Video Scaler v8.1

LogiCORE IP Product Guide

Vivado Design Suite

PG009 November 18, 2015

XILINX
ALL PROGRAMMABLE™

XILINX

Table of Contents

Introduction

The Xilinx LogiCORE™ IP Video Scaler core is an optimized hardware block that converts an input color image of one size to an output image of a different size. This highly configurable core supports in-system programmability on a frame basis. The Video Scaler core resamples the incoming video data stream using a separable polyphase H/V filter arrangement to preserve hardware resources.

Features

- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface for dynamic control
- Supports 2-12 taps in both H and V domains
- Supports 2-16, 32 and 64 phases
- Software drivers available for EDK Video Scaler core
- Dynamically configurable filter coefficients
- Supports 8, 10, and 12 bits per color component input and output
- Supports YC4:2:2, YC4:2:0, RGB/4:4:4 chroma formatting
- Supports smooth real-time shrink/zoom, including non-integer phase offsets
- Multiple resource-sharing options
- Supports spatial resolutions from 32x32 up to 4096x4096
    - Supports 1080P60 in all supported device families (1)
    - Supports 4kx2k at the 24 Hz in supported high performance devices

1. Performance on low power devices can be lower.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream(2)
Resources	See Table 2-1 through Table 2-6.
Provided with Core	
Documentation	Product Guide
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog
Constraints File	XDC
Simulation Models	Encrypted RTL, VHDL or Verilog Structural, C Model
Supported Software Drivers (4)	Standalone
Tested Design Flows	
Design Entry Tools	Vivado® Design Suite
Simulation (5)	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis Tools	Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG1037 AXI Reference Guide [Ref 3].
3. HDL test bench and C-Model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-VID-SCALER.htm.
4. Standalone driver details can be found in the SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from the Xilinx Wiki page.
5. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

Video scaling is the process of converting an input color image of dimensions X_{in} pixels by Y_{in} lines to an output color image of dimensions X_{out} pixels by Y_{out} lines.

The Xilinx Video Scaler LogiCORE™ IP converts a specified rectangular area of an input digital video image from the original sampling grid to a desired target sampling grid (Figure 1-1).

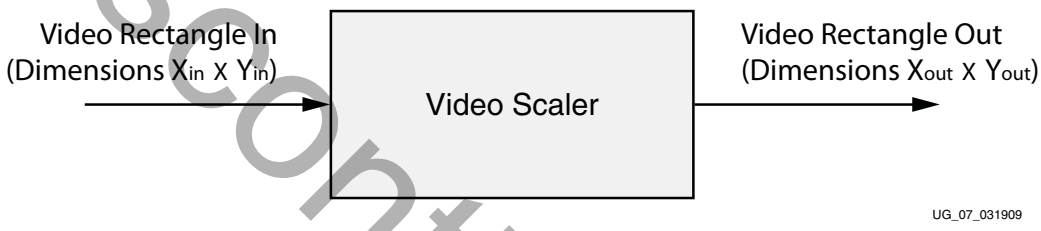

Figure 1-1: **High Level View of the Functionality**

The input image must be provided in raster scan format (left to right and top to bottom). The valid outputs are also given in this order.

Video scaling is a form of 2D filter operation which can be approximated with the equation shown in Equation 1-1.

Equation 1-1

$$Pix_{out}(x,y) = \sum_{HTaps-1}^{i=0} \sum_{VTaps-1}^{j=0} Pix_{in}[x - (HTaps/2) + i, y - (VTaps/2) + j] \times Coef(i,j)$$

In this equation, x and y are discrete locations on a common sampling grid; Pix_{out} (x, y) is an output pixel that is being generated at location (x, y); Pix_{in} (x, y) is an input pixel being used as part of the input scaler aperture; Coef(i, j) is an array of coefficients that depend upon the application; and HTaps, VTaps are the number of horizontal and vertical taps in the filter.

The coefficients in this equation represent weights applied to the set of input samples chosen to contribute to one output pixel, according to the scaling ratio.

Polyphase Concept

For scaling, the input and output sampling grids are assumed to be different, in contrast to the example in the preceding section. To express a discrete output pixel in terms of input pixels, it is necessary to know or estimate the location of the output pixel relative to the closest input pixels when superimposing the output sampling grid upon the input sampling grid for the equivalent 2-D space. With this knowledge, the algorithm approximates the output pixel value by using a filter with coefficients weighted accordingly. Filter taps are consecutive data-points drawn from the input image.

As an example, Figure 1-2 shows a desired 5x5 output grid (“O”) superimposed upon an original 6x6 input grid (“X”), occupying common space. In this case, estimating for output position (x, y) = (1, 1), shows the input and output pixels to be co-located. You can weight the coefficients to reflect no bias in either direction, and can even select a unity coefficient set. Output location (2, 2) is offset from the input grid in both vertical and horizontal dimensions. Coefficients can be chosen to reflect this, most likely showing some bias towards input pixel (2, 2), etc. Filter characteristics can be built into the filter coefficients by appropriately applying anti-aliasing low-pass filters.

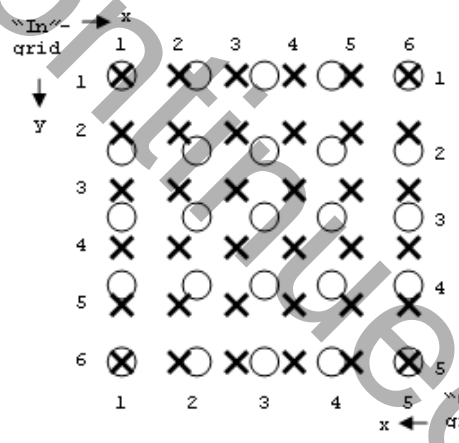

Figure 1-2: **5x5 Output Grid (“O”) Super-imposed over 6x6 Input Grid (“X”)**

The space between two consecutive input pixels in each dimension is conceptually partitioned into a number of bins or phases. The location of any arbitrary output pixel always falls into one of these bins, thus defining the phase of coefficients used. The filter architecture should be able to accept any of the different phases of coefficients, changing phase on a sample-by-sample basis.

A single dimension is shown in Figure 1-3. As illustrated in this figure, the five output pixels shown from left to right could have the phases 0, 1, 2, 3, 0.

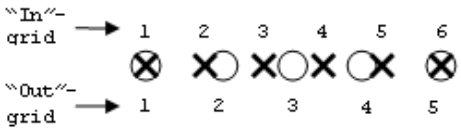

Figure 1-3: **Super-imposed Grids for 1 Dimension**

The examples in Figure 1-2 and Figure 1-3 show a conversion where the ratio $X_{in}/X_{out} = Y_{in}/Y_{out} = 5/4$. This ratio is known as the Scaling Factor, or SF. The horizontal and vertical Scaling Factors can be different. A typical example is drawn from the broadcast industry, where some footage can be shot using 720p (1280x720), but the cable operator needs to deliver it as per the broadcast standard 1080p (1920x1080). The SF becomes 2/3 in both H and V dimensions.

Typically, when $X_{in} > X_{out}$, this conversion is known as horizontal down-scaling (SF > 1). When $X_{in} < X_{out}$, it is known as horizontal up-scaling (SF < 1).

The set of coefficients constitute filter banks in a polyphase filter whose frequency response is determined by the amount of scaling applied to the input samples. The phases of the filter represent subfilters for the set of samples in the final scaled result.

The number of coefficients and their values are dependent upon the required low-pass, anti-alias response of the scaling filter; for example, smaller scaling ratios require lower passbands and more coefficients. Filter design programs based on the Lanczos algorithm are suitable for coefficient generation. Moreover, MATLAB® product fdatool/fvtool can be used to provide a wider filter design toolset. More information about coefficients is located in Coefficients in Chapter 3.

A direct implementation of Equation 1-1 suggests that a filter with VTaps x HTaps multiply operations per output are required. However, the Xilinx Video Scaler supports only separable filters, which completes an approximation of the 2-D operation using two 1-D stages in sequence – a vertical filter (V-filter) stage and a horizontal filter (H-filter) stage. The intermediate results of the first stage are fed sequentially to the second stage.

The vertical filter stage filters only in the vertical domain, for each incrementing horizontal raster scan position x, creating an intermediate result described as Vpix (Equation 1-2).

$$VPix_{int}[x,y] = \sum_{VTaps-1}^{i=0} Pix_{in}[x, y-(VTaps/2)+i] \times Vcoef[i] \qquad \text{Equation 1-2}$$

The output result of the vertical component of the scaler filter is input into the horizontal filter with the appropriate rounding applied. The separation means this can be reduced to the shown VTaps and HTaps multiply operations, saving FPGA resources (Equation 1-3).

$$Pix_{out}[x,y] = \sum_{HTaps-1}^{i=0} VPix_{int}[x-(HTaps/2)+i,y] \times Hcoef[i] \quad \text{Equation 1-3}$$

Notice that the difference between the Bilinear, Bicubic, and Polyphase architectures are marked by a difference in coefficients only. The Video Scaler core spans everything from 2-12 taps or coefs, so all three architectures are supported.

Given that statement, the implementation is always a Polyphase implementation. The Bilinear or Bicubic implementations are only an optimized polyphase implementation.

Bilinear is where the Video Scaler emulates a Bilinear implementation when you select two taps, and Bicubic where the Video Scaler emulates a Bicubic implementation when you select four taps.

