Designing with the EZ-USB™ FX3 slave FIFO interface

About this document

Scope and purpose

This application note 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.

Intended audience

This document is primarily intended for anyone who wants to design with the EZ-USB™ FX3 slave FIFO interface.

Software version

EZ-USB™ FX3 SDK1.3.4

Table of contents

Table of contents

Table of contents

1 Introduction

The EZ-USB™ FX3 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 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 an 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 Spartan 6 FPGA from Xilinx and Cyclone III FPGA from Altera 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 SP601 evaluation kit for the Spartan 6 FPGA from Xilinx and a Cyclone III Starter Board from Altera for the Cyclone III FPGA from Altera, an FX3 development kit (DVK), and the FX3 software development kit (SDK).

2 More information

Infineon provides a wealth of data at **www.cypress.com** to help you to select the right device for your design, and to help you to integrate the device into your design quickly and effectively.

- Overview: **USB portfolio**, **USB roadmap**
- USB 3.0 product selectors: FX3, FX3S, CX3, HX3
- Application notes: Infineon offers a large number of USB application notes covering a broad range of topics, from basic to advanced level. Recommended application notes for getting started with FX3 are:
  - **AN75705** – Getting started with EZ-USB™ FX3
  - **AN70707** – EZ-USB™ FX3/FX3S hardware design guidelines and schematic checklist
  - **AN65974** – Designing with the EZ-USB™ FX3 slave FIFO interface
  - **AN75779** – How to implement an image sensor interface with EZ-USB™ FX3 in a USB video class (UVC) framework
  - **AN86947** – Optimizing USB 3.0 throughput with EZ-USB™ FX3
  - **AN84868** – Configuring an FPGA over USB using EZ-USB™ FX3
  - **AN68829** – Slave FIFO interface for EZ-USB™ FX3: 5-bit address mode
  - **AN76348** – Differences in implementation of EZ-USB™ FX2LP and EZ-USB™ FX3 applications
  - **AN89661** – USB RAID 1 disk design using EZ-USB™ FX3S
- Code examples:
  - **USB Hi-Speed**
  - **USB Full-Speed**
  - **USB SuperSpeed**
- Technical reference manual (TRM):
  - **EZ-USB™ FX3 technical reference manual**
- Development kits:
  - **CYUSB3KIT-003, EZ-USB™ FX3 SuperSpeed explorer kit**
  - **CYUSB3KIT-001, EZ-USB™ FX3 development kit**
  - Models: **IBIS**

2.1 EZ-USB™ FX3 software development kit

Infineon delivers the complete software and firmware stack for FX3 to easily integrate SuperSpeed USB into any embedded application. The **software development kit** (SDK) comes with tools, drivers, and application examples, which help accelerate application development.

2.2 GPIF II designer

The **GPIF II designer** is a graphical software that allows designers to configure the GPIF II interface of the EZ-USB™ FX3 USB 3.0 Device Controller.

The tool allows users the ability to select from one of five Infineon-supplied interfaces, or choose to create their own GPIF II interface from scratch. Infineon has supplied industry-standard interfaces such as asynchronous and synchronous slave FIFO, and asynchronous and synchronous SRAM. Designers who already have one of these pre-defined interfaces in their system can simply select the interface of choice, choose from a set of standard parameters such as bus width (x8, x16, x24, x32) endianness, clock settings, and then compile the interface. The tool has a streamlined three-step GPIF interface development process for users who need a

More information

customized interface. Users can first select their pin configuration and standard parameters. Secondly, they can design a virtual state machine using configurable actions. Finally, users can view the output timing to verify that it matches the expected timing. After this three-step process is complete, the interface can be compiled and integrated with FX3.

3 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, 24-bit and 32-bit parallel data bus
- Enables interface frequencies up to 100 MHz
- Supports 14 configurable control pins when a 32-bit data bus is used; all control pins can be either input/output or bidirectional
- Supports 16 configurable control pins when a 16/8 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.

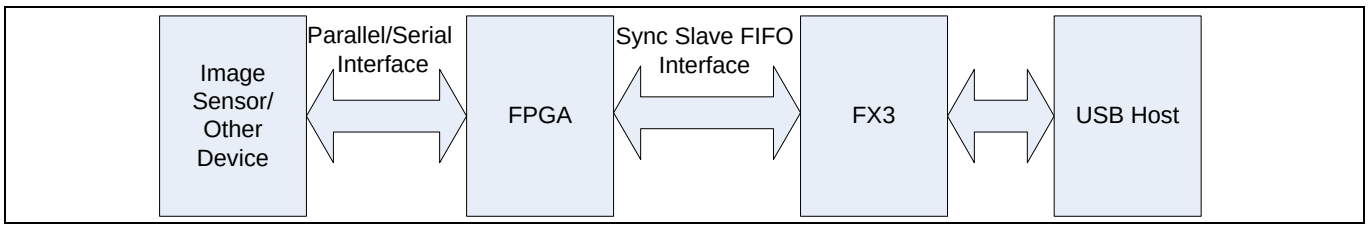

Figure 1 Example application diagram

4 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. **Table 1** describes the signals shown in **Figure 2**.

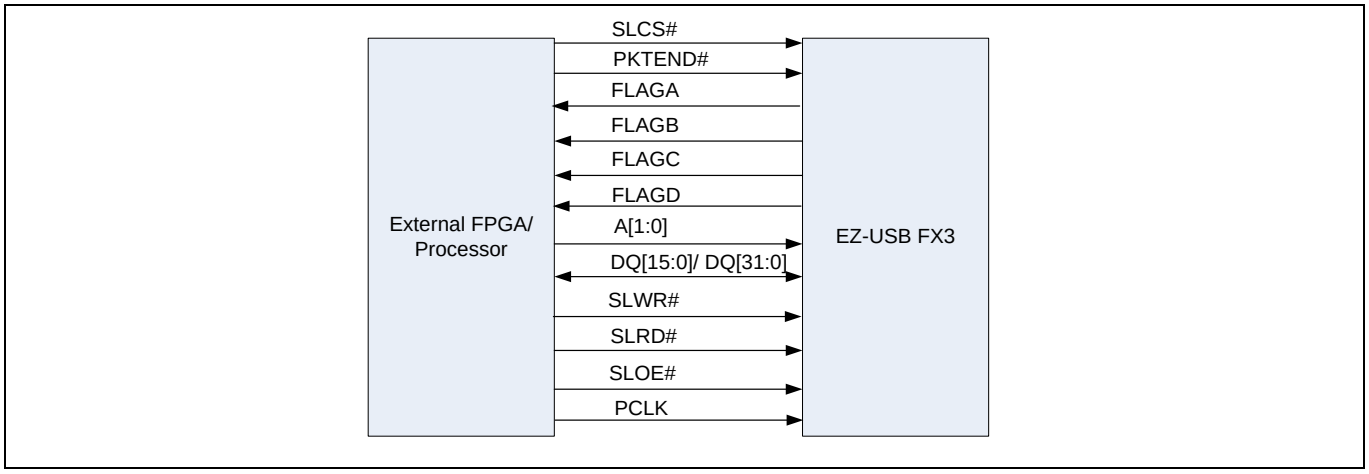

Figure 2 Synchronous slave FIFO interface diagram

Table 1 Synchronous slave FIFO interface signals

Signal name	Signal description
SLCS#	This is the chip select signal for the slave FIFO interface. It must be asserted to access the 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/FLAGC/ FLAGD	These are the flag outputs from FX3. The flags indicate the availability of an FX3 socket.1 In the attached example projects, FLAGA and FLAGB are used for the slave FIFO write operation and FLAGC and FLAGD are used for the slave FIFO read operation.
A[1:0]	This is the 2-bit address bus of slave FIFO.
DQ[15:0]/ DQ[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.

4.1 Difference between slave FIFO with two and five address lines

The synchronous slave FIFO interface with two address lines supports 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.

4.2 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/I^2S) 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 GPIF II designer tool. More information is provided in the **Flag configuration** section.

Table 2 Pin mapping for slave FIFO interface

EZ-USB™ FX3 pin	Pin naming as in SuperSpeed explorer kit/ interconnect board	Synchronous slave FIFO interface with 8-bit data bus	Synchronous slave FIFO interface with 16-bit data bus	Synchronous slave FIFO interface with 24-bit data bus	Synchronous slave FIFO interface with 32-bit data bus
GPIO[17]	CTL[0]	SLCS#	SLCS#	SLCS#	SLCS#
GPIO[18]	CTL[1]	SLWR#	SLWR#	SLWR#	SLWR#
GPIO[19]	CTL[2]	SLOE#	SLOE#	SLOE#	SLOE#
GPIO[20]	CTL[3]	SLRD#	SLRD#	SLRD#	SLRD#
GPIO[21]	CTL[4]	FLAGA	FLAGA	FLAGA	FLAGA
GPIO[22]	CTL[5]	FLAGB	FLAGB	FLAGB	FLAGB
GPIO[23]	CTL[6]	FLAGC	FLAGC	FLAGC	FLAGC
GPIO[24]	CTL[7]	PKTEND#	PKTEND#	PKTEND#	PKTEND#
GPIO[25]	CTL[8]	FLAGD	FLAGD	FLAGD	FLAGD
GPIO[28]	CTL[11]	GPIO	A1	GPIO	A1
GPIO[29]	CTL[12]	GPIO	A0	GPIO	A0
GPIO[0:7]	DQ[0:7]	DQ[0:7]	DQ[0:7]	DQ[0:7]	DQ[0:7]
GPIO[8]	DQ[8]	A0	DQ[8]	DQ[8]	DQ[8]
GPIO[9]	DQ[9]	A1	DQ[9]	DQ[9]	DQ[9]

Synchronous slave FIFO interface

EZ-USB™ FX3 pin	Pin naming as in SuperSpeed explorer kit/ interconnect board	Synchronous slave FIFO interface with 8-bit data bus	Synchronous slave FIFO interface with 16-bit data bus	Synchronous slave FIFO interface with 24-bit data bus	Synchronous slave FIFO interface with 32-bit data bus
GPIO[10:15]	DQ[10:15]	Available as GPIOs	DQ[10:15]	DQ[10:15]	DQ[10:15]
GPIO[16]	PCLK	PCLK	PCLK	PCLK	PCLK
GPIO[33:40]	DQ[16:23]	Available as GPIOs	Available as GPIOs	DQ[16:23]	DQ[16:23]
GPIO[41]	DQ[24]	GPIO	GPIO	A0	DQ24
GPIO[42]	DQ[25]	GPIO	GPIO	A1	DQ25
GPIO[43:44]	DQ[26:27]	Available as GPIOs	Available as GPIOs	Available as GPIOs	DQ[26:27]
GPIO[45]	IO45	GPIO	GPIO	GPIO	GPIO
GPIO[46]	DQ[28]	GPIO/UART_RTS	GPIO/UART_RTS	GPIO/UART_RTS	DQ[28]
GPIO[47]	DQ[29]	GPIO/UART_CTS	GPIO/UART_CTS	GPIO/UART_CTS	DQ[29]
GPIO[48]	DQ[30]	GPIO/UART_TX	GPIO/UART_TX	GPIO/UART_TX	DQ[30]
GPIO[49]	DQ[31]	GPIO/UART_RX	GPIO/UART_RX	GPIO/UART_RX	DQ[31]
GPIO[50]	I2S_CLK	GPIO/I2S_CLK	GPIO/I2S_CLK	GPIO/I2S_CLK	GPIO/I2S_CLK
GPIO[51]	I2S_SD	GPIO/I2S_SD	GPIO/I2S_SD	GPIO/I2S_SD	GPIO/I2S_SD
GPIO[52]	I2S_WS	GPIO/I2S_WS	GPIO/I2S_WS	GPIO/I2S_WS	GPIO/I2S_WS
GPIO[53]	RTS/SCK	GPIO/SPI_SCK/UART_RTS	GPIO/SPI_SCK/UART_RTS	GPIO/SPI_SCK/UART_RTS	GPIO/UART_RTS
GPIO[54]	CTS/SSN	GPIO/SPI_SSN/UART_CTS	GPIO/SPI_SSN/UART_CTS	GPIO/SPI_SSN/UART_CTS	GPIO/UART_CTS
GPIO[55]	TX/MOSI	GPIO/SPI_MISO/UART_TX	GPIO/SPI_MISO/UART_TX	GPIO/SPI_MISO/UART_TX	GPIO/UART_TX
GPIO[56]	RX/MISO	GPIO/SPI_MOSI/UART_RX	GPIO/SPI_MOSI/UART_RX	GPIO/SPI_MOSI/UART_RX	GPIO/UART_RX
GPIO[57]	I2S_MCLK	GPIO/I2S_MCLK	GPIO/I2S_MCLK	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.*

5 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 two-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. The GPIF-II designer uses active LOW GPIO setting for the DMA flags for the timing diagrams shown in the following sections.

5.1 Synchronous slave FIFO interface timing

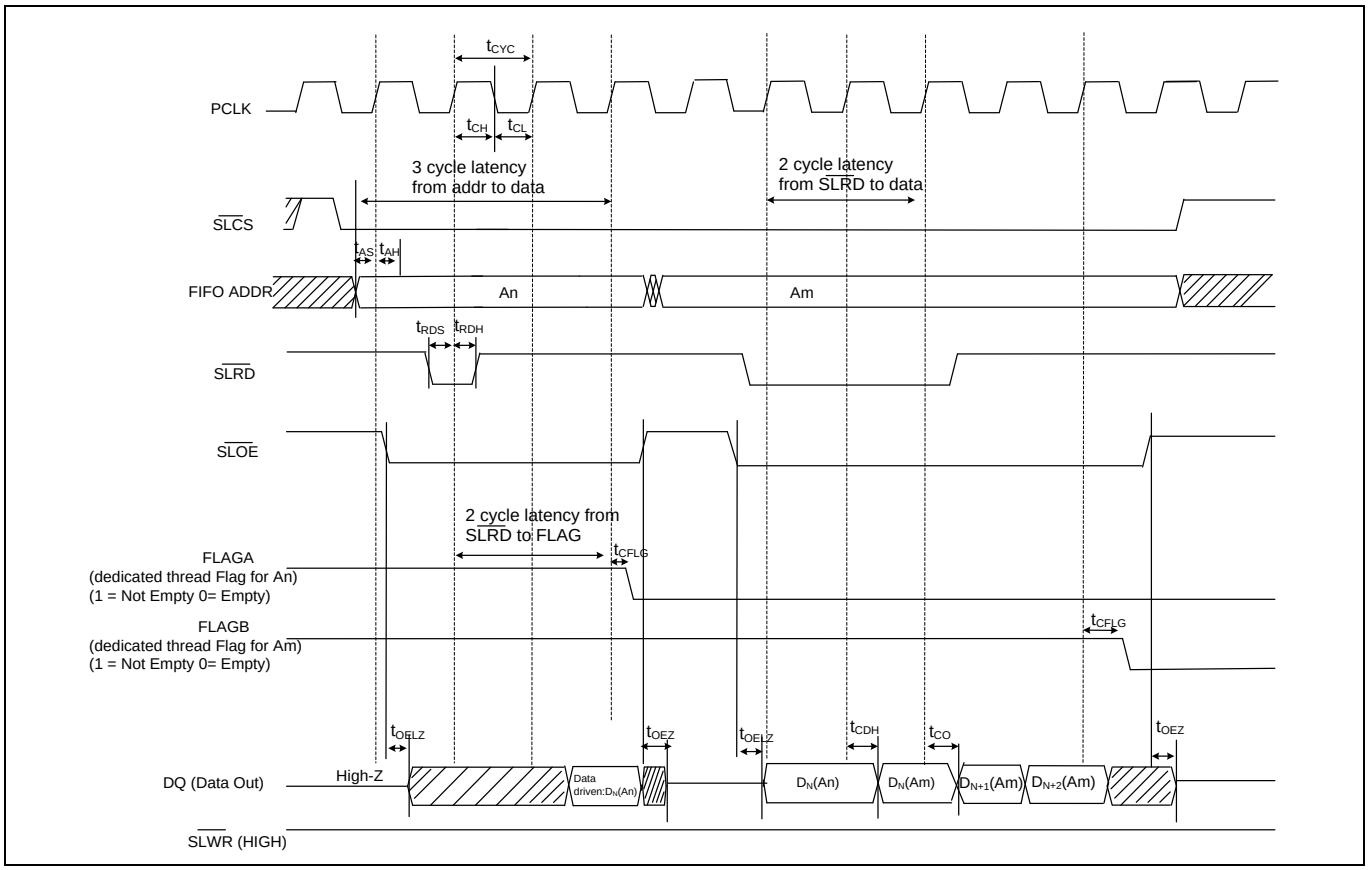

Figure 3 Synchronous slave FIFO read sequence

5.2 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 t_{CO} (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.

Slave FIFO access sequence and interface timing

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.

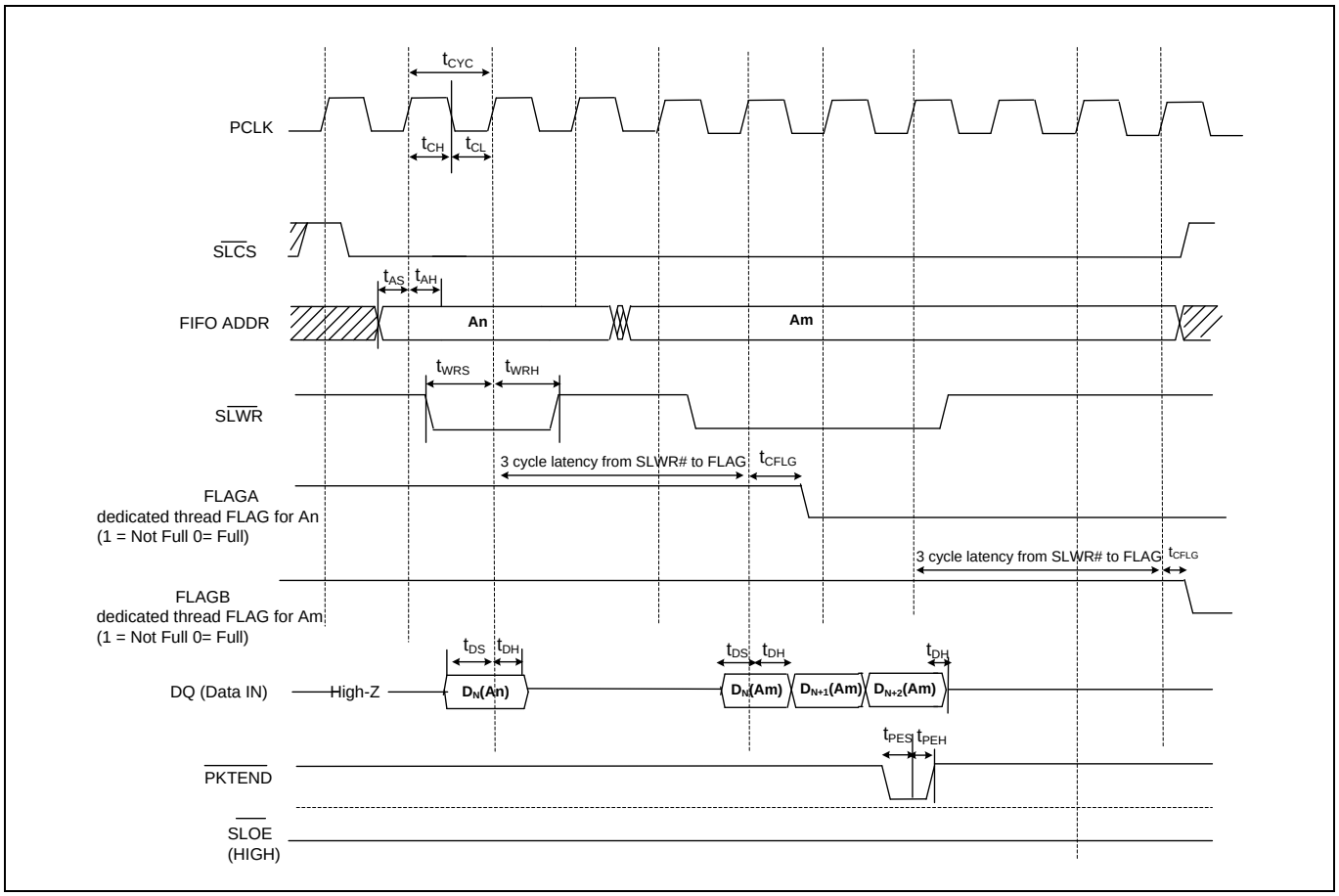

Figure 4 Synchronous slave FIFO write sequence

Slave FIFO access sequence and interface timing

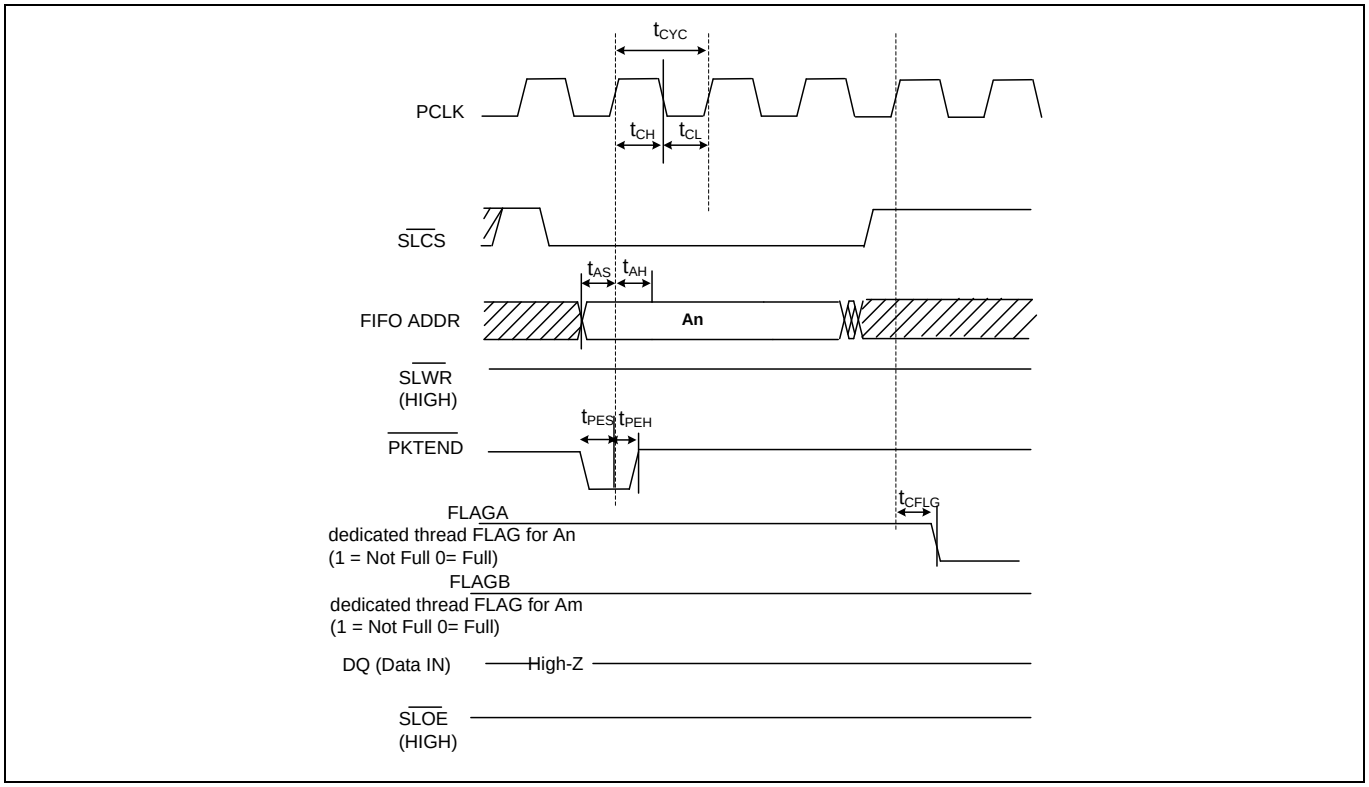

Figure 5 Synchronous ZLP write cycle timing

5.3 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 t_{CFLG} 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.

