AN86947

Optimizing USB 3.0 Throughput With EZ-USB® FX3™

Authors: Manaskant Desai, Karthik Sivaramakrishnan
Associated Project: Yes
Associated Part Family: EZ-USB FX3
Software Version: FX3 SDK v1.3.3
Related Application Notes: For a complete list, click here

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN86947.

USB 3.0 delivers a significant performance improvement over previous generations because it uses a 5-gigabit-per-second (SuperSpeed) signaling rate. However, the maximum achievable throughput also depends on critical factors such as host PC controller type, operating system, and USB design (transfer type and buffer sizes). This application note describes strategies for using an EZ-USB® FX3™ controller to maximize USB 3.0 data transfers. For complete list of USB SuperSpeed Code Examples, visit http://www.cypress.com/?rID=101781.

Contents

1 Introduction

Cypress's EZ-USB FX3, the next-generation USB 3.0 peripheral controller, provides highly integrated and flexible features that enable you to add USB 3.0 functionality to any system. USB 3.0 has a signaling rate of 5 Gbit per second, or 10 times the rate supported by USB 2.0. An FX3 device enables a throughput close to the theoretical maximum supported by the USB 3.0 specification.

FX3 supports all SuperSpeed transfer types: control, isochronous, bulk, and interrupt. However, control transfers are not recommended for moving large amounts of data. Therefore, this application note summarizes the guidelines for achieving the maximum throughput only for the isochronous, bulk, and interrupt transfer types.

FX3 is commonly used as a bridge device that transfers data between the USB 3.0 Host and an external device, such as an image sensor, ASIC, or FPGA. The data throughput obtained by an FX3-based application depends on multiple factors: USB transfer type; transfer parameters, such as burst size and the amount of data buffering used within the FX3 device; and the host controller and OS platform used.

Using an FX3 development board, this application note tests and compares transfer strategies using various hosts and operating systems. It uses a set of simple FX3 firmware examples to study the impact of each factor that influences throughput. It describes performance optimization guidelines for a given application and concludes with FX3 configuration details to achieve the maximum USB 3.0 throughput for a given application.

Note: This application note lists the USB 3.0 throughput numbers only with the data generated internal to FX3. The USB 3.0 throughput numbers will be lower when data is coming from the external device into FX3 through GPIF II. Refer to AN65974 to get the throughput numbers when the FPGA is writing data into the FX3.

2 Related Resources

Cypress provides a wealth of data at www.cypress.com to help you select the right device for your design and quickly and effectively integrate the device into your design. For a comprehensive list of resources, see the knowledge base article KBA87889, How to Design with EZ-USB FX3 and FX3S™.

- **Overview:** USB Portfolio, USB Roadmap
- **USB 3.0 Product Selectors:** FX3, FX3S, CX3™, HX3, West Bridge® Benicia™
- **Application Notes:** Cypress offers many USB application notes covering a broad range of topics, from basic to advanced level. The recommended application notes for getting started with FX3 are:
  - AN75705 – Getting Started with EZ-USB FX3
  - AN76405 – EZ-USB FX3 Boot Options
  - AN70707 – EZ-USB FX3/FX3S Hardware Design Guidelines and Schematic Checklist
  - AN65974 – Designing with the EZ-USB FX3 Slave FIFO Interface
  - AN75779 – How to Implement an Image Sensor Interface with EZ-USB FX3 in a USB Video Class (UVC) Framework
  - AN86947 – Optimizing USB 3.0 Throughput with EZ-USB FX3
  - AN84868 – Configuring an FPGA over USB Using Cypress EZ-USB FX3
  - AN68829 – Slave FIFO Interface for EZ-USB FX3: 5-Bit Address Mode
  - AN73609 – EZ-USB FX2LP™/ FX3 Developing Bulk-Loop Example on Linux
  - AN77960 – Introduction to EZ-USB FX3 High-Speed USB Host Controller
  - AN76348 – Differences in Implementation of EZ-USB FX2LP and EZ-USB FX3 Applications
  - AN89661 – USB RAID 1 Disk Design Using EZ-USB FX3S
- **Code Examples**:
  - USB Hi-Speed
  - USB Full-Speed
  - USB SuperSpeed
- **Technical Reference Manual (TRM):** EZ-USB FX3 Technical Reference Manual
- **Development Kits (DVKs):**
  - CYUSB3KIT-003, EZ-USB FX3 SuperSpeed Explorer Kit
  - CYUSB3KIT-001, EZ-USB FX3 Development Kit
- **Models:** IBIS

2.1 EZ-USB FX3 Software Development Kit

Cypress delivers the complete software and firmware stack for FX3 so you can easily integrate SuperSpeed USB into any embedded application. The Software Development Kit (SDK) comes with tools, drivers, and application examples, which help accelerate application development.

2.2 GPIF II Designer

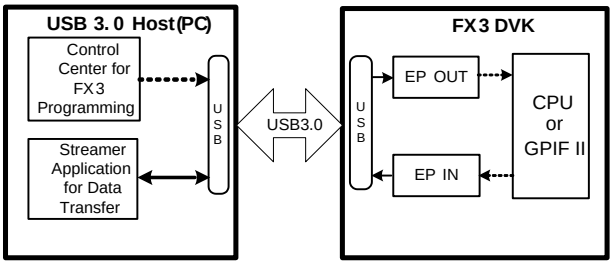
The GPIF II Designer is graphical software that allows you to configure the GPIF II interface of the EZ-USB FX3 USB 3.0 Device Controller. It lets you select from one of five Cypress-supplied interfaces or create your own GPIF II interface from scratch. Cypress supplies industry-standard interfaces such as asynchronous and asynchronous Slave FIFO, synchronous SRAM, and asynchronous SRAM. If you already have one of these predefined interfaces in your system, you can simply select the interface you want; select from a set of standard parameters such as bus width (x8, 16, and x32), endianess, and clock settings; and compile the interface.