Basic Architecture

The Xilinx Video Scaler input can process a live video feed, or can read from external memory. The output could feed directly to another processing stage in real time, but also could feed an external frame buffer (for example, for a VGA controller, or a Picture-in-Picture controller). Whatever the configuration, you must assess, given the clock-frequency available, how much time is available for scaling, and define:

1. Whether to source the scaler using live video or an input-side frame buffer.
2. Whether the scaler feeds out directly to the next stage or to an output-side frame buffer.

Prior to buffering the active part of a live video stream, its timing is specific, regular and defined. Over-zealous deassertion of `s_axis_video_tready` causes system-level errors in this case. Generally, this could be a danger in cases where the input data is to be up-scaled by the video scaler. In these cases, the scaler may need to process input data more than once to generate the desired scaled outputs.

Prior to buffering, the timing of a live video stream is specific, regular, and periodic. When a frame is buffered, on average, the amount of valid AXI4-Stream transaction per-frame time should be equal to the number of pixels per frame. If the Video Scaler core de-asserts `s_axi4s_video_tready` for extended periods of time, this condition can not be met.

When the input data is to be up-scaled, the Video Scaler may need to process input data more than once to generate the desired scaled outputs. For example, when up-scaling by a factor of 2, two lines must be output for every input line. The scaler core clock-rate (`core_clk`) must allow for this, especially considering the architectural specifics within the scaler that take advantage of the high speed features of the FPGA to allow for resource sharing.

Feeding data from an input frame buffer is more costly, but allows you to read the required data as needed, but still have one "frame" period in which to process it. Refer to Chapter 2, Throughput.

Some observations (not exclusively true for all conversions):

- Generally, when up-scaling, or dealing with high definition (HD) rates, it is simplest to use an input-side frame buffer. This does depend upon the available clock rates.
- When down-scaling, it is often the case that the input-side frame buffer is not required, because for every input line the scaler is required to generate a maximum of one valid output line.

It is not possible to feed the output directly to a display driver. Usually, a frame buffer is required to smooth the output data over an output frame period.

Applications

- Broadcast Displays, Cameras, Switchers, and Video Servers
- LED Wall
- Multi-Panel Displays
- Digital Cinema
- Projectors
- Medical Endoscope
- Video Surveillance
- Consumer Displays
- Video Conferencing
- Machine Vision

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Video Scaler product web page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The Video Scaler 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* [Ref 3] for additional information.

Performance

The following sections detail the performance characteristics of the Video Scaler core for Vivado Design Suite implementations.

Maximum Frequency

This section contains typical clock frequencies for the target devices.

These figures are typical and have been used as target clock frequencies for the Video Scaler core in the slowest speed grade for each device family. The Fmax data is a result of identical constraints applied to all three clocks: `s_axis_video_aclk`, `core_clk` and `m_axis_video_aclk`.

The maximum achievable clock frequency and all resource counts can be affected by tool options, additional logic in the FPGA device, using a different version of Xilinx tools. To assist with making system-level and board-level decisions, Table 2-1 through Table 2-3 show results of F_{MAX} observations for a broad range of scaler configurations, covering all speed-grades of the supported devices. This characterization data has been collated through multiple iterations of each configuration.

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

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FMax (MHz) for each Speed-Grade		
								-1	-2	-3
Yes	1920x1080	1920x1080	10	11x11	64	3(444)	1	294	324	370

1. Speedfile: XC7V585T FFG1157 Advanced 1.04k 2012-04-09

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

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FMax (MHz) for each Speed-Grade		
								-1	-2	-3
Yes	1920x1080	1920x1080	10	11x11	64	3(444)	1	294	332.5	370

1. Speedfile: XC7K70T FF484 ADVANCED 1.04c 2012-04-09

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

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FMax (MHz) for each Speed-Grade		
								-1	-2	-3
Yes	1920x1080	1920x1080	10	11x11	64	3(444)	1	181	222.5	247

1. Speedfile: XC7A100T FF484 ADVANCED 1.03k 2012-04-09

Latency

Latency through the Video Scaler is the number of cycles between applying the first (left-most) pixel of the top line at the core input and receiving the first pixel of the first scaled line at the core output.

Latency through the Video Scaler core is heavily dependent on the configuration applied in the GUI. In particular, increasing the number of vertical taps increases the latency by one line period. Additional fixed delays include input buffering, output buffering and filter latency.

Assuming that the performance of the Video Scaler core was not throttled by either the `s_axi4s_valid` or `m_axi4s_ready` AXI4 pins, the latency can be approximated as:

Max (Input Line-Length, Output Line-Length) x (2 + round_up (Number of V Taps / 2))

Throughput

Video Scaler core throughput is the number of complete frames of video data that can be scaled per second. Throughput through the Video Scaler is heavily dependent on the GUI settings.

Throughput is dependent on:

- Input and output image sizes and the clock-frequencies used.
- Scale Factor used
- The number of engines.

- Maximum line length to be handled in the system (into and out from the scaler)
- Maximum number of lines per frame (in and out)
- Maximum frame refresh rate
- Chroma format (4:4:4, 4:2:2, or 4:2:0)
- Clock F_{MAX} (for all of `core_clk`, `s_axis_video_aclk`, `m_axis_video_aclk`: depends upon the selected device)

For example, you can decide to use the scaler in its dual-engine parallel Y/C configuration to achieve the scale factor and frame rate desired. Using a dual-engine scaler allows the scaler to process more data per frame period at the cost of increased resource usage.

The size of the scaler implementation is determined by the number of taps and phases in the filter and the number of engines. The number of taps and phases do not impact the clock frequency.

Resource Utilization

Table 2-4 through Table 2-6 were generated using Vivado Design Suite with default tool options for characterization data. UltraScale™ results are expected to be similar to 7 series results.

Table 2-4: **Resource Usage for Virtex-7 Devices**

AXI4-Lite	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Htaps	Vtaps	Max Phases	Chroma Format	LUTs	FFs	36k BRAMs	18k BRAMs	DSP48s
Yes	1920x1080	1920x1080	8	4	4	4	4:02:02	1745	3127	10	1	12
Yes	1920x1080	1920x1080	10	4	4	4	4:02:02	1703	3076	11	6	12
Yes	1920x1080	1920x1080	12	4	4	4	4:02:02	1767	3179	12	5	12
Yes	1920x1080	1920x1080	8	8	8	4	4:02:02	1767	3443	16	1	20
Yes	1920x1080	1920x1080	8	11	11	4	4:02:02	1850	4115	20	1	26
Yes	1920x1080	1920x1080	8	4	4	64	4:02:02	1678	3064	10	1	12
Yes	512x1080	512x1080	8	4	4	4	4:02:02	1577	3004	3	7	12
Yes	1920x1080	1920x1080	8	4	4	4	4:04:04	1643	3236	10	9	28
Yes	1920x1080	1920x1080	10	11	11	64	4:04:04	1906	5508	43	2	70
No	1920x1080	1920x1080	8	4	4	4	4:02:02	754	969	7	1	10

Table 2-5: **Resource Usage for Kintex-7 Devices**

AXI4-Lite	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Htaps	Vtaps	Max Phases	Chroma Format	LUTs	FFs	36k BRAMs	18k BRAMs	DSP48s
Yes	1920x1080	1920x1080	8	4	4	4	4:02:02	1745	3127	10	1	12
Yes	1920x1080	1920x1080	10	4	4	4	4:02:02	1702	3076	11	6	12
Yes	1920x1080	1920x1080	12	4	4	4	4:02:02	1765	3179	12	5	12
Yes	1920x1080	1920x1080	8	8	8	4	4:02:02	1763	3443	16	1	20
Yes	1920x1080	1920x1080	8	11	11	4	4:02:02	1851	4115	20	1	26
Yes	1920x1080	1920x1080	8	4	4	64	4:02:02	1675	3064	10	1	12
Yes	512x1080	512x1080	8	4	4	4	4:02:02	1571	3004	3	7	12
Yes	1920x1080	1920x1080	8	4	4	4	4:04:04	1644	3236	10	9	28
Yes	1920x1080	1920x1080	10	11	11	64	4:04:04	1906	5508	43	2	70
No	1920x1080	1920x1080	8	4	4	4	4:02:02	753	969	7	1	10

Table 2-6: **Resource Usage for Artix-7 Devices**

AXI4-Lite	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Htaps	Vtaps	Max Phases	Chroma Format	LUTs	FFs	36k BRAMs	18k BRAMs	DSP48s
Yes	1920x1080	1920x1080	8	4	4	4	4:02:02	1744	3127	10	1	12
Yes	1920x1080	1920x1080	10	4	4	4	4:02:02	1698	3076	11	6	12
Yes	1920x1080	1920x1080	12	4	4	4	4:02:02	1769	3179	12	5	12
Yes	1920x1080	1920x1080	8	8	8	4	4:02:02	1768	3443	16	1	20
Yes	1920x1080	1920x1080	8	11	11	4	4:02:02	1852	4115	20	1	26
Yes	1920x1080	1920x1080	8	4	4	64	4:02:02	1674	3064	10	1	12
Yes	512x1080	512x1080	8	4	4	4	4:04:04	1641	3236	10	9	28
Yes	1920x1080	1920x1080	10	11	11	64	4:04:04	1906	5508	43	2	70
No	1920x1080	1920x1080	8	4	4	4	4:02:02	749	969	7	1	10

Core Interfaces and Register Space

Port Descriptions

The Video Scaler 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 the I/O diagram of the Video Scaler core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the `IRQ` pin are present only when the core is configured via the GUI with an AXI4-Lite control interface. The `INTC_IF` interface is present only when the core is configured via the GUI with the INTC interface enabled.

