLOE#	SLOE#	The output enable signal. It causes the data bus of the Slave FIFO interface to be driven by FX2LP or FX3. It must be asserted for performing read transfers from the Slave FIFO.
PKTEND#	PKTEND#	The PKTEND# signal is asserted to write a short packet or a zero-length packet to the Slave FIFO
FIFOADR[1:0]	A[1:0]	2-bit address lines to select one of the four (EP2, EP4, EP6, EP8) endpoints in FX2LP. For FX3, these two bit address lines are used to address four sockets.*
FD[15:0]	DQ[31:0]	Data bus of the Slave FIFO interface. Data bus width supported by FX2LP is 8-bit or 16-bit. Data bus width supported by FX3 is 8-bit or 16-bit, or 24-bit or 32-bit.
IFCLK	PCLK	The Slave FIFO interface clock. The maximum FX2LP frequency is 48 MHz. The maximum FX3 frequency is 100 MHz.

*Refer to AN75705 and AN65974 for more details on FX3 sockets.

9.1 Flag Usage

The external processor monitors the FLAG signals for flow control. Four flags (Flag A, Flag B, Flag C, Flag D) report the status of the FX2LP FIFOs. The FLAGA, FLAGB, and FLAGC pins can operate in either of two modes: Indexed (selected by pins) or Fixed, as selected by the PINFLAGSAB and PINFLAGSCD registers. The FLAGD pin operates in Fixed mode only. The FLAGA-FLAGC pins can be configured independently; Some pins can be in Fixed mode while others are in Indexed mode. Flag pins configured for Indexed mode report the status of the FIFO currently selected by the FIFOADR[1:0] pins. Refer to the “Slave FIFOs” chapter of the EZ-USB® Technical Reference Manual.

The FX3 Slave FIFO interface is more flexible than the FX2LP Slave FIFO interface. The FX3 Slave FIFO interface is developed by configuring GPIF II to act as a slave and with the help of a state diagram. Cypress provides a library of common GPIF II interfaces, including synchronous and asynchronous FIFOs. It can be customized by using a graphical entry tool called GPIF II Designer, if needed. In the standard implementation, two flags are configured to show empty/full/partial status for a dedicated thread or the current thread being addressed. Refer to AN65974 for more details. More flags can be added if needed.

10 UVC Camera Designs Based on FX2LP and FX3

This section compares FX2LP and FX3 designs for a UVC (USB video class) camera.

UVC is a USB standard class that allows a video streaming device to be connected to a USB host to stream video. A typical application is a webcam, which can be installed and used without a custom driver.

10.1 Image Sensor Interface

The various signals associated with image transfer are as follows. These are unidirectional signals from the image sensor to the FX3 interface.

- FV: Frame Valid (indicates the start and stop of a frame)
- LV: Line Valid (indicates start and stop of a line)
- PCLK: Pixel Clock (the data output of the image sensor is synchronized with the pixel clock)
- Data: 8- to 32-bit data lines of image data

Figure 6 shows a timing diagram of the FV, LV, PCLK and data signals. The FV signal is asserted to indicate the start of a frame. Then the image data is transferred line by line. The LV signal asserts during each line transfer while the image sensor provides data. Image sensors transfer with bus widths of 8 bits to 32 bits.

