AN4053

Streaming Data Through Isochronous or Bulk Endpoints on EZ-USB® FX2™ and FX2LP™

Author: Rama Sai Krishna Vakkantula

Associated Project: Yes

Associated Part Family: CY7C68013A

Software Version: NA

Related Application Notes: AN65209, AN61345

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN4053.

USB applications that use data types, such as audio and video, require a flow of continuous high-speed data. This data is termed as streaming because it flows in an uninterrupted stream. This application note discusses USB high-bandwidth delivery mechanisms that support streaming data, and includes code to implement and exercise FX2LP high-bandwidth endpoints. A companion PC application is provided to select various transfer types and to measure transfer rates. The example code attached to this application note demonstrates how to use the Cypress USB Frameworks to implement alternate USB settings, enabling the host to select different transfer rates. For the complete list of USB Hi-Speed code examples, visit http://www.cypress.com/?rID=101782.

Contents

1 Introduction....1
2 Associated Project Files....2
3 Streaming Applications....2
4 High-Bandwidth Transfers....3
4.1 Data PID Sequencing (HS Operation)....4
5 Cypress SuiteUSB....5
6 Streaming Firmware Example....5
7 Using the Cypress USB Frameworks....6
7.1 fw.c....7
7.2 dscr.a51....7
7.3 CyStream.c....9
8 Running CyStreamer.hex....11
9 Performance Analysis....12
10 Interfacing FX2LP with Image Sensor....14
11 References....14
12 Summary....14

1 Introduction

USB provides for streaming data with a transfer type called “Isochronous” (ISO). In Latin, *iso* means equal, and *chron* means time: Equal time is given in every USB time frame for data packets that contain streaming data.

USB ISO transfers have two important characteristics:

1. Unlike CONTROL, BULK, or INTERRUPT transfers, ISO transfers have no handshakes or re-tries for damaged or missing packets. Such overhead interferes with the central ISO feature of guaranteed data delivery in every time frame.
2. Although the USB ISO transfer mechanism provides the data buckets on a guaranteed basis, it is the responsibility of the data provider and consumer to fill and empty the buckets quickly to ensure an uninterrupted stream of data.

Although damaged or missing ISO packets are not re-sent, the consuming end can *detect* their occurrence and take corrective action. For example, a missing data chunk can be replaced by interpolation or repeating previous data. However, a well-designed streaming application should ensure the both the data provider (for example, a camera) and consumer (for example, a PC) give the streaming data high priority so that it does not miss loading or unloading a packet.

INTERRUPT Endpoints can also be Streamers.

The USB Spec, version 2.0, defines a high-bandwidth endpoint for high-speed (480 Mbits/sec) operation only. A high-bandwidth endpoint can provide or accept more than one packet per microframe, and a maximum of up to three packets. Therefore, INTERRUPT transfers can also be used for streaming data in a high-speed device. In full-speed devices, INTERRUPT transfers are not well-suited for high-bandwidth streaming apps because the INTERRUPT endpoint's maximum packet size is 64 bytes, compared to the ISO's maximum packet size of 1023 bytes.

This application note shows how to program FX2LP for high-speed USB ISO streaming data. It also implements full-speed transfer examples for comparison. The earlier FX2 (not LP) device can also be used with a minor difference (explained in this document). For bandwidth comparison purposes, a set of BULK transfers is also implemented.

2 Associated Project Files

The zip file accompanying this application note (*AN4053.zip*) includes the following folders:

1. FX2LP Streamer source code

The Keil project file CYStream.Uv2 is located in the `Firmware\CyStreamer` folder. Double-click this to launch the Keil project that implements the FX2LP firmware to provide and consume continuous streaming data. The output of the Keil IDE is the *CyStream.hex* file, suitable for loading into the FX2LP Development Board.

2. VS_Control_Center

This Microsoft Visual Studio C# application is used to download the FX2LP hex code (*CyStream.hex*) into an FX2LP Development Board. The executable is in the `\bin` directory, or you can load the *.sln* (solution) file with Microsoft Visual Studio to inspect or modify the source code.

3. VS_Streamer

This Microsoft Visual Studio C++ project is a host-side application that streams data from the PC to the FX2LP Development Board, running the CyStream code. This app is used to test the various bandwidth settings in the FX2LP streaming firmware. The executable is in the `\bin` directory, or you can load the *.sln* file to inspect or modify the source code.

You get this application along with the Cypress SuiteUSB installation. This application is located at: `Cypress Suite USB 3.4.7\CyAPI\examples\Streamer`

You also get the C# based streamer application along with the Cypress SuiteUSB installation. The C# version is located in the following path: `Cypress Suite USB 3.4.7\CyUSB.NET\examples\Streamer`

The following sections give more detail on ISO streaming, along with information required to implement ISO data streams using FX2LP.

