

# Digilent Synchronous Parallel Interface (DSTM)

Revision: September 2, 2010



1300 NE Henley Court, Suite 3  
Pullman, WA 99163  
(509) 334 6306 Voice | (509) 334 6300 Fax

## Introduction

This document describes the operation and specifications of the Digilent Synchronous Parallel Interface (DSTM). This interface is implemented in various Digilent products to provide a communication and data transfer interface between a personal computer and a Digilent programmable logic system board.

The Digilent Adept 2 SDK contains the DSTM API, which provides the ability to perform Digilent Synchronous Parallel Interface data transfer operations between a host PC and a supported Digilent system board. The data transfer operations supported by Adept Suite DLLs require that certain logic be implemented in the FPGA/CPLD on the Digilent system board to provide the peripheral side of the interface. This document describes the requirements for this implementation.

## Functional Description

The Digilent Synchronous Parallel Interface uses an 8-bit bidirectional parallel data bus and nine handshaking lines to control the data transfer. Data transfer is synchronous; it operates based on a clock generated by the device's USB microcontroller. The data transfer speed that can be achieved depends on the particular communications subsystem and the implemented gate array logic.

The term "host" refers to the host PC running the Adept 2 application. The term "device" refers to the Digilent system board and its gate array logic. "Upload" refers to a device-to-host transfer and "download" refers to a host-to-device transfer.

The Digilent Synchronous Parallel Interface uses the Cypress USB controller's slave FIFO mode to allow for the highest possible transfer rates. The slave FIFO mode is configured for synchronous operation. Two different FIFOs are used for upload and download. These FIFOs are quad buffered where the buffer size depends on the USB connection used: 64 bytes for USB 1.1 and 512 bytes for USB2. With a USB2 connection, both FIFOs are 2k bytes large. The upload FIFO will automatically commit the filled buffers to the USB domain, allowing the PC to read. The device gate array logic must implement a controller to read from and write to these FIFOs.

There are two distinct API calls used to perform DSTM transfers. Each is functionally different in both the host Adept software, the Digilent USB controller, and the gate array logic of the device.

DstmIO is monitored by the USB controller's firmware. The firmware sets the upload flag according to the requested number of upload data bytes and commits the last bytes of the upload transfer. The flag will be low while the requested number of bytes is not written to the upload FIFO. The gate array logic can rely on the upload flag to not write more data than was requested in the DSTM API. This interaction with the firmware introduces 4+ us delays after each 512 bytes (64 bytes for USB 1.1) written to the upload FIFO and the download speed is also decreased.

DstmIOEx offers faster upload transfer rates with more complex gate array logic. This API call requires that the gate array logic write the same number of bytes to the IN FIFO as specified in the API's *cbIn* parameter and may need to insert PKTEND at the end of the transfer. When the device sends more data than requested, the host will generate an error. When the IN (upload) length is not a multiple of 512 (USB2) or 64 (USB 1.1) and the peripheral does not assert the PKTEND signal, then the remaining bytes will not be read by the host.

The order of events when calling DSTM APIs is as follows:

- The EPPEN signal is set low to disable the EPP interface.
- The necessary configurations are applied and the USB controller is put in slave FIFO mode.
- The STMEN signal is set high to signal the enabling of the DSTM interface.
- The gate array logic must select between the download and upload FIFOs when performing read, write, or PKTEND operations, using the FIFOADR[1:0] signals:
  - “00” for download transfer
  - “10” for upload transfer
- Data can be read from the download FIFO when the FLAGA signal (empty flag of the download FIFO) is low. The SLOE signal enables the USB FIFO to drive the data bus. On the rising edge of the IFCLK while the SLRD signal is low the USB FIFO will increment the FIFO pointer and output the next data byte.
- Data can be written to the upload FIFO when the FLAGB signal (full flag of the upload FIFO) is low. On the rising edge of IFCLK while the SLWR signal is low the USB FIFO will write the data from the data bus to the FIFO.
- The PKTEND signal should be activated for one IFCLK cycle to commit the last written bytes in the buffer so the PC can read out.