Figure 6. Image Sensor Interface Timing Diagram

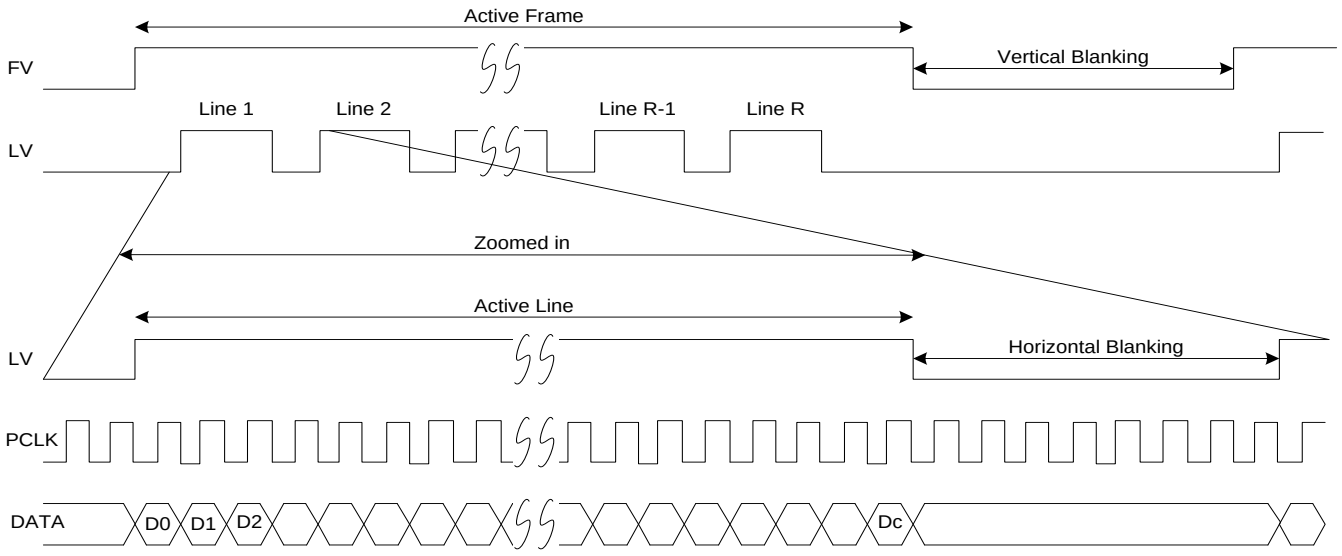


Figure 7 shows a system-level diagram of a USB camera.

Figure 7. USB Camera Block Diagram

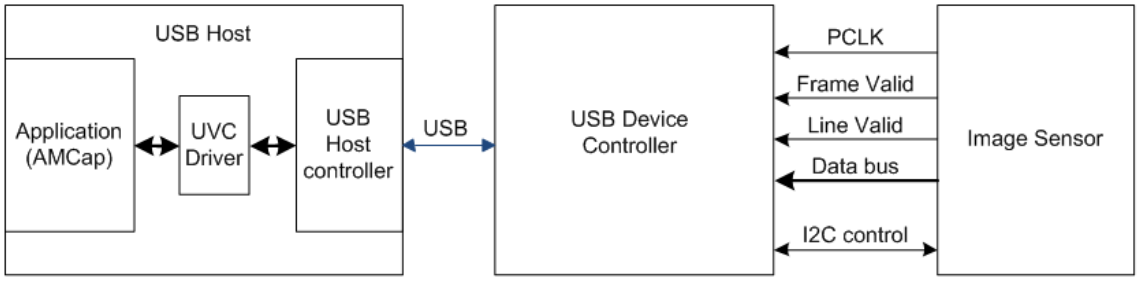


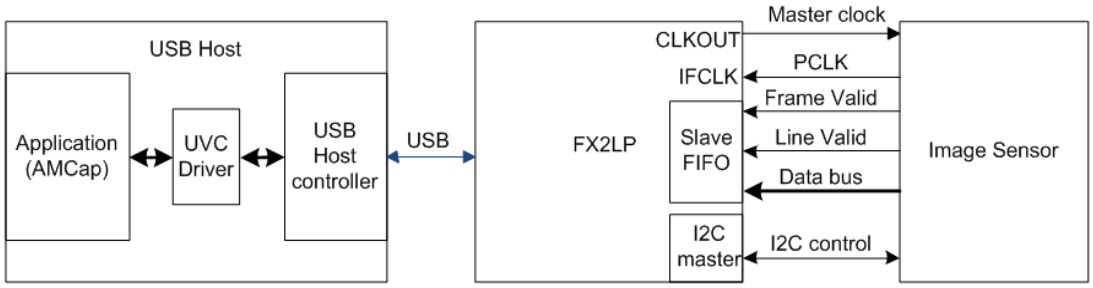
Image sensors typically use an I²C interface to allow a controller to configure the image sensor registers. The I²C block of FX2LP or FX3 can act as an I²C master to configure the image sensor with the correct parameters.