3 Streaming Applications

When you plug a USB device into a PC, the PC uses a process called enumeration to learn the nature of the device and its requirements. During enumeration, the host interrogates the device to determine details, such as which device driver to use and the number and nature of endpoints (data sources and sinks) included in the device. For ISO endpoints, the host reads descriptors to determine what the device expects during ISO transfers. A key ISO parameter is the required bandwidth, expressed in the *wMaxPacketSize* field of the ISO endpoint descriptor. The ISO bandwidth limits are as follows:

- For full-speed transfers, the host can accommodate one ISO packet per 1-millisecond frame, with a packet size up to 1023 bytes. 1023 bytes every millisecond gives a maximum transfer rate of 1.023 megabytes per second.
- For high-speed transfers the host can accommodate up to three ISO packets per 125 -microsecond frame, with a packet size up to 1024 bytes. 3072 bytes every 1/8 millisecond gives a maximum transfer rate of 24.576 megabytes per second.

Data is streamed without interruption only if the two sides, provider and consumer, can move the data in and out of endpoint FIFOs quickly to achieve the desired transfer rate. This need for speed is the reason FX2/FX2LP contains a hardware data mover called the General Programmable Interface (GPIF).

USB provides a mechanism called Alternate Settings to let the host choose between multiple streaming bandwidths. For example, a USB camera might provide the Alternate Settings in Table 1.

Table 1. Three Bandwidth Settings for a USB Camera

Alternate Setting	wMaxPacketSize
1	1023
2	512
3	256

Because this USB camera may not be the only device on the bus, the PC must allocate its finite USB bandwidth between all connected devices.

Consider that during this bandwidth negotiation process, the host determines that it can give the USB camera only 700 bytes per frame on a guaranteed basis. If the camera ISO endpoint descriptor contains only one setting of 1023 bytes, the host will not configure (use) the camera and the application fails. However, because the camera provides other choices, the host can select Alternate Setting #2. It sends a message to the camera, (using SET_INTERFACE) containing the alternate setting it requires. This tells the camera firmware to reduce its bandwidth requirement, perhaps by lowering the frame rate.

The ISO bandwidth alternate settings begin with 1 because the default setting, 0, must not consume any ISO bandwidth. These packet sizes are maxima; the device may transfer a smaller number of bytes in any packet. A zero-length ISO packet also has a meaning. It is sent by the device to indicate it was unable to supply data to fill the 'just-in-time' ISO IN request.

The peripheral designer may choose to trade larger packet sizes for scheduling efficiency for example, one 1024-byte packet instead of four 256-byte packets. This decision usually depends on the number and size of data buffers in the peripheral. To accommodate this flexibility, USB defines a *bInterval* field for a high-bandwidth endpoint. This number tells the host how many microframes to skip between transfer requests. For most high-bandwidth applications, including the one described in this application note, *bInterval*s are set to 1, indicating 'schedule a transfer in every microframe'.

Although BULK endpoints are not considered high-bandwidth endpoints, they can also be used to stream data if enough buffering is provided on both sides to handle the bursty (unscheduled) nature of BULK endpoints. This is not recommended because your camera might work on a lightly loaded USB system, but when more devices such as USB disk drives are plugged in, the BULK transfers to your camera may slow down.

A USB streaming app is designed by considering the following points:

- Transfer throughput: Number of bytes per second required
- Is FX2LP fast enough to move the data using programmed (MCU-involved) transfers, or should the GPIF be used to transfer FIFO data?
- Buffering requirements of the system: ISO endpoints are, by specification, double-buffered, but ICs, such as FX2LP, provide up to four buffers if required.
- Suitable endpoint type to meet the required bandwidth and data rates
- Number of packets required per microframe
- PC driver support for the selected high-bandwidth transfer

4 High-Bandwidth Transfers

USB divides bus time into fixed-length segments called frames. For full-speed devices, the host issues a frame marker (SOF, Start of Frame) every millisecond. For high-speed devices, the host also issues a microframe every 125 microseconds.

A high-speed endpoint that requires more than 1024 bytes per microframe is called a high-bandwidth endpoint. A high-bandwidth endpoint can transfer up to three packets per microframe, each packet containing a maximum of 1024 bytes. The number of transfers per microframe is defined in the *wMaxPacketSize* field of the endpoint descriptor. Because they use microframes, high-bandwidth endpoints are possible only for high-speed operations.

According to the USB 2.0 specification, a periodic endpoint (ISO or Interrupt) must specify its required bus access period. This is done by setting the *bInterval* field of the endpoint descriptor. The *bInterval* field of the endpoint descriptor defines the maximum rate at which the endpoint is polled by the host. This provides a mechanism for slowing down the rate at which the host will service the endpoints.

To keep track of the data packet transfer during the same microframe, high-bandwidth isochronous transfers use a mechanism called the Packet ID (PID) sequencing. To design high-bandwidth I*SO* endpoints, it is important to understand the data PID sequencing, as described in the following section.

4.1 Data PID Sequencing (HS Operation)

Full-speed USB devices use two data PIDs, DATA0 and DATA1. The USB 2.0 specification adds two PIDs for high-speed operation, MDATA and DATA2. PID sequencing detects lost or damaged packets in a microframe. Table 2 shows the PID sequence ordering for IN and OUT ISO transfers of different size payloads.

