LogiCORE IP Video In to AXI4-Stream v3.0

Product Guide

Vivado Design Suite

PG043 April 2, 2014

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx LogiCORE™ IP Video In to AXI4-Stream core is designed to interface from a video source (clocked parallel video data with synchronization signals - active video with either syncs, blanks or both) to the AXI4-Stream Video Protocol Interface.This core works with the timing detector portion of the Xilinx Video Timing Controller (VTC) core. This core provides a bridge between a video input and video processing cores with AXI4-Stream Video Protocol interfaces.

Features

- Video (clocked parallel video data with synchronization signals - active video with either syncs, blanks or both) input
- AXI4-Stream master interface for output
- Interface to Xilinx Video Timing Controller core for video timing detection
- Handles asynchronous clock boundary crossing between video clock domain and AXI4-Stream clock domain
- Selectable FIFO depth from 64–8192 locations
- Selectable input data width of 8–256 bits
- Support for interlaced operation

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4-Stream(2)
Resources	See Table 2-1 to Table 2-3.
Provided with Core	
Documentation	Product Guide
Design Files	Verilog Source Code
Example Design	Provided Separately (3) See XAPP521 [Ref 3] and XAPP721 [Ref 4]
Test Bench	Verilog
Constraints File	XDC
Simulation Models	Verilog Source Code
Supported Software Drivers	N/A
Tested Design Flows	
Design Entry Tools	Vivado® Design Suite IP Integrator
Simulation(4)	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis Tools	Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of (UG761) *AXI Reference Guide* [Ref 5].
3. Example designs are provided in FPGA device-specific application notes
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Many Xilinx® video processing cores utilize the AXI-4 Stream video protocol to transfer video between cores. Between systems, video is commonly transmitted with explicit blanking and sync signals for horizontal and vertical timing, and a data valid signal. Digital visual interface (DVI) is an example of such a transmission mode. The Video In to AXI4-Stream core converts incoming video with explicit sync and timing to the AXI4-Stream Video protocolto interface with Xilinx video processing cores that use this protocol.

The Video In to AXI-4 Stream core accepts video inputs. For this document, video is defined as parallel video data with a pixel clock and one of the following sets of timing signals:

- Vsync, Hsync, and Data Valid
- Vblank, Hblank, and Data Valid
- Vsync, Hsync, Vbank, Hblank, and Data Valid

Any of these sets of signals is sufficient for the operation of the Video In to AXI4-Stream core. The particular choice is important to the Video Timing Controller (VTC) detector, so you should specify the set of timing signals when you generate the VTC core. The output side of the core is an AXI4-Stream interface in master mode. This interface consists of parallel video data, `tdata`, handshaking signals `tvalid` and `tready`, and two flags, `tlast` and `tuser`, which identify certain pixels in the video stream. The flag `tlast` designates the last valid pixel of each line, and is also known as end of line (`EOL`). The flag `tuser` designates the first valid pixel of a frame, and is known as start of frame (`SOF`). These two flags are necessary to identify pixel locations on the AXI4 stream bus because there are no sync or blank signals. Only active pixels are carried on the bus. The *Video IP: AXI Feature Adoption* section of the UG761 AXI Reference Guide *[Ref 5]* describes the video over AXI4 Stream Video protocol in detail.

A block diagram of a Video In to AXI4-Stream core with a video timing generator is shown in Figure 1-1.

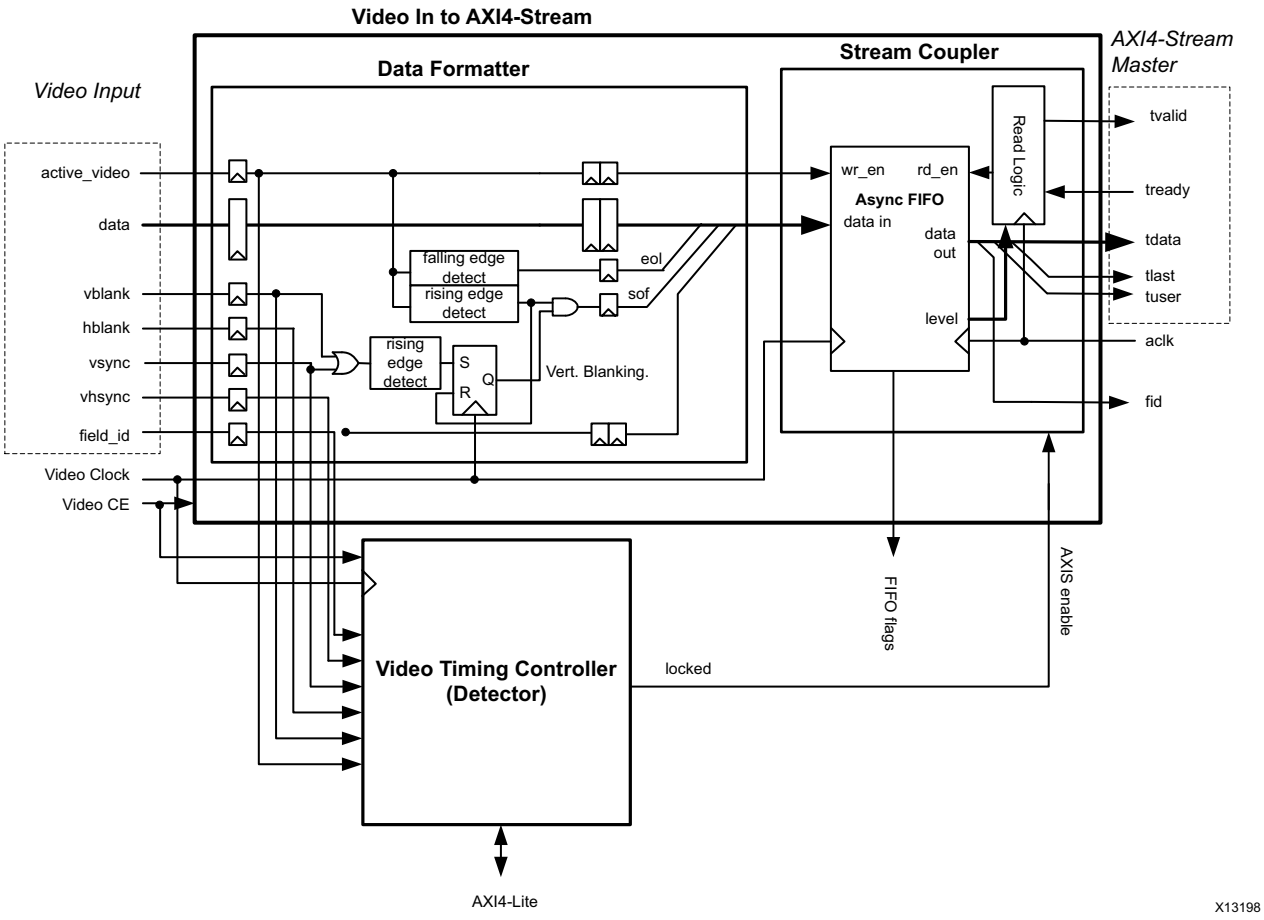

Figure 1-1: **Block Diagram of Video In to AXI4-Stream Core with the Video Timing Controller**

The core is designed to be used in parallel with the detector functionality of the VTC. The video timing detector detects the line standard of the incoming video, and makes the detected timing values, such as the number of active pixels per line and the number of active lines available to video processing cores downstream of the Video In to AXI4-Stream core via an AXI4-Lite interface. It is recommended to connect the "locked" status output of the video timing detector to the `axis_enable` input of the Video In to AXI4-Stream core in order to inhibit the AXI4-Stream bus when the video input is missing or unstable. The detector locked indicator from the Video Timing Controller is bit 8 of the INTC_if register.

Feature Summary

The Video In to AXI4-Stream core converts a video input, consisting of parallel video data, video syncs, blanks and data valid, to an AXI4-Stream master bus that follows the AXI4-Stream Video protocol. The core provides a pass-thru of all timing signals for the

Xilinx video timing controller, although the signals for the Video timing Controller are not required to pass through the Video In to AXI4 Stream core.

The core handles the asynchronous clock boundary crossing between the video clock domain and the AXI4-Stream clock domain. The data width is selectable from 8 to 256 depending on the number of components required for the video format, the number of bits per component, and the number of pixels per clock. Interlaced operation is supported. There is an input FIFO with selectable depth from 32 to 8192 locations.

Applications

- Video input to AXI4-Stream Video Protocol interface for parallel, clocked video sources:
  - DVI
  - HDMI
  - Image Sensors
  - Other clocked, parallel video sources

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The Video In to AXI4-Stream core is compliant with the AXI4-Stream Video Protocol. Refer to the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 5] for additional information.

Performance

The following sections detail the performance characteristics of the Video In to AXI4-Stream core.

Maximum Frequencies

This section contains typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. Refer to Table 2-1 and Table 2-3 for device-specific information.

Latency

When the downstream processing block on the AXI4-Stream bus can take data at the pixel rate or faster, the typical latency through the Video In to AXI4-Stream core is 6 cycles of `vid_io_in_clk` + 3 cycles of `aclk` .

If the downstream block takes pixels at a slower rate, the FIFO is used to balance the mismatch in the input and output rates over the course of lines and frames. This storage of pixels in the FIFO adds to the latency and varies according to the data flow in and out of the core.

Throughput

The average data rates of active pixels on the AXI4-Stream interface matches the average rate of active pixels in on the Video bus. However, the clock rates of the input and output need not match. Because the AXI4-Stream bus does not carry blank pixels, the clock rate can be lower than the video clock rate and still have sufficient bandwidth to meet the average rate requirement. Additional FIFO depth is required to smooth the mismatch in instantaneous rates. Both the input video pixel clock (`Fvclk`) and the rate of the AXI4-Stream Clock (`Faclk`) are limited by the overall `Fmax`.

