AN68829

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

Author: Sonia Gandhi
Associated Project: No
Associated Part Family: EZ-USB FX3
Software Version: EZ-USB FX3 SDK 1.3.3
Associated Application Notes: AN75705, AN70707, AN65974

AN68829 discusses the 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. For USB SuperSpeed Code Examples, click here.

Contents

Introduction

Cypress's EZ-USB FX3 is the USB 3.0 peripheral controller that allows developers to add USB 3.0 functionality to any system. The controller works well with applications such as imaging and video devices, printers, scanners, and data acquisition systems.

EZ-USB FX3 has a fully configurable, parallel general programmable interface, called GPIF II, that 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 popular interfaces such as asynchronous SRAM, and asynchronous and synchronous address data multiplexed (ADMux) interfaces.

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 describes the details of the Slave FIFO interface.

GPIF II

GPIF II enables functionality similar to, but more advanced than, the FX2LP 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 offers the following features:

- Functions as a master or a slave
- Provides 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 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 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 asynchronous/synchronous Slave FIFO interface.

Slave FIFO Interface

For the GPIF II descriptor implementation for the Slave FIFO interface, refer to the Cypress-Supplied Interfaces section of the GPIF II Designer tool (part of the FX3 SDK installation).

On the Start Page of the GPIF II Designer tool, under Cypress Supplied Interfaces, are the relevant projects (as shown in Figure 1):

- async_slave_fifo_5bit: This is the implementation for the asynchronous Slave FIFO interface with a 5-bit address bus.
- sync_slave_fifo_5bit: This is the implementation for the synchronous Slave FIFO interface with a 5-bit address bus.

Figure 1. GPIF II Designer Tool With Cypress Supplied Interfaces

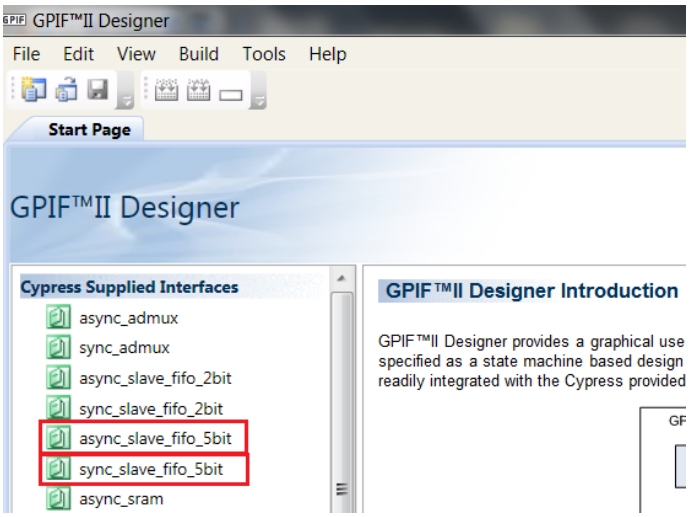


Figure 2 shows the interface diagram for the asynchronous Slave FIFO interface, while Figure 3 shows the interface diagram for the synchronous Slave FIFO interface. FX3 firmware for 5-bit Asynchronous Slave FIFO and 5-bit Synchronous Slave FIFO are available in the *firmware/slavefifo_examples* folder of the FX3 SDK.

Figure 2. Asynchronous Slave FIFO Interface Diagram

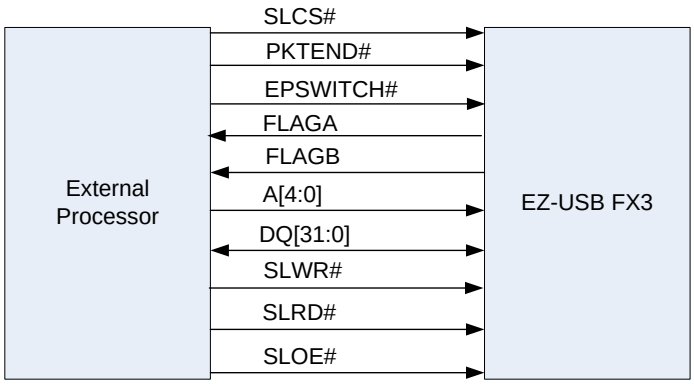
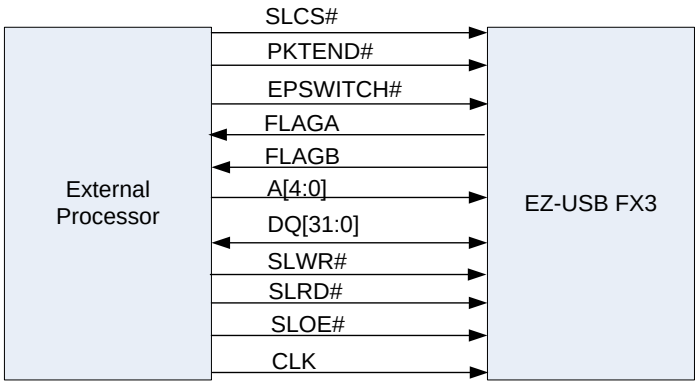


Figure 3. Synchronous Slave FIFO Interface Diagram



Difference Between Slave FIFOs with Two and Five Address Lines

The synchronous Slave FIFO interface with two address lines supports access to up to four sockets. If access to more than four sockets is required, then you should use the synchronous Slave FIFO interface with five address lines. In addition to the extra address lines, this interface has a signal called “EPSWITCH#.” Because more pins are used for this purpose, fewer pins are available for use as flags, requiring the flag to be 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 the address to FLAG valid state is incurred at the beginning of every transfer.
- Whenever a socket address is switched, multiple cycles of latency are incurred to complete socket switching. Empirical values of delays observed during socket switching is documented in Appendix.