Table 2. High-Bandwidth ISO Transfer PID Sequences

Available Packets	Direction	Data PID Sequence		
		Pkt 1	Pkt 2	Pkt 3
3	IN	DATA2	DATA1	DATA0
2	IN	DATA1	DATA0	–
1	IN	DATA0	–	–
3	OUT	MDATA	MDATA	DATA2
2	OUT	MDATA	DATA1	–
1	OUT	DATA0	–	–

For IN transfers, the host expects to receive up to the number of packets per microframe reported by the device in its *wMaxPacketSize* field during enumeration. The first DATA PID sent by the device tells the host how many more IN tokens it should issue in the microframe: DATA2 means two more, DATA1 means one more, and DATA0 means no more. If the device has no data, it sends a zero-length packet (ZLP) using the DATA0 PID.

For OUT transfers, the host uses the MDATA (More Data) PID for all packets except the final one, which is a DATA PID indicating how many MDATA PIDs the host sent.

4.1.1 FX2 (not LP) ISO IN Transfer Consideration

During enumeration, a high-speed ISO device informs the host about how many additional packets in a microframe it expects to use. The device indicates this in an endpoint descriptor field called *wMaxPacketSize*. Bits 12:11 specify none, 1 additional, or 2 additional packets per microframe. These are called transaction opportunities in the USB 2.0 Specification because they represent a maximum number of packets, which the device is not required to use in any microframe.

FX2 is responsible for sending the correct DATA PIDs, depending on how many actual IN FIFO buffers it filled and committed to USB transfer by the time the first host IN token arrives in a microframe. The code example reports three IN packets (max) possible in a microframe in its *wMaxPacketSize* field. However, if it has only committed two packets in a microframe, it must send the first packet using the DATA1 PID (Table 2).

To automatically accomplish the correct ISO-IN PID sequencing, FX2LP provides four registers called EPnISOINPKTS where n = 2, 4, 6, or 8, corresponding to its large, high-bandwidth endpoints.

b7	b6	b5	b4	b3	b2	b1	b0
AADJ	0	0	0	0	0	INPPF1	INPPF0
0	0	0	0	0	0	0	1

The IN Packets per Frame (INPPF) defaults to 1, but may be set to 1, 2 or 3. The Auto Adjust (AADJ) should be set to 1 to enable the automatic PID sequencing. If AADJ = 0, FX2LP unconditionally starts an ISO IN transfer using the PID indicated by INPFF[1:0].

Because the earlier FX2 device does not have this automatic mechanism (no AADJ bit), the programmer must take steps to ensure that the correct Data PID sequence is sent during the ISO IN transfers. Follow any of these methods to do this:

1. Ensure, by design, that all IN FIFOs are loaded and armed in time for every microframe. If the INPPF field is set to 2, for example, ensure that two buffers are loaded and committed to USB for every transfer.
2. If it is known that the data is short for the next microframe, enable any of the non-used FIFOs to send a ZLP. This is simple and quick; just load the FIFO's byte count register with the value zero.
3. If an outside source loads USB FIFOs, any unused IN FIFO can be armed to send a ZLP by selecting the FIFO with FIFOADR[1:0] and asserting the PKTEND pin. PKTEND is used to commit a partially filled (or empty) FIFO for USB transmission.

The host driver may or may not be able to gracefully handle a USB PID sequence error. Cypress provides a host driver that handles the situation where the device sends an incorrect PID sequence by propagating this indication up to the client application. This should never trigger using FX2LP because it automatically sends the correct Data PID sequences when AADJ = 1.

No registers need to be set for transfers through isochronous OUT endpoints. The device firmware must only report the number of packets desired per microframe in the endpoint descriptor.

5 Cypress SuiteUSB

The folders necessary to write and test code for this application note are included in the accompanying zip file. For more extensive support, Cypress provides the Cypress SuiteUSB, a set of USB development tools for Visual Studio plus example Keil firmware projects. Cypress SuiteUSB can be used to create .NET Windows applications for all Cypress USB 2.0 families. The suite includes *cyusb.sys*, a Cypress Windows driver compatible with the development tools.

Cypress SuiteUSB can be downloaded from http://www.cypress.com/?rID=34870. The default installation path for Cypress SuiteUSB is `C:\Cypress\Cypress Suite USB 3.4.7`.

6 Streaming Firmware Example

The application included with this note, CyStream, demonstrates USB isochronous streaming and BULK transfers using an EZ-USB FX2/FX2LP development board. This code example implements a test device that provides and consumes data streams for the Windows Streamer application.

The CyStream project creates a single interface (Number 0) with two different configuration descriptors, one for high-speed (HS) operation and the other for full-speed (FS) operation. Table 3 shows the seven HS alternate settings and Table 4 shows the four FS alternate settings.

