Video In to AXI4-Stream v4.0

LogiCORE IP Product Guide

Vivado Design Suite

PG043 November 18, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx LogiCORE™ IP Video In to AXI4-Stream v4.0 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 input (clocked parallel video data with synchronization signals - active video with either syncs, blanks or both)
- AXI4-Stream master interface
- Interface to Xilinx Video Timing Controller core for video timing detection
- Support for common or independent clock modes between AXI4-Stream and video clock domains
- Selectable FIFO depth from 32–8192 locations
- Selectable input data width of 8–256 bits
- Support for interlaced operation
- Component width conversion for 8, 10, 12, and 16 bits

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4-Stream(2)
Resources	Performance and Resource Utilization web page
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
Simulation(4)	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis Tools	Vivado Synthesis
Support	
Provided by Xilinx at the, 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 AXI4-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 protocol to interface with Xilinx video processing cores that use this protocol.

The Video In to AXI4-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 *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.

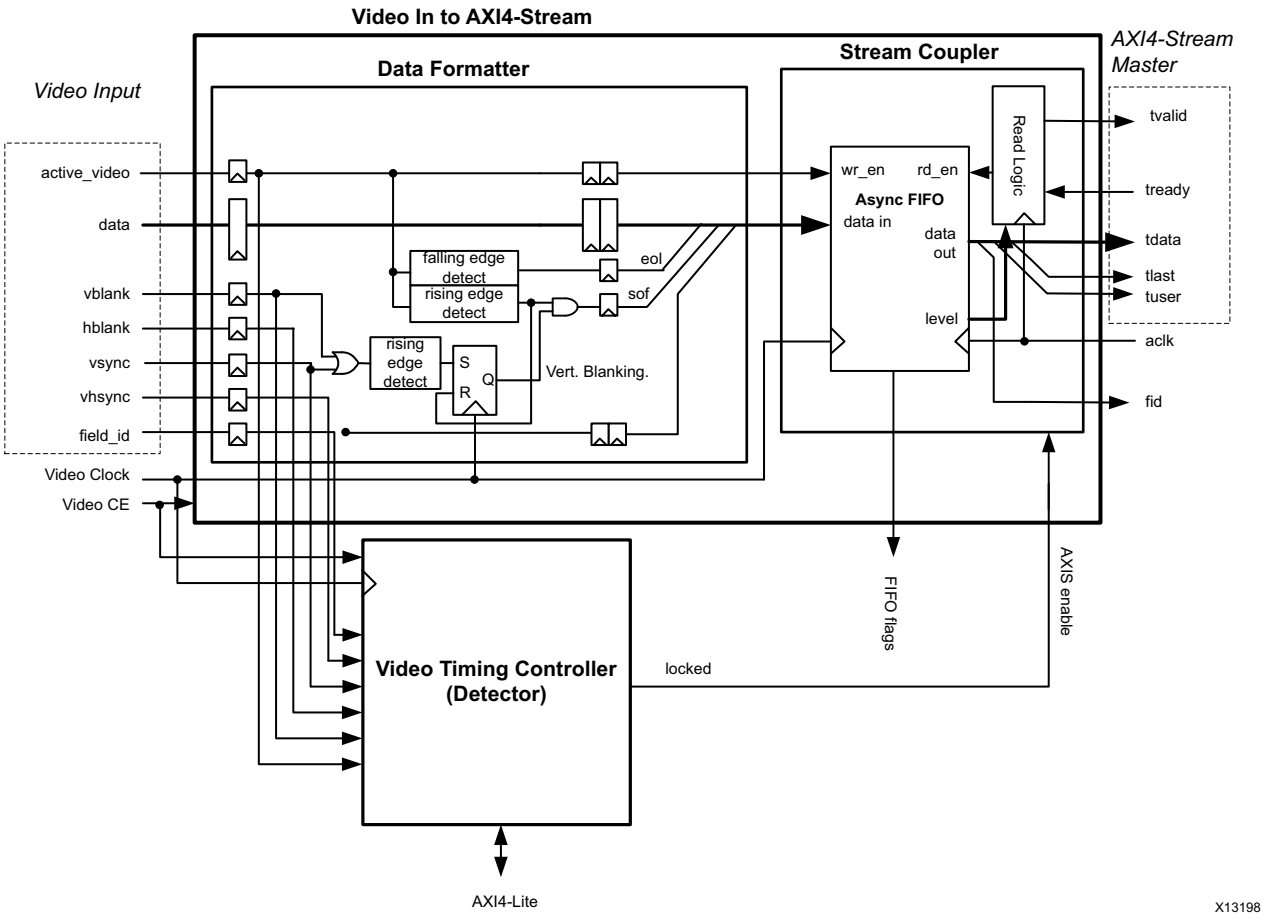

Figure 1-1: **Block Diagram of Video In to AXI4-Stream Core with the Video Timing Controller**

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 *Vivado AXI Reference Guide* (UG1037) [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.

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

For details about resource utilization, visit Performance and Resource Utilization.

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.