If `Faclk` is equal to or greater than `Fvclk`, only the minimum buffer size (32 locations) is required. This assumes that the cores connected downstream of the Video In to AXI4-Stream core can sink data at the full video rate. For example, the downstream core can accept data in a virtually continuous stream with gaps occurring only following `EOL`, and each line consecutively with line gaps only preceding `SOF`. In this scenario, the FIFO empties after the `EOL` on each line.

If `Faclk` is less than `Fvclk`, additional buffering is required. The FIFO must be large enough to handle the differential in the rate that pixels are coming in on the video clock, and the slower rate that they can go out on the AXI4-Stream bus using `aclk`. For `aclk` frequencies above the line average but below that of `vclk`, the input FIFO depth must be:

FIFO depth min =32 + Active Pixels * Fvclk/Faclk

If the downstream processing core accepts data at a lower rate than the `aclk`, additional buffering is required in an amount sufficient to prevent the FIFO from overflowing during frame transmission.

Resource Utilization

The information in Table 2-1 through Table 2-3 is a guide to the resource utilization and maximum clock frequency of the Video In to AXI4-Stream core for 7 series FPGA families. (Zynq™-7000 utilization and Fmax is similar to Artix-7 devices.) UltraScale™ results are expected to be similar to 7 series results. This core does not use any dedicated I/O or CLK resources. The design was tested using Vivado Design Suite tools with default tool options for characterization data.

Table 2-1: **Virtex-7 FPGA Performance**

Data Width	FIFO Depth	LUTs	FFs	RAM 36	Fmax (MHz)
8	32	75	118	0	344
24	1024	94	190	1	288
64	8192	177	331	17	242
256	1024	223	1155	7.5	276

Table 2-2: **Kintex-7 FPGA and Zynq-7000 Device with Kintex Based Programmable Logic Performance**

Data Width	FIFO Depth	LUTs	FFs	RAM 36	Fmax (MHz)
8	32	75	118	0	336
24	1024	94	190	1	274
64	8192	177	331	17	242
256	1024	223	1155	7.5	264

Table 2-3: **Artix-7 FPGA and Zynq-7000 Device with Artix Based Programmable Logic Performance**

Data Width	FIFO Depth	LUTs	FFs	RAM 36	Fmax (MHz)
8	32	76	118	0	250
24	1024	94	190	1	204
64	8192	177	331	17	148
256	1024	354	1155	7.5	193

Core Interfaces

Port Descriptions

The Video In to AXI4-Stream core uses industry-standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the Video In to AXI4-Stream core. Not all of the timing signals are required by this core, however it also passes these signals to a Xilinx Video Timing Controller which, depending on its configuration, may require certain signals. Therefore all timing signals are present. For the Video In to AXI4 Stream core, the data valid is always required. Also, either a vertical sync or a vertical blank input is required.

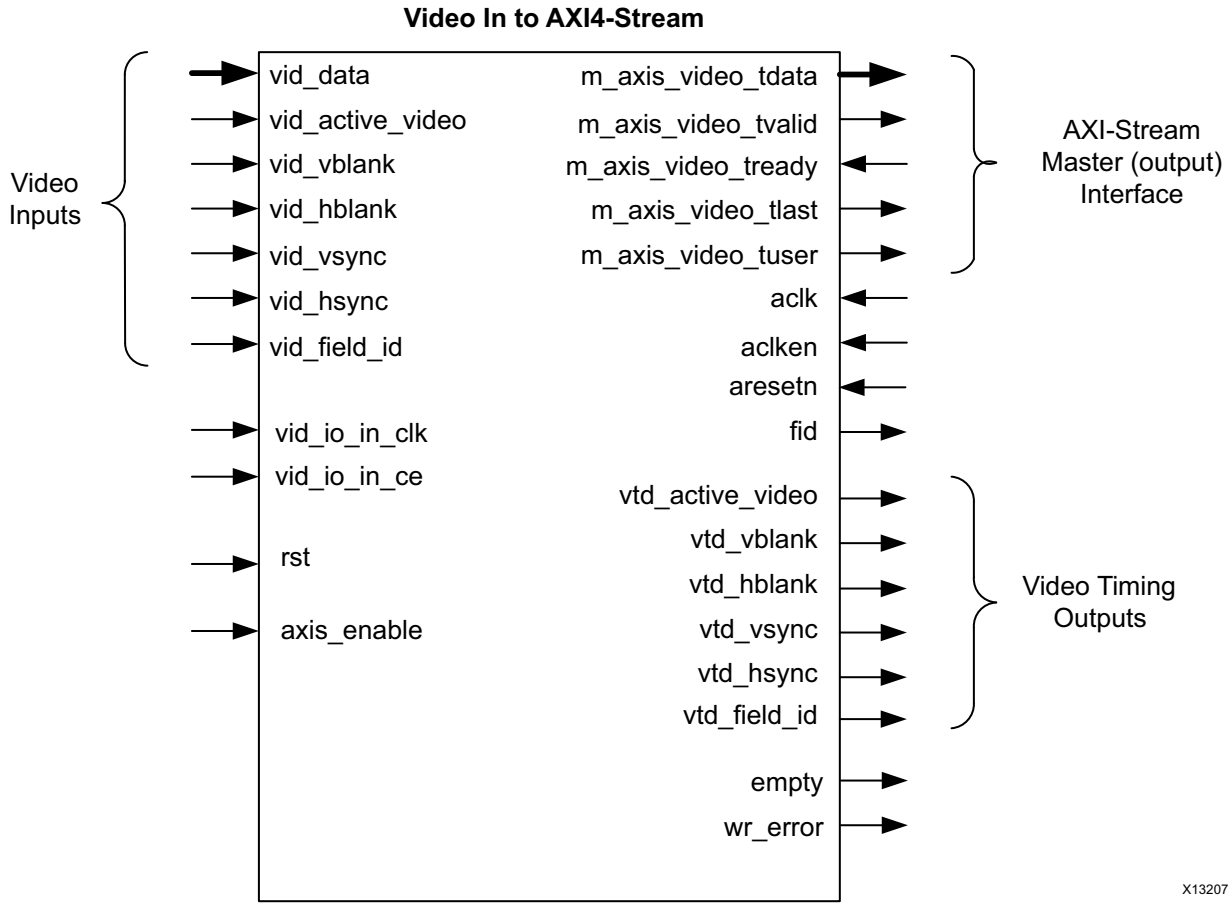

Figure 2-1: **Video In to AXI4-Stream Core Top-Level Signaling Interface**

Common Interface

Table 2-4: **Port Name I/O Width Description**

Signal Name	Direction	Width	Description
rst	In	1	Core reset. Active High
wr_error	Out	1	Active HIGH FIFO write error flag. Synchronous with vid_io_in_clk. 1 = FIFO write was attempted when FIFO was full.
empty	Out	1	Active HIGH FIFO empty flag. Synchronous with aclk. 1 = FIFO read was attempted when FIFO was empty. Due to EOL flushing, this flag will be asserted at the end of every line during normal operation.
axis_enable	In	1	Enable the AXI4-Stream bus. 1 = Enable AXI4-Stream bus to operate beginning at the start of the next frame. 0 = Inhibit AXI4-Stream operation by forcing m_axis_video_tdata LOW.
vid_io_in_ce	In	1	Clock enable for vid_io_in_clock. 1= enable. Tie HIGH if not use.

Table 2-4: **Port Name I/O Width Description *(Cont'd)***

Signal Name	Direction	Width	Description
fid	Out	1	Field ID for AXI4-Stream bus. Used only for interlace. 0= even field, 1= odd field This bit changes coincident with SOF on the AXI4-Stream bus. It should be connected to the field ID bit of the next device downstream that is field-aware or left unconnected if there are no field ID devices downstream.
vid_io_in_clk	In	1	Video input clock

Video Timing Interface

Table 2-5: **Port Name I/O Width Description**

Signal Name	Direction	Width	Description
vtd_vsync	Out	1	Vertical synch video timing signal.
vtd_hsync	Out	1	Horizontal synch video timing signal.
vtd_vblank	Out	1	Vertical blank video timing signal.
vtd_hblank	Out	1	Horizontal blank video timing signal.
vtd_active_video	Out	1	Active video flag. 1 = active video, 0 = blanked video
vtd_field_id	Out	1	VTC field ID. Used only for interlace. 0= even field, 1= odd field.

Video Input Interface

The Video In to AXI4-Stream core receives standard video data using the Video Input interface and transmits data using AXI4-Stream interfaces that implement a video protocol. See the *AXI Reference Guide* (UG761) [Ref 5] for more information.

Table 2-6: **Port Name I/O Width Description**

Signal Name	Direction	Width	Description
vid_active_video	In	1	Video data valid. 1 = active video, 0 = blanked video
vid_vsync	In	1	Vertical synch video timing signal. Active High
vid_hsync	In	1	Horizontal synch video timing signal. Active High
vid_vblank	In	1	Vertical blank video timing signal. Active High
vid_hblank	In	1	Horizontal blank video timing signal. Active High
vid_data	In	8-256	Parallel video input data.
vid_field_id	In	1	Video field. Used only for interlace. 0= even field, 1= odd field. Tie LOW for non-interlace operation.

AXI4-Stream Interface

AXI4-Stream Signal Names and Descriptions

Table 2-7 describes the AXI4-Stream signal names and descriptions.

Table 2-7: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
m_axis_video_tdata	Out	8 to 256	Output Video Data
m_axis_video_tvalid	Out	1	Output Valid
m_axis_video_tready	In	1	Output Ready
m_axis_video_tuser	Out	1	Output Video Start Of Frame
m_axis_video_tlast	Out	1	Output Video End Of Line
ACLK	In	1	AXI4-Stream Clock
ACLKEN	In	1	AXI4-Stream Clock Enable
ARESETn	In	1	AXI4-Stream Active Low synchronous
RESET	In	1	AXI4-Stream Reset. Active Low synchronous

The ACLK, ACLKEN, and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and AXI4-Lite control interfaces in the system.

