AN65974

Designing with the EZ-USB FX3 Slave FIFO Interface

Author: Rama Sai Krishna V

Software Version: EZ-USB FX3 SDK1.3.4
Related Application Notes: AN75705, AN68829

More code examples? We heard you.
For a consolidated list of USB SuperSpeed Code Examples, visit www.cypress.com/101781.

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

Contents

1 Introduction

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

EZ-USB FX3 has a fully-configurable parallel, general programmable interface, called GPIF II, which can connect to an external processor, ASIC, or FPGA. GPIF II is an enhanced version of the GPIF in FX2LP, Cypress's flagship USB 2.0 product. GPIF II provides glueless connectivity to popular devices such as FPGAs, image sensors, and processors with interfaces such as the synchronous address data multiplexed interface.

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

This application note begins with 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 Xilinx Spartan 6 FPGA and Altera Cyclone III FPGA are provided. The corresponding FX3 firmware project for synchronous Slave FIFO is also included as part of the example. These examples have been developed using a Xilinx SP601 evaluation kit for the Spartan 6 FPGA and an Altera Cyclone III Starter Board for the Cyclone III FPGA, an FX3 development kit (DVK), and the FX3 software development kit (SDK).

2 More Information

Cypress 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: Cypress 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 Cypress 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

Cypress 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 Cypress-supplied interfaces, or choose to create their own GPIF II interface from scratch. Cypress 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, 16, 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 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, 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

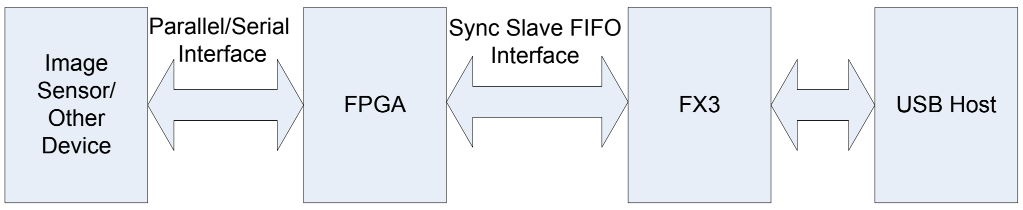


Image Sensor/ Other Device
Parallel/Serial Interface
FPGA
Sync Slave FIFO Interface
FX3
USB Host

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

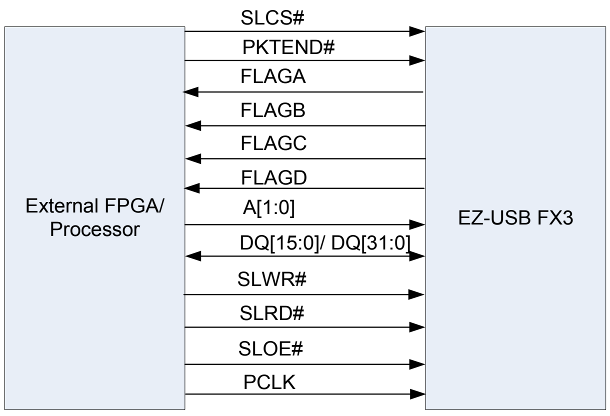


Table 1. Synchronous Slave FIIFO Interface Signals

Signal Name	Signal Description
SLCS#	This is the chip select signal for the Slave FIFO interface. It must be asserted to access 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^{2}S) 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	A1	A1
GPIO[29]	CTL[12]	GPIO	A0	A0	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]
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[33:41]	DQ[16:24]	Available as GPIOs	Available as GPIOs	DQ[16:24]	DQ[16:24]

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[42:44]	DQ[25:27]	Available as GPIOs	Available as GPIOs	Available as GPIOs	DQ[25:27]
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.