Table 3. FX2LP HS Configuration Descriptor Alternate Settings for Interface 0

Alternate Setting	Number of Endpoints	Endpoint Number Direction (Type)	Maximum Packet Size (Bytes)
0	1	2 IN (Bulk)	512
1	1	2 OUT (Bulk)	512
2	2	2 IN (Bulk)	512
		6 OUT (Bulk)	512
3	1	2 IN (Isochronous)	3x1024
4	1	2 OUT (Isochronous)	3x1024
5	1	2 IN (Isochronous)	1x1024
6	2	2 IN (Isochronous)	1x1024
		6 OUT (Isochronous)	1x1024

Table 4. FX2LP FS Configuration Descriptor Alternate Settings for Interface 0

Alternate Setting	Number of Endpoints	Endpoint Number Direction (Type)	Maximum Packet Size (Bytes)
0	1	2 IN (Bulk)	64
1	1	2 OUT (Bulk)	64
2	1	2 IN (Isochronous)	1023
3	1	2 OUT (Isochronous)	1023

The host app (Streamer.exe) can select different transfer parameters (alternate settings) to measure their throughput rates. To measure maximum achievable throughput rates, it is important not to introduce any delay on the FX2LP side, such as delays incurred while transferring bytes in and out of endpoint FIFOs. For this reason, the FX2LP code in this application note handles IN and OUT transfers with minimal overhead:

- IN Transfers — when an IN FIFO becomes available, the 8051 simply re-arms the IN FIFOs for 1024-byte HS transfers or 1023-byte FS transfers. The FIFO data loaded during initialization is re-sent every time.
- OUT Transfers — the 8051 does not transfer any bytes out of the OUT FIFO; it only re-arms the next OUT transfer by loading an arbitrary byte count into the Byte Count Low (EPxBCL) register.

Note Because real applications *do* manage FIFO data, FX2LP provides two hardware mechanisms for minimizing its transfer overhead.

1. Data transfer is handled by an external controller with direct hardware connections to the FX2LP endpoint FIFOs. This keeps the 8051 out of the data path. The FIFO connections include an 8-bit or 16-bit data bus, read and write strobes, and FIFO status flags.
2. FX2LP FIFOs can be configured to operate in an automatic mode, by which packets are automatically armed for USB transfer after the number of bytes in an endpoint FIFO matches the level set in a register.

The example presented in this application note uses the manual mode setting of the endpoint FIFO, where the 8051 is responsible for checking FIFO status and committing endpoint data.

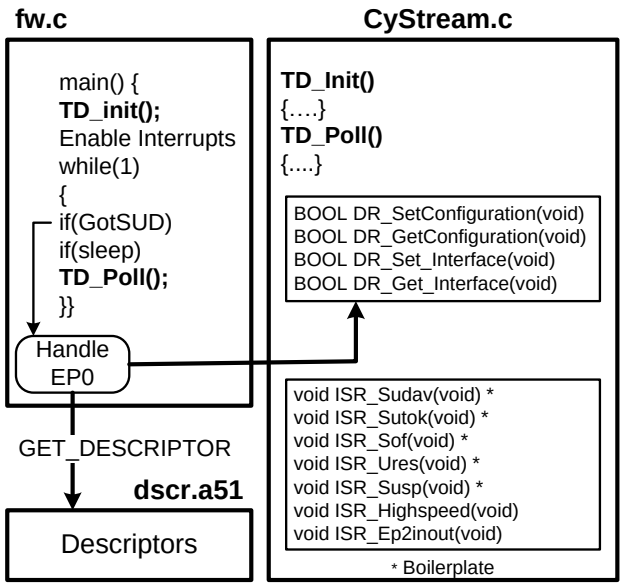
7 Using the Cypress USB Frameworks

The code example included with this application note demonstrates how to set up FX2LP endpoints for streaming data. This section covers the major routines of the code in detail. The example includes the following source files:

1. Cypress Frameworks file, *fw.c*
2. Descriptor file, *dscr.a51*
3. CyStream.c file

These code modules fit together, as shown in the following figure. This organization applies to any FX2LP USB application based on the Cypress firmware Frameworks.

Figure 1. CyStream Code Modules



7.1 fw.c

This Cypress-written firmware, Frameworks, handles low-level USB details. You do not need to modify this file. *Fw.c* contains the project's main() function, which calls **TD_Init()** once at startup, and then repeatedly calls **TD_Poll()** during operation. This endless loop also handles the CONTROL endpoint (EP0) SETUP packets. For GET_DESCRIPTOR requests, it uses the descriptor data you supply in dscr.a51. For other host requests, it calls into your app to handle various actions, such as changing an interface's alternate setting. The endless loop also handles USB suspend and resume events (sleep).

7.2 dscr.a51

This is an 8051 assembly language module containing descriptor data for your particular USB device. This file contains .db (define byte) statements to list the descriptor table data, in the following order:

1. Device Descriptor
   a. USB Spec Version (2.0)
   b. Device Standard Class (none)
   c. EP0 MaxPacketSize (64)
   d. Vendor ID = 0x04B4 = Cypress
   e. Product ID = 0x1003 = "EZ-USB Example Streamer Device".
   f. Manufacturer and Product String Indices
   g. Number of configurations = 1
2. Device Qualifier: For a device that can operate at either full- or high-speed, this gives settings for the 'other' (non-current) speed.
3. HS Configuration
   a. Number of Interfaces = 1.
   b. Index to select this Configuration = 1
   c. Seven Interface-Endpoint Descriptor pairs that lay out the seven alternate setting parameters in Table 3.
4. FS Configuration

a. Number of Interfaces = 1.

b. Index to select this Configuration = 1

c. Four Interface-Endpoint Descriptor pairs that lay out the four alternate setting parameters in Figure 4.