Due to increased latencies and additional interface protocol requirements, it is recommended that the synchronous Slave FIFO interface with five address lines be used only if the application requires access to more than four GPIF II sockets. For details on the Slave FIFO interface with two address lines, refer to the application note AN65974 – Designing with the EZ-USB® FX3™ Slave FIFO Interface.

The remainder of this application note describes the details of the synchronous Slave FIFO interface with five address lines only. The following section describes the pin mapping of the signals shown in the Slave FIFO interface diagrams.

Pin Mapping of Slave FIFO Descriptors

Table 1 shows the pin mapping of the Slave FIFO descriptors in the SDK. It also shows the GPIO pins and other serial interfaces (UART, SPI, and I^2S) 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

Threads and Sockets

This section briefly explains the elements 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 flows through a peripheral is equal to the number of its sockets. The socket implementation includes a set of registers that 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 the 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 the RAM used for the intermediate storage of data transferred through the FX3 device. The FX3 firmware allocates DMA buffers from the RAM, and their addresses are stored as part of the DMA descriptors. For more information on DMA and socket, refer to chapter 5 (FX3 DMA Subsystem) of FX3 Technical Reference Manual

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. For example, take a data stream from the GPIF II block to the USB block. The GPIF II socket can tell the USB socket that it has filled data in a DMA buffer, and the USB socket can tell the GPIF II socket that a DMA buffer is empty. This implementation is called an “automatic DMA channel.” It is typically used when the FX3 CPU does not have to modify any data in a data stream.

Alternatively, the GPIF II socket can send an interrupt to the FX3 CPU to notify it that the GPIF II 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 empty a DMA buffer. Then, the FX3 CPU can relay this information back to the GPIF II socket. This implementation is called a “manual DMA channel.” It is typically 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. This latency can be a problem for interfaces that have no flow control. One such example is an image sensor interface.

The GPIF II block addresses this issue through the use of multiple GPIF threads. The GPIF II block implements four GPIF threads. Only one GPIF thread can transfer data at a time. The GPIF II state machine must select an active GPIF thread to transfer data.

The GPIF thread selection mechanism is like a mux. The GPIF II state machine uses internal control signals or external inputs to select the active GPIF thread. Switching the active GPIF thread will switch the active socket for the data transfer, thereby changing the DMA buffer used for data transfers. The GPIF thread switch has no latency. For more information on GPIF II, refer to chapter 7 (General Programmable Interface II) of FX3 Technical Reference Manual.

Assigning Sockets to Threads and EPSWITCH#

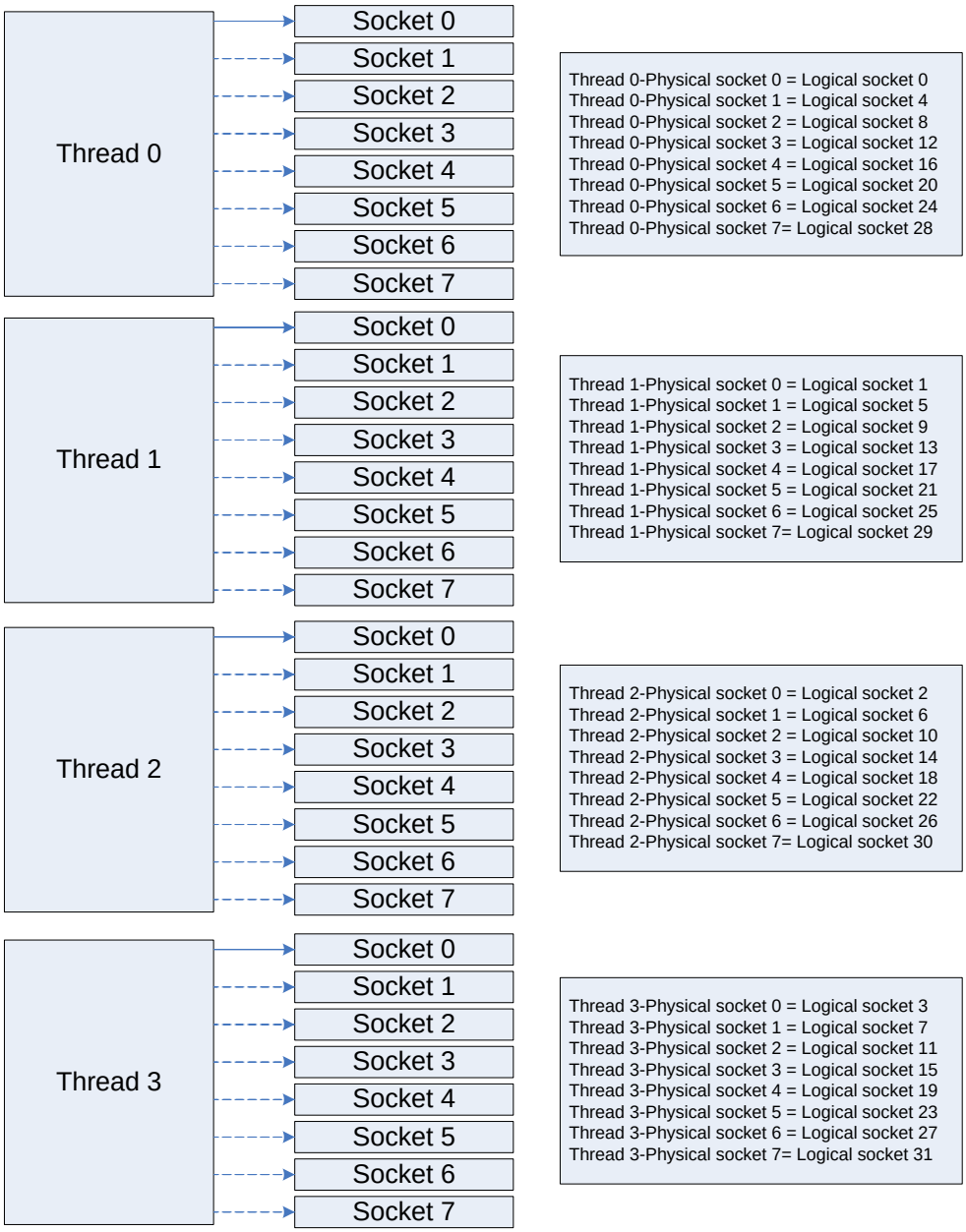
FX3 FIFOs are associated with sockets. Sockets on the GPIF II side are similar to endpoints on the USB interface.