Slave FIFO access sequence and interface timing

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
t_{CYC}	Clock period	10	–	ns
t_{CH}	Clock HIGH time	4	–	ns
t_{CL}	Clock LOW time	4	–	ns
t_{RDS}	SLRD# to CLK setup time	2	–	ns
t_{RDH}	SLRD# to CLK hold time	0.5	–	ns
t_{WRS}	SLWR# to CLK setup time	2	–	ns
t_{WRH}	SLWR# to CLK hold time	0.5	–	ns
t_{CO}	Clock to valid data	–	7	ns
t_{DS}	Data input setup time	2	–	ns
t_{DH}	CLK to data input hold	0	–	ns
t_{AS}	Address to CLK setup time	2	–	ns
t_{AH}	CLK to Address hold time	0.5	–	ns
t_{OELZ}	SLOE# to data low-Z	0	–	ns
t_{CFLG}	CLK to flag output propagation delay	–	8	ns
t_{OEZ}	SLOE# deassert to data HI-Z	–	8	ns
t_{PES}	PKTEND# to CLK setup	2	–	ns
t_{PEH}	CLK to PKTEND# hold	0.5	–	ns
t_{CDH}	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 GPIF II 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.

6 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**.*

Threads and sockets

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

Note: *For more information on FX3 threads and sockets, refer to Section 7.4.6 of the **EZ-USB™ FX3 technical reference manual**.*

7 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. Hence, 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.

8 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 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 flag types are described in the following sections. Different flag configurations result in different latencies, which are summarized in **Table 4**.

8.1 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.

Here, 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**.

8.2 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.)

Flag configuration

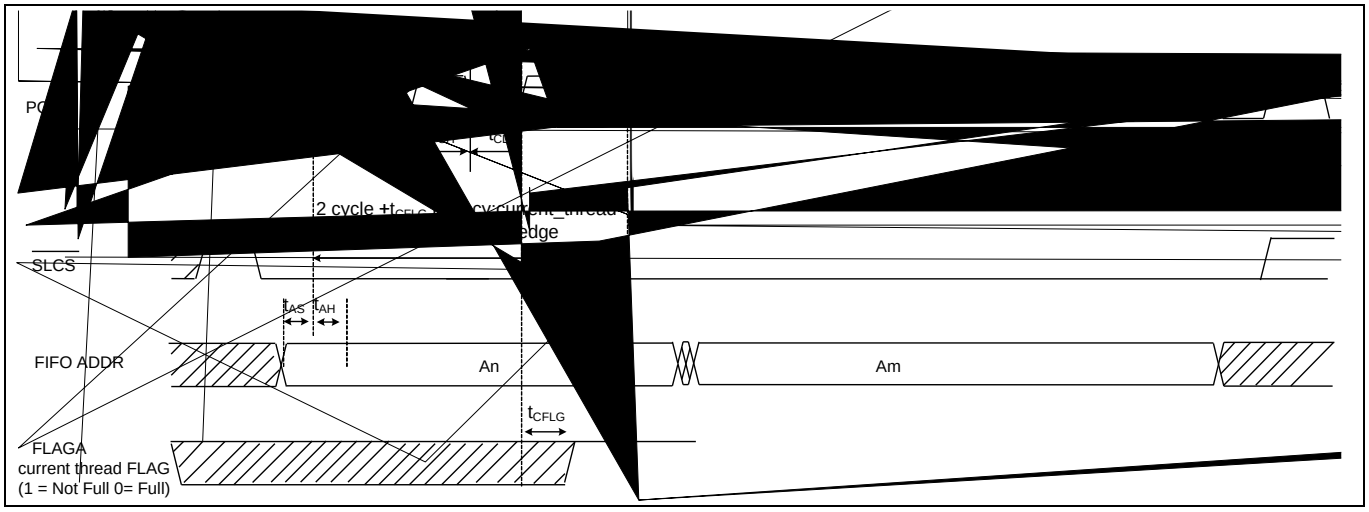

Figure 6 Additional latency incurred at start of transfer when using a current thread flag

*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**.

8.2.1 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 GPIF II designer for a particular flag. Examples and screenshots of the GPIF II designer settings for flags are available in the **Flag configuration** section.

Flag configuration

Table 4 Latencies associated with different flag configurations

Flag configuration	GPIF II 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 + t_{CFLG} (external device can sample valid flag on the fourth clock edge)	2 cycles + t_{CFLG} (external device can sample valid flag on the third clock edge)	N/A
Full/Empty flag for currently addressed thread	Current_thread_DMA_Ready	2 cycles + t_{CFLG} (external device can sample valid flag on the third clock edge)	3 cycles + t_{CFLG} (external device can sample valid flag on the fourth clock edge)	2 cycles + t_{CFLG} (external device can sample valid flag on the 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 CyU3PGpifSocketConfigure() API. **Note** Watermark is in terms of a 32-bit data word. Examples: CyU3PgpifSocketConfigure

Flag configuration

Flag configuration	GPIF II 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)	
					(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 + t_{CFLG} (external device can sample valid flag on the third clock edge)	Dependent on watermark level	Dependent on watermark level	Set watermark level by calling the CyU3PGpiIfSocketConfigure() 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 following sections describe how to configure flags using the GPIF II designer tool and the EZ-USB™ FX3 SDK.

9 GPIF II designer

9.1 Implementing a synchronous slave FIFO interface

You will find the GPIF II implementation of the slave FIFO interface by installing the GPIF II designer tool. You can install the GPIF II designer tool from **www.cypress.com**. When you launch GPIF II designer, you will find the **Supplied Interfaces** on the Start Page.

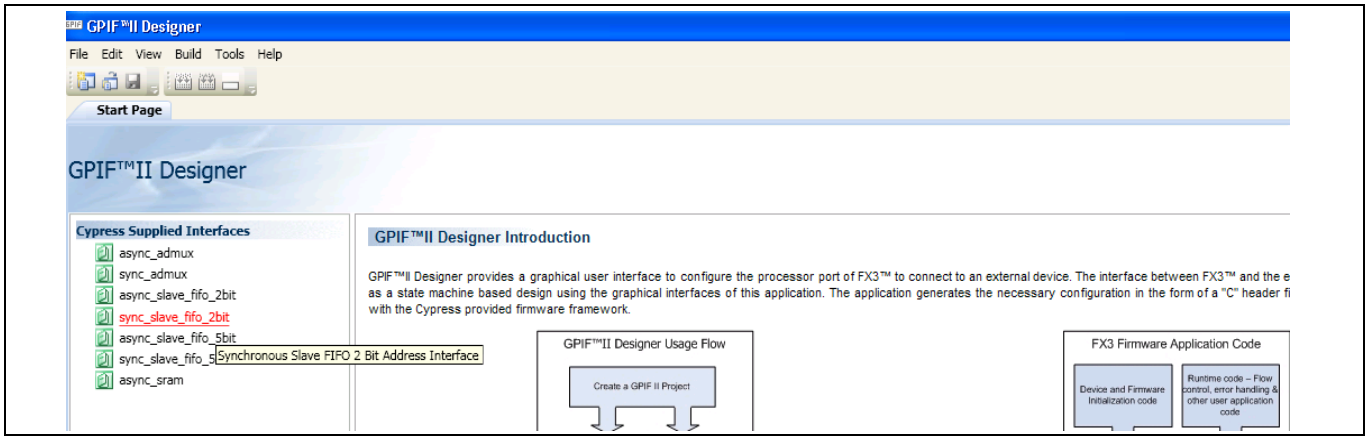

Figure 7 Slave FIFO projects in GPIF II designer –supplied interfaces

The sync_slave_fifo_2bit project is the GPIF II implementation of the synchronous slave FIFO interface with a two-bit address. The following section explains how a partial flag may be configured using GPIF II designer.

9.2 Configuring a partial flag

A partial flag is configured using the following steps:

- The partial flag setting must be selected in the GPIF II designer tool.
- The watermark level for the partial flag must be set using the CyU3PGpifSocketConfigure() API in firmware

In the GPIF II designer, open the **sync_slave_fifo_2bit** project from the 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”.

GPIF II designer

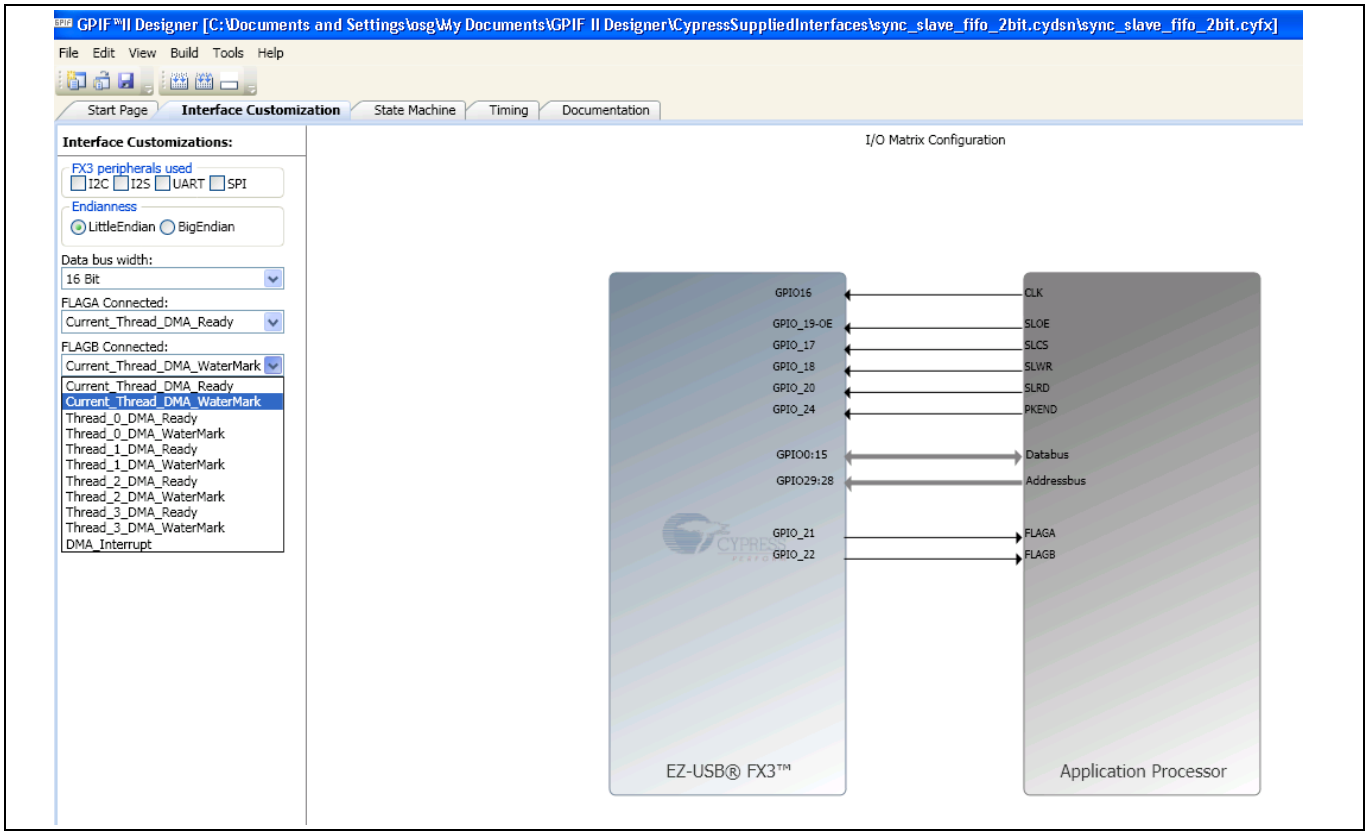

Figure 8 Flag settings in GPIF II designer-supplied interfaces sync_slave_fifo_2bit

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.

GPIF II designer

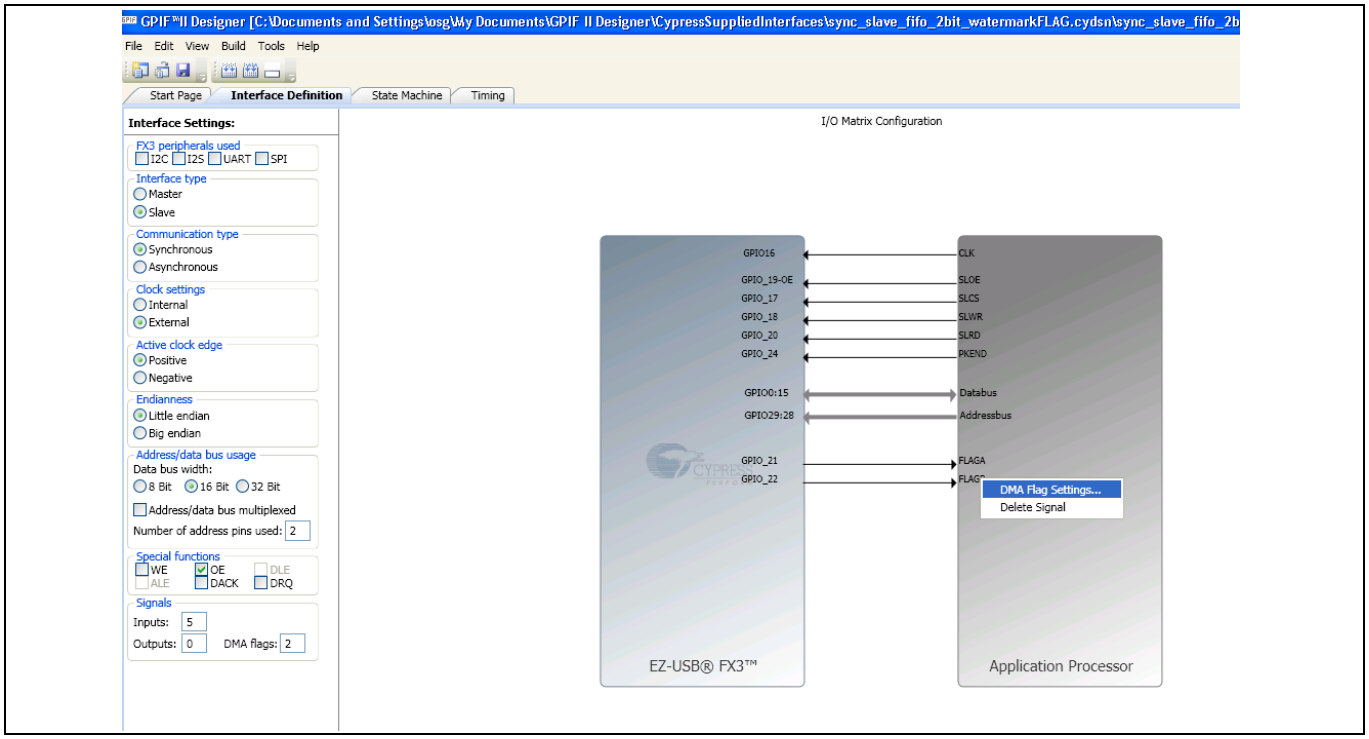

Figure 9 Flag settings in new project by "Save Project as Editable" on the default sync_slave_fifo_2bit

The following figure shows you how to select the flag configuration.

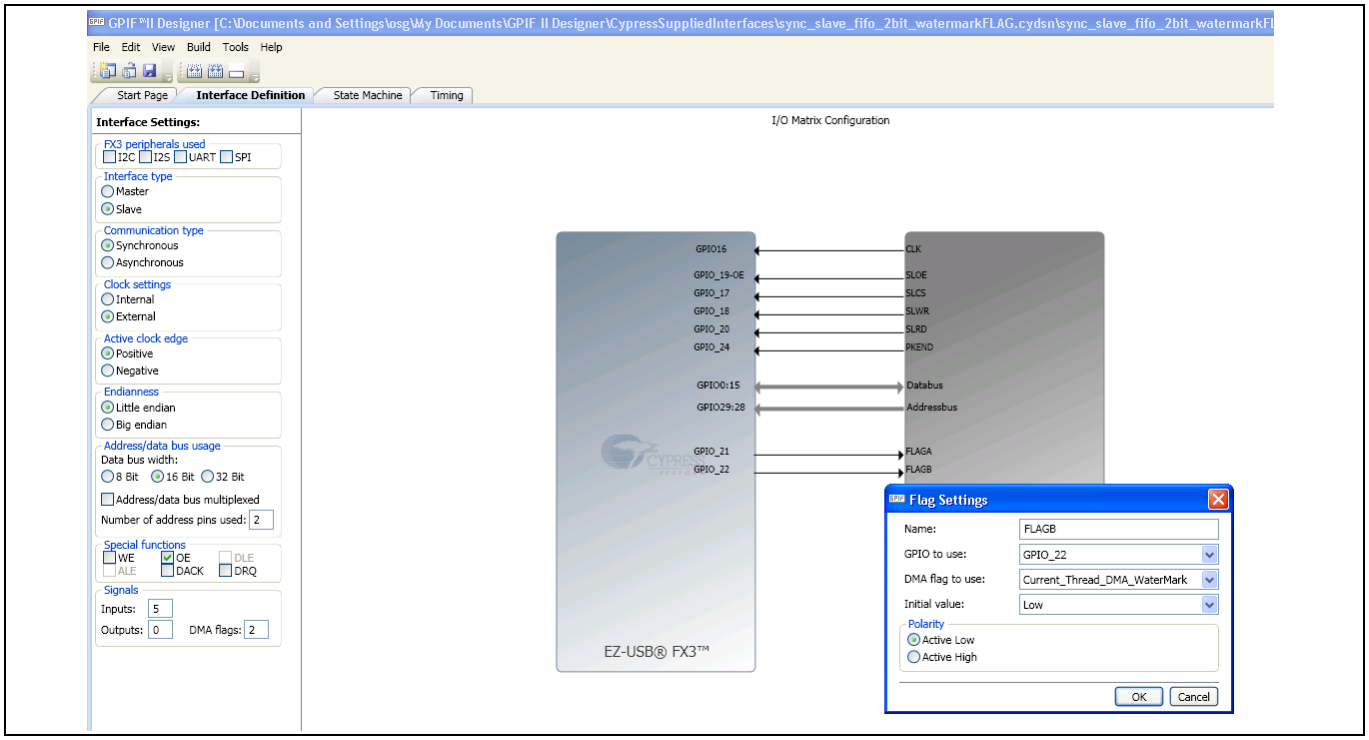

Figure 10 Selecting specific flag settings

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.

9.3 General formulae 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 formulae 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. The data word size will be based on bus width. For example, if the bus width is 24-bit, the number of bytes that may be read or written will be the number of data words (obtained using the following formulae) x 3 (bus width in bytes).*

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 a 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.

9.4 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.*

9.4.1 Example 1

Slave FIFO with 32-bit data bus:

- In GPIF II 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 (the partial flag) goes LOW in the same cycle in which the last word of data is written.

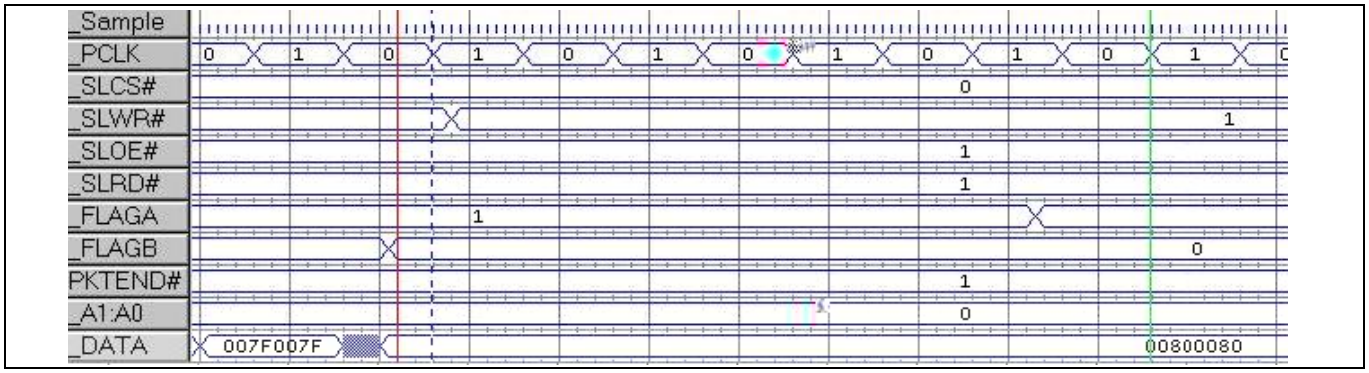

Figure 11 Burst write transfer with 32-bit data bus width (FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 4)

9.4.2 Example 2

Slave FIFO with 32-bit data bus:

- In GPIF II 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 (the 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 **General formulae for using partial flags** section 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

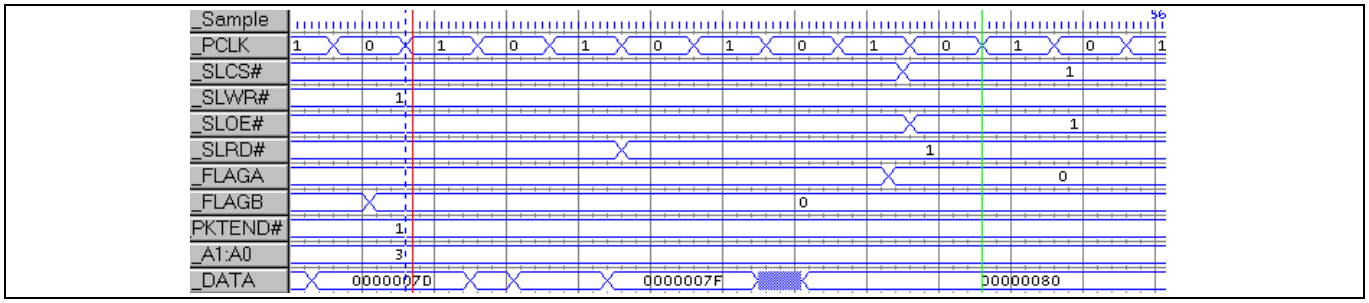

Figure 12 Burst read transfer with 32-bit data bus width (FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 4)

9.4.3 Example 3

Slave FIFO with 16-bit data bus:

- In GPIF II 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 (the 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 (the 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 **General formulae for using partial flags** section 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

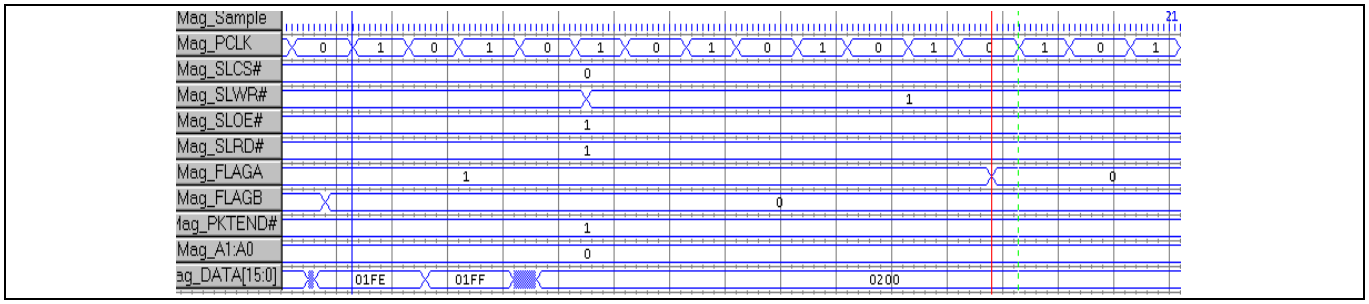

Figure 13 Burst write transfer with 16-bit data bus width (FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 3)

9.4.4 Example 4

Slave FIFO with 16-bit data bus:

- In GPIF II 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 (the 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.

