AN65974

Designing with the EZ-USB® FX3™ Slave FIFO Interface

Author: Rama Sai Krishna V
Associated Project: Yes
Software Version: EZ-USB FX3 SDK1.3
Related Application Notes: AN75705

AN65974 describes the synchronous Slave FIFO interface of EZ-USB® FX3™. The hardware interface and configuration settings for the FLAGs are described in detail with examples. The application note includes references to GPIF™ II Designer to make the Slave FIFO interface easy to design with. Two complete design examples are provided to demonstrate how you can use the synchronous Slave FIFO to interface an FPGA to FX3.

Contents

Introduction 2
GPIF II 2
Synchronous Slave FIFO Interface 3
Difference between Slave FIFO with Two and Five Address Lines 3
Pin Mapping of Slave FIFO Interface 4
Slave FIFO Access Sequence and Interface Timing 5
Synchronous Slave FIFO Interface Timing 5
Synchronous Slave FIFO Read Sequence 5
Synchronous Slave FIFO Write Sequence 7
Threads and Sockets 8
DMA Channel Configuration 9
Flag Configuration 9
Dedicated Thread Flag 9
Current Thread Flag 10
GPIFII Designer 12
Implementing a Synchronous Slave FIFO Interface 12
Configuring a Partial FLAG 12
General Formulas for Using Partial FLAGs 15
CyU3PgpifSocketConfigure() API Usage Examples 15
Other Considerations When Using Partial FLAG 17
Error Conditions Due To Flag Violations 17
Slave FIFO Firmware Examples in the SDK 19
Design Example1: Interfacing an Xilinx FPGA to FX3's Synchronous Slave FIFO Interface 20
Hardware Setup 20
Jumper and Switch Settings 21
Firmware and Software Components 22
FX3 Firmware Details 23
FPGA Implementation Details 26
Project Operation 33
Design Example2: Interfacing an Altera FPGA to FX3's Synchronous Slave FIFO Interface 39
Hardware Setup 39
Jumper and Switch Settings 40
Firmware and Software Components 41
FX3 Firmware Details 42
FPGA Implementation Details 45
Project Operation 51
Associated Project Files 57
Summary 58
About the Author 58
Troubleshooting 59
Appendix 61
Short Packet Example 61
Zero Length Packet (ZLP) Example 62
Document History 65
Worldwide Sales and Design Support 66

Introduction

The EZ-USB FX3, Cypress's next-generation USB 3.0 peripheral controller, enables developers to add USB 3.0 functionality to any system. The controller works well with applications such as imaging and video devices, printers, and scanners.

EZ-USB FX3 has a fully configurable parallel, general programmable interface, called GPIF II, which can connect to an external processor, ASIC, or FPGA. 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 devices such as FPGAs, image sensors, and processors with interfaces such as the synchronous address data multiplexed interface.

One popular implementation of GPIF II is the synchronous Slave FIFO interface. This interface is used for applications in which the external device connected to EZ-USB FX3 accesses the FX3 FIFOs, reading from or writing data to them. Direct register access is not possible over the Slave FIFO interface.

This application note begins with a brief introduction to GPIF II and then describes the details of the synchronous Slave FIFO interface. This document also provides two complete design examples that show you how to implement a master interface compatible with synchronous Slave FIFO on an FPGA. The Verilog and VHDL files for Xilinx Spartan 6 FPGA and Altera Cyclone III FPGA are provided. The corresponding FX3 firmware project for synchronous Slave FIFO is also included as part of the example. These examples have been developed using a Xilinx SP601 evaluation kit for the Spartan 6 FPGA and an Altera Cyclone III Starter Board for the Cyclone III FPGA, an FX3 development kit (DVK), and the FX3 software development kit (SDK).

GPIF II

GPIF II is a programmable state machine that provides the flexibility of implementing an industry-standard or proprietary interface. It can function either as a master or slave.

GPIF II has the following features:

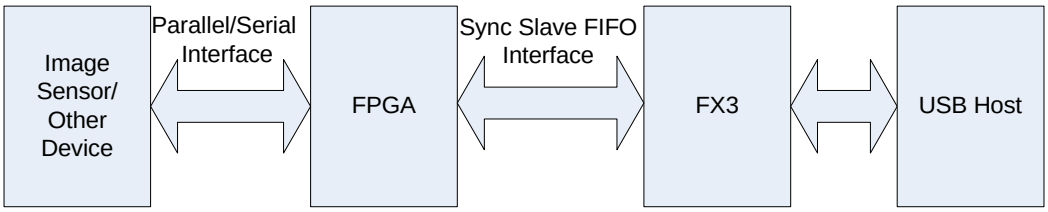
- Functions as master or slave
- Offers 256 firmware programmable states
- Supports 8-bit, 16-bit, and 32-bit parallel data bus
- Enables interface frequencies up to 100 MHz
- Supports 14 configurable control pins when a 32-bit data bus is used; all control pins can be either input/output or bidirectional
- Supports 16 configurable control pins when a 16/8 data bus is used; all control pins can be either input/output or bidirectional

GPIF II state transitions occur based on control input signals. Control output signals are driven by GPIF II state transitions. The behavior of the state machine is defined by a descriptor, which is designed to meet the required interface specifications. The GPIF II descriptor is essentially a set of programmable register configurations. In the EZ-USB FX3 register space, 8 kB is dedicated as GPIF II waveform memory, where the GPIF II descriptor is stored.

A popular implementation of GPIF II is the synchronous Slave FIFO interface, which is described in detail in the following sections.

Figure 1 shows an example application diagram where the synchronous Slave FIFO interface is used.

Figure 1. Example Application Diagram



Synchronous Slave FIFO Interface

The synchronous Slave FIFO interface is suitable for applications in which an external processor or device needs to perform data read/write accesses to EZ-USB FX3's internal FIFO buffers. Register accesses are not done over the Slave FIFO interface. The synchronous Slave FIFO interface is generally the interface of choice for USB applications, to support high throughput requirements.

Figure 2 shows the interface diagram for the synchronous Slave FIFO interface. The signals shown in Figure 2 are described in Table 1.

Figure 2. Synchronous Slave FIFO Interface Diagram

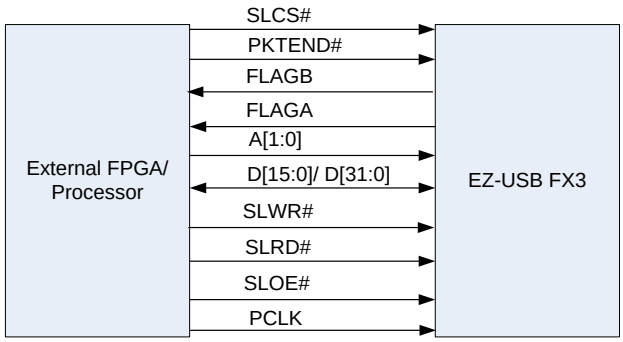


Table 1. Synchronous Slave FIIFO Interface Signals

Signal Name	Signal Description
SLCS#	This is the chip select signal for the Slave FIFO interface. It must be asserted to access Slave FIFO.
SLWR#	This is the write strobe for the Slave FIFO interface. It must be asserted for performing write transfers to Slave FIFO.
SLRD#	This is the read strobe for the Slave FIFO interface. It must be asserted for performing read transfers from Slave FIFO.
SLOE#	This is the output enable signal. It causes the data bus of the Slave FIFO interface to be driven by FX3. It must be asserted for performing read transfers from Slave FIFO.
FLAGA/ FLAGB	These are the FLAG outputs from FX3. The FLAGs indicate the availability of an FX3 socket.[1]
A[1:0]	This is the 2-bit address bus of Slave FIFO.
D[15:0]/ D[31:0]	This is the 16-bit or 32-bit data bus of Slave FIFO.
PKTEND#	This signal is asserted to write a short packet or a zero length packet to Slave FIFO.
PCLK	This is the Slave FIFO interface clock.

[1] The Threads and Sockets section explains the concept of sockets for data transfers. The FLAGs are described in detail in the Flag Configuration section.

Difference between Slave FIFO with Two and Five Address Lines

The synchronous Slave FIFO interface with two address lines supports access to up to four sockets. To access more than four sockets, the synchronous Slave FIFO interface with five address lines should be used. In addition to the extra address lines, this interface also has a signal called EPSWITCH#. Due to the increased number of pins, fewer pins are available for use as FLAGs; for this reason, the FLAG is configured as a current_thread FLAG.

Extra latencies are incurred when using the synchronous Slave FIFO interface with five address lines:

- A two-cycle latency from address to FLAG valid is incurred at the beginning of every transfer.
- Whenever a socket address is switched, multiple cycles of latency are incurred to complete the socket switching.

Due to the increased latencies and additional interface protocol requirements, it is recommended that you use the synchronous Slave FIFO interface with five address lines only if the application requires access to more than four GPIF II sockets. For more information about this interface, refer to the application note AN68829 – Slave FIFO Interface for EZ-USB FX3: 5-Bit Address Mode.

The following sections of this application note describe the synchronous Slave FIFO interface with two address lines.

Pin Mapping of Slave FIFO Interface

Table 2 shows the default pin mapping of the Slave FIFO interface. The table also shows the GPIO pins and other serial interfaces (UART/SPI/I2S) available when GPIF II is configured for the Slave FIFO interface.

The pin mapping may be changed if needed and FLAGs may be added or reconfigured using the GPIFII Designer tool. More information is provided in the section Flag Configuration.

Table 2. Pin Mapping for Slave FIFO Interface

EZ-USB FX3 Pin	Synchronous Slave FIFO Interface with 16-bit Data Bus	Synchronous Slave FIFO Interface with 32-bit Data Bus
GPIO[17]	SLCS#	SLCS#
GPIO[18]	SLWR#	SLWR#
GPIO[19]	SLOE#	SLOE#
GPIO[20]	SLRD#	SLRD#
GPIO[21]	FLAGA	FLAGA
GPIO[22]	FLAGB	FLAGB
GPIO[23]	FLAGC	FLAGC
GPIO[24]	PKTEND#	PKTEND#
GPIO[25]	FLAGD	FLAGD
GPIO[28]	A1	A1
GPIO[29]	A0	A0
GPIO[0:15]	DQ[0:15]	DQ[0:15]
GPIO[16]	PCLK	PCLK
GPIO[33:44]	Available as GPIOs	DQ[16:27]
GPIO[45]	GPIO	GPIO
GPIO[46]	GPIO/UART_RTS	DQ28
GPIO[47]	GPIO/UART_CTS	DQ29
GPIO[48]	GPIO/UART_TX	DQ30
GPIO[49]	GPIO/UART_RX	DQ31
GPIO[50]	GPIO/I2S_CLK	GPIO/I2S_CLK
GPIO[51]	GPIO/I2S_SD	GPIO/I2S_SD
GPIO[52]	GPIO/I2S_WS	GPIO/I2S_WS
GPIO[53]	GPIO/SPI_SCK /UART_RTS	GPIO/UART_RTS
GPIO[54]	GPIO/SPI_SSN/UART_CTS	GPIO/UART_CTS
GPIO[55]	GPIO/SPI_MISO/UART_TX	GPIO/UART_TX
GPIO[56]	GPIO/SPI_MOSI/UART_RX	GPIO/UART_RX
GPIO[57]	GPIO/I2S_MCLK	GPIO/I2S_MCLK

Note For the complete pin mapping of EZ-USB FX3, refer to the EZ-USB FX3 SuperSpeed USB Controller datasheet.

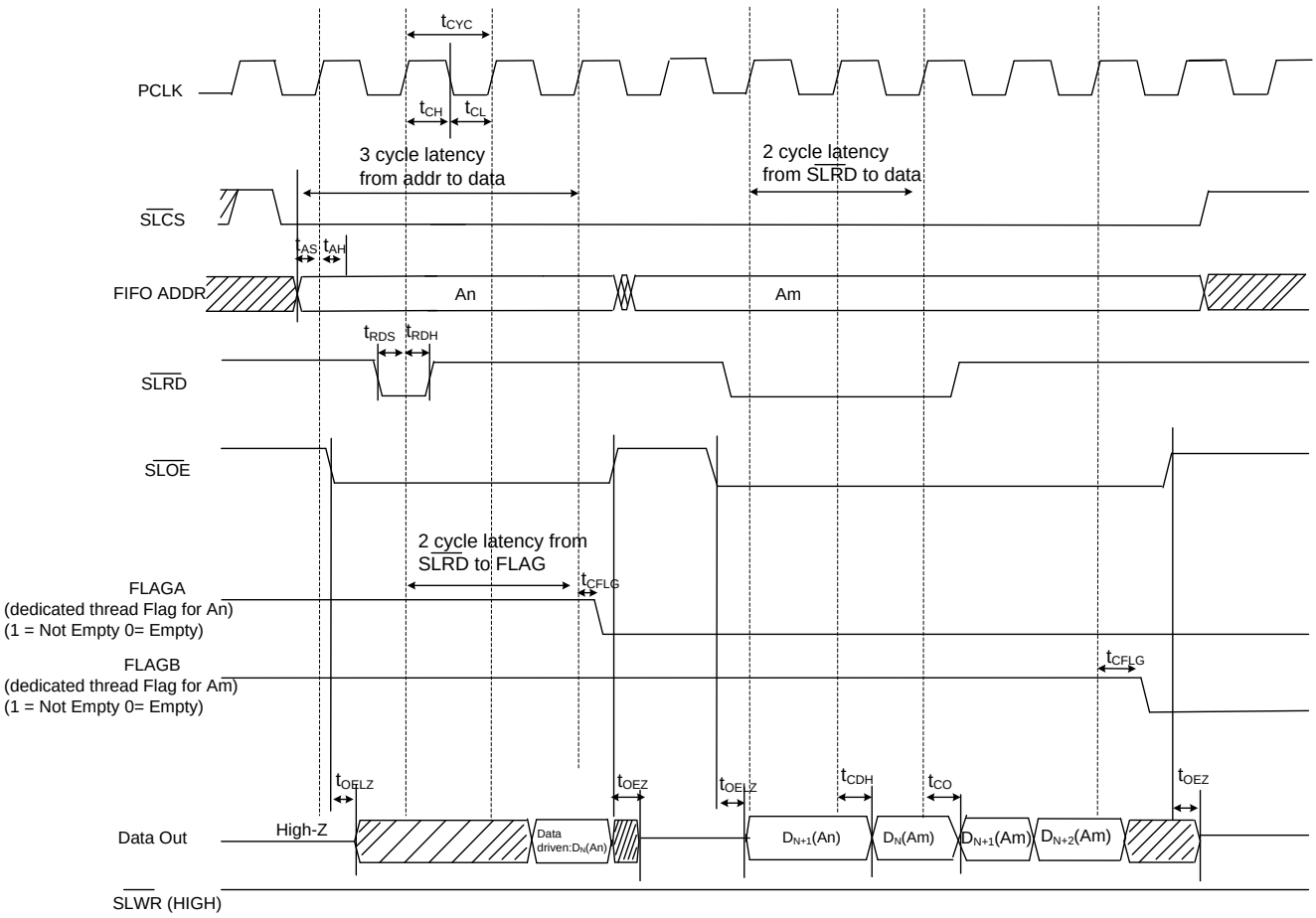
Slave FIFO Access Sequence and Interface Timing

This section describes the access sequence and timing of the synchronous Slave FIFO interface.

An external processor or device (functioning as the master of the interface) may perform single-cycle or burst data accesses to EZ-USB FX3's internal FIFO buffers. The external master drives the 2-bit address on the ADDR lines and asserts the read or write strobes. EZ-USB FX3 asserts the FLAG signals to indicate empty or full conditions of the buffer.

Synchronous Slave FIFO Interface Timing

Figure 3. Synchronous Slave FIFO Read Sequence



Synchronous Slave FIFO Read Sequence

The sequence for performing reads from the synchronous Slave FIFO interface is:

1. FIFO address is stable and SLCS# is asserted.
2. SLOE# is asserted. SLOE# is an output enable only whose sole function is to drive the data bus.
3. SLRD# is asserted.

The FIFO pointer is updated on the rising edge of the PCLK while SLRD# is asserted. This action starts the propagation of data from the newly addressed FIFO to the data bus. After a propagation delay of tCO (measured from the rising edge of PCLK), the new data value is present. N is the first data value read from the FIFO. To drive the data bus, SLOE# must also be asserted.

The same sequence of events is shown for a burst read.

Note For burst mode, the SLRD# and SLOE# remain asserted during the entire duration of the read. When SLOE# is asserted, the data bus is driven (with data from the previously addressed FIFO). For each subsequent rising edge of PCLK while the SLRD# is asserted, the FIFO pointer is incremented and the next data value is placed on the data bus.

FLAG Usage: The external processor for flow control monitors FLAG signals. FLAG signals are outputs from EZ-USB FX3 and may be configured to show empty/full/partial status for a dedicated thread or the current thread being addressed.

Figure 4. Synchronous Slave FIFO Write Sequence

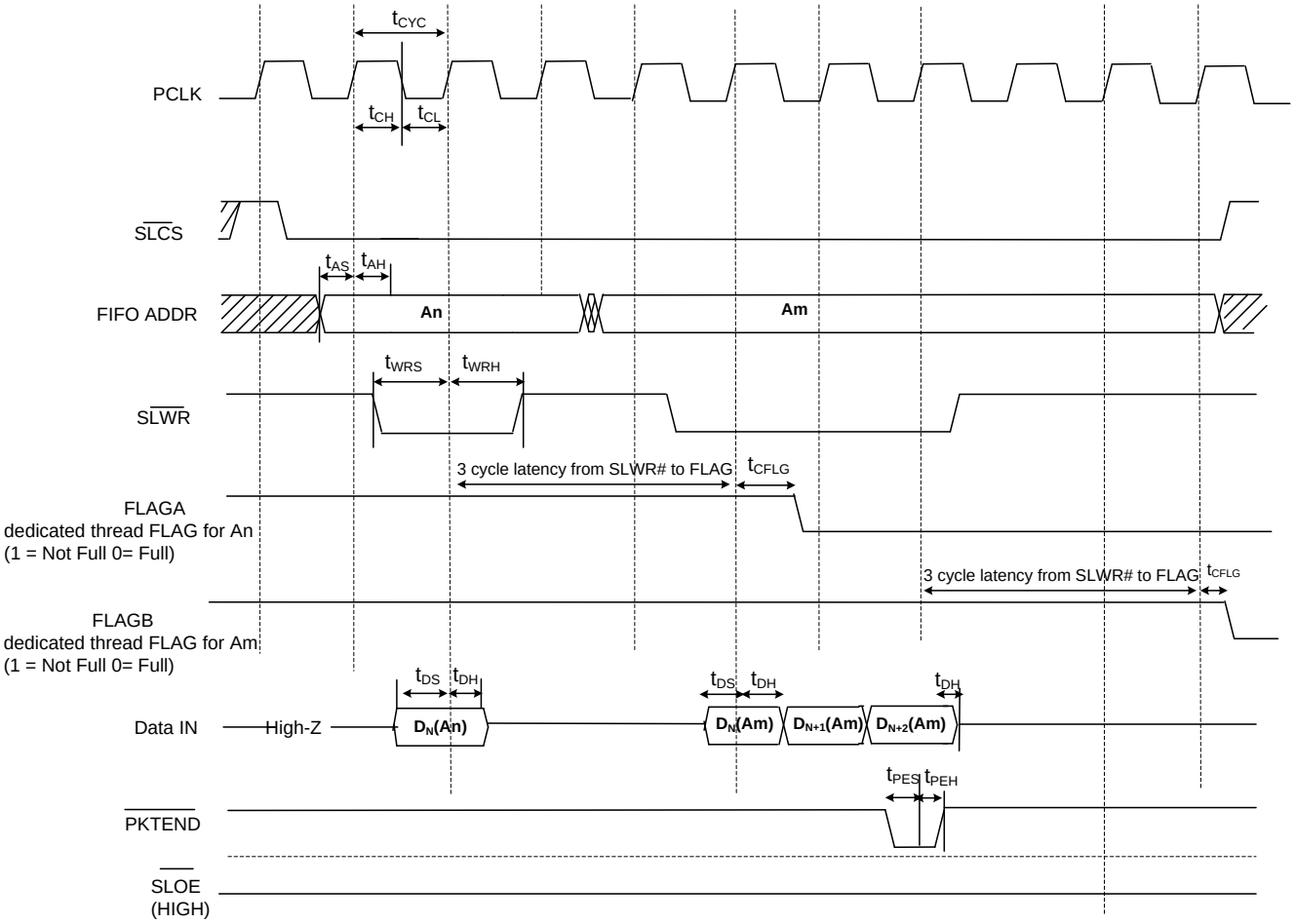
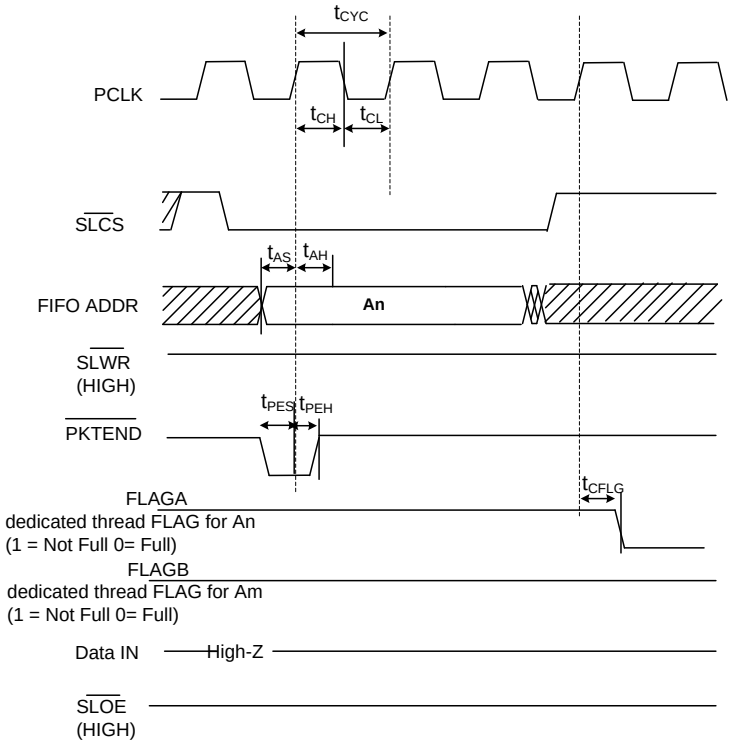


Figure 5. Synchronous ZLP Write Cycle Timing



Synchronous Slave FIFO Write Sequence

The sequence for performing writes to the synchronous Slave FIFO interface is:

1. FIFO address is stable and the signal SLCS# is asserted.
2. External master/peripheral outputs the data onto the data bus.
3. SLWR# is asserted.
4. While the SLWR# is asserted, data is written to the FIFO; on the rising edge of the PCLK, the FIFO pointer is incremented.
5. The FIFO flag is updated after a delay of tCFLG from the rising edge of the clock.

The same sequence of events is shown for a burst write.

Note For the burst mode, SLWR# and SLCS# are left asserted for the entire duration of the burst write. In the burst write mode, after the SLWR# is asserted, the value on the data bus is written into the FIFO on every rising edge of PCLK. The FIFO pointer is updated on each rising edge of PCLK.

Short Packet: A short packet can be committed to the USB host by using the PKTEND# signal. The external device/processor should be designed to assert the PKTEND# along with the last word of data and SLWR# pulse corresponding to the last word. The FIFOADDR lines must be held constant during the PKTEND# assertion. On assertion of PKTEND# with SLWR#, the GPIF II state machine interprets the packet to be a short packet and commits it to the USB interface. If the protocol does not require any short packets to be transferred, the PKTEND# signal may be pulled high.

Note that in the read direction, there is no specific signal to indicate that a short packet is sourced from the USB. The external master must monitor the empty FLAG to determine when all the data has been read.

Zero Length Packet: The external device/processor can signal a zero length packet (ZLP) by asserting PKTEND#, without asserting SLWR#. SLCS# and address must be driven, as shown in Figure 5.

FLAG Usage: The external processor monitors the FLAG signals for flow control. FLAG signals are outputs from the EZ-USB FX3 device that may be configured to show empty/full/partial status for a dedicated thread or the current thread being addressed.

Table 3. Synchronous Slave FIFO Timing Parameters

Parameter	Description	Min	Max	Unit
FREQ	Interface clock frequency	–	100	MHz
tCYC	Clock period	10	–	ns
tCH	Clock high time	4	–	ns
tCL	Clock low time	4	–	ns
tRDS	SLRD# to CLK setup time	2	–	ns
tRDH	SLRD# to CLK hold time	0.5	–	ns
tWRS	SLWR# to CLK setup time	2	–	ns
tWRH	SLWR# to CLK hold time	0.5	–	ns
tCO	Clock to valid data	–	8	ns
tDS	Data input setup time	2	–	ns
tDH	CLK to data input hold	0	–	ns
tAS	Address to CLK setup time	2	–	ns
tAH	CLK to Address hold time	0.5	–	ns
tOELZ	SLOE# to data low-Z	0	–	ns
tCFLG	CLK to flag output propagation delay	–	8	ns
tOEZ	SLOE# de-assert to data Hi Z	–	8	ns
tPES	PKTEND# to CLK setup	2	–	ns
tPEH	CLK to PKTEND# hold	0.5	–	ns
tCDH	CLK to data output hold	2	–	ns
Note Three-cycle latency from ADDR to DATA				

The following sections describe the configuration of the FLAG signals using GPIFII Designer and the EZ-USB FX3 SDK. Before describing the various FLAG configurations, it is important to introduce the concept of threads, sockets, and DMA channel.