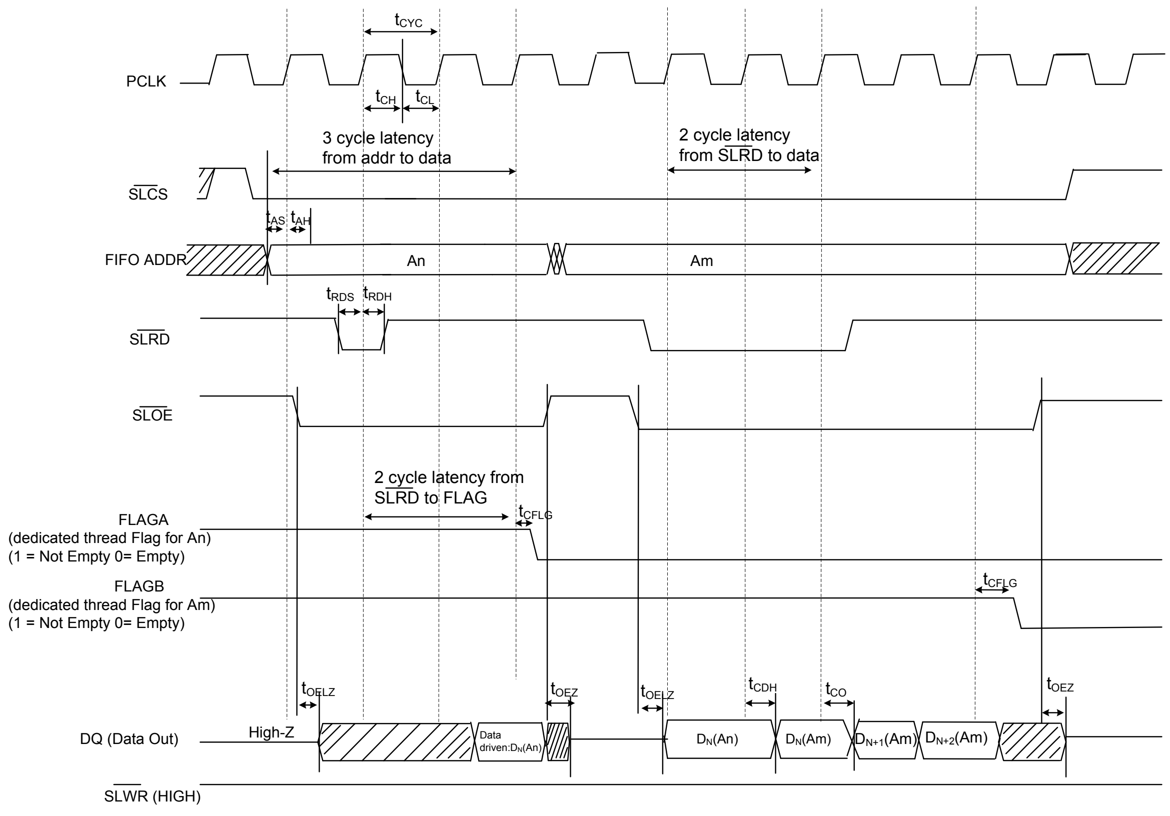
5 Slave FIFO Access Sequence and Interface Timing

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.

5.1 Synchronous Slave FIFO Interface Timing

Figure 3. Synchronous Slave FIFO Read Sequence


PCLK
tCYC
tCH
tCL
3 cycle latency from addr to data
2 cycle latency from SLRD to data
SLCS
tAS
tAH
FIFO ADDR
An
Am
tRDS
tRDH
SLRD
SLOE
2 cycle latency from SLRD to FLAG
tCFLG
FLAGA (dedicated thread Flag for An) (1 = Not Empty 0= Empty)
FLAGB (dedicated thread Flag for Am) (1 = Not Empty 0= Empty)
tOELZ
tOEZ
tCDH
tCO
DQ (Data Out)
High-Z
Data driven:DN(An)
DN(An)
DN(Am)
DN+1(Am)
DN+2(Am)
SLWR (HIGH)

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.