## Signal Descriptions

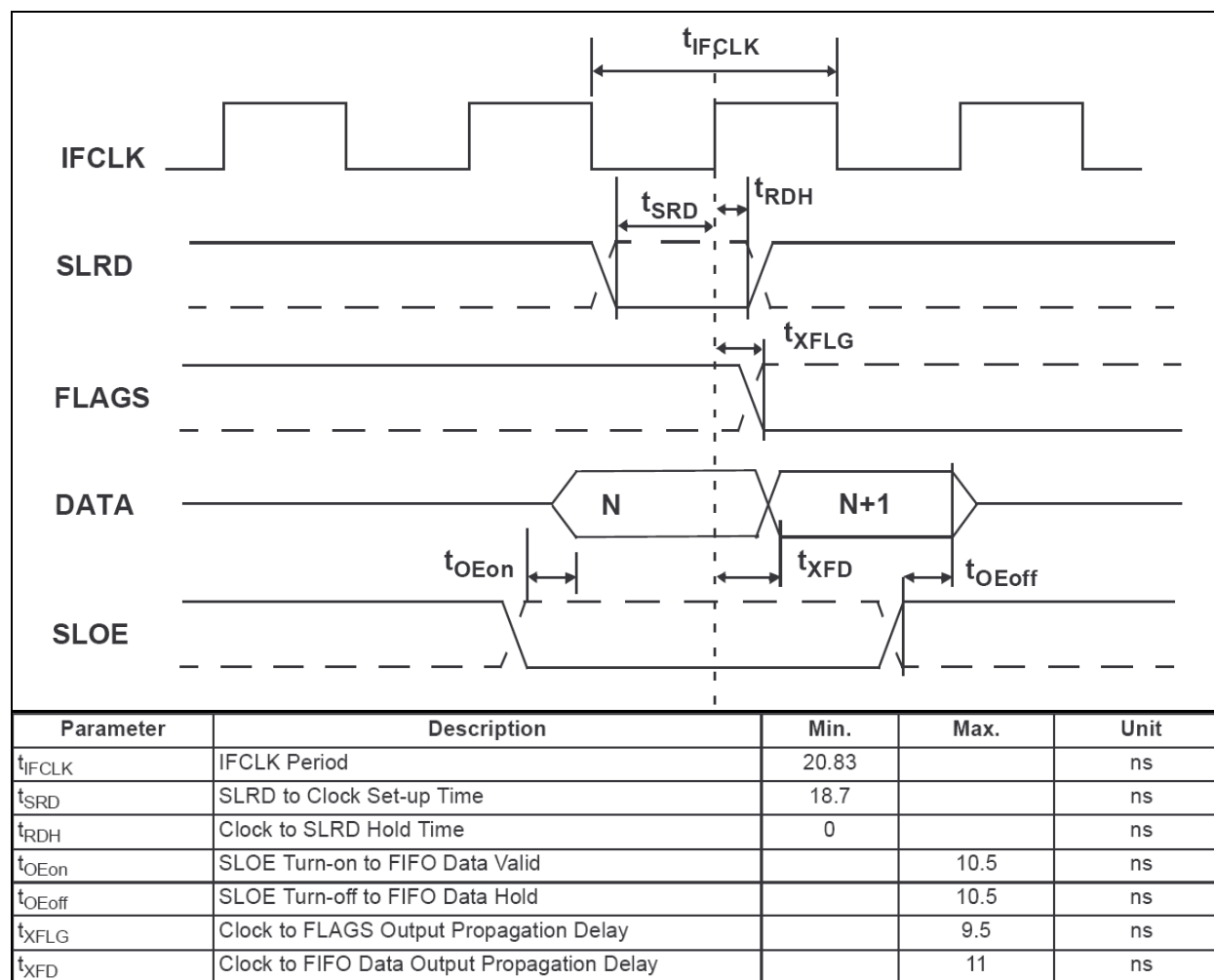
In the following description, signals described as being sourced by the “host” are generated by the Digilent communication interface and are inputs to the logic in the FPGA/CPLD on the Digilent programmable logic system board. The term “peripheral” refers to logic implemented in the FPGA/CPLD on the system board. Signals sourced by the peripheral are outputs from the logic implemented in the FPGA/CPLD.

