LogiCORE IP AXI4-Stream to Video Out v2.01a

Product Guide

PG044 October 16, 2012

Table of Contents

SECTION I: SUMMARY

Introduction

The Xilinx LogiCORE™ IP AXI4-Stream to Video Out core is designed to interface from the AXI4-Stream interface implementing a Video Protocol to a video source (parallel video data, video syncs, and blanks). The core works in conjunction with Xilinx Video Timing Controller (VTC) core, providing a bridge between video processing cores with AXI4-Stream interfaces and a video output.

Features

- AXI4 Stream slave interface for input
- Video (parallel video data, video syncs, and blanks) output
- In slave timing mode, interface to Xilinx Video Timing Controller core for video timing generation
- In master timing mode, automatically synchronizes AXI4-Stream Video to video timing.
- Automatically synchronizes video timing to AXI4-Stream video
- Handles asynchronous clock boundary crossing between AXI4-Stream clock domain and video clock domain
- Selectable FIFO depth from 64 -8192 locations
- Selectable data width for 8-64 bits

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000 [2], Artix™-7, Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Stream[3]
Resources	See Table 2-1 through Table 2-8.
Provided with Core	
Documentation	Product Guide
Design Files	ISE: Verilog source code Vivado: Verilog source code
Example Design	Not Provided
Test Bench	Verilog
Constraints File	Not Provided
Simulation Models	Verilog source code
Supported Software Drivers	Not Applicable
Tested Design Flows	
Design Entry Tools	Vivado™ 2012.3 Design Suite[5], ISE 14.3
Simulation[4]	Mentor Graphics ModelSim, Xilinx® ISim
Synthesis Tools	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Supported in ISE Design Suite implementations only.
3. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG761 AXI Reference Guide.
4. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.
5. Supports only 7 series devices.

Overview

Many Xilinx video processing cores utilize the AXI-4 Stream Interface implementing a Video Protocol (as defined in the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 1]) to transfer video between IP cores. Conversely, between systems, video is commonly transmitted with explicit blanking and sync signals for horizontal and vertical timing, and a data valid signal. DVI is an example of such a transmission mode. The AXI4-Stream to Video Out core converts AXI4-Stream Video protocol from Xilinx video processing cores that use this protocol to video output with explicit sync and timing to interface with common video systems that use timing information.

The AXI4-Stream to Video Out core input is an AXI4-Stream interface in slave mode. This interface consists of parallel video data, `tdata`, handshaking signals `tvalid` and `tready`, and two flags, `tlast` and `tuser` which serve to identify certain pixels in the video stream. The `tlast` signal designates the last valid pixel of each line, and is also known as the end of line (EOL). The `tuser` signal 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 interface because there are no sync or blank signals. Only actives pixel are carried on the bus.

The AXI4-Stream to Video Out core outputs video. For the purposes of this document, video is defined as the following signals:

- Parallel video data
- Pixel clock
- Vsync
- Hsync
- Vbank
- Hblank
- DE (data enable)

An AXI4-Stream to Video Out core with a Video Timing Controller (VTC) block diagram is shown in Figure 1-1.

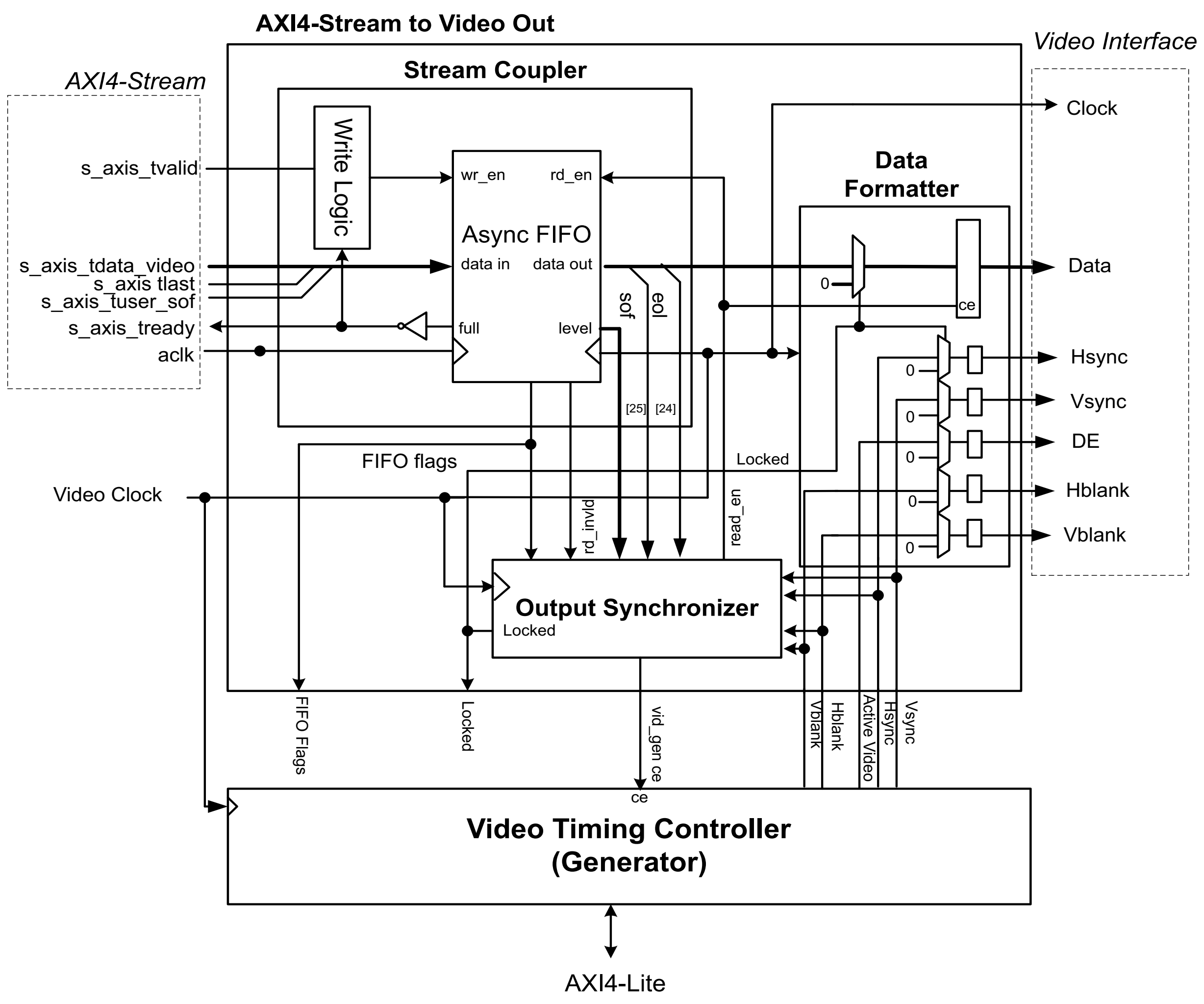


Figure 1-1: **Block Diagram of AXI4-Stream to Video Out Core and Usage with the Video Timing Controller**

The core is designed to be used in parallel with the generator functionality of the VTC. The VTC produces video timing signals based on video timing parameters such as the number of active pixels per line and the number of active lines via an AXI4-Lite interface. The Output Synchronizer section of the core synchronizes timing from the VTC to the video data from the AXI4-Stream Bus.

There are two timing modes supported: slave timing mode and master timing 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 1-2 shows an example of slave timing mode.

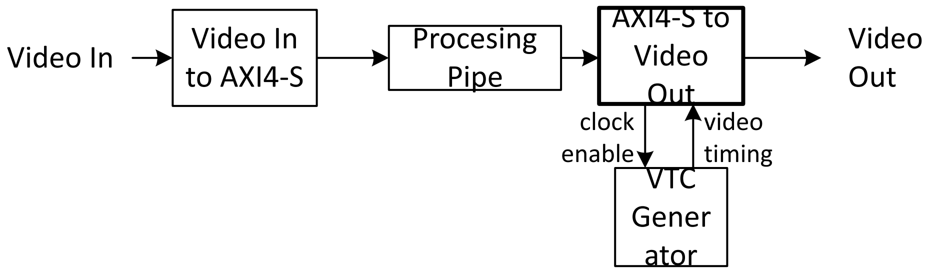


Figure 1-2: **Slave Timing Mode Example System**

Master timing mode is used when the video source is within the processing pipeline and can be controlled by the VTC, for example, from a multi-frame buffer. Figure 1-3 shows an example of master timing mode.

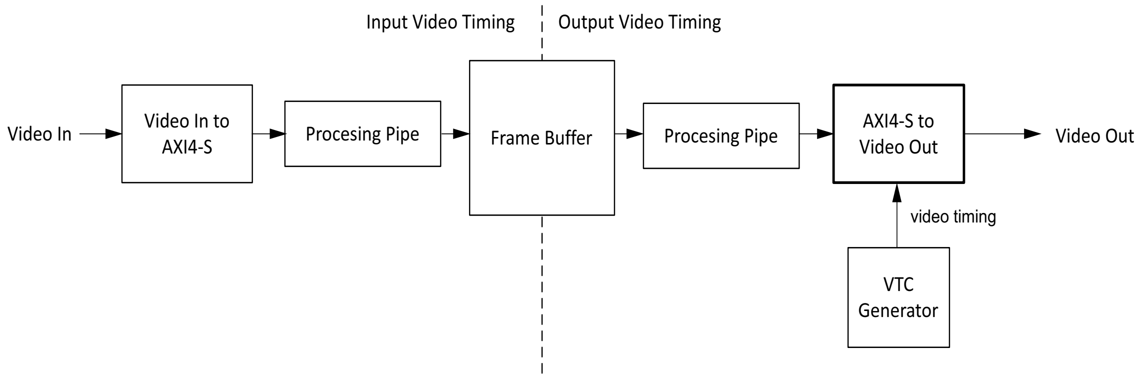


Figure 1-3: **Master Timing Mode Example System**

In this case, the attached VTC generator is the timing master the AXI4-Stream to 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 de-assert 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 on down is synchronized with the VTC generator.

Furthermore, the frame buffer and output pipeline can be synchronized to an external timing reference by synchronizing the VTC to an external reference and having the AXI4-Stream to Video Out core synchronize the video pipeline to the VTC.

In other words, the VTC is synced to an external source, and the output section of the video pipeline is synced to the VTC, and therefore also synced to the external reference. This configuration is shown in Figure 1-4.

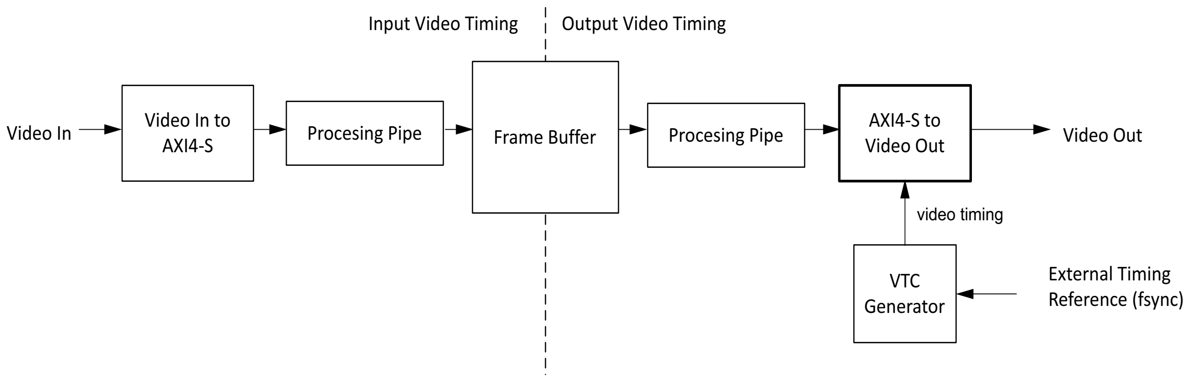


Figure 1-4: **Master Timing Mode Example System**

Feature Summary

The AXI4-Stream to Video Out core converts an AXI4-Stream Slave interface that conforms with the AXI4-Stream Video protocol to a video output, consisting or parallel video data, video syncs, blanks, and data enable. The core interfaces to the Xilinx VTC, which provides timing signals.