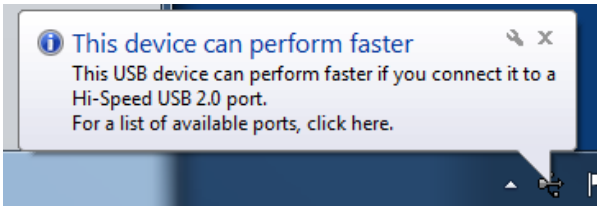
5. Two string descriptors.

7.2.1 About FS/HS Operation

When a USB peripheral attaches to a host, the host's internal USB hub (or external hub, if there is one) conducts a negotiation with the peripheral to determine its operating speed. The host never actually *selects* an operating speed. Instead, it uses the speed reported by the hub. As long as the device remains plugged in, the hub-reported operating speed never changes. Only device removal and re-attachment starts the speed discovery process again.

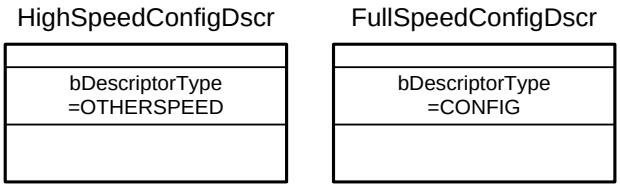
For a device that can operate at either FS or HS, the speed reported by the hub is the 'current' speed, and the speed at which it *can* operate is the other speed. Having two Configuration Descriptors, one for each operating speed, allows the host to determine that the device supports another speed if the device is plugged into a different port. This allows Windows to put up the message, shown in Figure 2, if you plug a HS device into a FS port.

Figure 2. Windows Detects a HS Device Plugged into a FS Port



It is the responsibility of the FX2LP firmware to detect the current operating speed and designate the proper Configuration Descriptor (FS or HS) as the one currently in use and the other one as corresponding to the non-selected (other) speed. The firmware does this by inserting the correct *bDescriptorType* value, CONFIG, or OTHERSPEED, into the second byte of each Configuration Descriptor.

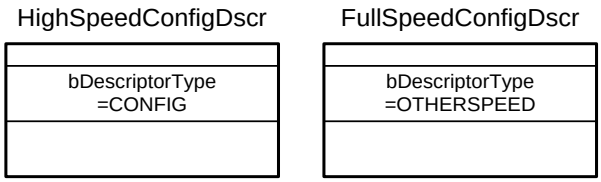
Figure 3. Descriptor Types after RESET



Following a USB bus RESET, the device reverts to FS operation. Therefore, in the USB Reset Interrupt Service Routine, the firmware marks the descriptors (Figure 3). It also sets pointers to each descriptor (*pConfigDscr* and *pOtherConfigDscr*) for use during enumeration. If the device is plugged into a full-speed port, this designation remains.

The FX2LP receives an interrupt (ISR_Highspeed) whenever the USB core successfully negotiates for high-speed operation. In this ISR, the firmware swaps designations, as shown in Figure 4, and sets the descriptor pointers accordingly.

Figure 4. Descriptor Types after HS Detected



7.2.2 About wMaxPacketSize

One item whose format may not be obvious is the *wMaxPacketSize* entry for a high-bandwidth endpoint. It may be tempting to report an endpoint that can handle three 1024-byte packets per microframe, as having a maximum packet size of 3*1024 or 3072 bytes. However, this is not defined in the USB 2.0 spec because a physical packet has a maximum size of 1024 bytes. The spec defines bits 10-0 as the *maxPacketSize* of maximum size 1024, and bits 12-11 as the *additional packets* per microframe: 00 for none, 01 for one, and 10 for 2. Therefore, populate your high-bandwidth endpoint descriptors using a value in Table 5, low byte first.

Table 5. Allowable maxPacketSize Values for High-Bandwidth Endpoints

Packets per Microframe	wMaxPacketSize
1	0x0400
2	0x0C00
3	0x1400

7.3 CyStream.c

This is the streaming application. The application code is written in this module with help from the USB Frameworks:

1. You write the TD_Init() and TD_Poll functions to suit your app.
2. Fw.c makes calls to specifically named functions in your code as it fields various Device Requests over Endpoint 0. A Cypress code template (peripheral.c) saves the work of creating the functions by providing a code skeleton containing all the function stubs. You code only the functions your app uses.
3. You supply Interrupt Service Routines to handle the interrupts your app uses. Most of these (shown with an asterisk) are simple acknowledgement housekeeping; you do not need to modify them if no further action is necessary.

The remainder of this section walks through the CyStream.c code.

7.3.1 TD_Init()

The initialization program does the following:

1. Sets the IFCONFIG register to 0x40 to select the 8051 48-MHz clock rate and to configure a group of I/O pins as Port (GPIO) pins (these pins can also serve as Slave FIFO or GPIF interface pins).
2. Turns off the four development board LEDs. Note that the board LEDs are turned on and off by reading predefined memory locations.
3. Enables EP2, disables all the others. Later, if the host changes an alternate setting to one that uses EP6-OUT, the code enables this endpoint at the time the setting changes.
4. Enables Start Of Frame (SOF) Interrupts. The SOF ISR handles LED blink timing.
5. Fills EP2-IN FIFO with 1024 data bytes and arms it for the first IN transfer. Having specific data in the FIFO is handy for viewing packets with a USB bus analyzer, but the data is not relevant for this application. Therefore, this step is optional.

