AN68829

Slave FIFO Interface for EZ-USB® FX3™: 5-Bit Address Mode

Author: Sonia Gandhi
Associated Project: No
Associated Part Family: None
Software Version: None
Associated Application Notes: None

Abstract

AN68829 discusses asynchronous and synchronous Slave FIFO interfaces for the EZ-USB® FX3™ SuperSpeed USB controller. This application note also describes the mode in which the interface supports a 5-bit address bus and lets you access all 32 internal sockets of EZ-USB FX3.

Contents

Introduction

Cypress's EZ-USB FX3 is the USB 3.0 peripheral controller, which allows developers to add USB 3.0 functionality to any system.

EZ-USB FX3 has a fully configurable, parallel, general programmable interface, called GPIF II, which can connect to any processor, ASIC, or FPGA. GPIF II is an enhanced version of the GPIF in FX2LP, the flagship USB 2.0 product from Cypress. GPIF II provides glueless connectivity to such popular interfaces as asynchronous SRAM, asynchronous and synchronous address data multiplexed interfaces, and parallel ATA.

A popular implementation of GPIF II is the Slave FIFO interface, designed for applications where the external device connected to FX3 addresses the FIFOs in FX3, reading from or writing data to them. Direct register accesses are not performed over the Slave FIFO interface. This application note begins with a brief introduction to GPIF II and then discusses the Slave FIFO interface.

GPIF II

EZ-USB FX3 offers a general programmable interface (GPIF II) that enables functionality similar to, but more advanced than, FX2LP's GPIF and Slave FIFO interfaces.

GPIF II is a programmable state machine that enables a flexible interface that may function either as a master or a

slave in industry-standard or proprietary interfaces. Both parallel and serial interfaces may be implemented with GPIF II.

GPIF II has the following features:

- Functions as a master or a slave
- Provides 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. Those signals are driven as a result of GPIF II state transitions. The behavior of the GPIF II state machine is defined by a GPIF II descriptor, which is designed so that the required interface specifications are met. The GPIF II descriptor is essentially a set of programmable register configurations. Eight kB in EZ-USB FX3's register space are dedicated as GPIF II waveform memory, in which the GPIF II descriptor is stored in a specific format.

A popular implementation of GPIF II is the asynchronous/synchronous slave FIFO interface.

Slave FIFO Interface

For the GPIFII descriptor implementation for the Slave FIFO interface, refer to the Cypress Supplied Interfaces section of the GPIFII Designer.

On the Start Page of the GPIFII Designer tool, under Cypress Supplied Interfaces, these are the relevant projects:

async_slave_fifo_5bit: This is the implementation for the asynchronous Slave FIFO interface with 5-bit address bus

sync_slave_fifo_5bit: This is the implementation for the synchronous Slave FIFO interface with 5-bit address bus

Figure 1 shows the interface diagram for the asynchronous Slave FIFO interface.

Figure 2 shows the interface diagram for the synchronous Slave FIFO interface.

Figure 1. Asynchronous Slave FIFO Interface Diagram

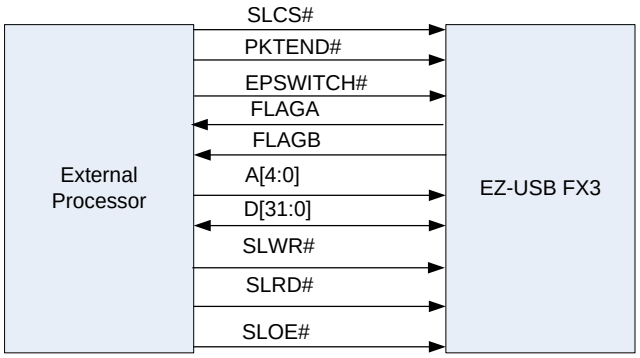
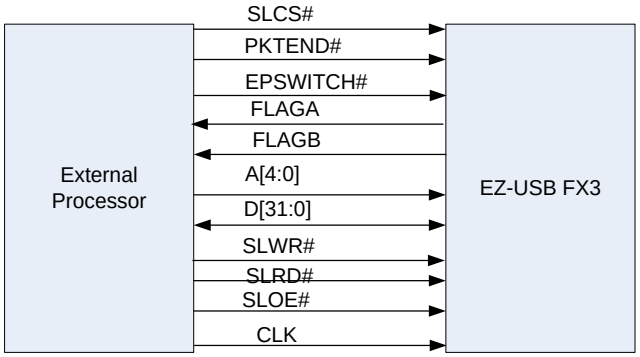


Figure 2. Synchronous Slave FIFO Interface Diagram



Pin Mapping of Slave FIFO Descriptors

The pin mapping of the Slave FIFO descriptors found in the SDK is shown in Table 1. This table also shows the GPIO pins and other serial interfaces (UART/SPI/I2S) available when GPIF II is configured for a Slave FIFO interface.