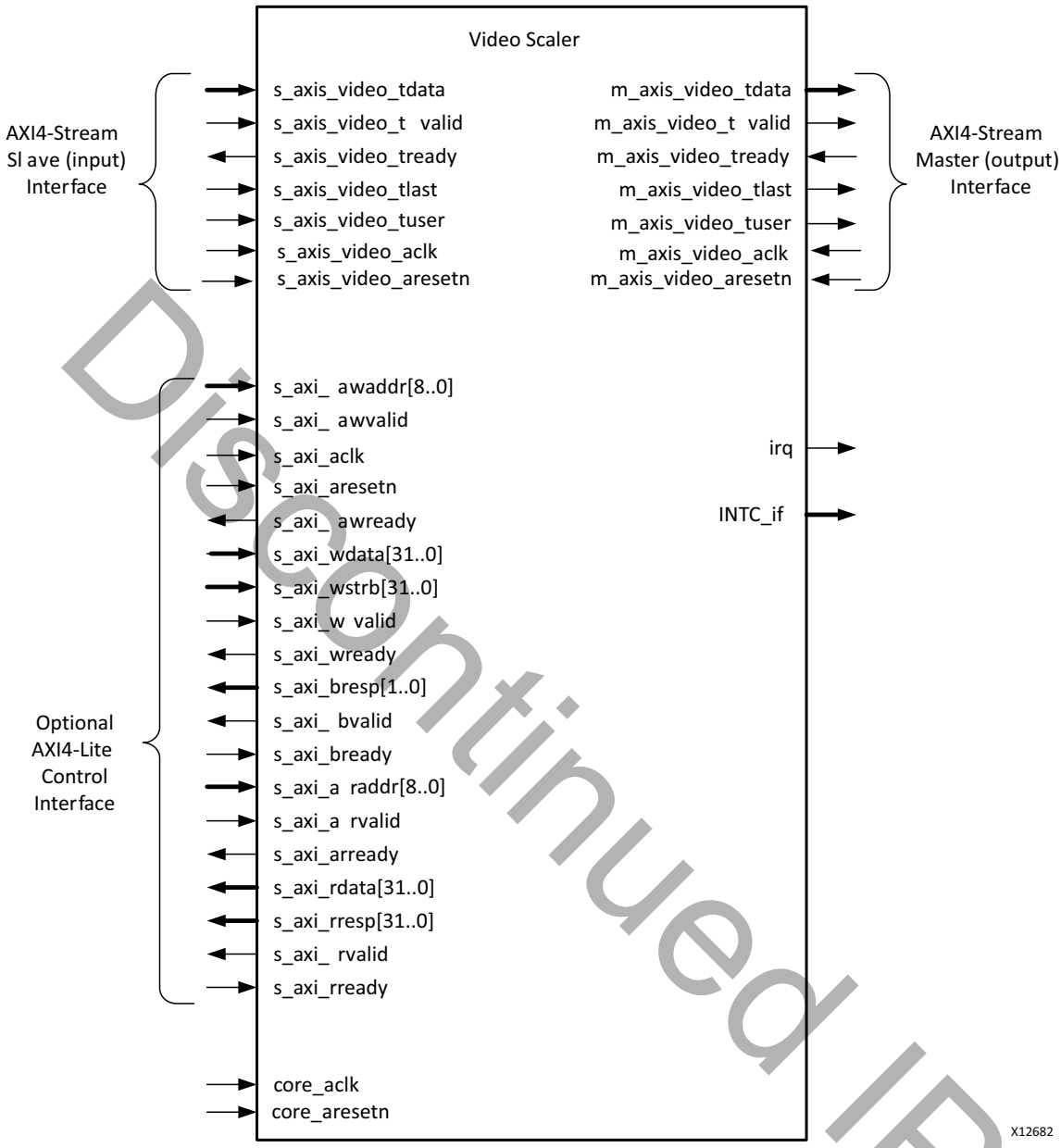

Figure 2-1: **Video Scaler Core Top-Level Signaling Interface**

Common Interface Signals

Table 2-7 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-7: **Common Interface Signals**

Signal Name	Direction	Width	Description
core_aresetn	In	1	Video Core Active Low Synchronous Reset.
core_clk	In	1	High-speed video core clock - used for internal processing, not for IO.
s_axis_video_aclk	In	1	Input clock - synchronous with incoming video data on s_axis_video AXI4-Stream interface. Also synchronous with AXI4-Lite interface.
m_axis_video_aclk	In	1	Output clock - synchronous with outgoing video data on m_axis_video AXI4-Stream interface.
intc_if	Out	6	Optional external interrupt controller Interface. Available only when INTC_IF is selected in the GUI
irq	Out	1	Optional interrupt request pin. Available only when INTC_IF is selected in the GUI.
s_axi_aclk	In	1	AXI4 - Lite interface clock and reset.
s_axi aresetn	In	1	AXI4 - Lite interface clock and reset.

The `s_axi_aclk` and `s_axi_aresetn` signals are shared between the core and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock, and reset pins: `S_AXI_ACLK` and `S_AXI_ARESETn`. Refer to The Interrupt Subsystem for a description of the `INTC_IF` and `IRQ` pins.

s_axi_aclk

The AXI4-Lite interface uses the `S_AXI_ACLK` pin as its clock source. This pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The Video Scaler core contains clock-domain crossing logic between the AXI4-Stream Video and `S_AXI_ACLK` (AXI4-Lite) clock domains. The core automatically ensures that the AXI4-Lite transactions complete even if the video processing is stalled with any of the `ARESETn` pins or with the video clock not running.

core_aresetn

The Video Scaler core has four reset sources: the `core_aresetn` pins (hardware reset), two `aresetn` sources are from the AXI4 Stream interface - `s_axis_video_aresetn` and `m_axis_video_aresetn` and the software reset option provided via the AXI4-Lite control interface (when present). When the AXI4Lite is present the software reset is a combination of the AXI4Lite reset and Slave AXI Stream reset else it is only the AXI Stream reset.

Note: CORE_ARESETn is by no means synchronized internally to AXI4-Stream frame processing, therefore de-asserting CORE_ARESETn while a frame is being process can lead to image tearing.

Within the appropriate clock domain, the external reset pulse needs to be held for 32 ACLK cycles to reset the core. The CORE_ARESETn signal only resets the AXI4-Stream interfaces. The AXI4-Lite interface is unaffected by the CORE_ARESETn signal to allow the video processing core to be reset without halting the AXI4-Lite interface.

Note: When a system with multiple-clocks and corresponding reset signals are being reset, the reset generator has to ensure all reset signals are asserted/de-asserted long enough that all interfaces and clock-domains in all IP cores are correctly reinitialized.

s_axi_aresetn

The S_AXI_ARESETn signal is synchronous to the S_AXI_ACLK clock domain, but is internally synchronized to the video clock domains. The S_AXI_ARESETn signal resets the entire core including the AXI4-Lite and AXI4-Stream interfaces.

s_axis_video_aclk

All signals on the Slave (data input) AXI4-Stream interface s_axis_video and AXI4-Lite component interfaces, must be synchronous to this clock signal.

All interface input signals are sampled on the rising edge of s_axis_video_aclk.

The s_axis_video_tready signal change occur after the rising edge of s_axis_video_aclk.

m_axis_video_aclk

All signals on the Master (data output) AXI4-Stream interface m_axis_video must be synchronous to this clock signal.

All interface input signals are sampled on the rising edge of m_axis_video_aclk.

All output signals changes occur after the rising edge of m_axis_video_aclk.

core_clk

This high-speed clock is used internally for video data processing. No scaler IO signals are synchronous to this clock signal.

s_axis_video_aresetn, m_axis_video_aresetn

The s_axis_video_aresetn and m_axis_video_aresetn pins are an active-Low, synchronous reset input pertaining to only AXI4-Stream interfaces, that is, the pins reset only the respective stream interfaces.

TIP: *The s_axis_video_aresetn pin also acts as software reset in fixed mode. See core_aresetn for more information.*

The ARESETn signal must be synchronous to the ACLK and must be held low for a minimum of 32 clock cycles of the slowest clock. The AXI4-Lite interface is unaffected by the ARESETn signal.

Data Interface

The Video Scaler core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *Vivado AXI Reference Guide* (UG1037), Video IP: AXI Feature Adoption section.

AXI4-Stream Signal Names and Descriptions

Table 2-8 describes the AXI4-Stream signal names and descriptions.

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	16, 24, 32, 40	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal
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
m_axis_video_tdata	Out	16, 24, 32, 40	Output Video Data
m_axis_video_tvalid	Out	1	Output Valid
m_axis_video_tready	In	1	Output Ready
m_axis_video_tuser	Out	1	Output Video Start Of Frame
m_axis_video_tlast	Out	1	Output Video End Of Line

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. The Video Scaler supports 4:2:x YC and 4:4:4/RGB video streams for 8, 10 and 12 bit video data.

For the 4:2:x YC case,

- Y always occupies bits (Video_Data_Width-1:0)
- C always occupies bits ((2*Video_Data_Width)-1: Video_Data_Width)

An example showing 10-bit YC data is shown in Figure 2-2.