ACLK

The AXI4-Stream must be synchronous to the clock signal ACLK. AXI4-Stream signals are sampled on the rising edge of ACLK. AXI4-Stream output signal changes occur after the rising edge of ACLK.

ACLKEN

The ACLKEN pin is an active-High, synchronous clock-enable input pertaining the AXI4-Stream interface. Setting ACLKEN Low (deasserted) halts the operation of the AXI4-Stream Bus despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is deasserted, core AXI4-Stream inputs are not sampled, except ARESETn, which supersedes ACLKEN.

ARESETn

The ARESETn pin is an active-low, synchronous reset input. ARESETn supersedes ACLKEN, and when set to 0, the core resets even if ACLKEN is deasserted.

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, if video data widths are not an integer multiple of 8, data must be padded with zeros on the MSB to form an N*8-bit wide vector before connecting to m_axis_video_tdata. Padding does not affect the size of the core.

Similarly, data on the Video in to AXI-4 Stream output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary. Figure 2-2 shows an example of this for 12-bit RGB data for one pixel per clock. For multiple pixels per clock, the pixels are packed together an packed to multiples of 8 bits as necessary. Figure <new figure> shows an example of three pixels per clock with 12-bit per component RGB data. Although this is the expected packing, the core itself does not parse the data. In other words, the AXI4-Stream output will be the video input padded to a multiple of 8 bits.

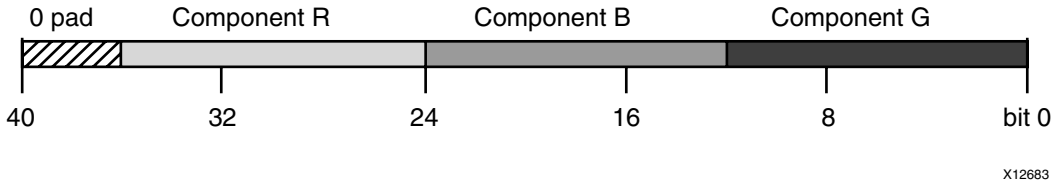

Figure 2-2: **RGB Data Encoding on m_axis_video_tdata**

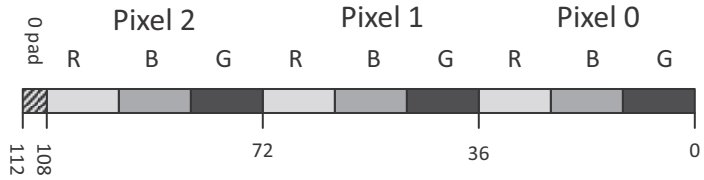

Figure 2-3: **Three Pixels per Clock Format on m_axis_video_tdata.**

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, ACLKEN, and ARESETn are high at the rising edge of ACLK. During valid transfers, DATA only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream Video protocol.

Driving m_axis_video_tready

The m_axis_video_tready signal can be asserted before, during, or after the cycle in which the Video in to AXI4-Stream core asserted m_axis_video_tvalid. The assertion of m_axis_video_tready may be dependent on the value of m_axis_video_tvalid. A slave that can immediately accept data qualified by m_axis_video_tvalid should preassert its m_axis_video_tready signal until data is received. Alternatively,

m_axis_video_tready can be registered and driven the cycle following VALID assertion. It is recommended that the AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

SOF - m_axis_video_tuser

The SOF signal, physically transmitted over the AXI4-Stream tuser0 signal, marks the first pixel of a video frame. The SOF pulse is 1 valid transaction wide, and must coincide with the first pixel of the frame. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The SOF signal may be asserted an arbitrary number of aclk cycles before the first pixel value is presented on tdata, as long as a tvalid is not asserted.

EOL Signal - m_axis_video_tlast

The EOL signal, physically transmitted over the AXI4-Stream tlast signal, marks the last pixel of a line. The EOL pulse is 1 valid transaction wide, and must coincide with the last pixel of a scanline, as seen in Figure 2-4.

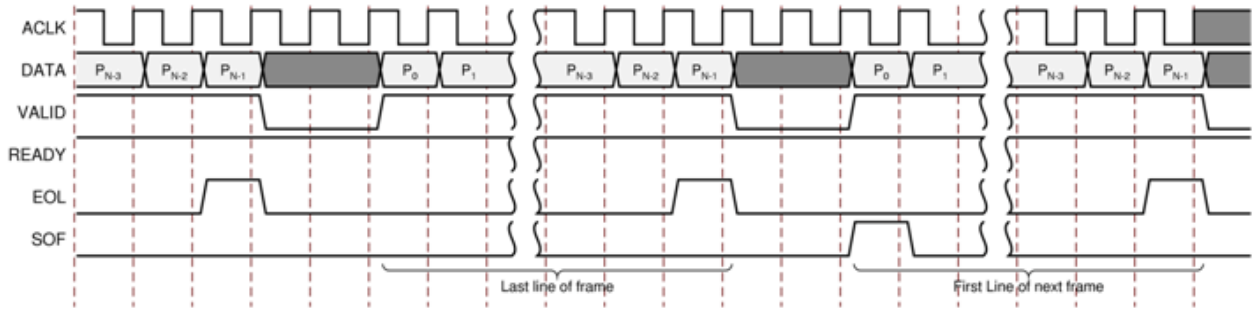

Figure 2-4: **Use of EOL and SOF Signals**

Designing with the Core

General Design Guidelines

The video inputs of the Video In to AXI4 Stream core should be connected to the input video source; for example, a DVI interface chip that produces parallel video data and timing signals. Not all of the timing signals are required by this core. However, the core passes these signals to a Xilinx Video Timing Controller which, depending on its configuration, may require certain timing signals. Use the set of timing signals required by the VTC detector. See the *Video Timing Controller Product Guide* (PG016) [Ref 8] for more details. For the Video In to AXI4 Stream core, the data valid signal is always required. Also, either a vertical sync or a vertical blank input is required.

The main output of the core is a master AXI-4 Stream bus that connects to downstream video processing blocks as shown in Figure 3-1. The master and slave interfaces share a common clock, reset, and clock enable.

As shown in Figure 3-1, the Video In to AXI4 Stream Core is generally used in conjunction with the Video Timing Controller, which detects the video timing parameters used by downstream processing blocks.

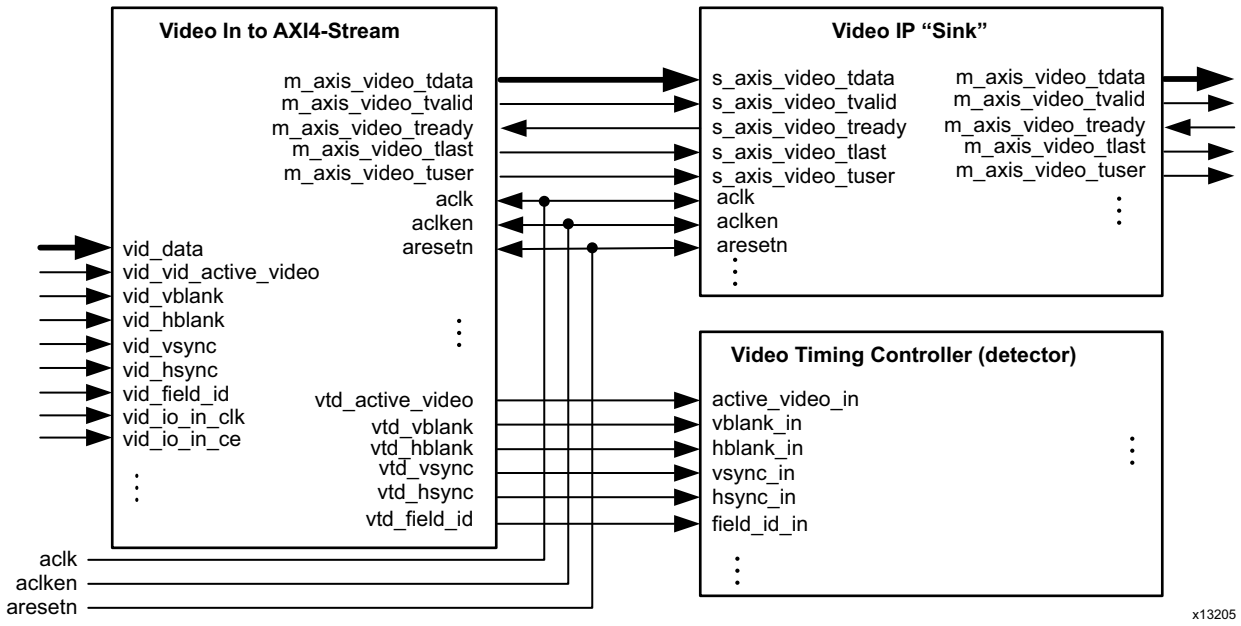

Figure 3-1: **Example of ACLK Routing and AXI4-Stream Interconnect**

Clocking

Two clocks are used in this core.

- Video input pixel clock
- AXI4-Stream clock

The video input clock corresponds to the video line standard used on the input. It is part of the video line standard and is used by both the Video In to AXI4-Stream core and by the corresponding Video Timing Controller core that is used to detect video timing.

The AXI4-Stream clock (`aclk`) is part of the AXI4-Stream bus. To minimize buffering requirements, this clock should be of equal or higher frequency than the video input clock. This clock can be slower than the video input clock, in which case, additional buffering is required to store pixels so that lines can be input at the burst rate of the video clock. This is discussed in the Buffer Requirements section. At a minimum, the `aclk` frequency must be higher than the average pixel rate.

Resets

Two external resets are provided: `rst`, which resets the entire core, and `aresetn`, which resets the AXI4-Stream interface. Both resets cause the FIFO to be reset. When asserted, the reset should be held for least two clock periods of the lowest frequency clock.

System Considerations

Buffer Requirements

The FIFO depth is selectable via the GUI when the core is generated. The buffering requirement for the asynchronous FIFO depends mainly on the relative frequency of the AXI4-Stream clock (`aclk`) to the video clock (`vid_io_in_clk`) frequency, and the line standard used.