If you need a customized interface, the tool offers a streamlined three-step GPIF interface development process that allows you first to select the pin configuration and standard parameters. Then you can design a virtual state machine using configurable actions. Finally, you can view the output timing to verify that it matches the expected timing. Once the three-step process is complete, you can compile the interface and integrate it with FX3.

3 Test Setup

Figure 1 shows the test setup for this application note. All testing is performed using the FX3 DVK connected to a PC host through its USB 3.0 connector.

Figure 1. Test Setup



The Control Center application is used to download the firmware image to FX3 internal RAM. After the download, FX3 is ready to perform data transfers using the Streamer application. For your convenience, Control Center, C++ Streamer application binaries, and the Cypress USB 3.0 driver (*CyUSB3.sys*) are also provided with the attachment. The Streamer application provided with the attachment is configured to give the best performance numbers.

To download the latest FX3 SDK, go to www.cypress.com/?rID=57990.

To find the C++ version of the Streamer application, use the following path after SDK installation:

C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\application\cpp\streamer\x86\Release

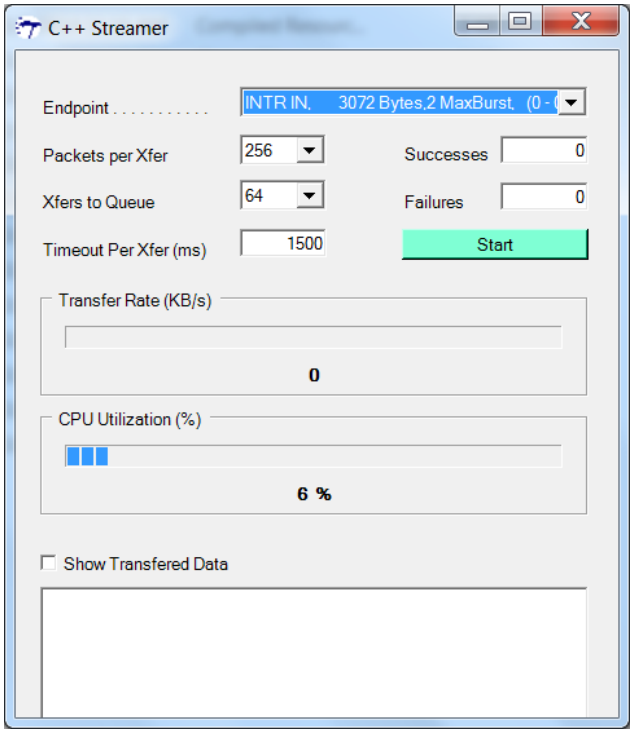
Note: On a 64-bit Windows machine, the Program Files directory is “Program Files (x86).”

Table 1 lists the four transfer tests. The Streamer application shown in Figure 2 supports these tests and serves as a “throughput meter.” This application note uses the C++ version of the Streamer application to measure the performance of three types of transfers (isochronous, bulk, and interrupt). Note that the C# version delivers a lower throughput due to application and driver-level overhead.

Table 1. Code Images for Four Transfer Tests

Test	FX3 code image	Description
1	USBIsoSourceSink	EP1-IN is a constant source of ISO data; EP1-OUT is a constant sink for ISO data. There is no GPIF II connection.
2	USBBulkSourceSink	Same as test 1, but bulk transfers are used.
3	USBIntrSourceSink	Same as test 1, but interrupt transfers are used.
4	GpifToUsb	The GPIF II is used for a maximum-speed transfer into EP-IN, which executes bulk transfers to the PC. This is the only test that uses the dotted path in Figure 1 from the GPIF II to EP1-IN.

Figure 2. C++ Streamer



The Streamer application user interface includes the following:

- **Endpoint:** This drop-down list allows you to select different transfer types and number of endpoint buffers.
- **Packets per Xfer:** A transfer is a collection of packets for one data set. A greater number of packets per transfer reduces the USB overhead and helps achieve a higher data rate.
- **Xfers to Queue:** This setting helps in initiating multiple transfers and adding them to the task queue. It reduces the latency between successive transfers on the host application side. Therefore, queuing a greater number of transfers yields a higher data rate.
- **Successes:** Increments to show the total number of packets successfully transferred during the streaming test.
- **Failures:** Increments whenever an error is reported in the transfer of a buffer. One possible failure is no data from the device.
- **Transfer Rate:** Provides live updating of the current throughput performance of the USB bus and EZ-USB FX3 over the selected endpoint.
- **CPU Utilization:** Provides a visual indication of the utilization of the computer's CPU while streaming over USB.
- **Timeout Per Xfer:** The transfer fails if there is no data from the device within the timeout value. The default value is 1500 ms.

Table 1 lists the four code images (*.img* files) that are downloaded to the FX3 DVK for bandwidth testing. These modules are available in the *.zip* file that accompanies this application note.

- The USBIsoSourceSink example is used to measure the performance of SuperSpeed isochronous transfers. This example uses a pair of IN and OUT endpoints, which continuously source or sink ISO (isochronous) data on the FX3 device side.
- The USBBulkSourceSink example is used to measure the performance of SuperSpeed bulk transfers. This example also uses a pair of IN and OUT endpoints, which continuously source or sink BULK data.
- The USBIntrSourceSink example is used to measure the performance of SuperSpeed interrupt transfers.

- The GpifToUsb example is used to measure the performance of GPIF-to-USB bulk data transfers. This example continuously reads data from a barebone GPIF II interface and sends it to the USB Host through an IN endpoint, without any firmware intervention.