The following signals make up the interface:

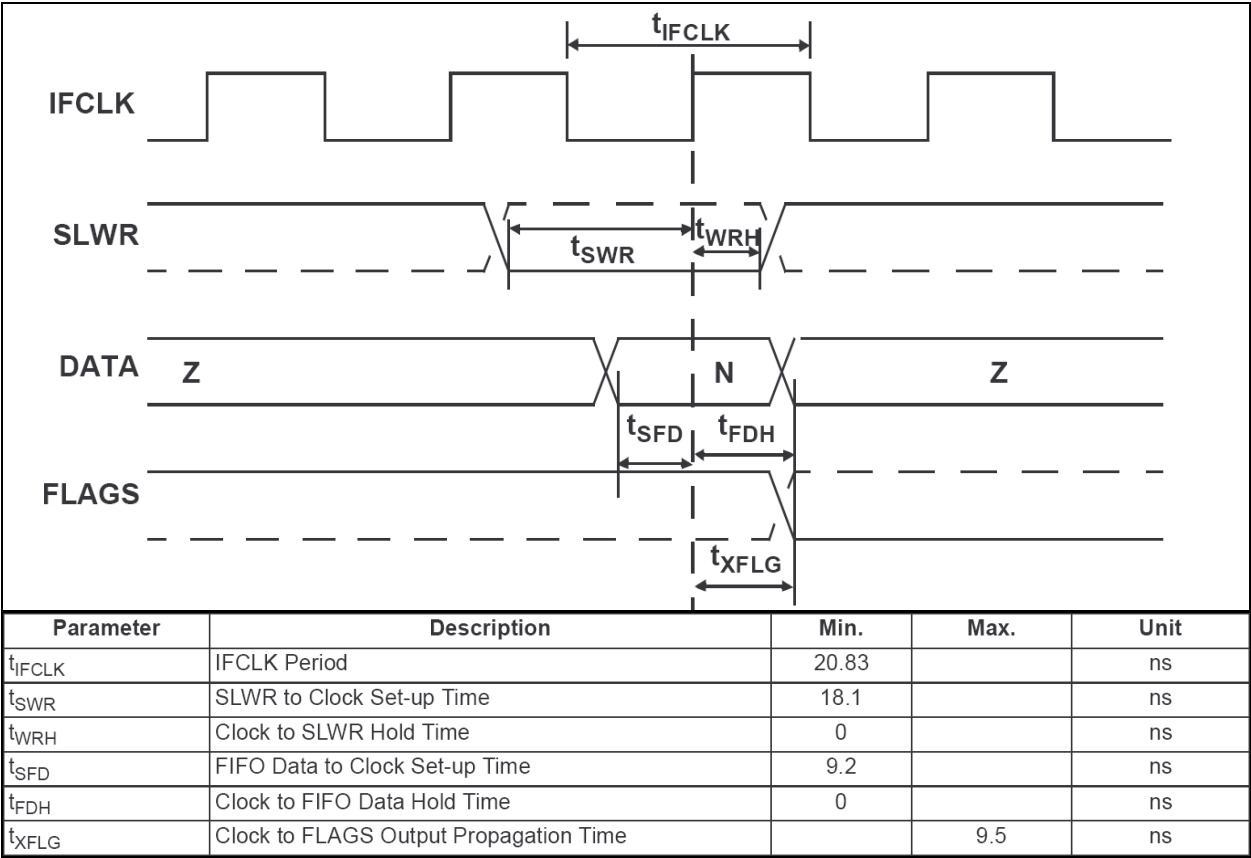
Name	Source	Description
IFCLK	host	48MHz interface clock. The synchronous FIFOs operate on the rising edge of this clock. The peripheral should use this clock to generate control signals and sample the flags.
SLCS/STMEN	host	Stream enable. Active high enable signal for the stream data transfer.
DB[7:0]	bidir	Data bus. The host is the source during read cycles and the peripheral is the source during write cycles.
FLAGA/EPPASTB	host	FIFO OUT flag. Active high FIFO flag that shows the empty state of the OUT (download) FIFO. While this flag is active the peripheral should not read from the OUT FIFO.
FLAGB/EPPDSTB	host	FIFO IN flag. Active high FIFO flag that shows the full state of the IN (upload) FIFO. While this flag is active the peripheral should not write to the IN FIFO.
FIFOADR[1:0]	peripheral	FIFO address. Selects which of the FIFOs is connected to the DB. <ul style="list-style-type: none"> <li>• “00” selects the OUT (download) FIFO</li> <li>• “10” selects the IN (upload) FIFO</li> </ul>
SLRD/EPPWAIT	peripheral	FIFO read next. Active low synchronous signal used to read from the OUT FIFO. The FIFO pointer is incremented on each rising edge of IFCLK while SLRD is asserted.
SLWR	peripheral	FIFO write. Active low synchronous signal used to write to the IN FIFO. Data on the DB is written to the FIFO and the FIFO pointer is incremented on each rising edge of IFCLK while SLWR is asserted.
SLOE	peripheral	FIFO output enable. Active low output enable for the FIFO. When SLOE is asserted, the DB is driven as an output and contains the data that the FIFO pointer is currently pointing to.
PKTEND	peripheral	IN transfer end. Active low signal for the end of the IN transfer. When asserting this signal, FIFOADR must select the IN FIFO.