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 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, as shown in Figure 4. The EPSWITCH# signal changes the physical socket to which the thread is currently mapped.

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

Figure 4. Logical-to-Physical Socket Address Mapping



The Slave FIFO interface, which supports 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 the 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.

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

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

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

Usage of EPSWITCH# Signal for Address Mapping

The EPSWITCH# signal 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 previously, this breaks down as follows:

A4:A0 = 00100b; (logical address)

That is: A1:A0 = 00b = 0x0 (thread number) and A4:A2 = 001b= 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 previously, to address logical socket number 0x4. Now the external device needs to address the logical socket 0x1, which 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 the logical socket 0x4, an EPSWITCH# assertion is not required because the physical socket number of thread 0 was already changed to 0x1.

The timing diagrams in Figure 5 through Figure 10 show the interface timing requirements for EPSWITCH# assertion.

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 is located in the firmware examples of the FX3 SDK in the folder

C:\Program Files\Cypress\EZ-USB FX3SDK\1.3\firmware\slavefifo_examples.[1]

For documentation on this API, see the *FX3 API Guide* in the folder *C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3\doc*.[1]

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.

[1] For 64-bit systems, the first folder in the path is Program Files(x86). The number 1.3 in the directory path is the version number of the SDK, and it can vary based on the latest release of the FX3 SDK.

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 thread only, regardless of which thread is being addressed on the address bus. 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. Similarly, 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 state machine must first 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 afterward, 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. 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 FX3 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 the 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 GPIF II Designer. Refer to the “Flag Configuration” section of AN65974 – Designing with the EZ-USB® FX3™ Slave FIFO Interface to learn about how these flags can be configured and the delays associated with them.

Slave FIFO Timing and Access Sequence

An external processor or device (functioning as the 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. FX3 asserts or deasserts the FLAG signals to indicate empty/full conditions of the buffer.

Figure 5 shows the timing and sequence for asynchronous Slave FIFO read accesses. Figure 6 shows the timing and sequence for asynchronous Slave FIFO write accesses. Figure 7 shows the ZLP protocol for asynchronous Slave FIFO protocol.