Table 1. Pin Mapping

EZ-USB FX3 Pin	Asynchronous Slave FIFO Interface with 16-bit Data bus	Synchronous Slave FIFO Interface with 16-bit Data bus	Asynchronous Slave FIFO Interface with 32-bit Data bus	Synchronous Slave FIFO Interface with 32-bit Data bus
GPIO[17]	SLCS#	SLCS#	SLCS#	SLCS#
GPIO[18]	SLWR#	SLWR#	SLWR#	SLWR#
GPIO[19]	SLOE#	SLOE#	SLOE#	SLOE#
GPIO[20]	SLRD#	SLRD#	SLRD#	SLRD#
GPIO[21]	FLAGA	FLAGA	FLAGA	FLAGA
GPIO[22]	FLAGB	FLAGB	FLAGB	FLAGB
GPIO[23]	EPSWITCH#	EPSWITCH#	EPSWITCH#	EPSWITCH#
GPIO[24]	PKTEND#	PKTEND#	PKTEND#	PKTEND#
GPIO[25]	A4	A4	A4	A4
GPIO[26]	A3	A3	A3	A3
GPIO[27]	A2	A2	A2	A2
GPIO[28]	A1	A1	A1	A1
GPIO[29]	A0	A0	A0	A0
GPIO[0:15]	DQ[0:15]	DQ[0:15]	DQ[0:15]	DQ[0:15]
GPIO[16]	Do Not Use	PCLK	Do Not Use	PCLK
GPIO[33:44]	Available as GPIOs	Available as GPIOs	DQ[16:27]	DQ[16:27]
GPIO[45]	GPIO	GPIO	GPIO	GPIO
GPIO[46]	GPIO/UART_RTS	GPIO/UART_RTS	DQ28	DQ28
GPIO[47]	GPIO/UART_CTS	GPIO/UART_CTS	DQ29	DQ29
GPIO[48]	GPIO/UART_TX	GPIO/UART_TX	DQ30	DQ30
GPIO[49]	GPIO/UART_RX	GPIO/UART_RX	DQ31	DQ31
GPIO[50]	GPIO/I2S_CLK	GPIO/I2S_CLK	GPIO/I2S_CLK	GPIO/I2S_CLK
GPIO[51]	GPIO/I2S_SD	GPIO/I2S_SD	GPIO/I2S_SD	GPIO/I2S_SD
GPIO[52]	GPIO/I2S_WS	GPIO/I2S_WS	GPIO/I2S_WS	GPIO/I2S_WS
GPIO[53]	GPIO/SPI_SCK /UART_RTS	GPIO/SPI_SCK /UART_RTS	GPIO/UART_RTS	GPIO/UART_RTS
GPIO[54]	GPIO/SPI_SSN/UART_CTS	GPIO/SPI_SSN/UART_CTS	GPIO/UART_CTS	GPIO/UART_CTS
GPIO[55]	GPIO/SPI_MISO/UART_TX	GPIO/SPI_MISO/UART_TX	GPIO/UART_TX	GPIO/UART_TX
GPIO[56]	GPIO/SPI_MOSI/UART_RX	GPIO/SPI_MOSI/UART_RX	GPIO/UART_RX	GPIO/UART_RX
GPIO[57]	GPIO/I2S_MCLK	GPIO/I2S_MCLK	GPIO/I2S_MCLK	GPIO/I2S_MCLK

Assigning Sockets to Threads and EPSWITCH#

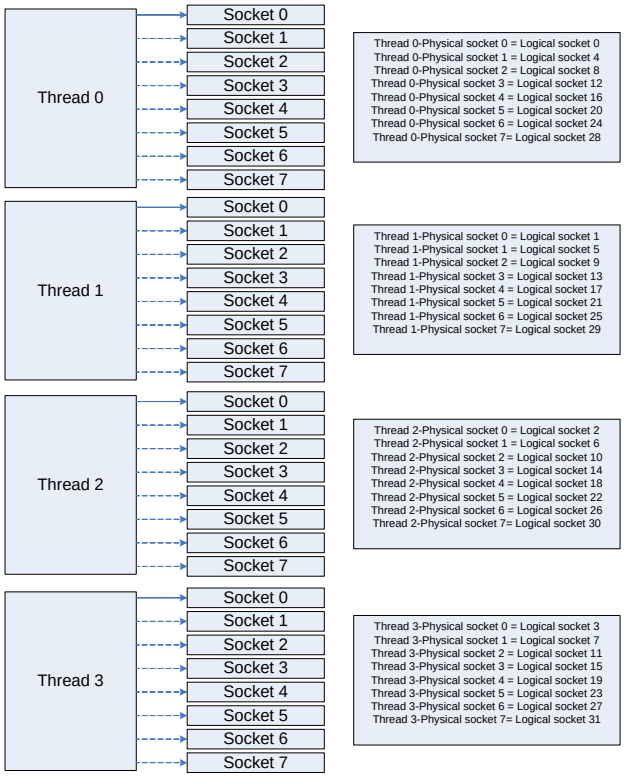
The EZ-USB FX3 FIFOs are associated with sockets. Sockets on the GPIF II side are similar to endpoints on the USB interface.