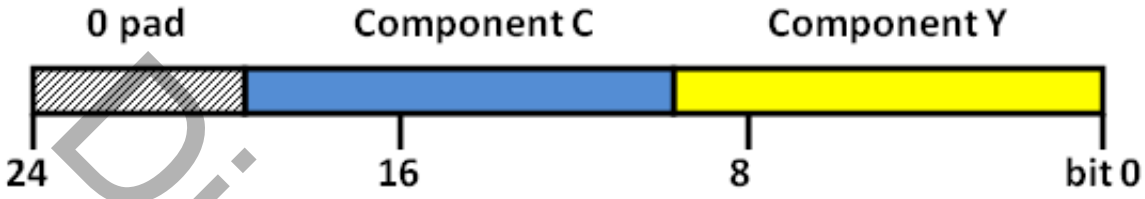

Figure 2-2: **YUV Data Embedding on TDATA**

For the RGB case,

- G occupies bits (Video_Data_Width-1:0)
- B occupies bits ((2*Video_Data_Width)-1: Video_Data_Width)
- R occupies bits ((3*Video_Data_Width)-1: (2*Video_Data_Width))

In all cases, the MSB of each component is the uppermost bit in the above scheme. 0-padding should be used for unused AXI4-Stream bits.

Figure 2-3 shows 12-bit RGB data.

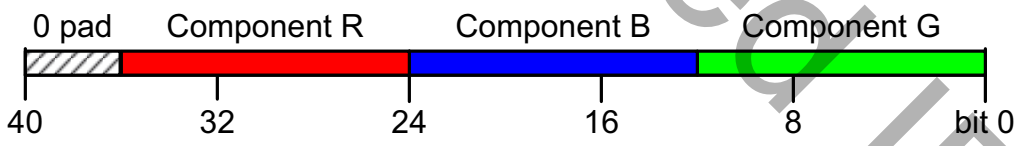

Figure 2-3: **RGB Data Embedding on TDATA**

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, and ARESETn are high at the rising edge of ACLK, as seen in Figure 2-4. 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 TVALID into Slave (Data Input) Interfaces.

Once `s_axis_video_tvalid` is asserted, no interface signals (except the Video Scaler core driving `s_axis_video_tready`) can change value until the transaction completes (`s_axis_video_tready`, `s_axis_video_tvalid` high on the rising edge of `s_axis_video_aclk`). Transactions can not be retracted or aborted. In any cycle following a transaction, `tvalid` can either be de-asserted or remain asserted to initiate a new transfer.

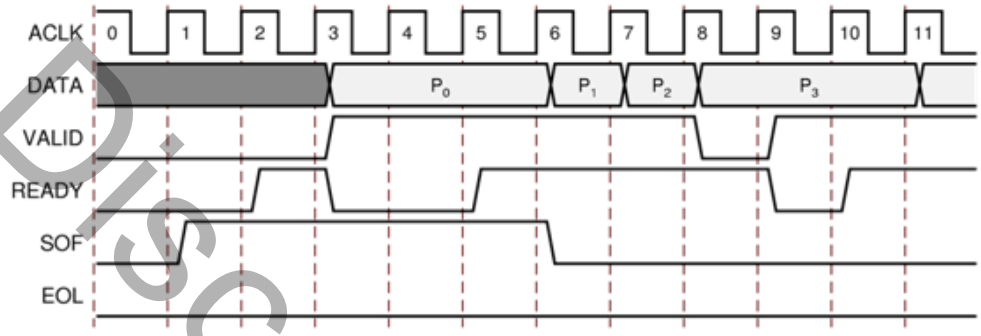

Figure 2-4: **Example of TREADY/TVALID Handshake, Start of a New Frame**

Guidelines on Driving TREADY into Master (Data Output) Interfaces

The `m_axis_video_tready` signal can be asserted before, during or after the cycle in which the Video Scaler core asserted `m_axis_video_tvalid`. The assertion of tready can be dependent on the value of `m_axis_video_tvalid`. A slave that can immediately accept data qualified by `tvalid`, should pre-assert its `m_axis_video_tready` signal until data is received. Alternatively, `m_axis_video_tready` can be registered and driven the cycle following `m_axis_video_tvalid` assertion.

RECOMMENDED: *It is recommended that the AXI4-Stream slave should drive `TREADY` independently, or pre-assert `TREADY` to minimize latency.*

Start of Frame Signals - m_axis_video_tuser0, s_axis_video_tuser0

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 2-4. 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 can be asserted an arbitrary number of AXI4-Stream clock cycles before the first pixel value is presented on `TDATA`, as long as a `TVALID` is not asserted.

End of Line Signals - m_axis_video_tlast, 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 2-5.

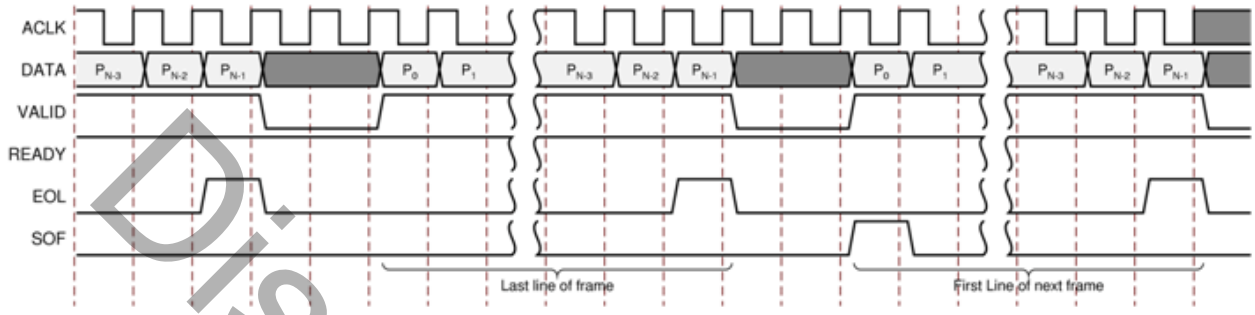

Figure 2-5: **Use of EOL and SOF Signals**

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The core has thirteen core specific registers which allow you to dynamically control the operation of the core.

Table 2-9: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	No	0x0	0: C_ENABLE 1: REG_UPDATE 2: reserved 3: CoefMemRdEn 31: SW_RESET (1: reset)
0x0004	STATUS	R/W	No	0	0: coef_fifo_rdy
0x0008	ERROR	R/W	No	0	0: EOL_ERROR 2: SOF_ERROR 4: COEF_WR_ERROR
0x000C	IRQ_ENABLE	R/W	No	0	0: coef_fifo_rdy_irq_en
0x0010	Version	R	No	0x0801000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	(Reserved)	-	-	-	-

Table 2-9: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0018	(Reserved)	-	-	-	-
0x001C	(Reserved)	-	-	-	-
0x0020	(Reserved)	-	-	-	-
0x0100	HSF	R/W	Yes	0	23-0: Horizontal Shrink Factor
0x0104	VSF	R/W	Yes	0	23-0: Vertical Shrink Factor
0x0108	Source_Video_Size	R/W	Yes	0	12-0: Source H Size 28-16: Source V Size
0x010C	H_Aperture	R/W	Yes	0	12-0: Aperture Start Pixel 28-16: Aperture End Pixel
0x0110	V_Aperture	R/W	Yes	0	12-0: Aperture Start Line 28-16: Aperture End Line
0x114	Output Size	R/W	Yes	0	12-0: Output H Size 28-16: Output V SIze
0x118	Num_Phases	R/W	Yes	0	6-0: Num_H_Phases 14-8: Num_V_Phases
0x11c	Active_Coefs	R/W	Yes	0	3-0: H Coeff set 7-4: V Coeff set
0x120	HPA_Y	R/W	Yes	0	20-0: Luma Horizontal Phase Offset
0x124	HPA_C	R/W	Yes	0	20-0: Chroma Horizontal Phase Offset
0x128	VPA_Y	R/W	Yes	0	20-0: Luma Vertical Phase Offset
0x12c	VPA_C	R/W	Yes	0	20-0: Chroma Vertical Phase Offset
0x130	Coef_Set_Wr_Addr	R/W	Yes	0	3-0: Coefficient Set for writing
0x134	Coef_Data_In	W	Yes (in Coef Loading FIFO)	0	Coefficient data input, when loading coefficients into scaler, 2 coefficients per write. 15-0: Coefficient 0 31-16: Coefficient 1
0x138	Coef_Set_Bank_Rd_Addr	R/W	Yes	0	First address-register when reading coefficients back from Scaler. This allows you to select which bank and which set to read. 1-0: Bank Select (00=HY; 01=HC, 10=VY, 11=VC 11-8: Set Select

Table 2-9: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x13c	Coef_mem_rd_addr	R/W	No	0	Second address-register when reading coefficients back from Scaler. This allows you to select the coefficient applied at which tap and phase within the selected bank. 3-0: Tap select 15-8: Phase select
0x140	Coef_mem_output	R	No	0	Read-coefficient data output.

1. Only available when the debugging features option is enabled in the GUI at the time the core is instantiated.

CONTROL (0x0000) Register

Bit 0 of the CONTROL register, SW_ENABLE, facilitates enabling and disabling the core from software. Writing '0' to this bit effectively disables the core halting further operations, which blocks the propagation of all video signals.

For the AXI4-Lite interface, after Power up, or Global Reset, the SW_ENABLE defaults to 0.