If the frequency of the AXI4-Stream clock (`Faclk`) is equal to or greater than the frequency of the Video input pixel clock (`Fvclk`), only the minimum buffer size (32 locations) is required. This assumes that the cores connected downstream of the Video In to AXI4-Stream core can sink data at the full video rate. For example, the downstream core can accept data in a virtually continuous stream with gaps occurring only following `EOL`, and each line consecutively with line gaps only preceding `SOF`. In this scenario, the FIFO empties after the `EOL` on each line.

If `Faclk` is less than `Fvclk`, additional buffering is required. The FIFO must store enough pixels to supply them continuously throughout the active line. Due to phasing requirements, the horizontal active period on the output overlaps the effective blanking period of pixels coming in from the AXI4-Stream bus. This means that the input FIFO must also be large enough to provide output pixels continuously during this time.

For AXI4-Stream clock frequencies above the line average but below that of the video input pixel clock, the minimum FIFO initial fill level must be:

FIFO depth min. = 32+ Active Pixels * `Fvclk/Faclk`

If the downstream processing core accepts data at a lower rate than the AXI4-Stream clock, Additional buffering is required in an amount sufficient to prevent the FIFO from overflowing during the course of a frame.

Timing Modes

In video processing, two basic configurations are used for output timing: with frame buffer, and without. For Xilinx reference designs, this usually means with VDMA or without VDMA. The configuration has implications for how the AXI4-Stream to Video Out (Video Out) core is configured, how the VTC operates, and how it interacts with Video Out. The presence or absence of the VDMA determines how the Video Out core synchronizes timing between the AXI4-Stream data and the VTC, and the timing mode in which the VTC operates.

There are two timing modes supported: slave timing mode and master timing mode. In slave mode, the VTC generator is a slave to the Video Out core which controls it through

clock enable. In master mode, the VTC is the timing master for the output side of the VDMA, the output processing cores, and the Video Out core. In master mode, the Video Out core does not control the VTC generator timing; instead, it uses the VTC timing as a reference, and synchronizes the video pipeline to it.

The timing mode (master or slave) is a configuration parameter of the Video Out core. When generating this core, this parameter must be set according to the configuration in which it will be used.

No VDMA - Slave Mode

In slave timing mode, the video source of the AXI4-Stream bus is the timing master. This is used when the video source is external and cannot be controlled. Figure 3-2 shows an example of slave timing mode.

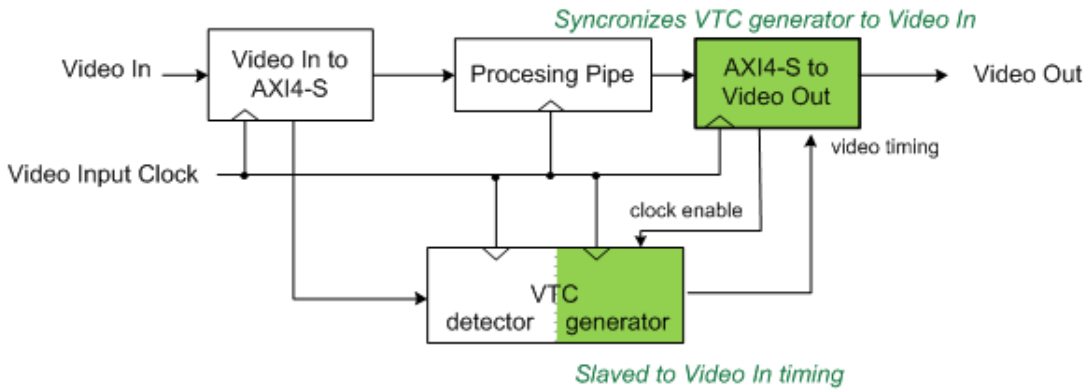

Figure 3-2: **Without VDMA - Slave Timing Mode**

In Slave mode, the Video Out core uses the clock enable to control the timing of the syncs from the VTC generator. Because there is no frame buffer, the video cannot be stopped without losing data. Therefore, the Video Out core regulates the syncs from the VTC generator so that they match the timing of the data in the pipeline.

With VDMA - Master Mode

When a VDMA or other frame-buffer is present, Master timing mode is used. Figure 3-3 shows an example of Master timing mode.

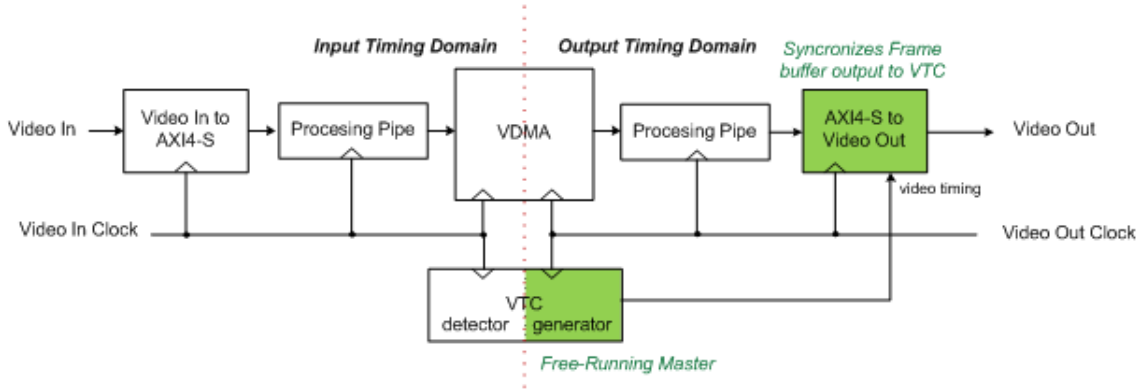

Figure 3-3: **With VDMA - Master Timing Mode**

In this case, the attached VTC generator is the timing master, and the Video Out core synchronizes the data in the processing pipeline to the video timing signals by applying back pressure to the processing pipeline. This means that you can deassert the `tready` signal to stop the flow of pixels. This back pressure is propagated through the processing pipe in the reverse direction of the data flow until it halts the frame buffer output. In this scenario, the video output processing pipeline, from the frame buffer onward, is synchronized with the VTC generator. The VDMA provides video data as it is requested by the Video Out core through the processing pipe.

Unlike prior use models for the VDMA and VTC, `Fsync` is not required and should not be used in this mode. Previously, the VTC issued a frame sync at the start of each frame, which was used in the VDMA to reset the output side of the frame buffer. Now, instead of resetting the frame buffer output at a certain time, the frame buffer output free runs. Synchronization is accomplished through AXI4-Stream back pressure, originating at the Video Out core. Because the feedback wiring and possible programming intricacies of `Fsync` timing are eliminated, the core is easier to use.

With VDMA and Genlock - Master Mode

The Master mode is also compatible with genlock. Figure 3-4 show a genlock configuration. In genlock, the VTC timing is synchronized to an external timing input via an `Fsync` signal into the VTC generator. The Video Out core synchronizes the video data to the VTC timing, and the upstream pipe including the frame buffer output is synchronized by back pressure on the AXI4-Stream.

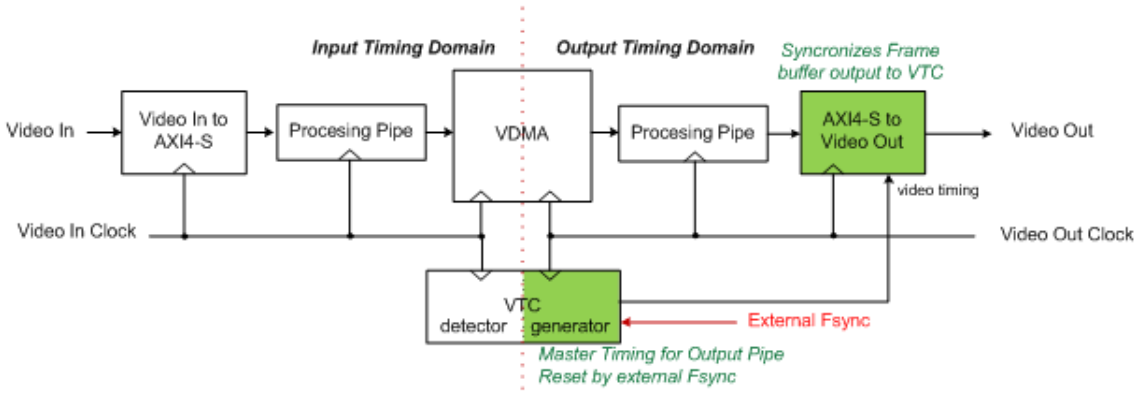

Figure 3-4: **Genlock with VDMA - Master Timing Mode**

Interlaced Operation

To support standard definition video input such as PAL and NTSC, the Video In to AXI4-Stream core supports interlace on the video (with timing) side, the video input has a `field_id` bit as part of its interface and embedded vertical blanks and horizontal blanks. The VTC has a corresponding `field_id` pin defined for this purpose.

Figure 3-5 shows the interfaces on Video In to AXI4-Stream, AXI4-Stream to Video Out, and VTC cores to support the video field ID with the interlace-related signals highlighted in red.