Note: The FX3 side of these test transfers does not introduce overhead, such as processing data in FIFOs or moving off-chip data. Processing overhead is application dependent and can decrease the maximum throughput numbers measured in this application note. Use the FX3 GPIF II and DMA capabilities judiciously to minimize any throughput decrease. Section 12 of this document gives performance tips to maximize FX3 throughput.

In all code examples, the instruction cache is enabled and the data cache is disabled. It is recommended that you leave the data cache OFF unless the firmware application needs to perform regular data manipulation.

The examples are tested using the FX3 DVK directly connected to a PC host using an Intel C216 chipset family USB 3.0 eXtensible host controller (XHCI). The operating system used is 64-bit Windows 7, and the Intel host driver version is 1.0.5.235.

4 Performance Summary

Table 2 shows the performance summary for the three transfer types using the three SourceSink firmware versions. “Burst length”[1] in the table refers to the burst size reported to the host in the USB descriptor and set in the firmware, and “No. of packets” refers to the isochronous packets transferred per service interval. This parameter applies only to isochronous and interrupt transfers. For interrupt transfers, the number of packets is always 1. For a detailed throughput analysis, refer to the Sections 6, 7, 8, and 9.

Table 2. Performance Summary Results

Transfer	Burst length	No. of packets	Buffer size (KB)	No. of buffers	Throughput (KB/s)
Isochronous	16	3	48	2	382,700
Bulk	16	NA	48	2	454,300
Interrupt	3	1	3	1	23,900

5 Operating Instructions

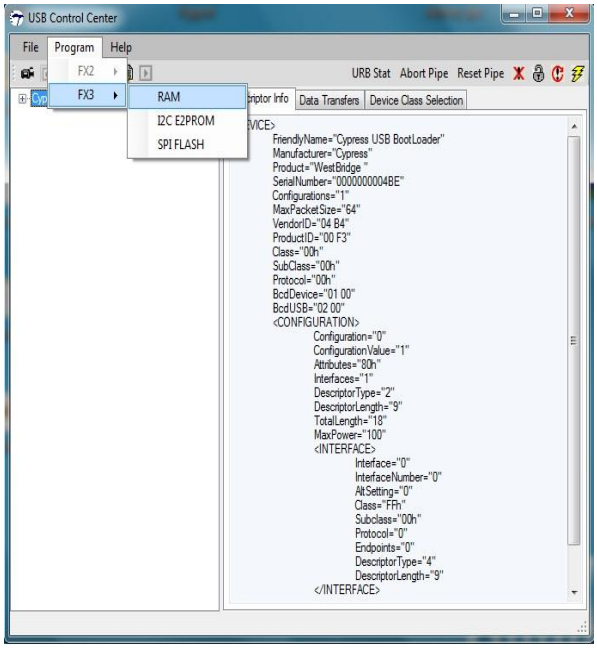
If you are using the FX3 DVK for the first time, refer to the application note AN75705 – Getting Started with EZ-USB FX3. You can measure the throughput for all of the previously mentioned transfer types by following these instructions:

1. Download the *.zip* file that contains the firmware, Control Center, C++ Streamer application, and Cypress USB 3.0 drivers attached with this application note.
2. Download one of the four image files using the USB Control Center (shown in Figure 3), which is provided in the attachment. This is also installed as part of the FX3 SDK *(<install directory>\Cypress\EZ-USB FX3 SDK\1.3\application\c_sharp\controlcenter\bin\Release*).
3. Open the C++ Streamer application provided in the attachment, select the parameters as shown in Figure 2, and click the **Start** button.

Data throughput is displayed in KB/s.

[1] USB 3.0 introduces data bursting, which allows a certain number of data packets to be transferred over an endpoint without requiring a handshake in between packets. An individual endpoint reports its burst capability (that is, the maximum number of packets per burst) to the host via the SuperSpeed Endpoint Companion descriptor associated with the particular endpoint.

Figure 3. USB Control Center



6 Isochronous Transfers

The isochronous transfer type is suitable for data streaming applications, such as audio and video. Isochronous transfers provide guaranteed data bandwidth on the SuperSpeed bus, but they do not include handshakes or retries.

SuperSpeed isochronous endpoints support a maximum data packet payload size of 1024 bytes. Burst transactions increase the data transfer rate because the producer of data need not wait for an ACK until a specified burst size has been completed. The FX3 device supports the maximum burst size of 16 that is defined by the USB specification. However, each SuperSpeed isochronous endpoint can request up to three burst transactions (iso-packets) in the same service interval[2], according to the USB 3.0 specification.

An isochronous transfer can specify the service interval as $2^{(bInterval-1)}$ x 125 µs, where bInterval ranges from 1 to 16. A SuperSpeed isochronous transfer type can move up to 1024 x 16 x 3 bytes per service interval. Taking the minimum service interval as 125 µs, the maximum theoretical bandwidth for SuperSpeed isochronous transfer can be calculated as (maximum packet size * burst size * number of iso-packets / service interval). That works out to 1024 x 16 x 3 / 125µ bytes/s or 375 Mbps (3 Gb/s).

Use the USBIsoSourceSink firmware example to measure SuperSpeed isochronous endpoint throughput on FX3. The throughput depends on the burst length, buffer size[3], iso-packets, and number of DMA buffers used.

You can change these parameters by using the definitions in the *cyfxisosrcsink.h* header file:

- The burst length for the isochronous endpoints is set using the CY_FX_ISO_BURST definition.
- The number of ISO packet bursts per service interval is set using the CY_FX_ISO_PKTS definition.
- The size of each DMA buffer used for transfer is set using the CY_FX_ISOSRCSINK_DMA_BUF_SIZE definition.
- The number of DMA buffers used on each endpoint is set using the CY_FX_ISOSRCSINK_DMA_BUF_COUNT definition.