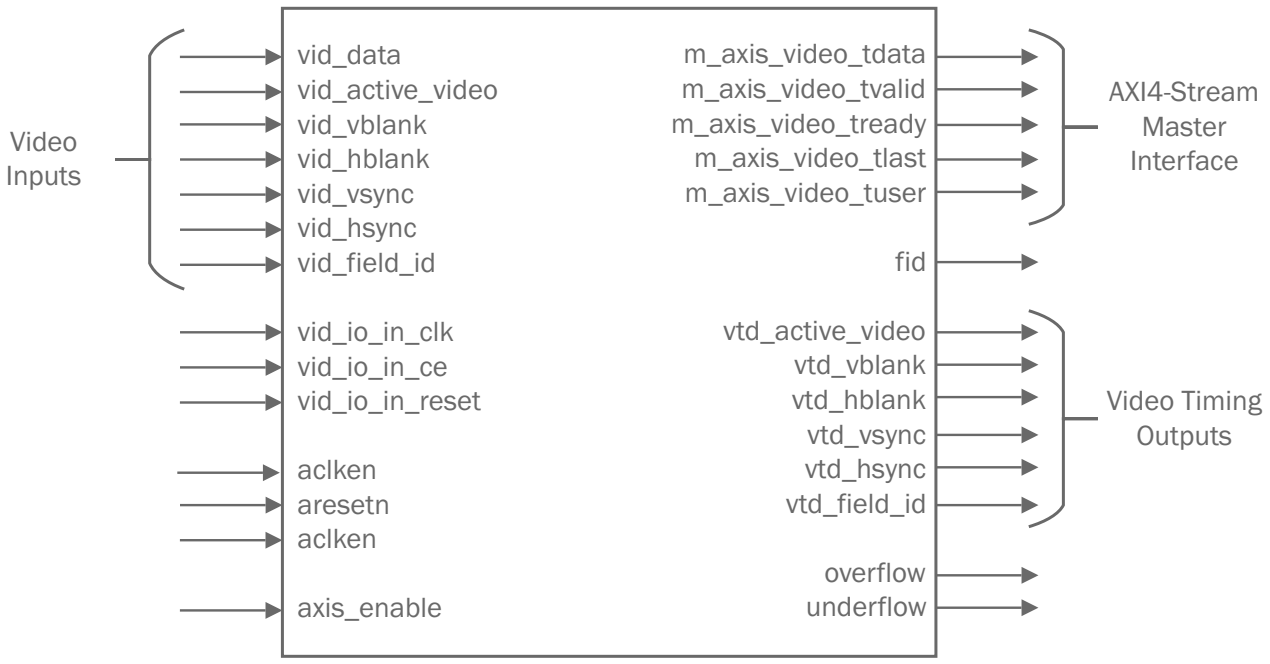

Figure 2-1: **Video In to AXI4-Stream Core Top-Level Signaling Interface**

Common Interface

Table 2-1: **Port Name I/O Width Description**

Signal Name	Direction	Width	Description
aclk	Input	1	AXI4-Stream ACLK.
aclken	Input	1	AXI4-Stream ACLKEN. Active High.
aresetn	Input	1	AXI4-Stream ARESETN. Active Low. Synchronous to ACLK.
axis_enable	Input	1	This input should be connected to the VTC detector locked status and is synchronous to vid_io_in_clk. 1 = Enable writes into FIFO 0 = Disable writes into FIFO
fid	Input	1	Field-ID for AXI4-Stream bus. Used only for interlaced video: 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, otherwise it should be left unconnected.
vid_io_in_clk	Input	1	Native video clock. Only available in independent clock mode.
vid_io_in_ce	Input	1	Native video clock enable
vid_io_in_reset	Input	1	Native video clock domain reset. Synchronous to vid_io_out_clk. Only available in independent clock mode. Active High.

Table 2-1: **Port Name I/O Width Description *(Cont'd)***

Signal Name	Direction	Width	Description
overflow	Output	1	Flag indicating that the FIFO has over-flowed. Synchronous to vid_io_in_clk. If an overflow occurs, this could indicate that the connected AXI4-Stream Slave is creating excessive back-pressure.
underflow	Output	1	Flag indicating that the FIFO has under-flowed. This should never occur under normal operation. Synchronous to aclk.

ACLK

The AXI4-Stream output signals are 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 to 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 reset, which is synchronous to the ACLK domain. When the bridge is in independent clock mode, this reset is used to reset the AXI4-Stream input side of the video bridge including the internal FIFO. Asserting either this reset causes the internal FIFO to be reset. In common clock mode this reset is used to reset the entire bridge.

Video Clock

The video input interface and video timing interface must be synchronous to vid_io_in_clk.

Video Clock Enable

The input signal vid_io_in_ce controls the clock enable of all registers in the video clock domain. This signal is typically used when the video clock domain is clocked at a higher rate then the video timing standard.

Video Reset

The video reset signal vid_io_in_reset signal is only available when the core is configured in independent clock mode. This active High signal is synchronous to

vid_io_in_clk and is used to reset the input side of the bridge. Asserting either this reset or aresetn causes the internal FIFO to be reset.

Video Timing Interface

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

Signal Name	Direction	Width	Description
vtd_vsync	Out	1	Vertical sync video timing signal.
vtd_hsync	Out	1	Horizontal sync 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. 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 2-3: **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 sync video timing signal. Active High
vid_hsync	In	1	Horizontal sync 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

Table 2-4 describes the AXI4-Stream signal names and descriptions. See *AXI4-Stream Video IP and System Design Guide* (UG934) for more information.

Table 2-4: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
m_axis_video_tvalid	Output	1	AXI4-Stream TVALID. Active video data enable
m_axis_video_tdata	Output	1	AXI4-Stream TDATA. Video data
m_axis_video_tuser	Output	1	AXI4-Stream TUSER. Start of Frame
m_axis_video_tlast	Output	1	AXI4-Stream TLAST. End of Line
m_axis_video_tready	Input	1	AXI4-Stream TREADY. Slave Ready

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 AXI4-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.