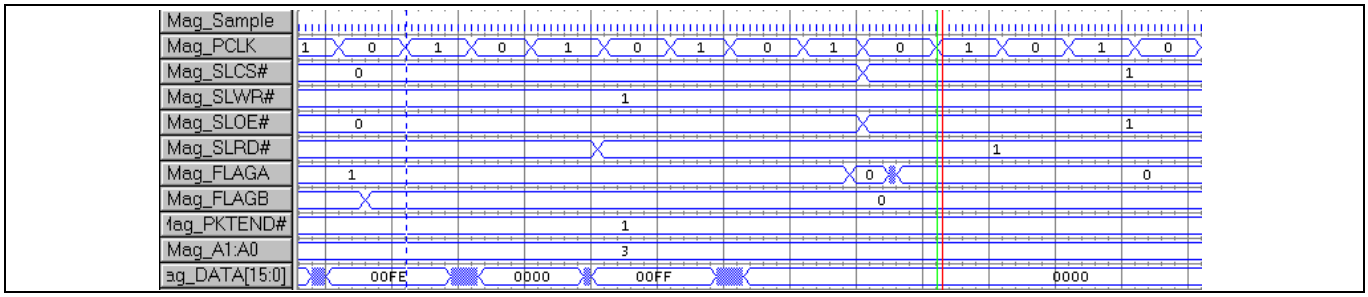

Figure 14 Burst read transfer with 16-bit data bus width (FLAGA= Current_thread_DMA_RDY, FLAGB= Current_thread_DMA_watermark, watermark = 2)

9.5 Other considerations when using the 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. The burst value can be set in the CyU3PGpifSocketConfigure() API as the last parameter. By setting the burst value, the DMA hardware will check for the watermark value after every burst and not after every single word. 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' in CyU3PGpifSocketConfigure() API according to the formula mentioned in the **General formulae for using partial flags** section.
  - 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.

9.6 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.

GPIF II designer

The error code indicates the thread on which the error occurred, as shown in **Table 5**. If one of these errors does occur, it 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 15**.

```
/* 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 overrun/underrun 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

GPIF II designer

PIB error code	Description
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

10 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 GPIF II designer tool, the updated configuration needs to be integrated into the firmware. After building the project in GPIF II 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\1.3\firmware\slavefifo_examples\slfifosync.*

This firmware example supports both 16-bit 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.*

11 Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

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

11.1 Hardware setup

The project provided in this example can be executed on a hardware setup consisting of an **FX3 development kit (CYUSB3KIT-001)** or **SuperSpeed explorer kit (CYUSB3KIT-003)** interconnected with a **Spartan 6 SP601 evaluation kit** from Xilinx. The FX3 board and the board from Xilinx are connected using a Samtec to FMC interconnect board. The **CYUSB3ACC-002 interconnect board** mates with the Samtec connector on the FX3 development kit (CYUSB3KIT-001) and the FMC connector on the board from Xilinx.

The **CYUSB3ACC-005 interconnect board** mates with the headers on the SuperSpeed Explorer Kit (CYUSB3KIT-003) and the FMC connector on the board from Xilinx. **Figure 15** shows the hardware setup using SuperSpeed Explorer Kit. See **Appendix B: Hardware setup using FX3 DVK (CYUSB3KIT-001)**. Other than the hardware setup, the following steps are common irrespective of the FX3 board that you are using.

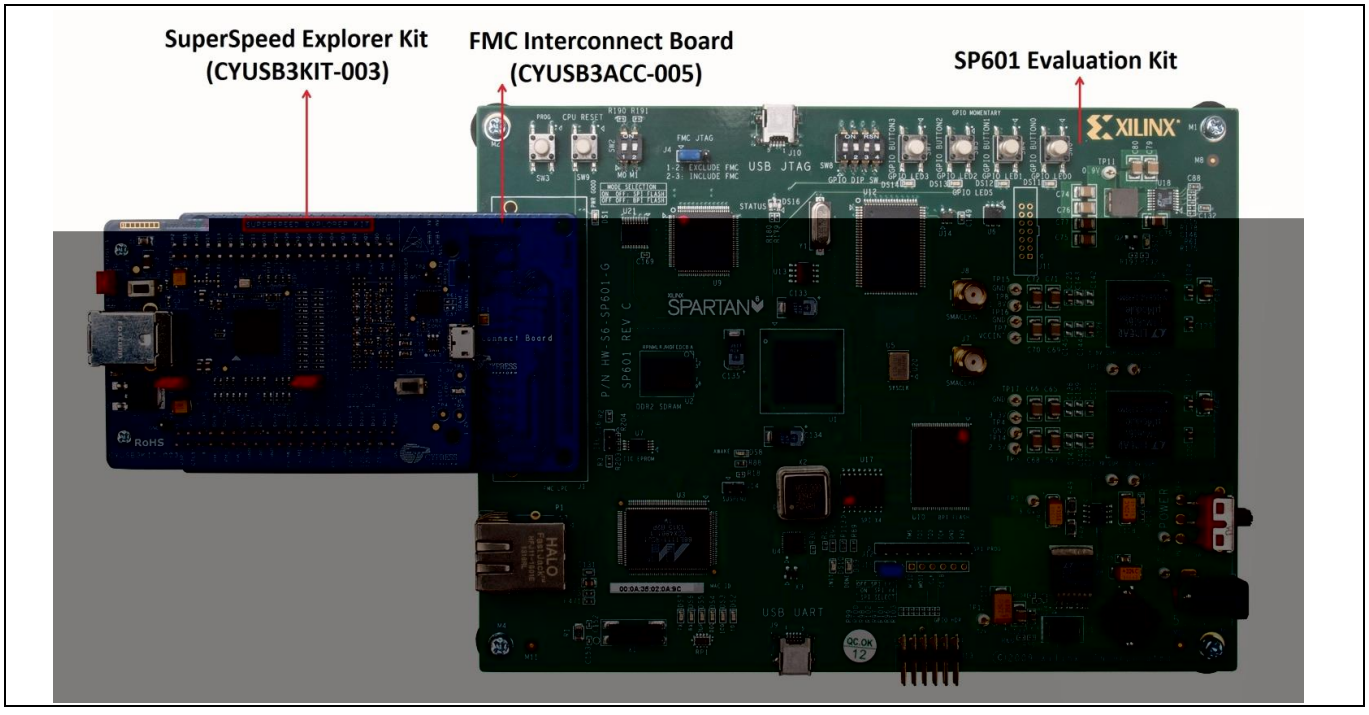

Figure 15 SuperSpeed explorer kit connected to SP601 board from Xilinx using FMC interconnect board

Note: Make sure that jumper J5 on the SuperSpeed Explorer kit is open in all applications where FPGA is interfaced to FX3 over the slave FIFO interface.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

11.2 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.

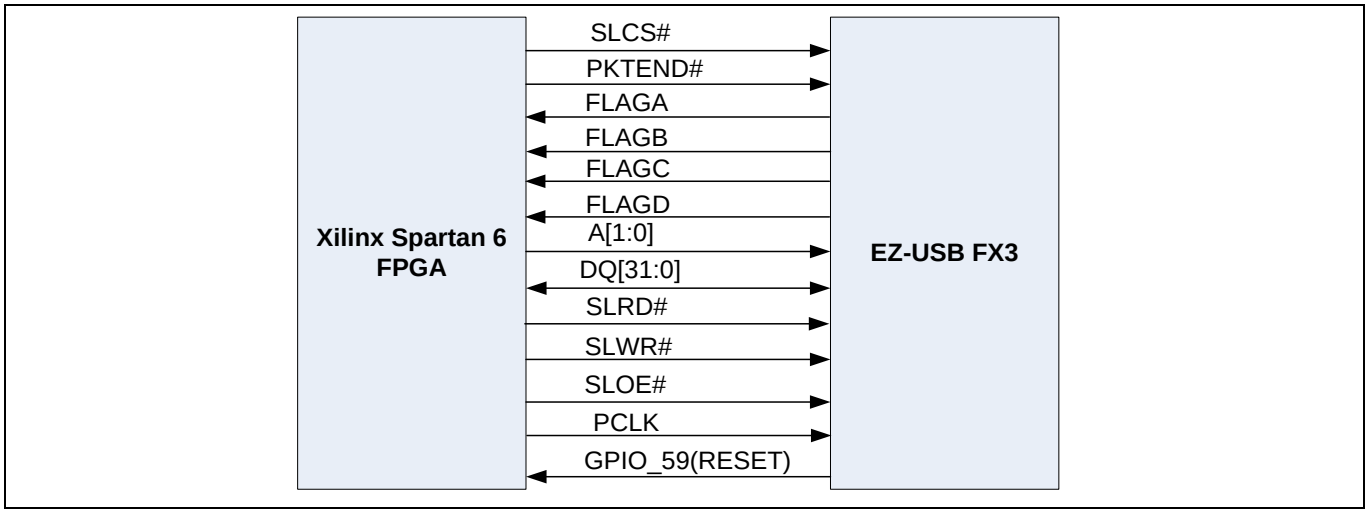

Figure 16 Interconnect diagram between FPGA and FX3

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.

11.3 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 VID/PID, 0x04B4/0x00F1. This enables the use of the Control center and sreamer utilities for initiating USB transfers.
- Integrates the synchronous slave FIFO descriptor, which:

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

- 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:

a) FLAGA: Full flag dedicated to thread0

b) FLAGB: Partial flag with watermark value 6, dedicated to thread0

c) FLAGC: Empty flag dedicated to thread3

d) FLAGD: Partial flag with watermark value 6, dedicated to thread3

Note: The GPIF II 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:
    - 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.
    - 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.

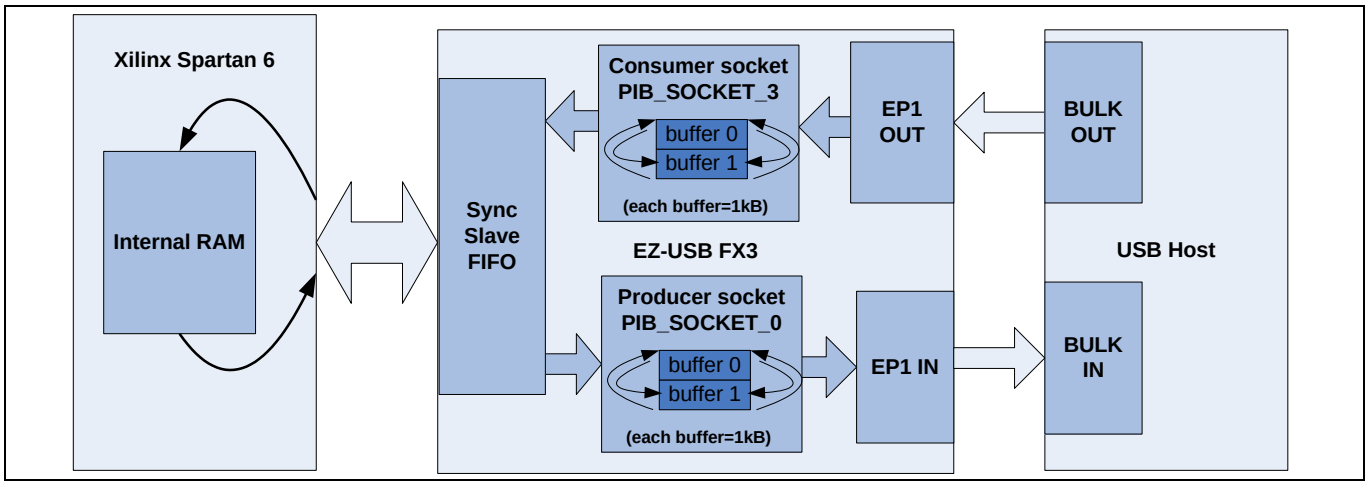

Figure 17 Setup for loopback transfer

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

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

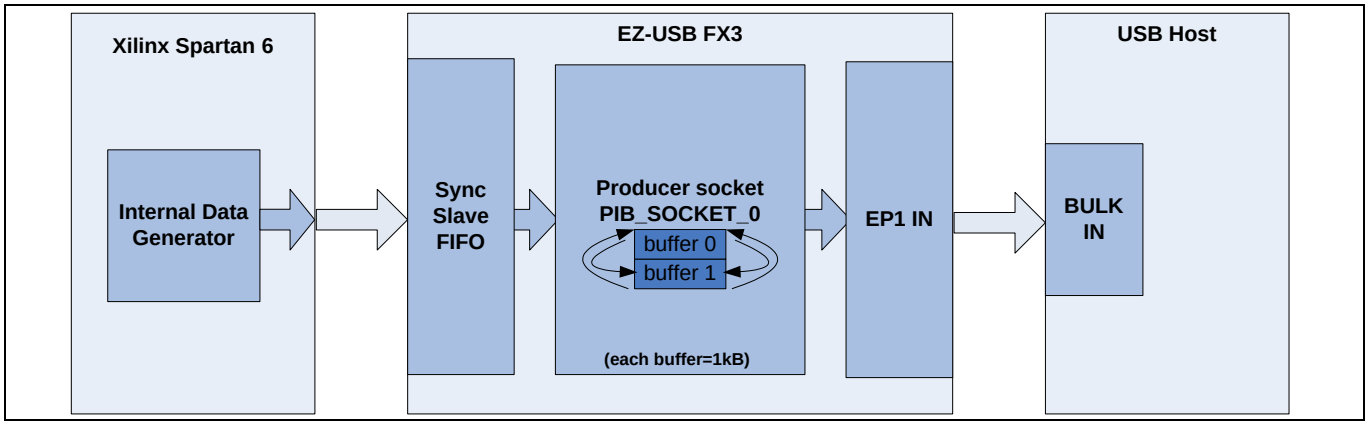

Figure 18 Setup for short packet and ZLP transfers

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:
  - 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.

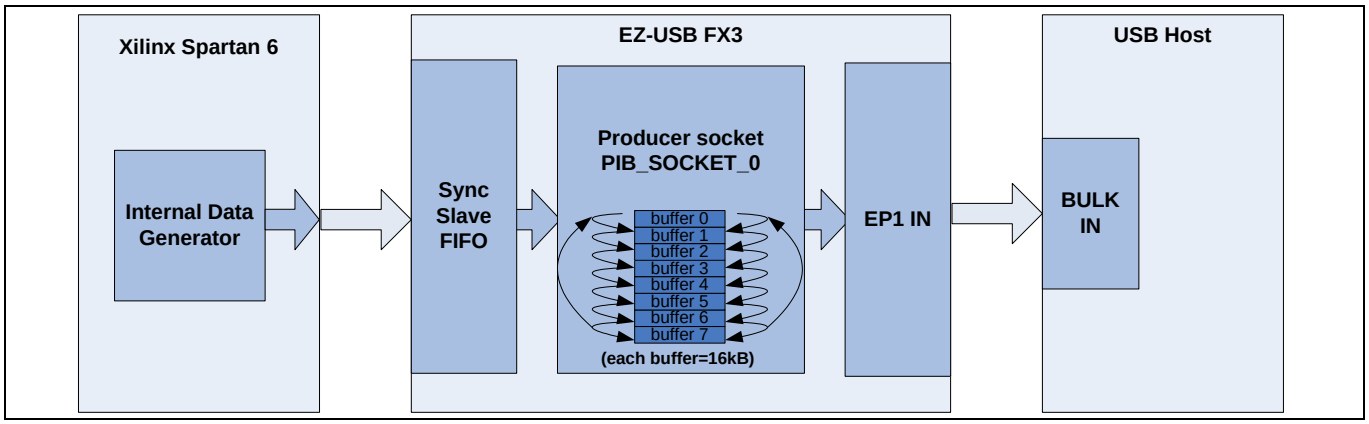

Figure 19 Stream IN transfer setup – Buffer count and size optimized for performance

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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

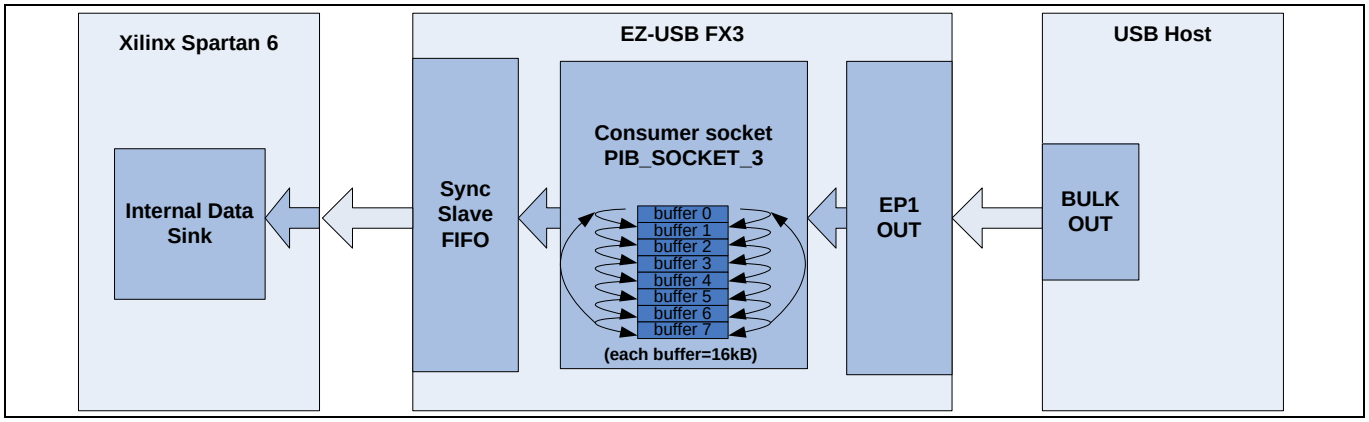

Figure 20 Stream OUT transfer setup – Buffer count and size optimized for performance

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)
```

11.4 FPGA implementation details

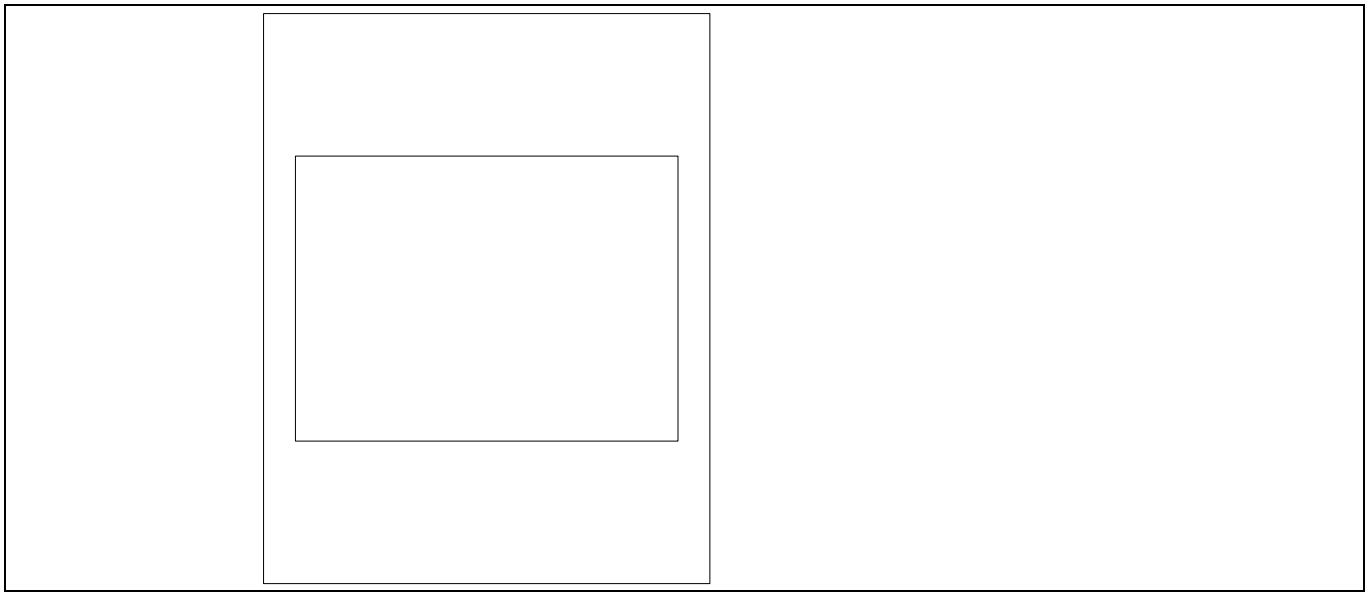

Figure 21 Spartan 6 (XC6SLX16) FPGA from Xilinx implementation using SP601 evaluation kit

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

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.

11.4.1 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.

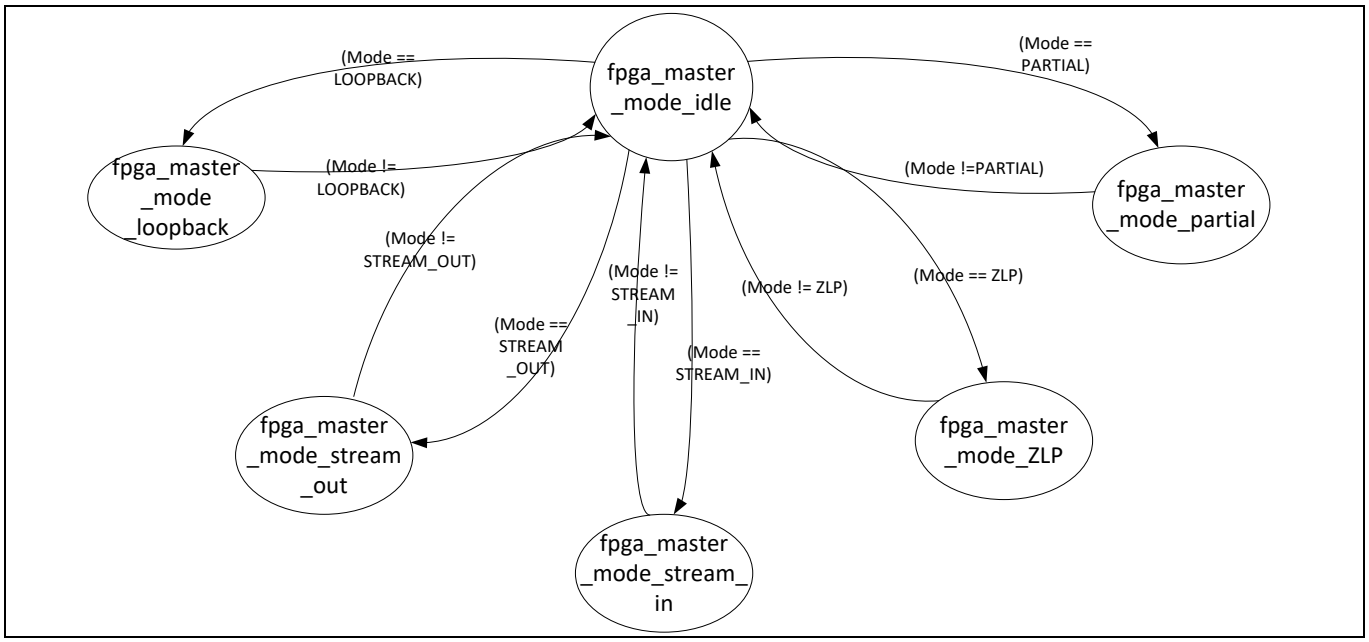

Figure 22 FPGA state machine for mode selection

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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

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.