[2] For interrupt and isochronous endpoints, the specified interval at which the endpoint must be serviced by the host is known as the "service interval." The service interval for an interrupt or isochronous pipe is specified via the endpoint descriptor.

[3] Configured for FX3 DMA buffers when initializing the DMA channels in FX3 firmware.

The CY_FX_ISO_BURST and CY_FX_ISO_PKTS parameters are set to 15 and 3, respectively, by default. Some USB Hosts may not be able to support the transfer bandwidth required by this setting, and they fail to select the configuration. In this case, you can lighten the bandwidth requirement by reducing the value of either of those parameters.

After changing any of these parameters, you should recompile the application. The resultant firmware binary (*USBIsoSourceSink.img*) can be loaded into FX3 RAM, and then the Streamer application can be used to measure the transfer performance.

The isochronous throughput results obtained by varying the burst length, iso-packets, and buffering are tabulated in Table 3 and Table 4. From those results, it is clear that the FX3 can deliver a performance close to the theoretical bandwidth using isochronous endpoints. Therefore, the SuperSpeed isochronous throughput mainly depends on the burst length and number of iso-packets per transfer.

Note: The throughput number for IN transfers with a burst length of 16 is significantly lower than that for transfers with a burst length of 15. This is due to slower request scheduling by the USB Host; it is not caused by any change in the data rate supported by the FX3 device. This can be verified by modifying the endpoint descriptor to report a burst length of 15 while keeping the device-side endpoint configuration for a burst length of 16.

Table 3. Isochronous IN Endpoint Throughput Results

Burst length	Iso-packets	Buffer size (KB)	No. of buffers	Throughput (KB/s)	Theoretical (KB/s)
8	1	8	1	63,900	64,000
8	3	24	1	191,900	192,000
12	1	12	1	96,000	96,000
12	3	36	1	287,900	288,000
15	3	15	1	119,800	360,000
15	3	15	2	239,500	360,000
15	3	45	2	359,900	360,000
16	3	48	2	139,400	384,000

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

Table 4. Isochronous OUT Endpoint Throughput Results

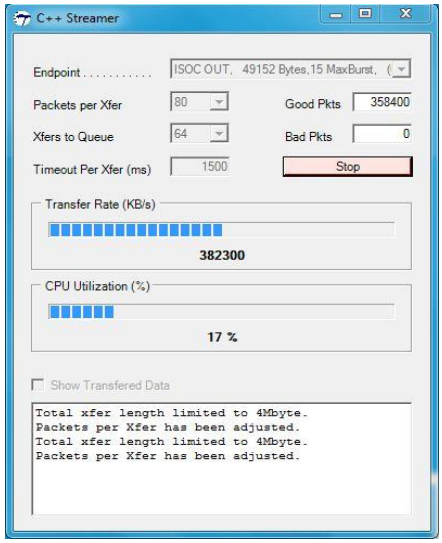
Burst length	Iso-packets	Buffer size (KB)	No. of buffers	Throughput (KB/s)	Theoretical (KB/s)
8	1	8	1	63,900	64,000
8	3	24	1	191,900	192,000
12	1	12	1	96,000	96,000
12	3	36	1	287,900	288,000
15	3	45	2	359,900	360,000
16	3	48	1	382,700	384,000
16	3	16	2	251,200	384,000

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

As shown in these results, the use of DMA buffers large enough to hold one service interval's worth of data is recommended for optimum results. The FX3 firmware processing time for each DMA buffer is about 40 µs, which is much lower than the duration of a service interval. If all data to be transferred during a service interval can be accommodated in one DMA buffer, a single DMA buffer is sufficient to obtain the optimum transfer rate.

The Streamer application indicating the isochronous throughput is shown in Figure 4. As you can see, selecting the maximum allowed values for the **Packets per Xfer** and **Xfers to Queue** options yields the best possible performance.

Figure 4. Isochronous OUT Endpoint Throughput



7 Bulk Transfers

Bulk transfers are more suitable for devices that must communicate a high volume of data at a varying rate. In such cases, the transfer can use all the available bandwidth. A SuperSpeed bulk transfer uses the bus depending on the free bandwidth and ensures data transfer and data integrity, but with no bandwidth guarantee. The bulk transfer type applies more to high-rate data transfer applications, such as mass storage devices. It can also be used for video data transfers when the USB Host can sustain the required transfer rate.

SuperSpeed bulk endpoints support a maximum data packet payload size of 1024 bytes. These endpoints also support burst sizes from 1 to 16. (A “burst” is a series of BULK packets that do not require individual ACKs from the receiving end.)

Because no fixed bandwidth is allocated for a bulk transfer, the maximum theoretical throughput cannot be determined. It depends on the free bandwidth available after accounting for the bandwidth allocated to all other devices connected to the same USB Host. If the entire bandwidth is available for a single bulk transfer, the maximum theoretical throughput for the bulk transfer will be about 4 Gbit/s, after reserving 20 percent of the possible transfers for link and protocol-level overheads.

The USBBulkSourceSink example is used to measure the FX3 SuperSpeed bulk endpoint throughput, which depends on the burst size, buffer size, and number of buffers.

You can vary these parameters by modifying the definitions provided in the *cyfxbulksrcsink.h* header file:

- The burst length for the endpoints is set using the CY_FX_EP_BURST_LENGTH definition.
- The size of each DMA buffer used for the data transfer is set using the CY_FX_BULKSRCSINK_DMA_BUF_SIZE definition.
- The number of DMA buffers used for each endpoint is set using the CY_FX_BULKSRCSINK_DMA_BUF_COUNT definition.