EZ-USB FX3 provides as many as four physical threads for data transfer over GPIF II. One socket at a time can be mapped to a thread.

Because EZ-USB FX3 implements only four physical threads, when more than four sockets are to be accessed over the Slave FIFO interface, a logical to physical socket address mapping is used. The EPSWITCH# signal changes the physical socket to which the thread is currently mapped.

This section explains how this logical to physical mapping is implemented and how to use the EPSWITCH# signal in addressing.

Figure 3. Logical to Physical Socket Address Mapping



The Slave FIFO interface supporting access to more than four sockets consists of five address lines, A4:A0. The value of A4:A0 on the interface represents the logical socket number.

When socket address A4:A0 is presented on the interface, FX3 interprets A1:A0 as the thread number and A4:A2 as the physical socket for thread A1:A0.

Example: When the external device needs to access logical socket 8, it should drive address 0x8 on the address bus. In this case:

A4:A0 = b'01000; i.e.A1:A0 = b'00 = 0x0 and A4:A2 = b'010 = 0x2.

After assertion of the EPSWITCH# signal, the physical socket of thread 0 changes to 0x2, and the data accesses will be performed on thread 0, physical socket 0x2, which is logical socket 8.

Usage of EPSWITCH# signal for address mapping

EPSWITCH# must be asserted to change the physical socket to which a particular thread is mapped.
To avoid unnecessary EPSWITCH# assertion cycles, the external device may maintain a record of the previously addressed physical socket for each thread.
By default, all four threads are mapped to their respective physical socket 0. This makes logical sockets 0, 1, 2, and 3 available as the default configuration.

After initialization, if the external device needs to address logical socket 0, 1, 2 or 3, the EPSWITCH# signal is not required to be asserted.

Now, suppose the external device needs to address logical socket 4. As explained above, this breaks down as

A4:A0 = b'00100; (logical address)

i.e.A1:A0 = b'00 = 0x0 (thread number) and A4:A2 = b'001= 0x1 (physical socket number).

Therefore, the physical socket for thread 0 now needs to be changed from the default of 0 to 0x1. In this case, assertion of the EPSWITCH# signal is required in the address phase.

After the physical socket number of thread 0 is changed, if the external device needs to go back to logical address 0, then again the physical socket number needs to be changed, and an EPSWITCH# assertion address phase is required.

Note that if the address is changed to a different thread, when you return to the original thread, an EPSWITCH# assertion is not needed (unless the physical socket needs to be changed again for the original thread).

For example, the physical socket number of thread 0 was changed to 0x1, as described above, to address logical socket number 0x4. Now, the external device needs to address logical socket 0x1.This maps to thread 1; physical socket 0. Because this is the default configuration for thread 1, an EPSWITCH# assertion is not required.

When the external device goes back to addressing logical socket 0x4, an EPSWITCH# assertion is not required. The reason is that the physical socket number of thread 0 had already been changed to 0x1.

The interface timing requirements for the EPSWITCH# assertion are shown in the timing diagrams in Figures 4 through 9.

DMA Channel Configuration

The firmware must configure a DMA channel with the required producer and consumer sockets.
Note that if data is to be transferred from the Slave FIFO interface to the USB interface, then P-port is the producer and USB is the consumer, and vice-versa.

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

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

Note that the P-port socket number in the DMA channel should be the logical socket number that will be addressed on A4:A0.

Use the CyUSBDmaChannelCreate() API to create the DMA channel.

An example of its usage may be found in the firmware examples of the FX3 SDK in the directory

C:\Program Files\Cypress\EZ-USB FX3 SDK\1.0\firmware\slavefifo_examples.

For documentation on this API, see the FX3 API Guide in the directory C:\Program Files\Cypress\EZ-USB FX3 SDK\1.0\doc.

Configuration of Flags

Flags may be configured as empty, full, partially empty, or partially full signals. They are routed directly from the DMA hardware engine internal to FX3, and they are not controlled by GPIF II.

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 indicates full/not full status if data is being written into the socket.

Dedicated Thread Flag

A flag can be configured to indicate the status of a particular thread. In this case, that flag is dedicated to that thread and always indicates the status of the socket mapped to that particular thread only, regardless of which thread is being addressed on the address bus. In this case, the external process or/device must track 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, then when the external processor performs accesses to thread 0, it must monitor FLAGA. When the external processor accesses thread 1, it must monitor FLAGB.

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 indicate the status of all four threads. However, a two-cycle latency is incurred when the current_thread flag is used for the 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.

Partial Flag

A flag can be configured to indicate the partially empty/full status of a socket. A watermark value is configured so that the flag gets asserted when the number of 32-bit words that may be read or written is less than or equal to the watermark value. Let’s say FLAGA is configured as a partial flag for thread 0, with a watermark value of 2. If the external processor/device performs write accesses to thread 0, then FLAGA gets asserted when there is room for two more 32-bit words to be written into the EZ-USB FX3’s FIFO buffer.
Note that the value of the watermark for a partial flag depends on the system, driven by the requirements and the latency with which the external processor can sample the FLAG.