Threads and Sockets

This section briefly explains the concepts that are needed for data transfers in and out of FX3:

- Socket
- DMA descriptor
- DMA buffer
- GPIF thread

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

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

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

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

Alternatively, the GPIF socket can send an interrupt to the FX3 CPU to notify it that the GPIF socket filled a DMA buffer. The FX3 CPU can relay this information to the USB socket. The USB socket can send an interrupt to the FX3 CPU to notify it that the USB socket emptied a DMA buffer. Then, the FX3 CPU can relay this information back to the GPIF socket. This is called the manual DMA channel implementation. This implementation is used when the FX3 CPU has to add, remove, or modify data in a data stream.

A socket that writes data to a DMA buffer is called a producer socket. A socket that reads data from a DMA buffer is called a consumer socket. A socket uses the values of the DMA buffer address, DMA buffer size, and DMA descriptor chain stored in a DMA descriptor for data management. A socket takes a finite amount of time (up to a few microseconds) to switch from one DMA descriptor to another after it fills or empties a DMA buffer. The socket cannot transfer any data while this switch is in progress.

EZ-USB FX3 provides four physical hardware threads for data transfer over the GPIF II. At a time, any one socket is mapped to a physical thread. By default, PIB socket 0 is mapped to thread 0, PIB socket 1 is mapped to thread 1, PIB socket 2 is mapped to thread 2, and PIB socket 3 is mapped to thread 3.

Note that the address signals A1:A0 on the interface indicate the thread to be accessed. FX3's DMA fabric then routes the data to the socket mapped to that thread. Therefore, when A1:A0 = 0, thread 0 is accessed, and any data that is transferred over thread 0 is routed to socket 0. Similarly, when A1:A0 = 1, data is transferred in and out of socket 1.

Note The Slave FIFO interface has only two address lines; hence, only up to four sockets may be accessed. To access more than four sockets, use the Slave FIFO interface with five address lines. Refer to application note AN68829 – Slave FIFO Interface for EZ-USB FX3: 5-Bit Address Mode.

The sockets to be accessed must be specified by configuring a DMA channel.

DMA Channel Configuration

The firmware must configure a DMA channel with the required producer and consumer sockets.

If data is to be transferred from the Slave FIFO interface to the USB interface, then P-port is the producer and USB is the consumer, and vice-versa.

So if data is to be transferred in both directions over the Slave FIFO interface, two DMA channels should be configured, one with P-port as the producer and another with P-port as the consumer.

The P-port producer socket is the socket that the external device will write to over the Slave FIFO interface and the P-port consumer socket is the one that the external device will read from over the Slave FIFO interface.

The P-port socket number in the DMA channel should be the socket number that will be addressed on A1:A0.

Multiple buffers can be allocated to a particular DMA channel when configuring the channel. Note that the FLAGs will indicate full/empty on a per buffer basis. (The maximum buffer size for any one buffer is 64 kB -16.)

For example, if two buffers of 1024 bytes are allocated to a DMA channel, the full FLAG will indicate full when 1024 bytes have been written into the first buffer. It will continue to indicate full until the DMA channel has switched to the second buffer. The time taken for the DMA channel to switch to the next buffer is not deterministic, although it is typically a few microseconds. The external master must monitor the FLAG to determine when the switching is complete and the next buffer has become available for data access.

The next section describes how FLAGs may be configured to indicate the status of different threads.

Flag Configuration

Flags may be configured as empty, full, partially empty, or partially full signals. These are not controlled by the GPIF II state machine, but by the DMA hardware engine internal to EZ-USB FX3. Flags are associated with specific threads or the currently addressed thread and indicate the status of the socket mapped to that thread.

Flags indicate empty or full, based on the direction of the socket (configured during socket initialization). Therefore, a flag indicates empty/not empty status if data is being read out of the socket and full/not full status if data is being written into the socket.

The different types of FLAGs that can be used are:

- Dedicated thread flag (empty/full or partially empty/full)
- Current thread flag (empty/full or partially empty/full)

The different types of FLAGs are described in the following sections. Different FLAG configurations result in different latencies, which are summarized in Table 4.

Dedicated Thread Flag

A flag can be configured to indicate the status of a particular thread. In this case, that flag is dedicated only to that thread and always indicates the status of the socket mapped to that particular thread only, irrespective of which thread is being addressed on the address bus.

In this case, the external processor/device must keep track of which flag is dedicated to which thread and monitor the correct flag every time a different thread is addressed.

For example, if FLAGA is dedicated to thread 0, and FLAGB is dedicated to thread 1, when the external processor accesses thread 0, it must monitor FLAGA.

When the external processor accesses thread 1, it must monitor FLAGB.

A flag may be dedicated for every thread that is going to be accessed. If the application needs to access four threads, then there may be four corresponding flags.

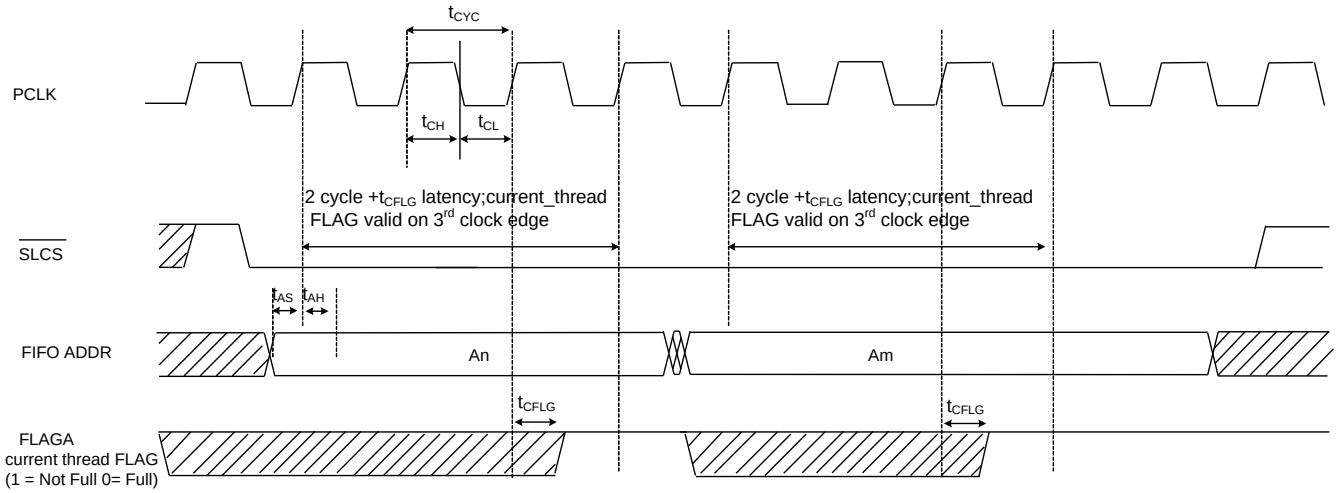
Note that when performing write transfers, a three-cycle latency for the flag is always incurred at the end of the transfer. The three-cycle latency is from the write cycle that causes the buffer to become full to the time the flag is asserted low. At the fourth clock edge, the external master can sample the flag low. This is shown in Figure 4.

When performing read transfers, a two-cycle latency for the flag is always incurred at the end of the transfer. The two-cycle latency is from the read (last SLRD# assertion) cycle that causes the buffer to become empty to the time the flag is asserted low. At the third clock edge, the external master can sample the flag low. This is shown in Figure 3.

Current Thread Flag

A flag can be configured to indicate the status of the currently addressed thread. In this case, the GPIF II state machine samples the address on the address bus and then updates the flags to indicate the status of that thread. This configuration requires fewer pins, because a single “current_thread” flag can be used to indicate the status of all four threads. However, two-cycle latency is incurred when the current_thread flag is used for a synchronous Slave FIFO interface because the GPIF II first must sample the address and then update the flag. The two-cycle latency starts when a valid address is presented on the interface. On the third clock edge after this, the valid state of the FLAG of the newly addressed thread can be sampled. (Note that the Slave FIFO descriptors included in the SDK use the “current_thread” flag configuration.)

Figure 6. Additional Latency Incurred at Start of Transfer When Using a Current Thread FLAG



Note When performing write transfers, a three-cycle latency is always incurred at the end of the transfer. The three-cycle latency is from the write cycle that causes the buffer to become full to the time the flag is asserted low. At the fourth clock edge, the external master can sample the flag low. This is shown in Figure 4.

When performing read transfers, a two-cycle latency for the flag is always incurred at the end of the transfer. The two-cycle latency is from the read (last SLRD# assertion) cycle that causes the buffer to become empty to the time the flag is asserted low. At the third clock edge, the external master can sample the flag low. This is shown in Figure 3.

Partial FLAG

A flag can be configured to indicate the partially empty/full status of a socket. A watermark value must be selected such that the flag is asserted when the number of 32-bit words that may be read or written is less than or equal to the watermark value.

Note The latency for a partial FLAG depends on the watermark value specified for the partial FLAG.

Table 4 summarizes the latencies incurred when using different FLAG configurations. The table also shows the setting that must be selected in GPIFII Designer for a particular FLAG. Examples and screenshots of the GPIFII Designer settings for FLAGs are available in the section Flag Configuration.

Table 4. Latencies Associated with Different FLAG Configuration

FLAG Configuration	GPIFII Designer FLAG Setting Selection	Address to FLAG Latency at Start of Transfer	FLAG Latency at End of Transfer		Additional API Call Required
			For Write Transfers to Slave FIFO (latency from last SLWR# assertion to full FLAG assertion)	For Read Transfers from Slave FIFO (latency from last SLRD# assertion to empty FLAG assertion)	
Full/Empty FLAG dedicated to a specific thread “n”	Thread_n_DMA_Ready	0 cycles	3 cycles + tCFLG (external device can sample valid FLAG on fourth clock edge)	2 cycles + tCFLG (external device can sample valid FLAG on third clock edge)	N/A
Full/Empty FLAG for currently addressed thread	Current_thread_DMA_Ready	2 cycles + tCFLG (external device can sample valid FLAG on thrid clock edge)	3 cycles + tCFLG (external device can sample valid FLAG on fourth clock edge)	2 cycles + tCFLG (external device can sample valid FLAG on third clock edge)	N/A
Partially full/empty FLAG dedicated to a specific thread “n”	Thread_n_DMA_Watermark	0 cycles	Dependent on watermark level	Dependent on watermark level	Set watermark level by calling the CyU3SocketConfigure() API. **Note** Watermark is in terms of a 32-bit data word. Examples: CyU3PgpifSocketConfigure (0,PIB_SOCKET_0,4,CyFalse,1) sets the watermark for thread 0 to 4 CyU3PgpifSocketConfigure (3,PIB_SOCKET_3,4,CyFalse,1) sets the watermark for thread 3 to 4
Partially full/empty FLAG for currently addressed thread	Current_thread_DMA_Watermark	2 cycles + tCFLG (external device can sample valid FLAG on 3rd clock edge)	Dependent on watermark level	Dependent on watermark level	Set watermark level by calling the CyU3SocketConfigure() API. **Note** Watermark is in terms of a 32-bit data word. Examples: CyU3PgpifSocketConfigure (0,PIB_SOCKET_0,4,CyFalse,1) sets the watermark for thread 0 to 4 CyU3PgpifSocketConfigure (3,PIB_SOCKET_3,4,CyFalse,1) sets the watermark for thread 3 to 4

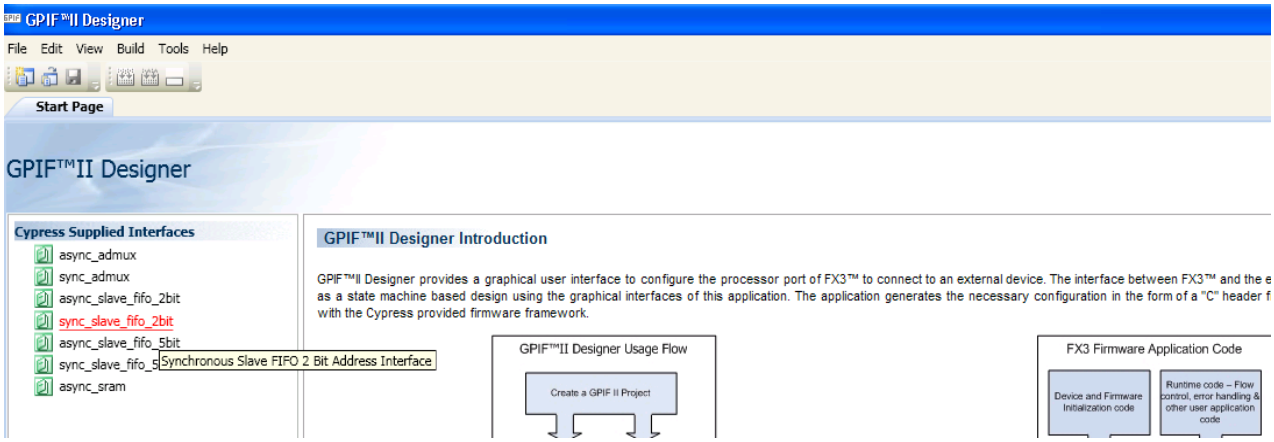
The next sections describe how to configure FLAGs using the GPIFII Designer tool and the EZ-USB FX3 SDK.

GPIFII Designer

Implementing a Synchronous Slave FIFO Interface

You will find the GPIF II implementation of the Slave FIFO interface by installing the GPIFII Designer tool. You can install the GPIFII Designer tool from Cypress's website at www.cypress.com. When you launch GPIFII Designer, you will find the **Cypress Supplied Interfaces** on the Start Page.

Figure 7. Slave FIFO Projects in GPIFII Designer – Cypress Supplied Interfaces



The sync_slave_fifo_2bit project is the GPIF II implementation of the synchronous Slave FIFO interface with a 2-bit address. The following section explains how a partial FLAG may be configured using GPIFII Designer.

Configuring a Partial FLAG

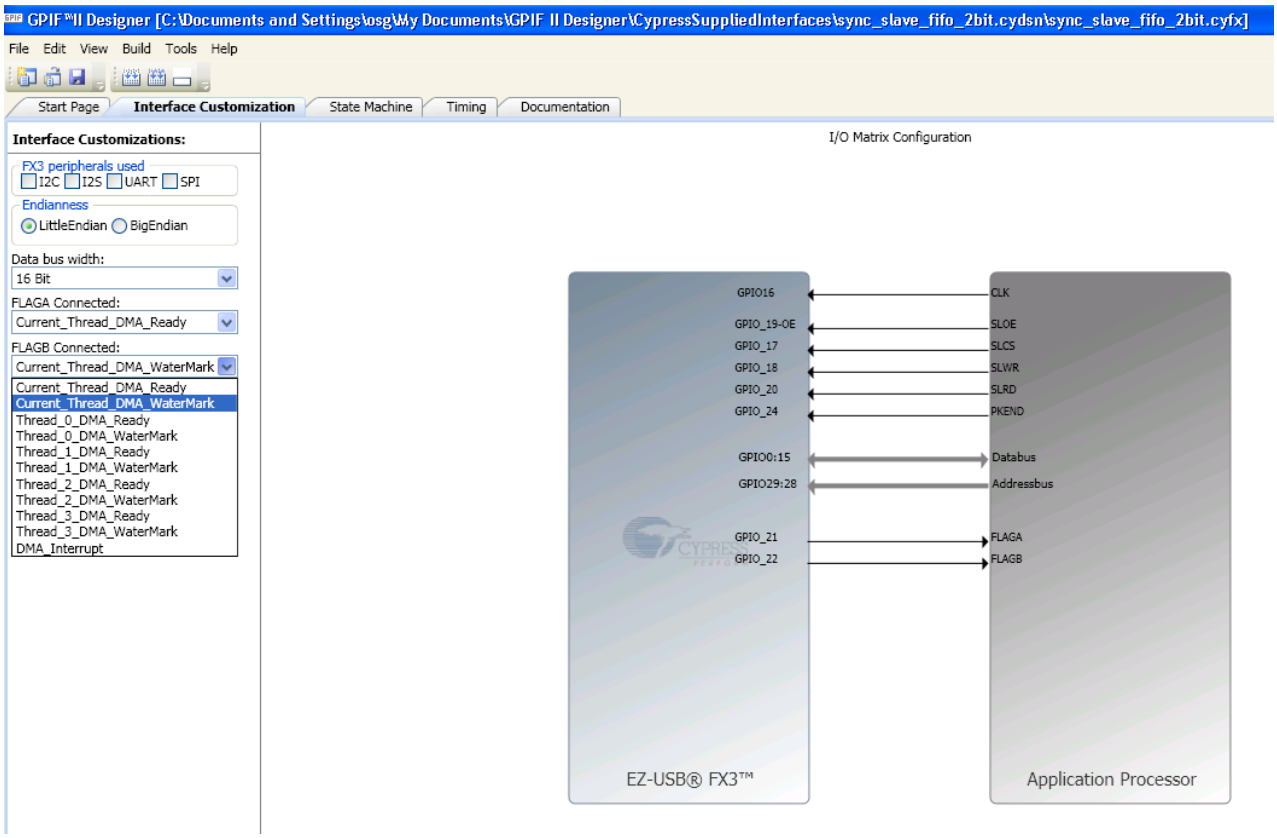
A partial FLAG is configured using the following steps:

- The partial FLAG setting must be selected in the GPIFII Designer tool.
- The watermark level for the partial FLAG must be set using the the CyU3PgpifSocketConfigure() API in firmware

In the GPIFII Designer, open the **sync_slave_fifo_2bit** project from Cypress Supplied Interfaces; under FLAGA Connected or FLAGB Connected, select **Current_Thread_DMA_Watermark** to configure the FLAG as a partial FLAG for current thread.

Or select **Thread_n_DMA_Watermark** to configure the FLAG as a partial FLAG dedicated for thread "n".

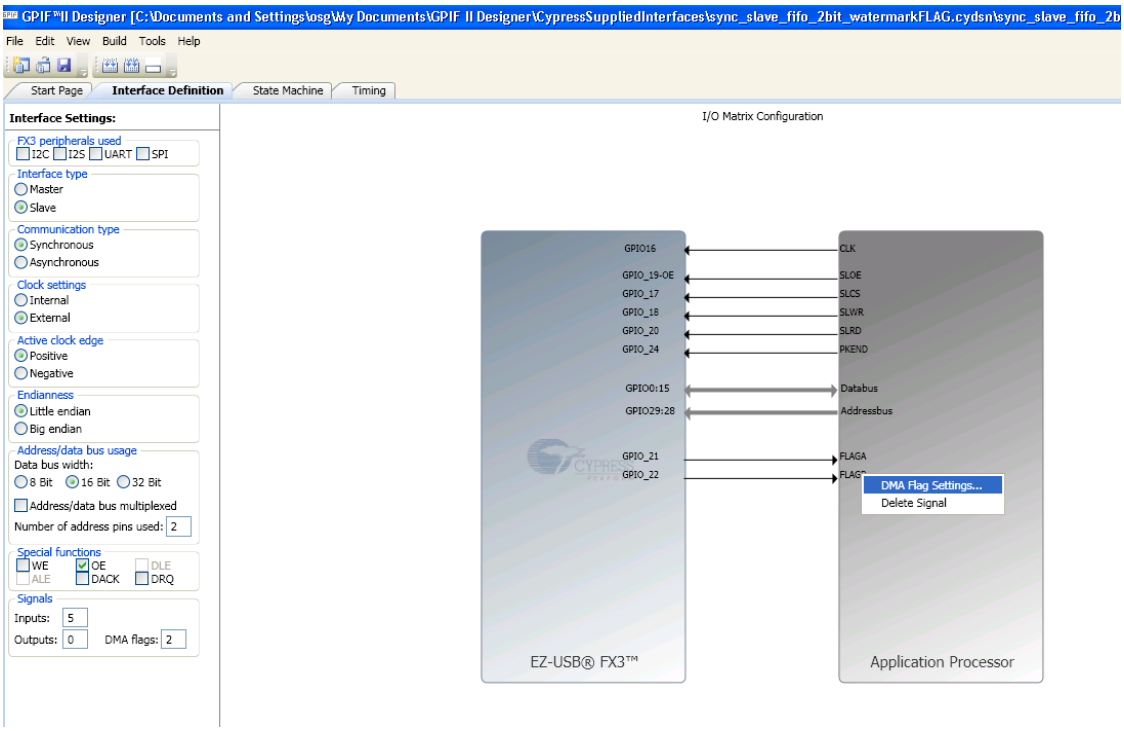
Figure 8. FLAG Settings in GPIFII Designer – Cypress Supplied Interfaces sync_slave_fifo_2bit



To add more FLAGs or make changes other than what is allowed in the *sync_slave_fifo_2bit.cyfx* project, click **File > Save Project as Editable**. This allows you to save the project under a different name, after which you can make changes to the newly saved project.

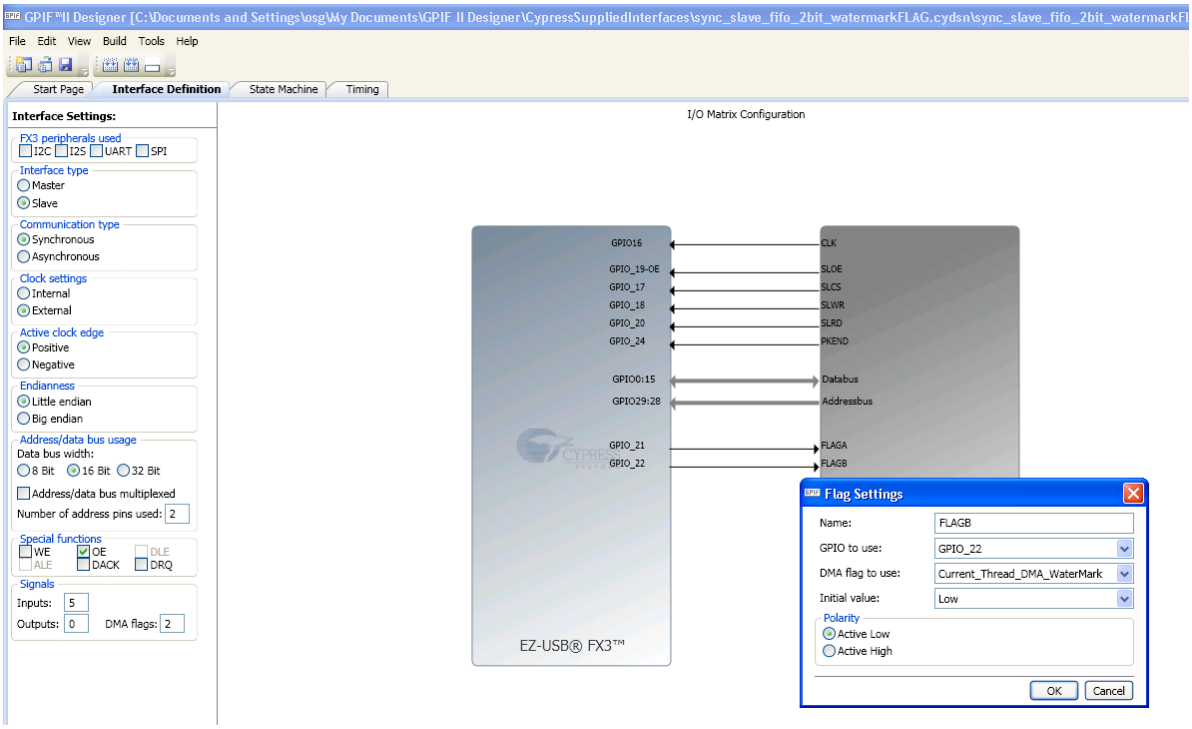
In this case, to configure a FLAG as a partial FLAG, right-click on the FLAG in the I/O Matrix Configuration diagram. Click **DMA Flag Settings**, as shown in the following figure.

Figure 9. FLAG Settings in New Project by “Save Project as Editable” on the Default sync_slave_fifo_2bit



The following figure shows you how to select the FLAG configuration:

Figure 10. Selecting Specific FLAG Settings



The second step to configure a partial FLAG is to specify a watermark value for the flag in the firmware project. In the *cyfxslfifo.c* file, add a call to the CyU3PgpifSocketConfigure() API to specify the watermark value. This call may be added just after the call to the CyU3PGPIFLoad() API. Refer to the EZ-USB FX3 SDK API Guide for a complete description of the CyU3PgpifSocketConfigure() API. One of the parameters input to this API is the watermark value.

The watermark value determines when a partial FLAG will be asserted. The following section describes the formula to calculate the number of data words that may be read or written after the partial FLAG is asserted.

General Formulas for Using Partial FLAGs

The previous sections described the possible configurations for FLAGs and the steps to configure a partial FLAG. This section explains how a watermark value should be determined for a partial FLAG.

The following formulas should be used to calculate the number of data words that may be read or written after the partial FLAG is asserted.

Note The watermark number specified in the CyU3PgpifSocketConfigure() API is in terms of a 32-bit data word.

1. When writing from an external master to the synchronous Slave FIFO:

   (a) The number of data words that may be written after the clock edge at which the partial FLAG is sampled low = watermark x (32/bus width) – 4

2. When reading into an external master from the synchronous Slave FIFO:

   (a) The number of data words available for reading (while keeping SLOE# asserted) after the clock edge at which the partial FLAG is sampled asserted = watermark x (32/bus width) – 1

   (b) There is already two-cycle latency from SLRD# to the data. Hence, the number of cycles for which SLRD# may be kept asserted after the clock edge at which the partial FLAG is sampled asserted = watermark x (32/bus width) –3.