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 64-bits, depending on the number of components required for the video format, and the number of bits per component. There is an output FIFO with selectable depth from 32 to 8192 locations.

Applications

- Video output providing parallel, clocked video output
  - DVI
  - HDMI
  - Other clocked, parallel video sinks

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite tools 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 AXI4-Stream to Video Out core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 1] for additional information.

Performance

The following sections detail the performance characteristics of the AXI4-Stream to Video Out 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 in Table 2-1 through Table 2-5 for device-specific information.

Latency

Slave Timing Mode

When the upstream processing block on the AXI4-Stream bus can source data at the pixel rate or faster, the typical latency through the AXI4 –Stream to Video Out core is 9 cycles of vid_out_clk + 3 cycles of aclk.

If the upstream block sources pixels at a slower rate, the FIFO will be used to even out the mismatch in the input and output rates over the course of lines and frames. This storage of pixels in the FIFO will add to the latency and will vary according to the data flow in and out of the core.

Master Timing Mode

In master timing mode, the latency is generally a function of the FIFO size, if pixels are supplied on the AXI4-Stream bus at the output video rate or higher. The latency is relatively steady at approximately Fifo Size - 10 video clocks.

If the upstream block sources pixels at a slower rate, the FIFO is used to balance out the mismatch in the input and output rates over the course of lines and frames. This emptying of the FIFO reduces the latency of the core at the end of lines and or frames, and varies according to the data flow in and out of the core.

Throughput

The average data rates of active pixels on the Video output bus matches the average rate of active pixels in on the AXI4-Stream bus. The data throughput is dictated by the video line standard and clock rate. However, the clock rates of the input need not match the output. Furthermore, since 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 in order to smooth the mismatch in instantaneous rates. Both the AXI4-Stream Clock (Faclk) and the rate of the AXI4-Stream Clock (Faclk) is limited by the overall Fmax.

If the sustained pixel rate available from the upstream AXI4-Stream (Faclk) is equal to or greater than Fvclk, only the minimum buffer size (32 locations) is required. In this scenario, the FIFO will go empty after the EOL on each line.

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

For upstream bandwidth above the line average but below that of Fvclk, the minimum FIFO initial fill level must be:

> FIFO Initial Fill Level = 32+ Thblank/Fvclk + (Thactive – Thblank)*Fvclk/Faclk.
>
> where Thblank is the duration of the horizontal blank and Thactive is the duration of the horizontal active period.

This is the level needed to keep the FIFO from running dry each line. This fill level is automatically established by the output synchronization mechanism, and is not dependent on the Hysteresis Level of the generated core.

The total FIFO depth must be slightly larger than the initial fill level to prevent overflow of the FIFO:

> FIFO depth min = 32 + FIFO initial Fill Level

In every case, this is less than a full line of data, so if a line buffer is provided, it will be sufficient. However, in many cases, a full line buffer is not required.

Resource Utilization

Resource Utilization using ISE Design Suite

The information presented in Table 2-1 through Table 2-5 is a guide to the resource utilization and maximum clock frequency of the AXI4-Stream to Video Out core for Virtex-7, Kintex-7, Artix-7, Virtex-6, and Spartan-6 FPGA families. (Zynq-7000 utilization and Fmax is similar to Artix-7 devices.) This core does not use any dedicated I/O or CLK resources. The design was tested using ISE® v14.3 tools with default tool options for characterization data.

IMPORTANT: *Data width in the tables refers to the aggregate data width of all the video components into and out of the core. For example, RGB data with 8-bits per component has a data width of 24.*

Table 2-1: **Spartan-6**

Data Width	FIFO Depth	LUTs	FFs	RAM 18/9	Fmax (MHz)
8	32	118	132	0 / 0	248
24	1024	174	199	1 / 1	223
64	8192	193	322	33 / 0	204

Table 2-2: **Virtex-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	100	132	0/0	428
24	1024	164	199	1 / 0	377
64	8192	175	306	16 / 1	309

Table 2-3: **Virtex-6**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	107	132	0/0	324
24	1024	157	199	1 / 0	320
64	8192	151	306	16 / 1	274

Table 2-4: **Kintex-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	106	132	0/0	432
24	1024	148	199	1 / 0	352
64	8192	167	306	16 / 1	320

Table 2-5: **Artix-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	100	132	0/0	314
24	1024	137	199	1 / 0	261
64	8192	154	306	16 / 1	258

Resource Utilization using Vivado Design Suite

The information presented in Table 2-6 through Table 2-8 is a guide to the resource utilization and maximum clock frequency of the AXI4-Stream to Video Out core for Virtex-7, Kintex-7, Artix-7, Virtex-6, and Spartan-6 FPGA families. (Zynq-7000 utilization and Fmax is similar to Artix-7 devices.) This core does not use any dedicated I/O or CLK resources. The design was tested using Vivado Design Suite v2012.3 tools with default tool options for characterization data.

Table 2-6: **Virtex-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	96	133	0/0	311
24	1024	92	200	1 / 0	282
64	8192	91	307	16 / 1	184

Table 2-7: **Kintex-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	97	133	0/0	322
24	1024	92	200	1 / 0	301
64	8192	92	307	16 / 1	180

Table 2-8: **Artix-7**

Data Width	FIFO Depth	LUTs	FFs	RAM 36/18	Fmax (MHz)
8	32	94	133	0/0	226
24	1024	94	200	1 / 0	203
64	8192	91	307	16 / 1	170

Core Interfaces

Port Descriptions

The AXI4-Stream to Video Out 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 AXI4-Stream to Video Out core. Not all of the timing signals are required internally by this core, however it also passes these signals out to the Video output. Therefore all timing signals are present. The data enable and vertical sync input are required. It is recommended that the Xilinx Video Timing Controller be set to output all syncs and blanks.

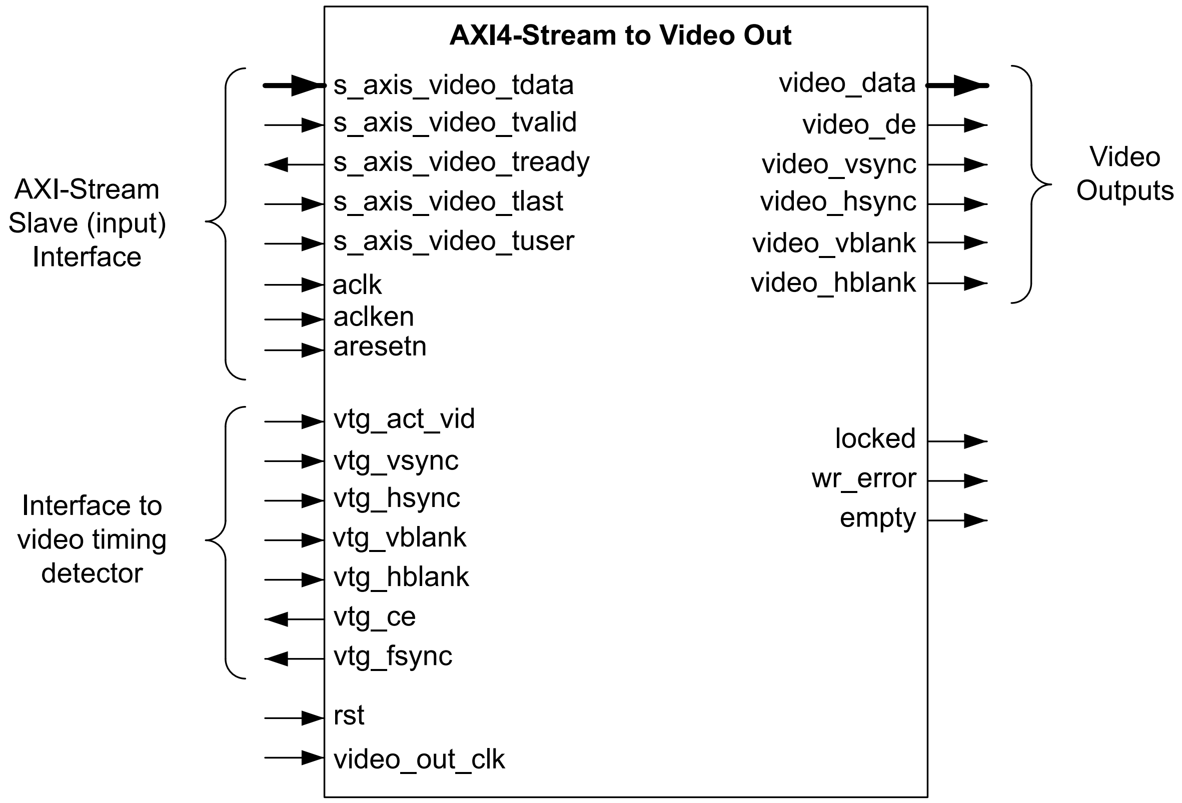


Figure 2-1: **AXI4-Stream to Video Out Core Top-Level Signaling Interface**

Common Interface

Table 2-9: **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. 1 = FIFO write was attempted when FIFO was full.
empty	Out	1	Active HIGH FIFO empty flag. 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.
locked	Out	1	Flag indicating whether the output timing is locked to the output video. 1= locked.

Timing Controller Interface

Table 2-10: **Port Name I/O Width Description**

Signal Name	Direction	Width	Description
vtc_vsync	In	1	VTC vertical sync. Active High
vtc_hsync	In	1	VTC horizontal sync. Active High
vtc_vblank	In	1	VTC vertical blank. Active High
vtc_hblank	In	1	VTC horizontal blank. Active High
vtc_act_vid	In	1	VTC active video signal. 1 = active video, 0 = blanked video
vtc_ce	Out	1	VTC clock enable. Used to halt the timing generator for synchronization purposes.
vtc_fsync	Out	1	Currently not used. Should be left unconnected.

Data Interface

The AXI4-Stream to Video Out core receives video via the AXI4-Stream slave interface defined in the *Video IP: AXI Feature Adoption* section of the (UG761) *AXI Reference Guide* [Ref 1]. This video is output as parallel video data along with timing signals from a VTC.

Video Output Interface

Table 2-11: **Video Output Interface**

Signal Name	Direction	Width	Description
vid_out_clk	In	1	Video clock
video_de	Out	1	Video output data enable. 1 = active video, 0 = blanked video
video_vsync	Out	1	Video output vertical sync. Active HIGH
video_hsync	Out	1	Video output horizontal sync. Active HIGH
video_vblank	Out	1	Video output vertical blank. Active HIGH
video_hblank	Out	1	Video output horizontal blank. Active HIGH
video_data	Out	8-64	Parallel video output data. Active HIGH

AXI4-Stream Signal Names and Descriptions

Table 2-12 describes the AXI4-Stream signal names and descriptions.

Table 2-12: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	8,64	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal

Table 2-12: **AXI4-Stream Data Interface Signal Descriptions *(Cont'd)***

Signal Name	Direction	Width	Description
s_axis_video_tready	Out	1	Input Ready
s_axis_video_tuser	In	1	Input Video Start Of Frame
s_axis_video_tlast	In	1	Input Video End Of Line
ACLK	In	1	Clock
ACLKEN	In	1	Clock Enable
ARESETn	In	1	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 interface must be synchronous to the core clock signal ACLK. AXI4-Stream input 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 (de-asserted) halts the operation 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 de-asserted, 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 de-asserted.

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, for some input data widths, data must be padded with zeros on the MSB to form a module N*8-bit wide vector before connecting to s_axis_video_tdata.