## Timing Diagrams

## FIFO Read

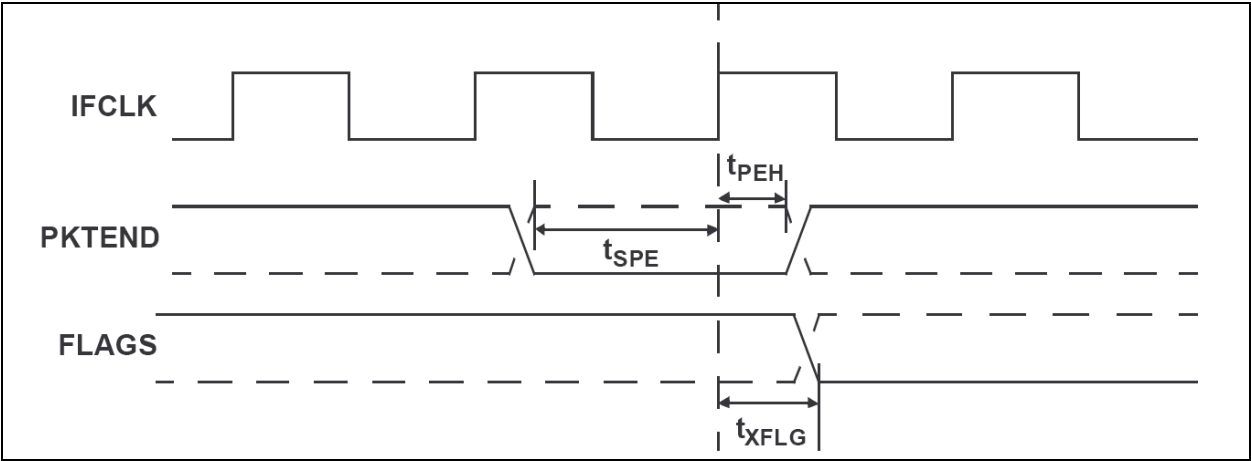


FIFO Write

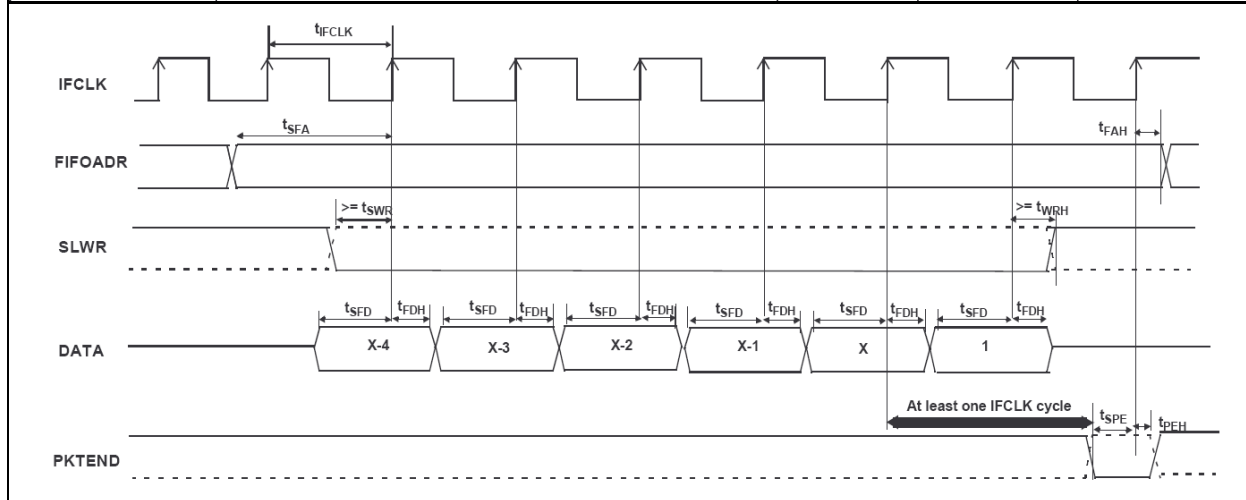


FIFO PKTEND Strobe

The PKEND signal is used with the DstmlIOEx API when the upload data length is not a multiple of 512 (USB2) or 64 (USB 1.1) to commit the remaining bytes for the host.