The bulk throughput results obtained for various burst lengths and buffer sizes are tabulated in Table 5 and Table 6.

Table 5. Bulk IN Endpoint Throughput Results

Burst length	Buffer size (KB)	No. of buffers	Throughput (KB/s)
8	16	1	263,100
8	16	2	448,300
12	24	1	305,000
12	24	2	450,000
16	48	2	454,300
16	16	2	351,800
16	16	1	120,500

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

Table 6. Bulk OUT Endpoint Throughput Results

Burst length	Buffer size (KB)	No. of buffers	Throughput (KB/s)
8	16	1	249,900
8	16	2	360,100
12	24	1	289,000
12	24	2	377,000
16	48	2	405,000
16	16	2	364,900
16	16	1	153,800

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

From these results, you can see that in the best case (no other connected USB Devices sharing bus bandwidth), bulk transfers can provide better throughput than isochronous transfers. The tables show that the SuperSpeed bulk endpoint throughput depends on the burst length, buffer size, and number of DMA buffers used.

As in isochronous transfer, using large DMA buffers, which can hold multiple bursts of data, improves the performance. The throughput dependency on the buffer size can be explained using the following calculations:

With a burst size of 16, the best throughput obtained is 454,300 KB/s.

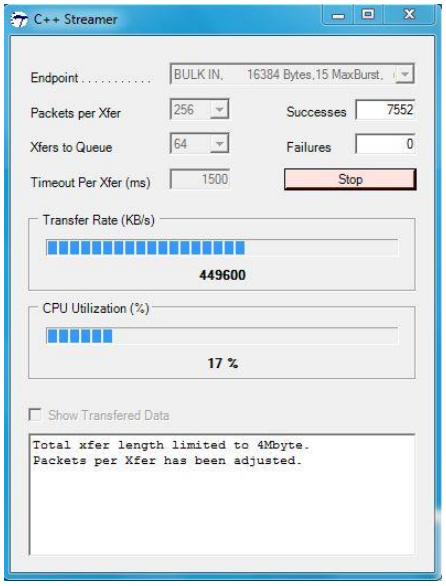
- Number of 16-KB bursts processed per second = 28,400
- Average time for a 16-KB data transfer = 35 µs
- Average time for a 32-KB data transfer = 70 µs

Each DMA buffer is turned around by the firmware application using a pair of CyU3PDmaChannelGetBuffer and CyU3PDmaChannelCommitBuffer API calls. Performance benchmarking shows that a pair of GetBuffer and CommitBuffer API calls costs about 40 µs of CPU time.

From Table 5 and Table 6, it can be seen that limiting each DMA buffer to one burst (16 KB) or less would cause the firmware processing to set the throughput limit. By increasing the size of each buffer to 32 KB (two bursts of 16K KB each), you can ensure that the average transfer time for one buffer is greater than the average firmware processing time for one buffer. Such a setting yields the optimum performance for bulk data transfers when there is firmware intervention.

The maximum throughput for the bulk IN endpoint is 454,300 KB/s or 3.7 Gbit/s, and the throughput for the bulk OUT endpoint is 405,000 KB/s or 3.31 Gbit/s. The Streamer application indicating the bulk throughput is shown in Figure 5.

Figure 5. Bulk IN Endpoint Throughput



8 Interrupt Transfers

The interrupt transfer type is more suitable for devices that require high data reliability for a small data volume with a guaranteed maximum service interval. The SuperSpeed interrupt transfers provide a guaranteed service interval and a guaranteed data transfer using handshakes and retries. The interrupt transfer type is more applicable to HID devices, such as mice or keyboards, and is not commonly used in throughput-critical applications.

SuperSpeed interrupt endpoints support a maximum data packet payload size of 1024 bytes. Per the USB 3.0 specification, SuperSpeed interrupt transfer supports a burst size up to only three packets. Moreover, it supports only one burst transaction per service interval. The service interval for interrupt transfers is defined in the same way as it is for isochronous transfers. The maximum theoretical bandwidth for a SuperSpeed interrupt transfer, using the service interval of one microframe, or 125 µs, can be calculated as (maximum packet size * burst size / service interval), which is (1024 x 3 / 125µ) bytes/s or 23.43 Mbps.

The USBIntrSourceSink example is used to measure the FX3 SuperSpeed interrupt endpoint throughput. As in isochronous transfer, the DMA buffer is chosen as equal to the amount of data to be transferred per service endpoint. A single DMA buffer is sufficient for obtaining the maximum throughput.

The burst size for the interrupt endpoint is set using the CY_FX_INTR_BURST_SIZE definition in the *cyfxintrsrcsink.h* header file.

The throughput obtained for interrupt transfers is close to the theoretical limits, and it is the same for both IN and OUT endpoints. Table 7 gives the throughput obtained for interrupt transfers at various burst size settings.