7.3.2 TD_Poll()

TD_Poll(), which the USB Frameworks calls in an endless loop, is quite simple. It only needs to check FIFO flags to determine when an endpoint FIFO needs re-arming. It does this in two similar code sections, one for high-speed and the other for full-speed operation. Each section does the following:

1. **Checks for not FULL condition of the EP2-IN FIFO.** This indicates that the host successfully read the FIFO IN data and FX2LP logic turned FIFO control over to the 8051. If it is not full:
   a. Turns on the IN LED and sets a time constant (inblink) for the SOF interrupt Service Routine (ISR) to turn it off.

   Re-arms the next IN transfer by loading the EP2-IN byte count registers with the size matching the current alternate setting. These values correspond to the rightmost column in Table 3 (HS) and Table 4 (FS).
2. **Checks for not EMPTY condition of the EP2-OUT FIFO.** This indicates that the host successfully sent OUT data to the EP2-OUT FIFO and the FX2LP logic has turned FIFO control over to the 8051. If it is not empty:
   a. Turns on the OUT LED and sets a time constant (outblink) for the SOF ISR to turn it off.
   b. Re-arms the next OUT transfer by loading a byte count into the EP2BCL register. The actual byte count value does not matter, but bit 7 must be set to set the SKIP bit. The SKIP bit enables the just- received packet to get ignored and its buffer is immediately made available for the next OUT packet.
3. **(HS only) Checks for not empty condition of the EP6-OUT FIFO**. This indicates that the host successfully sent OUT data to the EP6-OUT FIFO and FX2LP logic has turned FIFO control over to the 8051. If not empty, it executes steps 2a and 2b but using the EP6BCL register instead.

7.3.3 Fielding Device Requests and IRQs

Most simple Device Requests are self-explanatory, such as DR_SetConfiguration that sets a local variable to the configuration index, and DR_GetConfiguration that returns the index. The simple ISRs do nothing more than clearing the IRQ flags.

7.3.4 CyStream.c functions

CyStream.c does its work in four functions:

- ISR_Ures
- ISR_Highspeed
- DR_Set_Interface
- DR_SOF

7.3.4.1 ISR_Ures

This ISR triggers at completion of a USB bus RESET. A high-speed device comes out of reset in the full-speed mode, so the ISR sets the FS Configuration Descriptor as the main one, and the HS Configuration Descriptor as the OTHERSPEED one (Figure 3). Then, the ISR turns off the HS LED (D2 on the EVKIT board) and sets a 2-second period using ‘blinkmask’ for the activity LED D5.

7.3.4.2 ISR_Highspeed

This ISR triggers when FX2LP enters the high-speed mode. The Configuration Descriptors are swapped (Figure 4), the high-speed LED2 is turned on, and the default EP2-IN is initialized. The variable ‘blinkmask’ is set for a 1-second D2 blink period at high-speed.

7.3.4.3 DR_SetInterface

The USB host uses the Set_Interface request to change an interface’s alternate settings. Because this design uses only one interface, no check is required for the interface index. The code needs to retrieve only the third byte in the SETUP packet (SETUPDAT[2]), which represents the requested Alternate Setting.

Two switch statements configure the endpoints according to the Alternate Setting, using the Table 3 parameters for high-speed operation and the Table 4 parameters for full-speed operation. The case statements do the following:

- Enable only the endpoints used by the setting
- For IN endpoints, reset the FIFOS to flush any stale data. For BULK IN endpoints, reset the endpoint’s data toggle to 0
- For OUT endpoints, arm the required number of FIFOs
- Update the 7-segment display to show the new Alternate Setting 0-6

7.3.4.4 DR_SOF

SOF interrupts are useful for timing events, such as LED blink rates. SOF interrupts occur every 1 mSec at full-speed and every 125 microseconds at high-speed. The SOF ISR counts off time and turns off the activity LED D5, the IN LED D4, and the OUT LED D3 when appropriate.

8 Running CyStreamer.hex

Like all Cypress EZ-USB chips, FX2LP uses RAM for program storage. For this reason, it powers on as a USB loader capable of moving the program code into RAM from various sources. It can use an attached EEPROM or it can come up as a USB device capable of loading code from a PC. The steps in this section show how to download and run the *CyStream.hex* file over USB.

1. Prepare the FX2LP board jumpers according to Table 6.

Table 6. EZ-USB FX2LP Board Jumper Settings

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

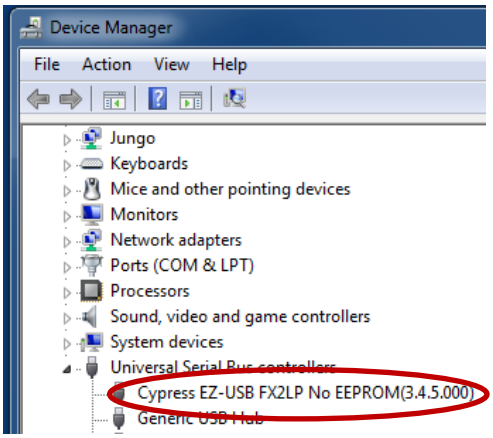
2. In the lower left corner of the board, move the EEPROM ENABLE slide switch (SW2) to the **NO EEPROM** (down) position. This allows the FX2LP chip to enumerate as a code loading the USB device. The other slide switch (SW1, EEPROM SELECT) can be in either position.
3. Plug the FX2LP board into a PC USB port. If this is the first time, you will see pop-up messages prompting you to install a USB driver. Navigate to:

   ```
   C:\Cypress\USB\
   CY3684_EZ-USB_FX2LP_DVK\
   1.0\Drivers\cyusbfx1_fx2lp
   ```

   Select the folder corresponding to your Windows OS.