10.2 Implementation with FX2LP

An image sensor can connect to FX2LP using two interfaces: Slave FIFO or GPIF. The FX2LP FIFO operates as a slave, and its GPIF operates as a master. The simplest image sensor interface uses the FX2LP Slave FIFO.

The block diagram in Figure 8 shows how to connect an image sensor to FX2LP.

Figure 8. UVC Camera Design Using FX2LP



FX2LP provides a master clock (CLKOUT pin) to the image sensor that eliminates the requirement for an extra crystal if the image sensor can use a 12-, 24-, or 48-MHz clock.

FX2LP is configured in synchronous Slave FIFO mode. FX2LP supplies a 12-MHz clock to the image sensor. The image sensor is configured using the FX2LP I²C module. Once the image sensor is configured, it outputs image data using a 6-MHz Pixel Clock (PCLK). The Frame Valid signal connects to the FX2LP SLCS# pin, and the Line Valid signal connects to the FX2LP SLWR# pin of the Slave FIFO interface. The FX2LP firmware adds the required UVC header to each video frame.

For better performance, using an FPGA is suggested for designing a USB2.0 Camera Interface. See this Knowledge Base Article.

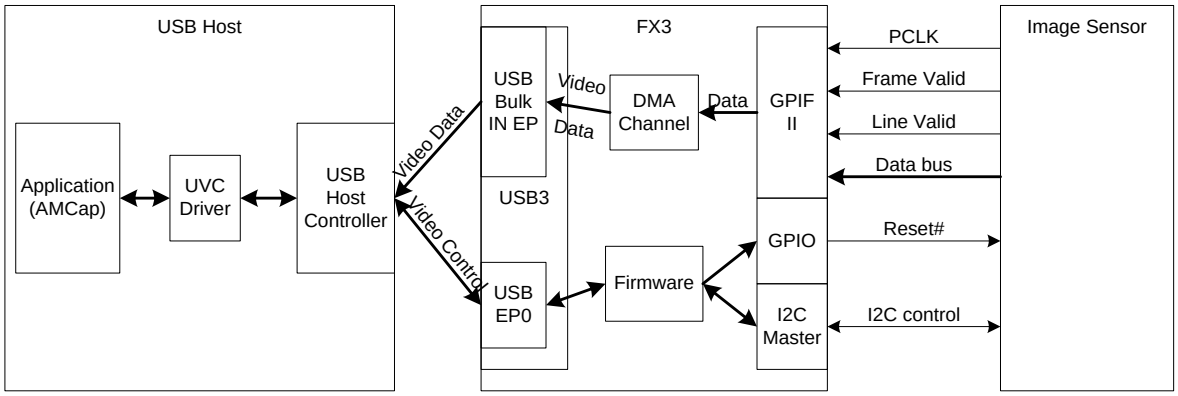
Using FX3 device can provide a better performance without an FPGA. See Implementation with FX3 for for details on USB Camera Interface using FX3 .

A standard host application such as AMCap or VLC Media Player communicates through the UVC driver to configure the image sensor over a video control interface and to receive video data over the video streaming interface.

10.3 Implementation with FX3

FX2LP contains separate interfaces for master or slave operation, whereas FX3 contains a unified GPIF II that can act as either master or slave. The image sensor interface is created using GPIF II Designer. This tool accepts state machine entry to create code that is included into an FX3 Eclipse project. Figure 9 shows the system level block diagram of UVC camera design using FX3.

Figure 9. UVC Camera Implementation Using FX3



The image sensor is configured using the FX3 I²C module. The FX3 DMA channel streams data from the image sensor to internal buffers, where the Arm MCU adds a UVC header to the image data. This video data is then sent to the video streaming endpoint. AN75779 includes more details on designing GPIF II state machine and application firmware. The FX3 design can also operate in USB 2.0 mode. The application detects the connection speed and reduces video bandwidth if operating at high speed.

Even though FX3 is a SuperSpeed device, there are advantages to using an FX3 design for USB 2.0 high-speed operation:

- The image sensor interface can operate with 24-bit or 32-bit image sensors. FX2LP maximum bus width is 16 bits.
- The UVC header can be added to every frame of data more efficiently using the ARM processor.
- FX3 can also act as a SPI master if the image sensor needs to be configured over the SPI interface.
- The FX2LP GPIF interface clock is limited to 48 MHz, whereas the GPIF II clock runs up to 100 MHz.
- The FX2LP maximum endpoint memory is 4 KB, while FX3 can use 512-KB program memory for endpoints.