CyU3PgpifSocketConfigure() API Usage Examples

This section provides some examples of the effect of the watermark value specified by using the CyU3PgpifSocketConfigure() API. Screenshots are provided to clearly show the behavior of a partial FLAG for different watermark values.

Note In these examples, the FLAG polarities are set to active low. Therefore, the FLAGs go low to indicate full/empty or partially full/empty.

Example 1

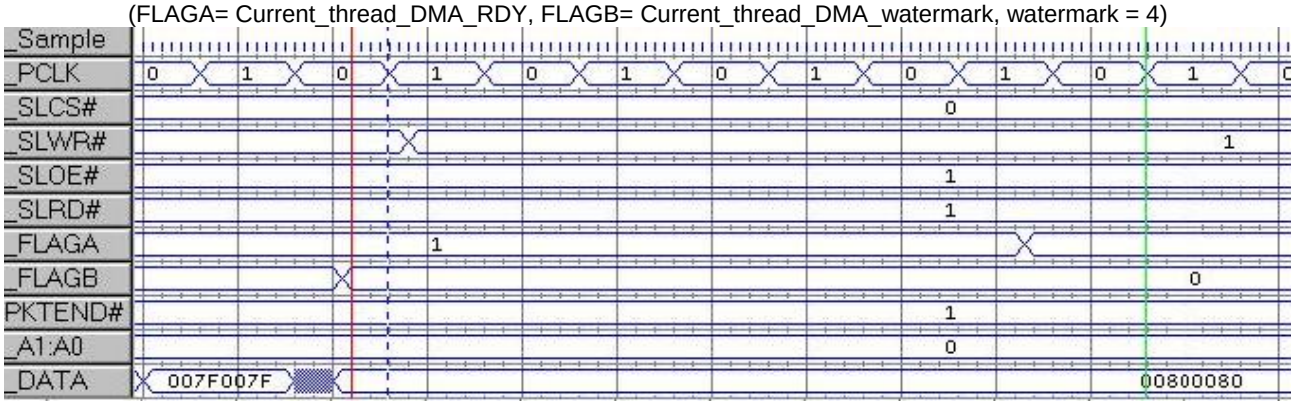
Slave FIFO with 32-bit data bus:

- In GPIFII Designer, FLAGA is configured as Current_thread_DMA_RDY and FLAGB is configured as Current_thread_DMA_watermark.
- CyU3PgpifSocketConfigure (0,PIB_SOCKET_0,4,CyFalse,1).
- Burst write is performed from the external FPGA to EZ-USB FX3 over Slave FIFO (last data to be written is 0x00800080).

The following figure is a logic analyzer screenshot of how the FLAGs go to 0 at the end of the transfer. You can see that FLAGB (partial FLAG) goes low in the same cycle in which the last word of data is written.

Figure 11. Burst Write Transfer with 32-Bit Data Bus Width

(FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 4)



Example 2

Slave FIFO with 32-bit data bus:

- In GPIFII Designer, FLAGA is configured as Current_thread_DMA_RDY and FLAGB is configured as Current_thread_DMA_watermark.
- CyU3PgpifSocketConfigure (3, PIB_SOCKET_3, 4, CyFalse, 1).
- Burst read is performed by external FPGA from EZ-USB FX3 over Slave FIFO (last data to be read is 0x00000080).

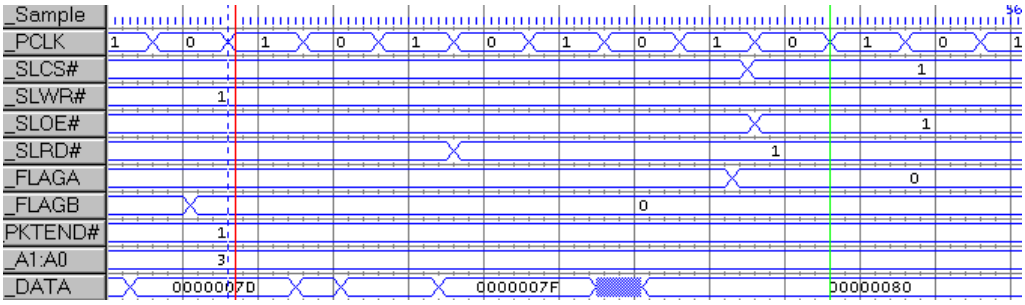
The following figure is a logic analyzer screenshot of how the FLAGs go to 0 at the end of the transfer. You can see that FLAGB (partial FLAG) goes low four cycles before the last data is read. That is, three words of data are available to be read out after the cycle in which FLAGB goes low.

Formula 2(a) from the section on General Formulas for Using Partial FLAGs can be applied to this example as follows:

Watermark value = 4, bus width = 32

Therefore, the number of 32-bit data words available for reading after the clock edge at which the partial FLAG is sampled asserted = 4 x (32/32) – 1 = 3

Figure 12. Burst Read Transfer with 32-Bit Data Bus Width
(FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 4)



Example 3

Slave FIFO with 16-bit data bus:

- In GPIFII Designer, FLAGA is configured as Current_thread_DMA_RDY and FLAGB is configured as Current_thread_DMA_watermark.
- CyU3PgpifSocketConfigure (0, PIB_SOCKET_3, 3, CyFalse, 1).
- Burst write is performed from external FPGA to EZ-USB FX3 over Slave FIFO. (last data to be written is 0x0200).

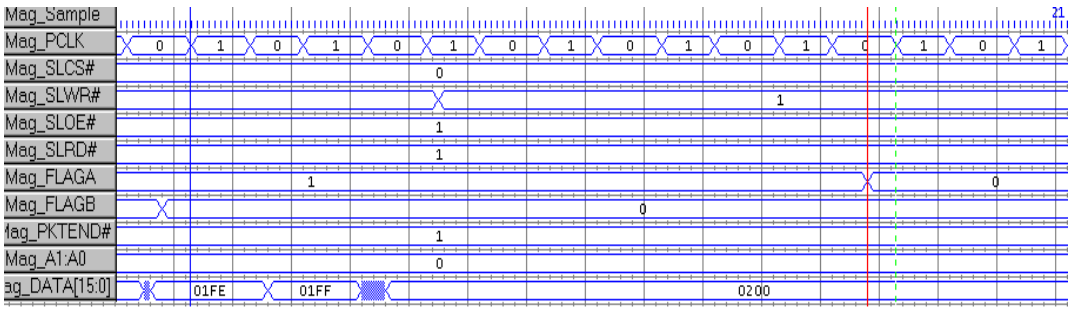
The following figure is a logic analyzer screenshot of how the FLAGs go to 0 at the end of the transfer. You can see that FLAGB (partial FLAG) goes low three cycles before the last data. This means, two words of data may be written after the cycle in which FLAGB goes low.

Formula 1 from the section on General Formulas for Using Partial FLAGs can be applied to this example as follows:

Watermark value = 3, bus width = 16

Therefore, the number of 16-bit data words that may be written after the clock edge at which the partial FLAG is sampled asserted = 3 x (32/16) – 4 = 2

Figure 13. Burst Write Transfer with 16-Bit Data Bus Width
(FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 3)



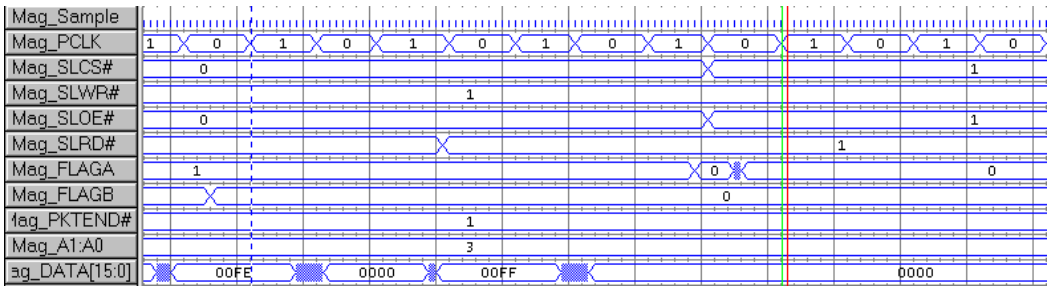
Example 4

Slave FIFO with 16-bit data bus:

- In GPIFII Designer, FLAGA is configured as Current_thread_DMA_RDY and FLAGB is configured as Current_thread_DMA_watermark.
- CyU3PgpifSocketConfigure (3, PIB_SOCKET_3, 2, CyFalse, 1).
- Burst read is performed by the external FPGA from EZ-USB FX3 over Slave FIFO (last data to be read is 0x0000).

The following figure is a logic analyzer screenshot of how the FLAGs go to 0 at the end of the transfer. You can see that FLAGB (partial FLAG) goes low four cycles before the last data. That is, three words of data may be read after the cycle in which FLAGB goes low.

Figure 14. Burst Read Transfer with 16-Bit Data Bus Width
(FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 2)



Other Considerations When Using Partial FLAG

- A partial FLAG may only be used to decide when to end a transfer. The full/empty FLAG must be monitored at the start of transfer to ensure availability of the socket. This means that a partial FLAG cannot be used by itself; it must be used in conjunction with a full/empty FLAG.
- The use of a partial FLAG may be completely avoided if the external master can implement a counting mechanism and always write an amount of data that equals the size of EZ-USB FX3's DMA buffer. The external master should count the data being written or read and ensure that it does not exceed the buffer size set up when creating the DMA channel. In this case, a full/empty FLAG should be monitored to decide when to begin a transfer.
- If a counting mechanism is not implemented as described in the previous step, and a partial FLAG is used, you need to do one of the following steps:
  - If the external master always bursts a fixed amount of data, this burst size must be considered when selecting a watermark value. For example, if the external master always writes in bursts of eight words, then the watermark value may be set such that the partial FLAG goes low when there is space for a burst of eight in EZ-USB FX3's DMA buffer. Then, having seen the partial FLAG as low, the external master can write one complete burst of eight. To achieve this, for the write (to Slave FIFO) direction in 16-bit mode, set the watermark value to '6'.
  - An alternative to the previous step is that after the partial FLAG goes to 0, instead of performing a burst access, the external master can switch to single cycle access mode. At each cycle, before doing a write, the external master can check the full/empty FLAG to ensure that the buffer still has space.

Error Conditions Due To Flag Violations

A data read or write access to the Slave FIFO interface must not be done when the partial FLAG or full/empty FLAG indicates that a buffer is not available.

If a read access is performed on an empty buffer, a buffer under-run error will occur. If a write access is performed on a full buffer, a buffer over-run error will occur. These errors can lead to data corruption at buffer boundaries. The Slave FIFO interface is in the PIB block domain of FX3 and any errors related to the interface are indicated by a PIB error.

If a PIB error occurs, a PIB interrupt will be triggered. The FX3 SDK allows a callback function to be registered for PIB interrupts. The callback function for the PIB interrupt can check the PIB error code. The following code is an example of how a callback function can be registered and the actual callback function that checks for under-run/over-run errors and prints out a debug message.

The error code indicates the thread on which the error occurred, as shown in Table 5. If one of these errors does occur, Cypress recommends that you analyze the interface timing carefully using a logic analyzer, paying special attention to the number of read or write cycles being executed after the partial FLAG goes to 0. For examples, see Figure 11 to Figure 14.

```
/* Register a callback for notification of PIB interrupts*/
CyU3PpibRegisterCallback(gpif_error_cb,intMask);

/* Callback function to check for PIB ERROR*/
void gpif_error_cb(CyU3PpibIntrType cbType, uint16_t cbArg)
{
       if(cbType==CYU3P_PIB_INTR_ERROR)
       {
              switch(CYU3P_GET_PIB_ERROR_TYPE(cbArg))
               {
                      case CYU3P_PIB_ERR_THR0_WR_OVERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR0_WR_OVERRUN");
                      break;
                      case CYU3P_PIB_ERR_THR1_WR_OVERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR1_WR_OVERRUN");
                      break;
                       case CYU3P_PIB_ERR_THR2_WR_OVERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR2_WR_OVERRUN");
                      break;
                      case CYU3P_PIB_ERR_THR3_WR_OVERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR3_WR_OVERRUN");
                      break;

                      case CYU3P_PIB_ERR_THR0_RD_UNDERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR0_RD_UNDERRUN");
                      break;
                      case CYU3P_PIB_ERR_THR1_RD_UNDERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR1_RD_UNDERRUN");
                      break;
                      case CYU3P_PIB_ERR_THR2_RD_UNDERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR2_RD_UNDERRUN");
                      break;
                      case CYU3P_PIB_ERR_THR3_RD_UNDERRUN:
                             CyU3PdebugPrint (4, "CYU3P_PIB_ERR_THR3_RD_UNDERRUN");
                      break;

                      default:
                             CyU3PdebugPrint (4, "No Underrun/Overrun Error");
                      break;
              }
        }
}
```

Table 5. PIB Error Codes for Over-run/Under-run Conditions

PIB Error Code	Description
CYU3P_PIB_ERR_THR0_WR_OVERRUN	Write overrun on thread 0 buffer
CYU3P_PIB_ERR_THR1_WR_OVERRUN	Write overrun on thread 1 buffer
CYU3P_PIB_ERR_THR2_WR_OVERRUN	Write overrun on thread 2 buffer
CYU3P_PIB_ERR_THR3_WR_OVERRUN	Write overrun on thread 3 buffer
CYU3P_PIB_ERR_THR0_RD_UNDERRUN	Read under-run on thread 0 buffer
CYU3P_PIB_ERR_THR1_RD_UNDERRUN	Read under-run on thread 1 buffer
CYU3P_PIB_ERR_THR2_RD_UNDERRUN	Read under-run on thread 2 buffer
CYU3P_PIB_ERR_THR3_RD_UNDERRUN	Read under-run on thread 3 buffer

Slave FIFO Firmware Examples in the SDK

This application note has described the synchronous Slave FIFO interface and how to configure FLAGs. After making the required configurations in the GPIFII Designer tool, the updated configuration needs to be integrated into the firmware. After building the project in GPIFII Designer, a header file *cyfxgpifconfig.h* is generated. This header file must be included in the firmware project. The EZ-USB FX3 SDK includes a firmware example that integrates the Slave FIFO interface.

After you install the EZ-USB FX3 SDK, the firmware example that integrates the synchronous Slave FIFO interface becomes available in the following directory: *[FX3 SDK Install Path]\EZ-USB FX3 SDK\1.2\firmware\slavefifo_examples\slfifosync.*

This firmware example supports both 16- and 32-bit data bus width. The constant CY_FX_SLFIFO_GPIF_16_32BIT_CONF_SELECT is defined in the header file *cyfxslfifosync.h*. To select the 32-bit data bus width, set this constant to '1'; to select the 16-bit data bus width, set this constant to '0'.

Note If the Slave FIFO should function at 100 MHz with 32-bit, you must configure the PLL frequency to 400 MHz. To do this, set the setSysClk400 parameter as an input to the CyU3PdeviceInit() function. For more information, refer to the API Guide available with the FX3 SDK.

Design Example1: Interfacing an Xilinx FPGA to FX3's Synchronous Slave FIFO Interface

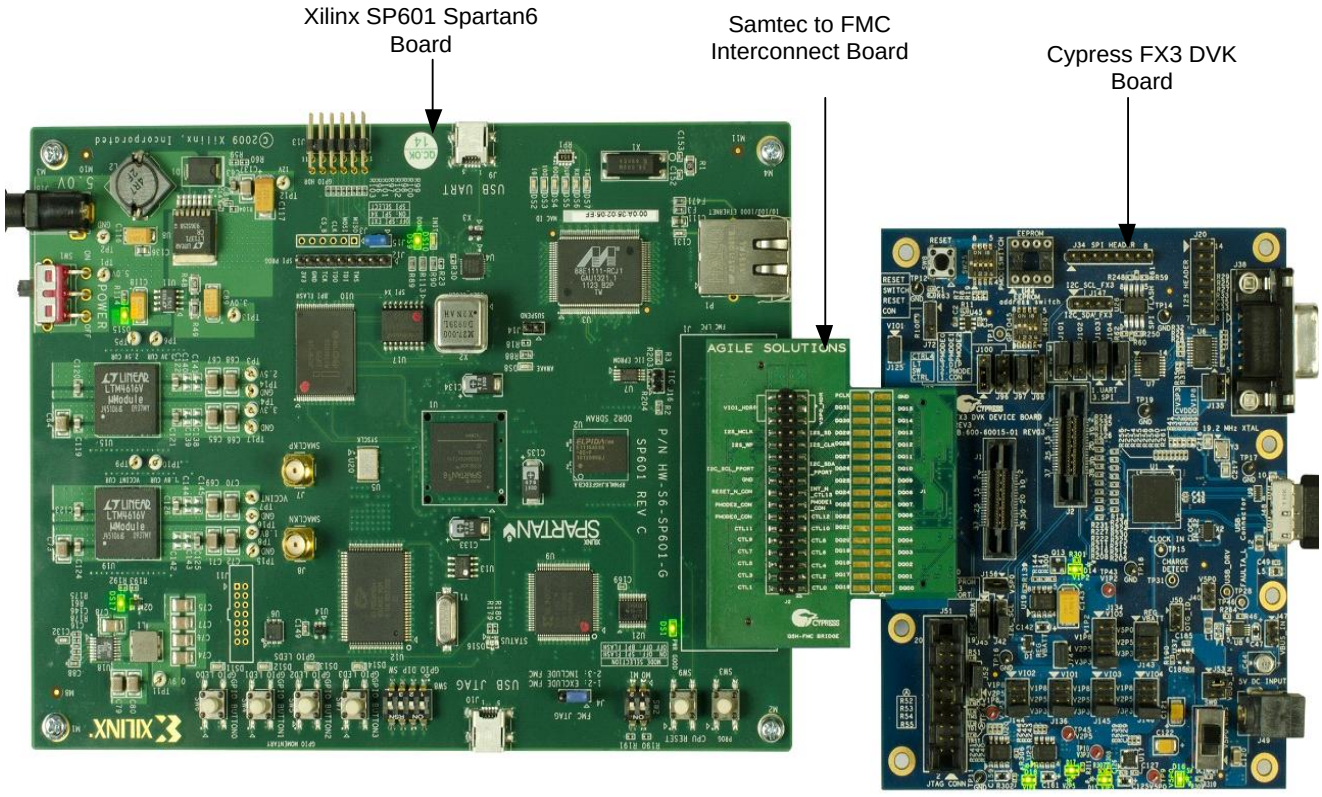
This section provides a complete design example in which a Xilinx Spartan 6 FPGA is connected to FX3 over the synchronous Slave FIFO interface. The hardware, firmware, and software components used to implement this design are discussed.

Hardware Setup

The project provided in this example can be executed on a hardware setup consisting of a Cypress FX3 DVK board interconnected with a Xilinx Spartan 6 SP601 evaluation kit. The FX3 board and the Xilinx board are connected using a Samtec to FMC interconnect board. The interconnect board mates with the Samtec connector on the Cypress FX3 DVK board and the FMC connector on the Xilinx board. Contact Cypress (fx3@cypress.com) to get the schematics of the Samtec-to-FMC interconnect board.

Figure 15 shows a picture of this hardware setup.

Figure 15. Cypress FX3 DVK Board Connected to Xilinx SP601 Board using FMC Interconnect Board



Jumper and Switch Settings

The FX3 DVK board jumper and switch settings to run this demo are listed in Table 6. Switch settings of the Xilinx Spartan 6 SP601 evaluation kit to select different transfer modes are listed in Table 7.

Table 6. FX3 DVK Jumper and Switch Settings

Sl. No.	Jumper/Switch	Pins to be Shorted using Jumpers	Function
1	J100	1 and 2	GPIO[21]/CTL[4] – configured as FLAGA
2	J136	3 and 4	VIO1(3.3V)
3	J144	3 and 4	VIO2(3.3V)
4	J145	3 and 4	VIO3(3.3V)
5	J146	3 and 4	VIO4(3.3V)
6	J134	4 and 5	VIO5(3.3V)
7	J135	2 and 3	CVDDQ(3.3V)
8	J143	3 and 4	VBATT(3.3V)
9	J101	1 and 2	GPIO[46] = UART_RTS
10	J102	1 and 2	GPIO[47] = UART_CTS
11	J103	1 and 2	GPIO[48] = UART_TX
12	J104	1 and 2	GPIO[49] = UART_RX
13	J96 and SW25	2 and 3	PMODE0 pin state (ON/OFF) selection using SW25. SW25.1 should be OFF
14	J97 and SW25	2 and 3	PMODE1 pin state (ON/OFF) selection using SW25. SW25.1 should be OFF
15	J98	1 and 2	PMODE2 pin floating
16	J72	1 and 2	RESET
17	J53	1 and 2	Bus powered
18	SW9	The switch should point to the direction labeled VBUS_IN	Bus powered
19	J156	Place a jumper to short	Powers Samtec connector
20	J45	2 and 3	GPIO[59] - Reset to the FPGA from FX3

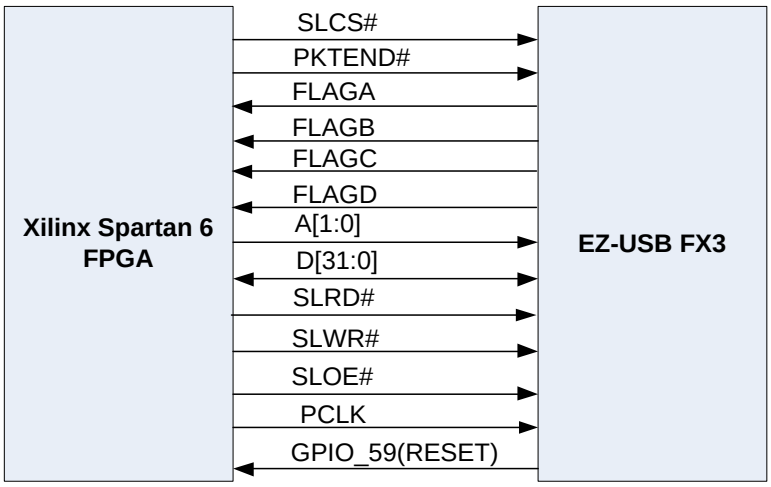
Note PMODE pins are set for USB boot. The jumpers that are not listed in the table can be left opened.

Firmware and Software Components

- FX3 synchronous Slave FIFO firmware project available with the FX3 SDK
- Control Center and Streamer software utilities available with the FX3 SDK

The following figure shows the interconnect diagram between the FPGA and FX3.

Figure 16. Interconnect Diagram between FPGA and FX3



The example includes the following components:

- Loopback transfer: In this component, the FPGA first reads a complete buffer from FX3 and then writes it back to FX3. The USB host should issue OUT/IN tokens to transmit and then receive this data. You can use the Control Center utility provided with the EZ-USB FX3 SDK for this purpose.
- Short packet: In this component, the FPGA transfers a full packet followed by a short packet to FX3. The USB host should issue IN tokens to receive this data.
- Zero length packet (ZLP) transfer: In this component, the FPGA transfers a full packet followed by a zero length packet to FX3. The USB host should issue IN tokens to receive this data.
- Streaming (IN) data transfer: In this component, the FPGA does one-directional transfers, that is, continuously writes data to FX3 over synchronous Slave FIFO. The USB host should issue IN tokens to receive this data. You can use the Control Center or Streamer utility provided with the EZ-USB FX3 SDK for this purpose.
- Streaming (OUT) data transfer: In this component, the FPGA does one-directional transfers, that is, continuously reads data from FX3 over synchronous Slave FIFO. The USB host should issue OUT tokens to provide this data. You can use the Control Center or Streamer utility provided with the EZ-USB FX3 SDK for this purpose.

FX3 Firmware Details

FX3 firmware is based on the example project contained in the FX3 SDK.

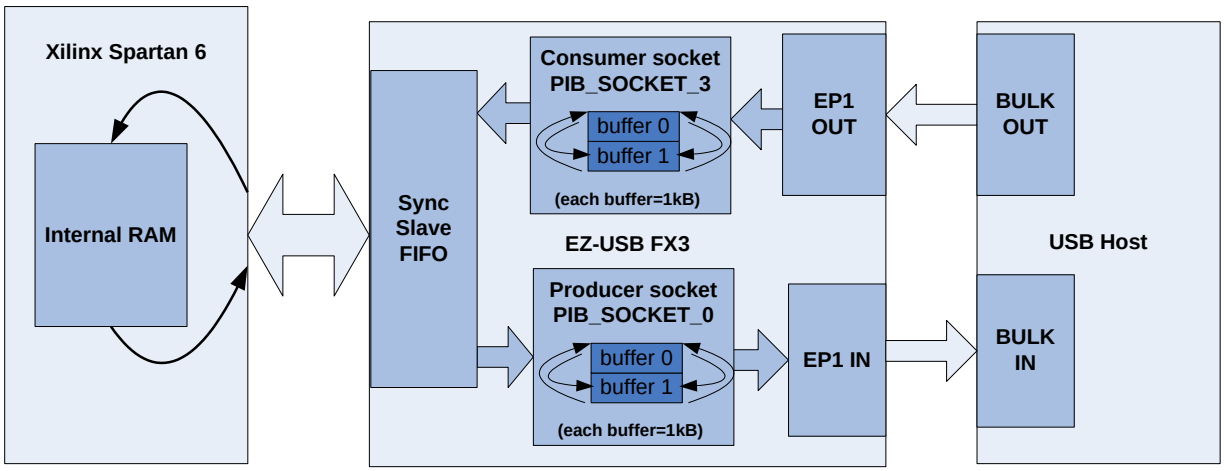
The main features of this firmware are:

- Enables both USB 3.0 and USB 2.0.
- Enumerates with the Cypress VID/PID, 0x04B4/0x00F1. This enables the use of the Cypress Control Center and Streamer utilities for initiating USB transfers.
- Integrates the synchronous Slave FIFO descriptor, which:
  - Supports access to up to four sockets
  - Configures data bus width to 32-bit
  - Works on the 100-MHz PCLK input clock
  - Configures four flags:
    i. FLAGA: Full flag dedicated to thread0
    ii. FLAGB: Partial flag with watermark value 6, dedicated to thread0
    iii. FLAGC: Empty flag dedicated to thread3
    iv. FLAGD: Partial flag with watermark value 6, dedicated to thread3

  Note The GPIFII Designer project provided with this application note demonstrates these settings. In addition, the firmware project shows the usage of the CyU3PgpifSocketConfigure() API to configure the watermark value.
- Configures the PLL frequency to 400 MHz. This is done by setting the setSysClk400 parameter as an input to the CyU3PdeviceInit() function.