Data on the AXI-4 Stream input s_axis_video_tdata is packed and padded to multiples of 8 bits as necessary. Figure 2-12 shows an example for 12-bit RGB data. Zero padding the most significant bits is only necessary for 10, 12, and 14 bit wide data. The size of the core is affected by the width of the video data and by the amount of padding required.

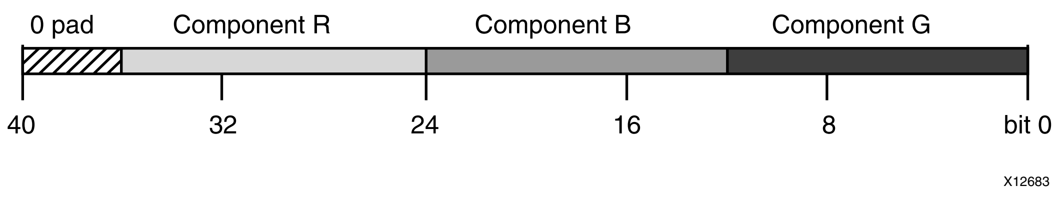


Figure 2-2: **RGB Data Encoding on m_axis_video_tdata**

READY/VALID Handshake

A valid transfer occurs whenever `READY`, `VALID`, `ACLKEN`, and `ARESETn` are high at the rising edge of `ACLK`, as seen in Figure 2-3. During valid transfers, `DATA` only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

Once `s_axis_video_tvalid` is asserted, no interface signals (except `s_axis_video_tready`) may change value until the transaction completes (`s_axis_video_tready`, `s_axis_video_tvalid` `ACLKEN` high on the rising edge of `ACLK`). Once asserted, `s_axis_video_tvalid` may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, `s_axis_video_tvalid` can either be de-asserted or remain asserted to initiate a new transfer.

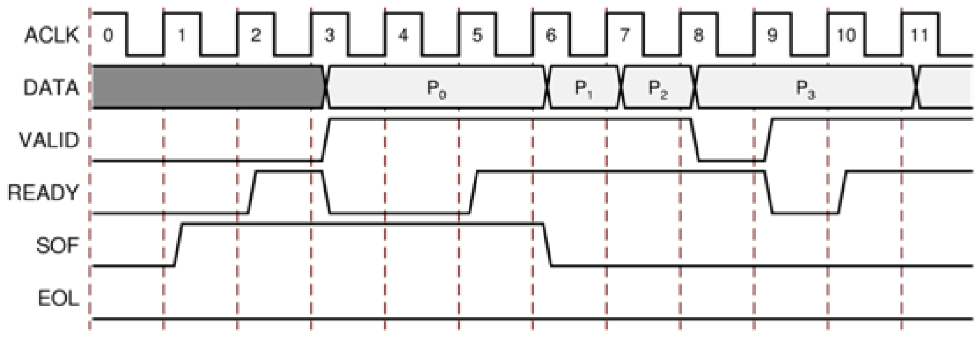


Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

Start of Frame Signals - s_axis_video_tuser

The Start-Of-Frame (`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, as seen in Figure 2-3. 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 `DATA`, as long as a `VALID` is not asserted.

End of Line Signals - s_axis_video_tlast

The End-Of-Line 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 scan-line, as seen in Figure 2-4.

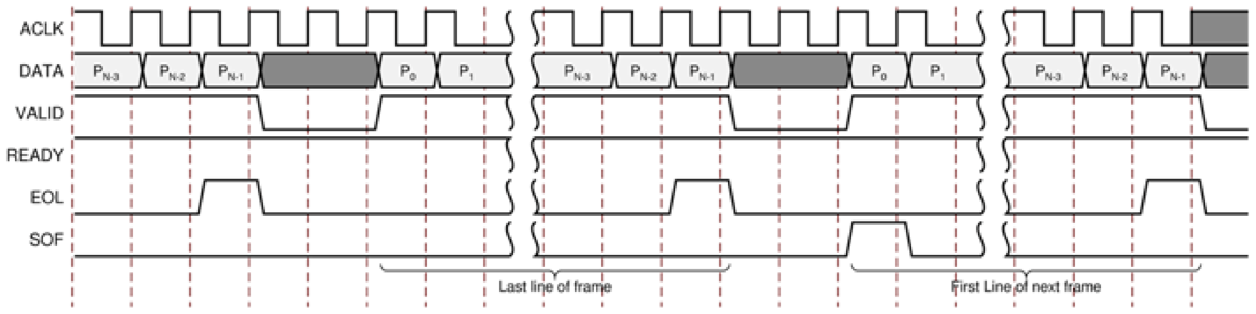


Figure 2-4: **Use of EOL and SOF Signals**

Designing with the Core

General Design Guidelines

The video outputs of the AXI4 Stream to Video Out core should be connected to the output video sink, for example, a DVI interface chip that accepts parallel video data and timing signals. Not all of the timing signals supplied by the Video Timing Controller core are required internally by this core, however it also passes these signals out to the Video output. Therefore all timing signals are present. The data enable and vertical sync input are required. It is recommended that the Xilinx Video Timing Controller be configured to output all syncs and blanks (horizontal sync, horizontal blank, vertical sync, vertical blank, and active video).

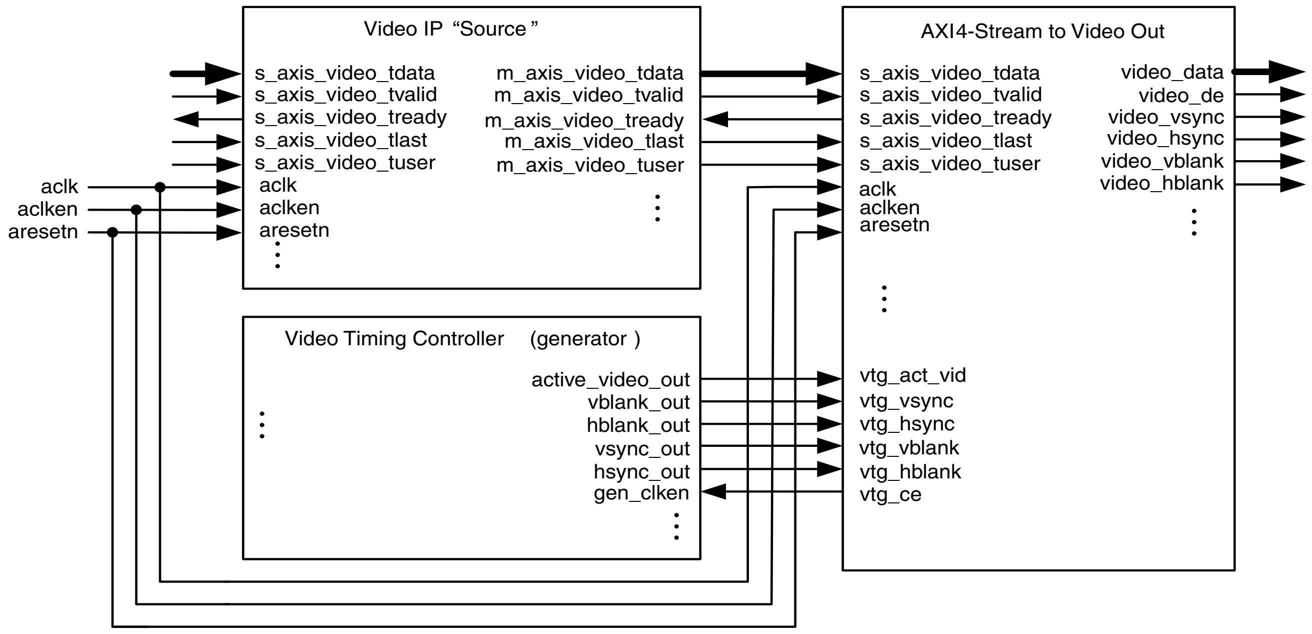


Figure 3-1: **Example of ACLK Routing and AXI4-Stream Interconnect**

The video data input of the core is a slave AXI-4 Stream Video Protocol interface, that connects to upstream 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 Figure 3-1 the AXI4-Stream to Video Out core is used in conjunction with the Video Timing Controller which generates the video timing used for the video output. The timing parameters for the Video Timing Controller must match the video line standard (active pixels per line and active lines per frame) of the video data on the AXI4-Stream interface. Otherwise, it is impossible for the output_sync section to lock the VTC timing to the incoming video. Also, the video clock must be synchronous with the original source of the video on the AXI4-Stream bus. That is, both the AXI4-Stream video and the video timing controller must be roughly isochronous such that the frame rates are identical, with no long term drift.

Clocking

There are two clocks used in this core.

- Video output pixel clock
- AXI4-Stream clock

The video output clock corresponds to the video line standard used on the output. It is part of the video line standard definition, and is used by both the AXI4-Stream to Video Out core and by the corresponding Video Timing Controller core that is used to generate 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 output clock. This clock can be slower than the video clock, in which case, additional buffering is required to store pixels so that lines can be output 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.

ACLKEN

The `ACLKEN` pin disables the AXI4-Stream side of the asynchronous FIFO, as seen in Figure 3-1.

The `ACLKEN` pin facilitates:

- Multi-cycle path designs (high speed clock division without clock gating),
- Standby operation of subsystems to save on power
- Hardware controlled bring-up of system components

IMPORTANT: *When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, to prevent transaction errors the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal (Figure 2-2).*

IMPORTANT: *When two cores connected via AXI4-Stream interfaces, where only the master or the slave interface has an ACLKEN port, which is not permanently tied high, the two interfaces should be connected via the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).*

Resets (ARESETn and rst)

In general, the core does not need to be reset during normal operation. The core will continuously and automatically attempt to lock the video output to the incoming AXI4-video stream until successful. The state machine in the output synchronizer block detects when the video is not locked to the AXI4-Stream. It resets the FIFO and starts an initialization sequence. Whenever it is detected that the output is not locked, the reset and initialize sequence is repeated.

There are two external resets provided: rst, which resets the entire core, and aresetn, which resets the AXI4-stream interface. Both resets cause the FIFO to be reset. The rst additionally causes the state machine of the out_sync module to be reset to a condition that is ready to attempt to lock on to the incoming AXI4-Stream. 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 data rate between the upstream processor via the AXI4-Stream clock (`aclk`) and the video output clock (`video_out_clk`) frequency, and also the line standard being used. Basically, if the upstream AXI4-Stream source cannot provide data at a sustained rate equal to or greater than the video clock rate, then additional buffering is required to store up incoming pixels ahead of when they are required such that a continuous stream of pixels can be generated at the output without emptying the FIFO prematurely. The AXI4-Stream to Video Out core will accept AXI4-Stream data as soon as it is available (i.e. it will not apply back pressure) until its FIFO is almost full.

If the sustained pixel rate available from the AXI4-Stream clock (Faclk) is equal to or greater than Video Output pixel clock (Fvclk), only the minimum buffer size (32 locations) is required. In this scenario, the FIFO will go empty after the EOL on each line.

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

For upstream bandwidth above the line average but below that of vclk, the minimum FIFO initial fill level must be:

FIFO Initial Fill Level = $32 + T_{hblank}/F_{vclk} + (T_{hactive} - T_{hblank})*F_{vclk}/F_{aclk}$.

where Thblank is the duration of the horizontal blank and Thactive is the duration of the horizontal active period.

This is the level needed to keep the FIFO from running dry each line. This fill level is automatically established by the output synchronization mechanism, and is not dependent on the Hysteresis Level of the generated core.

The total FIFO depth must be slightly larger than the initial fill level to prevent overflow of the FIFO:

FIFO depth min = 32 + FIFO initial Fill Level

In every case, this is less than a full line of data, so if a line buffer is provided, it will be sufficient. However, in many cases, a full line buffer is not required.