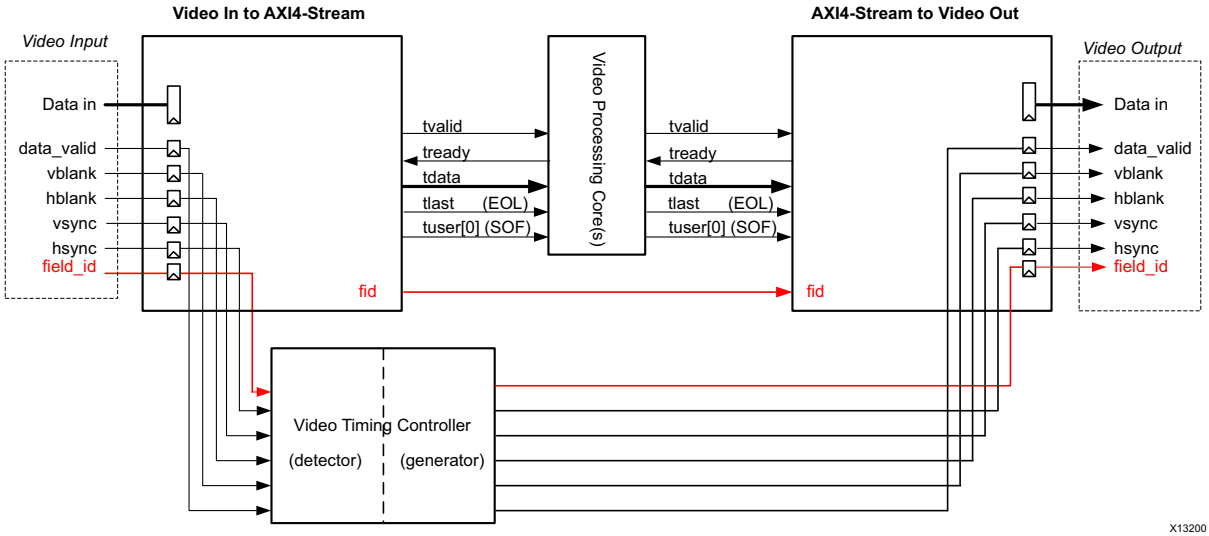

Figure 3-5: **Interlace Signals on Video Cores**

Most video processing cores are field-agnostic, and not aware of whether the picture being processed is an odd or even frame, or a progressive field. Therefore, interlace has no impact on these cores. The Video In to AXI4-Stream core has a frame ID output, `fid`, timed to the

AXI4-Stream bus. This signal can be used as needed in the system. The only cores that use this `fid` bit are the AXI4-Stream to Video Out, VDMA, and Video Deinterlacer cores.

The AXI4-Stream to Video Out core has a field ID input, `fid`, sampled in time with the AXI4-Stream input bus. This `fid` bit must be asserted by the upstream source of AXI4-Stream video. For systems without a frame buffer or de-interlacing, the field ID input originates from the Video In core, as shown in Figure 3-6.

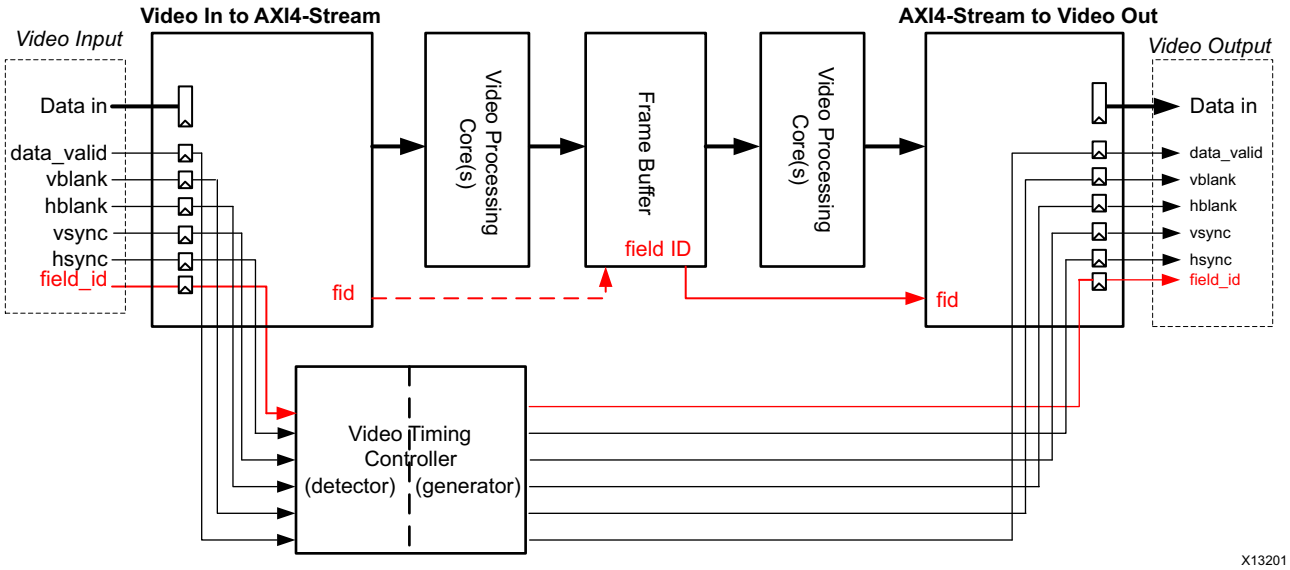

Figure 3-6: **Field ID Connections with a Frame Buffer**

For systems with a frame buffer, the field ID input can come from any core containing a frame buffer. The field ID from the Video In to AXI4-Stream core can be used by the frame buffer if necessary, shown in Figure 3-6.

Note: In Figure 3-6, the AXI4-Stream to Video Out core is operating in slave mode.

A deinterlacer can be used after the Video In to AXI4-Stream core to convert the video format from interlaced to progressive. In this case, the deinterlacer uses the field ID bit, `fid`, from the Video In to AXI4-Stream core, as shown in Figure 3-7.

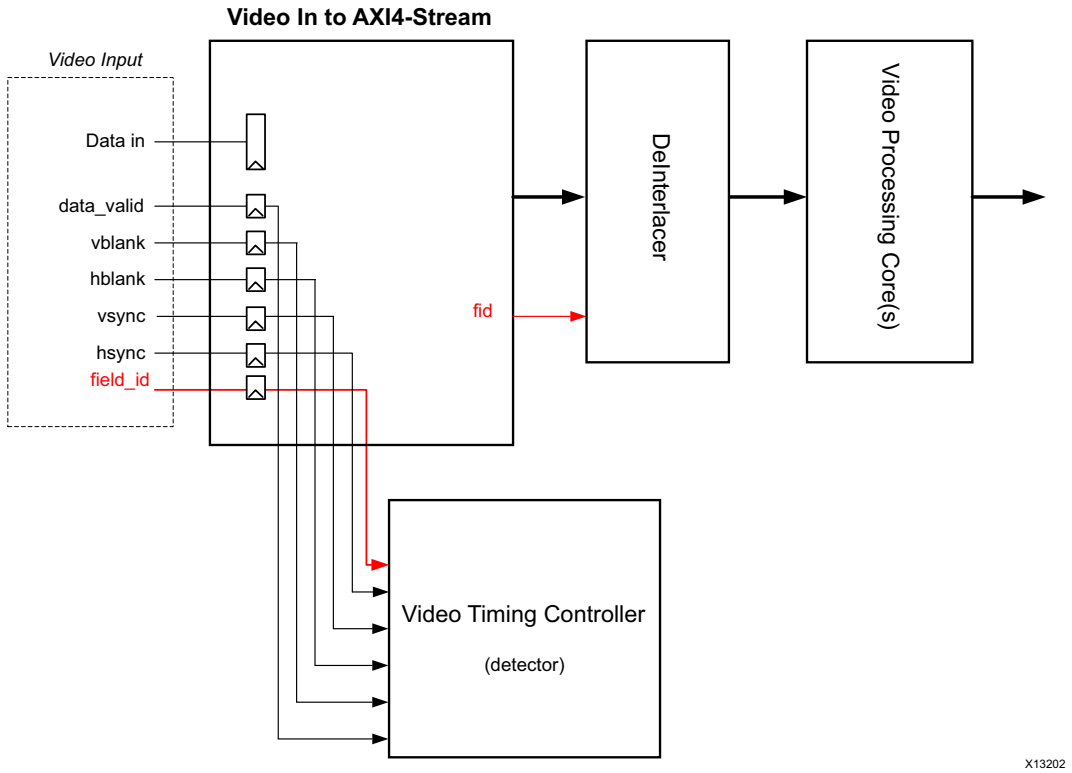

Figure 3-7: **Field ID Connections with a DeInterlacer**

On the Video In to AXI4-Stream core, the `fid` bit changes coincident with `SOF` and remains constant throughout the remainder of the field. On the AXI4-Stream to Video Out core, the `fid` bit is sampled coincident with `SOF` in Figure 3-8. Therefore, the Video In to AXI4-Stream can provide the field bit directly to the AXI4-Stream to Video Out core if no intervening frame buffer exists. When a deinterlacer or frame buffer is used, a similar scheme can be employed: generate the field ID coincident with the start of the field, and on the receiving side sample the field ID coincident with the first received pixel.

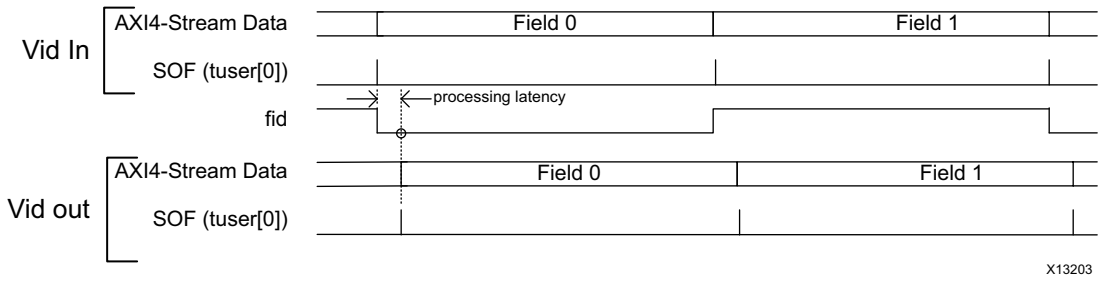

Figure 3-8: **Timing of Field ID for AXI4-Stream**

Module Descriptions

Figure 1-1 shows a block diagram of the Video In to AXI4-Stream core. The video connections are on the left, and the AXI4-Stream interface is on the right. There are two main blocks, the data formatter and the stream coupler. The stream coupler contains an Asynchronous FIFO.