Asynchronous Slave FIFO Read Sequence

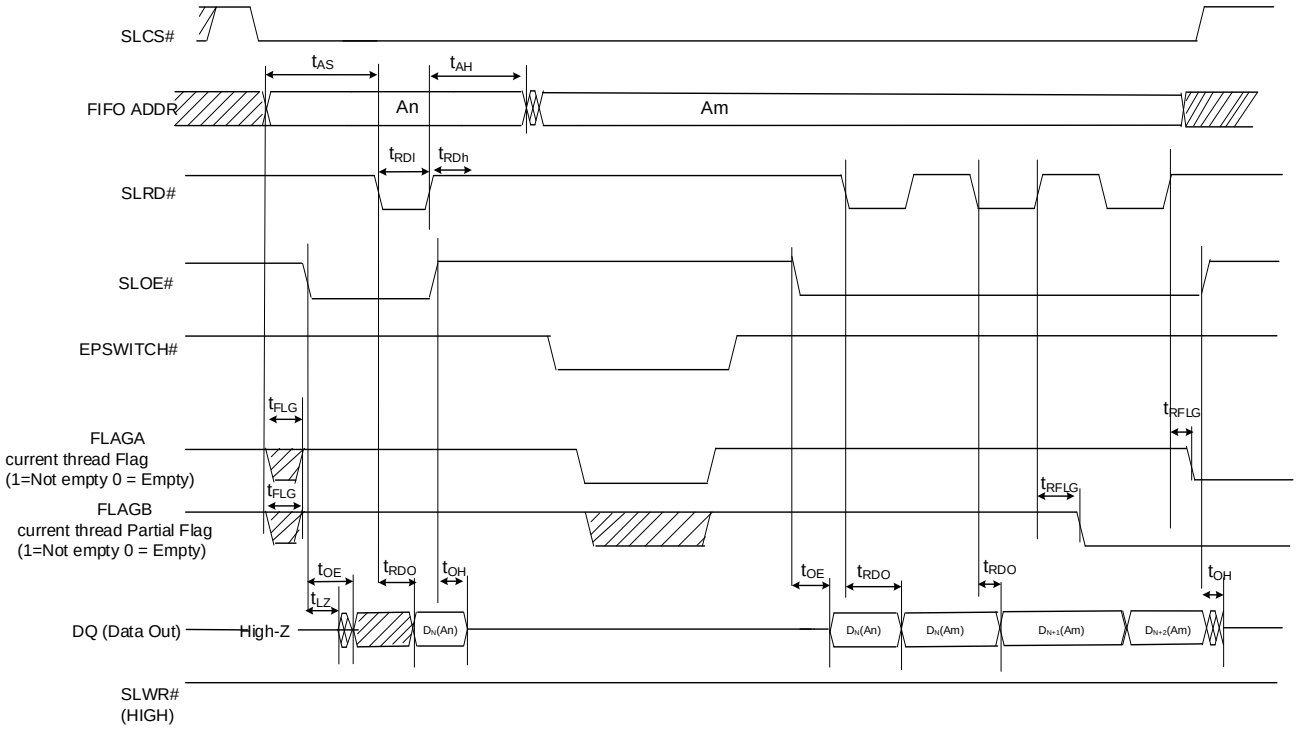
The sequence for performing reads from the asynchronous Slave FIFO interface is as follows:

1. The FIFO address is stable and the SLCS# signal is asserted.
2. Assert EPSWITCH# (per the discussion in Usage of EPSWITCH# Signal for Address Mapping) and wait until the valid status of FLAG (FLAGA becomes 1, indicating data availability in the DMA buffer), and deassert it.
3. SLOE# is asserted, which results in driving the data bus.
4. SLRD # is asserted.
5. Data from the FIFO is driven after the assertion of SLRD#. This data is valid after a propagation delay of t_{RDO} from the falling edge of SLRD#.
6. The FIFO pointer is incremented on deassertion of SLRD#.

In Figure 5, data N (D_N) is the first valid data read from the FIFO. For data to appear on the data bus during the read cycle, SLOE# must be in an asserted state. SLRD# and SLOE# can also be tied.

A burst read follows the same sequence of events. Note that in the burst-read mode, during the SLOE# assertion, the data bus is in a driven state (data is driven from a previously addressed FIFO). After assertion of SLRD#, data from the FIFO is driven on the data bus (SLOE# must also be asserted). The FIFO pointer is incremented after deassertion of SLRD#.

Figure 5. Asynchronous Slave FIFO Read



Note Refer to Table 2 for the values of the timing parameters.

Asynchronous Slave FIFO Write Sequence

The sequence for performing writes to the synchronous Slave FIFO interface is as follows.

1. The FIFO address is stable and SLCS# is asserted.
2. Assert EPSWITCH# (per the discussion in Usage of EPSWITCH# Signal for Address Mapping) and wait until the valid status of FLAG (FLAGA becomes 1, indicating the availability of DMA buffer), and deassert it.
3. SLWR# is asserted. SLCS# must be asserted with SLWR# or before SLWR# is asserted.
4. Data must be present on the bus t_{WRS} before the deasserting edge of SLWR#.
5. Deassertion of SLWR# causes the data to be written from the data bus to the FIFO, and then the FIFO pointer is incremented.
6. The FIFO flag is updated after t_{WFLG} from the deasserting edge of SLWR#.

A burst write follows the same sequence of events. Note that in the burst write mode, after SLWR# deassertion, the data is written to the FIFO, and then the FIFO pointer is incremented.

Short Packet: A short packet can be committed to the USB host by using the PKTEND#. The external device or 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.

Zero-Length Packet: 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, but 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.

FLAG usage: The external processor monitors the FLAG signals for flow control. FLAG signals are FX3 outputs that can be configured to show empty, full, and partial status for a dedicated address or the current address.