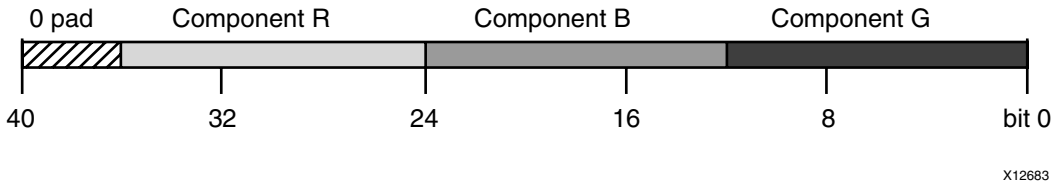

Figure 2-2: **RGB Data Encoding on m_axis_video_tdata**

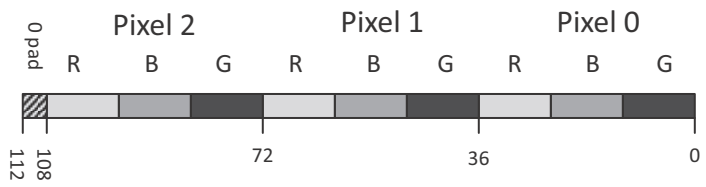

Figure 2-3: **Three Pixels per Clock Format on m_axis_video_tdata.**

The bridge is also able to perform component width conversion from the input to output for any combination of width including: 8, 10, 12, and 16 bit. The example shown in Figure 2-4 illustrates trimming the component width from 12 bits on AXI4-Stream input to 8 bits on the Video output. The four LSB's of each component are trimmed and the remaining data is packed onto the output video bus.

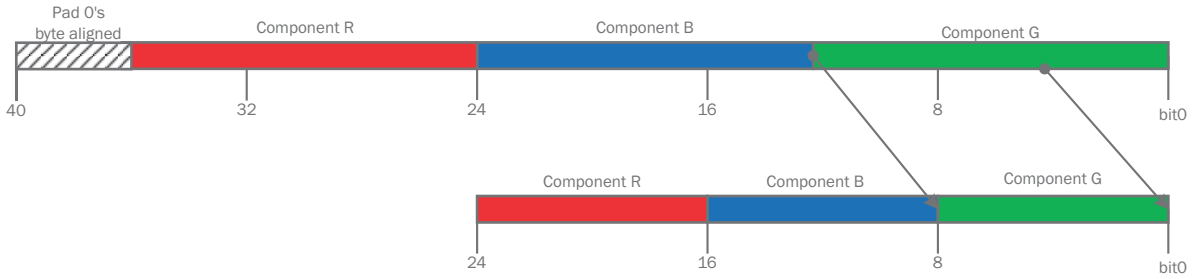

Figure 2-4: **Component Width Trimming from AXI4-Stream input to Video Output (12b to 8b)**

The example shown in Figure 2-5 illustrates padding the component width from 8 bits on AXI4-Stream to 12 bits on the Video output. The four LSB's on the output of each component are padded to zeros and the upper MSB's are mapped onto the bus from the AXI4-Stream input.

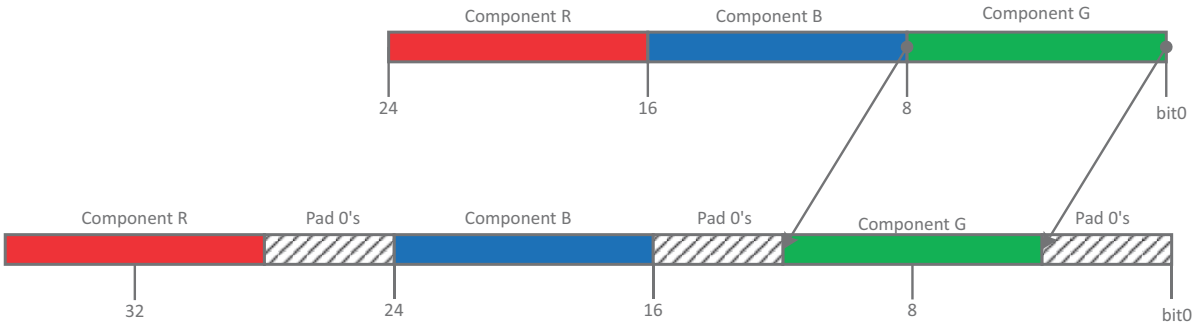

Figure 2-5: **Component Width Padding from AXI4-Stream input to Video Output (8b to 12b)**

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 tuser 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 2-6.

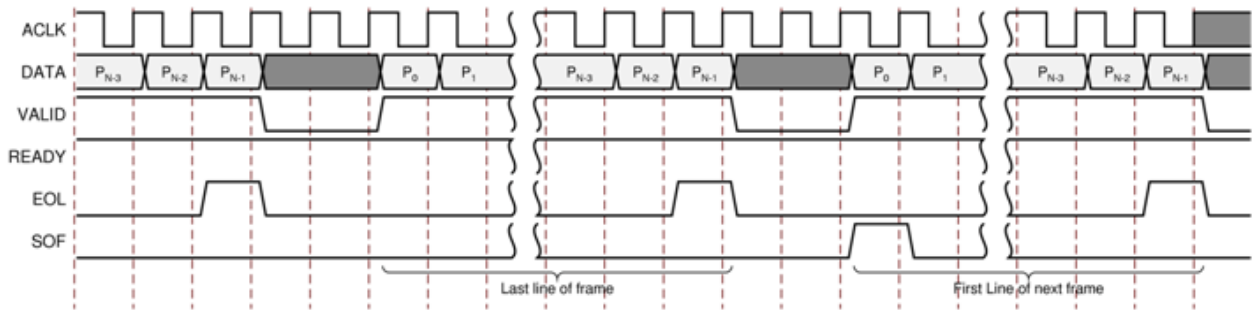

Figure 2-6: **Use of EOL and SOF Signals**

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 9] 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 AXI4-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.