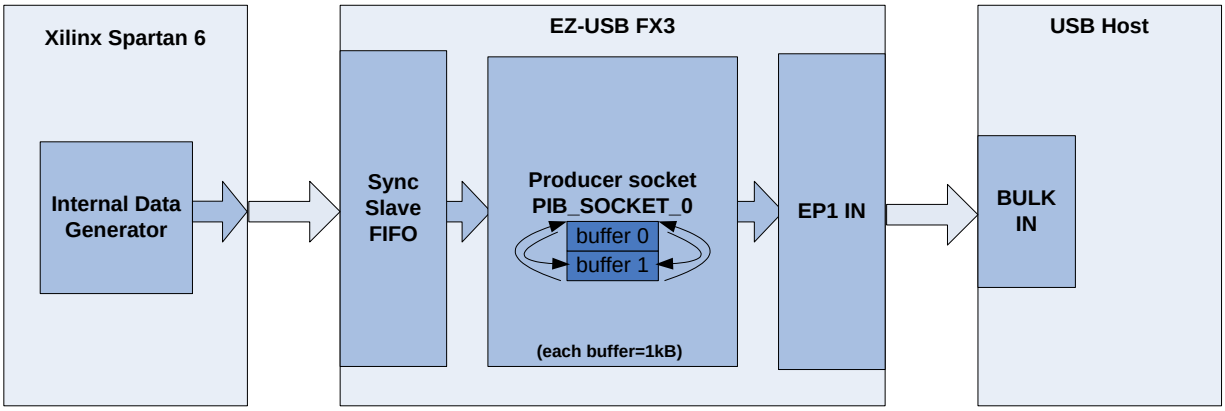
  Note This setting is essential for the functioning of Slave FIFO at 100 MHz with 32-bit data.
- Sets up the DMA channels:
  - For loopback transfers, short packet, and ZLP transfer, two DMA channels are created:
    i. A P2U channel with PIB_SOCKET_0 as the producer and UIB_SOCKET_1 as the consumer. The DMA buffer size is 512 or 1024 depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 2.
    ii. A U2P channel with PIB_SOCKET_3 as the consumer and UIB_SOCKET_1 as the producer. The DMA buffer size is 512 or 1024 depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 2.

Figure 17. Setup for Loopback Transfer



Note Only the P2U channel is used for short packets and ZLPs.

Figure 18. Setup for Short Packet and ZLP Transfers

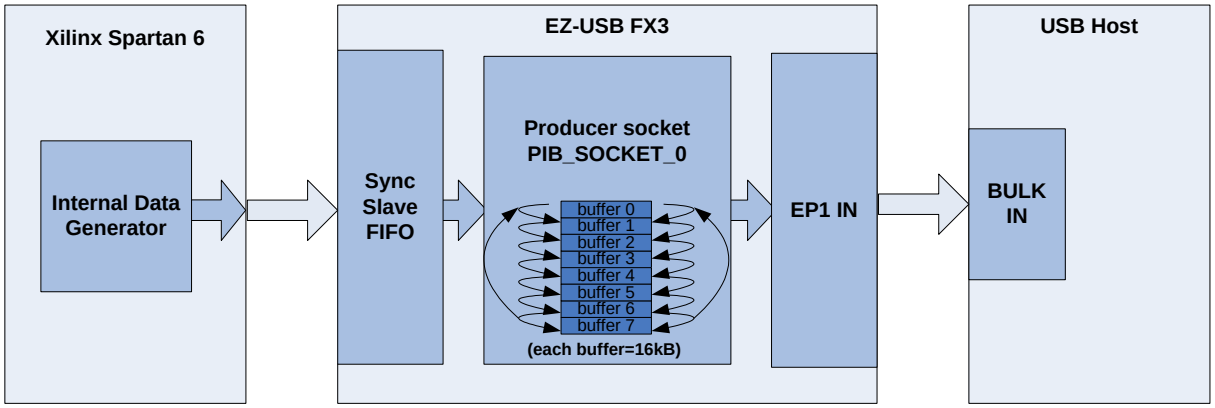


The DMA channels described in this section are set up if the following define is enabled in the *cyfxslfifosync.h* file in the FX3 firmware project provided with this application note.

```
/* set up DMA channel for loopback/short packet/ZLP transfers */
#define LOOPBACK_SHRT_ZLP
```

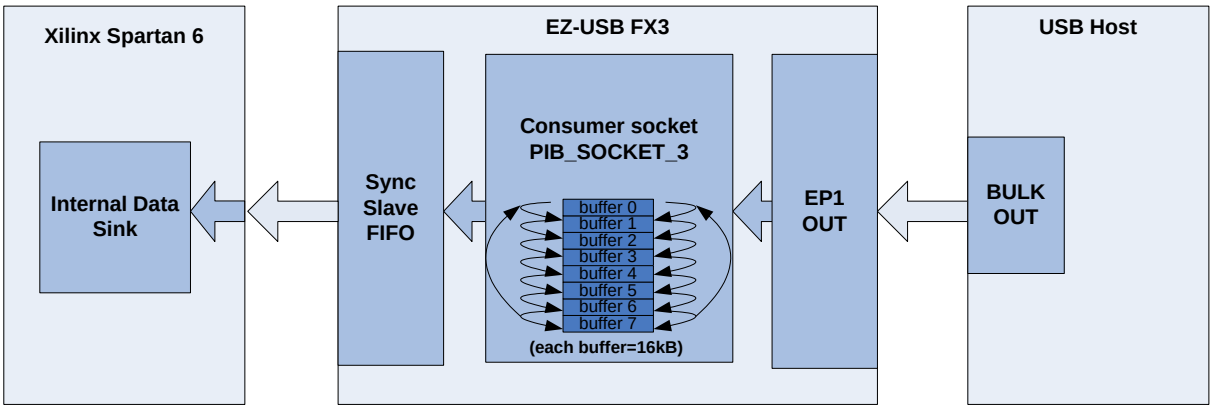
- For streaming, two DMA channels are created:
  i. A P2U channel with PIB_SOCKET_0 as the producer and UIB_SOCKET_1 as the consumer. The DMA buffer size is 16×1024 (for a USB 3.0 connection) or 16×512 (for a USB 2.0 connection) depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 8. This buffer size and count is chosen to provide high throughput performance.

Figure 19. Stream IN Transfer Setup – Buffer Count and Size Optimized for Performance



  ii. A U2P channel with PIB_SOCKET_3 as the consumer and UIB_SOCKET_1 as the producer. The DMA buffer size is 16×1024 (for a USB 3.0 connection) or 16×512 (for a USB 2.0 connection). The DMA buffer count is 4. The buffer count can be increased to enhance performance, but the buffer count of the P2U channel should be reduced. This is because the FX3 SDK does not provide enough buffer memory such that both channels can have a buffer size of 16×1024 and buffer count of 8.

Figure 20. Stream OUT Transfer Setup – Buffer Count and Size Optimized for Performance



The DMA channels described in this section are set up if the following define is enabled in the *cyfxslfifosync.h* file in the FX3 firmware project provided with this application note.

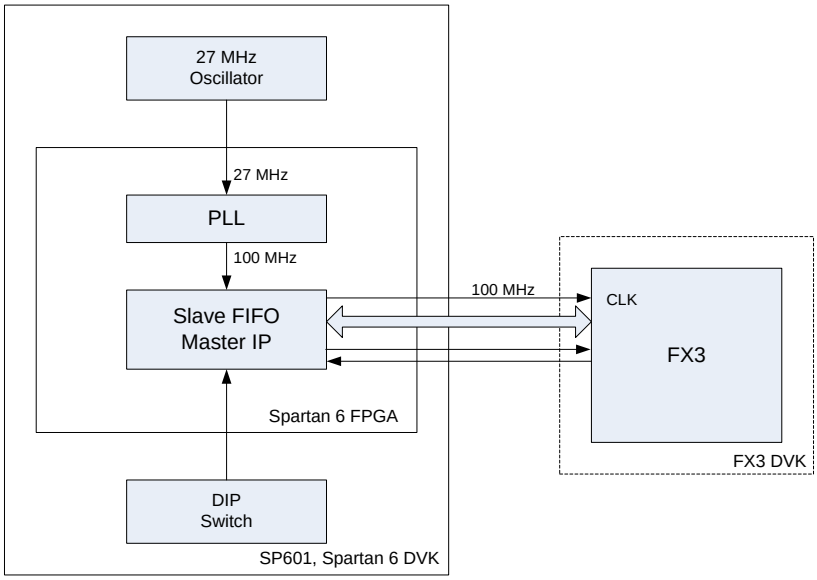
```
/* set up DMA channel for stream IN/OUT transfers */
#define STREAM_IN_OUT
```

The buffer count allocated to the P2U and U2P DMA channels can be controlled by using the following defines, also present in the *cyfxslfifosync.h* file:

```
/* Slave FIFO P_2_U channel buffer count */
#define CY_FX_SLFIFO_DMA_BUF_COUNT_P_2_U      (4)
/* Slave FIFO U_2_P channel buffer count */
#define CY_FX_SLFIFO_DMA_BUF_COUNT_U_2_P      (8)
```

FPGA Implementation Details

Figure 21. Xilinx Spartan 6 (XC6SLX16) FPGA Implementation using SP601 Evaluation Kit



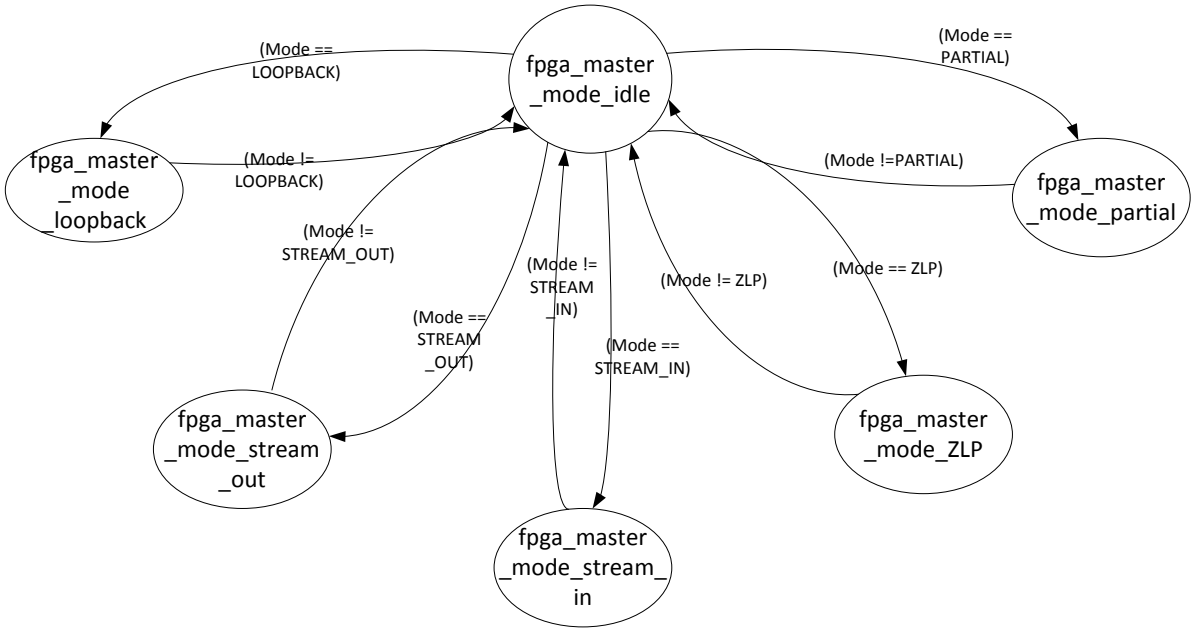
To demonstrate the maximum performance of FX3, the GPIF interface runs at 100 MHz. The SP601 has an onboard 27-MHz single-ended oscillator. The FPGA uses a PLL to generate a 100-MHz clock from the 27-MHz clock.

The state implementations of the different types of transfers are described in the following sections.

FPGA Master Mode State Machine

A state machine is implemented to select the transfer mode of the FPGA master. The four transfer modes are Loopback, Short Packet (Partial), Zero-length Packet, Stream IN, and Stream OUT transfers.

Figure 22. FPGA State Machine for Mode Selection



State fpga_master_mode_idle:

If transfer mode is not selected, FPGA master remains in this state.

State fpga_master_mode_partial:

If mode = PARTIAL, the state machine will enter this state. If mode ! = PARTIAL, the state machine will enter in the fpga_master_mode_idle state from this state.

State fpga_master_mode_zlp:

If mode = ZLP, the state machine will enter this state. If mode ! = ZLP, the state machine will enter in the fpga_master_mode_idle state from this state.

State fpga_master_mode_stream_in:

If mode = STREAM_IN, the state machine will enter this state. If mode ! = STREAM_IN, the state machine will enter in the fpga_master_mode_idle state from this state.

State fpga_master_mode_stream_out:

If mode = STREAM_OUT, the state machine will enter this state. If mode ! = STREAM_OUT, the state machine will enter in the fpga_master_mode_idle state from this state.

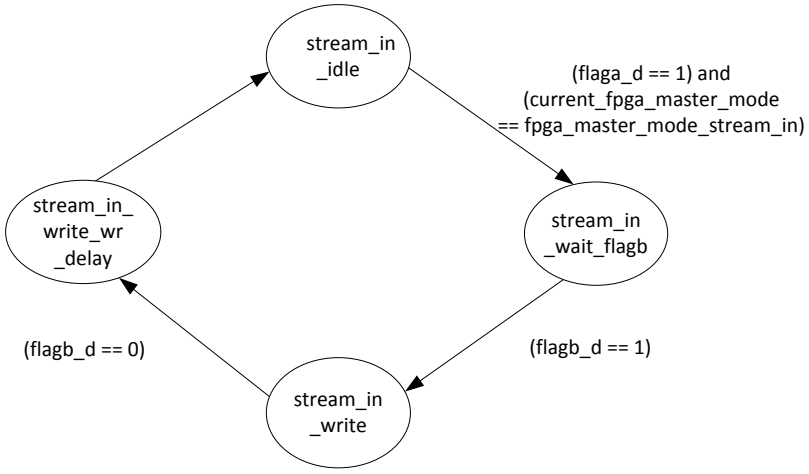
State fpga_master_mode_loop_back:

If mode = LOOPBACK, the state machine will enter this state. If mode ! = LOOPBACK, the state machine will enter in the fpga_master_mode_idle state from this state.

Stream IN Example [FPGA writing to Slave FIFO]

The state machine implemented in Verilog RTL for the stream IN transfers is shown in the following figure.

Figure 23. FPGA State Machine for Stream IN



State stream_in_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State stream_in_wait_flagb:

Whenever flaga_d = 1 and FPGA master mode is stream_in, the state machine will enter this state and wait for flagb_d.

State stream_in_write:

Whenever flagb_d = 1, the state machine will enter this state and start writing to the Slave FIFO interface. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State stream_in_write_wr_delay:

Whenever flagb_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

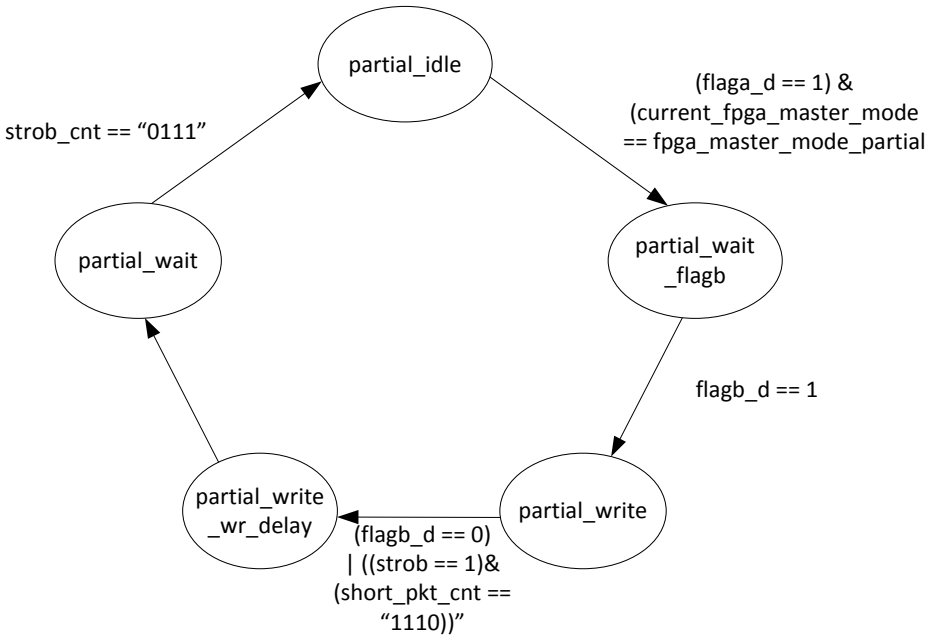
After one clock cycle, the state machine will enter the stream_in_idle state.

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the flagb_d (flopped output of flagb) as 0.

Short Packet Example [FPGA writing full packet followed by short packet to Slave FIFO]

This example demonstrates the short packet commit procedure using PKTEND#. The state machine implemented in Verilog RTL for the short-packet example is shown in the following figure.

Figure 24. State Machine for Short Packet Transfer



State partial_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State partial_wait_flagb:

Whenever flaga_d = 1 and FPGA master mode is partial, the state machine will enter this state.

State partial_write:

Whenever flagb_d = 1, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State partial_write_wr_delay:

Whenever flagb_d = 0 or (strob = 1 and short_pkt_cnt = ”1110”), the state machine will enter this state. If strob = 1, FPGA master will commit a short packet.

The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

After one clock cycle, the state machine will enter the partial_wait state.

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0.

State partial_wait:

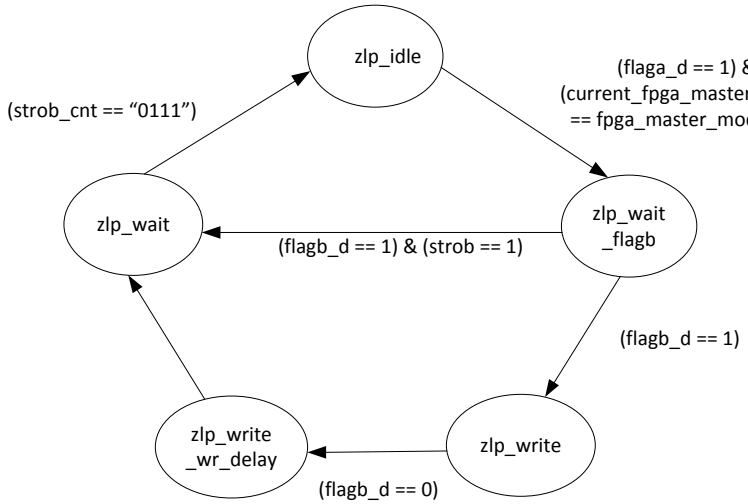
Whenever strob_cnt = 0111, the state machine will enter the partial_idle state.

As the watermark value is 6, flaga is expected to go to 0, only six clock cycles after the partial flag (flagb). This state holds the execution for more than four clock cycles to ensure the availability of a valid status on flaga.

Zero Length Packet Example [FPGA writing full packet followed by ZLP to Slave FIFO]

This example demonstrates the ZLP commit procedure using PKTEND#. The state machine implemented in Verilog RTL for the ZLP example is shown in the following figure.

Figure 25. State Machine for ZLP Transfer



State zlp_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State zlp_wait_flagb:

Whenever flaga_d = 1 and FPGA master mode is zlp, the state machine will enter this state.

State zlp_write:

Whenever flagb_d = 1, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State zlp_write_wr_delay:

Whenever flagb_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

After one clock cycle, the state machine will enter the zlp_wait state.

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0

State zlp_wait:

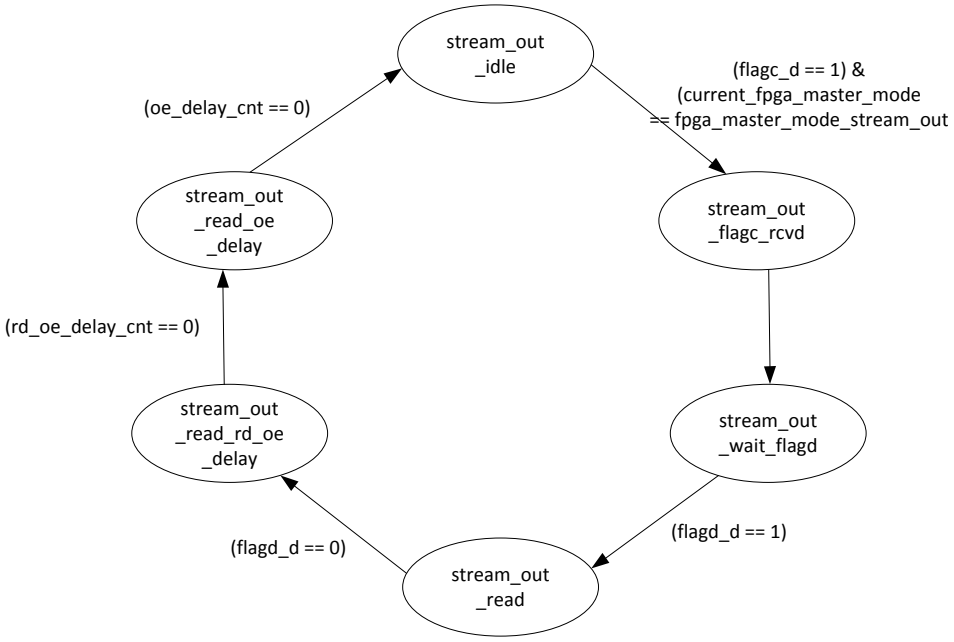
Whenever flagb_d = 1 and strob = 1, the state machine will enter this state from the zlp_wait_flagb state and commit a zlp packet into slavefifo.

As the watermark value is 6, flaga is expected to go to 0, only six clock cycles after the partial flag (flagb). This state holds the execution for more than four clock cycles to ensure the availability of a valid status on flaga.

Whenever strob_cnt = 0111, the state machine will enter the zlp_idle state.

State Machine Implemented in Verilog RTL for Stream OUT Example

Figure 26. State Machine for Stream OUT Transfer



State stream_out_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State stream_out_flagc_rcvd:

Whenever flagc_d = 1 and FPGA master mode is stream out, the state machine will enter this state.

State stream_out_wait_flagd:

After one-clock cycle, the state machine will enter this state.

State stream_out_read:

Whenever flagc_d = 1, the state machine will enter this state. Here the state machine will assert read control signals as:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

State stream_out_read_rd_oe_delay:

Whenever flagc_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

According to formula (2b) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLRD# asserted for three cycles after the partial flag (flagd) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLRD# for a count of one cycle after sampling flagd_d (flopped output of flagd) as 0.

State stream_out_read_oe_delay:

Whenever rd_oe_dealy_cnt = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

If oe_delay_cnt = 0, the state machine will enter the stream_out_idle state from this state.

Loopback Example [FPGA reading from Slave FIFO and writing the same data back to Slave FIFO]

The state machine moves through six states before completing one loopback cycle. The state machine along with the corresponding actions is shown in the following figure.

Figure 27. State Machine for Loopback Transfer

State loop_back_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_flagc_rcvd:

Whenever the flagc_d = 1 and FPGA master mode is loop back, the state machine will enter this state.

State loop_back_wait_flagd:

After one clock cycle, the state machine will enter this state and wait for flagd.

State loop_back_read:

If flagd_d = 1, the state machine will enter this state. Here the state machine will assert read control signals as:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_read_rd_oe_delay:

Whenever flagc_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

According to formula (2b) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLRD# asserted for three cycles after the partial flag (flagd) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLRD# for a count of one cycle after sampling flagd_d (flopped output of flagd) as 0.

State loop_back_read_oe_delay:

Whenever rd_oe_dealy_cnt = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_wait_flaga:

If oe_delay_cnt = 0, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

State loop_back_wait_flagb:

If flaga_d = 1, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

State loop_back_write:

If flagb_d = 1, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] =0

State loop_back_write_wr_delay:

If flagb_d = 0, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0.