As shown in the pin mapping in Table 1, only two pins are available for use as flags.

Typically, “current_thread” flags are used in 5-bit address Slave FIFO implementation, if all four threads are to be accessed (due to pin limitations).

You can configure the FLAG settings using GPIFII Designer.

Short Packets and Zero Length Packets

Short Packet

The PKTEND# signal is used to commit a short packet. As shown in the timing diagrams, the external processor must assert the PKTEND# signal along with the last word of data to be written (i.e., along with the assertion of SLWR# with the last word of the packet on the data bus). SLCS# must also be asserted.

Zero Length Packet (ZLP)

The PKTEND# signal and the EPSWITCH# signal must be asserted together for a ZLP to be committed. SLWR# should not be asserted for a ZLP transfer. SLCS# should be asserted. This will cause the first available buffer of the socket being addressed to be committed to the consumer (typically USB) with zero bytes of data.

Slave FIFO Timing and Access Sequence

An external processor or device (functioning as master of the interface) may perform single-cycle or burst data accesses to FX3's internal FIFO buffers. The external device presents the 5-bit address on the FIFO ADDR lines (with or without EPSWITCH#) before asserting the read or write strobes. The flag signals are asserted or deasserted by FX3 to indicate empty/full conditions of the buffer.

Figure 4 shows the timing and sequence for asynchronous Slave FIFO read accesses.

Figure 5 shows the timing and sequence for asynchronous Slave FIFO write accesses.

Figure 6 shows the ZLP protocol for asynchronous Slave FIFO protocol.

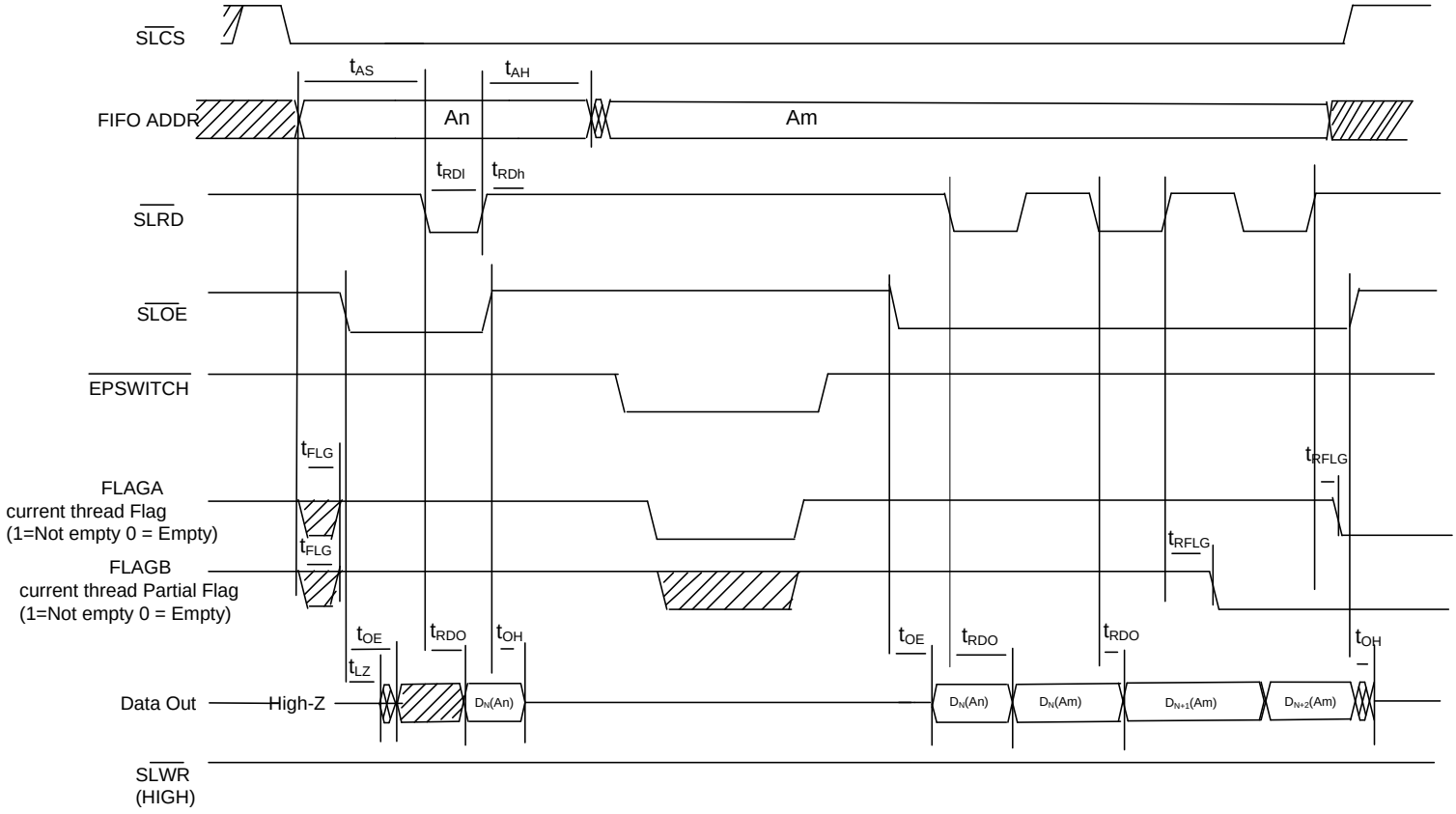
Figure 7 shows the timing and sequence for synchronous Slave FIFO read accesses.