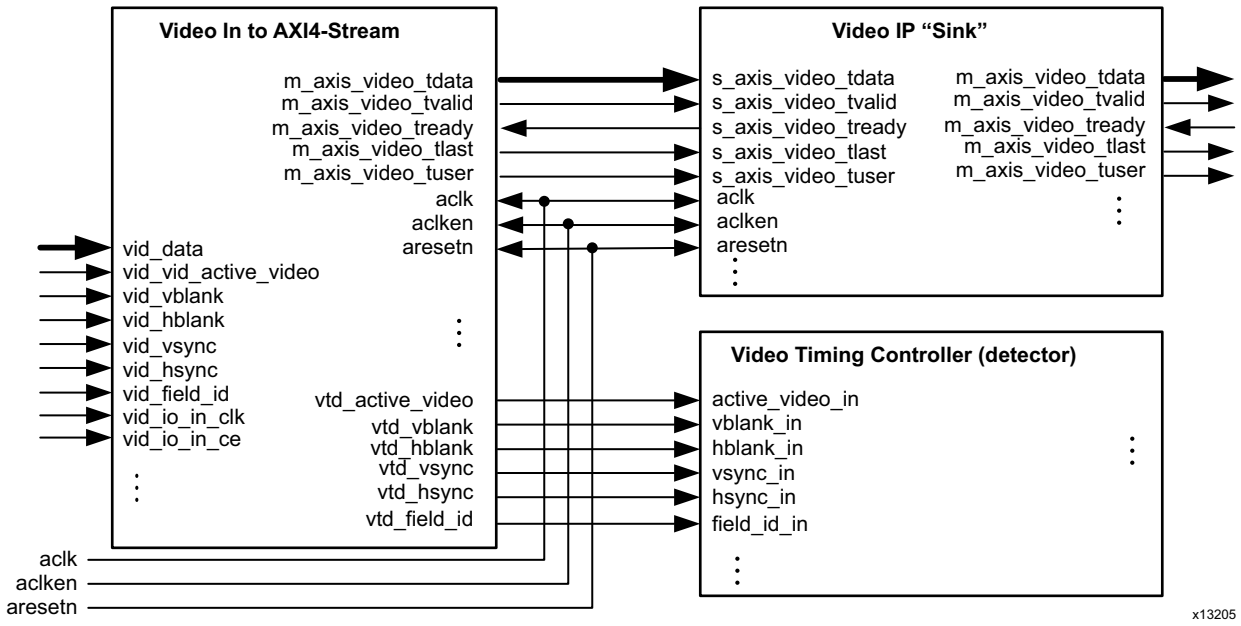

Figure 3-1: **Video In to AXI4-Stream Connectivity**

Clocking

There are two clocking modes used by the Video In to AXI4-Streambridge, either common (synchronous) or independent (asynchronous). The common clocking mode is used when the native and AXI4-Stream sides of the bridge are running from a common synchronous clock. The common clocking mode disables the clock domain crossing logic in the internal FIFO, therefore saving resources. The independent clocking mode is used when the bridge requires asynchronous and independent clocks for the native and AXI4-Stream sides of the bridge.

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

When the core is in common clock mode, there is only a single reset input port `aresetn` that is used to reset both the AXI4-Stream output and Video input sides of the bridge. In independent clock mode, an additional reset port `vid_io_in_reset` is used to reset the

Video output side of the bridge. To reset the entire core in independent clock mode, both resets much be asserted. In independent clock mode, both resets are OR'ed together and synchronized for the purpose of resetting the FIFO; therefore, asserting either reset causes the FIFO to be flushed. Resets must be synchronous to their respective clock domains. The bridge requires that resets be externally synchronized to the destination clock domain as necessary to avoid metastability. When asserted, the reset should be held for at 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.

Slave Mode

The slave timing mode has the following characteristics:

- Automatically minimizes back-pressure to upstream masters
- Automatically minimizes latency and buffering to downstream sinks

The slave timing mode is able to achieve these desired behaviors because of its control over the VTC generator's lag (or phase) relative to the incoming stream. If you require manual control over the VTC's phase relationship to the stream, it is recommend to use the master mode, although this is not a common use case.

To minimize back-pressure through de-assertion of `TREADY`, the bridge insures that the internal FIFO remains empty during the initial coarse alignment phase of the synchronizer where the SOF of the incoming stream and VTC are aligned. After coarse alignment, there is zero lag between the bridge and VTC reducing the probability of the FIFO filling up during fine alignment. Minimizing the phase difference between the incoming stream and VTC input timing ensures that the bridge queues only the minimum number of samples required to achieve lock, which reduces latency. The bridge synchronizer is able to precisely lag the VTC based on the amount of stalls in the incoming stream automatically increasing the cushion in the FIFO to absorb stalls.

Based on your configuration of the FIFO hysteresis (or fill level), the synchronization behavior and latency can be effected. The hysteresis setting forces the bridge to queue up the desired number of samples before starting the synchronization process. The hysteresis

level is meant to be an aid to the synchronizer in situations where it cannot automatically establish the lag or as a method to speed up lock times. Typically, the hysteresis should be set to a small value to allow the synchronizer to automatically establish the lag and reduce latency. The lag information from the synchronizer after lock is achieved can be used to optimize the FIFO depth option in the GUI for a particular input stream.

Figure 3-2 shows an example of slave timing mode.