Parameter	Description	Min.	Max.	Unit
$t_{IFCLK}$	IFCLK Period	20.83		ns
$t_{SPE}$	PKTEND to Clock Set-up Time	14.6		ns
$t_{PEH}$	Clock to PKTEND Hold Time	0		ns
$t_{XFLG}$	Clock to FLAGS Output Propagation Delay		9.5	ns

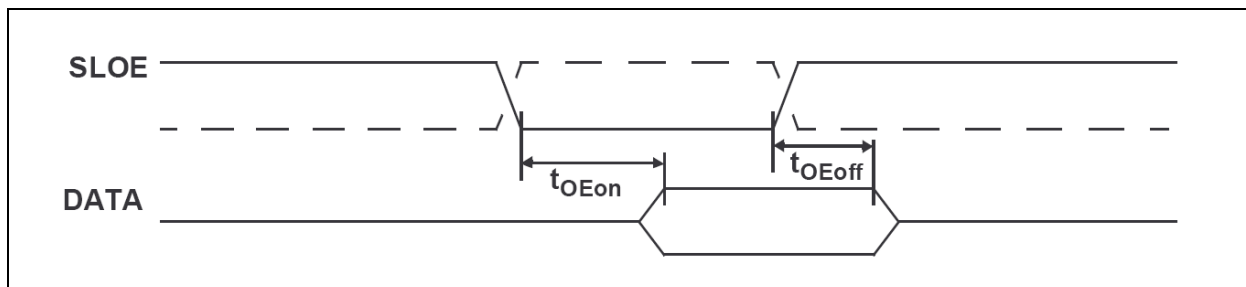


There is no specific timing requirement that needs to be met for asserting the PKTEND pin with regards to asserting SLWR. PKTEND can be asserted with the last data value clocked into the FIFOs or thereafter. The only consideration is that the set-up time  $t_{SPE}$  and the hold time  $t_{PEH}$  must be met.

Although there are no specific timing requirements for the PKTEND assertion, there is a specific corner case condition that needs attention while using PKTEND to commit a one-byte packet. There is an additional timing requirement that needs to be met when the FIFO is configured to operate auto mode and you want to send two packets back-to-back: a full packet committed automatically followed by a short one-byte packet committed manually using the PKTEND pin. In this scenario, you must assert PKTEND at least one clock cycle after the rising edge that caused the last byte to be clocked into the previous auto-committed packet.

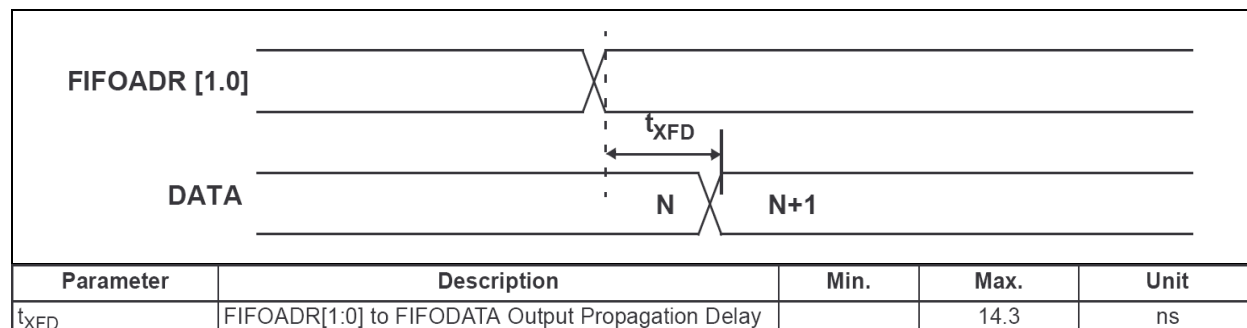
The figure above shows a scenario where two packets are being committed. The first packet is committed automatically when the number of bytes in the FIFO reaches X and the second one-byte short packet is committed manually using PKTEND. Note that there is at least one IFCLK cycle timing between the assertion of PKTEND and clocking of the last byte of the previous packet (causing the packet to be committed automatically). Failing to adhere to this timing will result in the FX2 failing to send the one-byte short packet.