Figure 8 shows the timing and sequence for synchronous Slave FIFO write accesses.

Figure 9 shows the ZLP protocol for the synchronous Slave FIFO protocol.

Figure 4. Asynchronous Slave FIFO Read

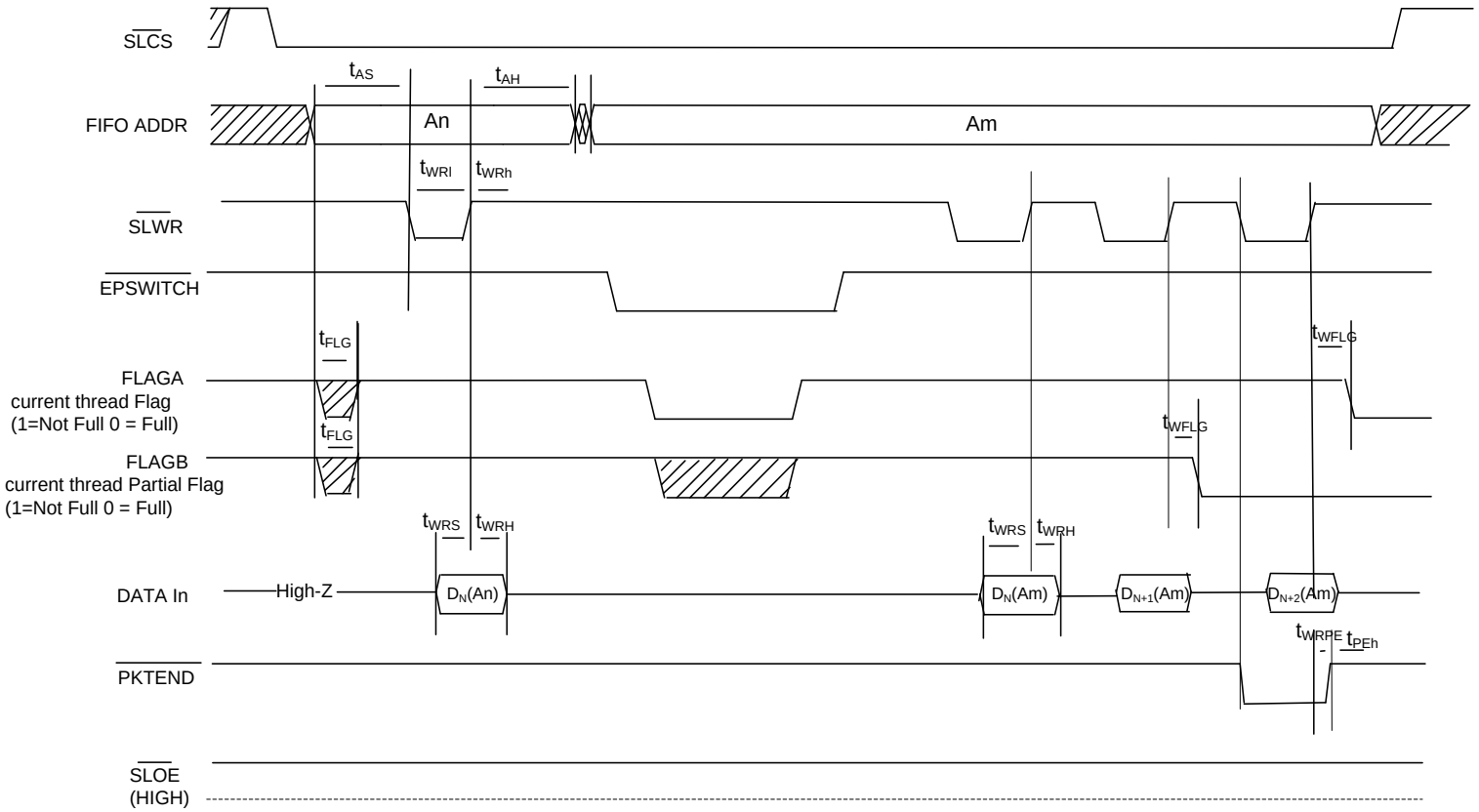
Asynchronous Read Cycle Timing



Note Refer to the EZ-USB FX3 datasheet for the values of the timing parameters.

Figure 5. Asynchronous Slave FIFO Write Sequence

Asynchronous Write Cycle Timing



Note Refer to the EZ-USB FX3 datasheet for the values of the timing parameters.