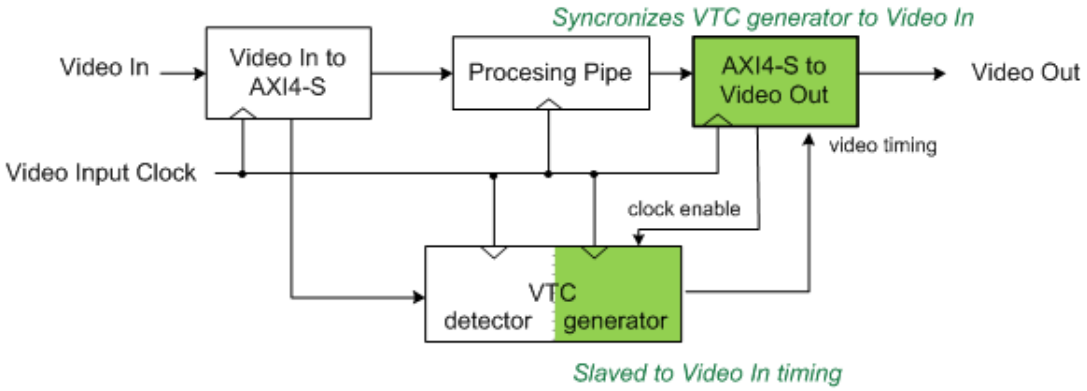

Figure 3-2: **Without VDMA - Slave Timing Mode**

When designing systems with no external frame buffer, it is important to consider EOL flushing by upstream AXI4-Stream sources. Although video timing is not embedded into the AXI4-Stream signal set, it is inherently carried in the stream by way of stall periods where `TVALID` is Low. In an ideal AXI4-Stream scanline in slave mode, video samples arrive back-to-back with no stalls in between subsequent pixels during active periods followed by blanking period where the stream is stalled for a consecutive number of clock cycles.

In imperfect AXI4-Stream sources, the stall period can become fragmented within a scanline, requiring the Video Out bridge to absorb fragments by establishing a fill level as described earlier. Upstream AXI4-Stream sources should be designed to flush the EOL pixel as quickly as possible within the period of a scanline to avoid requiring multi-line buffering. Refer to *AXI4-Stream Video IP and System Design Guide* (UG934) to understand active, line, and frame pixel rates. In general, when the AXI4-Stream source cannot maintain the line pixel rate, line buffering is not sufficient to handle stalls, and therefore the Video Out bridge is not able to synchronize to the stream.

Master Mode

The master timing mode is used when the VTC phase relationship to the incoming stream should not be automatically adjusted by the bridge as in the slave timing mode. Instead, at the start of coarse alignment, the bridge immediately buffers incoming samples without regard to any phase relationship with the VTC. After the VTC first SOF is detected and the FIFO hysteresis level is met, the bridge starts reading samples according to the VTC timing. if necessary, you must manually adjust the phase relationship of the VTC with respect to the stream.

Figure 3-3 shows an example of Master timing mode.

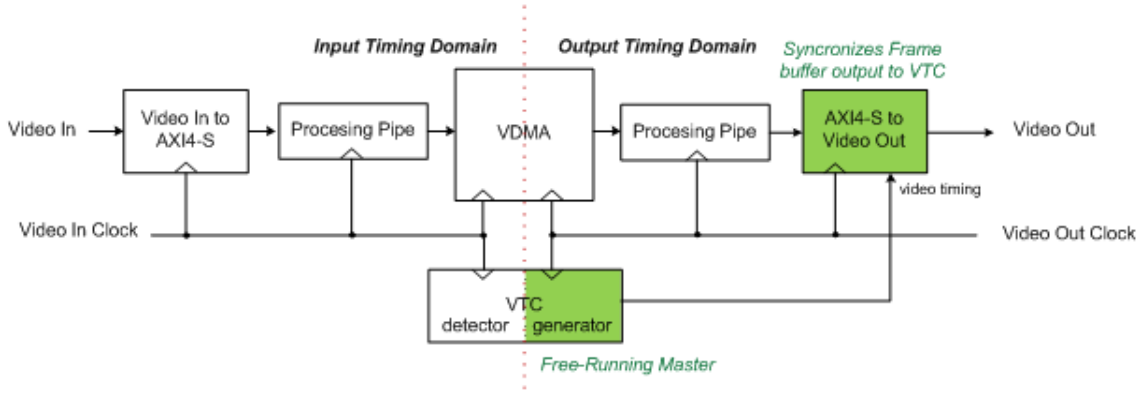

Figure 3-3: **With VDMA - Master Timing Mode**

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.

Master Mode with Fsynch

You have control over phase relationship between the incoming stream and VTC. Figure 3-4 shows an example configuration of the Video Out bridge and VTC using the external Fsync to manually adjust the phase. Pulsing the Fsync signal causes the VTC timing to be reset on demand. Refer to the *Video Timing Controller LogiCORE IP Product Guide* [Ref 6] for more information.

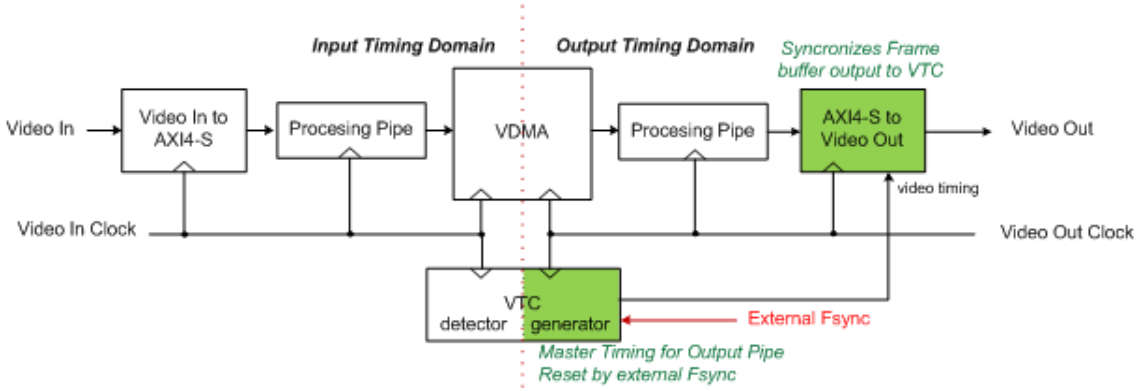

Figure 3-4: **Genlock with VDMA - Master Timing Mode**

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 `vid_field_id` bit as part of its interface and embedded vertical blanks and horizontal blanks. The VTC has a corresponding `vid_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.