11.4.2 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.

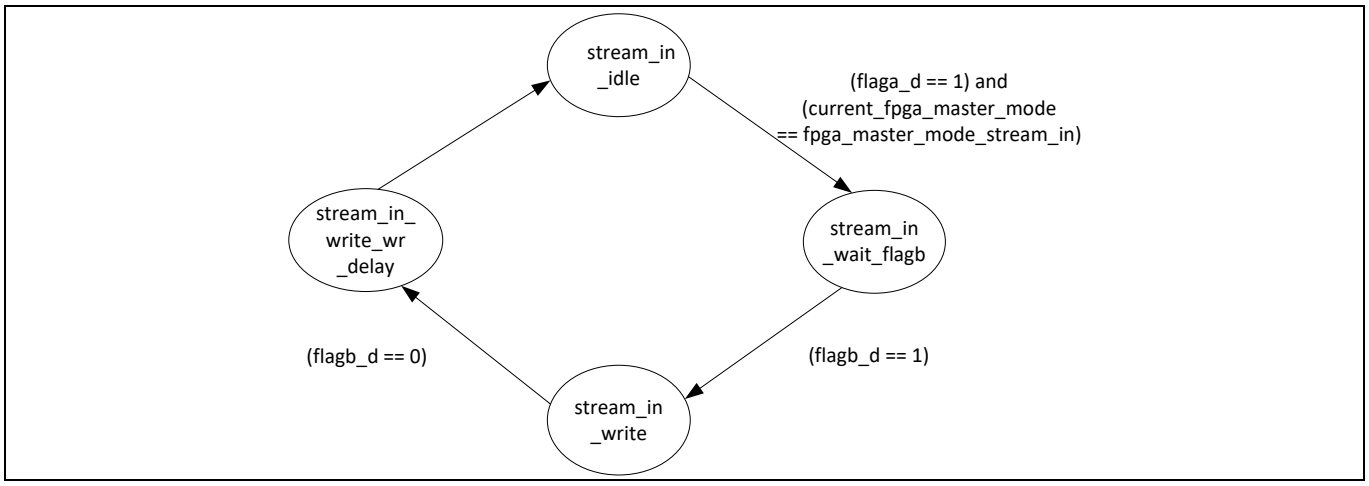

Figure 23 FPGA state machine for stream IN

State stream_in_idle:

This state initializes all the registers and signals used in the state machine. The slave FIFO control line status 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

11.4.3 State stream_in_write_wr_delay

Whenever flagb_d = 0, the state machine will enter this state. The slave FIFO control line status 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 **General formulae for using partial flags** section, 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.

11.4.4 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.

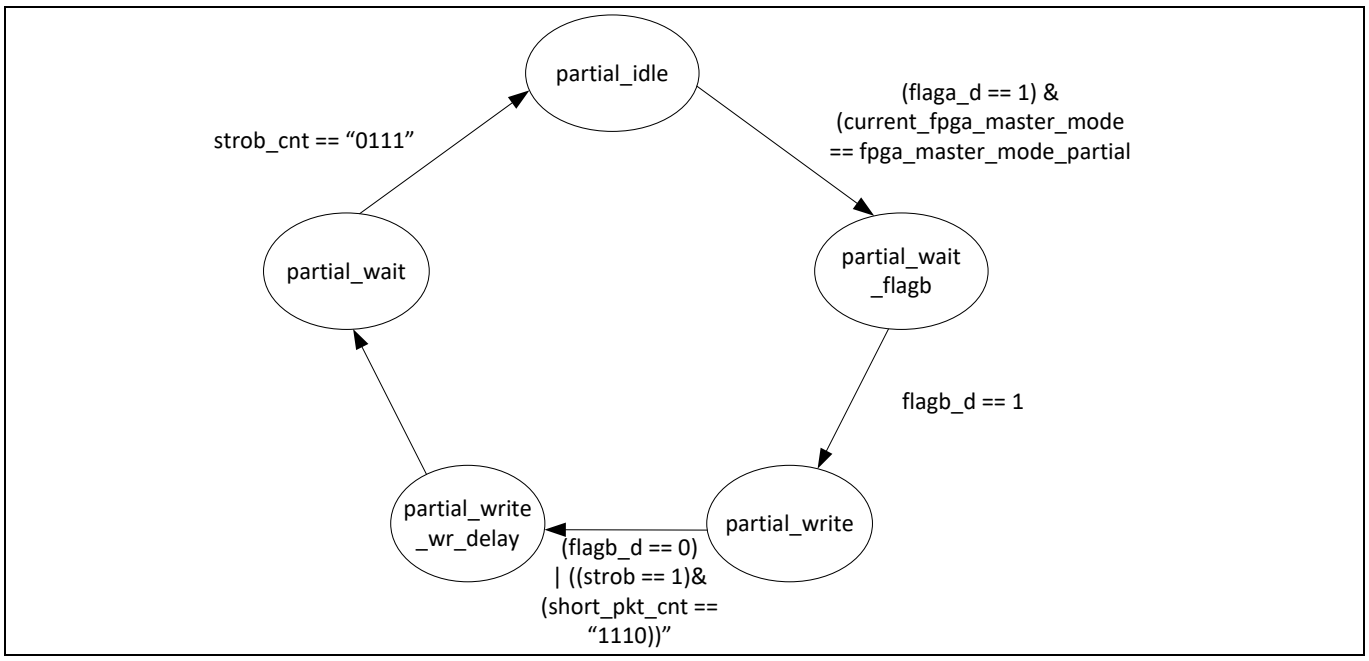

Figure 24 State machine for short packet transfer

State partial_idle:

This state initializes all the registers and signals used in the state machine. The slave FIFO control line status 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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

According to formula (1) in the General formulae for using partial flags section, 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.

11.4.5 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.

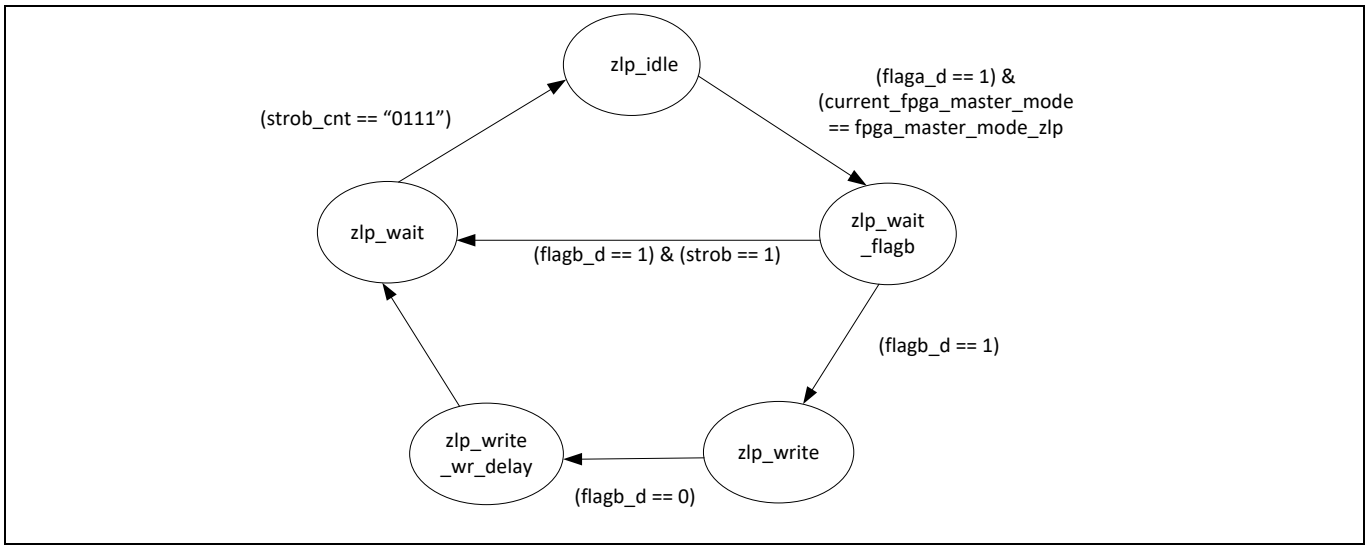

Figure 25 State machine for ZLP transfer

State zlp_idle:

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

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

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 **General formulae for using partial flags** section, 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.

11.4.6 State machine implemented in Verilog RTL for stream OUT example

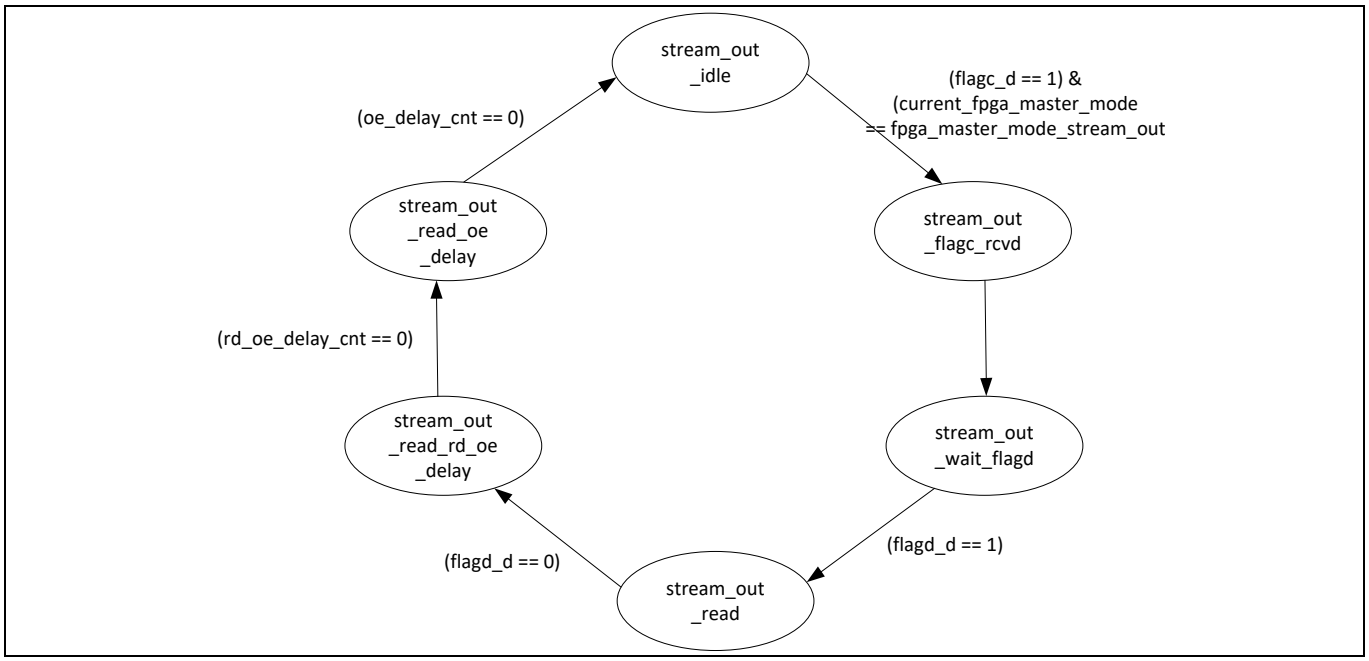

Figure 26 State machine for stream OUT transfer

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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

State stream_out_read:

Whenever 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 stream_out_read_rd_oe_delay:

Whenever flagd_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 **General formulae for using partial flags** section, 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.

11.4.7 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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

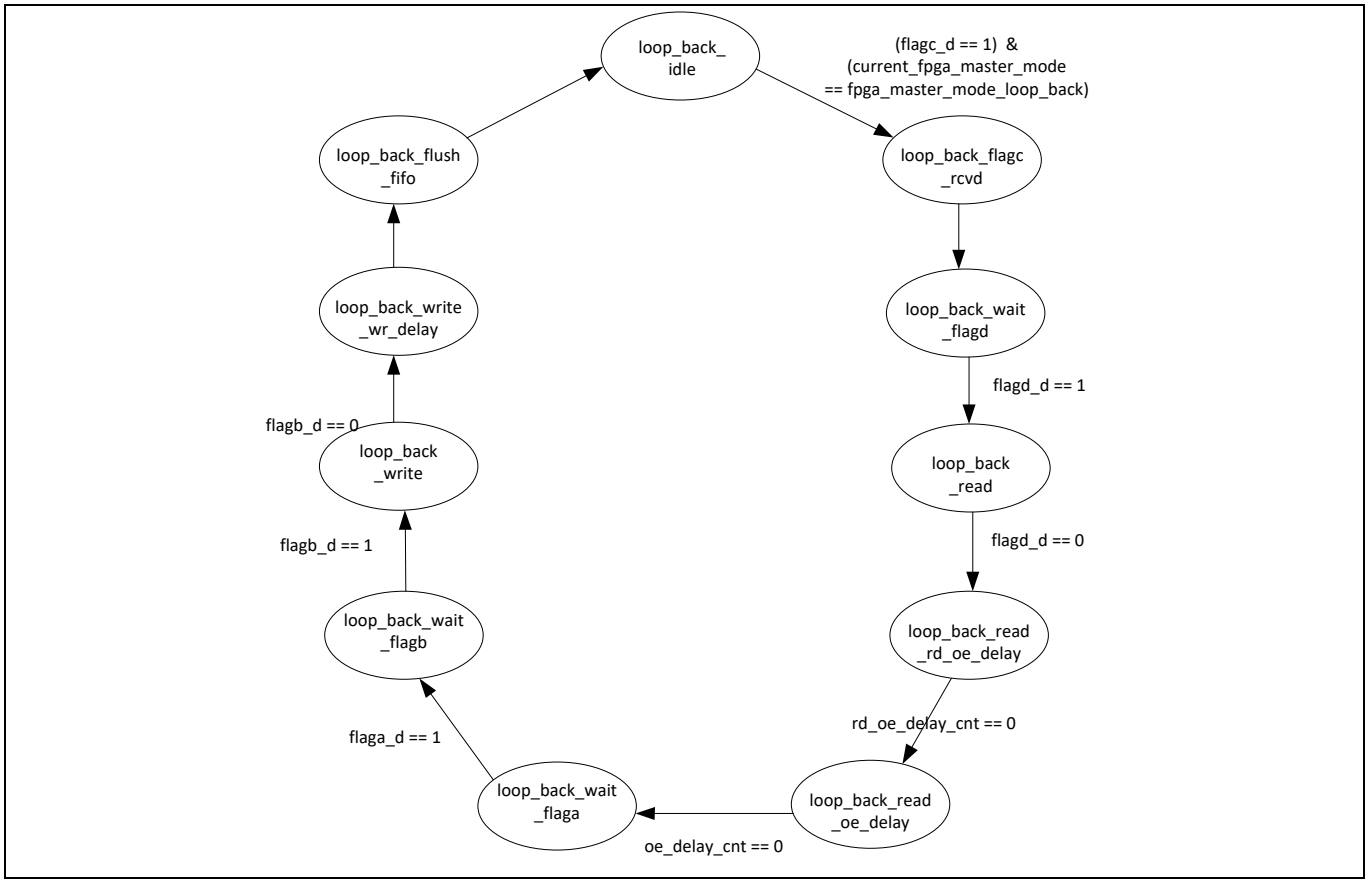

Figure 27 State machine for loopback transfer

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 flagd_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

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

According to formula (2b) in the **General formulae for using partial flags** section, 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 **General formulae for using partial flags** section, 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

11.5 Project operation

11.5.1 Steps to test loopback transfer

1. Make sure that jumper J5 is open if you are using the SuperSpeed Explorer Kit. Use **Table 7** to set up the jumper and switches if you are using the FX3 development kit (CYUSB3KIT-001). Connect the FX3 DVK board with the SP601 DVK board from Xilinx using the FMC connector and power on the FX3 DVK board before powering on the SP601 DVK from Xilinx.
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.

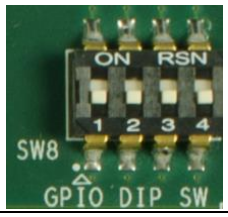

Figure 28 SW8 on SP601 board from Xilinx – All OFF before FPGA and FX3 configuration

3. 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.
4. Program the Spartan 6 FPGA from Xilinx with the file, *slavefifo2b_fpga_top.bit*. The FPGA can be programmed with any standard programmer such as the iMPACT application available with the **ISE Design Suite 14.7** from Xilinx. The FX3 DVK should be programmed before programming the FPGA board from Xilinx. The FX3 DVK will not enumerate on the host PC if the FPGA has already been programmed, because it drives the PMODE lines.

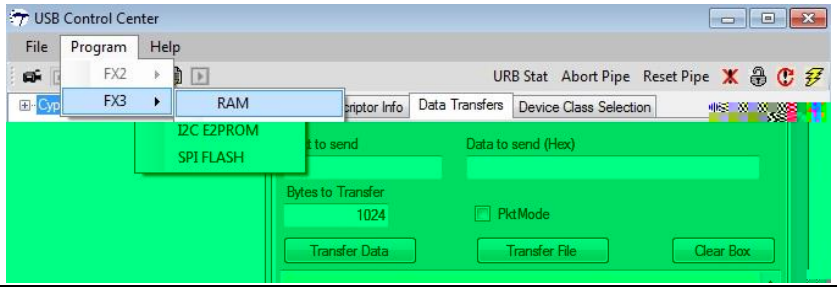

Figure 29 Programming FX3 firmware using Control center

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

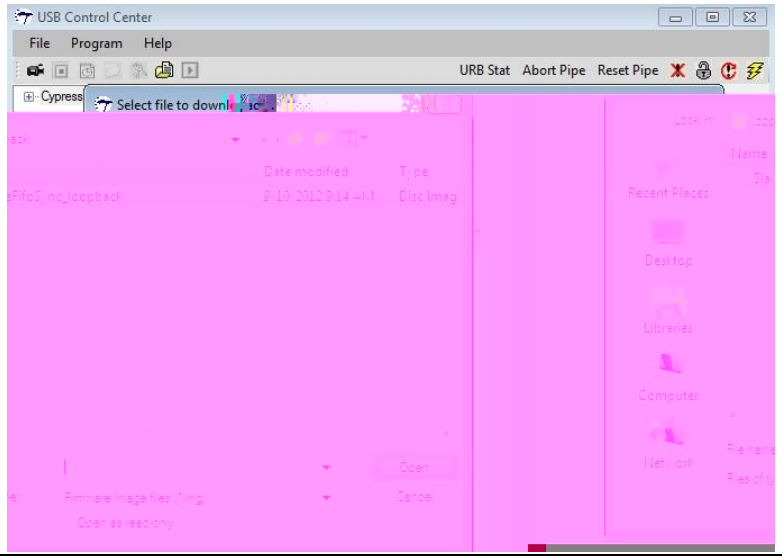

Figure 30 Programming FX3 firmware for loopback testing using Control center

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 6**.

5. Position 1 and 3 on SW8 on the SP601 board from Xilinx must be turned to the ON position to perform loopback transfers.

Table 6 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
FF	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 / partial packets (Stream Bulk OUT packets from the host)
OFF	ON	OFF	ON	Loopback transfer mode
OFF	X	X	X	Invalid

6. 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.

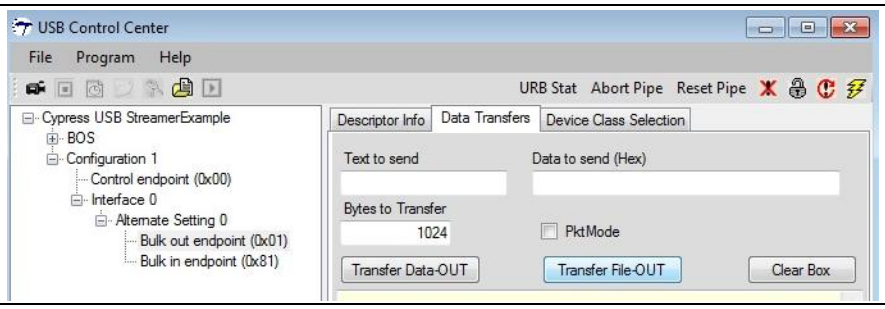

Figure 31 Initiate bulk OUT transfer using transfer file-OUT

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

7. 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.

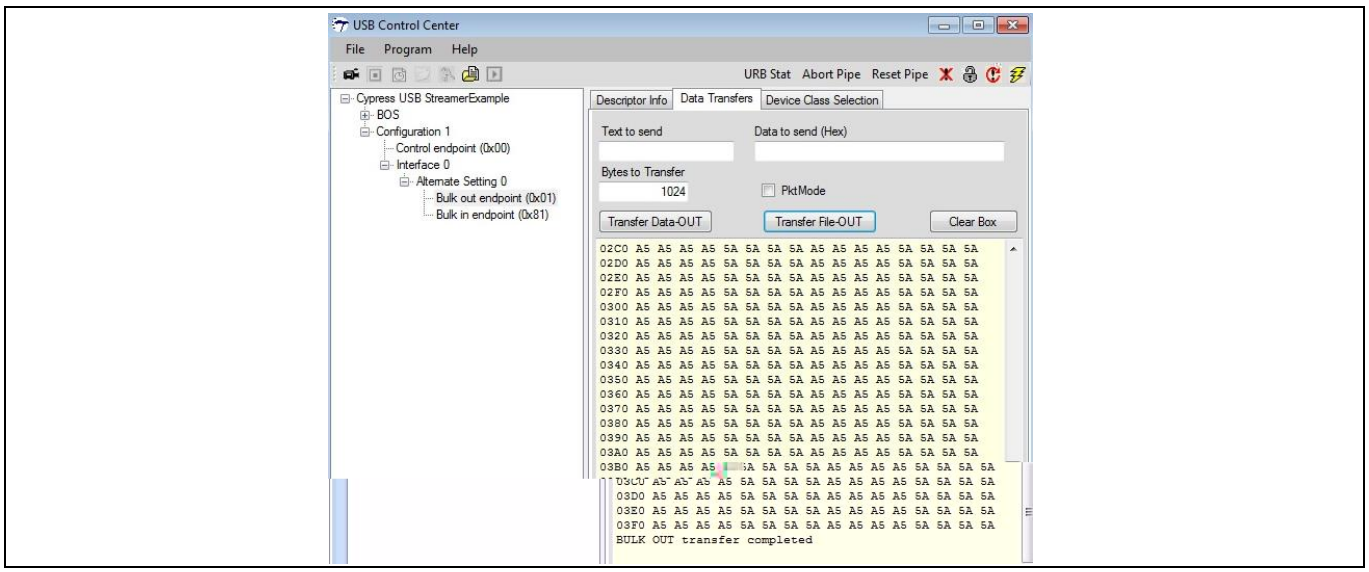

Figure 32 Data pattern transferred by selecting TEST.txt file for transfer file-OUT

8. The FPGA is already in a state where it is waiting for FLAGC to equal 1. As soon as the data is available in the buffer of PIB_SOCKET_3, the FPGA will read it. The FPGA will then loop back the same data and write it to FX3's PIB_SOCKET_0.
9. 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.