The SW_ENABLE flag is not synchronized with the AXI4-Stream interfaces: Enabling or Disabling the core takes effect immediately, irrespective of the core processing status.

Bit 1 of the CONTROL register, REG_UPDATE is a write-done semaphore for the host processor, which facilitates committing all user and timing register updates simultaneously.

Bit 3 of the CONTROL register, CoefMemRdEn, is used to control the readback of coefficients, and is described in Chapter 3, Designing with the Core.

Some core registers are double-buffered. For these cases, one set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers get copied over to the active set at the end of the AXI4-Stream frame, if and only if REG_UPDATE is set. Setting REG_UPDATE to 0 before updating multiple register values, then setting REG_UPDATE to 1 when updates are completed ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

Bit 31 of the CONTROL register, SW_RESET facilitates software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress is likely to cause image tearing.

STATUS (0x0004) Register

Bits of the STATUS register can be cleared individually by writing '1' to the bit position.

All bits of the STATUS register can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS register remain set after an event associated with the particular STATUS register bit, even if the event condition is not present at the time the interrupt is serviced. Bits of the STATUS register can be cleared individually by writing '1' to the bit position to be cleared.

Bit 0 of the STATUS register, coef_fifo_rdy, indicates that the core is ready to accept a coefficient. Following the input of the final coefficient, this bit is set low. You must not write further coefficients to the core until this bit goes high.

ERROR (0x0008) Register

Bits of the ERROR register facilitate debugging a video system.

Bit 0 of the ERROR register, EOL_ERROR, and bit 2 of the ERROR register SOF_ERROR, indicate that a framing error has been detected on the AXI4-Stream slave interface. If the scaler is receiving streaming video input, this typically happens if the streaming video source has been disconnected then reconnected. If the scaler is receiving video from a frame buffer (AXI-VDMA), typically the cause is either the frame sizes programmed to the AXI-VDMA and the Scaler are different, or the master and slave side of the AXI4-Stream interface have been reset independently.

Bit 4 of the ERROR register COEF_WR_ERROR indicates that a coefficient was written into the core before the core was not ready for it. The core is ready for a coefficient only when the status bit(0) is high.

Bits of the ERROR register remain set after an event associated with the particular bit occurred, even if the event condition is not present at the time the ERROR register value is polled. Bits of the ERROR register can be cleared individually by writing '1' to the bit position to be cleared.

IRQ_ENABLE (0x000C) Register

Any bits of the STATUS register can generate a host-processor interrupt request via the IRQ pin. The Interrupt Enable register facilitates selecting which bits of STATUS register asserts IRQ. Bits of the STATUS registers are masked by (AND) corresponding bits of the IRQ_ENABLE register and the resulting terms are combined (OR) together to generate IRQ.

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this

Read-Only value to verify that the software is matched to the correct version of the hardware.

HSF (0x0100) and VSF (0x0104) Registers

The HSF and VSF registers are the horizontal and vertical shrink-factors that must be supplied.

They should be supplied as integers, and can typically be calculated as follows:

hsf=round[((1+aperture_end_pixel-aperture_start_pixel)/(output_h_size))*2^20]

and

vsf=round[((1+aperture_end_line-aperture_start_line)/(output_v_size))*2^20]

Hence, up-scaling is achieved using a shrink-factor value less than one. Down-scaling is achieved with a shrink-factor greater than one.

You can work this calculation backwards. For a desired scale-factor, you can calculate the output size or the input size. This is application-dependent. Smooth zoom/shrink applications can take advantage of this approach, using the following start-phase controls.

The allowed range of values on these parameters is 1/12 to 12: (0x015555 to 0xC00000).

Source Video Size (0x0108) Register

The `SOURCE_VIDEO_SIZE` register encodes the number of active pixels per scan line and the number of active scan lines per frame.

The lower half-word (bits 12:0) encodes the number of active pixels per line. The upper half-word (bits 28:16) encodes the number of active lines per frame.

Supported values for both are between 32 and the user-provided values entered in the GUI. To avoid processing errors, you should restrict values written to `SOURCE_VIDEO_SIZE` to the range supported by the particular core instance.

H_Aperture (0x010c) and V_Aperture (0x110) Registers

The `H_APERTURE` and `V_APERTURE` registers define the size and location of the input rectangle within the incoming source video frame. They control the action of cropping from the input image if this is required, and are explained in detail in Scaler Aperture in Chapter 3

Output_Size (0x0114) Register

The OUTPUT_SIZE register defines the H and V size of the output rectangle. They do not determine anything about the target video format. You must determine what do with the scaled rectangle that emerges from the scaler core.

Num_Phases (0x0118) Register

Although you must specify the maximum number of phases (max_phases) that the core supports in the GUI, it is not necessary to run the core with a filter that has that many phases. Under some scaling conditions, you may want a large number of phases, but under others you may need only a few, or even only one. This dynamic control allows you to assert how many phases are in the selected set of coefficients. Non power-of-two numbers of phases are supported. It can vary between 2 and 16 inclusive, but also can be set to 32 or 64.

Active_Coefs (0x011c) Register

The scaler can store up to max_coef_sets coefficient sets internally. You can select which coefficient sets to use using ACTIVE_COEFS. Horizontal and Vertical operations can use different coefficients.

HPA_Y (0x0120), HPA_C (0x0124), VPA_Y (0x0128), and VPA_C (0x012c) Registers

By default, the scaler assumes that at the left and top edges of the image, the initial phase of coefficients is ZERO. These controls allow you to provide non-zero values if so desired.

Internally to the core, the scaler accumulates the 24-bit shrink-factor (hsf, vsf) to determine phase and filter aperture and coefficient phase. These four values allow you to preset the fractional part of the accumulations horizontally (hpa) and vertically (vpa) for luma (y) and chroma (c). When dealing with 4:2:2, luma and chroma are always vertically co-sited. Hence the start_vpa_c value is ignored.

How you use these parameters is important for scaling interlaced formats cleanly. On successive input fields, the start_vpa_y value needs to be modified. Also, when the desired result is a smooth shrink or zoom over a period of time, you can get better results by changing these parameters for each frame.

The allowed range of values on these parameters is -0.99 to 0.99: (0x100001 to 0x0FFFFF). The default value for these parameters is 0.

Coef_set_wr_addr (0x0130) Register, Coef_data_in (0x0134) Register

You can load custom coefficients using this interface. The scaler can store up to max_coef_sets coefficient sets internally. `coef_set_wr_addr` sets the set location of the set to which you intend to write. To cause the scaler to actively use the new set, you must set the `Active_Coefs` register accordingly. The coefficient loading mechanism is described in detail in Coefficients in Chapter 3.

Coef_set_bank_rd_addr (0x0138) Register, Coef_mem_rd_addr (0x013c) Register, Coef_mem_output(0x0140) Output

You can choose to read the coefficients currently in the scaler, for the sake of interest or for verification. Coefficients can be read using this interface. The coefficient readback mechanism is described in detail in Coefficients in Chapter 3.

Fixed Mode

When using this mode, the values are fixed at compile time. The system does not need to drive any of the parameters. The Vivado GUI prompts you to specify:

- coefficient file (.coe)
- hsf
- vsf
- aperture_start_pixel
- aperture_end_pixel
- aperture_start_line
- aperture_end_line
- output_h_size
- output_v_size
- num_h_phases
- num_v_phases

Fixed mode has the following restrictions:

- A single coefficient set must be specified using a .coe file; this is the only way to populate the coefficient memory.
- Coefficients can not be written to the core; the coefficient writing function is disabled.
- You can not specify h_coeff_set or v_coeff_set; there is only one set of coefficients.
- You can not specify start_hpa_y, start_hpa_c, start_vpa_y, start_vpa_c; they are set internally to zero.
- The control register is always set to "0x00000003," fixing the scaler in active mode.

The Interrupt Subsystem

STATUS register bits can trigger interrupts so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system. Irrespective of whether the AXI4-Lite control interface is present or not, the Video Scaler core detects AXI4-Stream framing errors, as well as the beginning and the end of frame processing.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (IRQ) is present. Events associated with bits of the STATUS register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (IRQ_ENABLE) are set. Once set by the corresponding event, bits of the STATUS

register stay set until the application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface, the application can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional INTC_IF port. This vector of signals gives parallel access to the individual interrupt sources, as seen in Table 2-10.

Unlike STATUS and ERROR flags, INTC_IF signals are not held, rather stay asserted only while the corresponding event persists.

Table 2-10: **INTC_IF Signal Functions**

INTC_IF Signal	Name	Function
0	eol_error	Indicates an that s_axis_video_tlast was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
1	Not used	
2	sof_error	Indicates an that s_axis_video_tuser was asserted when the core did not expect it. The expected position of this pulse is defined according to the source video resolution settings.
3	Not used	
4	intr_coef_wr_error	Indicates that a coefficient was written into the core when the core was not ready for it. Core is only ready for a coefficient when the status bit(0) is high.

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected INTC_IF signals as edge triggered interrupt sources. The INTC IP provides functionality to mask (enable or disable), as well as identify individual interrupt sources from software. Alternatively, for an external processor or MCU, you can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

Scaler Architectures

The scaler supports the following possible arrangements of the internal filters.

- Option 1: Single-engine for sequential YC processing
- Option 2: Dual Engine for parallel YC processing
- Option 3: Triple engine for parallel RGB/4:4:4 processing

When using RGB/4:4:4, only Option 3 can be used. Selecting Option 1 or Option 2 significantly affects throughput trading versus resource usage. These three options are described in detail in this chapter.