Figure 6. Asynchronous Slave FIFO Write Sequence

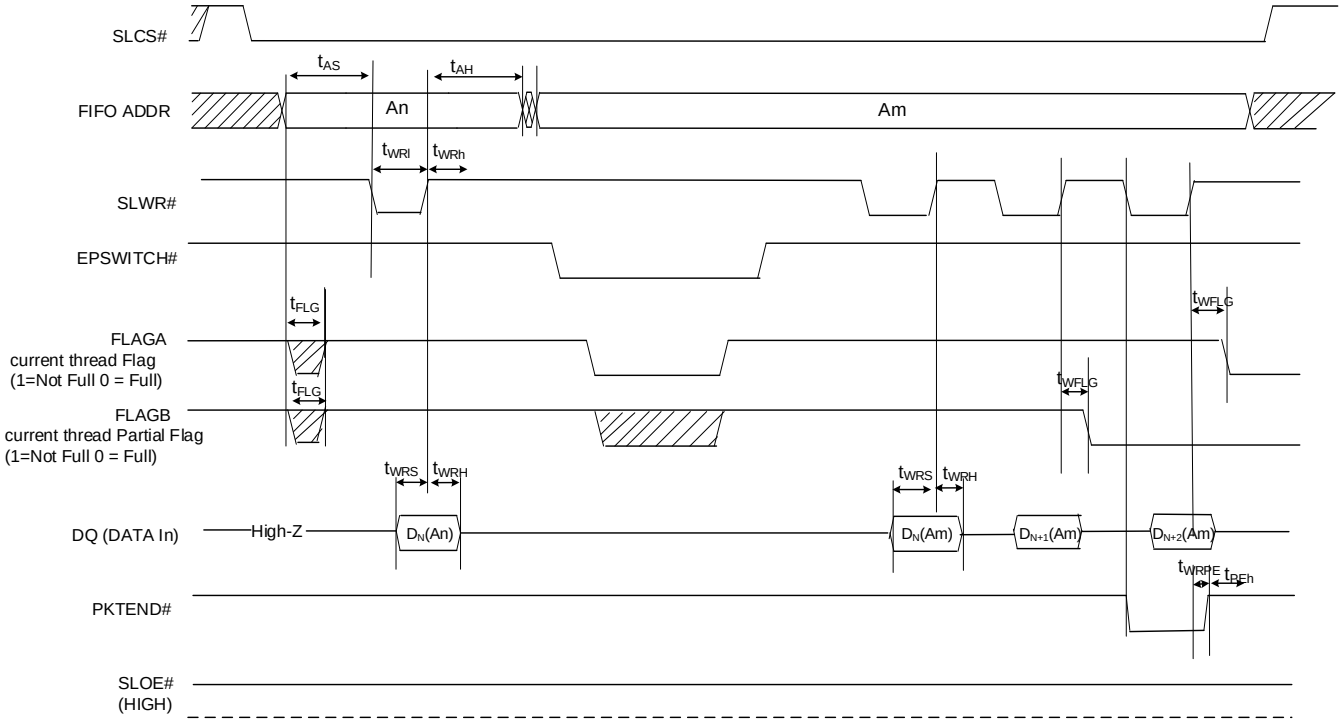
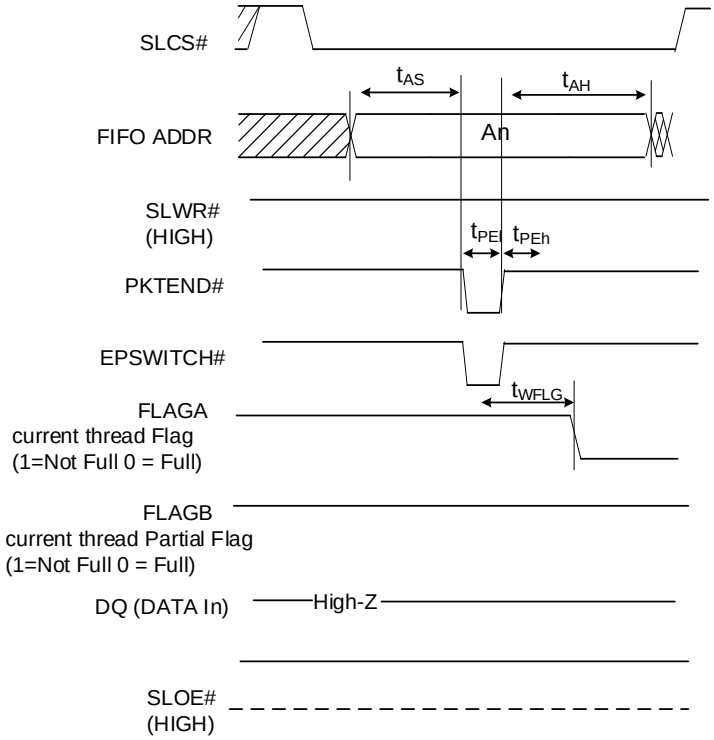


Figure 7. Asynchronous Slave FIFO ZLP Protocol



Note Refer to Table 2 for the values of the timing parameters.

Table 2. Asynchronous Slave FIFO Timing Parameters

Parameter	Description	Min	Max	Unit
t_{RDI}	SLRD# LOW	20	–	ns
t_{RDh}	SLRD# HIGH	10	–	ns
t_{AS}	Address to SLRD#/SLWR# setup time	7	–	ns
t_{AH}	SLRD#/SLWR#/PKTEND to address hold time	2	–	ns
t_{RFLG}	SLRD# to FLAGS output propagation delay	–	35	ns
t_{FLG}	ADDR to FLAGS output propagation delay	–	22.5	ns
t_{RDO}	SLRD# to data valid	–	25	ns
t_{OE}	OE# low to data valid	–	25	ns
t_{LZ}	OE# low to data low-Z	0	–	ns
t_{OH}	SLOE# deassert data output hold	–	22.5	ns
t_{WRI}	SLWR# LOW	20	–	ns
t_{WRh}	SLWR# HIGH	10	–	ns
t_{WRS}	Data to SLWR# setup time	7	–	ns
t_{WRH}	SLWR# to data hold time	2	–	ns
t_{WFLG}	SLWR#/PKTEND to FLAGS output propagation delay	–	35	ns
t_{PEI}	PKTEND# LOW	20	–	ns
t_{PEH}	PKTEND# HIGH	7.5	–	ns
t_{WRPE}	SLWR# deassert to PKTEND# deassert	2	–	ns

Synchronous Slave FIFO Read Sequence

The sequence for performing reads from the synchronous Slave FIFO interface is as follows.