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

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

Figure 4. Synchronous Slave FIFO Write Sequence

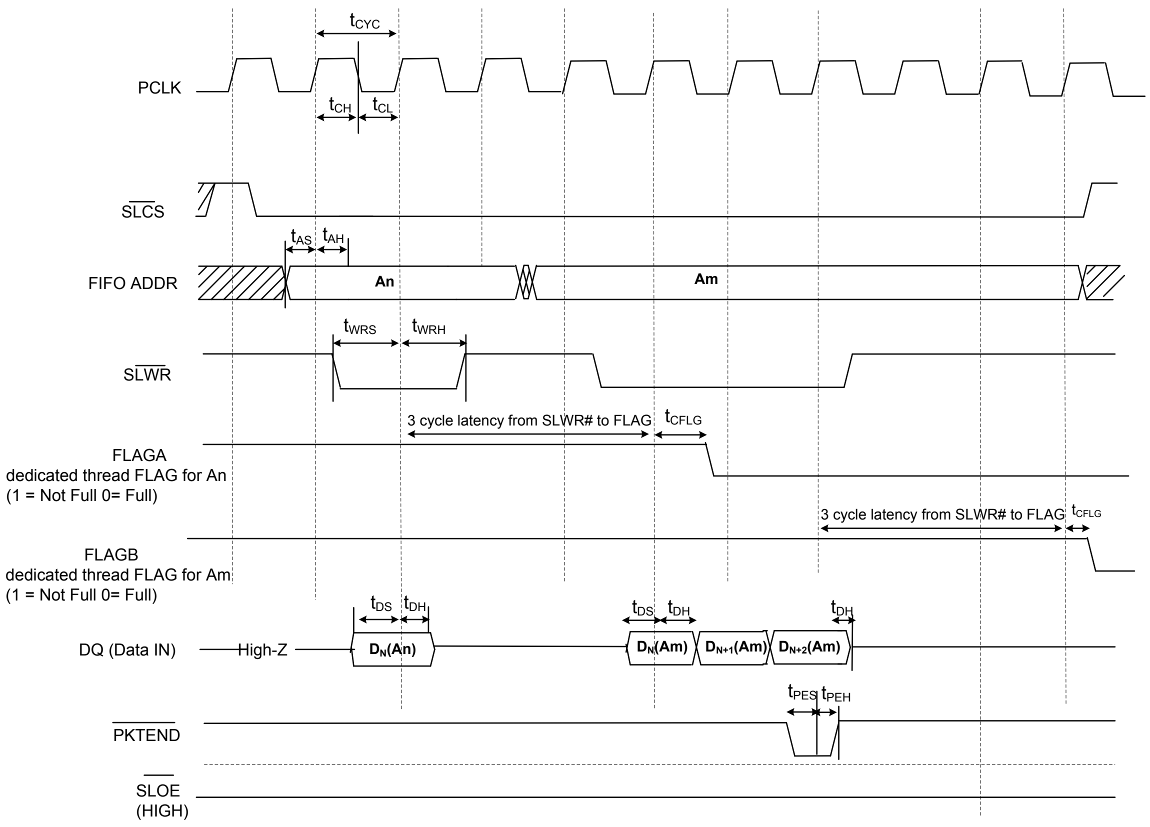
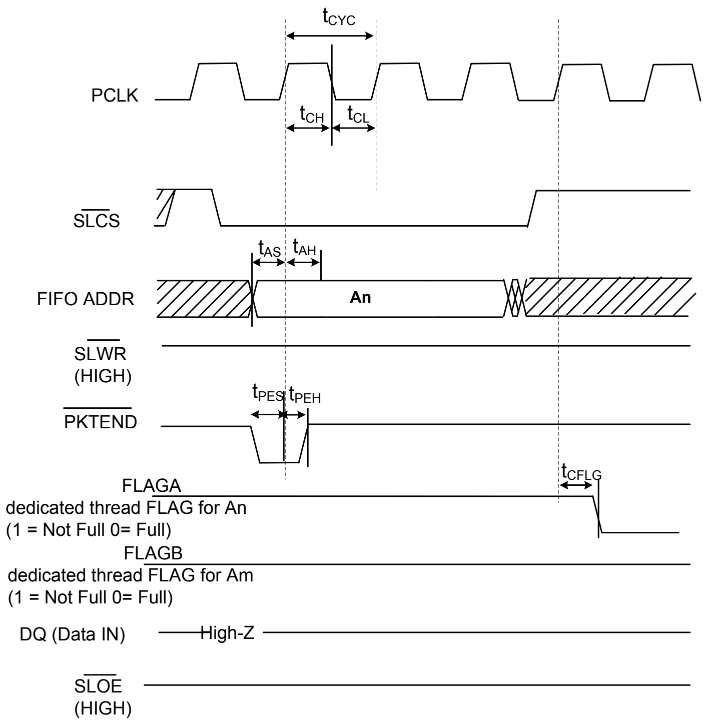