Architecture Descriptions

Single-Engine for Sequential YC Processing

This is the most complex of the three options because Y, Cr, and Cb operations are multiplexed through the same filter engine kernel.

One entire line of one channel (for example luma) is processed before the single-scaler engine is dedicated to another channel of the same video line. The input buffering arrangement allows for the channels to be separated on a line-basis. The internal data path bit widths are shown in Figure 3-1, as implemented for a 4:2:2 or 4:2:0 scaler. DataWidth can be set to 8, 10, or 12 bits.

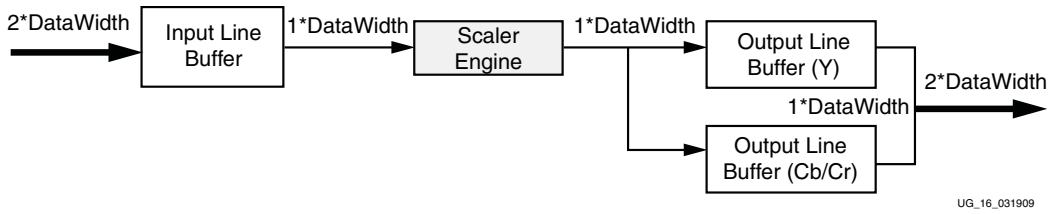

Figure 3-1: **Internal Data Path Bitwidths for Single-Engine YC Mode**

The scaler module is flanked by buffers that are large enough to contain one line of data, double buffered.

At the input, the line buffer size is determined by the parameter `max_samples_in_per_line`. At the output, the line-buffer size is determined by the parameter `max_samples_out_per_line`. These line buffers enable line-based arbitration, and avoid pixel-based handshaking issues between the input and the scaler core. The input line buffer also serves as the "most recent" vertical tap (that is, the lowest in the image) in the vertical filter.

4:2:0 Special Requirements

When operating with 4:2:0, it is also important to include the following restriction: **when scaling 4:2:0, the vertical scale factor applied at the vsf input must not be less than (2^{20})*144/1080**. This restriction has been included because Direct Mode 4:2:0 requires additional input buffering to align the chroma vertical aperture with the correct luma vertical aperture. In a later release of the video scaler, this restriction is removed.

Dual-Engine for Parallel YC Processing

For this architecture, separate engines are used to process Luma and Chroma channels in parallel as shown in Figure 3-2.

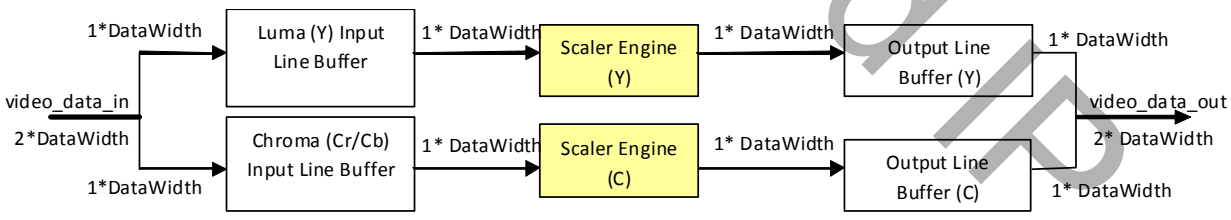

Figure 3-2: **Internal Data Path Bitwidths for Dual-Engine YC Mode**

For the Chroma channel, Cr and Cb are processed sequentially. Due to overheads in completing each component, the chroma channel operations for each line require slightly more time than the Luma operation. It is worth noting also that the Y and C operations do not work in synchrony.

Triple-Engine for RGB/4:4:4 Processing

For this architecture, separate engines are used to process the three channels in parallel, as shown in Figure 3-3.

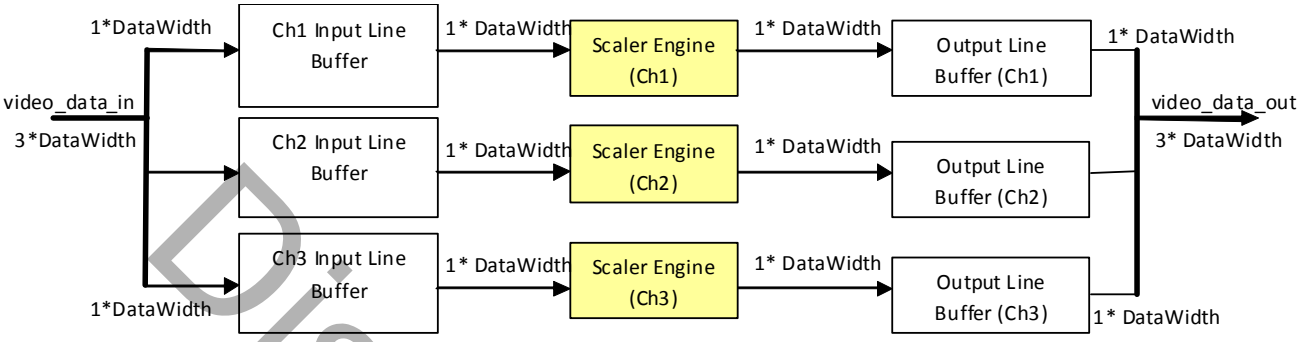

Figure 3-3: **Internal Data Path Bitwidths for Triple-Engine RGB/4:4:4 Architecture**

For this case, all three channels are processed in synchrony.

Clocking

The Video Scaler core has three clocks associated with the video data path:

- `s_axis_video_aclk` handles the clocking of data into the core.
- `core_clk` is used internally to the core.
- `m_axis_video_aclk` is the clock that is used to read video data out from the core.

Figure 3-4 shows the top level buffering, indicating the different clock domains, and the scope of the control state-machines.

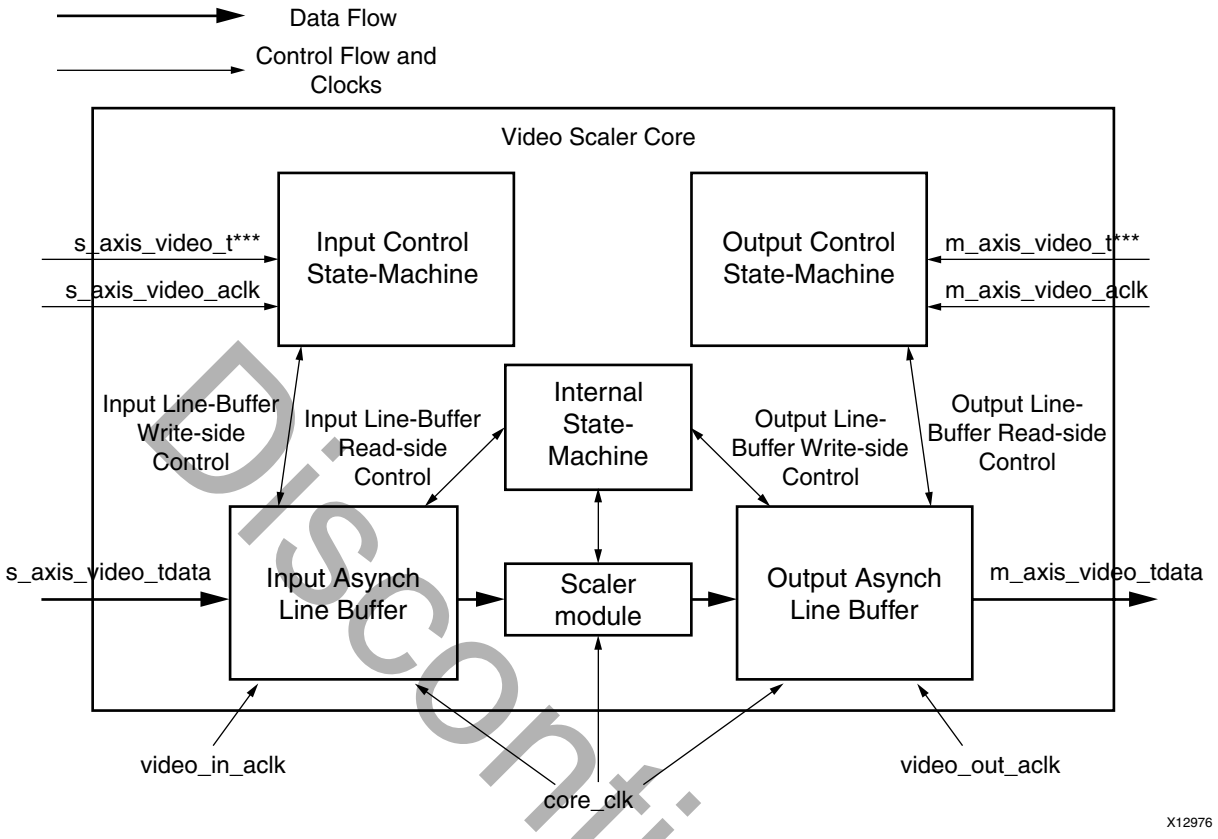

Figure 3-4: **Block Diagram Showing Clock Domains**

To support the many possibilities of input and output configurations, and to take advantage of the fast FPGA fabric, the central scaler processing module uses a separate clock domain from that used in controlling data I/O. More information is given in Performance in Chapter 2 about how to calculate the minimum required operational clock frequency. It is also possible to read the output of the scaler using a 3rd clock domain. These clock domains are isolated from each other using asynchronous line buffers as shown in Figure 3-4. The control state-machines monitor the I/O line buffers. They also monitor the current input and output line numbers.