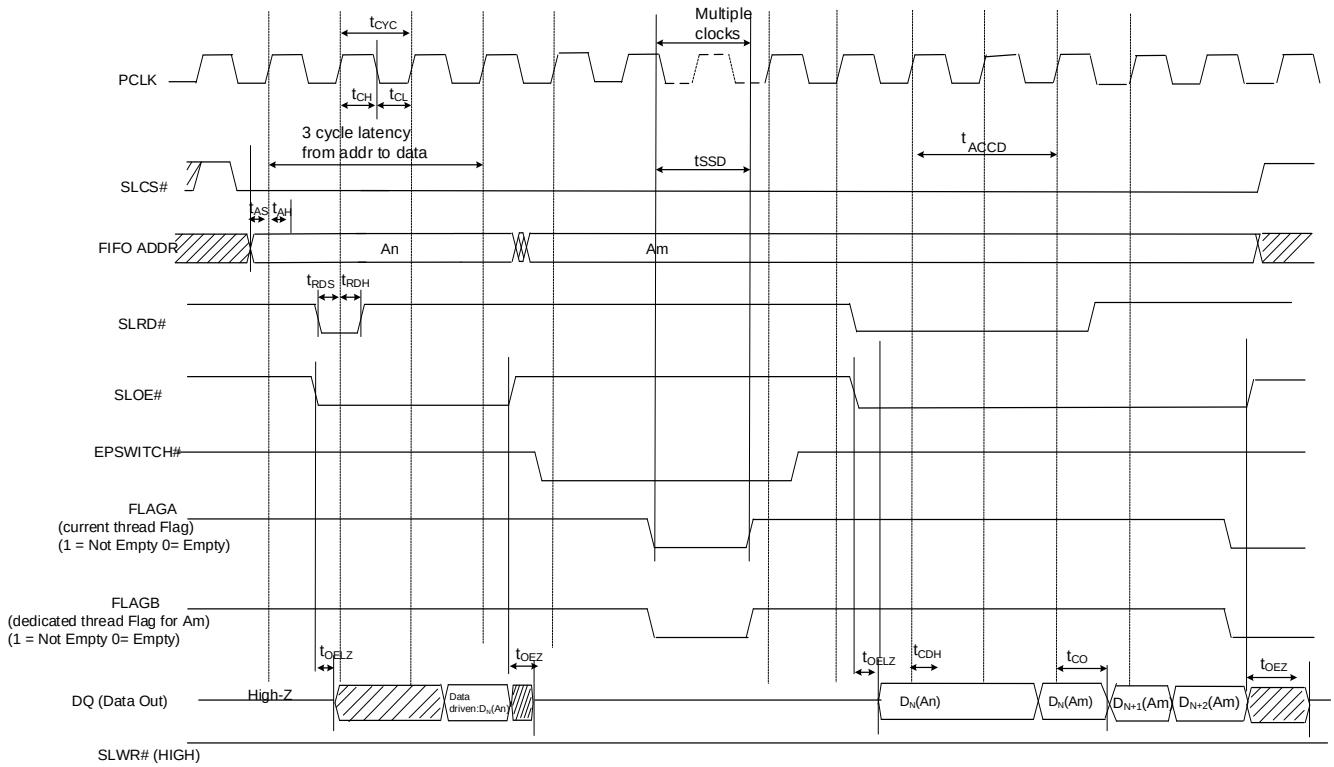
1. The FIFO address is stable and SLCS# is asserted.
2. Assert EPSWITCH# (per the discussion in Usage of EPSWITCH# Signal for Address Mapping) and wait until the valid status of FLAG (FLAGA becomes 1, indicating data availability in the DMA buffer) and deassert it.
3. SLOE# is asserted. SLOE# is an output enable signal, which has driving the data bus as the only function.
4. 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. D_N is the first data value read from the FIFO. To drive the data bus, SLOE# must also be asserted.

A burst read follows the same sequence of events. Note that for burst mode, the SLRD# and SLOE# are left 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 SLRD# is asserted, the FIFO pointer is incremented and the next data value is placed on the data bus.

FLAG usage: The FLAG signals are monitored by the external processor for flow control. FLAG signals are outputs from FX3 that may be configured to show empty/full/partial status for a dedicated thread or the current thread being addressed.

Figure 8. Synchronous Slave FIFO Read Sequence



Note Refer to Table 3 for values of the timing parameters.

Synchronous Slave FIFO Write Sequence

The sequence for performing writes to the synchronous Slave FIFO interface is as follows.

1. The FIFO address is stable and the signal SLCS# is asserted.
2. Assert EPSWITCH# (per the discussion in Usage of EPSWITCH# Signal for Address Mapping) and wait until the flag becomes valid (FLAGA becomes 1, indicating the availability of DMA buffer) and deassert it.
3. External master/peripheral outputs the data onto the data bus
4. SLWR# is asserted.
5. While SLWR# is asserted, data is written to the FIFO and on the rising edge of PCLK, the FIFO pointer is incremented.
6. The FIFO flag is updated after a delay of t_{CFLG} from the rising edge of the clock.