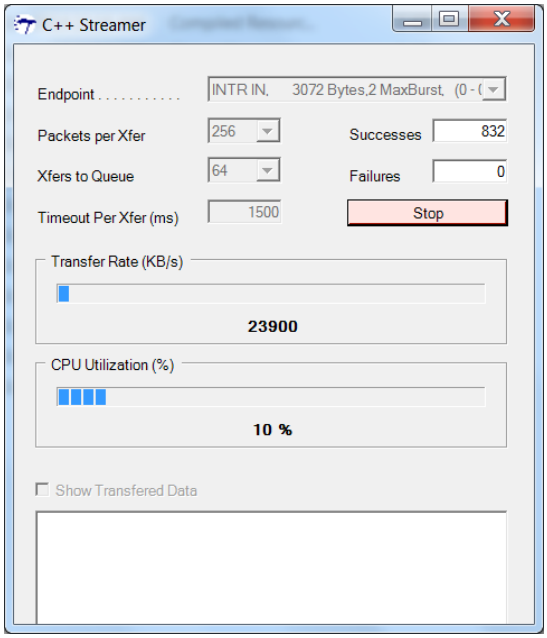
Table 7. Interrupt IN/OUT Endpoint Throughput Results

Burst length	Buffer size (KB)	No. of buffers	Throughput (KB/s)	Theoretical (KB/s)
1	1	1	7900	8000
2	2	1	15,900	16,000
3	3	1	23,900	24,000

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

The Streamer application indicating the interrupt throughput is shown in Figure 6.

Figure 6. Interrupt IN Endpoint Throughput



9 GPIF-to-USB Performance on an AUTO DMA Channel

The transfer performance obtained in all the previous examples is influenced by the firmware execution, and it gives good results when the size of the DMA buffers is kept large. The GpifToUsb firmware example attached with this application note measures how the data transfer performance varies with the endpoint burst setting, as well as the size and number of DMA buffers.

This example uses a minimal GPIF state machine that continuously fills data whenever a DMA buffer is available on the GPIF side. This state machine does not require an external device to drive any signals and commits full DMA buffers only at the fastest possible rate. The GPIF II Designer project for this state machine is attached with the application note in the *continuous_read.cydsn* folder.

GPIF II data is sent to the USB Host over a bulk IN endpoint through an AUTO DMA channel (which does not require any firmware intervention). There is no firmware involvement in the data path in this example, so these results can be used to determine how the transfer throughput is affected solely by the burst size, buffer size, and number of buffers.

These parameters are varied using definitions in *the cyfxgpiftousb.h* header file:

- The burst size for the endpoint is set using the CY_FX_EP_BURST_LENGTH definition.
- The size of each DMA buffer used is set using the CY_FX_DMA_BUF_SIZE definition.
- The number of DMA buffers used is set using the CY_FX_DMA_BUF_COUNT definition.

The maximum theoretical performance possible is determined by the GPIF interface data rate. In this case, the GPIF interface is run at a clock rate of 100.8 MHz with a 32-bit-wide data bus. This translates to a maximum possible data rate of 403.2 MB/s.

Figure 7 shows a graph that illustrates how the transfer performance varies with the burst length setting for various amounts of DMA buffering used. Each of the curves in the graph represents different amounts of total DMA buffering, ranging from a single buffer holding one burst of data to four buffers holding three full bursts.

The performance obtained with a single DMA buffer is consistently low because the GPIF transfer pauses while the USB transfer is happening, and vice versa. The graph shows that the transfer performance increases with burst length, up to a burst length of 8 KB and then levels off. It also shows that the transfer performance increases with a buffer size up to a total buffer depth of four burst transfers and then levels off.

This graph shows that the optimal settings for GPIF-to-USB bulk-based applications are a burst length of 8 KB and a total buffering of about 32 KB. Further increases in burst length and buffering give only marginal performance improvements at the expense of increased RAM use.

Figure 7. Variation of USB Performance with Burst Size

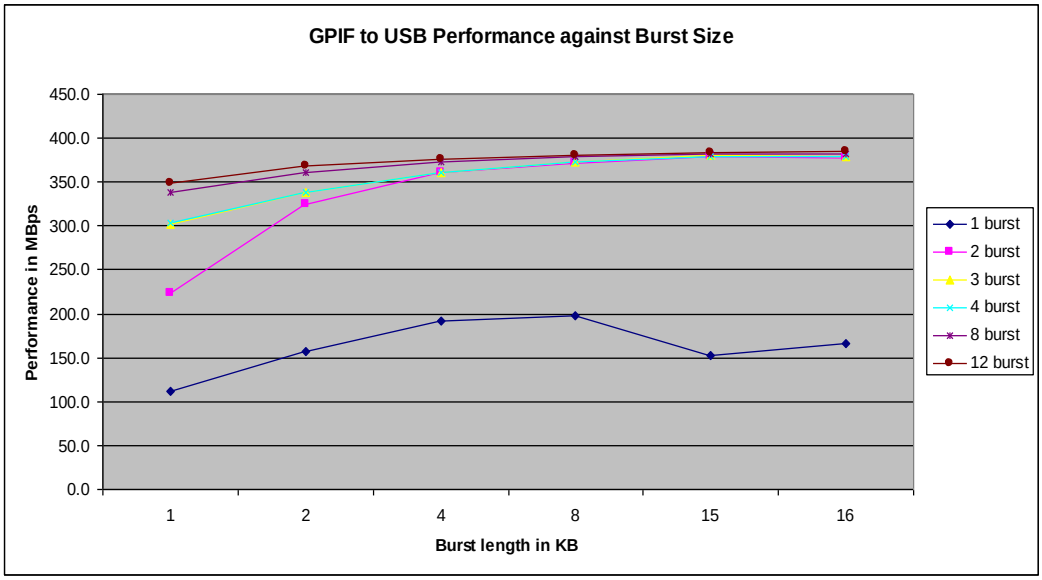


Table 8. indicates the throughput numbers obtained using this firmware application for various values of the endpoint burst length, DMA buffer size, and number of DMA buffers. It shows how the GPIF-to-USB transfer performance varies with the individual DMA buffer size for various burst lengths. Four DMA buffers are used in all cases, and the variation is only in the size of an individual DMA buffer.

Figure 8 shows that DMA buffers that can hold two burst transfers of data will deliver significantly better performance than buffers that can hold one burst transfer of data. Further increases in buffer size have little to no benefit. This graph shows that the ideal DMA buffer size is two times the endpoint burst length.