Additional Buffering Requirements for Master Timing Mode

In master timing mode, the FIFO must be large enough to accommodate pixels that arrive on the AXI4 Stream bus prior to their being output on the video bus. In this mode, the VTC cannot be halted. If the timing between the video source and the video output is tightly controlled then additional buffering may not be required. If, however, video data on the AXI4-Stream bus leads the video timing signals by many clock cycles, the FIFO must be sized to handle the storage of the incoming pixels until they are needed at the output, without overflowing the FIFO.

Module Descriptions

As shown in Figure 1-1, the AXI4-Stream to Video Output core works with the generator portion of the Video Timing Controller (VTC) core.

The timing outputs of the VTC connect as inputs to the AXI4-Stream to Video Output core. When locked, these timing signals are output along with video data from the FIFO as part of the video interface. These signals are also used by the Output Synchronizer, to compare to flags from the FIFO in order to guide the control action for locking the timing generator with FIFO data.

Figure 1-1 shows a block diagram of the AXI4-Stream to Video Out core. There are three main blocks, the stream coupler, the data formatter, and the output synchronizer. The AXI4-Stream interface is on the left, and the video connections are on the right.

Stream Coupler

The Stream Coupler block consists mainly of an asynchronous FIFO and write logic for the input side of the FIFO. Reading of the FIFO is controlled by the Output Synchronizer. 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 are dependent on the ratio of the AXI4-Stream data rate to the video clock rate. This is described further in System Considerations in Chapter 3.

Asynchronous FIFO

The crossing of clock domains dictates that this be an asynchronous FIFO. The FIFO designed for this core has two important distinguishing features:

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

This asynchronous nature of the FIFO presents challenges for the fill level indicators and the flags. The chief risk is that pointer values could glitch when being sampled from one clock domain to another. Thus, there must be pointer synchronization across clock domains, and there must be a handshaking protocol to ensure that all pointer updates are seen even when the clock rate in the two domains are radically different. Figure 3-2 is a block diagram of the asynchronous FIFO.

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

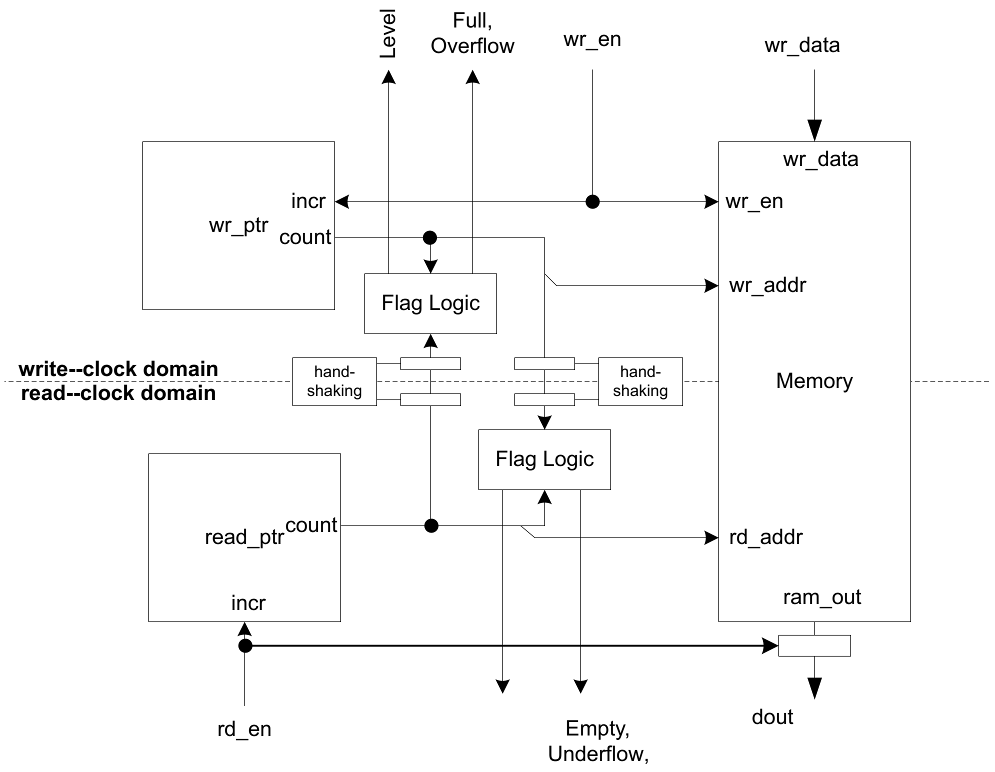


Figure 3-2: **Block Diagram of Bridge Core Asynchronous FIFO**

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 via handshaking. The main reason is that 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 3-3. The first two register delays are required to resolve metastability. The third is to ensure that the register has time to take the data before the handshaking is returned. Otherwise if the clock in one domain were several times faster than the other, there is a chance that the handshaking could be returned, before the pointer register is updated in the slower clock domain.

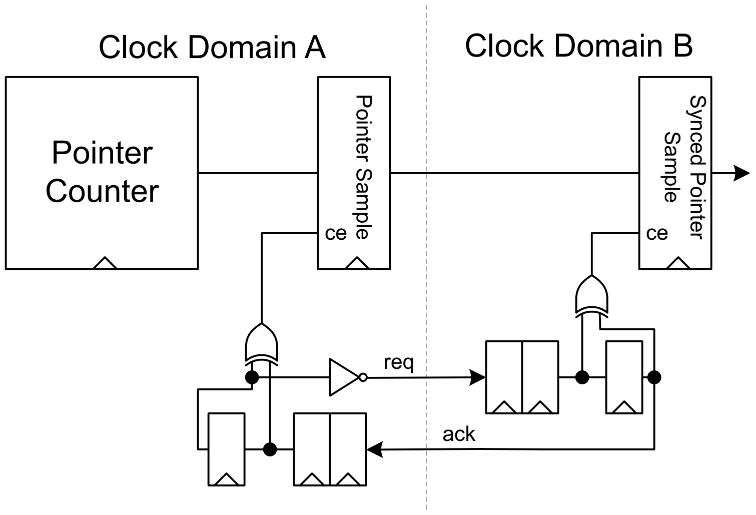


Figure 3-3: **Synchronization and Handshaking for Clock Domain Crossing of Pointers**

The extra delay for turning around the handshaking signal ensures that data will be transferred reliably no matter what the relative clock rates. i.e. if Fa > 2Fb each "req"is still guaranteed to update the sync pointer in domain B. By the same token, if Fb > 2Fa, each ack is guaranteed to update the pointer sample in domain A. Figure 3-4 shows an example of pointer synchronization between clock domains, and the handshaking scheme shown in Figure 3-3. This handshaking scheme, utilizes two states: request and acknowledge. The "request" state is when Req and Ack are not equal, and "acknowledge" is when they are equal.

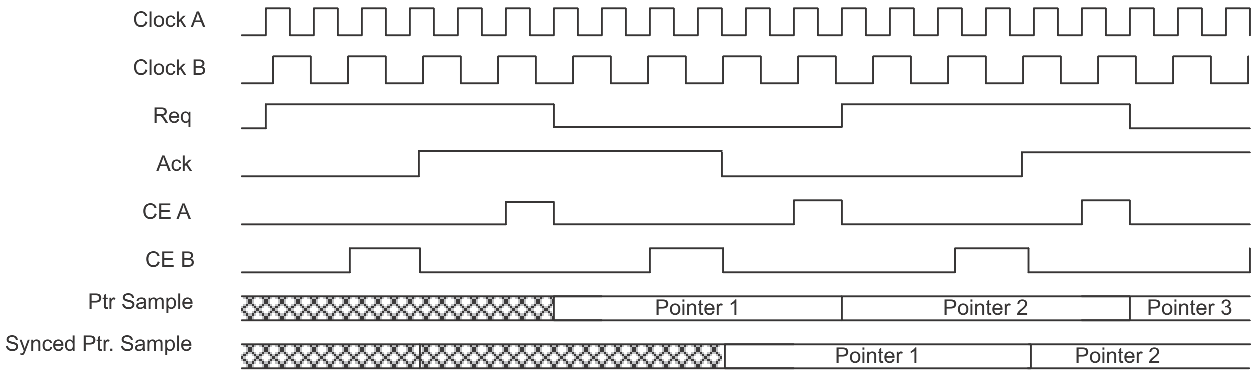


Figure 3-4: **Waveform Diagram of Handshaking and Clock Domain Crossing of Pointers**

This sample and hold method with handshaking delays the capture of the pointers by several clock edges in each domain but the pointer transfers are always reliable and glitch free. The latency of the pointers causes pessimism in the level outputs, and the flags. That is, the empty flag will persist in the read domain for several clocks after a write has occurred, etc. Also, the level output will not necessarily be monotonic. This is taken into account in the design of the bridges by providing a small "cushion" or minimum fill level in the FIFOs so that the pointer pessimism does not cause artifacts.

Underflow Prevention

Besides synchronizing flags, it is not enough to signal an empty condition, because "reads" of the FIFO will not stop on empty. Additional read operations will be performed in order to get the EOF to the output of the FIFO. This will happen routinely, and the FIFO must not lose any data. That is, underflow is not allowed. When new data is eventually written to the FIFO, reading must pick up with the first new valid pixel. To do this, the read pointer is inhibited when the FIFO is empty.

The empty flag asserts coincident with the last available location being clocked into the output register. The read pointer, however is not advanced to match the write pointer but remains pointing at the last valid pixel that was read. Subsequent reads, when empty is asserted, will cause the read_error flag to be asserted, flagging the pixel from the FIFO as invalid. The data output, however will not change.

Thus, when a read occurs to an empty FIFO, the invalid flag (read_error) is set, and the read pointer does not increment. In this way, 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 extra "revolution" bits on the pointers in addition to the address bits as shown in Figure 3-5.

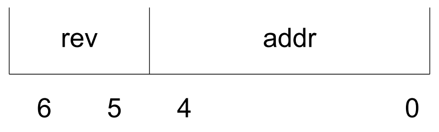


Figure 3-5: **Pointer Format for a 32 Location FIFO**

At the cost of a couple of extra bits in the pointers, this allows the level calculations and flags to be unambiguously determined in full and empty conditions.

Write Logic

The function of the write logic is to control the handshaking for the AXI4-Stream bus and to accept pixels from this bus as rapidly as possible. In general, the strategy for Video Over 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 AXI4-Stream to Video Out core is, by definition, at the end of the pipeline, it strives to fill its FIFO as fast as possible.

The Write Logic controls the tready handshaking signal based on the full flag from the FIFO. Whenever there is room in the FIFO, tready is asserted. When tvalid becomes active while tready is asserted, the FIFO is written. Thus, the tready is asserted except when there is almost no space available in the FIFO. 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 and thus no pixels are entering the FIFO.

At the end of each line, the EOL must be flushed through from the FIFO to the video output of the core. This is done so that the complete line of video can be output while input to the core may be stalled, waiting for pixels from the next line.

This flushing requirement presents a challenge since it is likely to happen when no data is coming into the FIFO. It requires generation of invalid pixels to flush out the valid pixels. These invalid pixels must be swallowed by the core prior to the start of the next active line on the output.

When the output synchronizer is locked, flushing of the EOL is accomplished by reading from the FIFO based on the active video flag from the VTC. In order to propagate the EOL pixel through to the output, this happens even if the FIFO is empty. 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 pixel will continue to propagate to the output. When new valid pixels are again available in the FIFO, they are read out to the output register during the blanking period.   In this way, invalid pixels are swallowed before the next active line begins.

Data Formatter

The Data Formatter block receives data from the Stream Coupler, timing from the VTC, and control from the Output Synchronizer. It registers the video data and timing signals to form the video output interface. The locked signal from the output synchronizer is used to enable the video interface. Before lock is achieved, the video outputs are all forced low. When lock is achieved, the video outputs become active at the end of the frame in progress, noted by the rising edge of `vblank`.