10.4 Use of an I²C Module in FX2LP and FX3

Image sensor registers are configured through the I²C interface. FX2LP and FX3 both have an I²C module, which can act as master. Standard API functions are provided to perform read and write operations over the I²C interface.

The `EZUSB_WriteI2C()` and `EZUSB_ReadI2C()` functions are used to write and read image sensor registers using the FX2LP I²C module. These two functions are part of *EZUSB.LIB*.

The `CyU3PI2cTransmitBytes()` and `CyU3PI2cReceiveBytes()` functions are used to write and read image sensor registers using the FX3 I²C module. These two functions are part of the *cyu3lpp.a* library. Refer to the project attached to the application note AN75779 for more details.

10.5 Debug FX2LP and FX3 Firmware Using UART

Serial port debugging makes it possible to print debug messages and the real-time values of variables to a standard terminal program such as TeraTerm or HyperTerminal. In the UVC camera application, register values can be verified by reading them back over the I²C interface and printing them on a terminal program.

AN58009 discusses the code to add debugging to an FX2LP firmware project. Com port settings are 38400 baud, no parity, one stop bit (38400, N, 8, 1). Refer to AN58009 for more details.

To enable this debug feature on FX3, initialize and configure the UART as described in the Debug Initialization section. The com port settings needed are 115200 baud, no parity, one stop bit (115200, N, 8, 1). Refer to the project attached to the application note AN75779 for more details.

11 Available Collateral

11.1.1 FX2LP Development Kit

- CY3684 EZ-USB FX2LP Development Kit

11.1.2 FX3 Development Kit

- CYUSB3KIT-003 EZ-USB® FX3™ SuperSpeed Explorer Kit
- CYUSB3KIT-001 EZ-USB® FX3™ Development Kit

FX2LP Datasheet

- CY7C68013A, CY7C68014A, CY7C68015A, CY7C68016A: EZ-USB® FX2LP™ USB Microcontroller High-Speed USB Peripheral Controller

11.1.3 FX3 Datasheet

- CYUSB301X

11.1.4 FX3 SDK

- EZ-USB FX3 Software Development Kit

11.1.5 FX2LP GPIF Designer

- GPIF Designer

11.1.6 FX3 GPIF II Designer

- GPIF™ II Designer

11.1.7 Application Notes

- AN75705 – Getting Started with EZ-USB® FX3™
- AN65209 – Getting Started with FX2LP™
- AN68829 – Slave FIFO Interface for EZ-USB® FX3™: 5-Bit Address Mode
- AN65974 – Designing with the EZ-USB® FX3 Slave FIFO Interface
- AN63787– EZ-USB® FX2LP™ GPIF and Slave FIFO Configuration Examples Using an 8-Bit Asynchronous Interface
- AN70707 – EZ-USB® FX3™/FX3S™ Hardware Design Guidelines and Schematic Checklist
- AN15456 – Guide to Successful EZ-USB® FX2LP™ Hardware Design
- AN76405 – EZ-USB® FX3 Boot Options
- AN50963 – EZ-USB® FX1™/FX2LP™ Boot Options
- AN75779 – How to Implement an Image Sensor with EZ-USB® FX3™ in a USB Video Class (UVC) Framework
- Visit www.cypress.com to download the latest version of the product collateral

12 About the Author

Name:	Rama Sai Krishna V
Title:	Applications Engineer Staff
Background:	Rama Sai Krishna holds an M.Tech in Systems and Control Engg. from IIT Bombay. He is currently working on Cypress USB peripherals.

Appendix A. Compiling FX2LP Project on Linux

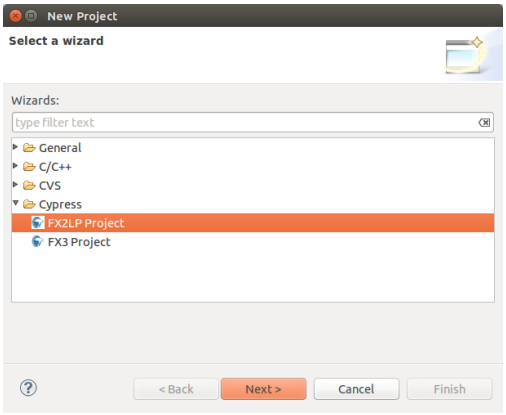
Note: Creating a New FX2LP project is not in the scope of this section.