Data Formatter

The data formatter derives the EOL and SOF flags required to transmit AXI4-Stream Video protocol. It also controls writing of the FIFO in the stream coupler. The flags are generated by looking at the edges of the data valid signal. Since only active pixels are carried on AXI4-Stream, the FIFO is only written when active pixels are present. There are input registers on all inputs to minimize input loading. The EOL flag is timed to be coincident with the last pixel before the falling edge of DE. Video data is delayed so that the falling edge of DE can be detected, and the EOL flag asserted coincident with the proper pixel as it is written to the FIFO. Similarly, the SOF flag is created based on the rising edge of DE, however it additionally requires knowledge of the vertical timing to identify the first line. This is done using the logical or OR vsync and vblank. The falling edge of either of these indicates that the input video is in the vertical blanking period. This enables SOF generation, and the SOF flag is asserted at the next rising edge of DE; the first valid pixel of the field.

The rising edge of DE also resets the vertical blanking flip-flop. Figure 3-9 shows the timing of outputs and internal signals relative to the inputs. The de_1 and de_2 signals are delayed versions of de. The outputs are highlighted in bold.

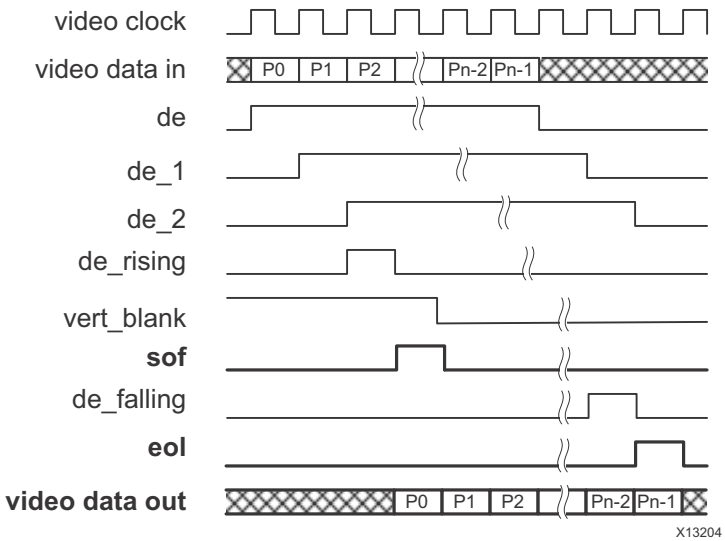

Figure 3-9: **Data Formatter Timing Diagram**

Stream Coupler

The Stream Coupler block consists mainly of an asynchronous FIFO and write logic for the input side of the FIFO. The Output Synchronizer controls the reading of the FIFO. The FIFO serves two primary purposes:

1. Clock domain crossing.
2. Buffering of data between the AXI4-Stream input and the video output.

The buffering requirements depend on the ratio of the AXI4-Stream clock rate to the video clock rate, as described in Buffer Requirements. The Stream Coupler also contains logic to ensure that the first AXI4-Stream transaction following a reset is the first pixel of the frame. Specifically, after a reset or loss of lock, AXI4-Stream `tdata_valid` is not asserted until the `axis_enable` input is active and an `SOF` is also present.

Asynchronous FIFO

The crossing of clock domains requires an asynchronous FIFO. The FIFO designed for this core has two distinguishing features:

1. Status flags and a fill level output in both clock domains.
2. An "invalid" (`read_error`) flag produced in parallel with output data for reads when the FIFO is empty. It also has pointer-inhibiting logic to prevent pointer crossings on underflow and overflow.

Asynchronous FIFO negatively impacts the fill level indicators and the flags. The primary risk is that pointer values could error when sampled from one clock domain to another. Therefore, pointer synchronization is required across clock domains, and a handshake protocol must be used to ensure that all pointer updates are registered, even when the clock rates in the two domains are radically different. Figure 3-10 is a block diagram of the asynchronous FIFO.

IMPORTANT: *Note the synchronizing logic and the handshake between clock domains. The size of the FIFO is set in the GUI when the core is generated.*

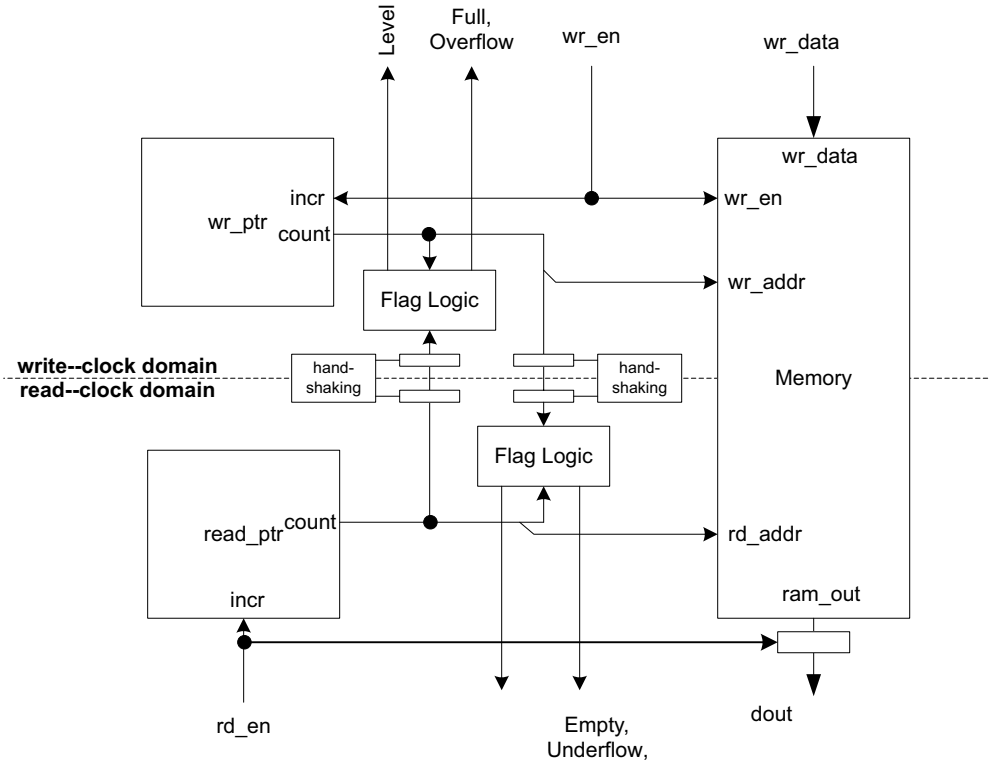

Figure 3-10: **Block Diagram of Bridge Core Asynchronous FIFO**

Synchronized Gray codes are commonly used in asynchronous FIFOs to eliminate the problems of synchronizing multiple counter bits changing on the same clock edge. However, instead of Gray code pointers, the FIFO for the AXI4-Stream to Video Out core uses binary pointers synchronized through handshaking. Calculating the fill level, which is used integrally in the read logic, is simple with binary pointers but impractical with Gray code pointers.

Clock Domain Crossing of Pointers

The synchronization and handshaking for pointers is shown in detail in Figure 3-11. The first two register delays are required to resolve metastability. The third ensures that the register has time to receive the data before the handshake is returned. Otherwise, if the clock in one domain were several times faster than the other, the handshake could be returned before the pointer register is updated in the slower clock domain.

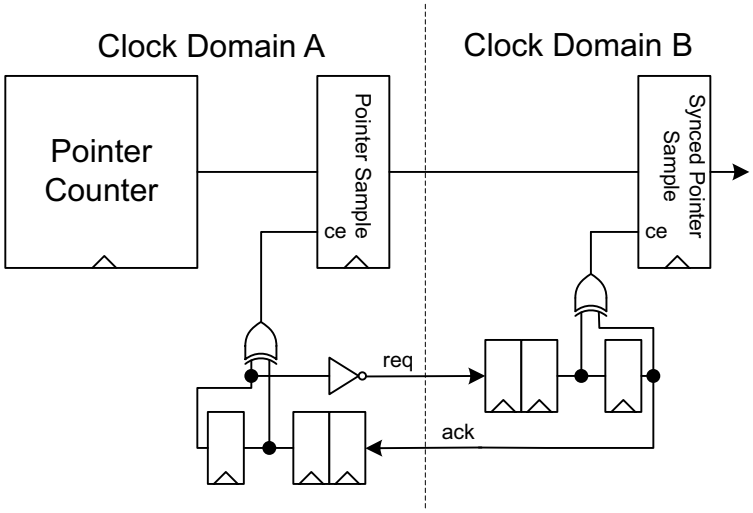

Figure 3-11: **Synchronization and Handshaking for Clock Domain Crossing of Pointers**

The extra delay for returning the handshake signal ensures that data is transferred reliably regardless of the relative clock rates. For example, if Fa > 2Fb, each "req"is still guaranteed to update the sync pointer in domain B. Likewise, if Fb > 2Fa, each `ack` is guaranteed to update the pointer sample in domain A. Figure 3-12 shows an example of pointer synchronization between clock domains, and the handshake scheme shown in Figure 3-11. This scheme uses two states: request and acknowledge. The request state denotes that Req and Ack are not equal, and acknowledge denotes that they are equal.