Figure 6. Asynchronous Slave FIFO ZLP Protocol

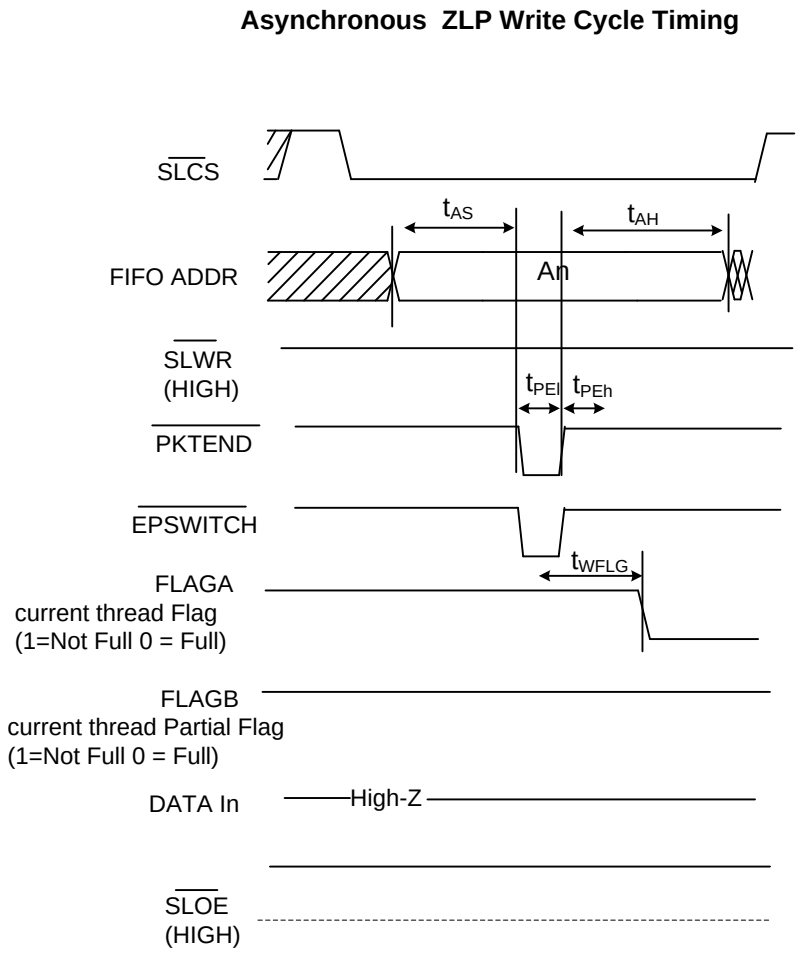


Figure 7. Synchronous Slave FIFO Read Sequence

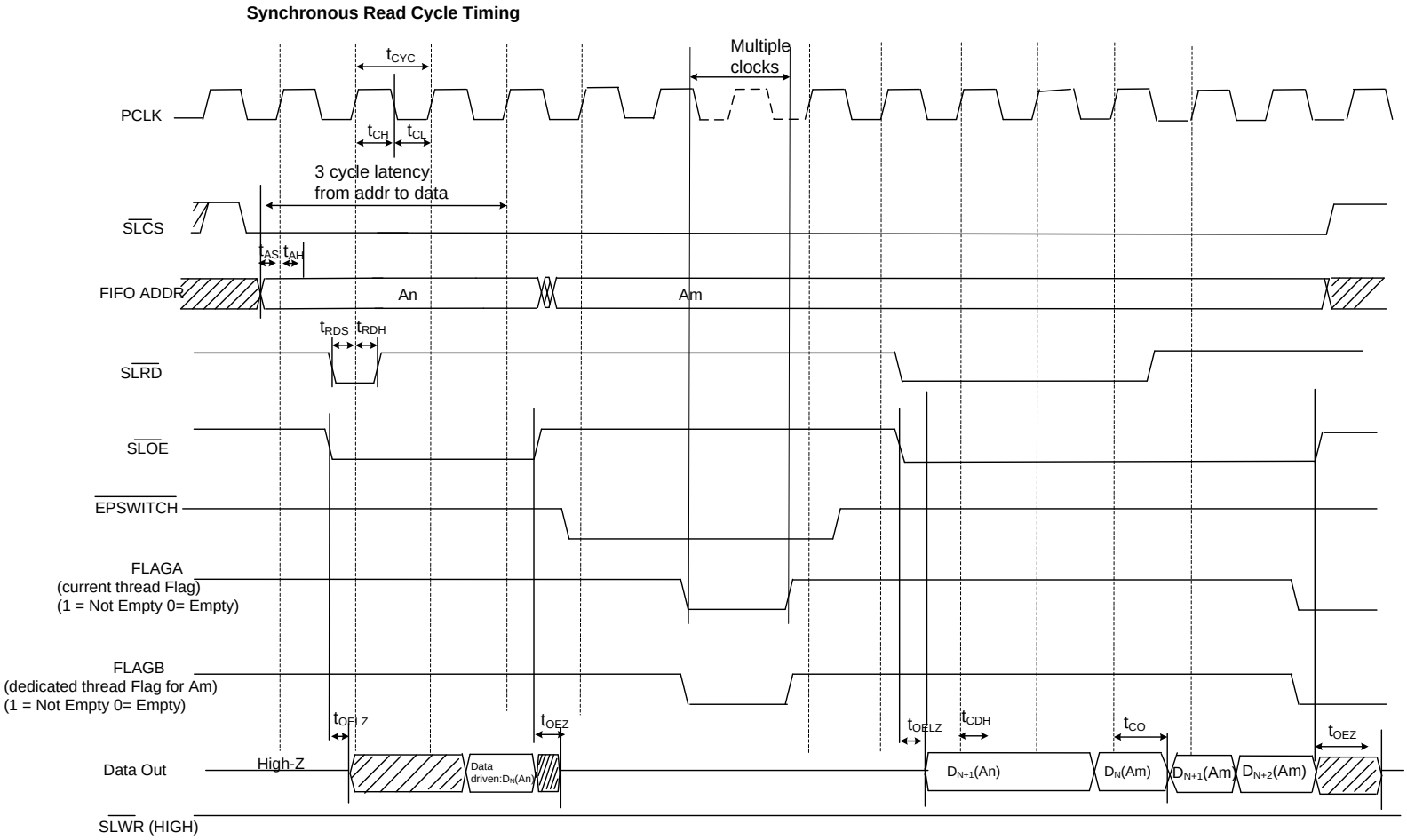


Figure 8. Synchronous Slave FIFO Write Sequence

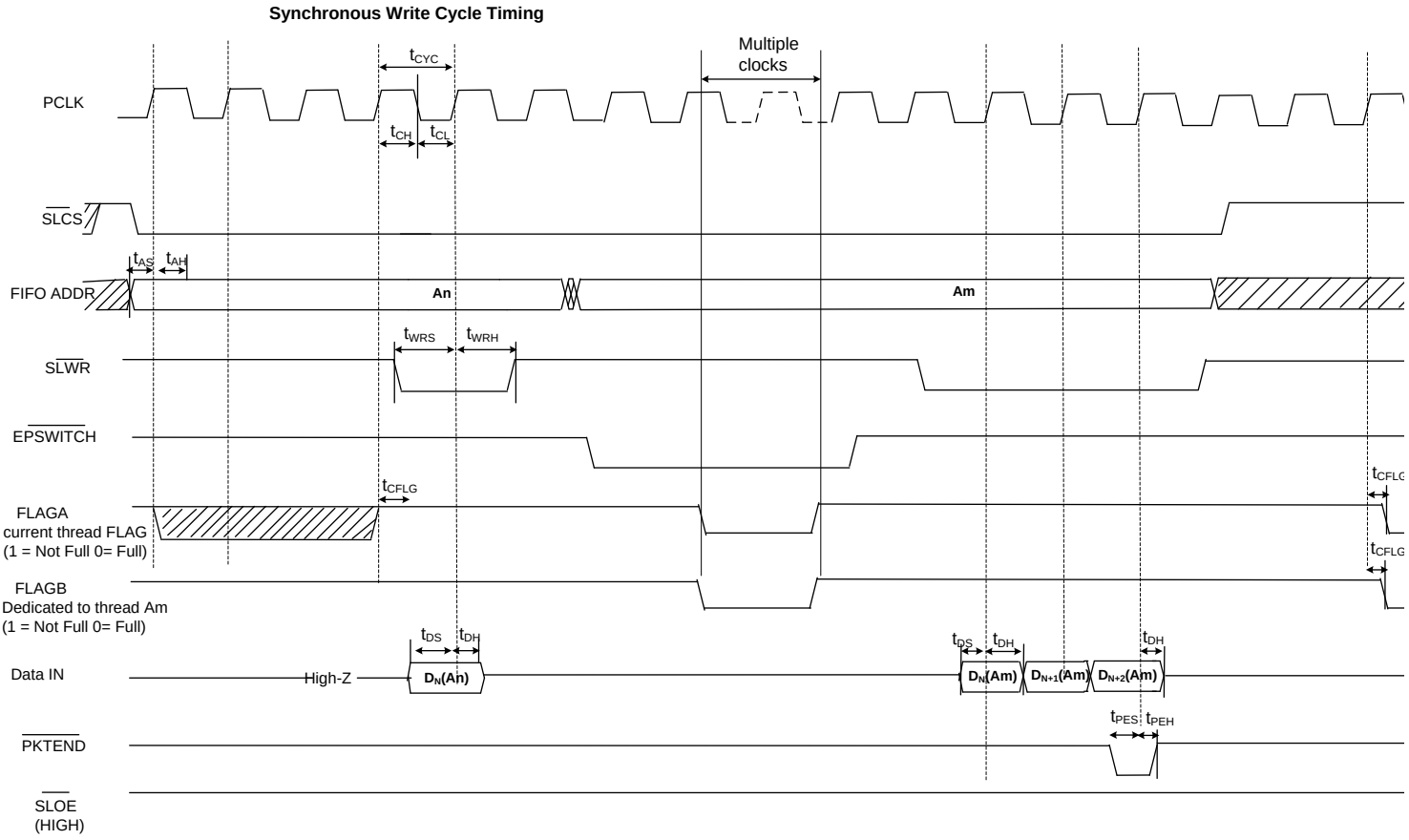
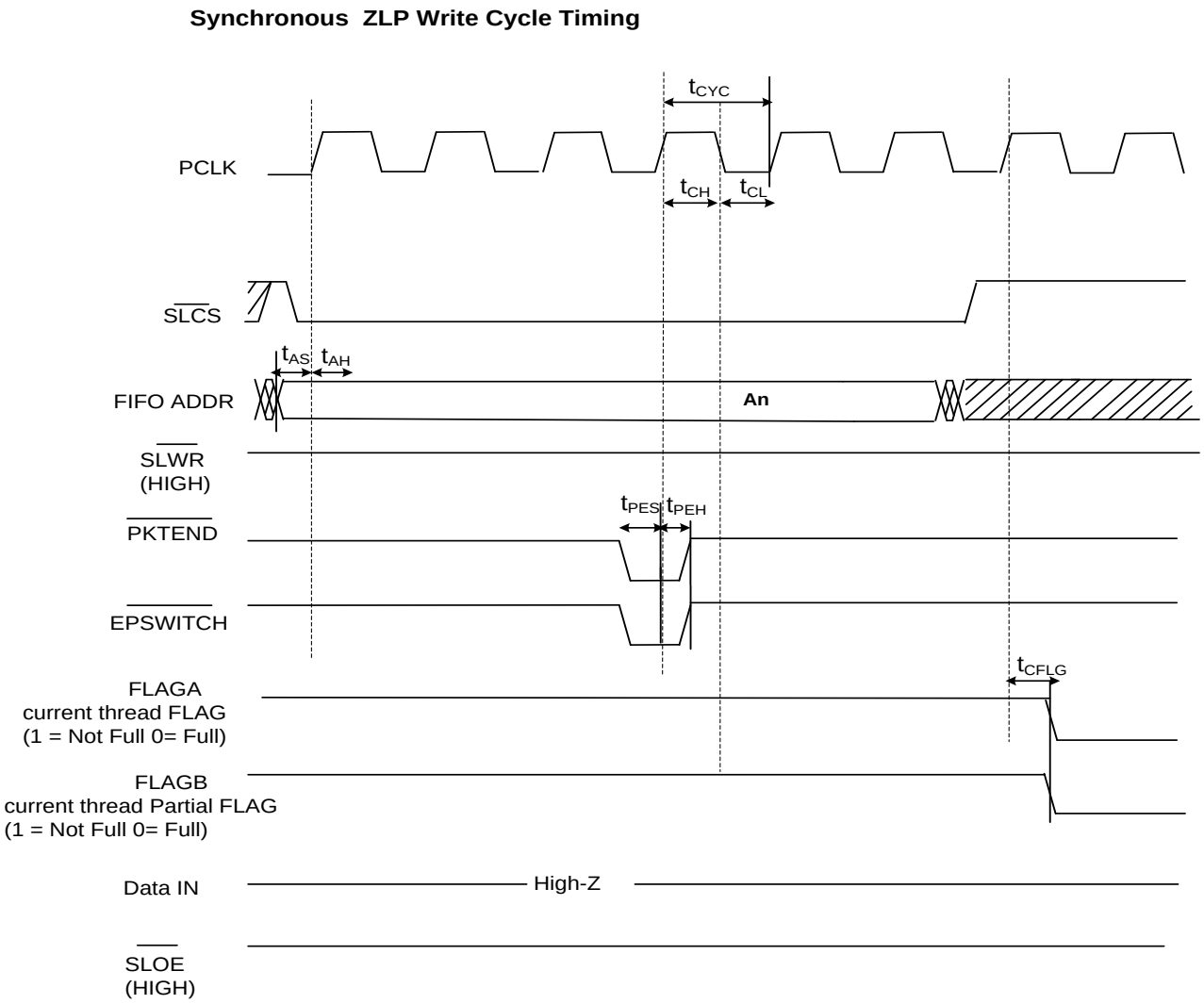


Figure 9. Synchronous Slave FIFO ZLP Protocol



About the Author

Name: Sonia Gandhi
Title: Applications Engineer Staff
Contact: osg@cypress.com

Document History

Document Title: Slave FIFO Interface for EZ-USB® FX3™: 5-Bit Address Mode
Document Number: 001-68829

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3217707	OSG	04/05/2011	New application note.
*A	3533843	OSG	02/23/2012	Updated timing diagrams Added a reference to where the projects can be found in GPIFII Designer Updated description of thread and socket mapping Updated description of FLAG configuration

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

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

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