State loop_back_flush_fifo:

After one clock cycle, the state machine will enter this state and flush the internal FIFO. The status of the Slave FIFO control line is:

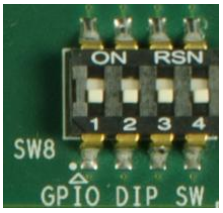
PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

Project Operation

Steps to Test Loopback Transfer

1. Connect the FX3 DVK board with the Xilinx SP601 DVK board using the FMC connector and power on both the FX3 DVK board and the Xilinx SP601 DVK board.
2. The FPGA and FX3 must be configured before the FPGA may start any transaction. The FPGA code uses GPIO inputs to determine which mode should be started.

Figure 28. SW8 on Xilinx SP601 Board – all OFF before FPGA and FX3 Configuration



3. Program the Xilinx Spartan 6 FPGA with the file *slavefifo2b_fpga_top.bit*. The FPGA can be programmed with any standard programmer such as the iMPACT application available with the Xilinx ISE Design Suite.
4. Program the FX3 device with the firmware image file *SF_loopback.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK.

Figure 29. Programming FX3 Firmware using Control Center

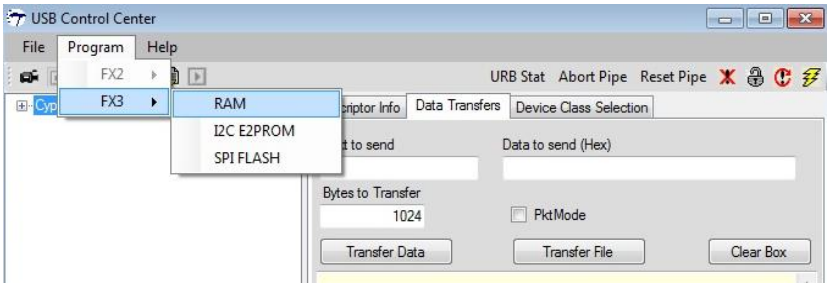
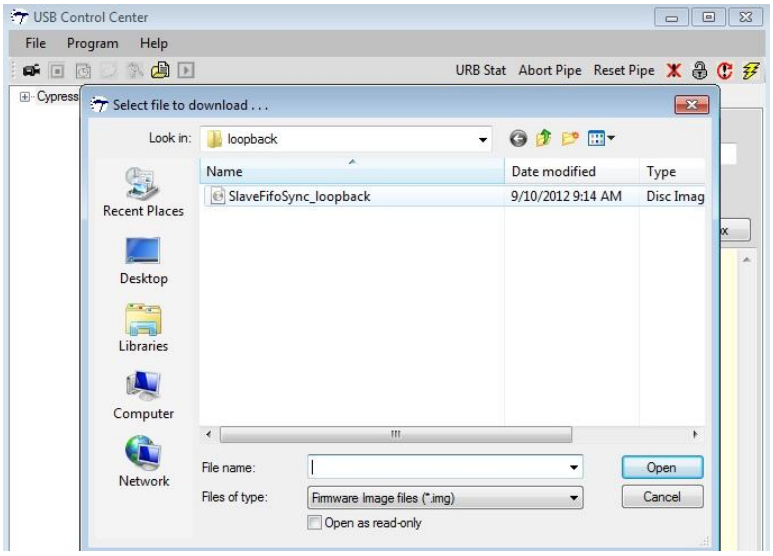


Figure 30. Programming FX3 Firmware for Loopback Testing using Control Center



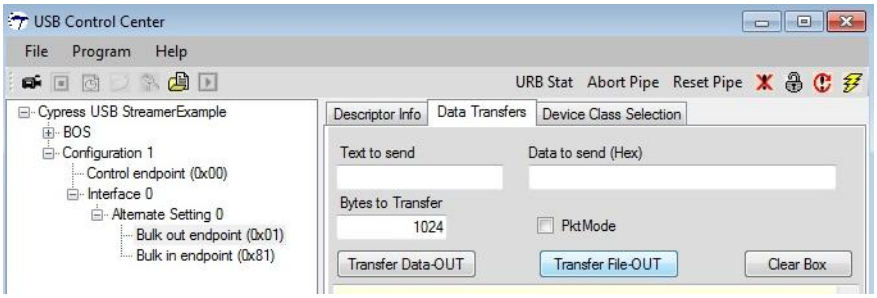
5. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port) before data transfers can be initiated. The FPGA uses GPIO inputs to determine which of these transfers to execute. Switch SW8 on the SP601 DVK board is used for this purpose. The switch setting required for the different transfers is shown in Table 7.
6. Position 1 and 3 on SW8 on the Xilinx SP601 board must be turned to the ON position to perform LoopBack transfers.

Table 7. Configuration of FPGA Transfer Modes in *slavefifo2b_fpga_top.bit*

SW8[4]	SW8[3]	SW8[2]	SW8[1]	FPGA Transfer Mode
OFF	OFF	OFF	ON	FPGA continuously writes a full packet followed by short packet
OFF	OFF	ON	OFF	FPGA continuously writes full packet followed by ZLP
OFF	OFF	ON	ON	FPGA continuously writes full packets (Stream Bulk IN packets from the host)
OFF	ON	OFF	OFF	FPGA continuously reads full packets (Stream Bulk OUT packets from the host)
OFF	ON	OFF	ON	LoopBack Transfer Mode
OFF	X	X	X	Invalid

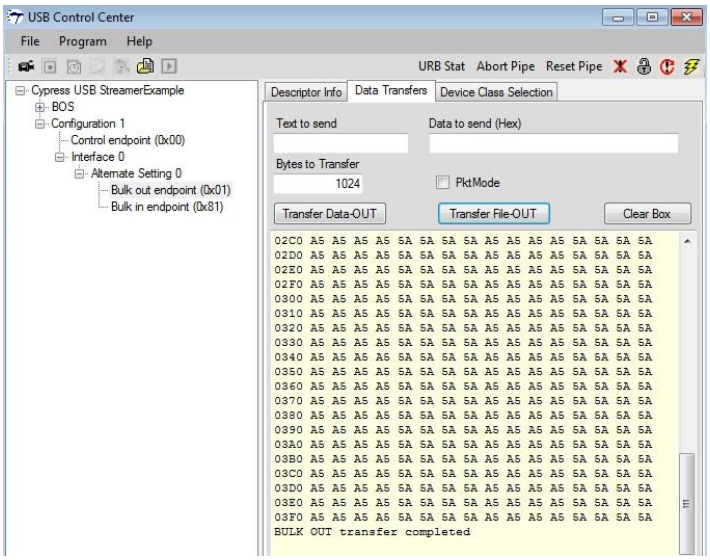
7. Now transfers can be initiated from the Control Center utility. First, initiate a Bulk OUT transfer from the USB host. Select the Bulk OUT endpoint in Control Center and click the **Transfer File-OUT** button.

Figure 31. Initiate Bulk OUT Transfer using Transfer File-OUT



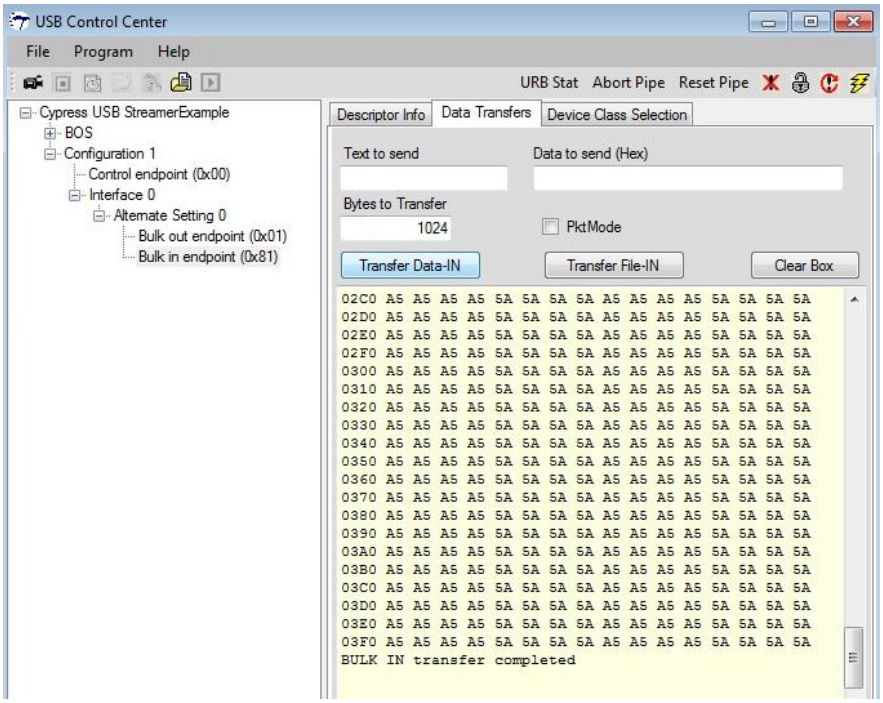
8. This allows you to browse and select a file containing the data to transfer. In the attachment to this application note, in the Loopback folder, you will find a *TEST.txt* file. This contains a data pattern, which will send out “0xA5A5A5A5 0x5A5A5A5A” in an alternating manner. Double-click to select the file and send the data.

Figure 32. Data Pattern Transferred by Selecting TEST.txt File for Transfer File-OUT



9. The FPGA is already in a state where it is waiting for FLAGA to equal 1. As soon as the data is available in the buffer of PIB_SOCKET_0, the FPGA will read it. The FPGA will then loop back the same data and write it to FX3's PIB_SOCKET_3.
10. From the USB host, you can issue a Bulk IN transfer. Select the BULK IN endpoint in Control Center and click **Transfer Data-IN**. The same data that was previously written is now read back.

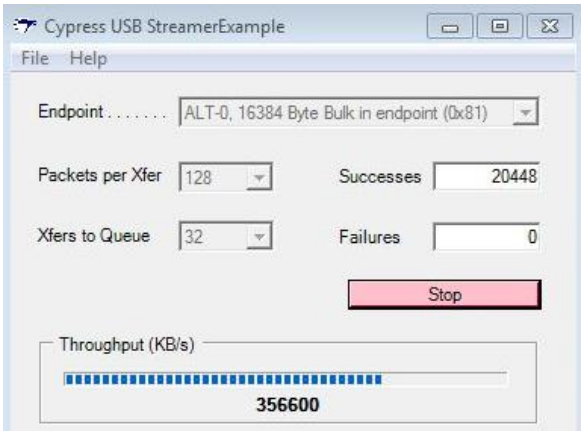
Figure 33. Complete Loopback Test by Initiating a BULK IN Transfer using Transfer Data-IN



Steps to Test Streaming Transfers

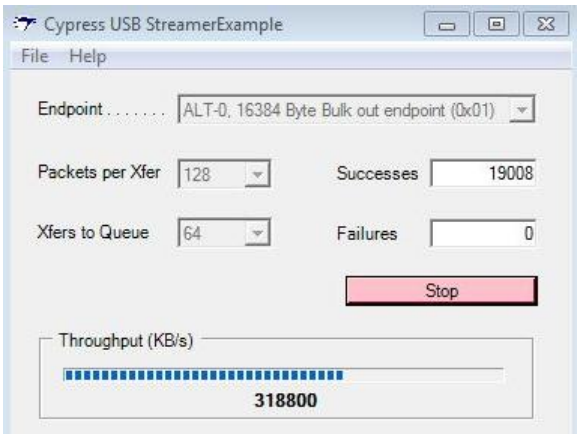
1. The FPGA can be kept programmed with the same bit file as previously mentioned for the streaming transfers. You only need to configure the switch settings, as shown in Table 7.
2. For Stream IN or OUT, program the FX3 device with the firmware image file *SF_streamIN.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK.
3. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port). Now data transfers can be initiated from the Control Center utility and the Streamer utility provided with the FX3 SDK.
4. Set the switch SW8 on the SP601 board appropriately for the required transfer, as shown in Table 7.
5. In the stream IN case, now the FPGA is already in a state where it is waiting for FLAGA to equal 1. As soon as the buffer is available, the FPGA will start writing continuously to FX3's PIB_SOCKET_0. From the USB host, you can issue continuous Bulk IN transfers. Select the BULK IN endpoint in the Cypress Streamer utility and click **Start**. The performance number is displayed. The performance shown in Figure 34 is observed on a Win7 64-bit PC with an Intel Z77 Express Chipset.

   Figure 34. Streaming IN Performance Displayed in the Cypress Streamer Utility



6. In the stream OUT case, the FPGA is already in a state where it is waiting for FLAGC to equal 1. As soon as the data is available, the FPGA will start reading continuously from FX3's PIB_SOCKET_3. From the USB host, you can issue continuous Bulk OUT transfers. Select the BULK OUT endpoint in the Cypress Streamer utility and click **Start**. The performance number is displayed. The performance shown in Figure 34 is observed on a Win7 64-bit PC with an Intel Z77 Express Chipset.

   Figure 35. Streaming OUT Performance Displayed in the Cypress Streamer Utility



The *SF_streamIN.img* can be used for both streaming IN and OUT. In the *SF_streamIN.img*, eight buffers are allocated to the P2U DMA channel and four buffers to the U2P channel. In the *SF_streamOUT.img*, eight buffers are allocated to the U2P DMA channel and four buffers to the P2U channel. Hence, higher P2U performance will be demonstrated by the *SF_streamIN.img* firmware and a higher U2P performance will be demonstrated by the *SF_streamOUT.img* firmware file.

Steps to Test Short Packet and ZLP Transfers

1. The FPGA can be kept programmed with the same bit file as previously mentioned for the streaming transfers. You only need to configure the switch settings, as shown in Table 7.
2. Program the FX3 device with the firmware image file *SF_shrt_ZLP.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK.
3. As soon as the FX3 firmware is programmed, a buffer allocated to PIB_SOCKET_0 becomes available. The FPGA is already in a state where it is waiting for this condition, by monitoring FLAGA. As soon as FLAG equals 1, the FPGA starts writing to FX3.
4. If the switch is configured for short packet transfers, the FPGA writes a full packet (1024 bytes) followed by a short packet. If the switch is configured for ZLP transfers, the FPGA writes a full packet (1024 bytes) followed by a ZLP.
5. Now the USB host can issue Bulk IN tokens. In the Control Center utility, select the Bulk IN endpoint and then click **Transfer Data-IN**. First, the full packet will be received. Click **Transfer Data-IN** again. Now, the short packet or ZLP will be received.

Figure 36. Full Packet Followed by Short Packet Received by Consecutive Transfer Data-IN Operations

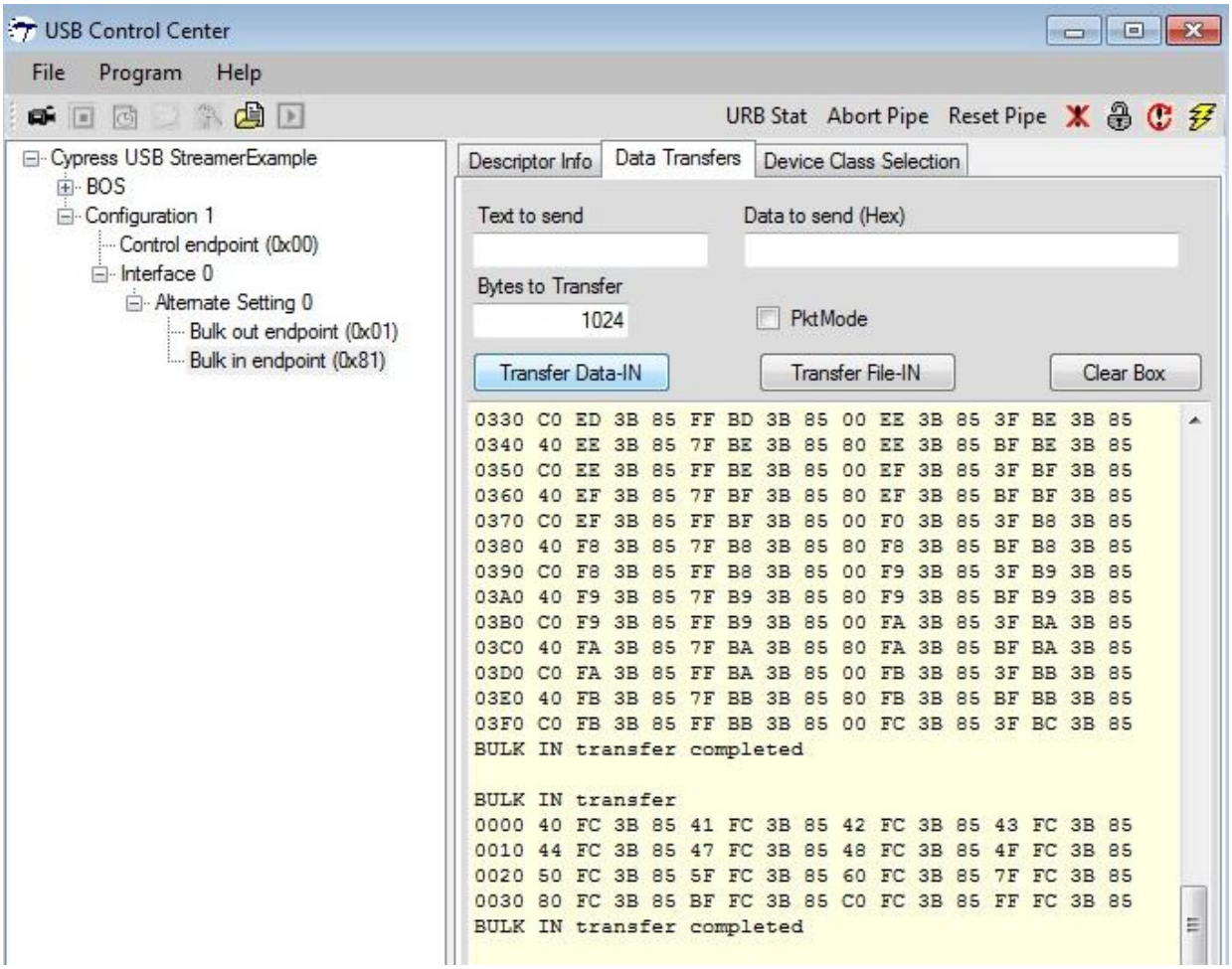
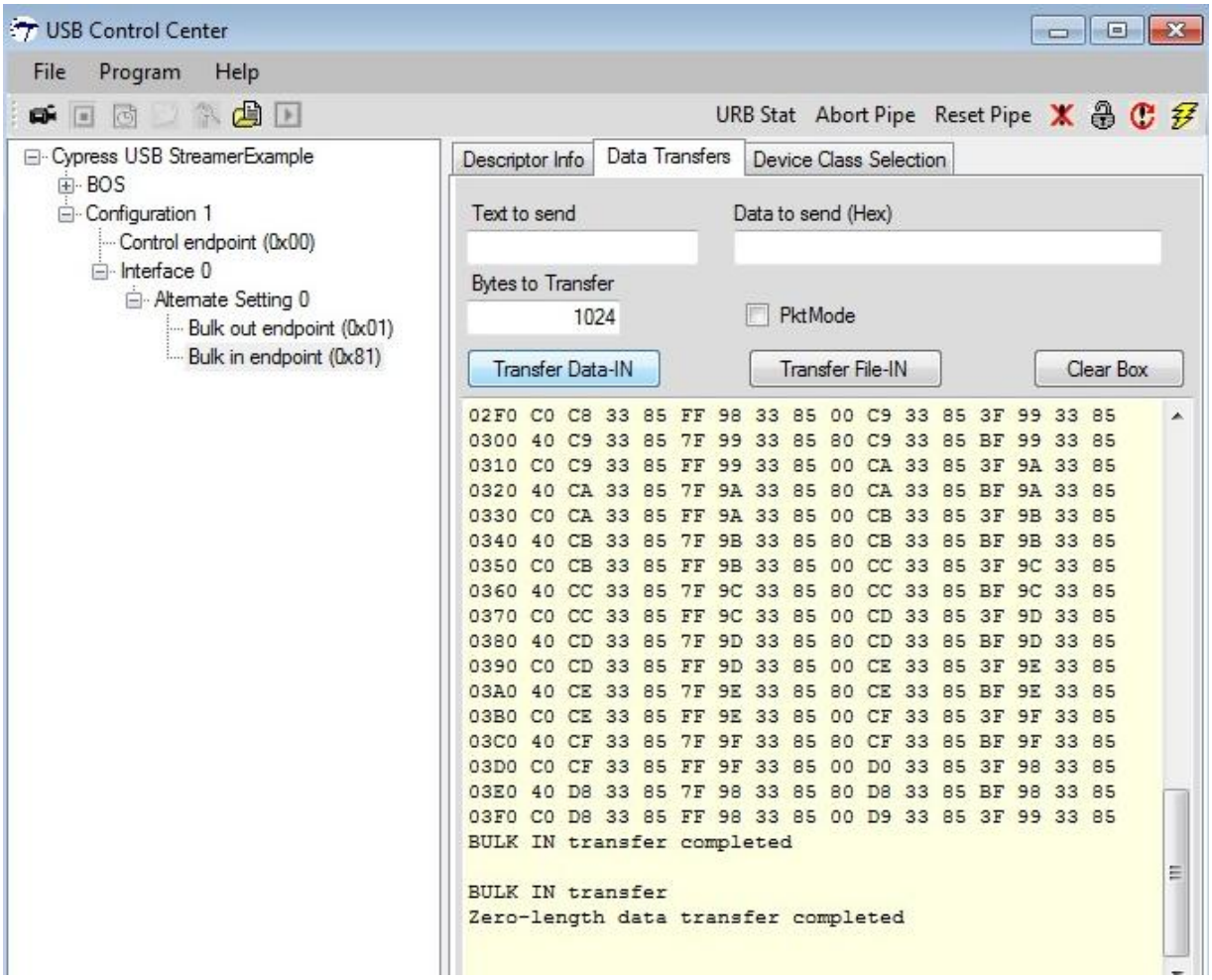


Figure 37. Full Packet Followed by Zero Length Packet Received by Consecutive Transfer Data-IN Operations



6. The FPGA is continuously writing data, so multiple BULK IN transfers can be done.

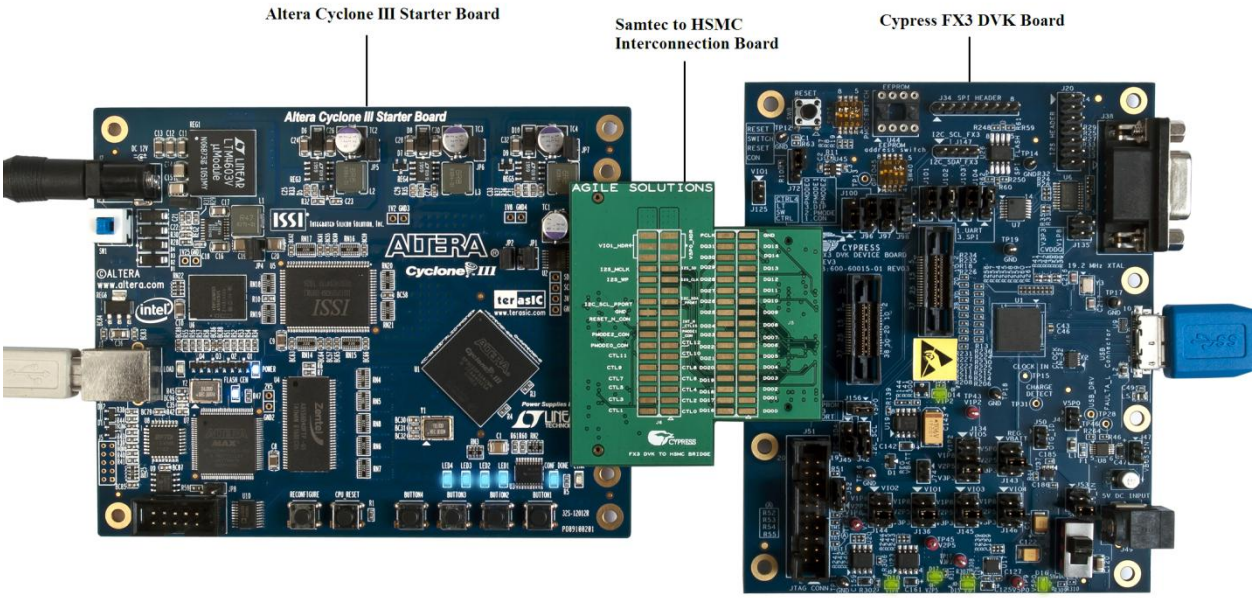
Design Example2: Interfacing an Altera FPGA to FX3's Synchronous Slave FIFO Interface

This section provides a complete design example in which an Altera Cyclone 3 FPGA is connected to FX3 over the synchronous Slave FIFO interface. The hardware, firmware, and software components used to implement this design are described here.

Hardware Setup

The project provided in this example can be executed on a hardware setup consisting of a Cypress FX3 DVK board interconnected with a Cyclone III FPGA Starter Board. The FX3 board and the Altera board are connected using a Samtec to HSMC interconnect board. The interconnect board mates with the Samtec connector on the Cypress FX3 DVK board and the HSMC connector on the Altera board. Contact Cypress(fx3@cypress.com) to get the schematics of the Samtec-to-HSMC interconnect board. Figure 38 shows a picture of this hardware setup.

Figure 38. Cypress FX3 DVK Board Connected to Altera Cyclone III Starter Board using HSMC Interconnect Board



Jumper and Switch Settings

The FX3 DVK jumper and switch settings to run this demo are listed in Table 8.