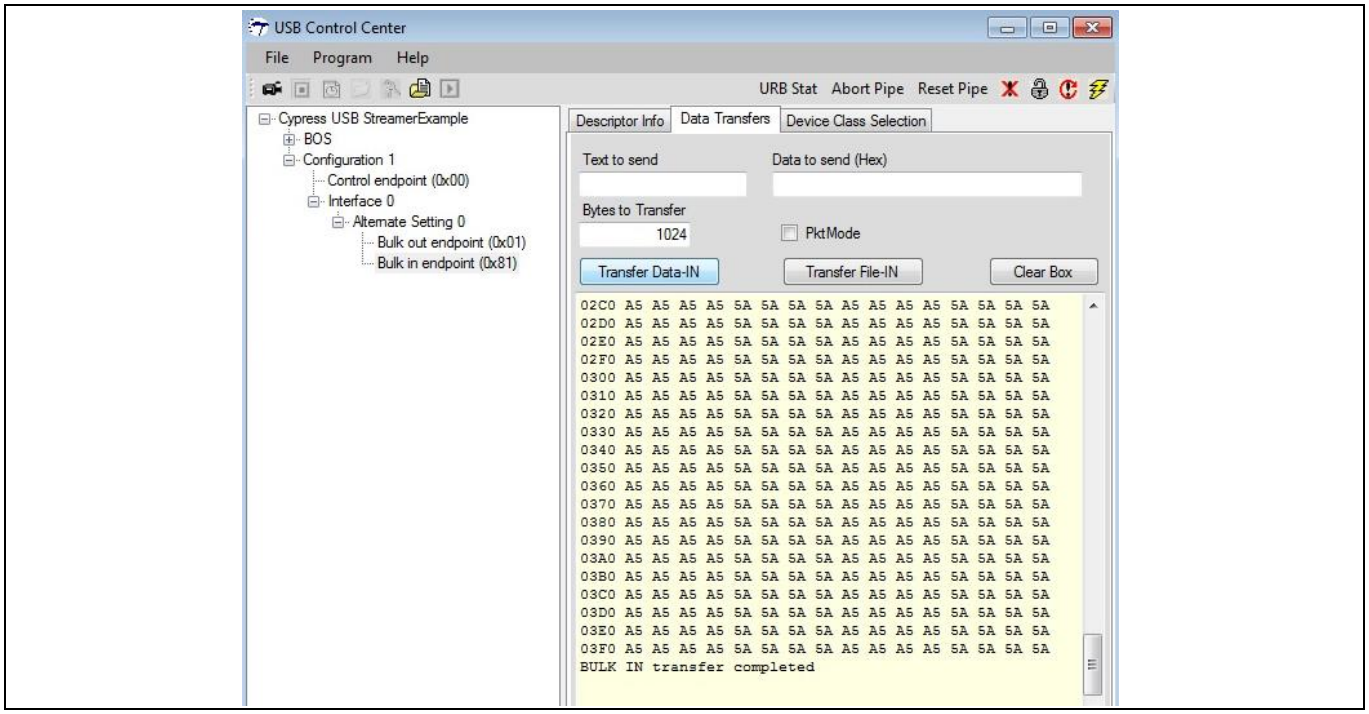

Figure 33 Complete loopback test by initiating a BULK IN transfer using transfer data-IN

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

11.5.2 Steps to test streaming transfers

Note: Always use the C++ Streamer utility provided with the FX3 SDK in the following path: <FX3_SDK_installation_path>\EZ-USB™ FX3 SDK\1.3\application\cpp\streamer\x86

Release 1.3 in this path is the FX3 SDK version number. It can be higher for future releases of the FX3 SDK.

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 6**.
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. Program the Spartan 6 FPGA from Xilinx with the file, *slavefifo2b_fpga_top.bit*. Ignore this step if the FPGA is already programmed as mentioned in step 1.
5. Set the switch SW8 on the SP601 board appropriately for the required transfer, as shown in **Table 6**.
6. 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 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.

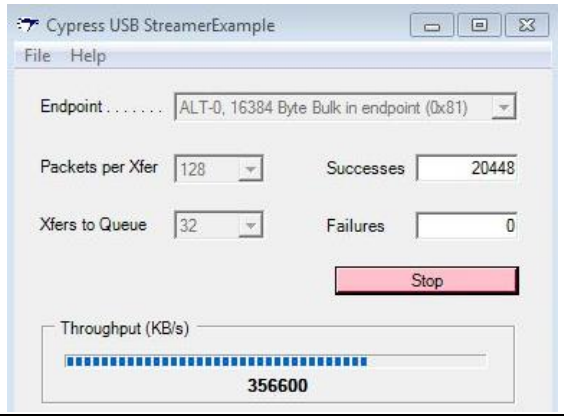

Figure 34 Streaming IN performance displayed in the streamer utility

7. 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 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.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

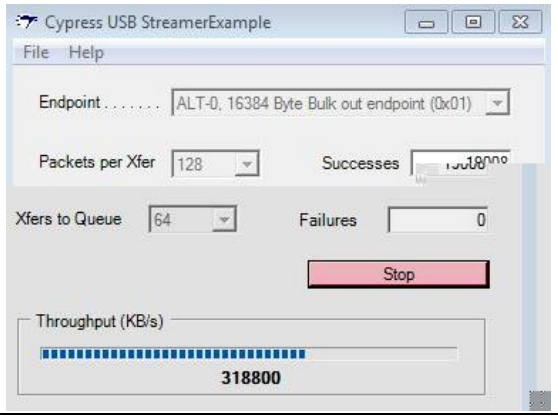

Figure 35 Streaming OUT performance displayed in the streamer utility

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* file, 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.

11.5.3 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 6**.
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. Program the Spartan 6 FPGA from Xilinx with the file, *slavefifo2b_fpga_top.bit*. Ignore this step if the FPGA is already programmed as mentioned in Step 1.
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 it is waiting for this condition, by monitoring FLAGA. As soon as flag equals 1, the FPGA starts writing to FX3.
5. 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*.
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 or ZLP will be received.

Note: If the ZLP transfer is done immediately after the short packet transfer, without resetting the FX3, the first transfer will be a short packet transfer. This holds true for a short packet transfer immediately after a ZLP transfer. This is because the FX3 buffer has not been flushed until the host has requested the data on the control center.

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

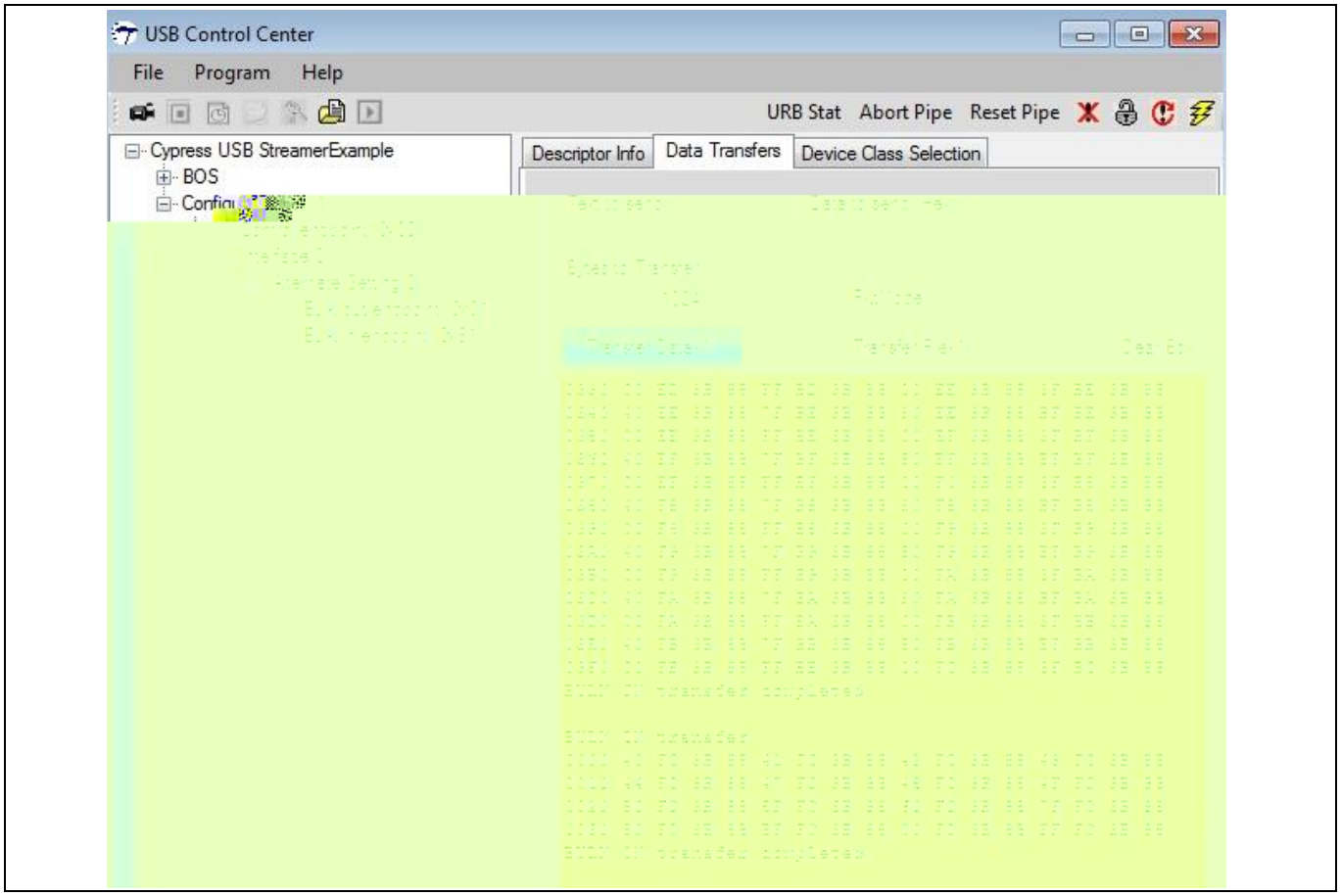

Figure 36 Full packet followed by short packet received by consecutive transfer data-IN operations

Design example 1: Interfacing an FPGA from Xilinx to FX3's synchronous slave FIFO interface

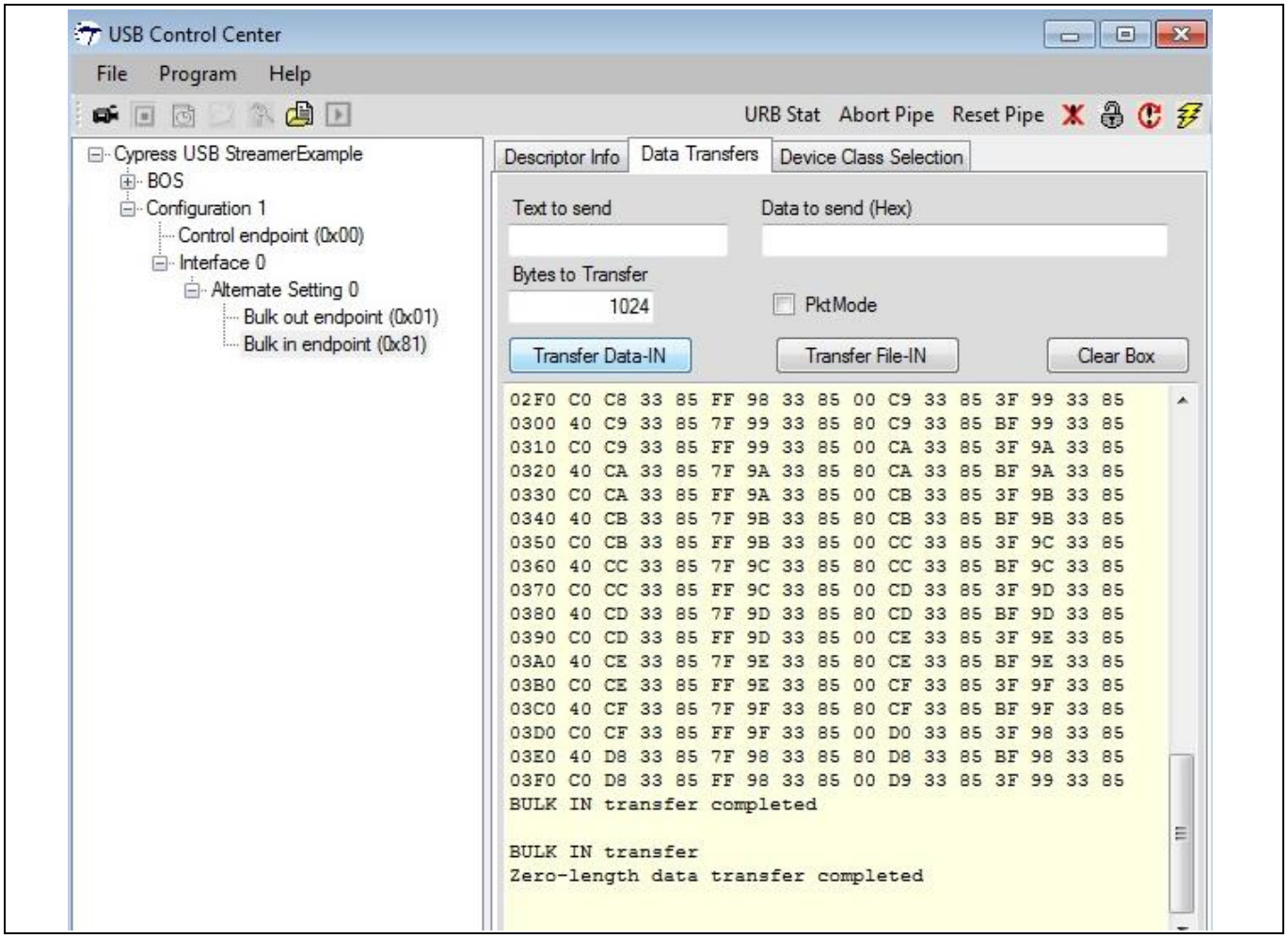

Figure 37 Full Packet followed by zero-length packet received by consecutive transfer data-IN operations

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

12 Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

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

12.1 Hardware setup

The project provided in this example can be executed on a hardware setup consisting of an **FX3 Development Kit (CYUSB3KIT-001) or SuperSpeed Explorer Kit (CYUSB3KIT-003)** interconnected with a Cyclone III FPGA Starter Board. The FX3 board and the board from Altera are connected using a Samtec to HSMC interconnect board. The **CYUSB3ACC-003 interconnect board** mates with the Samtec connector on the FX3 development kit (CYUSB3KIT-001) and the HSMC connector on the board from Altera.

The **CYUSB3ACC-006 interconnect board** mates with the headers on the SuperSpeed Explorer Kit (CYUSB3KIT-003) and the HSMC connector on the board from Altera.

Figure 38 shows the hardware setup using SuperSpeed Explorer Kit. Refer to **Appendix B: Hardware setup using FX3 DVK (CYUSB3KIT-001)**. Other than the hardware setup, the following steps are common irrespective of the FX3 board that you are using.

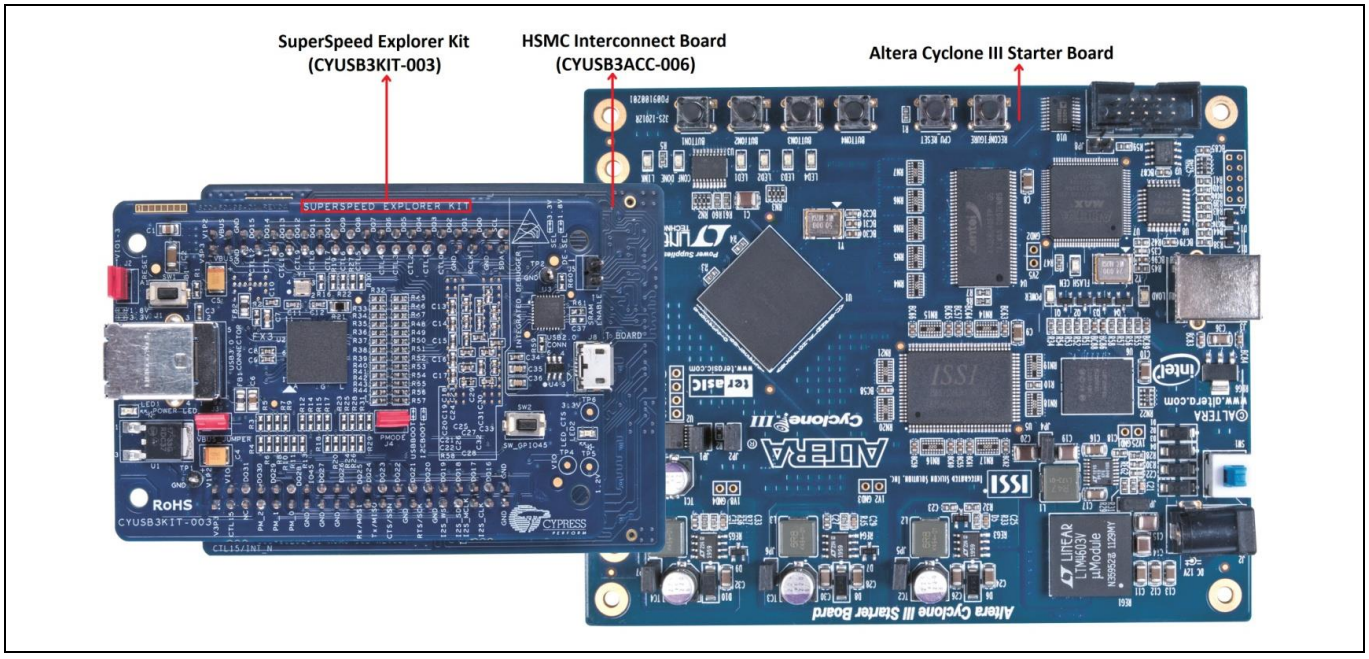

Figure 38 SuperSpeed explorer kit connected to Cyclone III Starter Board from Altera using HSMC interconnect board

Note: Make sure that jumper J5 on the SuperSpeed explorer kit is open in all applications where FPGA is interfaced to FX3 over slave FIFO interface.

12.2 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**.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

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

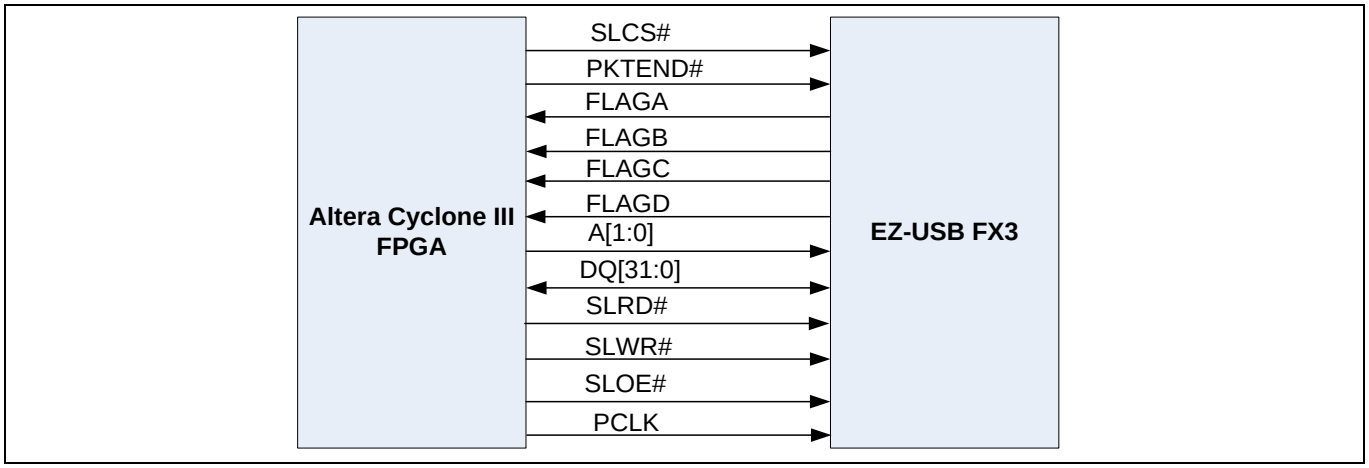

Figure 39 Interconnect diagram between FPGA and FX3

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.

12.3 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 VID/PID, 0x04B4/0x00F1. This enables the use of the 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:

  a) FLAGA: Full flag dedicated to thread0

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

b) FLAGB: Partial flag with watermark value 6, dedicated to thread0

c) FLAGC: Empty flag dedicated to thread3

d) FLAGD: Partial flag with watermark value 6, dedicated to thread3

Note: The GPIF II 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:
    - 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.
    - 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.

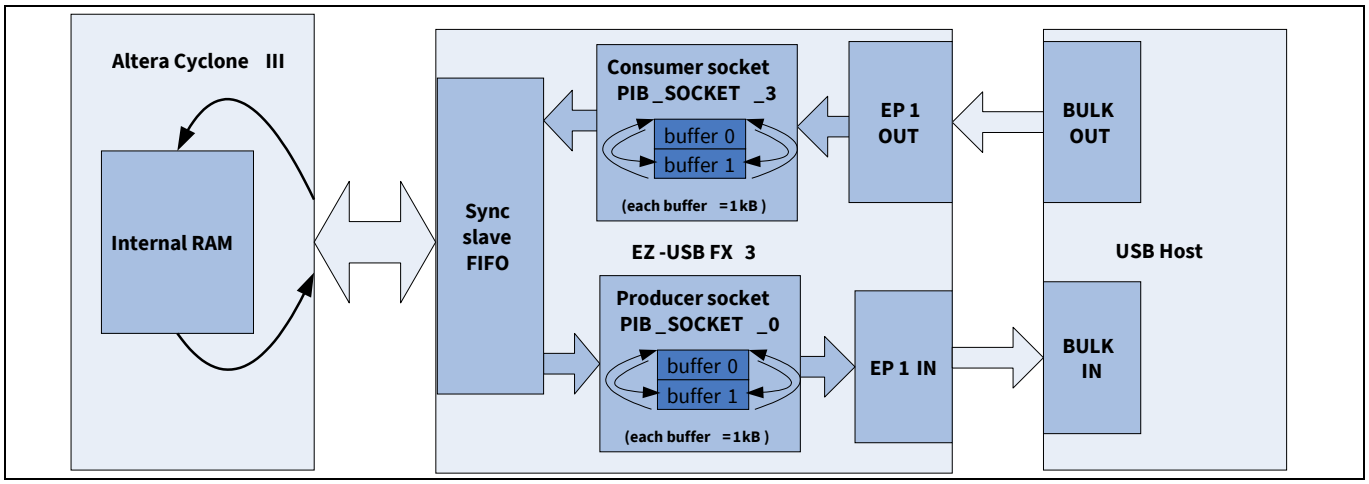

Figure 40 Loopback transfer setup

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

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

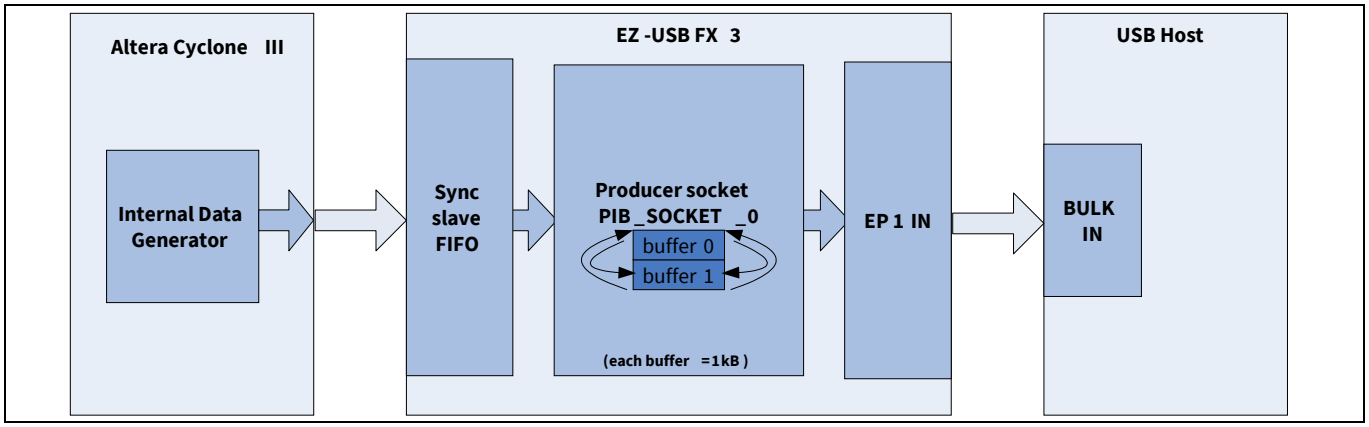

Figure 41 Short packet and ZLP transfer setup

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:
  - 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.