To determine whether or not the scaler meets the application requirements, calculate the minimum clock frequencies required to make the intended conversions possible.

The following definitions are necessary to calculate the minimum clock-frequencies:

Subject Image	The area of the active image that is driven into the scaler. This may or may not be the entire image, dependent upon your requirements. It is of dimensions (**SubjWidth** x **SubjHeight**).
Active Image	The entire active input image, some or all of which include the Subject Image, and is of dimensions (**ActWidth** x **ActHeight**).
FullFrame	The entire frame of the input image including the Active Image and Blanking (**FullFrameWidth** x **FullFrameHeight**).

F'core_clk	The core_clk frequency. Data is read from the internal input line buffer, processed and written to the internal output buffer using this clock.
F'input_clk	The s_axis_video_aclk frequency.
FLineIn	The input Line Rate – could be driven by input rate or scaler LineReq rate. FLineIn must represent the maximum burst frequency of the input lines. For example, 720P exhibits an FLineIn of 45 kHz. (FLineIn = Number of input lines, including blanking * FFrameIn)
FFrameIn	The fixed frame refresh rate (Hz) – same for both input and output.
Output Image	The area of the active output image that is driven out of the scaler. It is of dimensions (**OutputWidth** x **OutputHeight**).
F'output_clk	The m_axis_video_aclk frequency.

To make the calculations according to the previous definitions and assumptions, it is necessary to distinguish between the following cases:

- **Live video mode:** An input video stream feeds directly into the scaler via AXI4-Stream.
  - Holding the input data stream off can cause distortion and errors to occur. The core must be configured and operated such that it doers not try to do this.
  - The system must be able to cope with the constant flow of video data.
- **Memory mode:** You can control the input feed using the AXI4-Stream back-pressure and handshaking by implementing an input frame buffer.

Live Video, page 36 and Memory Mode, page 44 detail some example cases that illustrate how to calculate the clock frequencies required to sustain the throughput required for given usage scenarios.

Of the three clocks, the simpler cases are the input and output clock signals, as outlined below:

- `s_axis_video_aclk`: Input Clock = F'input_clk

  This should be of a sufficiently high frequency to deliver all active pixels in an input frame into the scaler during one frame period, adding a safety margin of around 10%.

  When the data is being fed from a live source (for example, 720P/60), the clock signal is driven from the video source.

  When driving the input frame from memory, it is not necessary to use the exact pixel rate clock. In this case the `video_in_clk` frequency must be high enough to pass a frame of active data to the core within one frame period, given that the interface accepts one pixel per clock period. Add around 10% to this figure to accommodate the various filter latencies within the core.

  For example, when the input resolution is 1280x720 and the frame rate is 60 Hz, live video usually delivers this format (720P60) with a pixel clock of 74.25 MHz. However, this accommodates horizontal and vertical blanking periods. The average active pixel rate

for 720P60 is around 55.3 MHz. So, for scaling 720P frames that are stored in memory, the clock can safely be driven at any frequency above approximately 61 MHz. Once the memory mode scaler reaches the end of a frame, it stops processing until the start of the next frame. So, faster clock rates are safe.

- `m_axis_video_aclk`: Output Clock = F'output_clk

  The minimum clock frequency that must be applied at the m_axis_video_aclk input depends upon whether the core is in Live mode or Memory mode.

TIP: *In both cases, note that the active part of the output frame changes size in comparison to the input frame, due to the actions of the scaler, but the frame-rate has not changed.*

  - **Live video mode**: When an input video stream feeds directly into the scaler via AXI4-Stream, the Video Scaler must pass the entire scaled image out via the output interface within the period of time it takes to input the subject image. The calculation can be approximated as follows:

    SubjectImageTimeTaken=SubjHeight/FLineIn

    NumOutputPixels=OutputWidth*OutputHeight

    MinFOut=(NumOutputPixels/SubjectImageTimeTaken)*1.1.

  The 10% margin of error has been added here to allow for headroom.

  - **Memory mode**: In Memory mode, the m_axis_video_aclk driven into the Video Scaler must be at a frequency high enough to pass one frame of active data of the output resolution, adding a safety margin of around 10%.

  Similar to the memory mode clock described above, this clock must be driven into the scaler at a frequency high enough to pass one frame of active data, adding a safety margin of around 10%.

TIP: *Note that the active part of the frame has now changed size due to the actions of the scaler, but the frame-rate has not changed.*

- `core_clk`: Core Clock = F'core_clk

  The minimum required clock frequency of this clock is a more complicated to calculate.

Live Video

"Live Video" refers to the situation where video input comes into the core via AXI4-Stream interface, without having been passed through a large (usually external) frame-buffer. In this situation, throttling of the input stream can cause distortion and video errors. In many cases, especially when the scaler is downscaling, it is not necessary for the scaler to assert back-pressure on its input interface. However, when upscaling, backpressure can be

required to generate more output data than was input to the core. Although some upscaling is possible, more flexibility is possible when using "Memory Mode" as described later.

If no input frame buffer is used, and the timing of the input video format drives the scaler, then the number of 'core_clk' cycles available per H period becomes important. **FLineIn** is a predetermined frequency in this case, often (but not necessarily) defined according to a known broadcast video format (for example 1080i/60, 720P, CCIR601, etc.).

The critical factors can be summarized as follows:

- **ProcessingOverheadPerComponent** –The number of extraneous cycles needed by the scaler to complete the generation of one component of the output line, in addition to the actual processing cycles. This is required due to filter latency and State-Machine initialization. For all cases in this document, this has been approximated as 50 to 75 cycles per component per line.
- **CyclesRequiredPerOutputLine** – This is the number of cycles the scaler requires to generate one output line, of multiple components. The final calculation depends upon the chroma format and the filter configuration (YC4:2:2 only).

  The general calculation is:

```
CyclesRequiredPerOutputLine=(CompsPerEngine*Max(output_h_size,SubjWidth))+OverHeadMult*ProcessingOverheadPerComponent
```

  The `CompsPerEngine` and `OverHeadMult` values can be extracted from Table 3-1.

Table 3-1: **Throughput Calculations for Different Chroma Formats**

Chroma Format	NumEngines	CompsPerEngine	OverHeadMult
4:4:4 (e.g., RGB)	3	1	1
4:2:2 High performance	2	1	2
4:2:2 Standard performance	1	2	3
4:2:0	1	2	3

NumEngines

This is the number of engines used in the implementation. For the YC4:2:2 case, a higher number of engines uses more resources - particularly block RAM and DSP48. In the 4:4:4 case, three engines are implemented.

CompsPerEngine

This is the largest number of full h-resolution components to be processed by this instance of the scaler. When using YC, each chroma component constitutes 0.5 in this respect.

OverHeadMult

For each component processed by a single engine, the ProcessingOverheadPerComponent overhead factor must be included in the equation. The number of times this overhead needs to be factored in depends upon the number of components processed by the worst-case engine.

For 4:4:4:

```
CycleRequiredPerOutputLine = 1*Max(OutputWidth,SubjWidth)) +
1*ProcessingOverheadPerComponent
```

For 4:2:2 Dual-Engine:

```
CycleRequiredPerOutputLine = 1*Max(OutputWidth,SubjWidth)) +
2*ProcessingOverheadPerComponent
```

For 4:2:2 Single-Engine:

```
CycleRequiredPerOutputLine = 2*Max(OutputWidth,SubjWidth)) +
3*ProcessingOverheadPerComponent
```

For 4:2:0:

```
CycleRequiredPerOutputLine = 2*Max(OutputWidth,SubjWidth)) +
3*ProcessingOverheadPerComponent
```

- **MaxVHoldsPerInputAperture** – This is the maximum number of times the vertical aperture needs to be 'held' (especially up-scaling):

```
MaxVHoldsPerInputAperture = CEIL (output_v_size/input_v_size)
```

Given the preceding information, it is now necessary to calculate how many cycles it take to generate the worst-case number of output lines for any vertical aperture:

- **MaxCoreClksTakenPerVAperture** – This is the number of core clock cycles it take to generate **MaxVHoldsPerInputAperture** lines.

```
MaxCoreClksTakenPerVAperture = CyclesRequiredPerOutputLine x
MaxVHoldsPerInputAperture
```