Table 8. FX3 DVK Jumper and Switch Settings

Sl. No.	Jumper/Switch	Pins to be Shorted using Jumpers	Function
1	J100	1 and 2	GPIO[21]/CTL[4] – configured as FLAGA
2	J136	2 and 5	VIO1(2.5V)
3	J144	2 and 5	VIO2(2.5V)
4	J145	2 and 5	VIO3(2.5V)
5	J146	2 and 5	VIO4(2.5V)
6	J134	3 and 6	VIO5(2.5V)
7	J135	2 and 3	CVDDQ(3.3V)
8	J143	3 and 4	VBATT(3.3V)
9	J101	1 and 2	GPIO[46] = UART_RTS
10	J102	1 and 2	GPIO[47] = UART_CTS
11	J103	1 and 2	GPIO[48] = UART_TX
12	J104	1 and 2	GPIO[49] = UART_RX
13	J96 & SW25	2 and 3	PMODE0 pin state (ON/OFF) selection using SW25. SW25.1 should be OFF
14	J97 & SW25	2 and 3	PMODE1 pin state (ON/OFF) selection using SW25. SW25.1 should be OFF
15	J98	1 and 2	PMODE2 pin floating
16	J72	1 and 2	RESET
17	J53	1 and 2	Bus powered
18	SW9	The switch should point to the direction labeled VBUS_IN	Bus powered
19	J156	Place a jumper to short	Powers Samtec connector

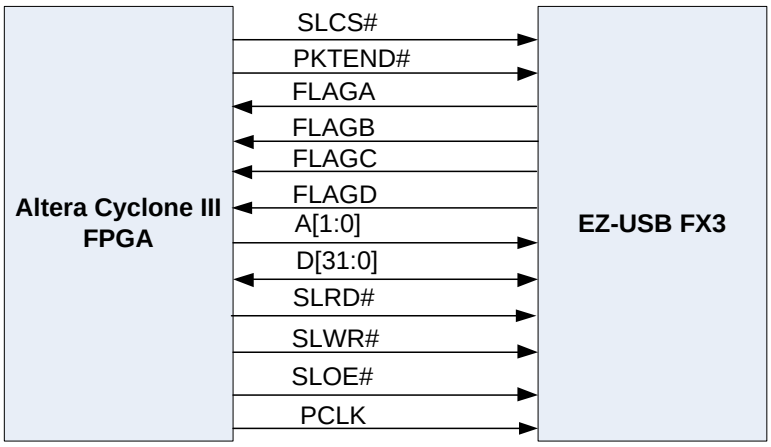
Note PMODE pins are set for USB boot. The jumpers that are not listed in the table can be left opened.

Firmware and Software Components

- FX3 synchronous Slave FIFO firmware project available with the FX3 SDK.
- Control Center and Streamer software utilities available with the FX3 SDK.

The following figure shows the interconnect diagram between the FPGA and FX3.

Figure 39. Interconnect Diagram between FPGA and FX3



This example has the following components:

- Loopback transfer: In this component, the FPGA first reads a complete buffer from FX3 and then writes it back to FX3. The USB host should issue OUT/IN tokens to transmit and then receive this data. The Control Center utility provided with the EZ-USB FX3 SDK can be used for this purpose.
- Short packet: In this component, the FPGA transfers a full packet followed by a short packet to FX3. The USB host should issue IN tokens to receive this data.
- Zero length packet (ZLP) transfer: In this component, the FPGA transfers a full packet followed by a zero length packet to FX3. The USB host should issue IN tokens to receive this data.
- Streaming (IN) data transfer: In this component, the FPGA does one directional transfers, that is, continuously writes data to FX3 over synchronous Slave FIFO. The USB host should issue IN tokens to receive this data. The Control Center or Streamer utility provided with the EZ-USB FX3 SDK can be used for this purpose.
- Streaming (OUT) data transfer: In this component, the FPGA does one directional transfers, that is, continuously reads data from FX3 over synchronous Slave FIFO. The USB host should issue OUT tokens to provide this data. The Control Center or Streamer utility provided with the EZ-USB FX3 SDK can be used for this purpose.

FX3 Firmware Details

The FX3 firmware is based on the example project available with the FX3 SDK.

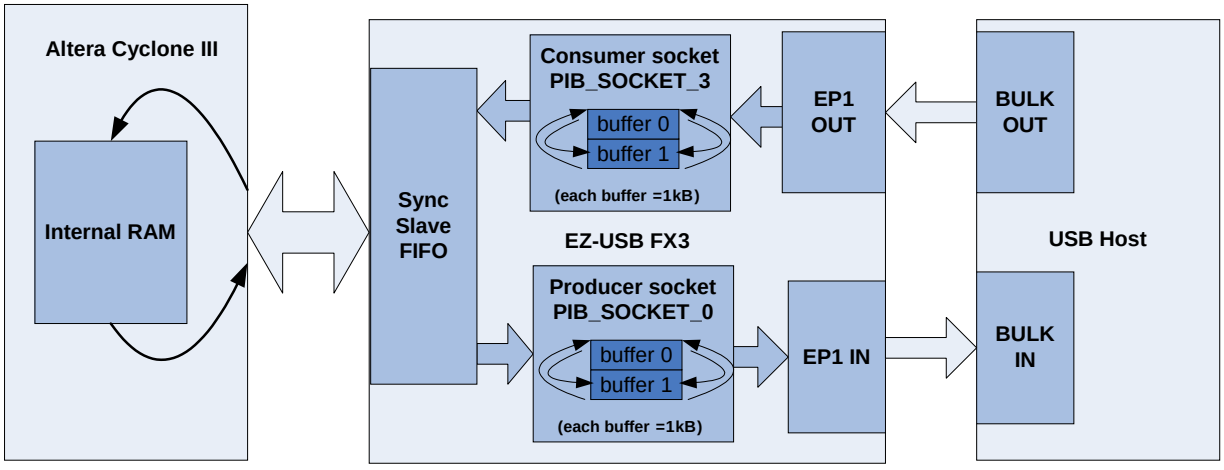
The main features of this firmware are:

- Enables both USB 3.0 and USB 2.0.
- Enumerates with the Cypress VID/PID, 0x04B4/0x00F1. This enables the use of the Cypress Control Center and Streamer utilities for initiating USB transfers.
- Integrates the synchronous Slave FIFO descriptor, which:
  - Supports access to up to four sockets
  - Configures data bus width to 32-bit
  - Works on the 100-MHz PCLK input clock
  - Configures four flags:
    i. FLAGA: Full flag dedicated to thread0
    ii. FLAGB: Partial flag with watermark value 6, dedicated to thread0
    iii. FLAGC: Empty flag dedicated to thread3
    iv. FLAGD: Partial flag with watermark value 6, dedicated to thread3

  Note The GPIFII Designer project provided with this application note demonstrates these settings. In addition, the firmware project shows the use of the CyU3PgpifSocketConfigure() API to configure the watermark value.
- Configures the PLL frequency to 400 MHz. This is done by setting the setSysClk400 parameter as an input to the CyU3PdeviceInit() function.

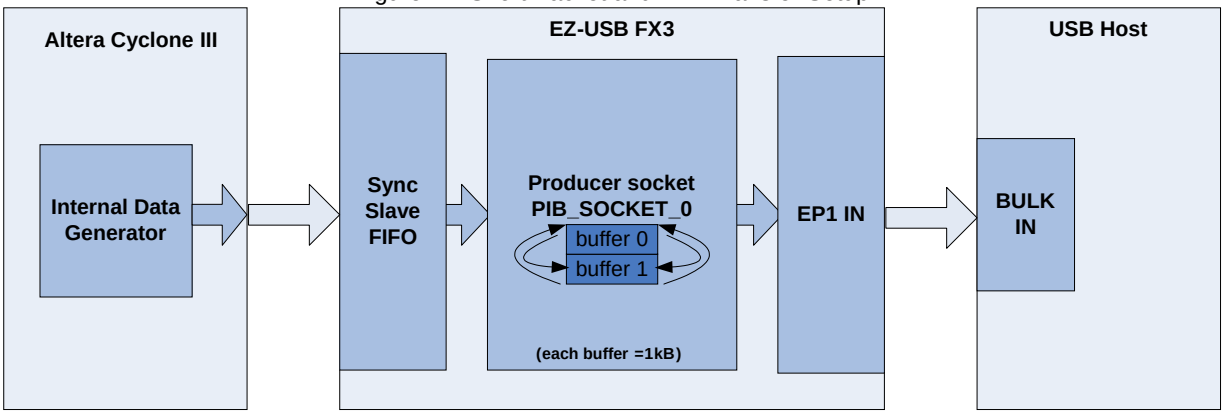
  Note This setting is essential for the functioning of Slave FIFO at 100 MHz with 32-bit data.
- Sets up the DMA channels:
  - For loopback transfers, short packet, and ZLP transfer, two DMA channels are created:
    i. A P2U channel with PIB_SOCKET_0 as the producer and UIB_SOCKET_1 as the consumer. The DMA buffer size is 512 or 1024 depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 2.
    ii. A U2P channel with PIB_SOCKET_3 as the consumer and UIB_SOCKET_1 as the producer. The DMA buffer size is 512 or 1024 depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 2.

Figure 40. Loopback Transfer Setup



Note Only the P2U channel is used for short packets and ZLPs.

Figure 41. Short Packet and ZLP Transfer Setup

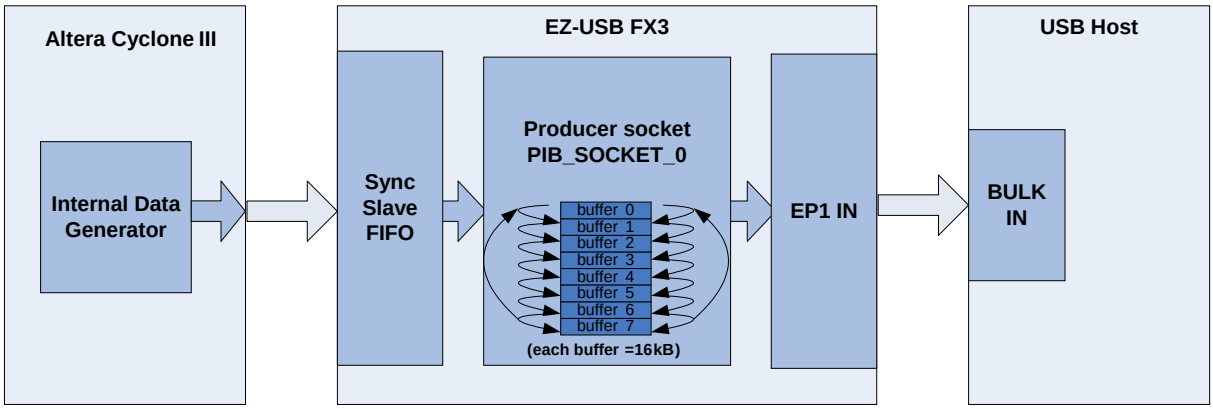


The DMA channels described in this section are set up if the following define is enabled in the *cyfxslfifosync.h* file in the FX3 firmware project provided with this application note.

```
/* set up DMA channel for loopback/short packet/ZLP transfers */
#define LOOPBACK_SHRT_ZLP
```

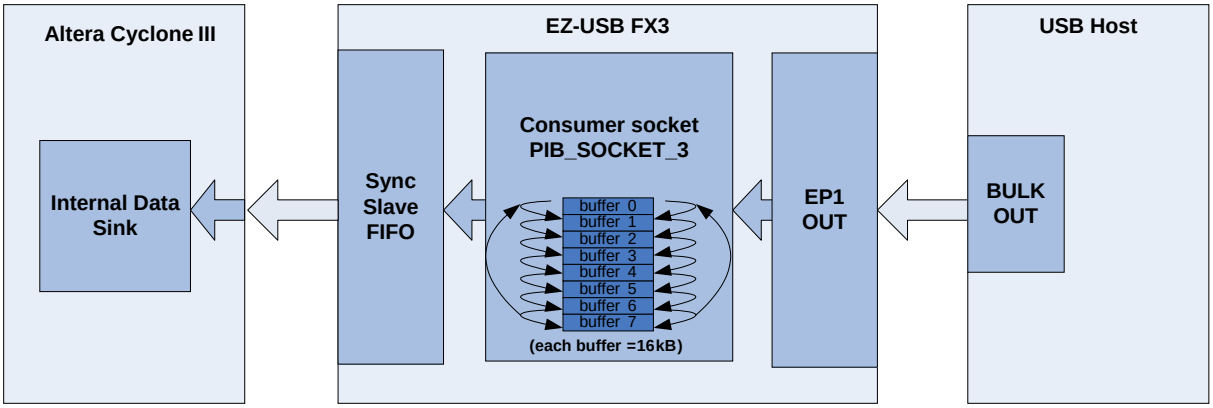
- For streaming, two DMA channels are created:
  i. A P2U channel with PIB_SOCKET_0 as the producer and UIB_SOCKET_1 as the consumer. The DMA buffer size is 16×1024 (for a USB 3.0 connection) or 16×512 (for a USB 2.0 connection) depending on whether the USB connection is USB 2.0 or USB 3.0. The DMA buffer count is 8. This buffer size and count is chosen to provide high throughput performance.

Figure 42. Stream IN Transfer Setup – Buffer Count and Size Optimized for Performance



  ii. A U2P channel with PIB_SOCKET_3 as the consumer and UIB_SOCKET_1 as the producer. The DMA buffer size is 16×1024 (for a USB 3.0 connection) or 16×512 (for a USB 2.0 connection). The DMA buffer count is 4. Note that the buffer count can be increased to enhance performance, but then the buffer count of the P2U channel should be reduced. This is because the FX3 SDK does not provide enough buffer memory such that both channels can have a buffer size of 16×1024 and buffer count of 8.

Figure 43. Stream OUT Transfer Setup – Buffer Count and Size Optimized for Performance



The DMA channels previously mentioned are set up if the following define is enabled in the *cyfxslfifosync.h* file in the FX3 firmware project provided with this application note.

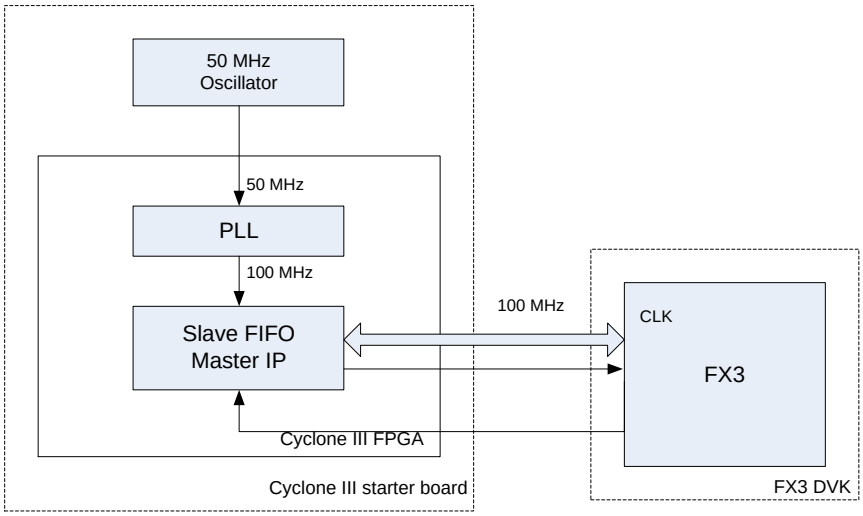
```
/* set up DMA channel for stream IN/OUT transfers */
#define STREAM_IN_OUT
```

The buffer count allocated to the P2U and U2P DMA channels can be controlled by using the following defines, also in the *cyfxslfifosync.h* file :

```
/* Slave FIFO P_2_U channel buffer count */
#define CY_FX_SLFIFO_DMA_BUF_COUNT_P_2_U      (4)
/* Slave FIFO U_2_P channel buffer count */
#define CY_FX_SLFIFO_DMA_BUF_COUNT_U_2_P      (8)
```

FPGA Implementation Details

Figure 44. Altera Cyclone III (EP3C25F324C6) FPGA Implementation Using SP601 Evaluation Kit



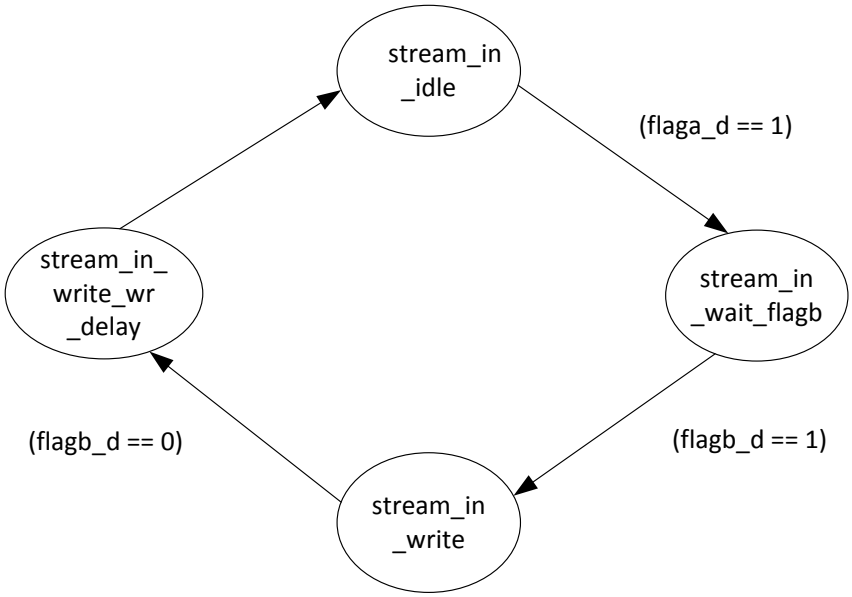
To demonstrate the maximum performance of FX3, the GPIF interface runs at 100 MHz. The Cyclone III starter board has an onboard 50 MHz single-ended oscillator. The FPGA uses a PLL to generate a 100-MHz clock from the 50-MHz clock.

Following are the state implementations of the different types of transfers.

Stream IN Example [FPGA writing to Slave FIFO]

The state machine implemented in Verilog RTL for the stream IN transfers is shown in the following figure.

Figure 45. FPGA State Machine for Stream IN



State stream_in_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State stream_in_wait_flagb:

Whenever flaga_d = 1, the state machine will enter this state and wait for flagb_d.

State stream_in_write:

Whenever flagb_d = 1, the state machine will enter this state and start writing to the Slave FIFO interface. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State stream_in_write_wr_delay:

Whenever flagb_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is as follows:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

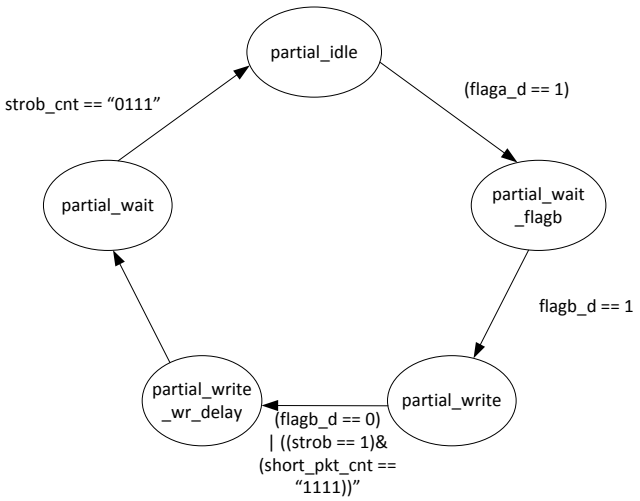
After one clock cycle, the state machine will enter the stream_in_idle state

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial FLAG(flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the flagb_d (flopped output of flagb) as 0.

Short Packet Example [FPGA writing full packet followed by short packet to Slave FIFO]

This example demonstrates the short packet commit procedure using PKTEND#. The state machine implemented in Verilog RTL for the short packet example is shown in the following figure.

Figure 46. State Machine for Short Packet Transfer



State partial_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State partial_wait_flagb:

Whenever flaga_d = 1, the state machine will enter this state.

State partial_write:

Whenever flagb_d = 1, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State partial_write_wr_delay:

Whenever flagb_d = 0 or (strob = 1 and short_pkt_cnt = "1111"), the state machine will enter this state. If strob = 1, the FPGA master will commit a short packet. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

After one clock cycle, the state machine will enter the partial_wait state.

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial FLAG(flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0.

State partial_wait:

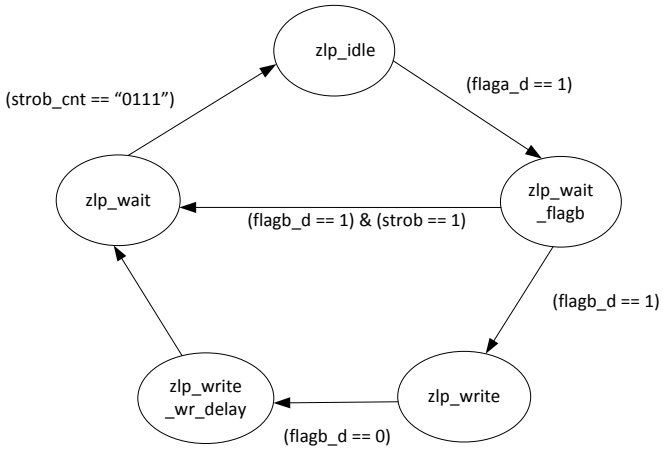
Whenever strob_cnt = 0111, the state machine will enter partial_idle state.

As the watermark value is 6, flaga is expected to go to 0, only six clock cycles after the partial flag (flagb). This state holds the execution for more than four clock cycles to ensure the availability of a valid status on flaga.

Zero Length Packet Example [FPGA writing full packet followed by ZLP to Slave FIFO]

This example demonstrates the ZLP commit procedure using PKTEND#. The state machine implemented in Verilog RTL for the ZLP example is shown in the following figure.

Figure 47. State Machine for ZLP Transfer



State zlp_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

State zlp_wait_flagb:

Whenever flaga_d = 1, the state machine will enter this state.

State zlp_write:

Whenever flagb_d = 1, the state machine will enter this state. The status of the Slave FIFO control line is

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] = 0

State zlp_write_wr_delay:

Whenever flagb_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:
PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] = 0

After one clock cycle, the state machine will enter the zlp_wait state.

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0.

State zlp_wait:

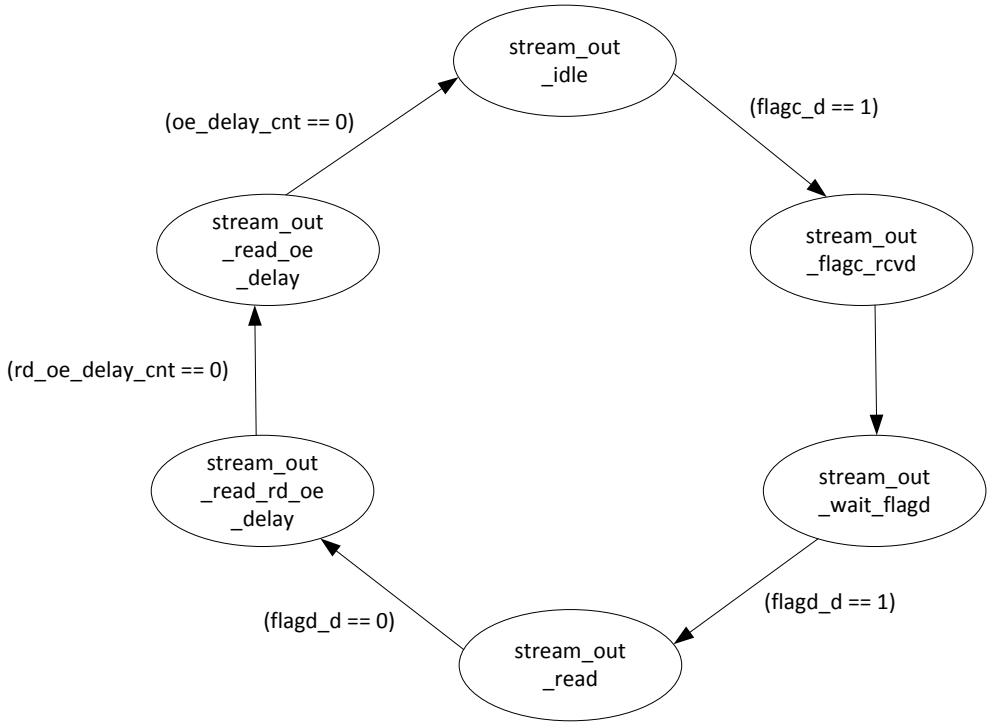
Whenever flagb_d = 1 and strob = 1, the state machine will enter this state from the zlp_wait_flagb state and commit a zlp packet into slavefifo.

As the watermark value is 6, flaga is expected to go to 0, only six clock cycles after the partial flag (flagb). This state holds the execution for more than four clock cycles to ensure the availability of a valid status on flaga.

Whenever strob_cnt = 0111, the state machine will enter zlp_idle state.

State Machine Implemented in Verilog RTL for Stream OUT Example

Figure 48. State Machine for Stream OUT Transfer



State stream_out_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State stream_out_flagc_rcvd:

Whenever flagc_d = 1, the state machine will enter this state.

State stream_out_wait_flagd:

After one clock cycle, the state machine will enter this state.

State stream_out_read:

Whenever flagc_d = 1, the state machine will enter this state. Here the state machine will assert read control signals as:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

State stream_out_read_rd_oe_delay:

Whenever flagc_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

According to formula (2b) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLRD# asserted for three cycles after the partial flag (flagd) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLRD# for a count of one cycle after sampling flagd_d (flopped output of flagd) as 0.