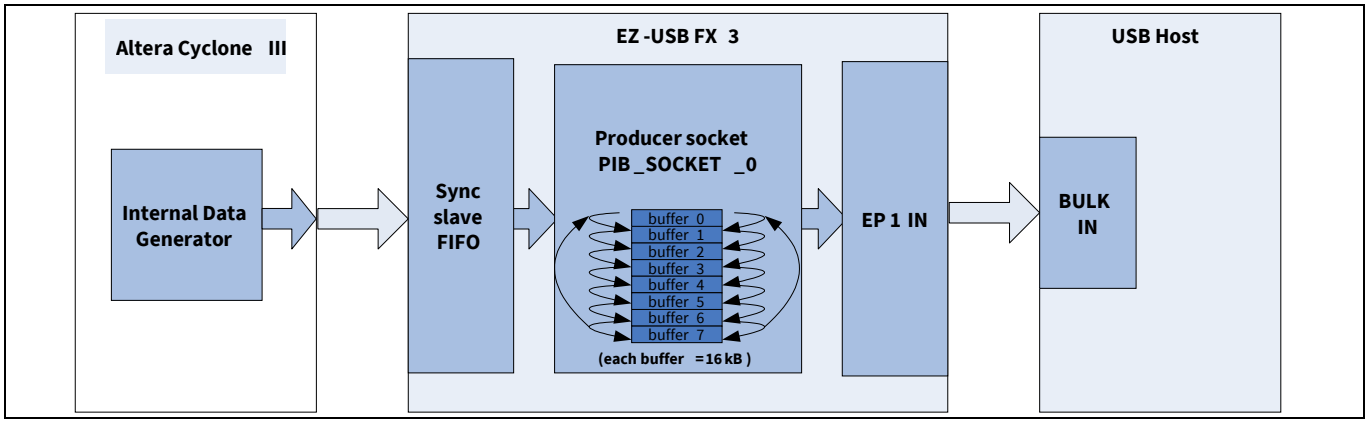

Figure 42 Stream IN transfer setup – Buffer count and size optimized for performance

  - 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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

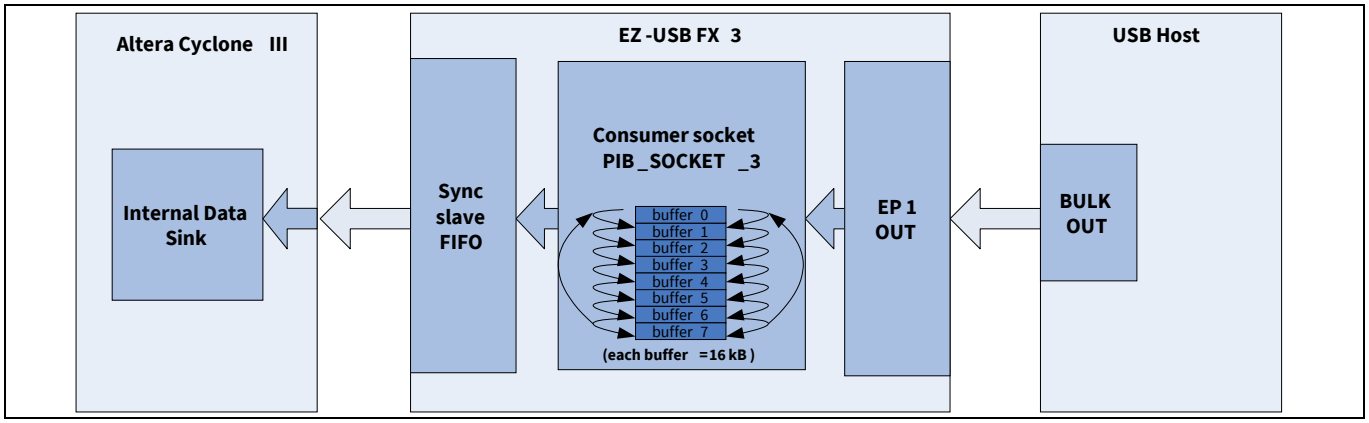

Figure 43 Stream OUT transfer setup – Buffer count and size optimized for performance