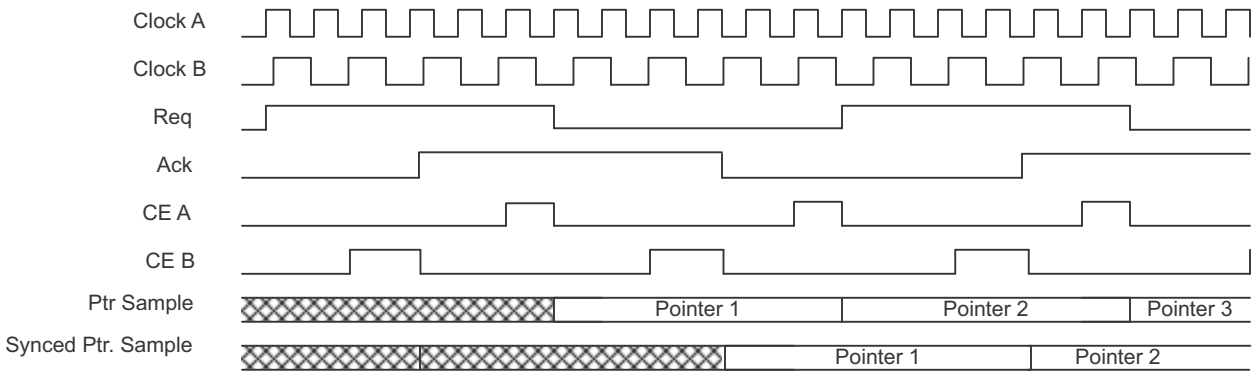

Figure 3-12: **Waveform Diagram of Handshaking and Clock Domain Crossing of Pointers**

This sample and hold method with handshake delays the capture of the pointers by several clock edges in each domain, but the pointer transfers are always reliable and error-free. Pointer latency causes negativity in the level outputs and the flags. The empty flag persists in the read domain for several clocks after a write has occurred. The level output is not necessarily monotonic and is accounted for in the bridge design by providing a small cushion or minimum fill level in the FIFOs so that the pointer negativity does not cause artifacts.

Underflow Prevention

In addition to synchronizing flags, signaling an empty condition is not sufficient because "reads" of the FIFO do not stop on empty. Additional read operations are performed so that the EOF is included in the output of the FIFO. This occurs automatically, and the FIFO must not lose any data. Underflow is not allowed. When new data is eventually written to the FIFO, reading must begin with the first new valid pixel. To do this, the read pointer is inhibited when the FIFO is empty.

The empty flag asserts coincides with the last available location clocked into the output register. The read pointer is not advanced to match the write pointer, but points to the last valid pixel that was read. When empty is asserted, subsequent reads cause the read_error flag to be asserted, flagging the pixel from the FIFO as invalid. The data output does not change.

When a read occurs to an empty FIFO, the invalid flag (read_error) is set, and the read pointer does not increment. The EOL can be advanced through the pipe by a series of reads on the empty FIFO. With each read, an invalid pixel backfills the advancing EOL, but the downstream logic can distinguish these from valid pixels.

Pointer Format

It is important to provide an accurate level, and to distinguish between full and empty conditions when the read and write pointers are equal. This is done by having "revolution" bits on the pointers, in addition to the address bits, as shown in Figure 3-13.

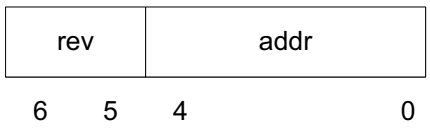

Figure 3-13: **Pointer Format for a 32 Location FIFO**

These extra bits in the pointers allow the level calculations and flags to be unambiguously determined in full and empty conditions.

Read Logic

The read logic controls the handshake for the AXI4-Stream bus and provides pixels to this bus as rapidly as possible. In general, the strategy for AXI4-Stream is downstream-greedy. That is, downstream modules take pixels as soon as they are available and there is buffer space to accommodate them. Since the Video In to AXI4-Stream core is at the front of the pipeline, it strives to empty its FIFO as fast as possible.

The Read Logic controls the tvalid handshaking signal based on the level and flags from the FIFO, and the tready signal returned from the downstream module. Whenever data is available in the FIFO, tvalid is asserted. When tready is returned active, the FIFO is read and the new pixel is again denoted by the tvalid being active. Thus, the tvalid is asserted except when there is no valid data available from the FIFO. The FIFO will only begin

filling if the downstream core cannot accept data as fast as it is coming in from the video bus. This will happen, for example, if the AXI4-Stream clock, `aclk`, is slower than the video clock. In this case, during the active portion of each line, pixels will be coming into the FIFO faster than they can be sent out on the AXI4-Stream bus. Thus the FIFO will begin to fill. Usually the FIFO will empty at the end of the active line when the downstream core is still taking pixels, but the incoming video data is in the horizontal blanking period.

At the end of each line, the `EOL` must be flushed through from the FIFO to the output registers. This is done so that the downstream core can access the complete line without having to wait through the horizontal blanking period for new pixels to push the `EOL` out.

This flushing requirement presents a challenge since this occurs during horizontal blanking, meaning that no data is coming into the FIFO. It requires generation of invalid pixels to flush out the valid pixels. Since no inactive pixels are allowed on the AXI4-Stream bus, these invalid pixels must be swallowed by the core prior to the output.

Flushing of the `EOL` is accomplished by reading from the FIFO whenever it is not empty or when it is empty and the last pixel has the EOF (`tlast`) flag set (assuming AXI4-Stream bus is not applying backpressure). In other words, read the FIFO until it is empty, and beyond when the `EOL` is present. When the FIFO is empty, and the last pixel has been read (the `EOL` pixel) the FIFO will mark subsequent pixels as invalid, but the `EOL` will continue to propagate to the output. Valid pixels get sent out on the AXI-4 Stream bus, invalid pixels do not. In this way, invalid pixels are removed before they get to the AXI-4 Stream bus.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows in the IP Integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 12]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9]

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado® Design Suite environment.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click on the selected IP or select the Customize IP command from the toolbar or popup menu.

For details, see the sections, "Working with IP" and "Customizing IP for the Design" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 11].

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 12] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl console.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Vivado Integrated Design Environment

The Video In to AXI4-Stream core is easily configured to meet the developer's specific needs through the Vivado Design Suite. This section provides a quick reference to parameters that can be configured at generation time.

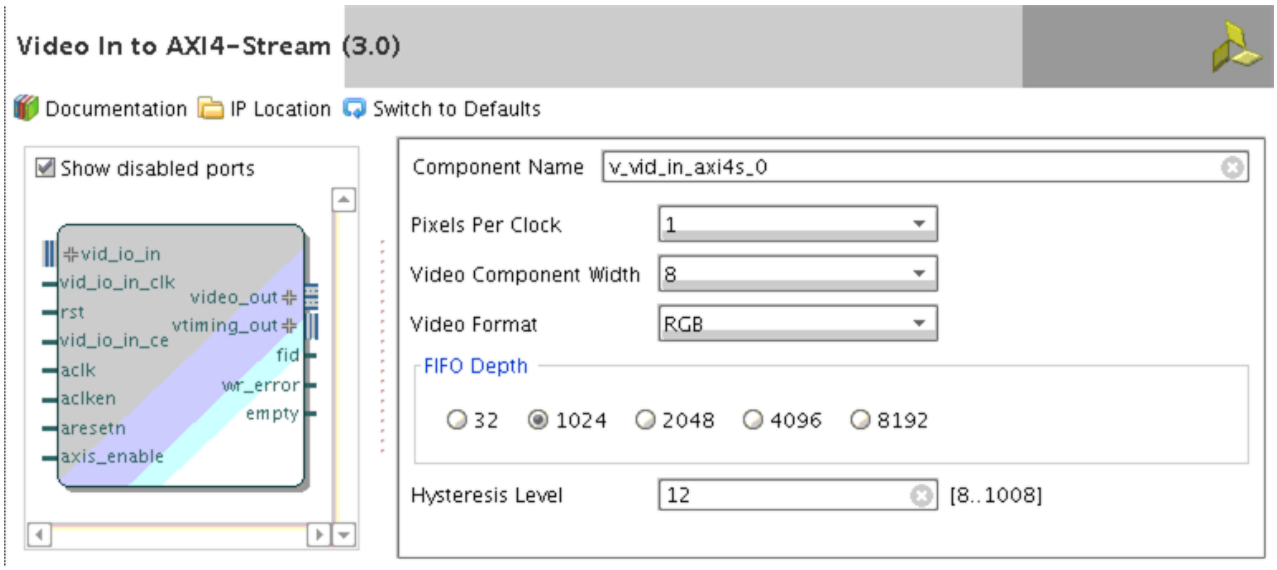

Figure 4-1: **Video In to AXI-4 Stream Vivado GUI**

The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed of characters: a to z, 0 to 9 and "_".
- **Pixels Per Clock**: Specifies the number of pixels to be output in parallel. This parameter affects the data bus width of the input and output. The options for pixels per clock are 1 to 4.
- **Component Data Width**: Specifies the bit width of input samples. This is used in conjunction with the Video Format to determine the width of the input video bus, `vid_data`, and the AXI4-Stream data bus, `m_axis_video_tdata`.
- **Video Format**: Specifies the video format used. The video formats are specified in the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 5]. The format selected determines the number of components used. The number of components (1-4) is multiplied by pixels per clock and the component width to determine the width of the video data bus, `v_data`. In turn, this width is rounded up to the nearest factor of 8 to determine the width of the AXI4-Stream data bus,

m_axis_video_tdata. For example, if the component width is 14, pixels per clock is 2, and the Video Format is RGB (3 components), the vid_data is 84 bits wide and m_axis_video_tdata is 88 bits. When using IP Integrator, this parameter is automatically computed based on the Video Format of the video IP core connected to the slave AXI-Stream video interface.

- **FIFO Depth**: Specifies the number of locations in the input FIFO. The options for FIFO depth are 32, 1024, 2048, 4096, and 8192.
- **Hysteresis Level**: Defines the "Cushion" level of the frame buffer. i.e. the number of locations that are considered the minimum fill level for FIFO operation to start. Generally, this value should be between 12 and 20. It must be at least 16 less than the depth of the FIFO, and at least 16 less than the number of active video lines.

Output Generation

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Required Constraints

The only constraints required are clock frequency constraints for the video clock, vid_io_in_clk , and the AXI4-Stream clock, aclk. Paths between the two clock domains should be constrained with a max_delay constraint and use the datapathonly flag, causing setup and hold checks to be ignored for signals that cross clock domains. These constraints are provided in the XDC constraints file included with the core.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. This core has not been characterized for use in low-power devices.

Clock Frequencies

The pixel clock frequency is the required frequency for this core. See Maximum Frequencies in Chapter 2.