### SLOE – FIFO Output Enable

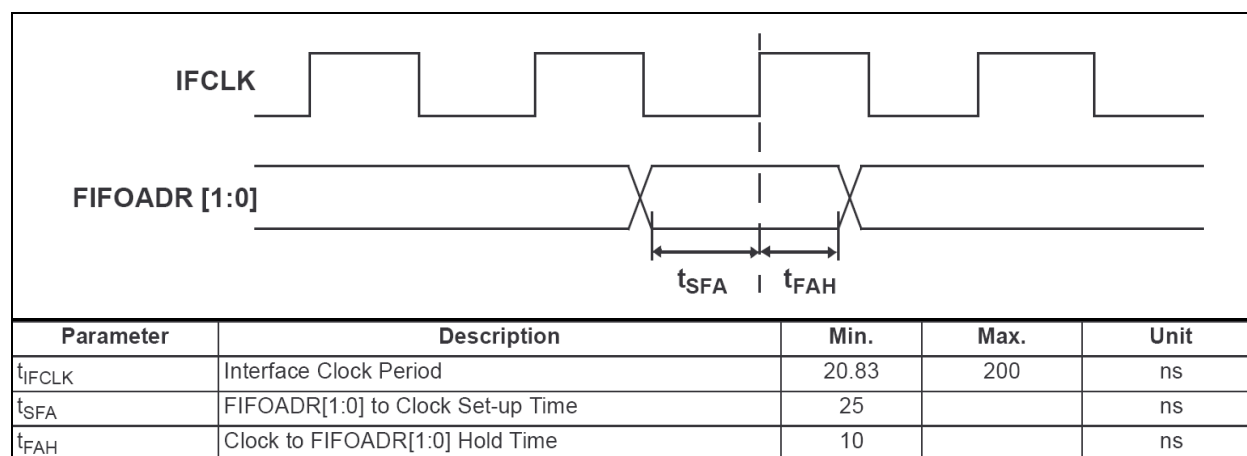


Parameter	Description	Min.	Max.	Unit
$t_{OEon}$	SLOE Assert to FIFO DATA Output		10.5	ns
$t_{OEoff}$	SLOE Deassert to FIFO DATA Hold		10.5	ns