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)
```

12.4 FPGA implementation details

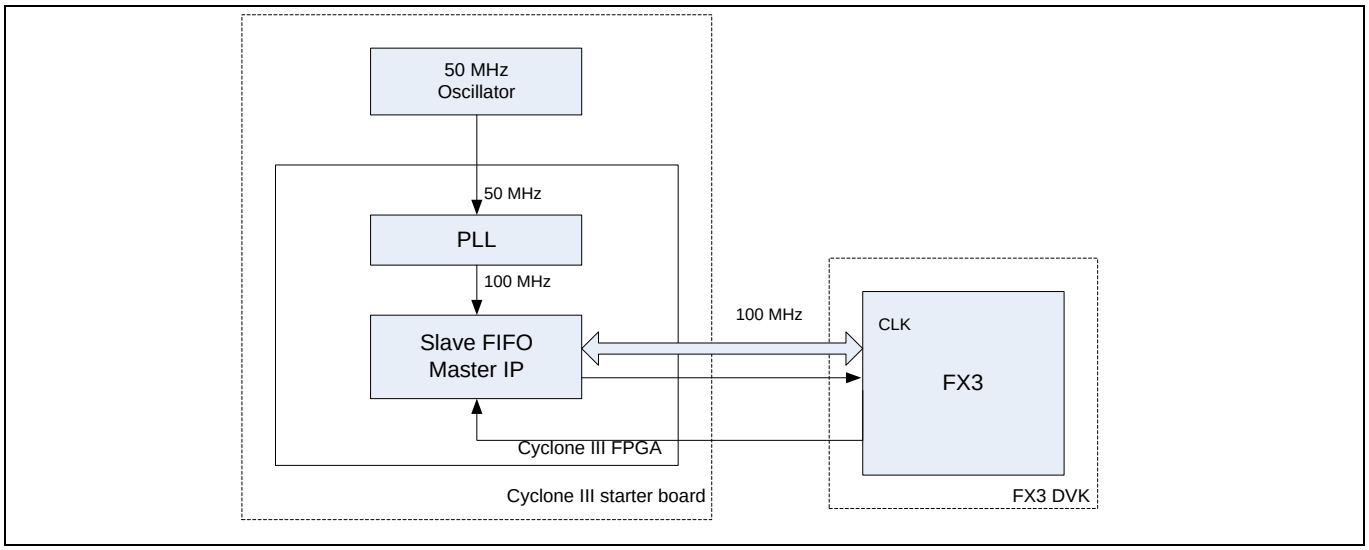

Figure 44 Cyclone III (EP3C25F324C6) FPGA from Altera implementation using SP601 evaluation kit

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

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.

12.4.1 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.

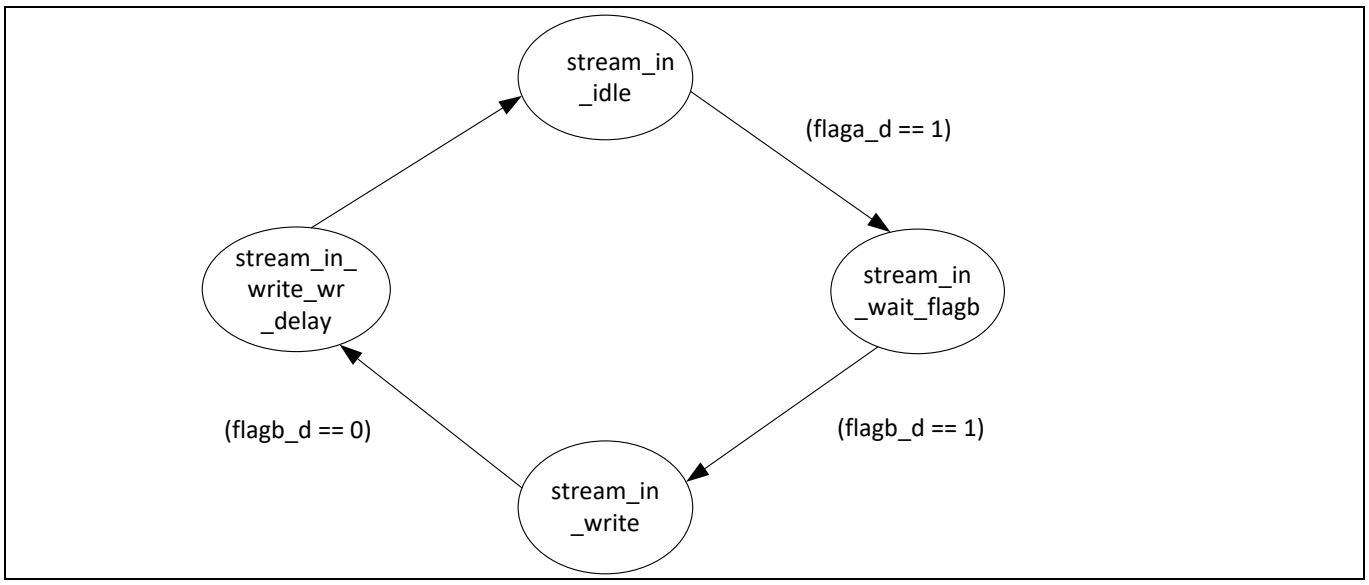

Figure 45 FPGA state machine for stream IN

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

12.4.2 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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

According to formula (1) in the **General formulae for using partial flags** section, 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.

12.4.3 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.

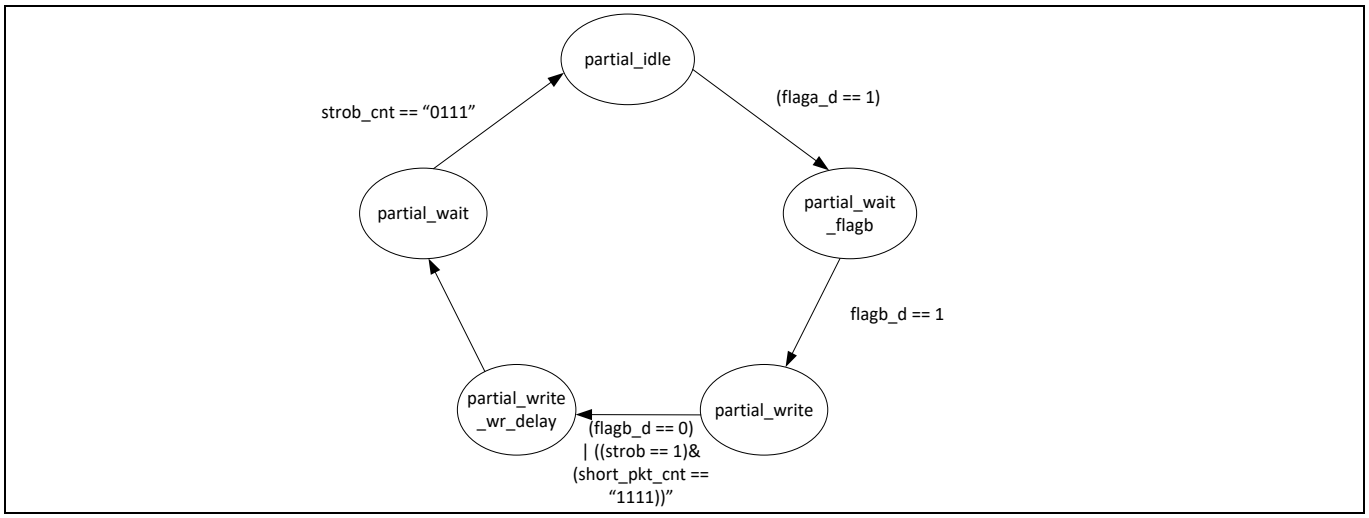

Figure 46 State Machine for Short Packet Transfer

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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

According to formula (1) in the **General formulae for using partial flags** section, 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.

12.4.4 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.

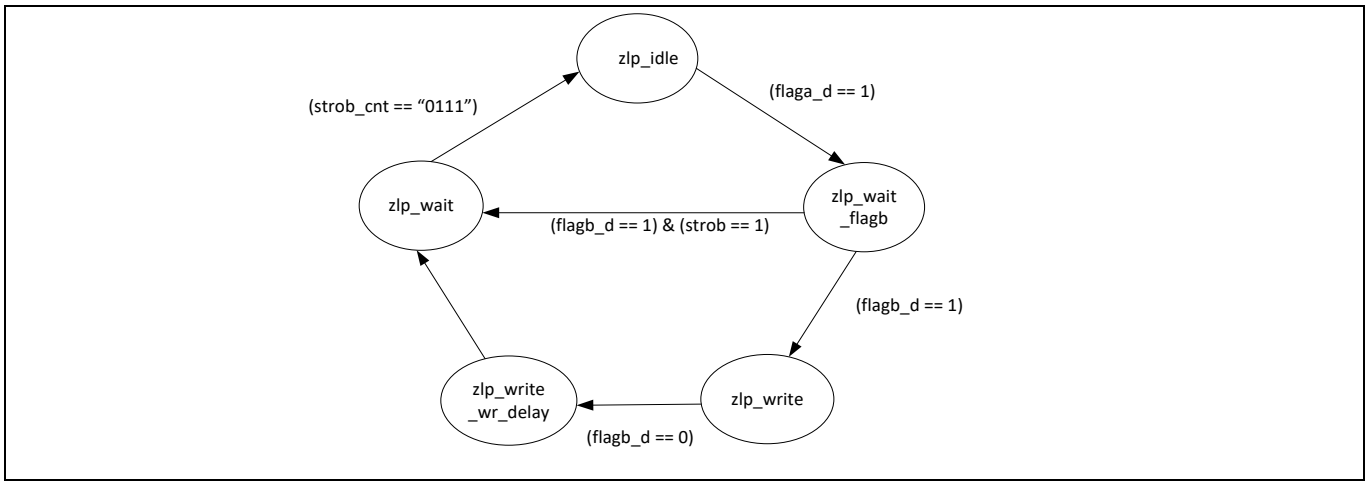

Figure 47 State machine for ZLP transfer

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:

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

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 **General formulae for using partial flags** section, 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.

12.4.5 State machine implemented in Verilog RTL for stream OUT example

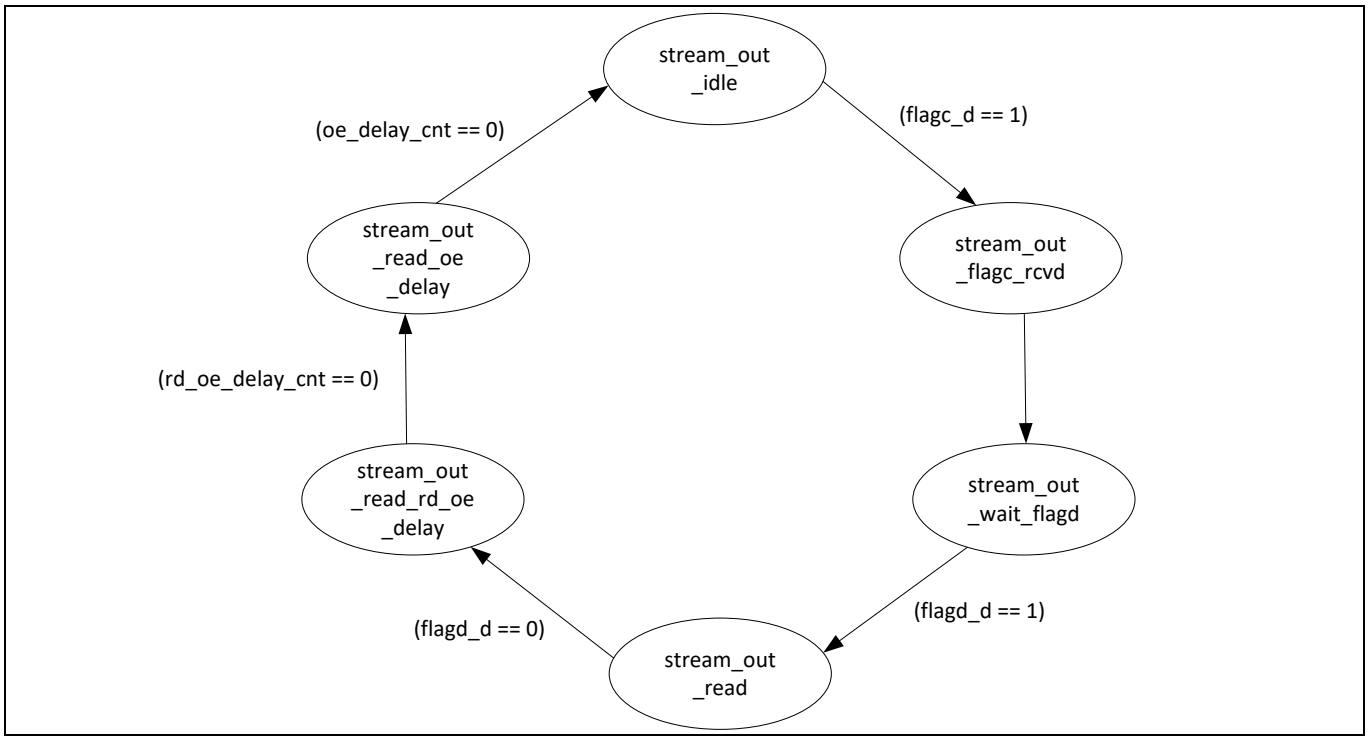

Figure 48 State machine for stream OUT transfer

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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

State stream_out_wait_flagd:

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

State stream_out_read:

Whenever 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 stream_out_read_rd_oe_delay:

Whenever flagd_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) the **General formulae for using partial flags** section, 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.

12.4.6 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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

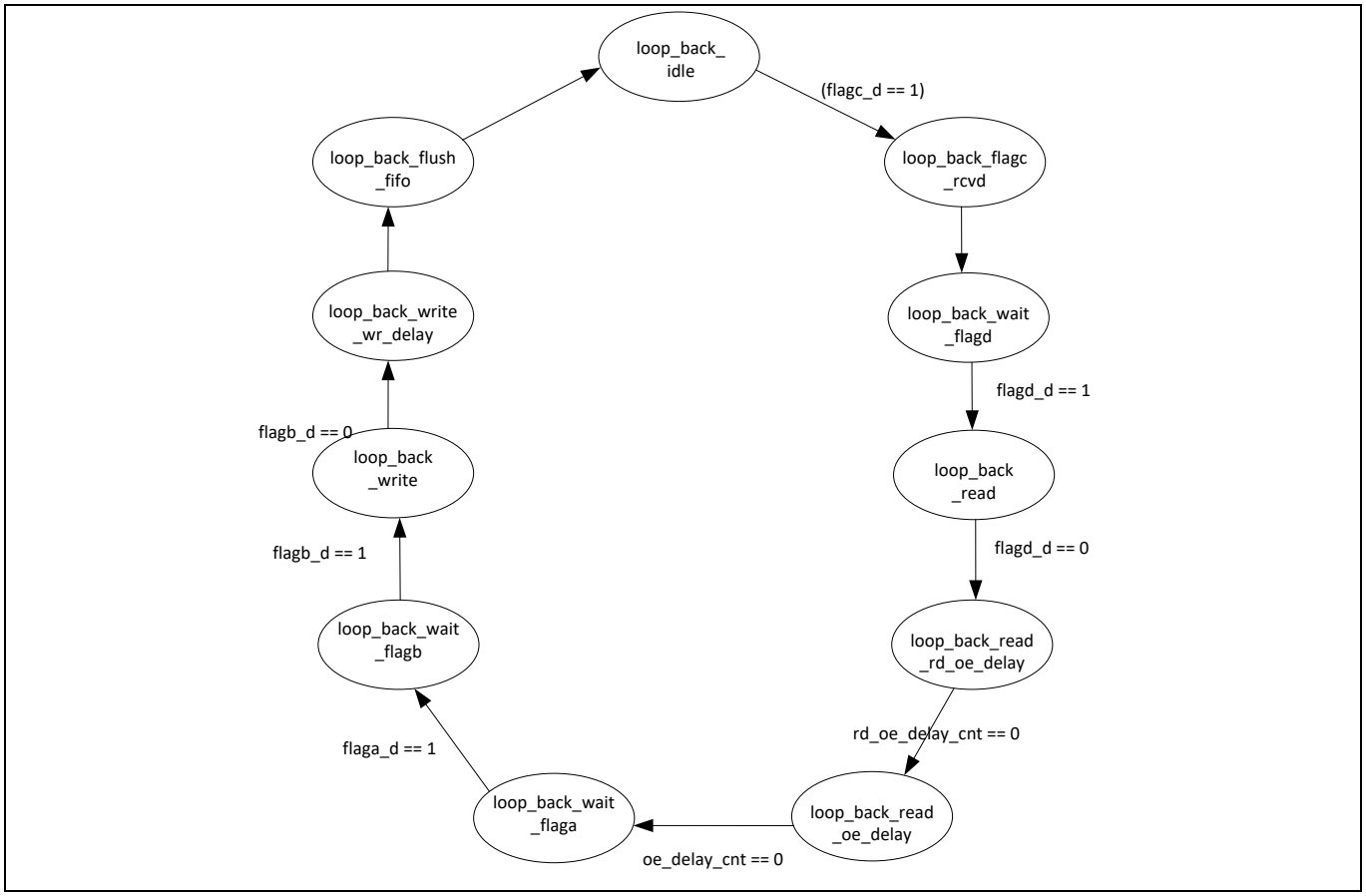

Figure 49 State machine for loopback transfer

State loop_back_idle:

This state initializes all the registers and signals used in the state machine. The slave FIFO control line status 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 flagd_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

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

According to formula (2b) in the **General formulae for using partial flags** section, 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 **General formulae for using partial flags** section, 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

12.5 Project operation

12.5.1 Steps to test loopback transfer

1. Make sure that jumper J5 is open if you are using the SuperSpeed Explorer Kit. Use **Table 7** to set up the jumper and switches if you are using the FX3 development kit (CYUSB3KIT-001). Connect the FX3 DVK board with the Cyclone III FPGA starter board from Altera using the HSMC connector and power on the FX3 DVK board before powering the Cyclone III FPGA starter board from Altera.
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 Cyclone III FPGA from Altera. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).

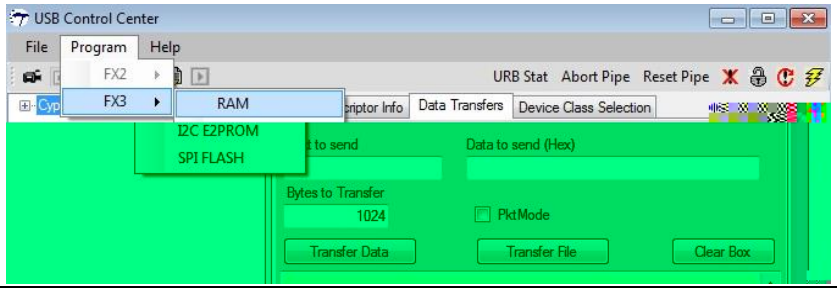

Figure 50 Programming FX3 firmware using Control center

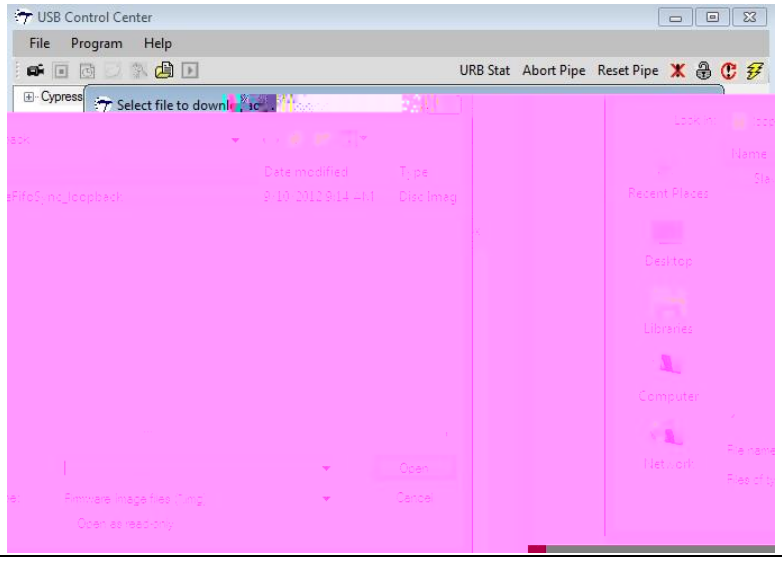

Figure 51 Programming FX3 firmware for loopback testing using Control center

3. Program the Cyclone III FPGA from Altera with the *slaveFIFO2b_loopback.sof* file. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with the **Quartus II Web Edition 12.1** 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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

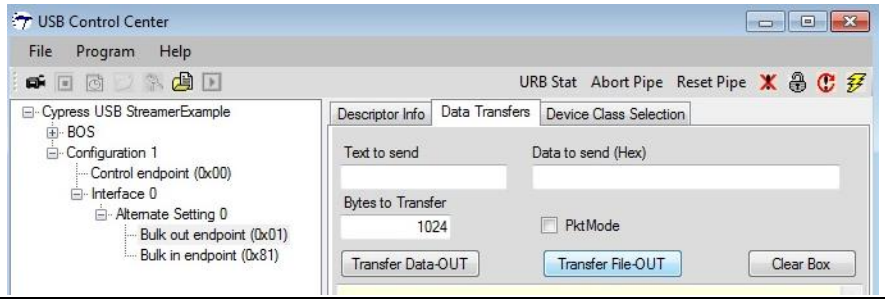

Figure 52 Initiate Bulk OUT transfer using transfer file-OUT

5. 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 the *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.

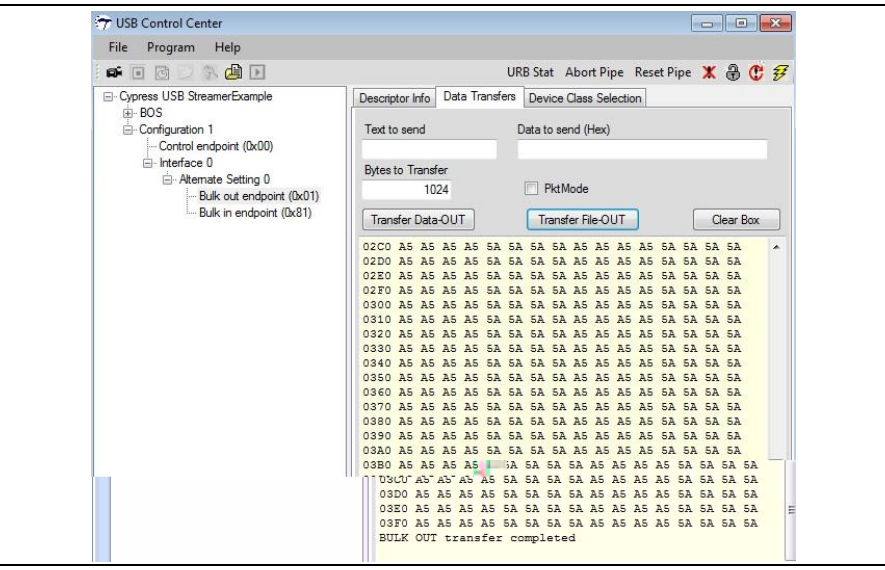

Figure 53 Data pattern transferred by selecting the TEST.txt file for transfer file-OUT

6. The FPGA is already in a state where it is waiting for FLAGC to equal 1. As soon as the data is available in the buffer of PIB_SOCKET_3, the FPGA will read it. The FPGA will then loop back the same data and write it to FX3's PIB_SOCKET_0.
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.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

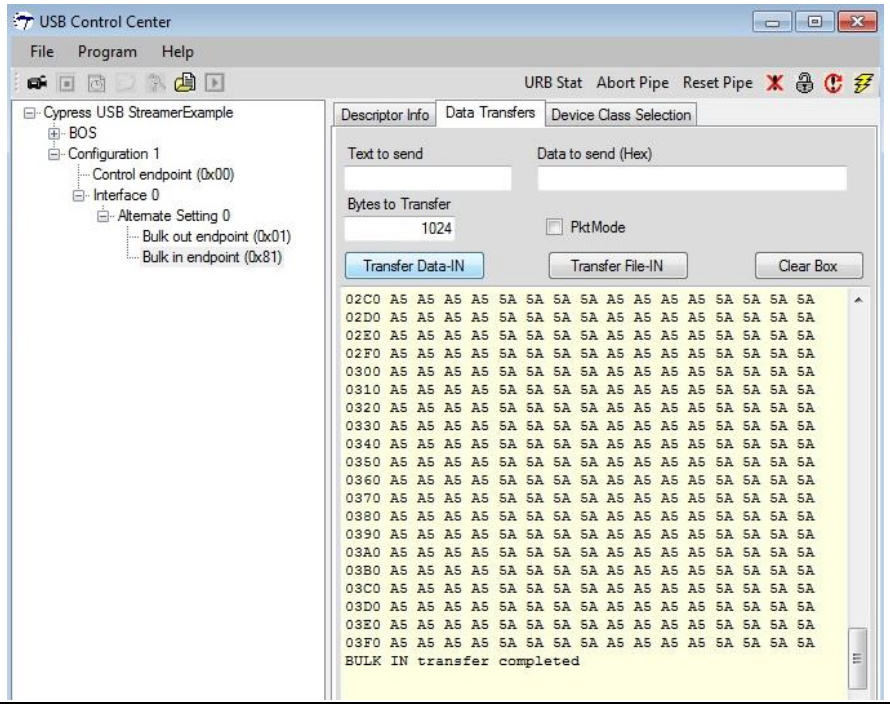

Figure 54 Complete loopback test by initiating a BULK IN transfer using transfer data-IN

12.5.2 Steps to test streaming transfers

Note: Always use the C++ Streamer utility provided with the FX3 SDK in the following path: <FX3_SDK_installation_path>\EZ-USB™ FX3 SDK\1.3\application\cpp\streamer\x86

Release 1.3 in the above path is the FX3 SDK version number. It can be higher for future releases of the FX3 SDK.

1. Connect the FX3 DVK board with the Cyclone III FPGA starter board from Altera using the HSMC connector and power on both the FX3 DVK board and the Cyclone III FPGA starter board from Altera.
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 Cyclone III FPGA from Altera. 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 Cyclone III FPGA from Altera with the *slaveFIFO2b_streamIN.sof* file for stream IN transfers or with the *slaveFIFO2b_streamOUT.sof* file 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 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 Z77 Express chipset from Intel.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

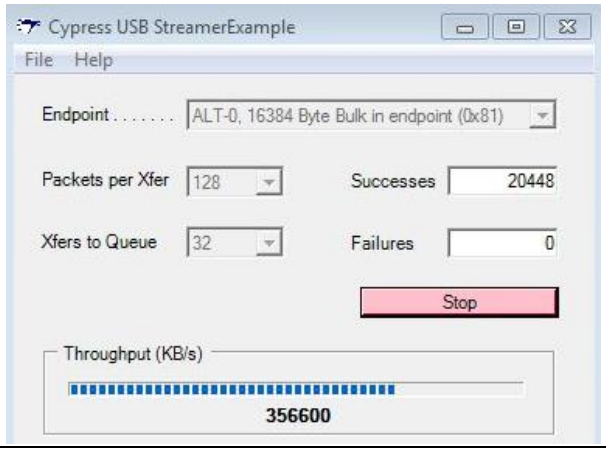

Figure 55 Streaming IN performance in streamer utility

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 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.

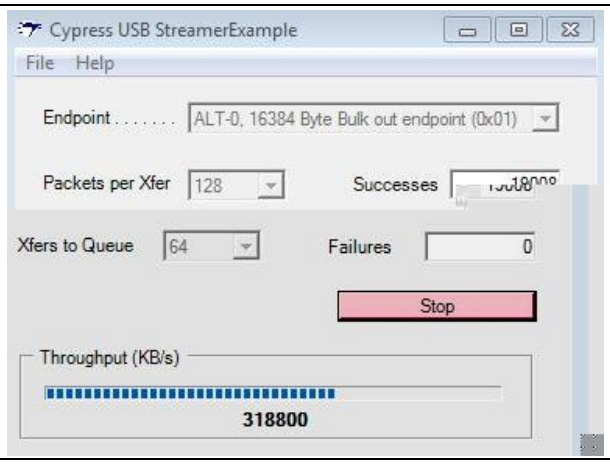

Figure 56 Streaming OUT performance in streamer utility

The *SF_streamIN.img* file 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 *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.

12.5.3 Steps to test short packet transfers

1. Connect the FX3 DVK board with the Cyclone III FPGA starter board from Altera using the HSMC connector and power on both the FX3 DVK board and the Cyclone III FPGA starter board from Altera.
2. For Stream IN or OUT, 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. FX3 should be programmed before programming the Cyclone III FPGA from Altera. After you download the firmware, the FX3 device will enumerate as a SuperSpeed device (if connected to a USB 3.0 port).

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

3. Program the Cyclone III FPGA from Altera with the *slaveFIFO2b_partial.sof* file. 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 the 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.

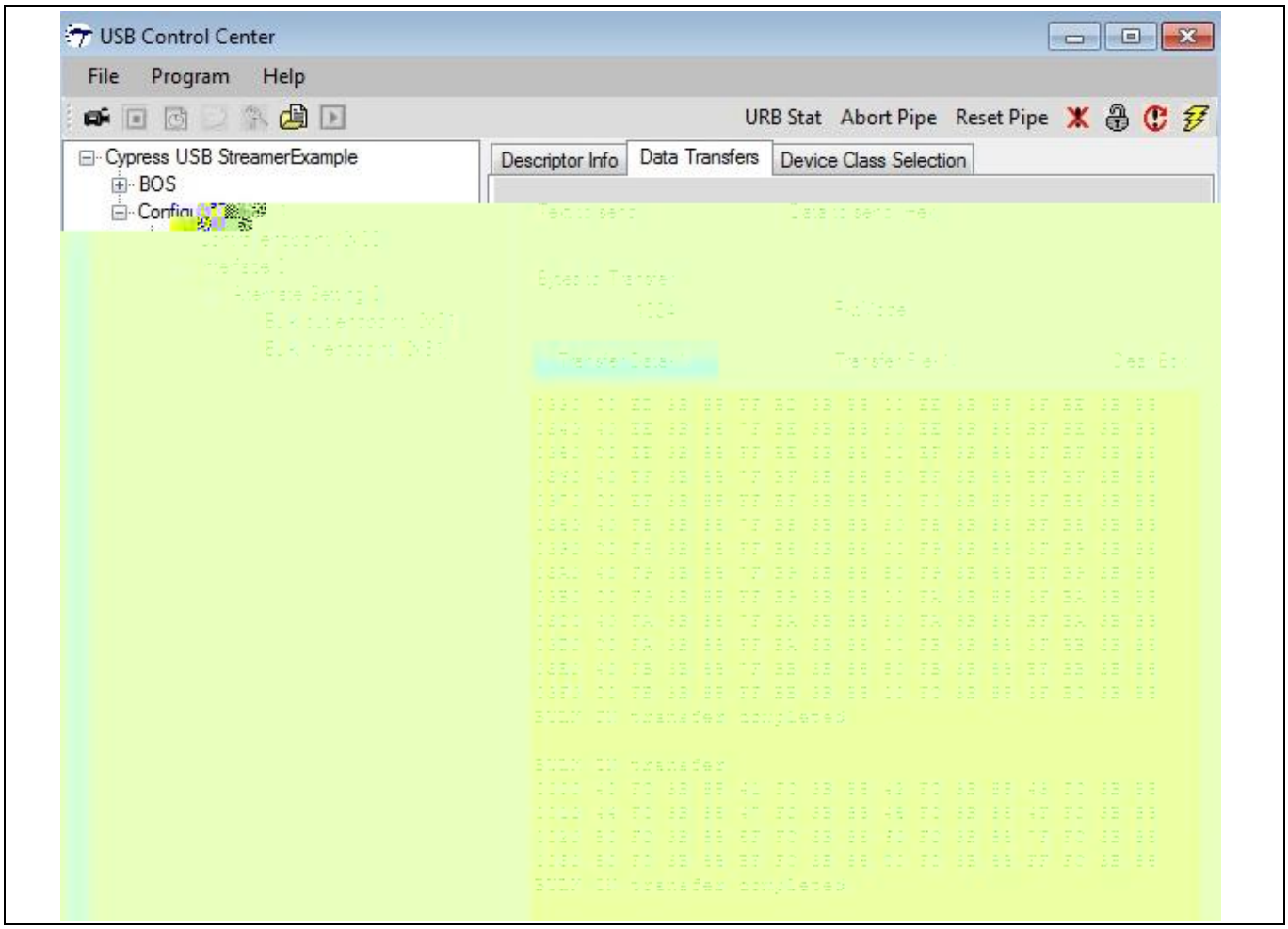

Figure 57 Full packet followed by short packet received by consecutive transfer data-IN operations

12.5.4 Steps to test ZLP transfers

1. Connect the FX3 DVK board with the Cyclone III FPGA starter board from Altera using the HSMC connector and power on both the FX3 DVK board and the Cyclone III FPGA starter board from Altera.
2. For Stream IN or OUT, 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. FX3 should be programmed before programming the Cyclone III FPGA from Altera. 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 Cyclone III FPGA from Altera with the *slaveFIFO2b_ZLP.sof* file. The FPGA can be programmed with any standard programmer such as the USB-Blaster application available with **Quartus** II software.

Design example 2: Interfacing an FPGA from Altera to FX3's synchronous slave FIFO interface

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 the 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.

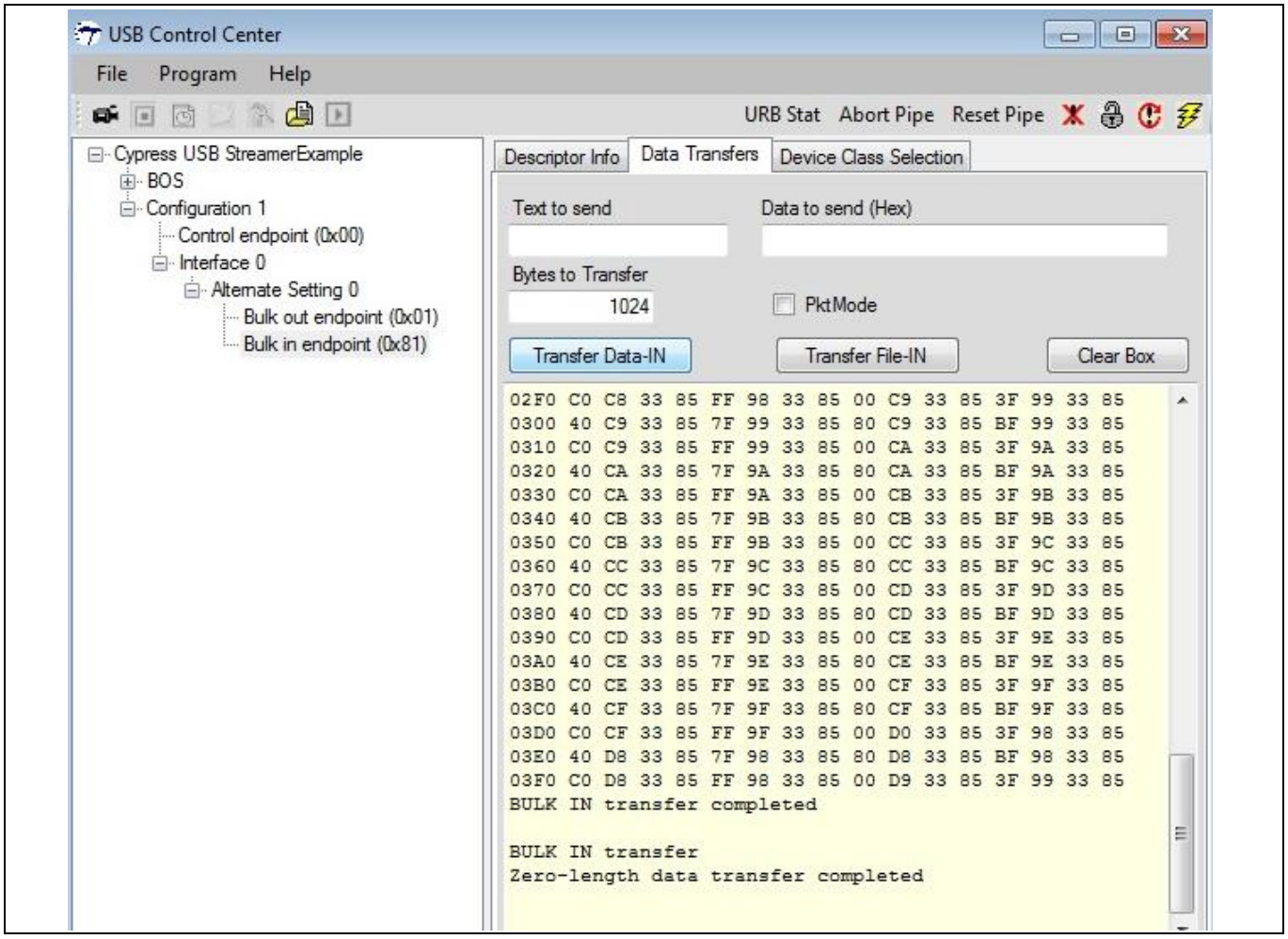

Figure 58 Full packet followed by zero-length packet received by consecutive transfer data-IN operations

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

13 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* FPGA source code from Xilinx in Verilog that supports all transfer types (stream IN, stream OUT, short packets, ZLP, and loopback). *Fpga_slavefifo2b_vhdl* FPGA source code from Xilinx in VHDL that supports all transfer types (stream IN, stream OUT, short packets, ZLP, and loopback).
	Fx3_slaveFIFO2b_altera	FPGA source code from Altera 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. This image is generated after making the following macro definition changes in the firmware source code. STREAM_IN_OUT' – Disabled LOOPBACK_SHRT_ZLP - Enabled
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. This image is generated after making the following macro definition changes in the firmware source code. STREAM_IN_OUT' – Enabled LOOPBACK_SHRT_ZLP - Disabled
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. This image is generated after making the following macro definition changes in the firmware source code. STREAM_IN_OUT' – Enabled LOOPBACK_SHRT_ZLP - Disabled
SF_shrt_ZLP.img		The FX3 firmware image that contains the slave FIFO implementation and sets up the DMA channels required for optimized performance when performing stream of short or Zero Length Packets (ZLP).

Associated project files

File/folder name	Description
	When you use this image, you will see continuous stream of a full packet followed by a Short or a Zero Length Packet. This image is generated after making the following macro definition changes in the firmware source code. STREAM_IN_OUT' – Disabled LOOPBACK_SHRT_ZLP - Enabled
TEST.txt	The file that contains the data pattern sent using the **Transfer File-OUT** button in the Control Center utility.

14 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 GPIF II designer tool may be used to configure flags. Two complete design examples are also presented.

15 Appendix A: Troubleshooting

Symptom 1

BULK IN and BULK OUT data transfers are failing.

```
BULK OUT transfer
BULK OUT transfer failed with Error Code:997
BULK IN transfer
BULK IN 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 FX3 SDK. Use this solution in your final firmware or contact **Cypress developer community** 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.

Appendix A: Troubleshooting

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.

Appendix A: Troubleshooting

```
/* 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 32-kHz 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;
    }
```

Symptom 5

DMA flags are not working as expected while reading/writing the data from FX3.

Debug steps

1. Ensure that the GPIF state machine GPIO setting (polarity/initial value) is correct in GPIF-II designer. Check **GPIF-Pin-Polarity** knowledge base article on how to interpret the polarity of GPIF-II pins based on the pin configuration chosen in the GPIF-II designer.
2. Probe the slave FIFO signals using HW logic analyzer close to FX3 pins.
3. Make sure that PCLK is provided to FX3 before checking the DMA flag status.
4. Enable UART prints and check whether FX3 is receiving data from the FPGA.
5. **Debugging-when-DMA-Flags-do-not-Work-as-Expected-while-Reading**. This KBA provides preliminary checks to debug if the read sequence mentioned in Section 5.1 is not working as expected.

Symptom 6

USB data transfer errors are seen when ZLP is followed by data packet within the same microframe.

Debug steps

See Errata 5. 'USB data transfer errors are seen when ZLP is followed by data packet within same microframe' in the **FX3 datasheet**.

Symptom 7

CyU3PGPIFload returns error 0x46:

Debug steps

This error may occur because of the following reasons:

1. Check the PMODE setting is correct. If this is not set correctly, the device could be identified as an unsupported device and the GPIF interface could be disabled. See **https://community.cypress.com/t5/USB-Superspeed-Peripherals/CyU3PGpifLoad-failed/td-p/264514**.

Appendix A: Troubleshooting

2. Check if the GPIF bus width is supported by the part number (see the **FX3 datasheet** – Ordering section).

Note: *Contact **Cypress developer community** if the issue in your application is still not solved.*

16 Appendix B: Hardware setup using FX3 DVK (CYUSB3KIT-001)

Figure 59 shows the hardware setup using FX3 development kit (CYUSB3KIT-001) and SP601 board from Xilinx.