Output Synchronizer

A major task for the AXI4-Stream to Video Out core is to synchronize the asynchronous and irregularly timed video data from AXI4 Stream interface with the periodic and repeatable timing signals from the VTC. In slave timing mode, the timing of the data in the pipeline on the AXI-Stream interface cannot be controlled. Therefore, the synchronization is done mainly by controlling the phase of the VTC. In master timing mode, the VTC controls the timing of data in the pipeline by providing the timing reset signal, Fsync, to the upstream video source. The synchronization is done exclusively by controlling when data exits the FIFO.

In either case, the synchronization function is performed by the Output Synchronizer block in Figure 1-1. Also, the data from the AXI4-Stream interface goes into a FIFO to be able to smooth out small and short-term latency variations that occur during the course of a frame. The phase of the VTC must have some additional lag, such that the FIFO will have a sufficient fill level to cushion the short-term variations.

For Slave timing mode, this cushion is controlled by the AXI4-Stream to Video Out core. For master timing mode, the VTC Fxync controls this cushion. The Fsync must lead the video timing signals sufficiently to allow for data to be propagated through the processing pipeline to the AXI4-Stream to Video Out core. However, it must not lead so much that the FIFO overflows when synchronizing the incoming AXI4-Stream video to the video timing signals from the VTC generator.

Since the video input source may be disconnected or change line standards, the Video Output Bridge must constantly compare the timing of incoming video data relative to the timing of the VTC. This makes it possible to detect timing mismatches so that it can re-synchronize after any type of interruption. Thus, it must be self monitoring and self synchronizing. Figure 3-10 shows a block diagram of the output synchronizer.

Algorithm

For either Slave or Master timing mode, the operation of the self synchronizing circuit can be broadly defined in two steps: initialization and run. In the run mode, alignment must be continually checked and if there is a discrepancy, a new initialization is required, as shown in Figure 3-6.

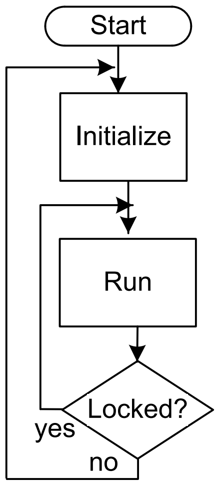


Figure 3-6: **Basic Self Synchronizer Flow Chart**

The initialization phase does an alignment of the VTC to the AXI4-Stream data, provides for some cushion in the FIFO, and precisely aligns the video data from the FIFO to the VTC generated video output signals. The main difference between slave and master timing mode is how the FIFO and/or VTC is used to align the pixels with the video timing signals. In slave timing mode, the output sync section controls both the reading of the FIFO and the VTC (by halting it). In master timing mode, the VTC generator is never halted. Only the reading of the FIFO is controlled by the output synchronizer module.

Algorithm Differences Between Slave and Master Timing Modes

The remainder of the discussion gives the specifics of the algorithm for slave timing mode. The master timing mode algorithm follows the same pattern except that there is no step to set the FIFO cushion in the initialization. Also, in master timing mode, the VTC is never halted. All alignment is done through control of the FIFO. The details of algorithm for this mode is shown in the state machine diagram in Figure 3-12.

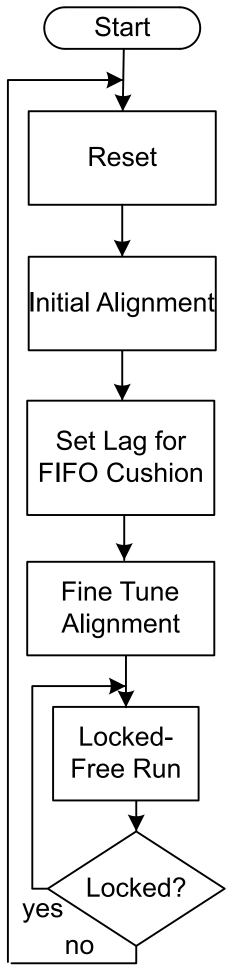


Figure 3-7: **Self Synchronizer Flow Chart Showing Initialization Steps - Slave Timing Mode**

Figure 3-7 shows the initialization steps. The initial alignment of VTC generated video output signals to the FIFO is done by letting the VTC free run until it reaches the point of the first valid pixel of the first line of the frame (SOF). At this point, the VTC is paused using the clock enable of the VTC. The FIFO is read as fast as possible, keeping it essentially empty, and the start of field (SOF) flag is monitored. When VTC SOF is asserted, the FIFO data and the VTC generated video output signals are roughly aligned. This is the initial alignment.

At this juncture, the FIFO is paused for some additional period, nominally about 16 clocks, such that when the FIFO output data and VTC generated video output signals are aligned, the FIFO has some pixels stored to cushion any short term variations in the input data rate.

With a rough alignment, and a cushion set for the FIFO, The last step is to fine tune the FIFO output/VTC generated video output signals alignment such that the end of line (EOL) flags read from the FIFO precisely match the last clock of Data Enable from the VTC before the horizontal blanking period. The fine tuning process is shown in more detail in Figure 3-8.

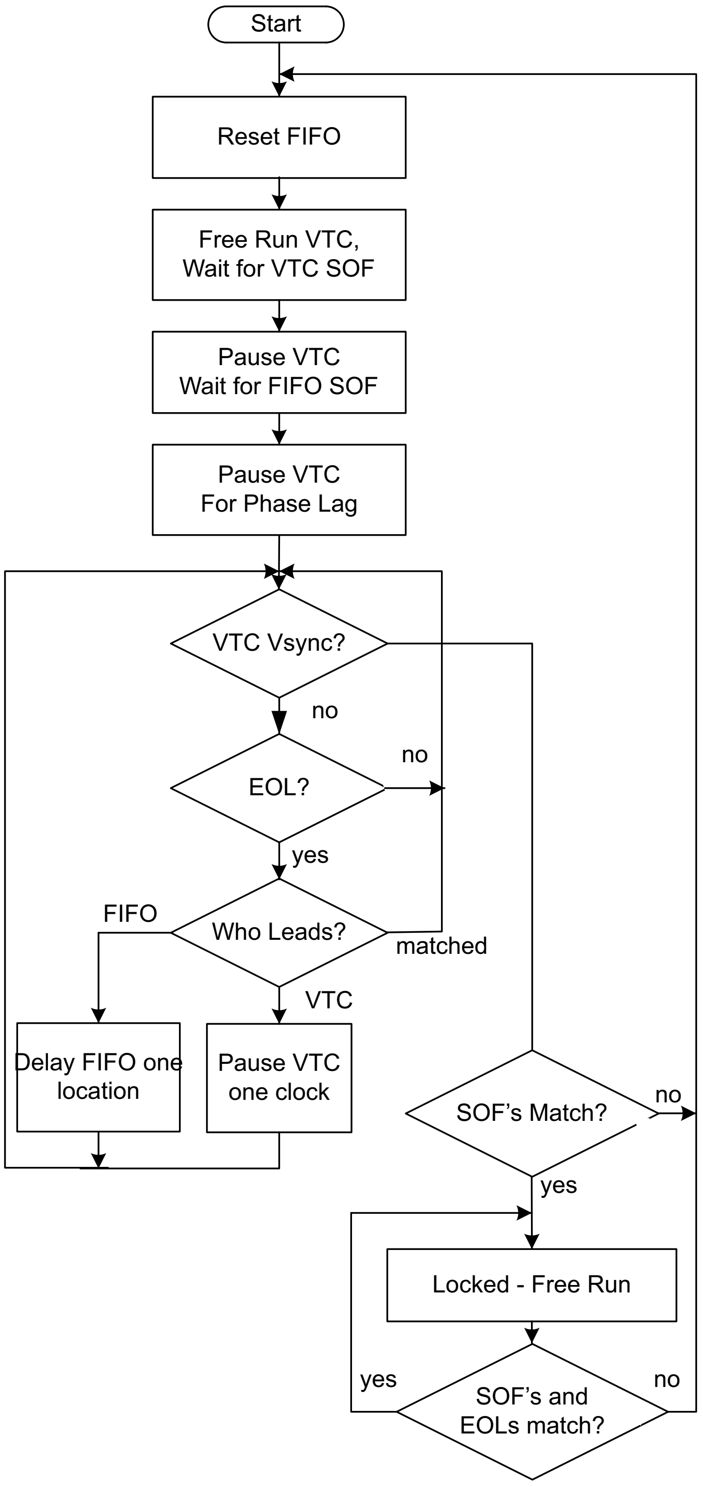


Figure 3-8: **Self Synchronizer Flow Chart Showing Fine Tuning Process - Slave Timing Mode**

During the first frame time after rough alignment, the EOLs from the FIFO and from the VTC (or rather the EOL signal derived from Data Enable) are compared every line. If the VTC is leading, it is paused by one clock. If the FIFO is leading, the reading of the FIFO is delayed by one location. In this manner, the EOLs will be aligned after several lines, typically a dozen or two. Typically, the FIFO will be leading because the VTC has an additional pause after rough alignment, but due to startup idiosyncrasies, this is not guaranteed. Delaying the FIFO read has the effect of building up a cushion in the FIFO. Even though the FIFO delay is exercised most often, the VTC delay must be included to assure reliable synchronization in every situation.

At the end of the frame, the EOL's are aligned via the fine tuning process, and there is a cushion of pixels in the FIFO. However, since the FIFO must empty completely at the end of each line in order to flush the EOL through to the video output, This cushion will be used up at the end of each line. Nevertheless, the phase lag of the VTC generated video output signals ensures that the cushion is built up at the start of the next line such that it can smooth out variations during the active line.

When the Vsync from the VTC becomes active, the EOLs from the FIFO and the VTC are aligned, and if everything is stable, the SOF's will also be aligned. If the SOF's are aligned, then the locked mode is entered. In this mode, the VTC free runs, and the FIFO is read as dictated by the VTC. The video output is enabled at the next rising edge of `vblank` such that pixel data, and timing signals are output from the bridge. This is the run state, that can continue indefinitely.

The EOL's and SOF's are continuously compared while the output is locked. If they ever do not match, the system is considered unlocked and a new initialization cycle is initiated. In the case of a disconnected or unstable input to the system, initialization cycles will be repeated until the system is stable.

Figure 3-9 shows the flow charts in previous figures and how the stages correspond in the various levels of detail.