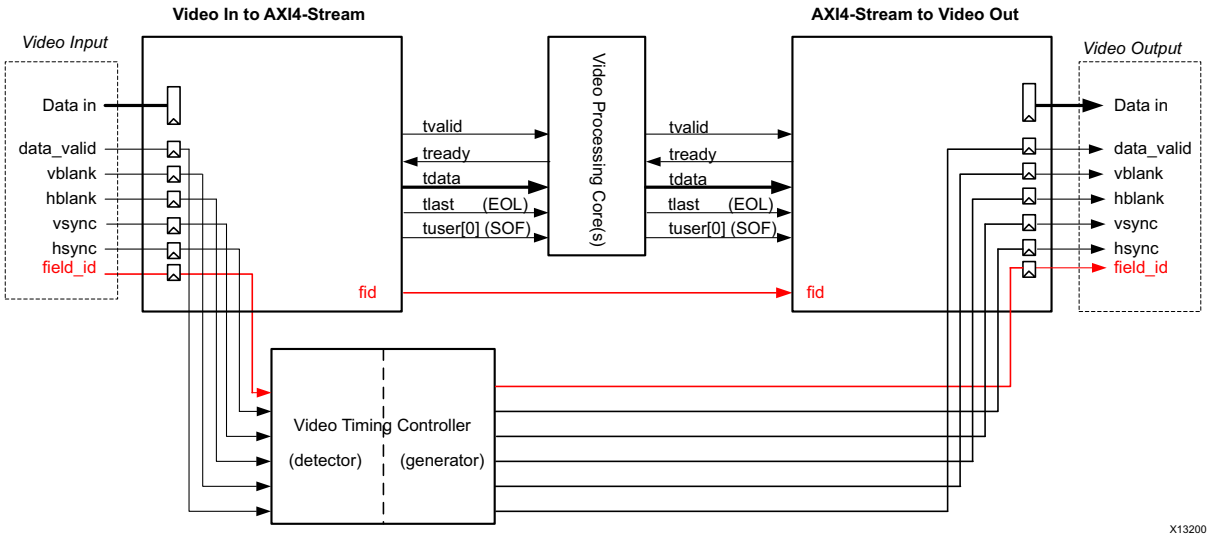

Figure 3-5: **Interlace Signals on Video Cores**

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.

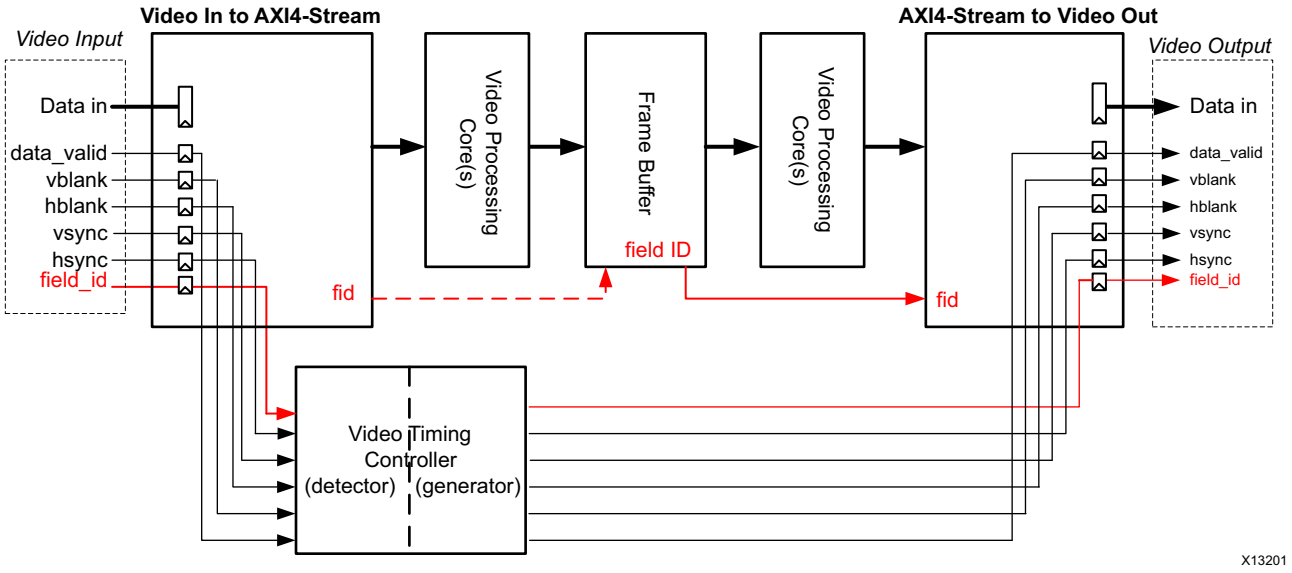

Figure 3-6: **Field ID Connections with a Frame Buffer**

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.