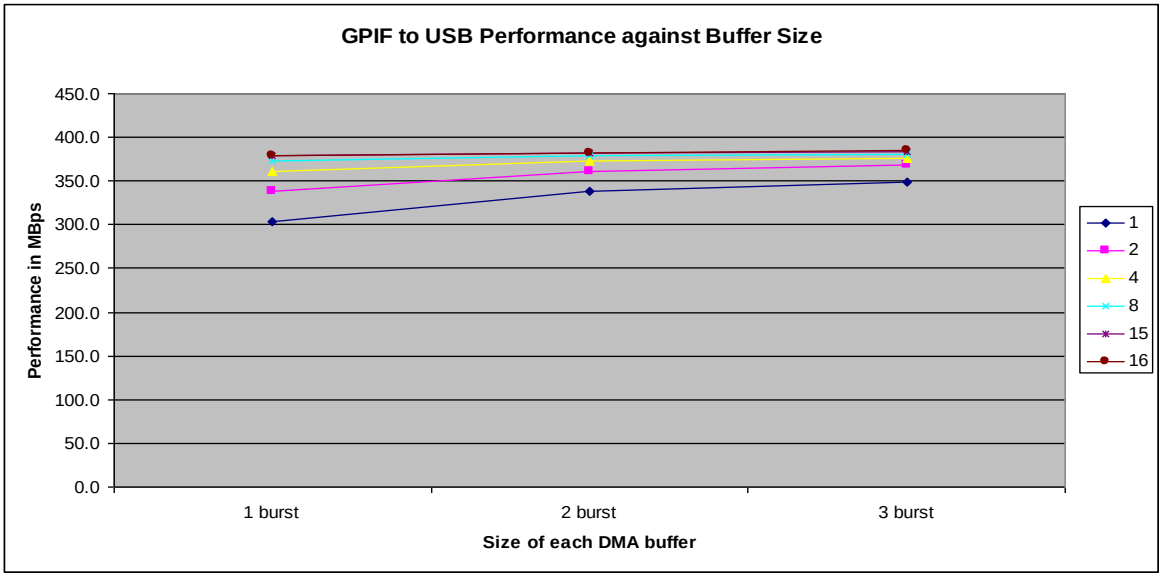
Table 8. GPIF-to-USB Bulk Transfer Throughput Results

Burst length (KB)	Size of each DMA buffer	Performance with 1 DMA buffer (KB/s)	Performance with 2 DMA buffers (KB/s)	Performance with 3 DMA buffers (KB/s)	Performance with 4 DMA buffers (KB/s)
1	1 burst	115,200	229,000	308,500	310,800
1	2 bursts	–	–	–	346,700
1	3 bursts	–	–	–	356,800
2	1 burst	160,600	333,100	346,000	347,100
2	2 bursts	–	–	–	369,600
2	3 bursts	–	–	–	377,300
4	1 burst	196,000	368,800	369,600	370,100
4	2 bursts	–	–	–	381,900
4	3 bursts	–	–	–	385,600
8	1 burst	202,100	380,700	382,100	382,100
8	2 bursts	–	–	–	387,700
8	3 bursts	–	–	–	389,500
16	1 burst	170,700	386,900	387,700	388,600

Burst length (KB)	Size of each DMA buffer	Performance with 1 DMA buffer (KB/s)	Performance with 2 DMA buffers (KB/s)	Performance with 3 DMA buffers (KB/s)	Performance with 4 DMA buffers (KB/s)
16	2 bursts	–	–	–	391,200
16	3 bursts	–	–	–	394,000

Note: These throughput numbers are measured by selecting **256 Packets per Xfer** and **64 Xfers to Queue** in the Streamer application. These settings are preselected if you are using the Streamer application from the attachment.

Figure 8. Variation of USB Performance with DMA Buffer Size



10 Host Controller Performance Comparison

The USB data throughput obtained also depends on the capabilities of the USB Host used. To illustrate this dependency, the SuperSpeed bulk IN throughput was measured for different USB 3.0 host controllers while keeping all other test parameters the same. The test was carried out using the USBBulkSourceSink example on multiple machines running Windows 7.

Table 9 compares the throughput of four built-in USB 3.0 host controllers. Intel's host controllers outperform other USB 3.0 host controllers by about 20 percent, as the table shows. Generally, the throughput with the USB 3.0 add-on cards will be less compared to the built-in USB 3.0 host controllers.

Table 9. Bulk IN Throughput for Host Controllers

USB 3.0 host controller (built-in)	Throughput (KB/s)	Driver version	PC information	OS	Application	Burst length	Max. packet size (bytes)	Packets/ xfer	Xfers to queue
Intel® USB 3.0 eXtensible Host Controller	450,400	1.0.9.254	Intel Core™ i5-3210M CPU, Intel 7 Series/c216 Chipset Family, 2.5 GHz, 4 GB RAM, Service Pack 1	Win 7 64 Bit	C++ Streamer	16	1024	256	64
Renesas Electronics USB 3.0 Host Controller	352,100	2.0.34.0	Intel Core i7 CPU, Chipset : Intel 3400 series, 4 GB RAM, Service Pack 1	Win 7 64 Bit	C++ Streamer	16	1024	256	64

USB 3.0 host controller (built-in)	Throughput (KB/s)	Driver version	PC information	OS	Application	Burst length	Max. packet size (bytes)	Packets/ xfer	Xfers to queue
ASMedia Host Controller	370,200	1.12.5.0	AMD FX™-4100 Quad Core Processor, 8 GB RAM, 3.60 GHz, Service Pack 1	Win 7 64 Bit	C++ Streamer	16	1024	256	64
AMD USB 3.0 Host Controller	362,900	1.1.0.153	AMD A6-3670 APU, 2.70 GHz, 8 GB RAM, Service Pack 1	Win 7 64 Bit	C++ Streamer	16	1024	256	64
NEC Electronics USB 3.0 Host Controller	300,200	1.0.19.0	Intel Core i5-2540M CPU, 2.60 GHz, 4 GB RAM, Service Pack 1	Win 7 64 Bit	C++ Streamer	16	1024 Bytes	256	64

11 Operating System Performance Comparison

The USB throughput can also depend on the operating system and the software running on the USB Host.