Clock Management

There are two clock domains for this core. The clock crossing boundary is handled by the FIFO and a handshake system for passing pointers between domains.

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Simulation

This chapter contains information about simulating IP in the Vivado® Design Suite environment. For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9].

Synthesis and Implementation

For details about synthesis and implementation, see "Synthesizing IP" and "Implementing IP" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Detailed Example Design

This chapter contains information about the provided example design in the Vivado® Design Suite environment.

Example Design

The Video In to AXI4-Stream core is used in several reference designs and application notes. For detailed examples of how to use this core, refer to the following:

- *Creating a Video Design From Scratch Tutorial from Avnet Electronics* Reference Design [Ref 2]
- *Bridging Xilinx Streaming Video Interface with AXI4-Stream Protocol* (XAPP521) [Ref 3]
- *Designing High-Performance Video Systems in 7 Series FPGAs with the AXI Interconnect* (XAPP741) [Ref 4]

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des

Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite environment.

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des.

Demonstration Test Bench

A demonstration test bench is provided which enables you to observe core behavior in a typical use scenario. You can observe various signals to within the design to gain detailed insight into its operation. There are no stimulus or results files, but the test bench module generates both input and expected data, and performs the comparison of output data to the expected data. Several small frames of parallel video are generated with different timing parameters and applied to the core. The core processes the parallel data through to the AXI4-Stream output. The resulting AXI4-Stream output bus is interfaced to an AXI4-Stream slave emulator. The data extracted by the emulator are compared to the expected parallel video data.

Directory and File Contents

The following file is expected to be generated in the in the demonstration test bench output directory:

- `tb_<IP_instance_name>.v`

Included in this file are the following modules:

- `tb_<IP_instance_name>`
- `timing_gen`
- `test_vid_in`
- `axis_emulation`

Test Bench Structure

Figure 6-1 shows the test bench structure.

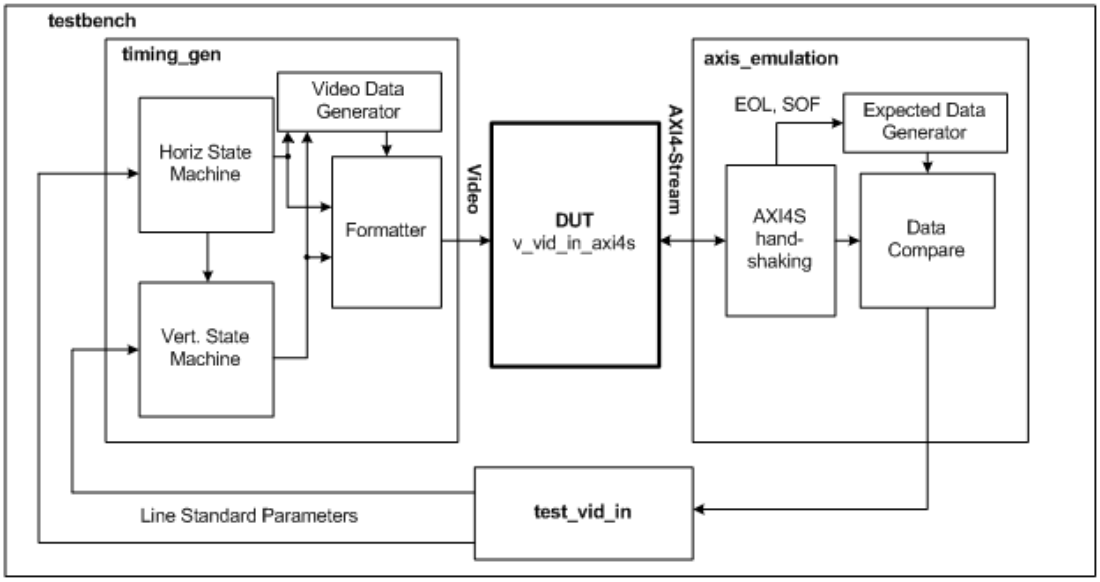

Figure 6-1: **Test Bench Structure**

The top-level entity test bench module instantiates the following modules:

- DUT

  The Video In to AXI4-Stream core instance under test.

- timing_gen

  The timing generator module generates video timing based on the parameters specified by the test program. It also generates the video data that is input to the DUT.

- test_vid_in

  The test program. This program controls the operation of the test bench

- axis_emulation

  The AXI4-Stream slave emulator simulates the AXI4-Stream slave interface driven out by the DUT. It generates handshaking, and receives data from the core. It also generates expected values and compares them to incoming video data.

Verification, Compliance, and Interoperability

Simulation

A test bench incorporating randomization of timing parameters was used to test the Video In to AXI4-Stream core. Testing included testing with multiple frames of data with many different timing parameters and frame sizes.

In addition to stand alone simulation, simulation was done on a pass-through video system consisting of the Video to AXI4-Stream Core core in a system with the AXI4-Stream to Video Out core and the VTC.

Hardware Testing

The Video In to AXI4-Stream core has been validated in hardware using a complete pass through design with an external HDMI video source as an input, and an HDMI video display to verify the output. Re-initialization and re-synchronization was tested by removing and re-applying the video source multiple times.

Interoperability

The AXI4-Stream output interface is compatible with any video processing block that implements the Video Over AXI4-Stream protocol.

The video input is compatible with digital video PHYs such as DVI that provide data in the format provided: Component video data, syncs, blanks, and data valid. With the addition of additional sync deembed logic external to the core, it can also interface with many other digital standards such as HDMI and SDI.

Migrating and Upgrading

This appendix contains information about migrating from an ISE design to the Vivado Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading their IP core, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migration to Vivado Design Suite, see *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Version 3.0 supports IP upgrade from version 2.01.a. It has the following changes from version 2.01.a:

Parameter Changes

The Pixels Per Clock parameter has been added. The default for this parameter is 1. With the default value, the port widths do not change from the earlier version.

Port Changes

Changed the following signal names:

- vid_in_clk to vid_io_in_clk
- vid_de to vid_active_video

Added interlace support and the following signals. Each input port has a default setting for progressive video:

- vid_io_in_ce input default = 1
- vid_field_id input default = 0
- fid output
- vtd_field_id output

Because of the port differences, version 3.0 is not directly compatible with the previous version.

Other Changes

In the customization GUI, combined the video format options "Sensor" and "Luma Only" to single "Mono / Sensor" video format option.

Appendix C

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Video In to AXI4-Stream, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support Web Case.

Documentation

This product guide is the main document associated with the Video In to AXI4-Stream. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Answer Records for the Video In to AXI4-Stream Core

AR 54538

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Please login to the WebCase tool to see your specific support options.

Debug Tools

There are many tools available to address Video In to AXI4-Stream design issues. It is important to know which tools are useful for debugging various situations.

Example Design

The Video In to AXI4-Stream core is used in several reference designs and application notes. Information about the example designs can be found in *Chapter 6, Example Design for the Vivado™ Design Suite*.

Vivado Lab Tools

Vivado inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx FPGA devices in hardware.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado Lab Tools is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Lab Tools for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

General Checks

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Stream Interfaces

Table C-1 describes how to troubleshoot the AXI4-Stream interface.

Table C-1: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
m_axis_video_tvalid stuck low, the downstream core is not receiving data	• Is the axis_enable input HIGH? • Is there a proper signal on vid_vsync or vid_vblank? At least one of these must be present for the core to output data. • On startup or reset, tvalid remains Low until SOF after axis_enable is asserted High.

Other Interfaces

Table C-2 describes how to troubleshoot third-party interfaces.

Table C-2: **Troubleshooting Third-Party Interfaces**

Symptom	Solution
Severe color distortion or color-swap when interfacing to third-party video IP.	Verify that the color component logical addressing on the AXI4-Stream `TDATA` signal is in. If misaligned: In HDL, break up the `TDATA` vector to constituent components and manually connect the slave and master interface sides.
Severe color distortion or color-swap when processing video written to external memory using the AXI-VDMA core.	Unless the particular software driver was developed with the AXI4-Stream TDATA signal color component assignments in mind, there are no guarantees that the software correctly identifies bits corresponding to color components. Verify that the color component logical addressing `TDATA` is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned: In HDL, break up the `TDATA` vector to constituent components, and manually connect the slave and master interface sides.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

http://For a glossary of technical terms used in Xilinx documentation, see the Xilinx Glossary.

http://For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des.

References

These documents provide supplemental material useful with this user guide:

1. *Synthesis and Simulation Design Guide* (UG626)
2. *Creating a Video Design From Scratch Tutorial from Avnet Electronics*. https://www.em.avnet.com/Support%20And%20Downloads/FMC_IMAGEON_Building_Video_Design_Tutorial_14_4_20130110.zip
3. *Bridging Xilinx Streaming Video Interface with AXI4-Stream Protocol* (XAPP521)
4. *Designing High-Performance Video Systems in 7 Series FPGAs with the AXI Interconnect* (XAPP741)
5. *AXI Reference Guide* (UG761)
6. *ISE to Vivado Design Suite Migration Guide* (UG911)
7. *Vivado Design Suite User Guide: Designing with IP* (UG896)
8. *Video Timing Controller Product Guide* (PG016)
9. *Vivado Design Suite User Guide: Logic Simulation* (UG900)

10. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
11. *Vivado Design Suite User Guide: Getting Started* (UG910)
12. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/02/2014	3.0	Added support for multiple pixels per clock.
12/18/2013	3.0	Added UltraScale Architecture support.
10/02/2013	3.0	Synch document version with core version. Updated Constraints and Migration chapters.
03/20/2013	4.0	Updated for core version. Removed ISE chapters. Updated Debugging appendix. Updated Core Interfaces. Updated Designing with the Core chapter.
10/16/2012	3.0	Updated for core version. Updated for ISE v14.3 and Vivado v2012.3. Added Vivado test bench.
07/25/2012	2.0	Updated for core version. Added Vivado information.
04/24/2012	1.0	Initial Xilinx release of core.

Please Read: Important Legal Notices