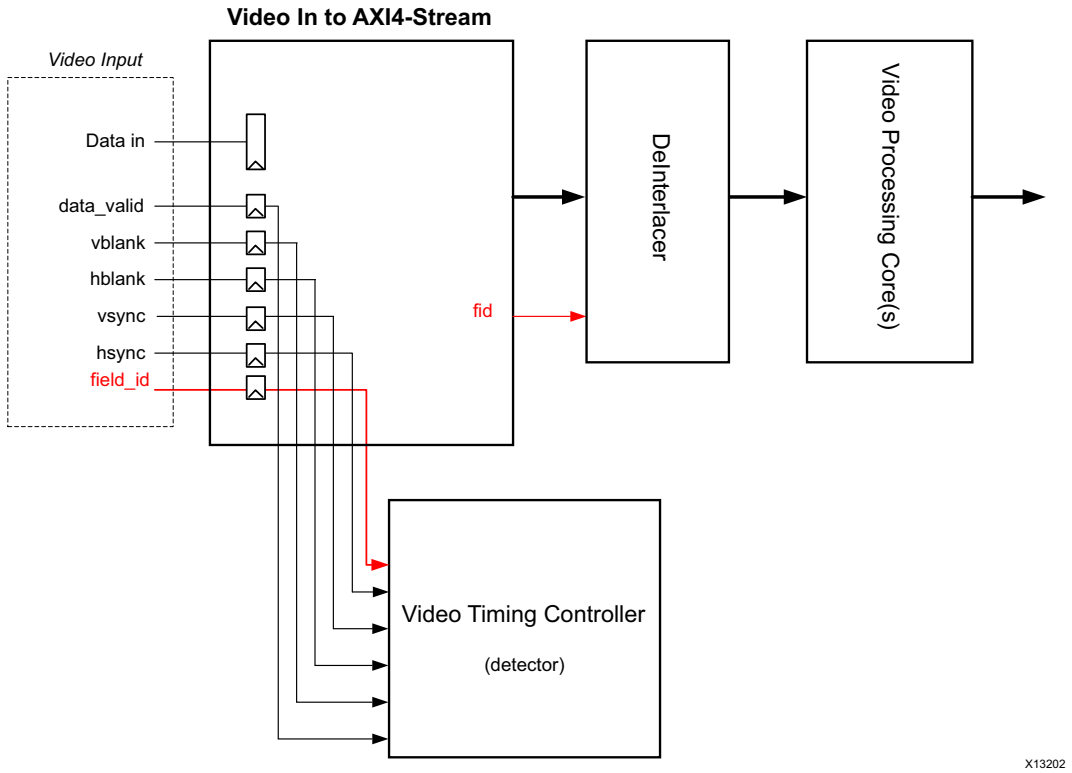

Figure 3-7: **Field ID Connections with a DeInterlacer**

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.

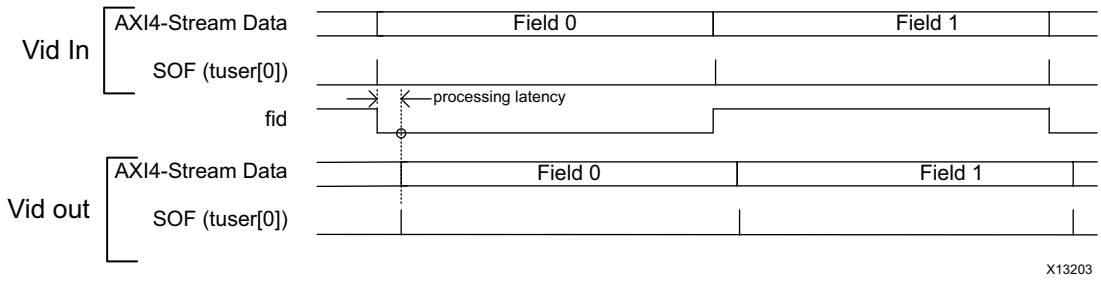

Figure 3-8: **Timing of Field ID for AXI4-Stream**

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.

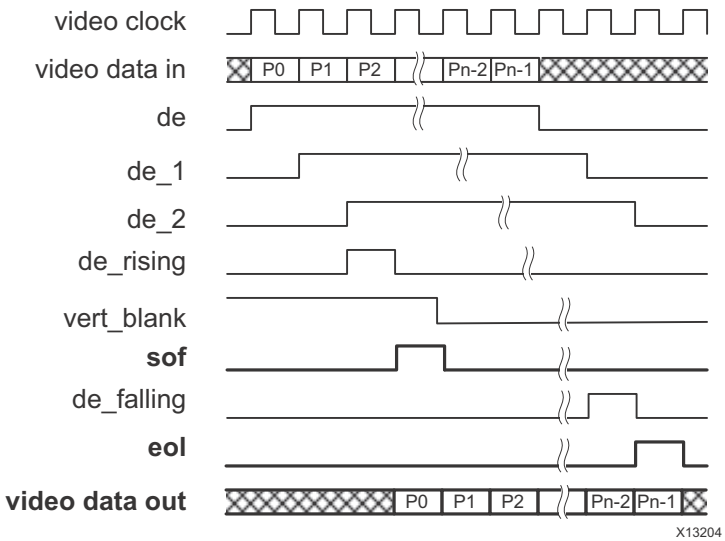

Figure 3-9: **Data Formatter Timing Diagram**

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.

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.

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 and 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 14]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 12]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10]

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 8] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 12].

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 14] 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.