### FIFOADR to DB – Address to Data

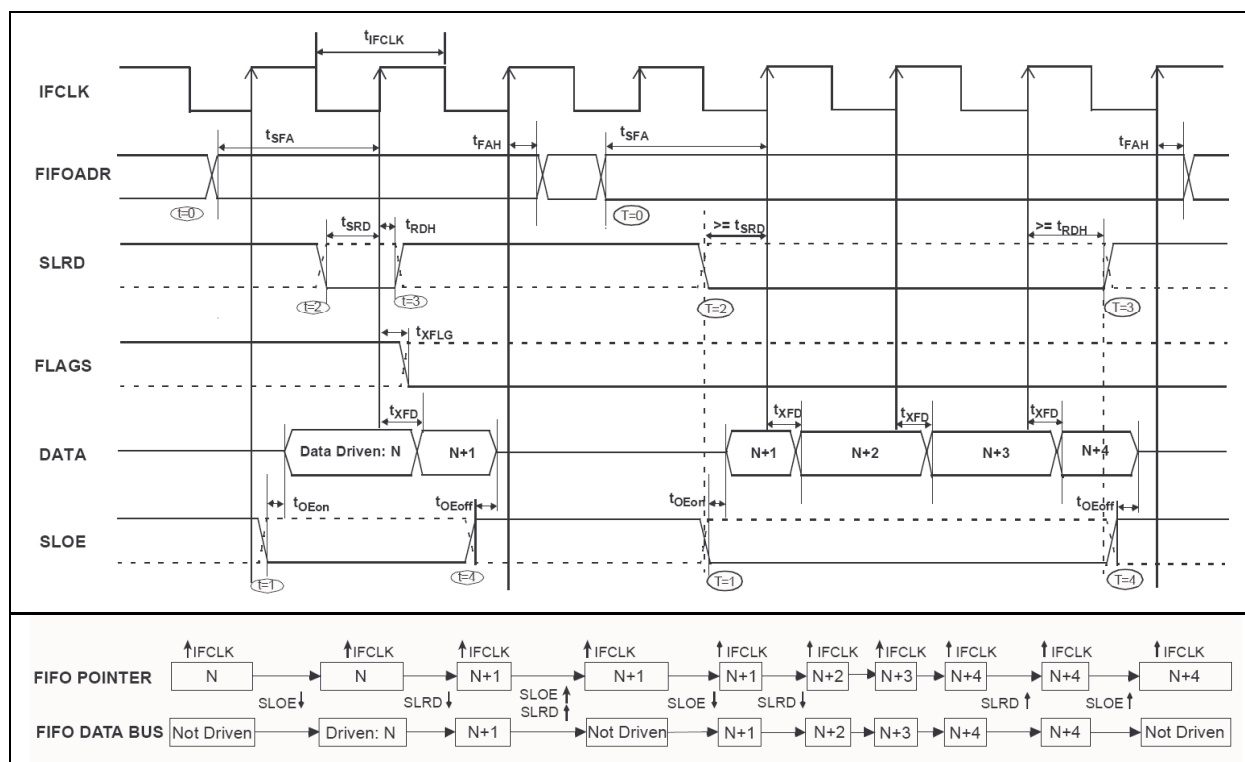


### FIFOADR – FIFO Address



## Sequence Diagrams

### Single and Burst Read Example



The diagram shows the timing relationship of the FIFO signals during a synchronous FIFO read using IFCLK as the synchronizing clock. The diagram illustrates a single read followed by a burst read.

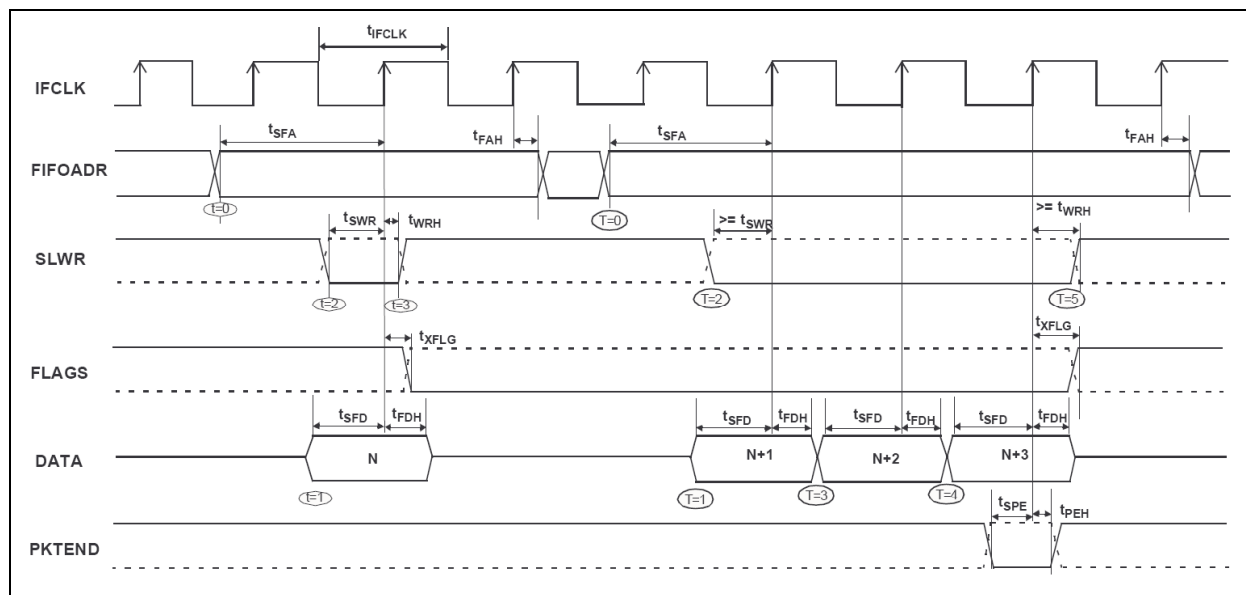
- At  $T = 0$ , the FIFO address is stable ( $t_{SFA}$  has a minimum of 25 ns). This means that when IFCLK is running at 48MHz, the FIFO address set-up time is more than one IFCLK cycle.
- At  $T = 1$ , SLOE is asserted. SLOE is an output-enable only, whose sole function is to drive the data bus. The data that is driven on the bus is the data that the internal FIFO pointer is currently pointing to. In this example, it is the first data value in the FIFO. Note that the data is prefetched and is driven on the bus when SLOE is asserted.
- At  $T = 2$ , SLRD is asserted. SLRD must meet the set-up time of  $t_{SRD}$  (time from asserting the SLRD signal to the rising edge of the IFCLK) and maintain a minimum hold time of  $t_{RDH}$  (time from the IFCLK edge to the deassertion of the SLRD signal).
- The FIFO pointer is updated on the rising edge of the IFCLK while SLRD is asserted. This starts the propagation of data from the newly addressed location to the data bus. After a propagation delay of  $t_{XFD}$  (measured from the rising edge of IFCLK) the new data value is present. N is the first data value read from the FIFO. In order to have data on the FIFO data bus, SLOE must also be asserted.