State stream_out_read_oe_delay:

Whenever rd_oe_dealy_cnt = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

If oe_delay_cnt = 0, the state machine will enter the stream_out_idle state from this state.

Loopback Example [FPGA reading from Slave FIFO and writing the same data back to Slave FIFO]:

The state machine moves through six states before completing one loopback cycle. The state machine along with the corresponding actions is shown in the following figure.

Figure 49. State Machine for Loopback Transfer

State loop_back_idle:

This state initializes all the registers and signals used in the state machine. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_flagc_rcvd:

Whenever the flagc_d = 1, the state machine will enter this state.

State loop_back_wait_flagd:

After one clock cycle, the state machine will enter this state and wait for flagd.

State loop_back_read:

If flagd_d = 1, the state machine will enter this state. Here the state machine will assert read control signals as follows:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_read_rd_oe_delay:

Whenever flagc_d = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 0; SLCS# = 0; SLWR# = 1; A[1:0] =3

According to formula (2b) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLRD# asserted for three cycles after the partial flag (flagd) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLRD# for a count of one cycle after sampling flagd_d (flopped output of flagd) as 0.

State loop_back_read_oe_delay:

Whenever rd_oe_dealy_cnt = 0, the state machine will enter this state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 0; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =3

State loop_back_wait_flaga:

If oe_delay_cnt = 0, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

State loop_back_wait_flagb:

If flaga_d = 1, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

State loop_back_write:

If flagb_d = 1, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 0; A[1:0] =0

State loop_back_write_wr_delay:

If flagb_d = 0, the state machine will enter the loop_back_wait_flaga state. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

According to formula (1) in the section General Formulas for Using Partial FLAGs, FX3 should sample SLWR# asserted for two cycles after the partial flag (flagb) goes to 0. Considering a one-cycle propagation delay through the FPGA and to the interface, the FPGA asserts SLWR# for a count of one cycle after sampling the partial flagb_d (flopped output of flagb) as 0.

State loop_back_flush_fifo:

After one clock cycle, the state machine will enter this state and flush the internal FIFO. The status of the Slave FIFO control line is:

PKTEND# = 1; SLOE# = 1; SLRD# = 1; SLCS# = 0; SLWR# = 1; A[1:0] =0

Project Operation

Steps to Test Loopback Transfer

1. Connect the FX3 DVK board with the Altera Cyclone III FPGA starter board using the HSMC connector and power on both the FX3 DVK board and the Altera Cyclone III FPGA starter board.
2. Program the FX3 device with the firmware image file *SF_loopback.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK. FX3 should be programmed before programming the Altera Cyclone III FPGA. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).

Figure 50. Programming FX3 Firmware using Control Center

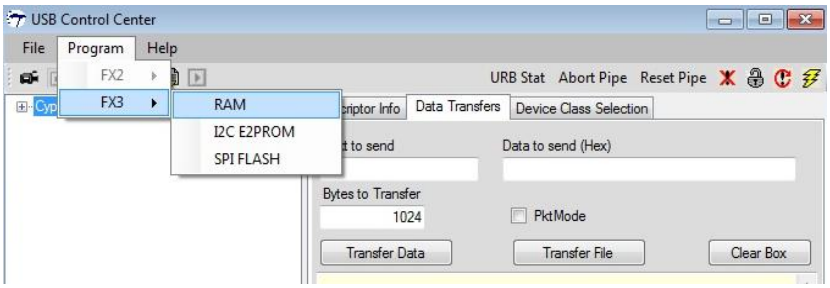
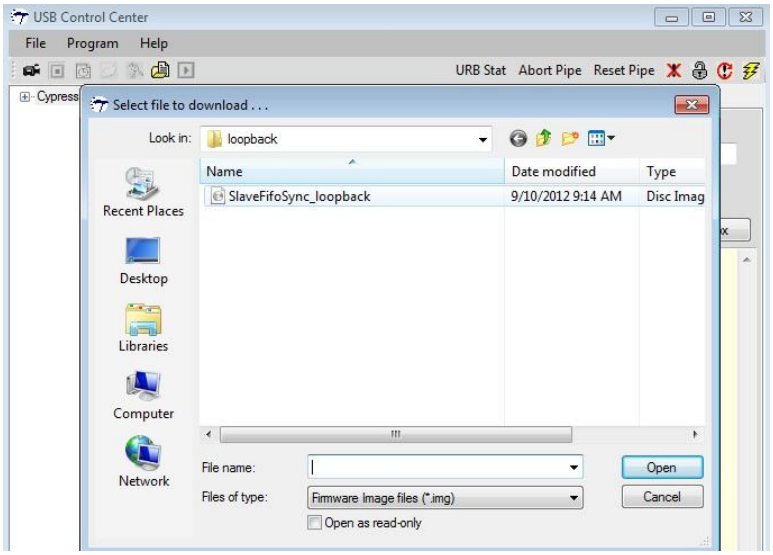
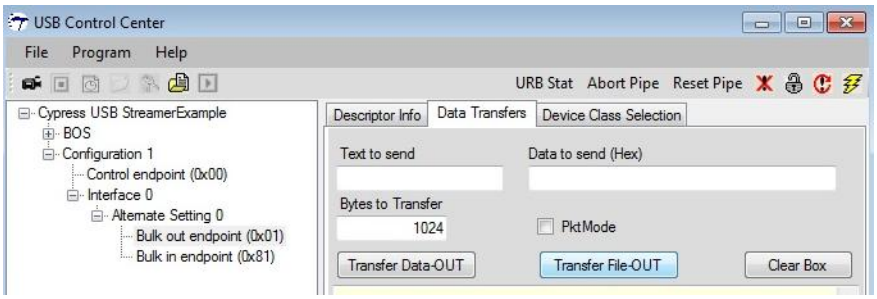


Figure 51. Programming FX3 Firmware for Loopback Testing Using Control Center



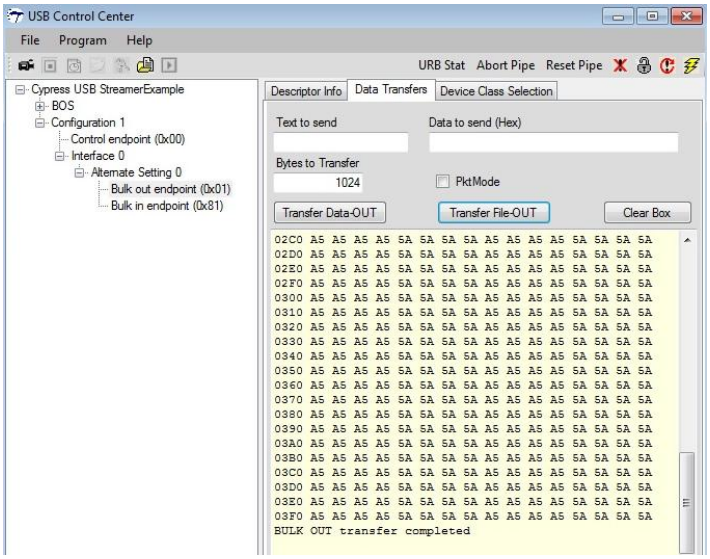
3. Program the Altera Cyclone III FPGA with the file *slaveFIFO2b_loopback.sof*. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with the Quartus II software.
4. Now transfers can be initiated from the Control Center utility. First, initiate a Bulk OUT transfer from the USB host. Select the Bulk OUT endpoint in Control Center and click the **Transfer File-OUT** button.

Figure 52. Initiate Bulk OUT Transfer using Transfer File-OUT



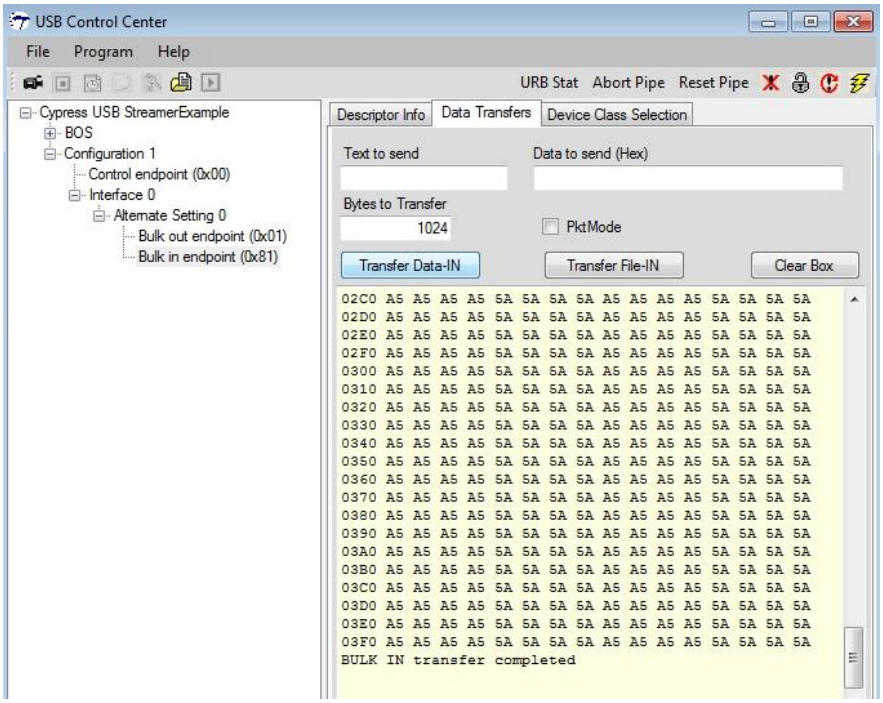
5. This will allow you to browse and select a file containing the data to transfer. In the attachment to this application note, in the Loopback folder, you will find a *TEST.txt file*. This contains a data pattern, which will send out "0xA5A5A5A5 0x5A5A5A5A" in an alternating manner. Double-click to select the file and send the data.

Figure 53. Data Pattern Transferred by Selecting TEST.txt File for Transfer File-OUT



6. The FPGA is already in a state where it is waiting for FLAGA to equal 1. As soon as the data is available in the buffer of PIB_SOCKET_0, the FPGA will read it. The FPGA will then loop back the same data and write it to FX3's PIB_SOCKET_3.
7. You can issue a Bulk IN transfer from the USB host. Select the BULK IN endpoint in Control Center and click **Transfer Data-IN**. The same data that was previously written is now read back.

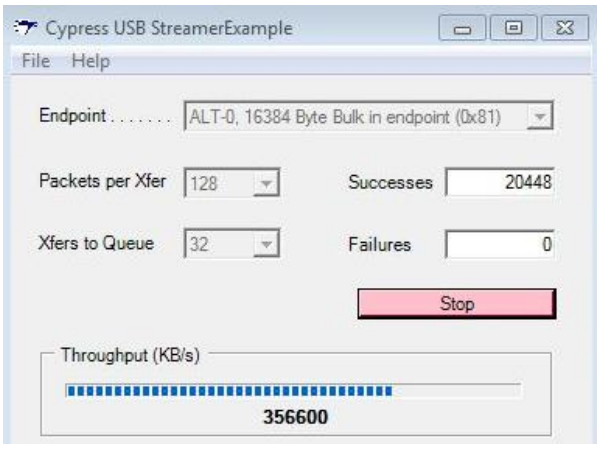
Figure 54. Complete Loopback Test by Initiating a BULK IN Transfer using Transfer Data-IN



Steps to Test Streaming Transfers

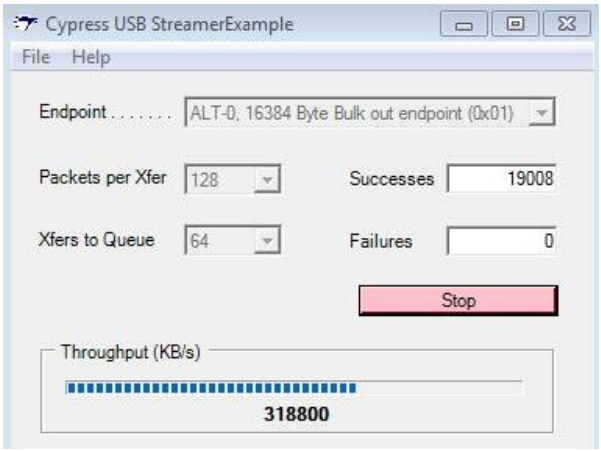
1. Connect the FX3 DVK board with the Altera Cyclone III FPGA starter board using the HSMC connector and power on both the FX3 DVK board and the Altera Cyclone III FPGA starter board.
2. For Stream IN or OUT, program the FX3 device with the firmware image file *SF_streamIN.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK. FX3 should be programmed before programming the Altera Cyclone III FPGA. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).
3. Program the Altera Cyclone III FPGA with the file *slaveFIFO2b_streamIN.sof* for stream IN transfers or with the file *slaveFIFO2b_streamOUT.sof* for stream OUT transfers. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with Quartus II software.
4. In the stream IN case, now the FPGA is already in a state where it is waiting for FLAGA to equal 1. As soon as the buffer is available, the FPGA will start writing continuously to FX3's PIB_SOCKET_0. From the USB host, you can issue continuous Bulk IN transfers. Select the BULK IN endpoint in the Cypress Streamer utility and click **Start**. The performance number is displayed. The performance shown in Figure 55 is observed on a Win7 64-bit PC with an Intel Z77 Express Chipset.

Figure 55. Streaming IN Performance in Cypress Streamer Utility



5. In the stream OUT case, the FPGA is already in a state where it is waiting for FLAGC to equal 1. As soon as the data is available, the FPGA will start reading continuously from FX3's PIB_SOCKET_3. From the USB host, you can issue continuous Bulk OUT transfers. Select the BULK OUT endpoint in the Cypress Streamer utility and click **Start**. The performance number is displayed. The performance shown in Figure 56 is observed on a Win7 64-bit PC with an Intel Z77 Express Chipset.

Figure 56. Streaming OUT Performance in Cypress Streamer Utility

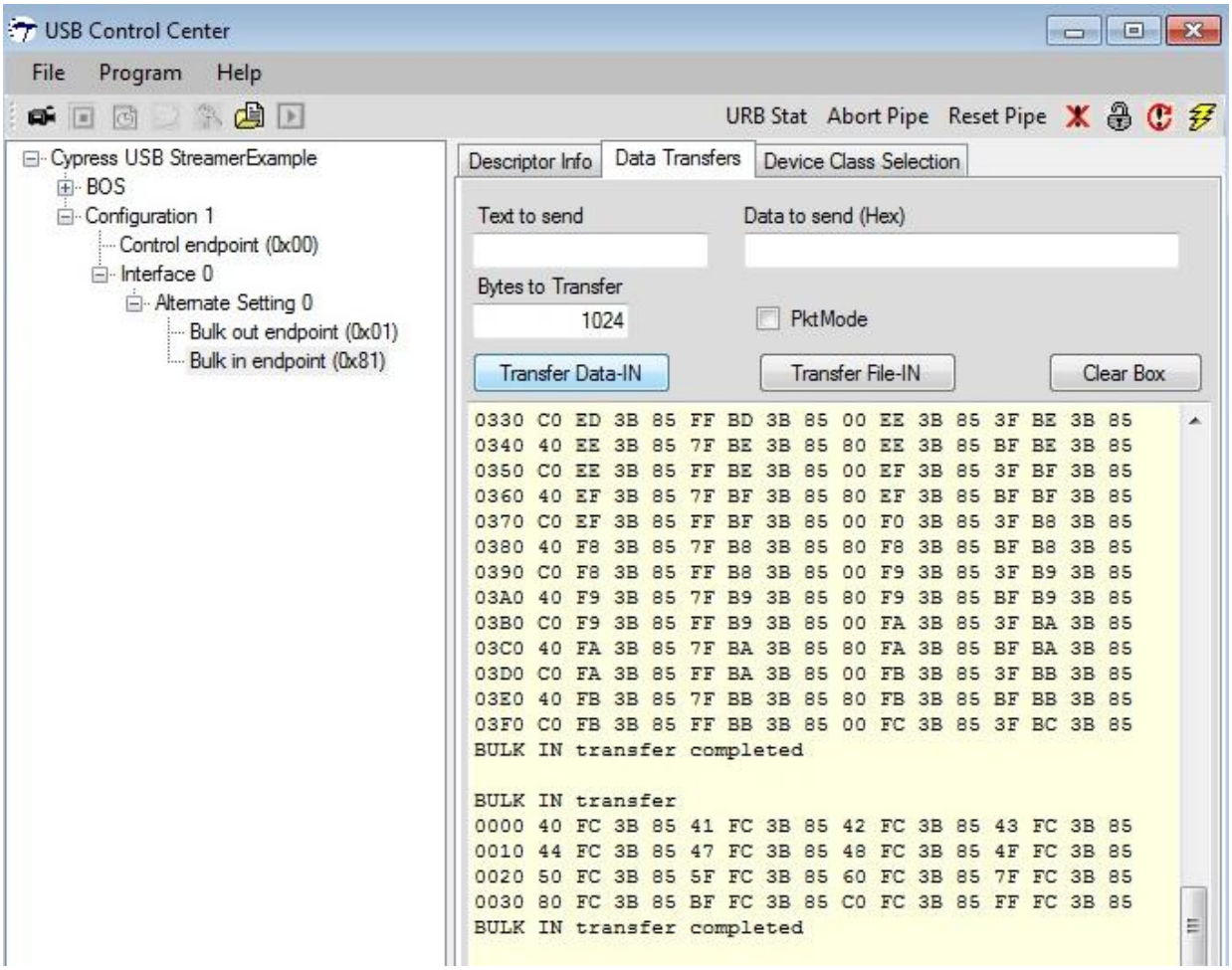


The *SF_streamIN.img* can be used for both streaming IN and OUT. In *SF_streamIN.img*, eight buffers are allocated to the P2U DMA channel and four buffers to the U2P channel. In the *SF_streamOUT.img*, eight buffers are allocated to the U2P DMA channel and four buffers to the P2U channel. Hence, higher P2U performance is demonstrated by the *SF_streamIN.img* firmware and a higher U2P performance is demonstrated by the *SF_streamOUT.img* firmware file.

Steps to Test Short Packet Transfers

1. Connect the FX3 DVK board with the Altera Cyclone III FPGA starter board using the HSMC connector and power on both the FX3 DVK board and the Altera Cyclone III FPGA starter board.
2. For Stream IN or OUT, program the FX3 device with the firmware image file *SF_streamIN.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK. FX3 should be programmed before programming the Altera Cyclone III FPGA. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).
3. Program the Altera Cyclone III FPGA with the file *slaveFIFO2b_partial.sof*. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with Quartus II software.
4. As soon as the FX3 firmware is programmed, a buffer allocated to PIB_SOCKET_0 becomes available. The FPGA is already in a state where is it waiting for this condition, by monitoring FLAGA. As soon as FLAG equals 1, the FPGA starts writing to FX3.
5. The FPGA writes a full packet (1024 bytes) followed by a short packet.
6. Now the USB host can issue Bulk IN tokens. In the Control Center utility, select the Bulk IN endpoint and then click **Transfer Data-IN**. First, the full packet will be received. Click **Transfer Data-IN** again. Now, the short packet will be received.

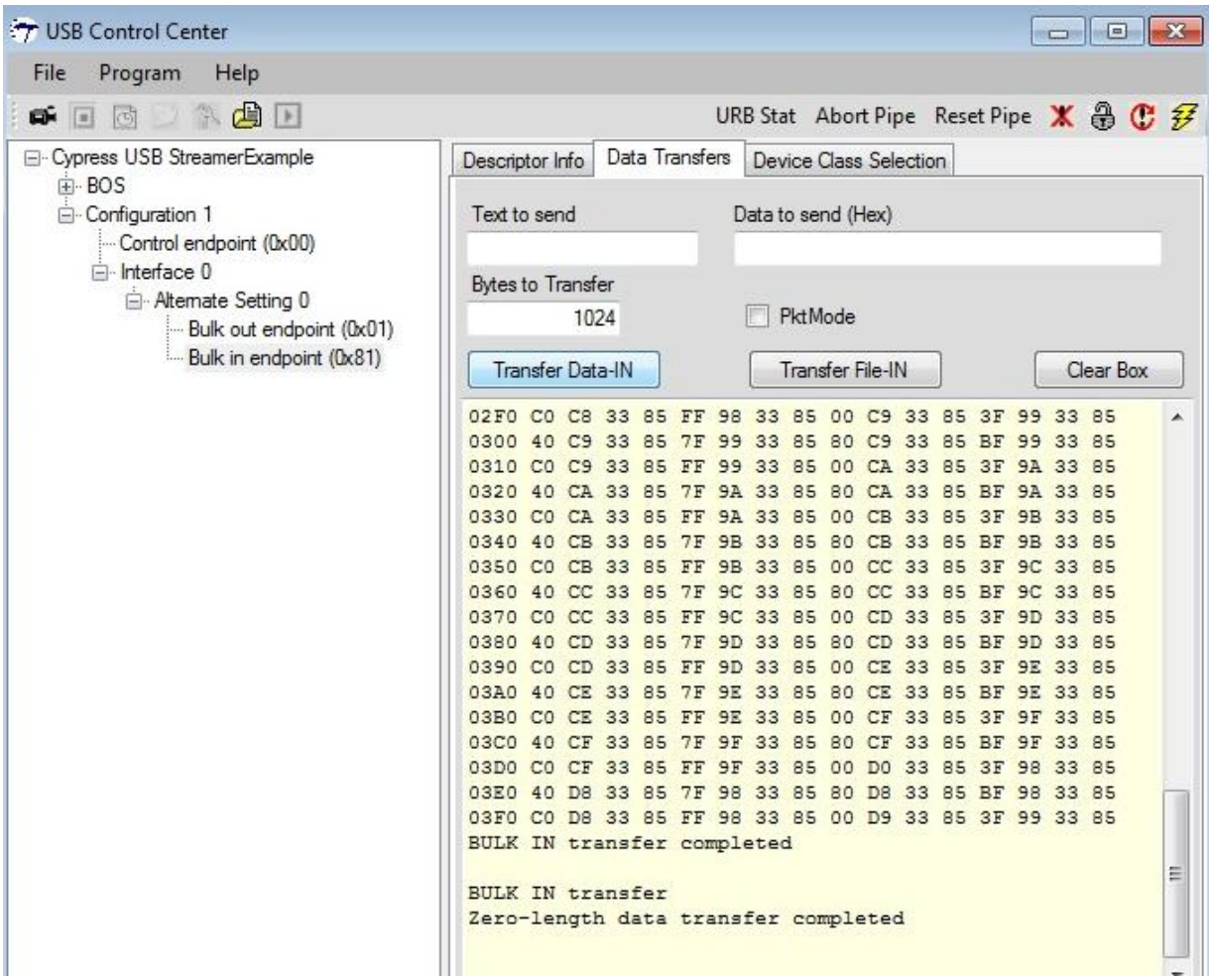
Figure 57. Full Packet Followed by Short Packet Received by Consecutive Transfer Data-IN Operations



Steps to Test ZLP Transfers

1. Connect the FX3 DVK board with the Altera Cyclone III FPGA starter board using the HSMC connector and power on both the FX3 DVK board and the Altera Cyclone III FPGA starter board.
2. For Stream IN or OUT, program the FX3 device with the firmware image file *SF_streamIN.img*. FX3 can be programmed from the USB host using the Control Center utility available with the FX3 SDK. FX3 should be programmed before programming the Altera Cyclone III FPGA. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).
3. Program the Altera Cyclone III FPGA with the file *slaveFIFO2b_ZLP.sof*. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with Quartus II software.
4. As soon as the FX3 firmware is programmed, a buffer allocated to PIB_SOCKET_0 becomes available. The FPGA is already in a state where is it waiting for this condition, by monitoring FLAGA. As soon as FLAG equals 1, the FPGA starts writing to FX3.
5. The FPGA writes a full packet (1024 bytes) followed by a ZLP.
6. Now the USB host can issue Bulk IN tokens. In the Control Center utility, select the Bulk IN endpoint and then click **Transfer Data-IN**. First, the full packet will be received. Click **Transfer Data-IN** again. Now, the ZLP will be received.

Figure 58. Full Packet Followed by Zero Length Packet Received by Consecutive Transfer Data-IN Operations



7. Note that the FPGA is continuously writing data, so multiple BULK IN transfers can be done.

Associated Project Files

File/Folder name		Description
FX3 Firmware		Source for FX3 firmware
FPGA Source files	fx3_slaveFIFO2b_xilinx	This folder contains the following subfolders: *fpga_slavefifo2b_verilog* Xilinx FPGA source code in Verilog that supports all transfer types (stream IN, stream OUT, short packets, ZLP, and loopback). *fpga_slavefifo2b_vhdl* Xilinx FPGA source code in VHDL that supports all transfer types (stream IN, stream OUT, short packets, ZLP, and loopback).
	fx3_slaveFIFO2b_altera	Altera FPGA source code in both Verilog and VHDL. This folder contains the subfolder for each transfer type and each of the following folders contains both Verilog and VHDL projects: *fpga_loopback* for loopback data transfers, *fpga_partial* for short packet data transfers, *fpga_streamIN* for stream IN data transfers, *fpga_streamOUT* for stream OUT data transfers, *fpga_zlp* for ZLP transfers.
SF_loopback.img		The FX3 firmware image that contains the Slave FIFO implementation and sets up the DMA channels required for loopback transfer. **Note** The only differences between the different FX3 firmware images provided is in the way the DMA channels are set up, for ease of demonstration of different types of transfers. However, any one firmware image can be used to demonstrate any type of transfer.
SF_streamIN.img		The FX3 firmware image that contains the Slave FIFO implementation and sets up the DMA channels required for optimized performance when performing stream IN transfers.
SF_streamOUT.img		The FX3 firmware image that contains the Slave FIFO implementation and sets up the DMA channels required for optimized performance when performing stream OUT transfers.
TEST.txt		The file that contains the data pattern sent using the **Transfer File-OUT** button in the Control Center utility.