Binary for FX2LP devices can be generated, using Eclipse IDE & SDCC (Small Device C Compiler) on Linux Host Machines. Follow the below instructions for setting the build environment & using generated FX2LP binary. The versions referred in below are based on FX3 SDK v1.3.3 for Linux.

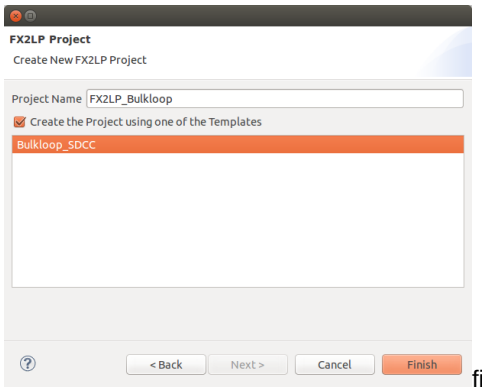
1. Download EZ-USB FX3 SDK for Linux.
2. Download the FX3_SDK_Linux_Support.pdf. Follow the steps provided in the SDK Installation.
3. Open a Linux terminal and run the below command to install latest Small Device C Compiler (SDCC).

   ```
   apt-get install SDCC
   ```

4. Now, run the ezUsbSuite application which is available in the eclipse folder and choose a workspace folder for the EZ USB Suite application.
5. From Eclipse IDE, select **File** > **New** > **Project**. Choose the FX2LP project in the Cypress folder and click **Next**.



6. Provide a project name and the template *Bulkloop_SDCC* and click **Finish**.



fi

The Bulk loop example for FX2LP device will be shown in the Eclipse IDE.

7. Select **Project** > **Build Project**.
8. The IDE uses SDCC for compiling and generating the hex file. After the build is finished the Release folder inside the chosen workspace folder shall contain a hex file with the given project name. Refer to the EZ-USB Development Kit for information on preparing the DVK Kit.
9. Refer to the *cyusb_linux_user_guide.pdf* available in the *cyusb_linux_1.0.4* folder for information about how to use the cyusb_linux Host application for downloading the firmware and running it.

Document History

Document Title: AN76348 - Differences in Implementation of EZ-USB® FX2LP™ and EZ-USB FX3™ Applications

Document Number: 001-76348

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3565979	RSKV	03/30/2012	New application note
*A	3943464	RSKV	03/26/2013	Updated Document Title to read as “Differences in Implementation of EZ-USB® FX2LP™ and EZ-USB FX3 Applications - AN76348” Updated Architectural Differences (Added Figure 1 and Figure 2, updated Table 1) Updated GPIF Versus GPIF II (Added Table 3) Updated Hardware Differences (Added references to hardware guidelines application notes, added reference to FX3 and FX2LP boot options application notes, updated Crystal/Clock (Updated Table 8.), updated Power Supply Configurations and Decoupling Capacitance Added Table 5. , updated Table 6. Updated Software Differences (Updated USB Host-Side Applications (Added Table 10). Added Slave FIFO Interfaces of FX2LP and FX3 Added UVC Camera Designs Based on FX2LP and FX3
*B	4383784	RSKV	05/19/2014	Updated the “FX3 Firmware Framework” section Modified the abstract Restructured the “Hardware Differences” section Updated the paths related to the FX3 SDK
*C	4651404	GAYA	02/05/2015	Updated Table 8 and Table 10 Updated list of FX3 Development Kit under Available Collateral Updated template Sunset review
*D	5699825	AESATP12	04/20/2017	Updated logo and copyright.
*E	6176678	SUDH	05/16/2018	Updated Development Tools Updated FX3 Firmware Framework Updated Device Initialization Updated USB Setup Callback Updated Flag Usage Updated Implementation with FX2LP Added New Section Appendix. A Compiling FX2LP Project on Linux

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

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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