The same sequence of events is shown for a burst read and is marked with time indicators  $T = 0$  through  $T = 5$ . For the burst mode, SLRD and SLOE are left asserted during the entire duration of the read. In the burst read mode, when SLOE is asserted, data indexed by the FIFO pointer is on the data bus. During the first read cycle, on the rising edge of the clock, the FIFO pointer is updated and



increments to point to address N+1. For each subsequent rising edge of IFCLK, while the SLWR is asserted, the FIFO pointer is incremented and the next data value is placed on the data bus.

### Single and Burst Synchronous Write



This diagram shows the timing relationship of the slave FIFO signals during a synchronous write using IFCLK as the synchronizing clock. The diagram illustrates a single write followed by burst write of three bytes and committing all four bytes as a short packet using the PKTEND pin.

- At  $T = 0$ , the FIFO address is stable. This means that when IFCLK is running at 48MHz, the FIFO address set-up time is more than one IFCLK cycle.
- At  $T = 1$ , the external master/peripheral outputs the data value onto the data bus with a minimum set up time of  $t_{SFD}$  before the rising edge of IFCLK.
- At  $T = 2$ , SLWR is asserted. The SLWR must meet the set-up time of  $t_{SWR}$  (time from asserting the SLWR signal to the rising edge of IFCLK) and maintain a minimum hold time of  $t_{WRH}$  (time from the IFCLK edge to the deassertion of the SLWR signal).
- While the SLWR is asserted, data is written to the FIFO and on the rising edge of the IFCLK, the FIFO pointer is incremented. The FIFO flag will also be updated after a delay of  $t_{XFLG}$  from the rising edge of the clock.

The same sequence of events is also shown for a burst write and is marked with time indicators  $T = 0$  through  $T = 5$ . For the burst mode, SLWR is left asserted for the entire duration of writing all the required data values. In burst write mode, once the SLWR is asserted, the data on the FIFO data bus is written to the FIFO on every rising edge of IFCLK. The FIFO pointer is updated on each rising edge of IFCLK.

In the diagram, once the four bytes are written to the FIFO, SLWR is deasserted. The short 4-byte packet can be committed to the host by asserting the PKTEND signal. There is no specific timing requirement that needs to be met for asserting PKTEND signal with regards to asserting the SLWR signal. PKTEND can be asserted with the last data value or thereafter. The only requirement is that the set-up time  $t_{SPE}$  and the hold time  $t_{PEH}$  must be met. In this example, the number of data values committed includes the last value written to the FIFO. Both the data value and the PKTEND signal are

clocked on the same rising edge of IFCLK. PKTEND can also be asserted in subsequent clock cycles. The FIFOADDR lines should be held constant during the PKTEND assertion.

Although there are no specific timing requirements for the PKTEND assertion, there is a specific corner case condition that needs attention while using the PKTEND to commit a one-byte packet. Additional timing requirements exist when using the PKTEND pin. In this case, the external master must assert the PKTEND pin at least one clock cycle after the rising edge while SLWR is active.

### **Design Tips**

The data transfer API calls on USB2 devices have at least 300-600us latency. Therefore, using fewer calls and transferring more data, each call will produce higher transfer rates.

The DstmIOEx API offers higher transfer rates, and can be implemented as follows:

- The upload data can be padded to be a multiple of 512 bytes (64 bytes for USB 1.1) and the PKTEND signal insertion is not required.
- The gate array logic does not need to include an upload transfer counter. This way, it will write more than was requested but the host application can ignore the error.