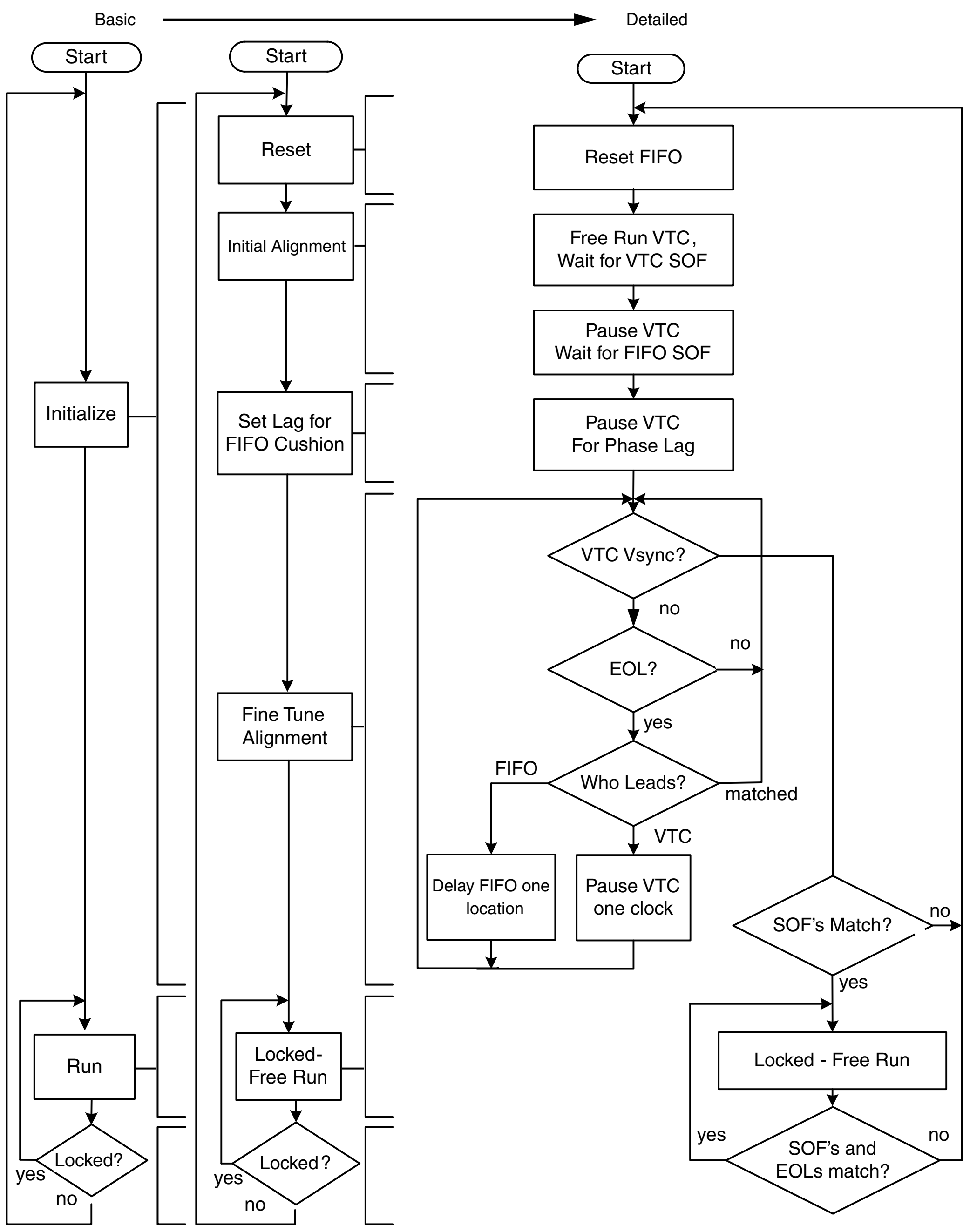


Figure 3-9: **Flow Charts from Basic to Detailed Showing Equivalencies - Slave Timing Mode**

Implementation

The output synchronizer is shown in Figure 3-10. This is essentially a state machine that implements the algorithm described in the previous section. The control outputs are all registered, and reflect the current state. That is, the control outputs are pre-decoded, rather than being decoded based on the current state.

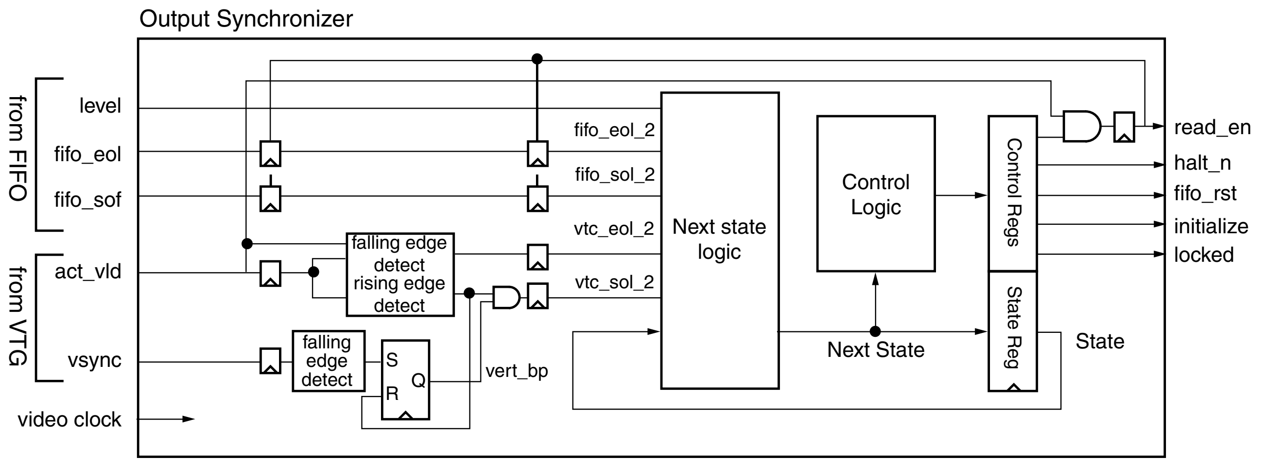


Figure 3-10: **Output Synchronizer Block Diagram**

Besides the state machine, there is logic to create virtual EOL and SOF flags from the VTC generated video output signals. These flags are created based on edges of the act_vid (a.k.a data enable or DE) from the VTC. The vsync input is used to distinguish the SOF. Also there is additional logic to create the FIFO read_en signal based on the act_vid from the VTC as well as control bits from the state machine. The read_en output has a register after the state machine to ensure that all outputs of the module are registered.

Slave Timing Mode

Figure 3-11 is a state diagram of the state machine of the output synchronizer for slave timing mode. Figure 3-12 is a state diagram of the output synchronizer state machine for master timing mode. They both execute the initialize-run algorithm described earlier in this document. Initialization consists of a coarse alignment, setting additional lag, and fine tuning. When the EOL's are aligned, and matching SOF's occur, the state machine goes to the "EOL's Matched" state. If matching SOF's occur a second time, the "Locked" state is entered, which concludes initialization. It will remain in this state as long the system is stable.

The initialization process is done over three frame times. The first is waiting for the FIFO SOF. The second is for the fine tuning to align EOL's. The third is basically a double check that the EOL's and SOF's are aligned. If an SOF mismatch occurs during fine tuning, it means that fine tuning has failed to align within the frame time. In this case, the initialization process is re-started. Also, in the "EOL's Matched" or "Locked" states, EOL and SOF matching is continuously checked. If there is any mismatch, it means the output is unlocked, and the initialization is re-started.

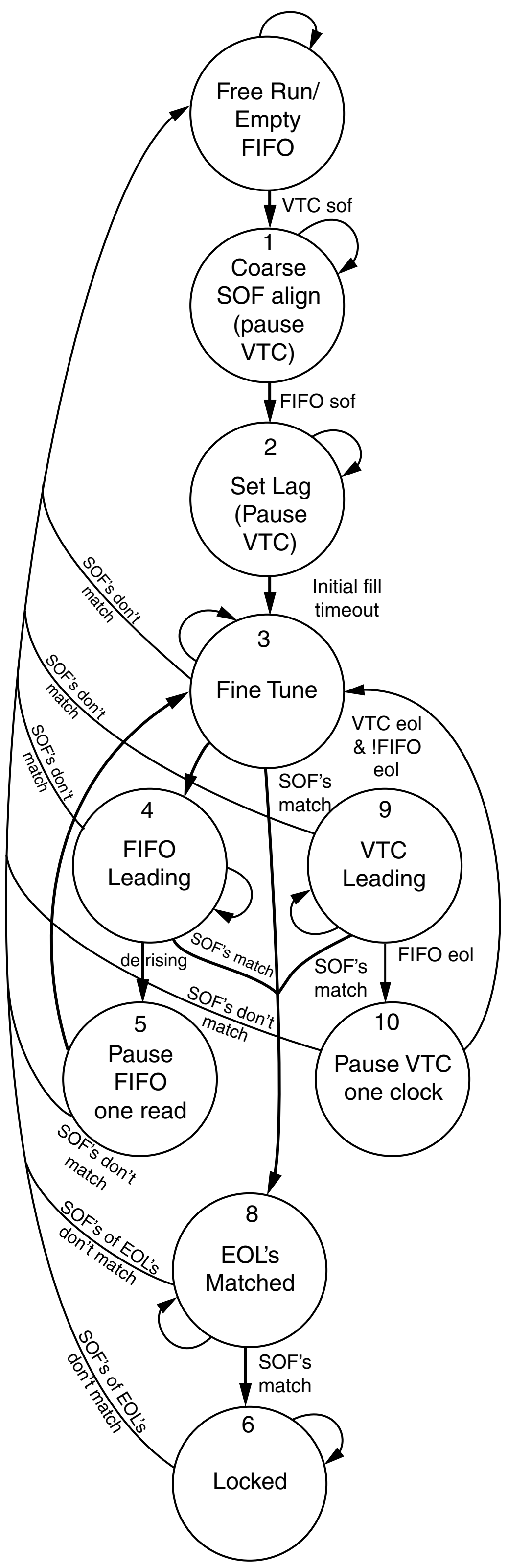


Figure 3-11: **Output Synchronizer State Diagram - Slave Timing Mode**

Figure 3-12 is a state diagram of the state machine of the output synchronizer for master timing mode.

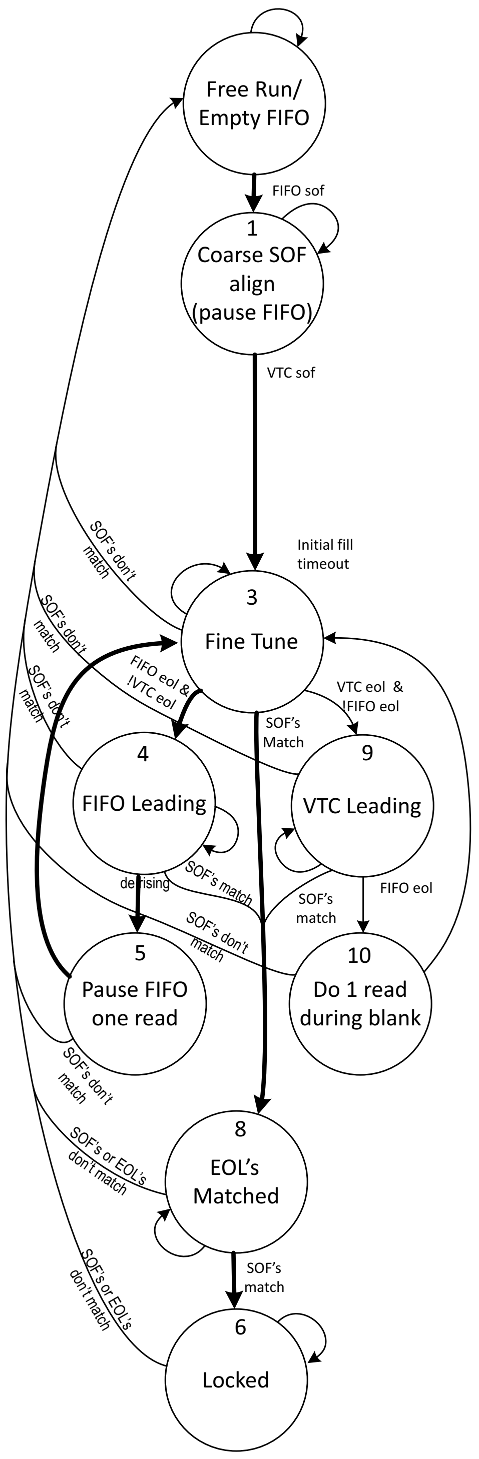


Figure 3-12: **Self Synchronizer Flow Chart Showing Initialization Steps - Slave Timing Mode**