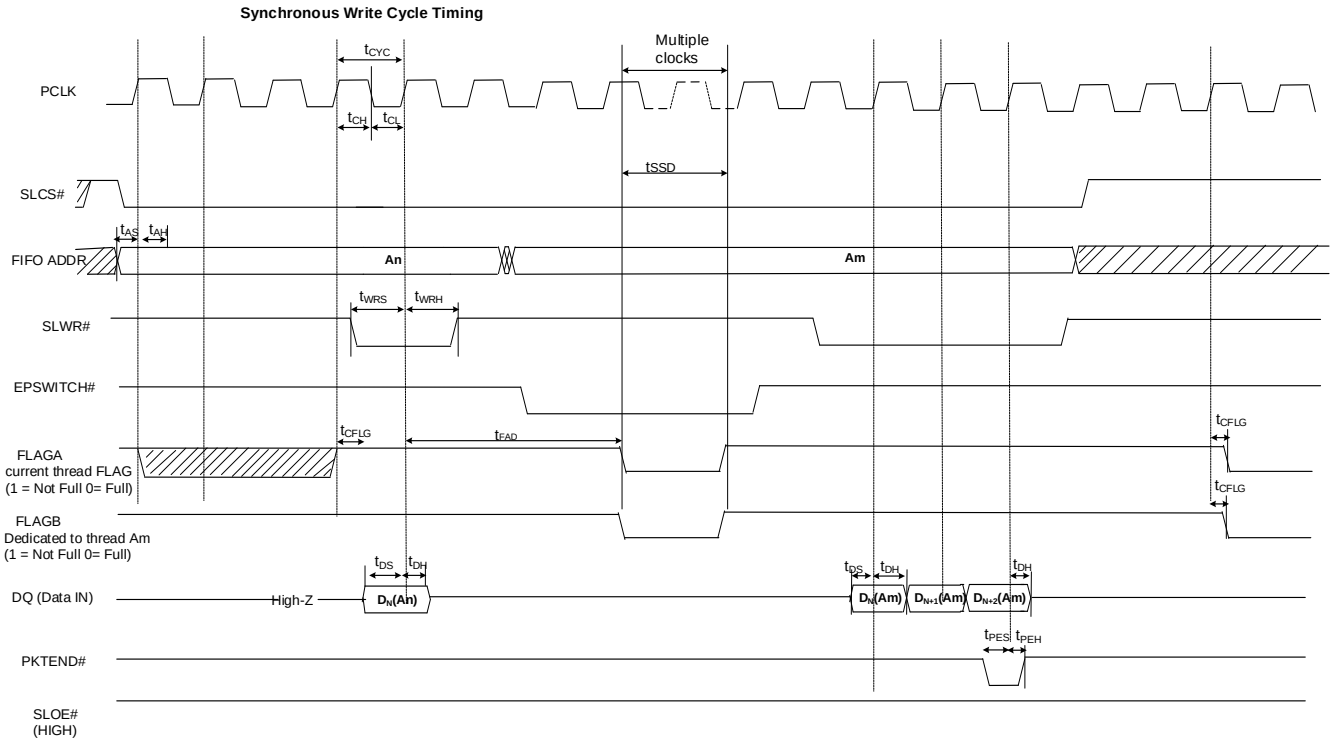
A burst write follows the same sequence of events. Note that for the burst mode, SLWR# and SLCS# remain asserted for the entire duration of the burst write. In the burst write mode, after SLWR# is asserted, the value on the data bus is written into the FIFO on every rising edge of PCLK. The FIFO pointer is updated on each rising edge of PCLK.

Short packet: 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 (that is, along with the assertion of SLWR# with the last word of the packet on the data bus). SLCS# must also be asserted.

The FIFOADDR lines must be held stable 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 has been sourced from USB. The external master must monitor the empty flag to determine when all the data has been read.