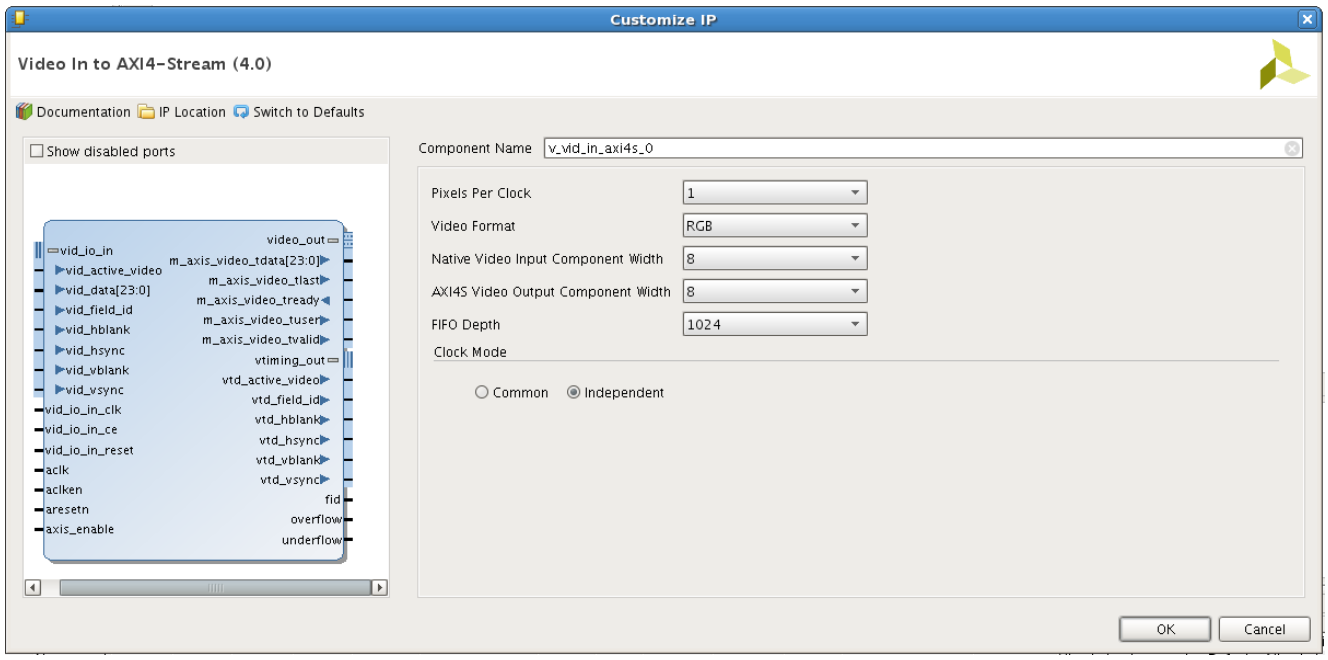

Figure 4-1: **Video In to AXI4-Stream Vivado GUI**

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, 2, or 4.
- **Input Component Width**: Specifies the video component bit width over the input video data bus.
- **Output Component Width**: Specifies the video component bit width over the output AXI4-Stream TDATA bus.
- **Clock Mode**: The clock mode is used to specify whether the AXI4-Stream output and Video input signals are clocked using common or independent clocks.

- **Video Format**: Specifies the video format used. The video formats are specified in the *Video IP: AXI Feature Adoption* section of the *Vivado AXI Reference Guide* (UG1037) [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.

Output Generation

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

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 10].

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 8].

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]
- *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.

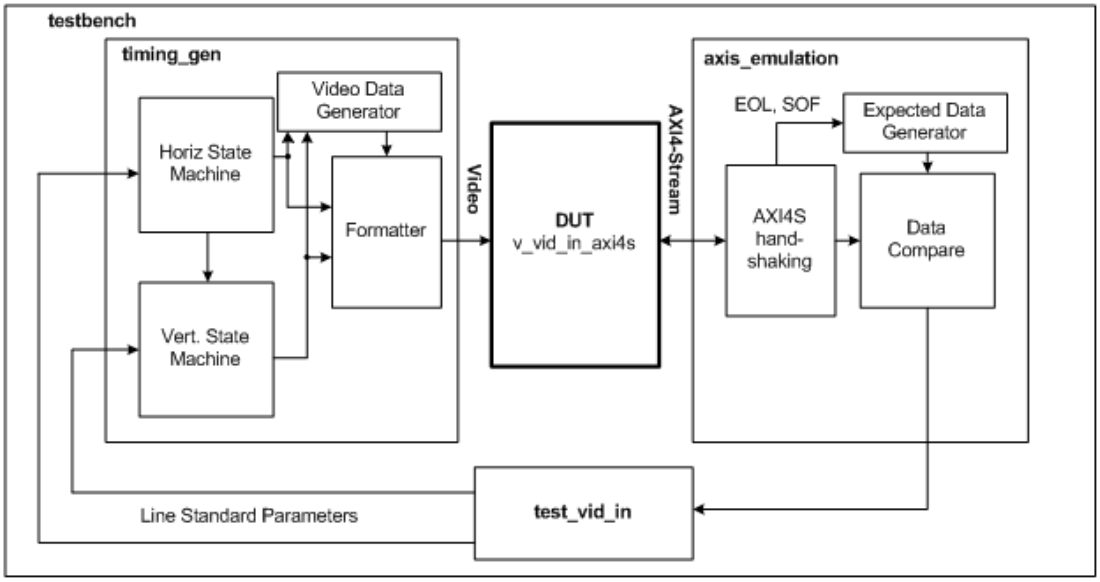

Figure 6-1: **Test Bench Structure**

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.

Appendix A

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 7].

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 4.0 supports IP upgrade from version 3.0. It has the following changes from version 3.0:

Parameter Changes

The Clock Mode parameter has been added to allow for common or independent clocks. The default clock mode is set to independent.

The Video Component Width parameter has been replaced by Input Video Component Width and Output Video Component Width parameters to allow for component width conversion.

The Hysteresis Level parameter has been removed.

Port Changes

Removed the following ports:

- rst
- wr_error
- empty

Added the following ports:

- vid_io_in_reset
- overflow
- underflow

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 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 or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. 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

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change 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, Open a WebCase by selecting the WebCase link located under Support Quick Links.

- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.

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 Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be

analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx.

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 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* (UG1037)
6. *Video Timing Controller LogiCORE IP Product Guide* (PG016)
7. *ISE to Vivado Design Suite Migration Guide* (UG911)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Video Timing Controller Product Guide* (PG016)
10. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
11. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)

12. *Vivado Design Suite User Guide: Getting Started* (UG910)

13. *AXI4-Stream Video IP and System Design Guide* (UG934)

14. *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
11/18/2015	4.0	Added UltraScale+ support.
09/30/2015	4.0	Updated System Clocking and Resets.
04/01/2014	3.0	Added support for multiple pixels per clock.
12/18/2013	3.0	Added UltraScale Architecture support.
10/02/2013	3.0	Sync 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