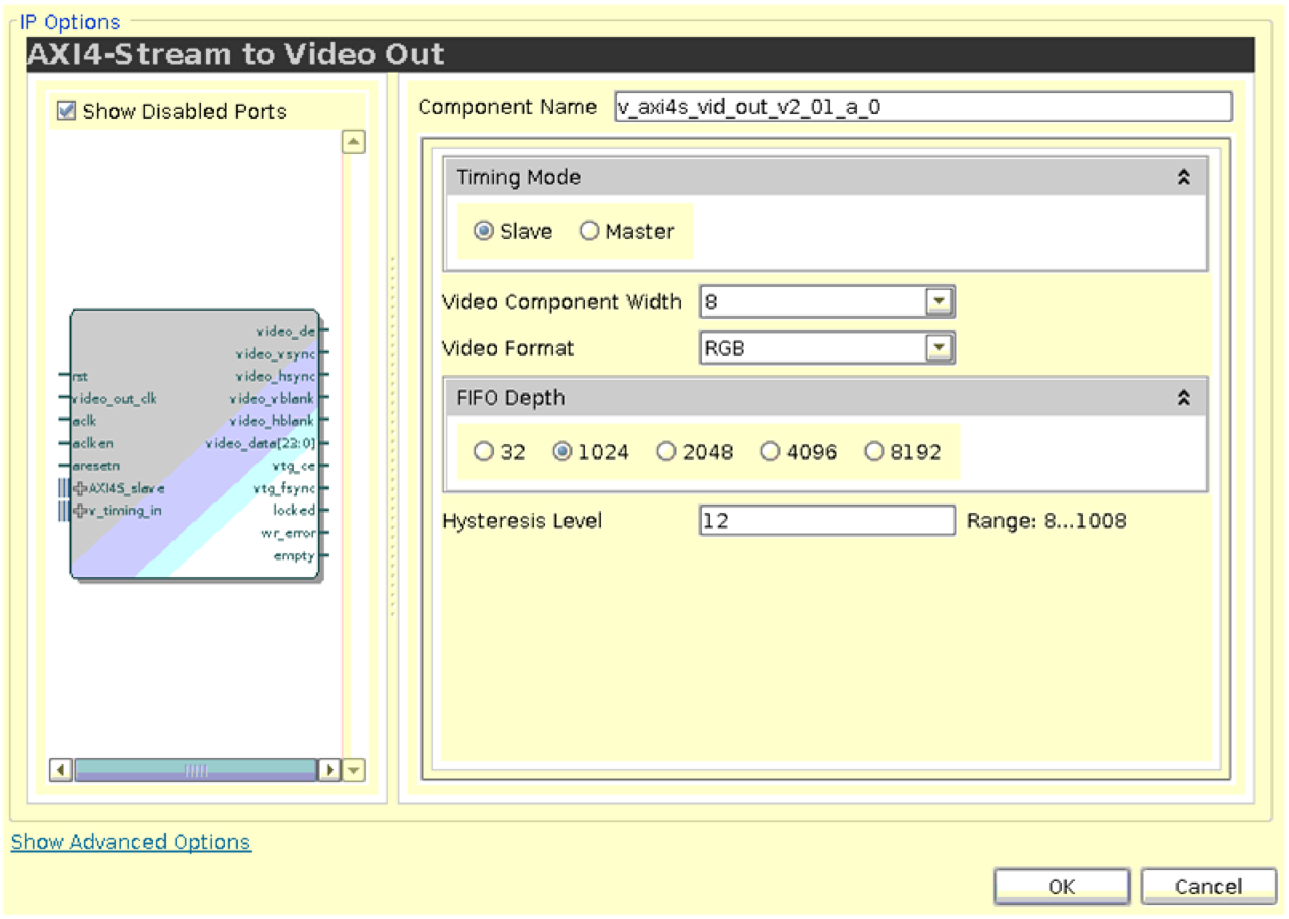
SECTION II: VIVADO DESIGN SUITE

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.

GUI

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



[✓ Show Disabled Ports]

Component Name: v_axi4s_vid_out_v2_01_a_0

Timing Mode: [✓ Slave] Master

Video Component Width: 8

Video Format: RGB

FIFO Depth: 32 [✓ 1024] 2048 4096 8192

Hysteresis Level: 12 Range: 8...1008

Figure 4-1: **AXI4-Stream to Video Out 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 from characters: a to z, 0 to 9 and "_".
- **Timing Mode**: Indicates whether the attached VTC timing generator is the timing slave (gets synchronized to AXI4-Stream video) or the timing master (dictates timing for the upstream video pipeline and the video output).
- **Component Width**: Specifies the bit width of input samples. This is used in conjunction with the Video Format to determine the width of the input AXI4-Stream data interface, `s_axis_video_tdata`, and the output video bus, `vid_data`.
- **Video Format**: Specifies the video format used. The video formats are specified in the *Video IP: AXI Feature Adoption* section of the (UG761) *AXI Reference Guide* [Ref 1]. The format selected determines the number of components used. The number of components (1-4) is multiplied by 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 10 and the Video Format is YUV 4:2:2 (2 components), the `vid_data` will be 20 bits wide and `s_axis_video_tdata` will be 24 bits.
- **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. This number should generally be in the 8-20 range, and must be at least 16 less than the depth of the FIFO.

Output Generation

When the core is generated, the source Verilog files are delivered. The source files are the same for both synthesis and simulation. Wrapper files with parameter settings and instantiation template files are also delivered.

Constraining the Core

Required Constraints

The only constraints required are clock frequency constraints for the video clock, `video_clk_out`, 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.

For example, the following XDC code that could be used to constrain the video clock and the AXI4-Stream Clock for 150MHz operation:

```
create_clock -add -name video_out_clk -period 3.333 [get_ports video_out_clk]
create_clock -add -name aclk          -period 3.333 [get_ports aclk]
set_max_delay -from [get_clocks {video_out_clk}] -to [get_clocks {aclk}]
-datapath_only 3
set_max_delay -from [get_clocks {aclk}]          -to [get_clocks {video_out_clk}]
-datapath_only 3
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. For a complete listing of supported devices, see the release notes for this core.

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 handshaking 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.

Detailed Example Design

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 per se, but rather, 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 video output. The resulting video output bus is interfaced to an AXI4 Stream slave emulator. The data extracted by the emulator are compared to the expected parallel video data.

Test Bench Structure

Figure 6-1 shows the test bench structure:

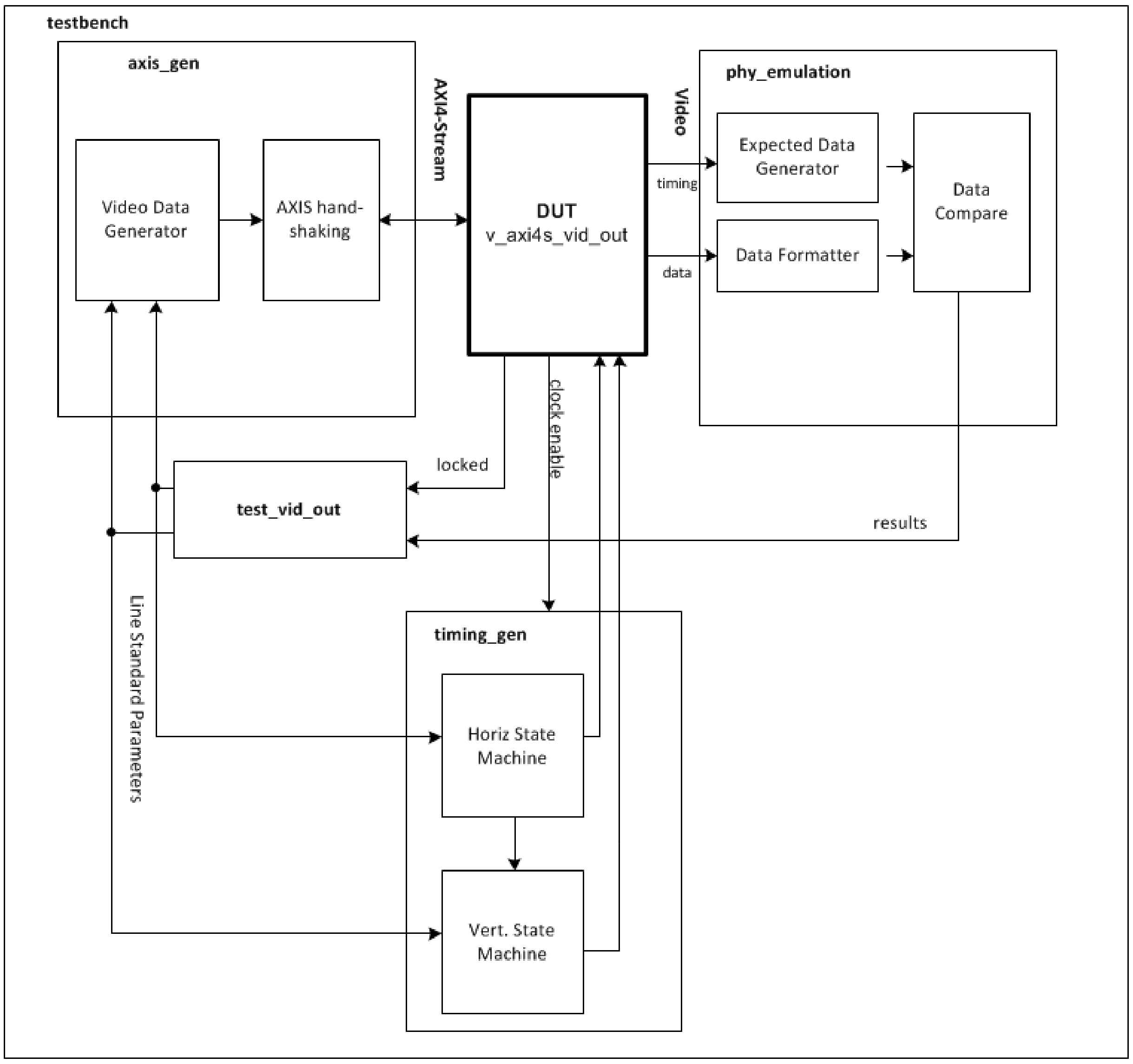


Figure 6-1: **Test Bench Structure**

The top-level entity ("testbench" in the **`cust_tb_v_vid_in_axi4s.v`**) file instantiates the following modules:

- DUT

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

- axis_gen

  AXI4-Stream generator. This module generates the video timing based on the parameters specified by the test program. It also emulates the AXI4-Stream master bus which interfaces to the DUT.

- timing_gen

The timing generator module generates the video timing based on the parameters specified by the test program. This emulates the function of the Video Timing Controller.

- test_vid_out

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

- phy_emulation

  The video PHY emulator simulates the video PHY interface driven by the DUT. It generates handshaking, and receives data from the core. It also generates expected values and compares these to incoming video data. Based on the syncs and blanks from the DUT, it creates an expected data value, and compares this to the incoming video.

Running the Simulation

The simulation source files for the core are the same verilog files delivered for synthesis. The test bench file is under the top level directory. The test bench file is `cust_tb_v_vid_in_axi4s.v`. This file contains all of the modules required for the test bench. The top level module name is "testbench". (UG626) *Synthesis and Simulation Design Guide* provides details on setting up and running simulations on Xilinx simulation tools.

When the test bench module is run, it produces the stimulus, applies it to the device under test, and prints out the line-standards run and the pass/fail results.

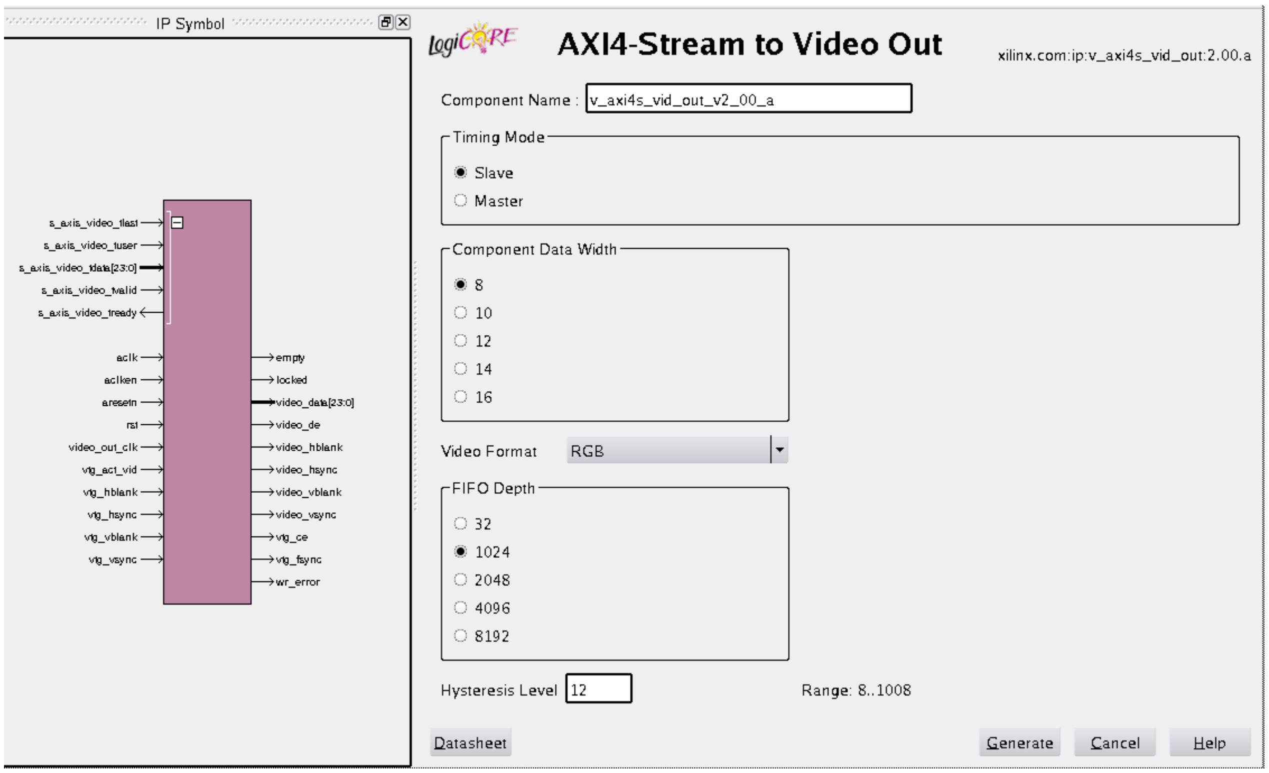
SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

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

GUI

The Xilinx AXI4-Stream to Video Out core is easily configured to meet the developer's specific needs through the CORE Generator™ GUI. This section provides a quick reference to parameters that can be configured at generation time.



AXI4-Stream to Video Out

xilinx.com:ip:v_axi4s_vid_out:2.00.a

Component Name : v_axi4s_vid_out_v2_00_a

Timing Mode
- [✓ Slave]
- Master

Component Data Width
- [✓ 8]
- 10
- 12
- 14
- 16

Video Format: RGB

FIFO Depth
- 32
- [✓ 1024]
- 2048
- 4096
- 8192

Hysteresis Level: 12 Range: 8..1008

Figure 7-1: **AXI4-Stream to Video Out CORE Generator 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:

- **Timing Mode**: Indicates whether the attached VTC timing generator is the timing slave (gets synchronized to AXI4-Stream video) or the timing master (dictates timing for the upstream video pipeline and the video output).
- **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 from characters: a to z, 0 to 9 and "_".
- **Component Width**: Specifies the bit width of input samples. This is used in conjunction with the Video Format to determine the width of the input AXI4-Stream data interface, `s_axis_video_tdata`, and the output video bus, `vid_data`.
- **Video Format**: Specifies the video format used. The video formats are specified in the *Video IP: AXI Feature Adoption* section of the UG761 AXI Reference Guide. The format selected determines the number of components used. The number of components (1-4) is multiplied by 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 10 and the Video Format is YUV 4:2:2 (2 components), the `vid_data` will be 20 bits wide and `s_axis_video_tdata` will be 24 bits.
- **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. This number must be at least 16 less than the depth of the FIFO.

Output Generation

When the core is generated, the source Verilog files are delivered. The source files are the same for both synthesis and simulation. Wrapper files with parameter settings and instantiation template files are also delivered.

Constraining the Core

Required Constraints

The only constraints required are clock frequency constraints for the video clock, `video_clk_out`, 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.

For example, here is some UCF code that could be used to constrain the video clock and the AXI4-Stream Clock for 150MHz operation:

```
NET vid_out_clk TNM_NET = vid_out_clk;
TIMESPEC TS_vid_out_clk = PERIOD vid_out_clk 150 MHz HIGH 50%;
NET aclk TNM_NET = aclk;
TIMESPEC TS_aclk = PERIOD aclk 150 MHz HIGH 50%;

TIMESPEC TS_video_out_clk_to_aclk = FROM video_out_clk TO aclk 3 NS DATAPATHONLY;
TIMESPEC TS_aclk_to_video_out_clk = FROM aclk TO video_out_clk 3 NS DATAPATHONLY;
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. For a complete listing of supported devices, see the release notes for this core.

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 handshaking 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.

Detailed Example Design

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 per se, but rather, 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 video output. The resulting video output bus is interfaced to an AXI4 Stream slave emulator. The data extracted by the emulator are compared to the expected parallel video data.

Test Bench Structure

Figure 9-1 shows the test bench structure:

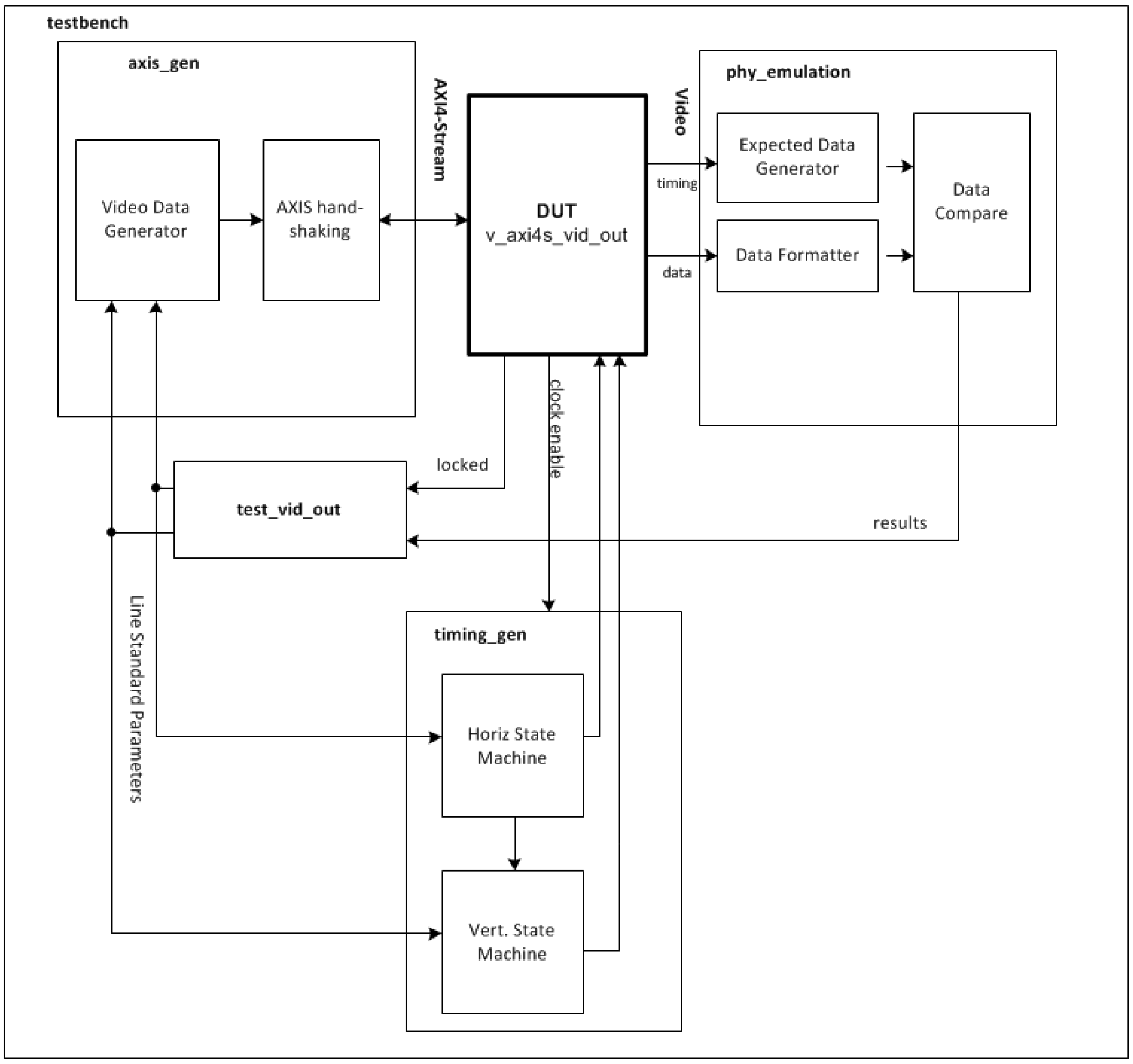


Figure 9-1: **Test Bench Structure**

The top-level entity ("testbench" in the **`cust_tb_v_vid_in_axi4s.v`**) file instantiates the following modules:

- DUT

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

- axis_gen

  AXI4-Stream generator. This module generates the video timing based on the parameters specified by the test program. It also emulates the AXI4-Stream master bus which interfaces to the DUT.

- timing_gen

The timing generator module, This module generates the video timing based on the parameters specified by the test program. This emulates the function of the Video Timing Controller.

- test_vid_out

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

- phy_emulation

  The video PHY emulator simulates the video PHY interface driven by the DUT. It generates handshaking, and receives data from the core. It also generates expected values and compares these to incoming video data. Based on the syncs and blanks from the DUT, it creates an expected data value, and compares this to the incoming video.

Running the Simulation

The simulation source files for the core are the same verilog files delivered for synthesis. The test bench file is under the top level directory. The test bench file is `cust_tb_v_vid_in_axi4s.v`. This file contains all of the modules required for the test bench. The top level module name is "testbench". (UG626) *Synthesis and Simulation Design Guide* provides details on setting up and running simulations on Xilinx simulation tools.

When the test bench module is run, it produces the stimulus, applies it to the device under test, and prints out the line-standards run and the pass/fail results.

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Simulation

A testbench incorporating randomization of timing parameters was used to test the AXI4-Stream to Data out Core. Tesing included the following:

- Testing with multiple frames of data with many different timing parameters and frame sizes.
- Testing for locking in both slave and master timing modes.
- Testing of locking, and re-locking after input interruption and line standard changes.

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

Hardware Testing

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

Interoperability

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

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

Migrating

For information about migration from ISE Design Suite to Vivado Design Suite, see *Vivado Design Suite Migration Methodology Guide* (UG911) [Ref 2].

For a complete list of Vivado User and Methodology Guides, see Vivado Design Suite - 2012.3 User Guides.

From version v2.00.a to v2.01a of the AXI4-Stream to Video Out core, the only significant change is the addition of functionality for delaying the enabling of the video outputs until the start of the vertical blanking period.

The v2.01a core is compatible with the previous version.

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/company/terms.htm.

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.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

1. UG761 AXI Reference Guide
2. Vivado Design Suite Migration Methodology Guide (UG911)
3. Vivado™ Design Suite user documentation

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.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
4/24/2012	1.0	Initial Xilinx release of core.
07/25/2012	2.0	Updated for core version. Added Vivado information.
10/16/2012	3.0	Updated for core version. Updated for ISE v14.3 and Vivado v2012.3 tools. Added Vivado test bench.

Notice of Disclaimer