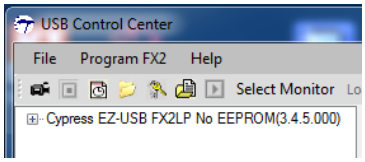
   You can confirm a successful driver install by viewing the **Device Manager** (see the following figure).

Figure 5. Driver Installed Correctly

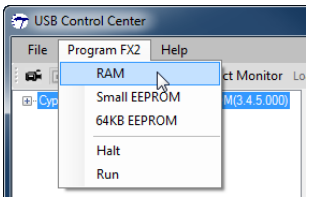


4. Launch the USB Control Center in the AN4053 folder. The executable is in the bin folder.
5. The FX2LP board is listed on the left panel as shown in Figure 6.

Figure 6. Device Appears in USB Control Center

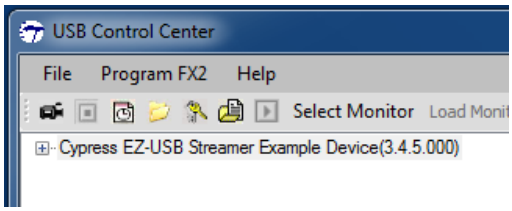


6. Click on the Cypress device entry to highlight it, then select **Program FX2** > **RAM**



7. Navigate to `AN4053\FX2LP streamer source code\firmwareCyStreamer` and double-click on the *CyStream.hex* file.
8. Watch the USB Control Center left panel. If the PC sound device is enabled, you hear the 'USB disconnect' sound immediately followed by the 'USB connect' sound, and a new USB device appears:

Figure 7. New USB Device



This is the Cypress ReNumeration process. The initial device seen in Figure 6 is the USB loader, hard-wired into the FX2LP device. After it loads your code into the FX2LP RAM, FX2LP electrically disconnects and reconnects as the new USB device seen in Figure 7, the one created by the code it loaded.

Note When you build a new version of CyStream.hex and want to download it into the FX2LP board, you must first press the board's RESET button. This disconnects the board from the USB. When you release the RESET button, FX2LP re-connects to the PC as the USB loader. It is a good idea to hold the RESET button down for about two seconds to give Windows enough time to register the disconnect-reconnect events.

The Streamer Example Device starts running, confirmed by the following indications:

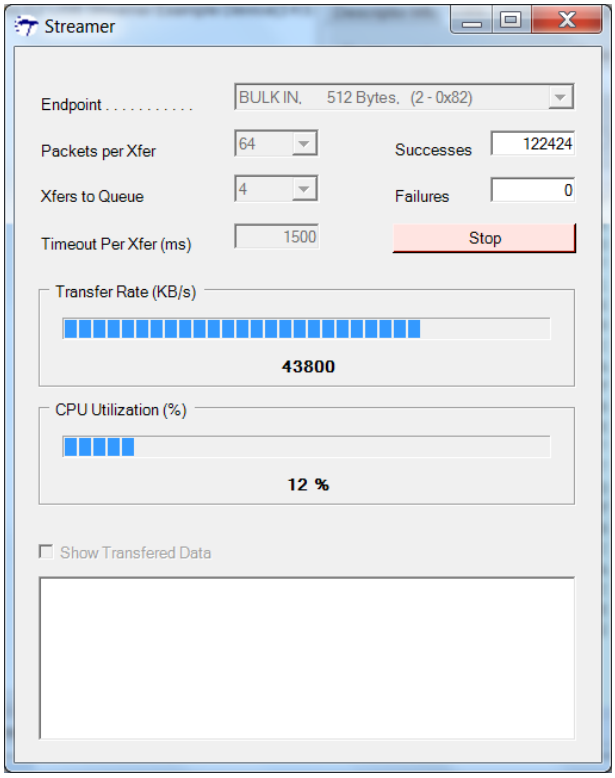
- The 7-segment readout indicates the current alternate setting. The default (power-on) setting is 0.
- LED D5 blinks once every second for a HS connection, and once every 2 seconds for a FS connection.
- LED D4 turns on when an IN FIFO is armed.
- LED D3 turns on when an OUT FIFO is armed.
- LED D2 is on for HS and off for FS operation.

9 Performance Analysis

The Streamer.exe in `AN4053\VS_Streamer` is used to evaluate the performance of a USB streaming application. This code uses the *CyUsb.sys* (AN4053\Driver) driver to interface with the FX2LP Development Board (CY3684) running the CYStream firmware.