Summary

The Slave FIFO interface is suitable for applications in which an external FPGA, processor, or device needs to perform data read/write accesses to the EZ-USB FX3's internal FIFO buffers.

This application note describes the synchronous Slave FIFO interface in detail. The different FLAG configurations available are also described along with an explanation of how the GPIFII Designer tool may be used to configure FLAGs. Two complete design examples are also presented.

About the Author

Name: Rama Sai Krishna V

Title: Applications Engineer Staff

Contact: rskv@cypress.com

Troubleshooting

Symptom 1
BULK IN and BULK OUT data transfers are failing.

```
BULK IN transfer
BULK IN transfer failed with Error Code:997
```

```
BULK OUT transfer
BULK OUT transfer failed with Error Code:997
```

Reason and Solution
FPGA connected to the Slave FIFO interface does not send continuous data or the interface clock (PCLK) is running at a slower frequency. In this scenario, there are chances that the USB link may be stuck in the low-power state.

Use the **CyU3PUsbLPMDisable** function to stop entering the low-power state, as shown here.

```
CyFxSlFifoApplnUSBEventCB (
    CyU3PUsbEventType_t evtype,
    uint16_t            evdata
    )
{
    switch (evtype)
    {
        case CY_U3P_USB_EVENT_SETCONF:
            /* Stop the application before re-starting. */
            if (glIsApplnActive)
            {
                CyFxSlFifoApplnStop ();
            }
            CyU3PUsbLPMDisable();
            /* Start the loop back function. */
            CyFxSlFifoApplnStart ();
```

Note This solution may fail in passing USB compliance. You can implement another solution from the **USBBulkSourceSink** example (look for "**CyU3PUsbSetLinkPowerState (CyU3PUsbLPM_U0)**"), which is provided with the SDK. Use this solution in your final firmware or contact Cypress Technical Support for help.

If you see failures of BULK IN transfers on the FX3 DVK even after implementing this solution, then make sure that you short the 1 and 2 pins of jumper J100. Shorting these pins allows the FLAGA to be available for the FPGA connected to the Slave FIFO interface.

Symptom 2

Not getting a ZLP after reading data, which is multiples of the packet size (1024 bytes for USB 3.0, 512 bytes for USB 2.0) and less than the DMA buffer size in FX3, even though the PKTEND# signal is asserted without asserting SLWR#.

Reason and Solution

Assume the DMA buffer size is configured to 2 KB. FPGA writes 1 KB of data to the DMA buffer and it wants that data to be committed to the USB host. The USB host requests 2 KB of data from the FX3.

In this scenario, FX3 needs to send a ZLP along with the 1 KB of data to terminate the requested transfer from the USB host.

The GPIF II state machine of the Slave FIFO interface will be in the "Write" state when the FPGA is writing data to it. To send a ZLP immediately after writing data that is multiples of the packet size, the FPGA needs to assert the PKTEND# signal for at least two clock cycles after the SLWR# signal is de-asserted. This is because the GPIF II state machine requires two clock cycles to move from the "Write" state to the "ZLP" state.

Symptom 3

FLAGA is low immediately after downloading the firmware into FX3.

Reason and Solution

Check whether the DMA channel creation is successful. If the DMA channel associated with GPIF thread 0 fails, then the FLAGs associated with that thread reflect the wrong status. You can check the return value of the function **CyU3PDmaChannelCreate** to know whether the DMA channel creation is successful.

The **CyU3PDmaChannelCreate** function fails, mainly due to the following reasons. In these two cases, this function returns the CY_U3P_ERROR_MEMORY_ERROR code.

- DMA buffer allocation failure. All available (SYS MEM – CODE MEM) memory can be allocated for DMA buffers. System memory (SYS MEM) depends on the part number that you are using and CODE MEM depends on the application code that you develop. Make sure that the total DMA buffer size (number of buffers * each DMA buffer size) is less than the total buffer space available (SYS MEM – CODE MEM).
- DMA buffer descriptor allocation failure: The buffer descriptors are structures that keep track of the size and state of each DMA buffer. The system has 512 descriptors. Each buffer requires one descriptor for AUTO channels and two descriptors for MANUAL channels.

The number of DMA buffers should meet both these conditions.

Symptom 4

Many DMA overflow errors when FPGA is writing data to the Slave FIFO interface of FX3 at 100 MHz over the 32-bit data bus.

Reason and Solution

If the FX3 master clock is set to 384 MHz when using a 19.2-MHz crystal or clock source, and if the GPIF II is configured as 32-bit wide and is running at 100 MHz, then it may lead to DMA overflow errors on the GPIF II.

The setSysClk400 parameter of the **CyU3PSysClockConfig_t** structure specifies whether the FX3 device's master clock is to be set to a frequency greater than 400 MHz. Set this parameter to set the master clock frequency of FX3 to 403.2 MHz during the **CyU3PDeviceInit** call. This structure is passed as a parameter to the **CyU3PDeviceInit** call in the **main** function.

```
/* setSysClk400 clock configurations */
        clkCfg.setSysClk400 = CyTrue;   /* FX3 device's master clock is set to a frequency >
400 MHz */
        clkCfg.cpuClkDiv = 2;           /* CPU clock divider */
        clkCfg.dmaClkDiv = 2;           /* DMA clock divider */
        clkCfg.mmioClkDiv = 2;          /* MMIO clock divider */
        clkCfg.useStandbyClk = CyFalse; /* device has no 32KHz clock supplied */
        clkCfg.clkSrc = CY_U3P_SYS_CLK; /* Clock source for a peripheral block  */

    /* Initialize the device */
    status = CyU3PDeviceInit (&clkCfg);
    if (status != CY_U3P_SUCCESS)
    {
        goto handle_fatal_error;
    }
```

Note Contact Cypress Technical Support if the issue in your application is still not solved.

Appendix

This section shows you how the two data transfer modes (short packet and ZLP) of Slave FIFO application works when the DMA buffer size in FX3 and the transfer buffer size on the USB host application is changed.

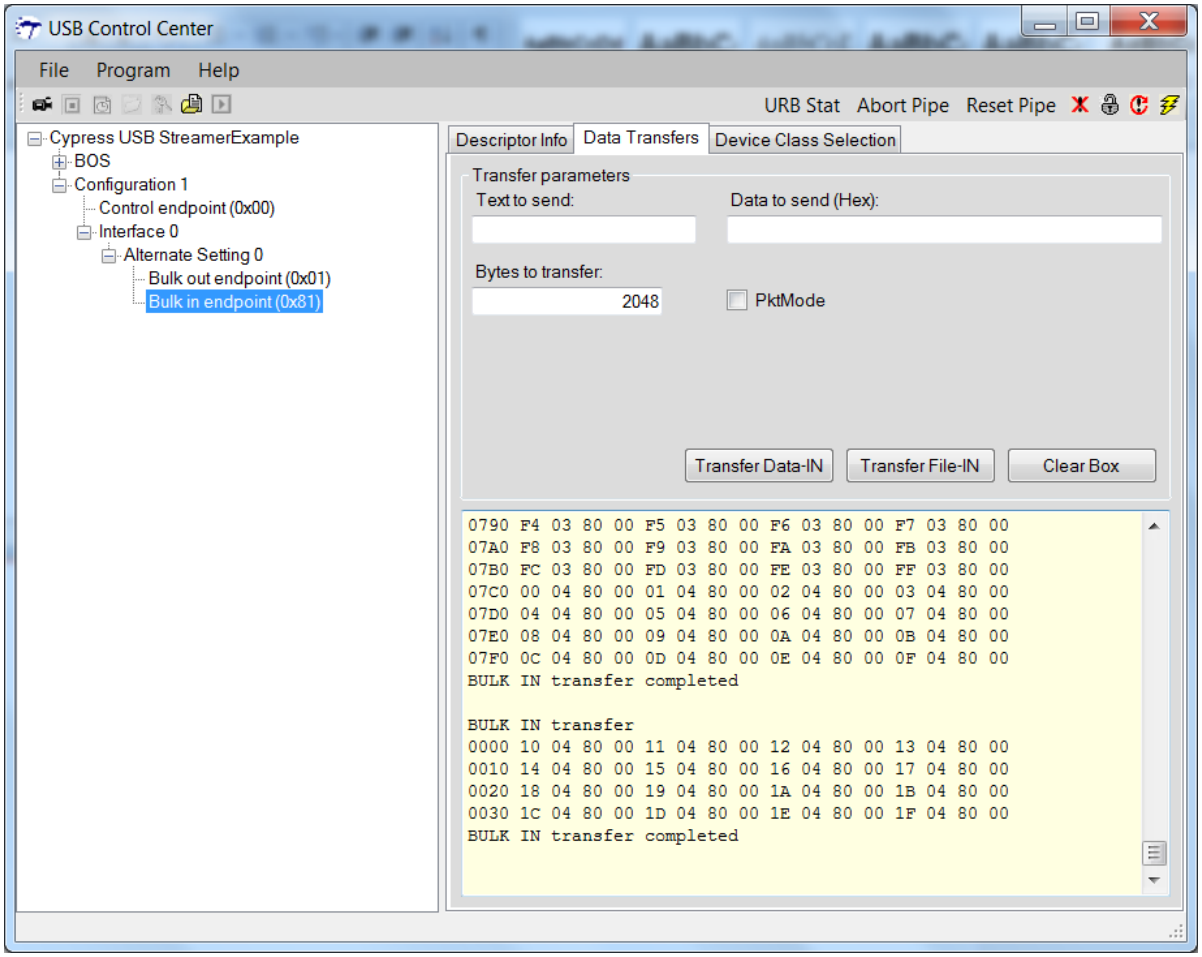
Short Packet Example

The DMA buffer size used in the provided FX3 firmware is 1024 bytes and the buffer count is 2. In this example, FPGA writes 1024 bytes of data to the first DMA buffer; then, it writes 64 bytes of data to the second DMA buffer.

You can read this data using the USB Control Center by entering 1024 in the **Bytes to transfer** field (transfer buffer size) and clicking the **Transfer Data-IN** button. You will get 1024 bytes of data; if you click the same button again, you will get the next 64 bytes of data written by the FPGA.

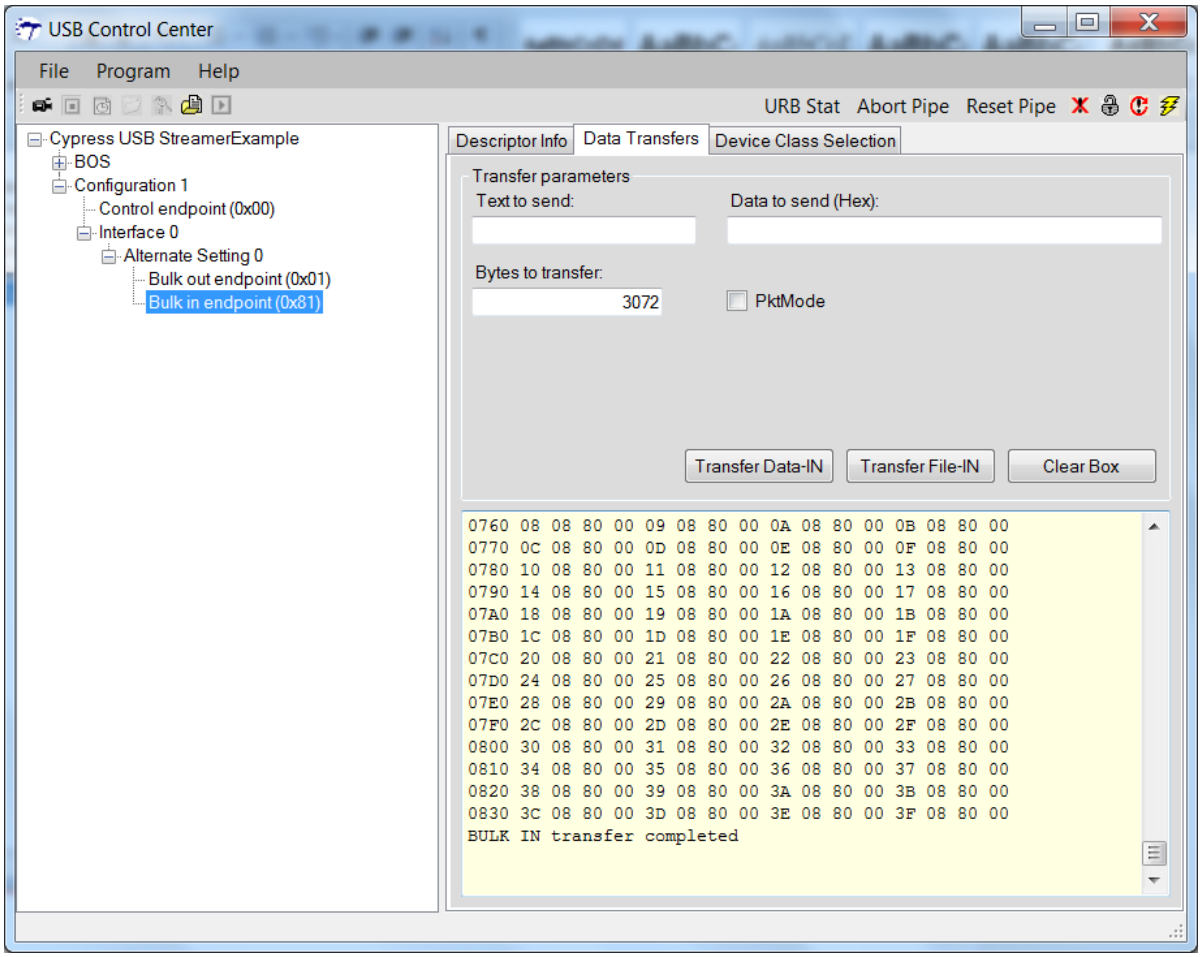
Assume that the DMA buffer size is modified to 2048 bytes. The FPGA writes 2048 bytes of data to the first DMA buffer and 64 bytes of data to the second DMA buffer. The FPGA fills the first DMA buffer completely and then writes a short packet to the next available DMA buffer. This (one full packet and one short packet) will be repeated if more DMA buffers are allocated. In the Control Center, read data by entering 2048 in the **Bytes to transfer** field and click the **Transfer Data-IN** button; this fetches you 2048 bytes of data. If you click the same button again, you will get the next 64 bytes of data written by the FPGA. This is shown in Figure 59.

Figure 59. Reading Short Packets Using Control Center



If you increase the transfer buffer size to 3072, then you will get 2112 (0x840) bytes of data every time you click **Transfer Data-IN**. This is illustrated in Figure 60.

Figure 60. Short Packet Data Transfers When Requested Bytes are More Than DMA Buffer Size

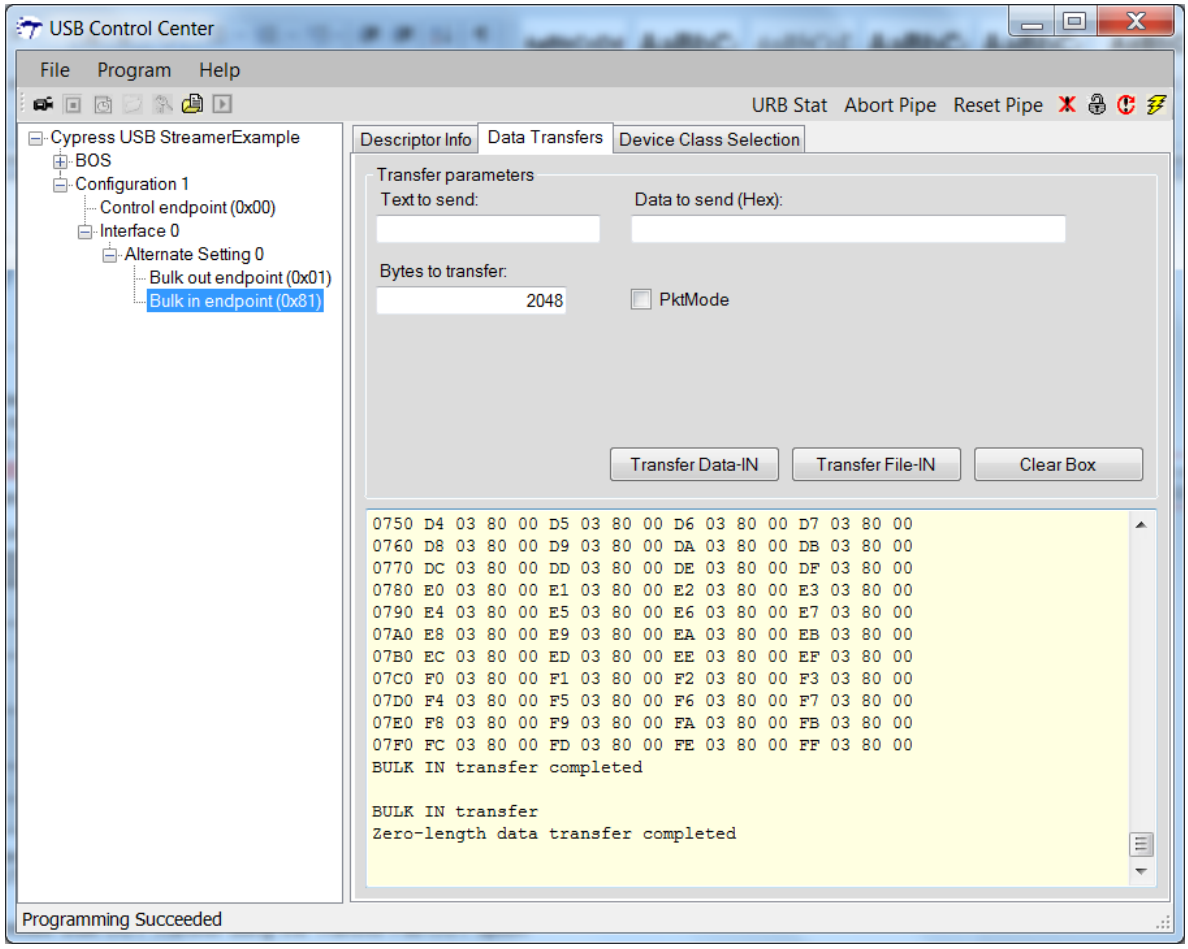


Zero Length Packet (ZLP) Example

The DMA buffer size used in the provided FX3 firmware is 1024 bytes and the buffer count is 2. In this example, FPGA writes 1024 bytes of data and asserts the control signals required to generate a ZLP. You can read this data using the USB Control Center, by entering 1024 in the **Bytes to transfer** field (transfer buffer size) and clicking the **Transfer Data-IN** button. You will get 1024 bytes of data; if you click the same button again, you will get a ZLP written by the FPGA.

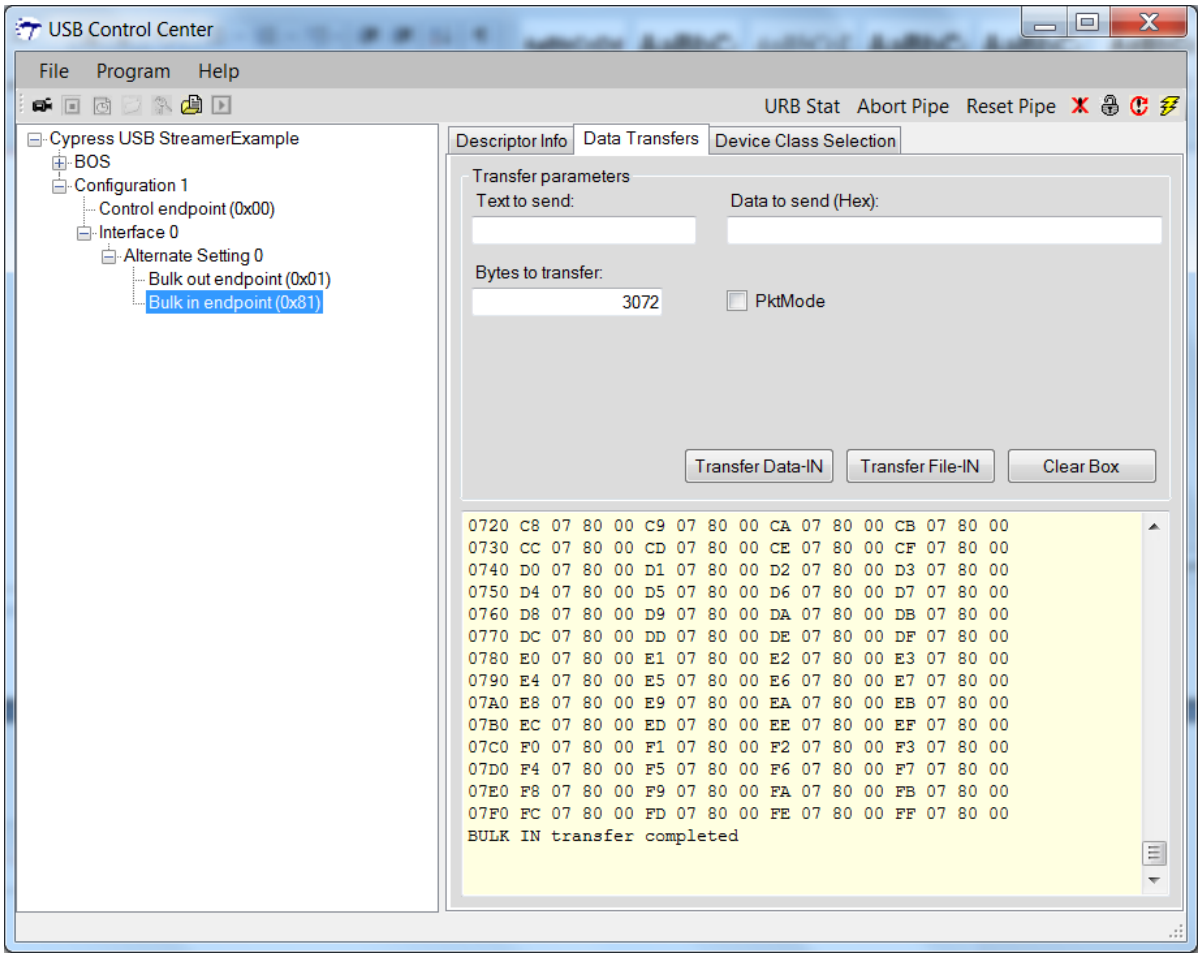
Assume that the DMA buffer size is modified to 2048 bytes. The FPGA writes 2048 bytes of data and asserts the control signals required to generate a ZLP. The FPGA fills the first DMA buffer completely and then generates the ZLP. This (one full packet and one ZLP) will be repeated if more DMA buffers are allocated. In the Control Center, read data by entering 2048 in the **Bytes to transfer** field and click the **Transfer Data-IN** button; this fetches you 2048 bytes of data. If you click the same button again, you will get a ZLP written by the FPGA. This is shown in Figure 61.

Figure 61. Reading ZLP Using Control Center



If you increase the transfer buffer size to 3072 (greater than the DMA buffer size), then you will get 2048 (0x800) bytes of data every time you click **Transfer Data-IN**. A ZLP is sent after 2048 bytes of data, but this will not be shown in the Control Center even though there is a ZLP on the physical USB bus. This is illustrated in Figure 62.

Figure 62. ZLP Transfers When Requested Bytes are More Than DMA Buffer Size



Document History

Document Title: AN65974 – Designing with the EZ-USB® FX3™ Slave FIFO Interface
Document Number: 001-65974

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3117206	OSG	12/21/2010	New application note.
*A	3261257	OSG	04/15/2011	Updated abstract; added information on synchronous Slave FIFO interfaces; expanded flag configuration section.
*B	3319015	OSG	07/18/2011	Updated description and pin mapping with information on 32-bit data bus support. Also added GPIO and serial interface availability to pin mapping. Clarified PKTEND# usage for short packets.
*C	3473293	OSG	12/22/2011	Updated Async Slave FIFO tPEL parameter Updated Sync Slave FIFO Read and Write Timing diagrams
*D	3656500	OSG	06/25/2012	Updated to new application note template Complete rewrite of application note
*E	3711053	OSG	08/13/2012	Corrected broken links in the document.
*F	3751229	OSG	09/21/2012	Added a design example describing how a Xilinx FPGA can be interconnected with FX3 over Slave FIFO Clarified the difference between Slave FIFO interface with 2 address lines and Slave FIFO interface with 5 address lines Added a timing diagram to show the latency when using a current thread FLAG Added an example application diagram
*G	3843462	OSG	12/17/2012	Template Update Added to the hardware setup options available for executing the design example provided Added pictures of the hardware setup Added a section on error conditions that may occur if FLAGs are violated
*H	4023240	RSKV	06/11/2013	Design example 1 is modified to have a Reset coming from FX3 to Xilinx FPGA. Operating instructions for Design example 1 are modified. Design example 2 (Interfacing FX3 to Altera FPGA) is added. Details of project files section is removed and Associated project files section is added at the end. Verilog and VHDL code is provided for both Xilinx and Altera FPGAs.
*I	4208738	RSKV	12/03/2013	Added the jumper and switch settings for both design examples. Corrected the Quartus II software download link. Added the Troubleshooting section. Added an Appendix to show the behavior of short packet and ZLP transfers when the DMA buffer size and the host application buffer size is changed.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com