SuperSpeed bulk IN throughput was measured for different operating systems while keeping the other test parameters the same. This test was carried out with the USBBulkSourceSink application using the configuration that delivers the maximum data transfer rate. The Intel 3.0 host was used on all operating systems (Windows, Mac OS, and Linux).

SuperSpeed isochronous IN throughput was measured for different operating systems. This test was carried out with the USBIsoSourceSink application using the configuration that delivers the maximum data rate. The Intel 3.0 host was used on all operating systems.

A console application based on the open source libusb library was used to measure throughput with Linux and Mac OS operating systems. To use this console application, download the FX3 SDK for Linux and FX3 SDK for Mac OS from www.cypress.com/?rID=57990.

The performance comparisons are tabulated in Table 10 and Table 11. From the results, you can see that the isochronous throughput for Linux is less compared with Windows and Mac OS X. It is less because the Linux kernel restricts the size of the USB transfer to less than or equal to 32 KB. Therefore, the burst size and the ISO-Mult setting should be set in such a way that their product does not exceed 32 KB.

Table 10. Bulk IN Throughput for Operating Systems

Operating system	Windows 7 (Intel host)	Windows 8 (Intel host)	Windows 10 (Intel host)	Mac OS X v10.10 (Intel host)	Linux (kernel 3.11.0-12, Intel host)
Throughput (KB/s)	454,300	453,500	454,400	420,100	425,400

Table 11. Isochronous IN Throughput for Operating Systems

Operating system	Windows 7 (Intel host)	Windows 8 (Intel host)	Windows 10 (Intel host)	Mac OS X v10.10 (Intel host)	Linux (kernel 3.11.0-12, Intel host)
Throughput (KB/s)	359,800	359,900	359,900	356,400	256,000

12 Other Factors Affecting Throughput

As described previously, throughput varies depending on the host controller and operating system. If you connect multiple USB Devices to the same host, throughput may drop. Cypress recommends that you use a USB-IF certified USB cable to achieve the best results. USB Devices connected to host adapter cards provide a lower throughput compared with integrated USB host controllers.

In applications in which data is being transferred from/to external devices connected over the GPIF II interface, the throughput also depends on the GPIF II data transfer rate and CPU processing time if it needs to touch the data.

When using bulk endpoints for such transfers, maximize the number of DMA buffers. This helps to counteract the impact on throughput resulting from long pauses in data transfer on the USB Host side.

When using isochronous transfers, Cypress recommends that each DMA buffer be capable of holding one service interval's worth of data. Because the host guarantees the transfer bandwidth in this case, a large number of buffers is not required.

Where no data manipulation is being performed by the FX3 firmware, use AUTO DMA channels to avoid any performance limitation due to the firmware design. If firmware-based data manipulation is required, it will be faster to move the data using DMA callbacks than it will be doing it from a thread. One such example is a USB Video Class application, in which header information is added to data blocks by the firmware. In addition, Cypress strongly recommends that you keep critical data processing code to a minimum by removing debug messages. AN75779 – How to Implement an Image Sensor Interface with EZ-USB FX3 in a USB Video Class (UVC) Framework provides an example of how the firmware implements the UVC.

Using the release version of the FX3 API and RTOS libraries is recommended; set the compiler optimization level to "-O2" or "-O3."

13 Other Resources

AN65974 – Designing with the EZ-USB FX3 Slave FIFO Interface explains the details of the Slave FIFO interface and includes performance measurements for a Slave FIFO application. Refer to the section, "Steps in streaming transfers," for throughput measured using a Slave FIFO application.

Refer to section 2.4 of the FX3 SDK troubleshooting guide for details on USB link-level factors that can affect system throughput. This document is part of the FX3 SDK installation in the *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\doc* folder. For more details on the USB SuperSpeed protocol, refer to the USB 3.0 specification (http://www.usb.org/developers/docs/).

14 Summary

This application note provided guidelines on how to achieve optimum USB 3.0 throughput using EZ-USB FX3. It analyzed changes in throughput caused by various transfer parameters to identify the most critical parameters for each transfer type. It also compared the throughput for host controllers and operating systems, describing the factors that affect USB throughput.

Document History

Document Title: AN86947 - Optimizing USB 3.0 Throughput with EZ-USB® FX3™

Document Number: 001-86947

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4084414	MDDD	08/01/2013	New Specification
*A	4305036	RSKV	03/12/2014	Control Center, C++ Streamer and USB 3.0 drivers are added to the attachment Attached C++ Streamer application is modified such that **Packets per Xfer** is defaulted to **256** and **Xfers to Queue** is defaulted to **64**. Streamer application user interface is explained Table 9 is updated with the more details of test setup (host controller driver version, PC information, Streamer application settings)
*B	4864015	MDDD	07/29/2015	Added ISOC throughput comparison table across different operating systems. Sunset review
*C	5701881	BENV	04/19/2017	Updated logo and copyright
*D	5861789	MDDD	08/23/2017	Added Windows 10 throughput results in Table 10 and Table 11 Sunset review

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

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