Figure 5. Synchronous ZLP Write Cycle Timing


tCYC
PCLK
tCH
tCL
SLCS
tAS
tAH
FIFO ADDR
An
SLWR (HIGH)
tPES
tPEH
PKTEND
tCFLG
FLAGA dedicated thread FLAG for An (1 = Not Full 0= Full)
FLAGB dedicated thread FLAG for Am (1 = Not Full 0= Full)
DQ (Data IN)
High-Z
SLOE (HIGH)

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.

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.

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.

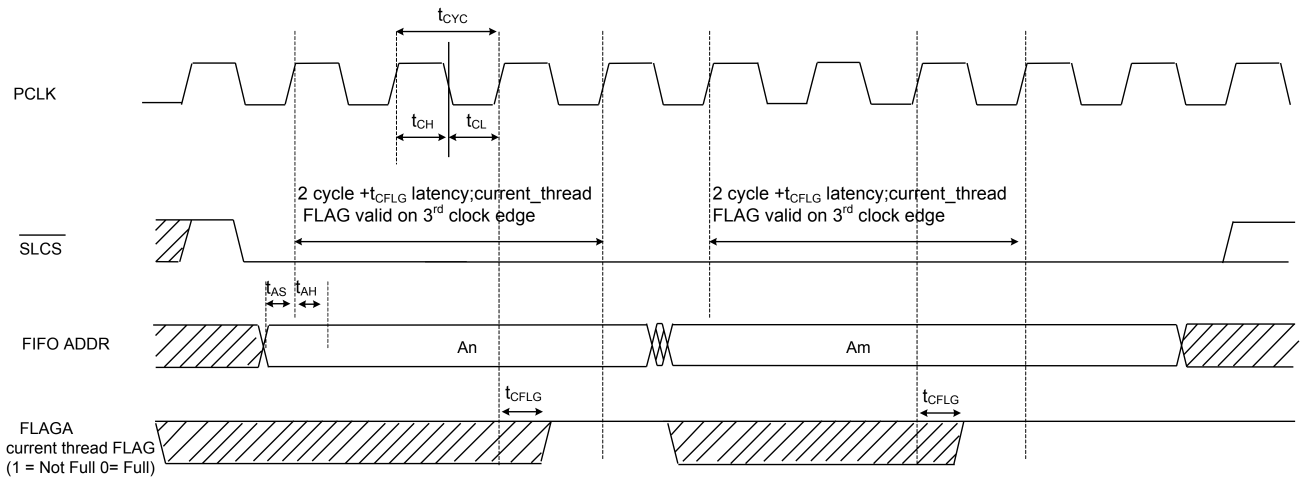
Note that when performing write transfers, a three-cycle latency for the flag is always incurred at the end of the transfer. The three-cycle latency is from the write cycle that causes the buffer to become full to the time the flag is asserted low. At the fourth clock edge, the external master can sample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

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


tCYC
PCLK
tCH
tCL
2 cycle +tCFLG latency;current_thread FLAG valid on 3rd clock edge
2 cycle +tCFLG latency;current_thread FLAG valid on 3rd clock edge
SLCS
tAS
tAH
FIFO ADDR
An
Am
tCFLG
tCFLG
FLAGA current thread FLAG (1 = Not Full 0= Full)

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.

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 (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 CyU3PGpilfSocketConfigure() 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.