Figure 9. Synchronous Slave FIFO Write Sequence

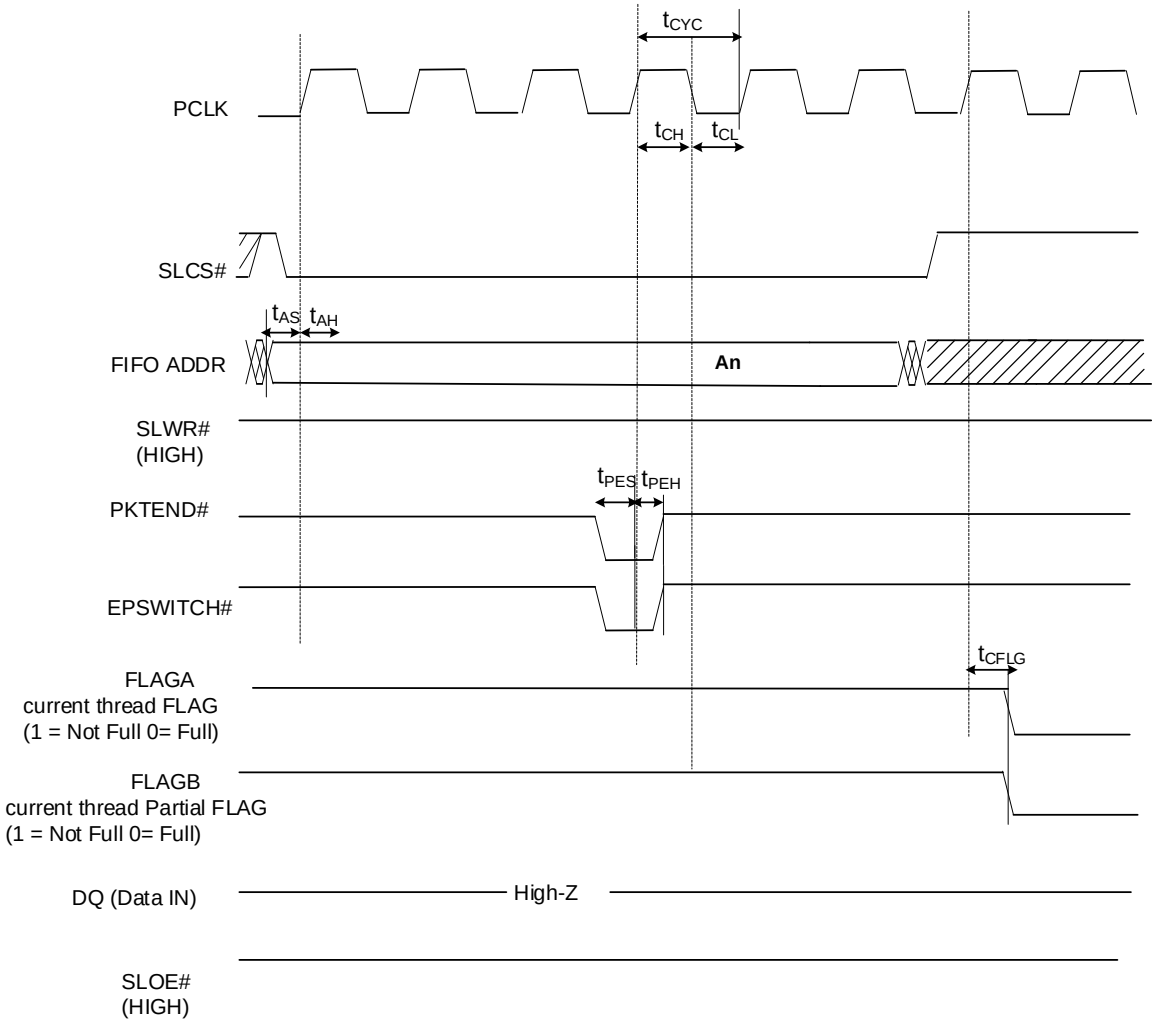


Note Refer to Table 3 for the values of the timing parameters.

Zero-Length Packet: 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, but 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.

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

Figure 10. Synchronous Slave FIFO ZLP Protocol



Note Refer to Table 3 for the values of the timing parameters.

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	–	8	ns
t_{DS}	Data input setup time	2	–	ns

Parameter	Description	Min	Max	Unit
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 FLAGS 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
t_{SSD}	Socket switching delay	2	68	Clock cycles
t_{ACCD}	Latency from SLRD# to Data	2	2	Clock cycles
t_{FAD}	Latency from SLWR# to FLAG	3	3	Clock cycles
Note Three-cycle latency from ADDR to DATA				

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 internal FIFO buffers. This application note described the synchronous Slave FIFO interface in detail. It also explained the different flag configurations available, along with how the GPIF II Designer tool can be used to configure flags.

Appendix

This section documents the average switching delay when switching between two sockets within the same thread at 100 MHz in Synchronous Slave FIFO mode. The delay is measured from the time EPSWITCH# is asserted by the Master to the time FLAG A is asserted by the FX3 Slave. In the test setup used for measurement, the DMA adapter was configured to give the highest priority to socket switching over other socket activities.

Table 4. Average Switching Delay Between Two Sockets Within the Same Thread at 100 MHz in Synchronous Slave FIFO Mode

From/To (sck)	AVERAGE Latency (ns)	AVG Latency (Clk Cycles)	Upper Bound (Clk Cycles)
CONS to CONS	480	48	68
PROD to CONS	451	45	50
PROD to PROD	237	24	29
CONS to PROD	286	29	54

Note These measurements were made in the synchronous Slave FIFO mode with an external (input to FX3) interface clock. In such a scenario, GPIF II runs at the interface clock frequency and the value set for clock frequency in *CyU3PPibInit* API is don't care. In the Asynchronous Slave FIFO mode, when there is no interface clock, GPIF II runs at the clock frequency specified in *CyU3PPibInit* API. Set this value to at least twice the frequency of the Master interface for better sampling.

Document History

Document Title: AN68829 – 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
*B	4426753	RSKV	07/04/2014	Added a section on Difference between Slave FIFO with Two and Five Address Lines Added a section to describe Threads and Sockets Added Asynchronous Slave FIFO Read Sequence Added Asynchronous Slave FIFO Write Sequence Added a table to list Asynchronous Slave FIFO Timing Parameters Added Synchronous Slave FIFO Read Sequence Added Synchronous Slave FIFO Write Sequence
*C	4708189	AMDK	04/10/2015	Added Appendix- added info on socket switching delay within same thread Updated figures 8 and 9 Updated Table 3 Added reference to TRM chapters on DMA and GPIF Added reference to 5-bit Slave FIFO example firmware available in FX3 SDK Updated template Sunset review
*D	5687845	AESATMP7	04/19/2017	Updated Cypress Logo and Copyright.

Worldwide Sales and Design Support

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

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/go/support

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