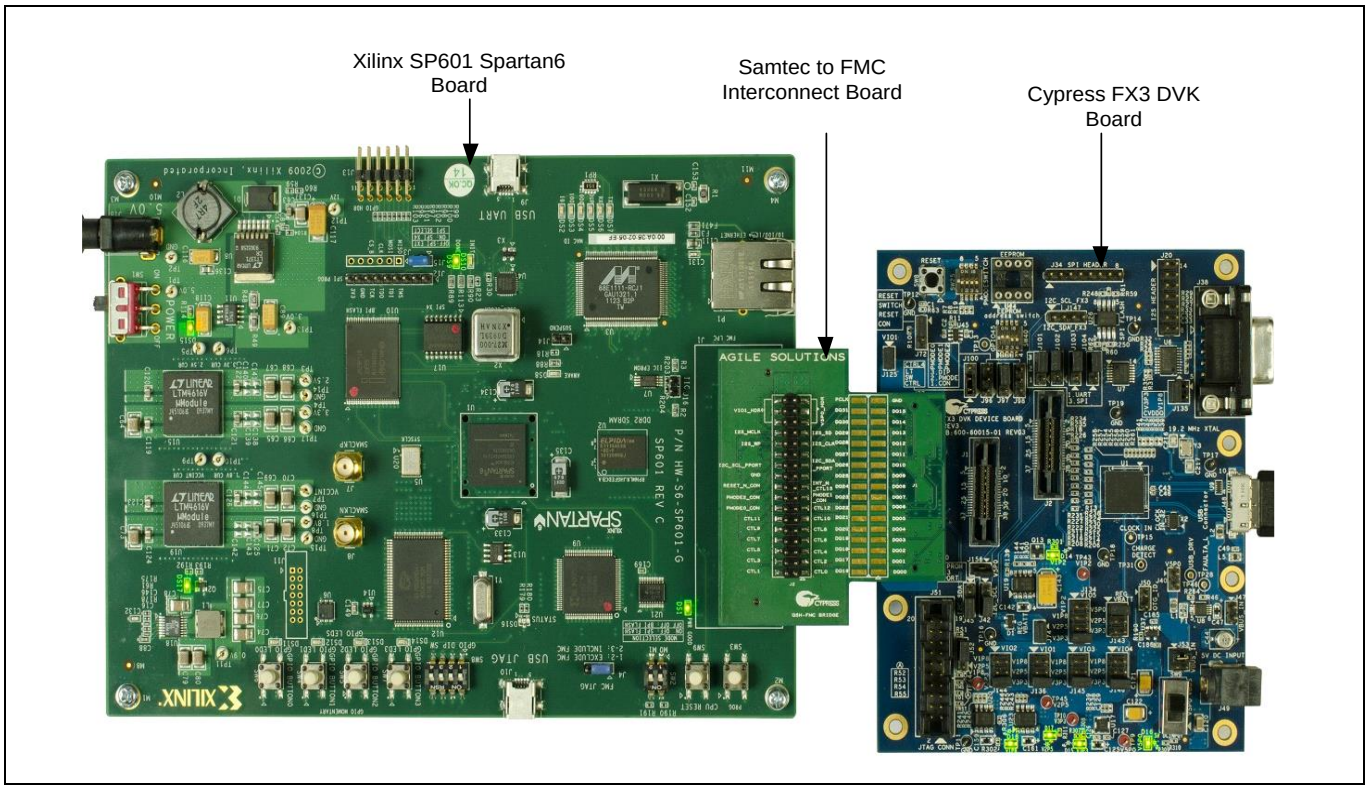

Figure 59 FX3 development kit connected to SP601 board from Xilinx using FMC interconnect board

Figure 60 shows the hardware setup using FX3 development kit (CYUSB3KIT-001) and Cyclone III board from Altera.

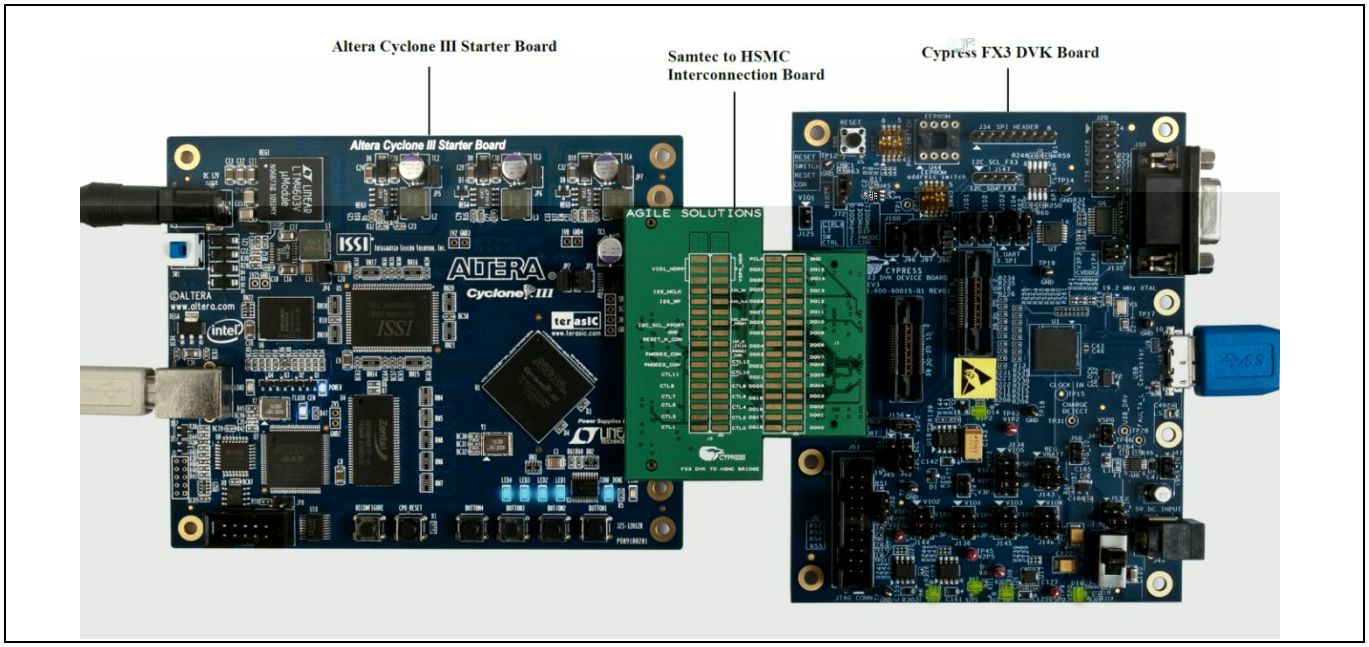

Figure 60 FX3 development kit connected to Cyclone III board from Altera using HSMC interconnect board

16.1 Jumper and switch settings

Table 7 lists the FX3 DVK board jumper and switch settings to run the demo. These settings are the same irrespective of the FPGA board connected to FX3 DVK.

Table 7 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(3.3 V)
3	J144	3 and 4	VIO2(3.3 V)
4	J145	3 and 4	VIO3(3.3 V)
5	J146	3 and 4	VIO4(3.3 V)
6	J134	4 and 5	VIO5(3.3 V)
7	J135	2 and 3	CVDDQ(3.3 V)
8	J143	3 and 4	VBATT(3.3 V)
9	J101	1 and 2	GPIO[53] = UART_RTS
10	J102	1 and 2	GPIO[54] = UART_CTS
11	J103	1 and 2	GPIO[56] = UART_TX
12	J104	1 and 2	GPIO[57] = 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	1–3 or 2–4	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 open.

17 Appendix C

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

17.1 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 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 61**.

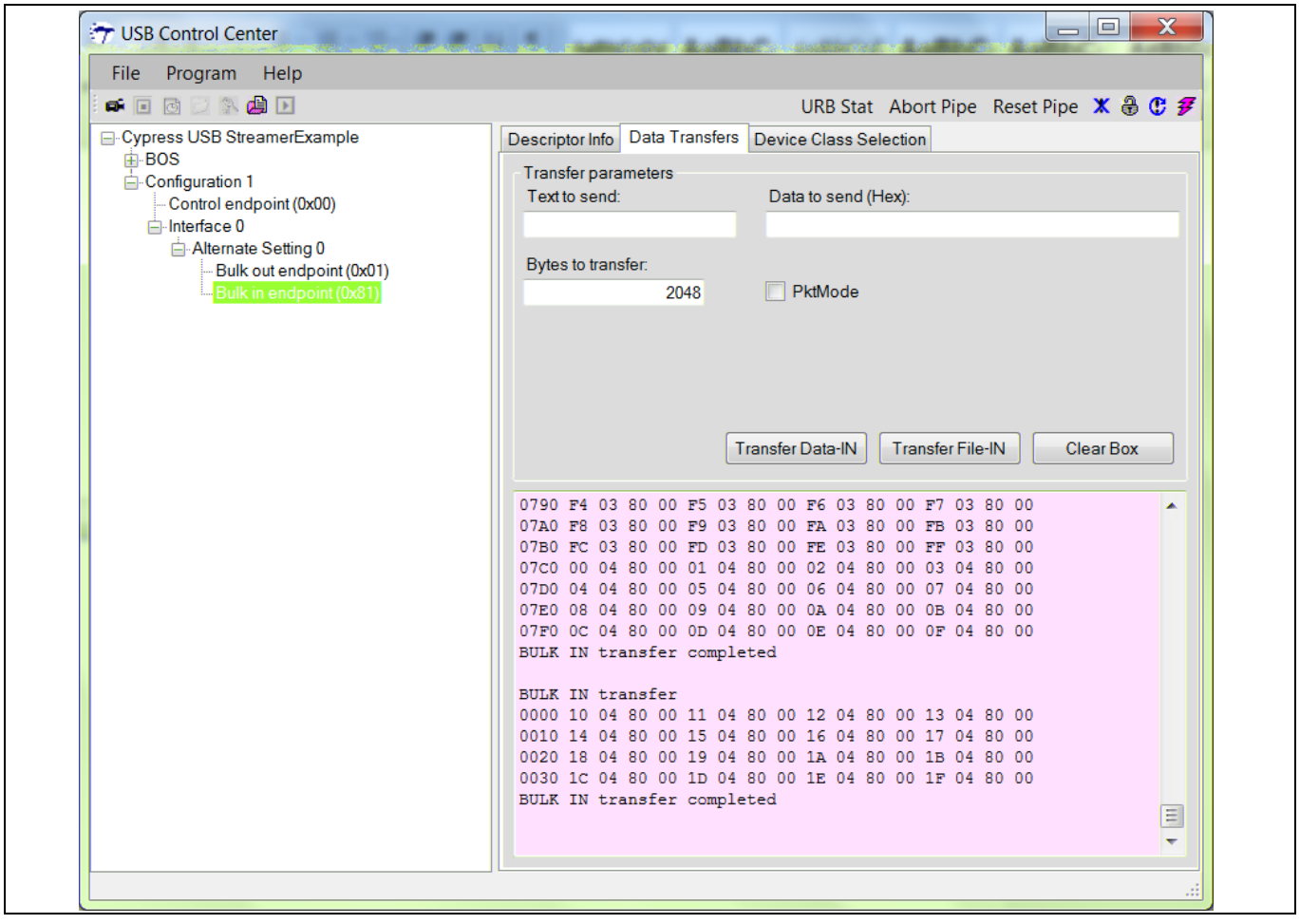

Figure 61 Reading short packets using Control Center

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 62**.

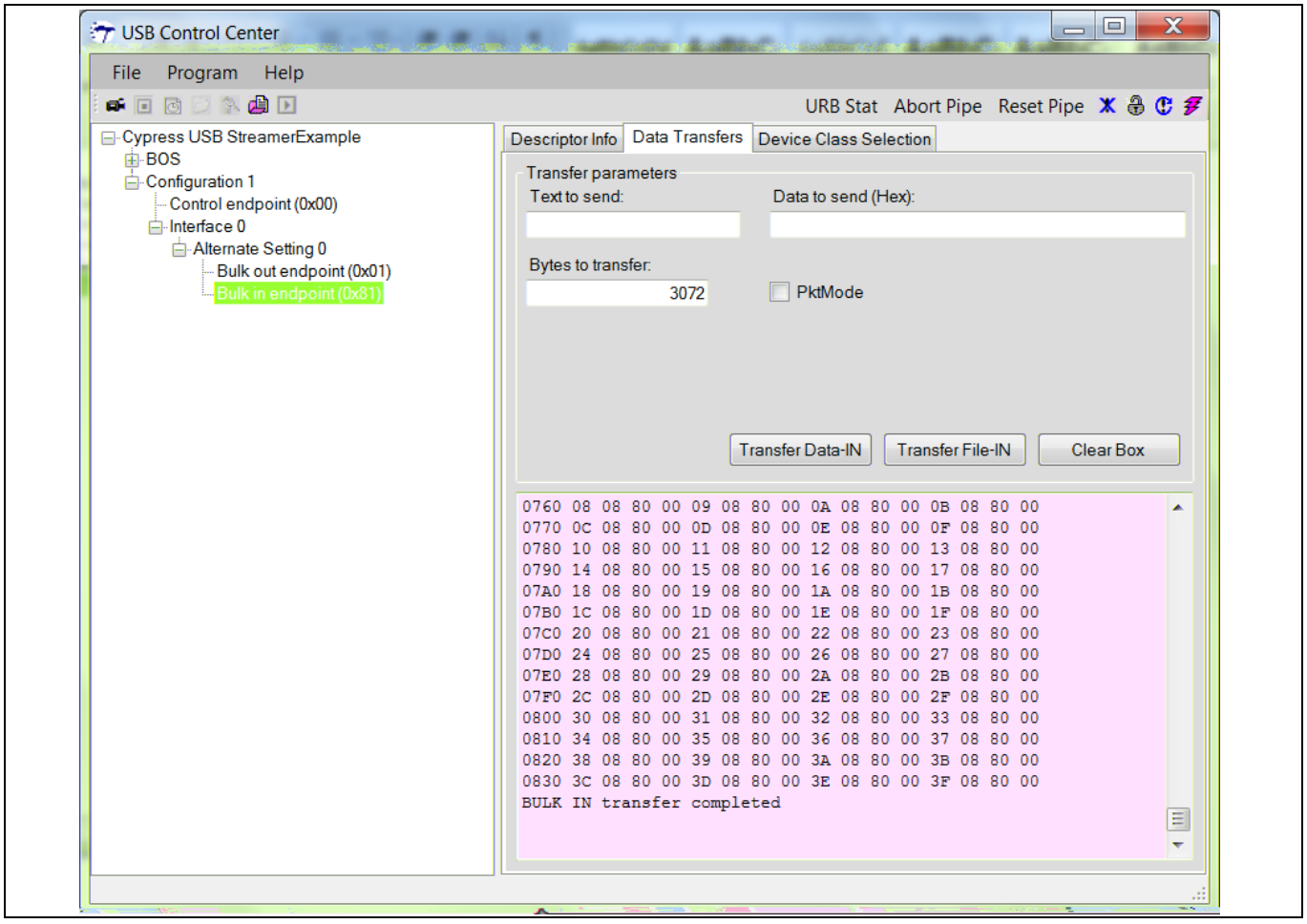

Figure 62 Short packet data transfers when requested bytes are more than DMA buffer size

17.2 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 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 63**.

Appendix C

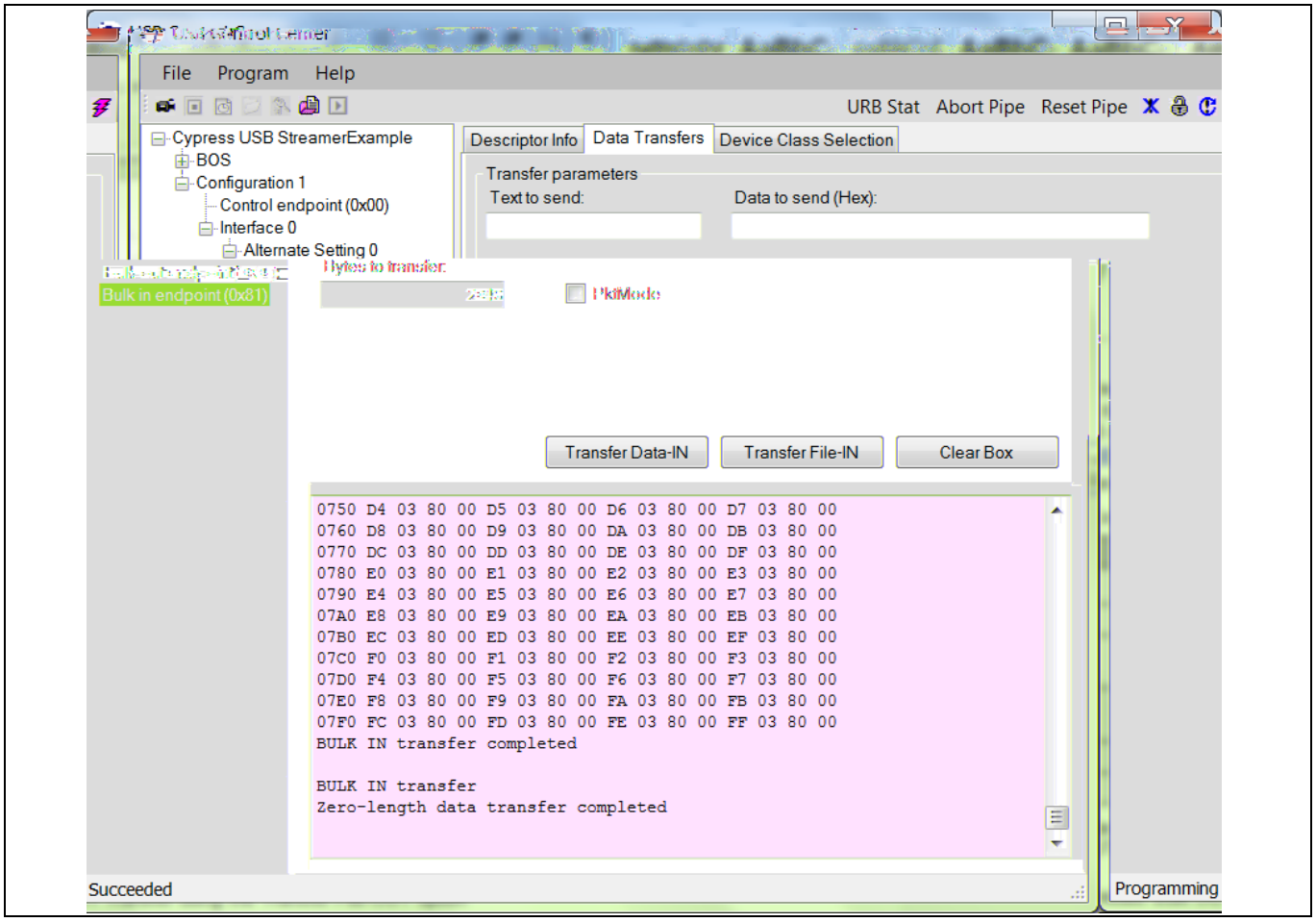

Figure 63 Reading ZLP using Control center

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 64**.

Appendix C

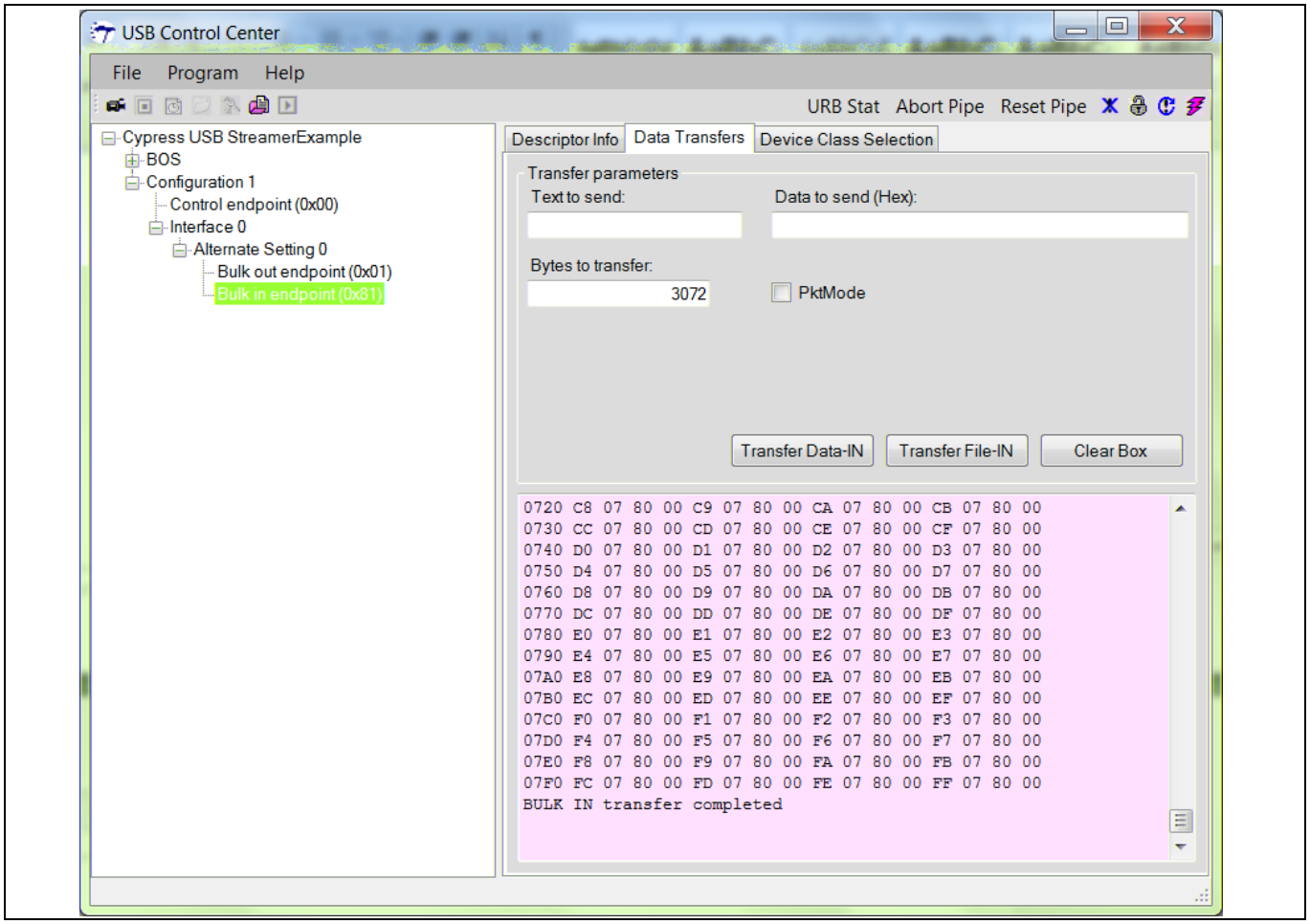

Figure 64 ZLP transfers when requested bytes are more than DMA buffer size

References

[1] **AN75705 - Getting started with EZ-USB™ FX3**

[2] **AN68829 - Slave FIFO interface for EZ-USB™ FX3: 5-bit address mode**

Revision history

Document version	Date of release	Description of changes
**	2010-12-21	New application note.
*A	2011-04-15	Updated abstract; added information on synchronous slave FIFO interfaces; expanded flag configuration section.
*B	2011-07-18	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	2011-12-22	Updated Async slave FIFO tPEL parameter Updated Sync slave FIFO Read and Write Timing diagrams
*D	2012-06-25	Updated to new application note template Complete rewrite of application note
*E	2012-08-13	Corrected broken links in the document.
*F	2012-09-21	Added a design example describing how a FPGA from Xilinx 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	2012-12-17	Template Update Updated the hardware setup options for executing the design example provided Added pictures of the hardware setup Added a section on error conditions that may occur if the flags are violated
*H	2013-06-11	Design example 1 is modified to have a Reset coming from FX3 to FPGA from Xilinx. Operating instructions for Design example 1 are modified. Design example 2 (Interfacing FX3 to FPGA from Altera) 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 FPGAs from Xilinx and Altera.
*I	2013-12-03	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.
*J	2014-06-18	Updated jumper J53 position in Table 6 and Table 8.
*K	2014-11-05	Updated the steps to run design examples 1 and 2 with the SuperSpeed Explorer Kit (CYUSB3KIT-003).

Revision history

Document version	Date of release	Description of changes
		Moved the hardware settings related to FX3 development kit (CYUSB3KIT-001) into Appendix A.
*L	2014-11-12	Added instructions to program FPGA in Project Operation section
*M	201511-04	Added instruction to program the FX3 DVK before the Xilinx FPGA board in the “Steps to Test Loopback Transfer” section. Updated tCO timings in Table 3. Added FX3 TRM reference for more details on GPIF threads and sockets Added VHDL test-bench code for FPGA from Xilinx Updated the firmware example code for enabling 16-bit interface using a macro
*N	2017-04-19	Updated logo and copyright
*O	2019-03-28	Updated Table 2: Pin Mapping Table for slave FIFO Interface Updated hyperlinks for ISE Suite from Xilinx and Quartus II web edition 12.1. Updated template
*P	2021-09-09	Updated **Table 2** Updated **Appendix A: Troubleshooting** Updated to Infineon template

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-09-09
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
001-65974 Rev. *P

IMPORTANT NOTICE
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.