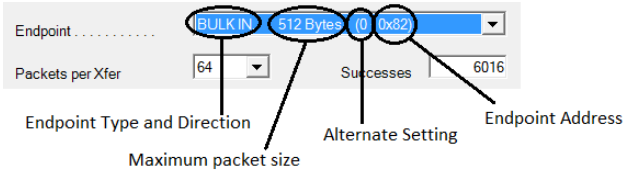
Following is a screen capture of the Streamer host application window, followed by a description of the fields shown in the window displayed in Figure 8.

Figure 8. Streamer.exe Application Window Display



Endpoint: This drop-down list indicates the alternate settings, 0-6 for high speed and 0-3 for full speed. This allows you to choose different transfer types and number of endpoint buffers.

Figure 9. Description of Endpoint Dropdown List



The description of the text that you see in the Endpoint drop-down list is shown in this section. Let us take the first one in the drop-down list (refer to Figure 9): BULK IN, 512Bytes, (0-0x82).

BULK IN – This field tells you the endpoint type and direction. Here in this example, BULK endpoint and you can perform IN transfers on it.

512Bytes – Maximum packet size of endpoint. In this case, 512 bytes.

0 – This field contains the Alternate Setting number. In this case, Alternate Setting – 0.

0x82 – Endpoint address. In this case, the endpoint number 0x02 and direction IN.

Packets per Xfer: A transfer is a collection of packets for one data set. More number of packets per each transfer reduces the USB overhead and helps in achieving higher data rate.

Xfers to Queue: This setting helps in initiating multiple transfers and adding them to the task queue. This reduces the latency between successive transfers on the host application side. Therefore, queuing more transfers gives higher data rate.

Successes: Increments to show the total number of packets successfully transferred during the streaming test.

Failures: Increments whenever there is an error reported in the transfer of a buffer. One possible failure mechanism is an FX2 (not LP) app that did not properly handle ISO IN PID sequencing.

Transfer Rate: Provides live updating of the current throughput performance of the USB bus and EZ-USB FX2LP over the selected endpoint.

CPU Utilization: Provides a visual indication of the utilization of the computer's CPU while streaming over USB.

This application was run on a system with an Intel IHC9 USB 2.0 host controller running Windows® 7. The results show that the average throughput that can be achieved with the BULK transfers is 43.8 MBps and the maximum throughput that can be achieved with ISO transfers is 24 MBps. Note that the performance numbers mentioned above are tested without connecting any external peripheral to FX2LP. Data is generated internal to FX2LP. If you are generating data from the external peripheral, then these performance numbers also depend on how fast that external peripheral can source data.

You might find on certain systems that the BULK transfer rate can exceed the ISO transfer rate. This is because on a lightly loaded bus, the streamer device uses most or all of the available USB BULK bandwidth allocation. However, when other USB devices start up, for example a USB disk drive playing a video, the available BULK bandwidth decreases while the ISO endpoints are guaranteed their negotiated bandwidth.

10 Interfacing FX2LP with Image Sensor

FX2LP can be interfaced to an image sensor to stream images using a vendor class driver. See KBA95736 - Interfacing FX2LP™ with Image Sensor.

11 References

USB 2.0 Specification – (http://www.usb.org/developers/docs/usb_20_070113.zip)

FX2LP Technical Reference Manual(TRM)

12 Summary

This application note describes USB high-bandwidth delivery mechanisms that support streaming data for applications like audio and video streaming. The associated firmware project “CYStream.Uv2” shows how to program FX2LP for high-speed USB ISO transfers. A set of BULK transfers are also implemented for bandwidth comparison purposes. It also demonstrates how to use the Cypress USB Frameworks to implement alternate USB settings, enabling the host to select different transfer rates. A companion PC application “Streamer.exe” is provided to select various transfer types and to measure transfer rates.

About the Author

Name: Rama Sai Krishna Vakkantula.

Title: Applications Engineer Staff

Document History

Document Title: AN4053 - Streaming Data Through Isochronous or Bulk Endpoints on EZ-USB® FX2™ and FX2LP™

Document Number: 001-15289

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1785566	AVF	08/18/2008	Reviewed the existing app note. Added spec number and new disclaimer; updated copyright.
*A	3102683	RSKV	12/15/2010	Figure 1 – TD_init Routine and Figure 2 – TD_poll Routine modified. Added Document History Page and updated as per application note template.
*B	3128466	RSKV	01/06/2011	Added Abstract section.
*C	3211712	AASI	03/31/2011	Added the project and updated the title.
*D	3298105	RSKV	06/30/2011	Modified part of the firmware source code in Figure 1. Added Reference section.
*E	3630276	RSKV	06/27/2012	Updated template (converted application note from FrameMaker to Word format). Updated document with respect to Cypress Suite USB. Major text edits.
*F	4136544	RSKV	09/26/2013	Updated the firmware project to use LEDs available on FX2LP DVK. Attachment contains source code of Streamer and Control Center applications. Most of the sections are re-written.
*G	4486771	AKSL	08/28/2014	Updated in new template. Completing Sunset Review.
*H	4663729	DBIR	02/17/2015	Added reference to “Interfacing FX2LP™ with Image Sensor - KBA95736”
*I	5701881	BENV	04/19/2017	Updated logo and copyright

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

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2008-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.