It is then necessary to decide the minimum 'core_clk' frequency required to achieve your goals according to this calculation:

```
MinF'core_clk = 1.05 * F'input_clk * MaxCoreClksTakenPerVAperture/
(NumInputClksPerInputLine + InputHblank)
```

where

```
NumInputClksPerInputLine = FullFrameWidth
```

and

```
InputHblank = FullFrameWidth - ActWidth
```

Some examples follow. ***They are estimates only, and are subject to change.***

Example 1: The Unity Case

1080i/60 YC4:2:2 'passthrough'
Vertical scaling ratio = 1.00
Horizontal scaling ratio = 1.00
FLineIn = 33750
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*1920 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(540/540) = 1
MaxCoreClksTakenPerVAperture = 3990 * 1 = 3990
NumInputClksPerInputLine + InputHblank = 2200 + 280 = 2480
MinF'core_clk = 1.05 x 74.25MHz x (3990/2480) = 125.4 Mhz
video_in_clk = Frequency defined by live-mode input pixel clock. Typically 74.25
MHz.
SubjectImageTimeTaken=540/33750 =  0.016
NumOutputPixels=1920x540 = 1036800
MinFOut (m_axis_video_aclk) = (1036800 / 0.016) * 1.1 = 71.28 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/1.0) = 0x100000
vsf=2^20 x (1/1.0) = 0x100000
```

Note: This case is possible with no input buffer using Spartan-6 because the MinF'clk is less than the core Fmax, as shown in Table 3-1.

Example 2: Up-scaling 640x480 60 Hz YC4:2:2 to 800x600
Assuming 31.5 kHz line rate
Vertical scale ratio = 1.25
Horizontal scale ratio = 1.25
FLineIn = 31500
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*800 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(600/480) = 2
MaxCoreClksTakenPerVAperture = 1750 * 2= 3500
NumInputClksPerInputLine + InputHblank = 857 + 217= 1074
MinF'core_clk = 1.05 x 27MHz x (3500/1074) = 92.4 Mhz
video_in_clk = frequency defined by live-mode input pixel clock. Typically 74.25
MHz.
SubjectImageTimeTaken=480/31500 = 0.0152
NumOutputPixels=800x600 = 480000
MinFOut (m_axis_video_aclk)= (480000 / 0.0152) * 1.1 = 34.65 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/1.25) = 0x0CCCCC
vsf=2^20 x (1/1.25) = 0x0CCCCC
```

Example 3: Up-scaling 640x480 60 Hz YC4:2:2 to 1920x1080p60
Assuming 31.5 kHz line rate
Vertical scale ratio = 3.0
Horizontal scale ratio = 2.2
FLineIn = 31500
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*1920 + 150 (approximately)
MaxVHoldsPerInputAperture =round_up(1080/480) = 3
MaxCoreClksTakenPerVAperture = 3990 * 3 = 11970
NumInputClksPerInputLine + InputHblank = 857 + 217 = 1074
MinF'core_clk = 1.05 x 27MHz x (11970/1074) = 316.0 Mhz
video_in_clk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=480/31500 = 0.0152
NumOutputPixels=1920x1080 = 2073600
MinFOut (m_axis_video_aclk) =(2073600 / 0.0152) * 1.1 = 149.69 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/2.2) = 0x55555
$vsf=2^{20}$ x (1/3.0) = 0x71c71

Note: Without an input frame buffer, this conversion only works in high speed grade Virtex and Kintex devices.

Example 4: Up-scaling 640x480 60 Hz YC4:2:2 to 1920x1080p60
Assuming 31.5 kHz line rate
Vertical scale ratio = 3.0
Horizontal scale ratio = 2.2
FLineIn = 31500
Dual-engine implementation

```
CyclesPerOutputLine = 1*1920 + 2*50 (approximately)
MaxVHoldsPerInputAperture =round_up(1080/480) = 3
MaxCoreClksTakenPerVAperture = 2020 * 3 = 6060
NumInputClksPerInputLine + InputHblank = 857 + 217 = 1074
MinF'core_clk = 1.05 x 27MHz x (6060/1074) = 160.0 Mhz
s_axis_video_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=480/31500 = 0.0152
NumOutputPixels=1920x1080 = 2073600
MinFOut (m_axis_video_aclk) = (2073600 / 0.0152) * 1.1 = 149.69 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/2.2) = 0x55555
$vsf=2^{20}$ x (1/3.0) = 0x71c71

Note: For a dual-engine implementation, without an input frame buffer, this conversion works in devices that support this clock-frequency.

Example 5: Down-scaling 800x600 60Hz YC4:2:2 to 640x480
Assuming 37.68 kHz line rate
Vertical scale ratio = 0.8
Horizontal scale ratio = 0.8
FLineIn = 37680
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*800 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(480/600) = 1
MaxCoreClksTakenPerVAperture = 1750 * 1= 1750
NumInputClksPerInputLine + InputHblank = 640 + 217 = 1074
MinF'core_clk = 1.05 x 27MHz x (1750/1074) = 46.2 Mhz
s_axis_video_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=600/37680 = 0.0159
NumOutputPixels=640x480 = 307200
MinFOut (m_axis_video_aclk) = (307200/0.0159) * 1.1 = 21.22 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/0.8) = 0x140000
$vsf=2^{20}$ x (1/0.8) = 0x140000

Note: This conversion works in any of the supported devices and speed grades.

Example 6: Down-scaling 1080P60 YC4:2:2 to 720P/60
67.5 kHz line rate
Vertical scale ratio = 0.6667
Horizontal scale ratio = 0.6667
FLineIn = 67500
Single-engine implementation

```
CyclesPerOutputLine = 2*1920 + 3*50 (approximately)
MaxVHoldsPerInputAperture = round_up(720/1080) = 1
MaxCoreClksTakenPerVAperture = 3990 * 1 = 3990
NumInputClksPerInputLine + InputHblank = 2200 + 280 = 2480
MinF'core_clk = 1.05 x 148.5MHz x (3990/2480) = 250.9 Mhz
s_axis_video_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=1080/67500 = 0.016
NumOutputPixels=1280x720 = 921600
MinFOut (m_axis_video_aclk) = (921600 / 0.016) * 1.1 = 63.36 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/0.6667) = 0x180000
$vsf=2^{20}$ x (1/0.6667) = 0x180000

Note: When using a single-engine, this conversion not function with or without frame buffers (see Memory Mode, page 44) unless using higher speed grade devices.

Example 7: Down-scaling 1080P60 YC4:2:2 to 720P/60
67.5 kHz line rate
Vertical scale ratio = 0.6667
Horizontal scale ratio = 0.6667
FLineIn = 67500
Dual-engine implementation

```
CyclesPerOutputLine = 1*1920 + 2*50 (approximately)
MaxVHoldsPerInputAperture = round_up(720/1080) = 1
MaxCoreClksTakenPerVAperture = 2020 * 1 = 2020
NumInputClksPerInputLine + InputHblank = 2200 + 280 = 2480
MinF'core_clk = 1.05 x 148.5MHz x (2020/2480) = 127.0 Mhz
s_axis_video_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=1080/67500 = 0.016
NumOutputPixels=1280x720 = 921600
MinFOut (m_axis_video_aclk) =(921600 / 0.016) * 1.1 = 63.36 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/0.6667) = 0x180000
$vsf=2^{20}$ x (1/0.6667) = 0x180000

Note: This conversion functions in any of the supported devices and speed grades.

Example 8: Down-scaling 720P/60 YC4:2:2 to 640x480
45 kHz line rate
Vertical scale ratio = 0.6667
Horizontal scale ratio = 0.5
FLineIn = 45000
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*1280 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(480/720) = 1
MaxCoreClksTakenPerVAperture = 2710 * 1 = 2710
NumInputClksPerInputLine + InputHblank = 1650 + 370 = 2020
MinF'core_clk = 1.05 x 74.25MHz x (2710/2020) = 104.6 Mhz
s_axis_video_aclk = frequency defined by live-mode input pixel clock. Typically
74.25 MHz.
SubjectImageTimeTaken=720/45000 = 0.016
NumOutputPixels=640x480 = 307200
MinFOut (m_axis_video_aclk) =(307200 / 0.016) * 1.1 = 21.12 MHz
```

Shrink-factor inputs:

$hsf=2^{20}$ x (1/0.5) = 0x200000
$vsf=2^{20}$ x (1/0.6667) = 0x180000

Note: This conversion functions in any of the supported devices and speed grades.

Example 9: Converting 720P/60 YC4:2:2 to 1080i/60 (1920x540)
45 kHz line rate
Vertical scale ratio = 0.75
Horizontal scale ratio = 1.5
FLineIn = 45000
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*1920 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(540/720) = 1
MaxCoreClksTakenPerVAperture = 3990 * 1 = 3990
NumInputClksPerInputLine + InputHblank = 1650 + 370 = 2020
MinF'core_clk = 1.05 x 74.25MHz x (3990/2020) = 154.0 Mhz
s_axis_video_aclk = Frequency defined by live-mode input pixel clock. Typically
74.25 MHz.
SubjectImageTimeTaken = 720/45000 = 0.016
NumOutputPixels=1920x540 = 1036800
MinFOut (m_axis_video_aclk) =(1036800 / 0.016) * 1.1 = 71.28 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/1.5) = 0x0AAAAA
vsf=2^20 x (1/0.6667) = 0x155555
```

Note: This conversion functions in Virtex and Kintex devices, and in higher speed grade Artix devices.

Example 10: Converting 720P/60 YC4:2:2 to 1080p/60
45 kHz line rate
Vertical scale ratio = 1.5
Horizontal scale ratio = 1.5
FLineIn = 45000
Dual-engine implementation

```
CyclesRequiredPerOutputLine = 1*1920 + 2*50 (approximately)
MaxVHoldsPerInputAperture = round_up(1080/720) = 2
MaxCoreClksTakenPerVAperture = 2020 * 2 = 4040
NumInputClksPerInputLine + InputHblank = 1650 + 370 = 2020
MinF'core_clk = 1.05 x 74.25MHz x (4040/2020) = 155.9 Mhz
s_axis_video_aclk = Frequency defined by live-mode input pixel clock. Typically
74.25 MHz.
SubjectImageTimeTaken=720/45000 = 0.016
NumOutputPixels=1920x1080 = 2073600
MinFOut (m_axis_video_aclk) = (2073600 / 0.016) * 1.1 = 142.56 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/1.5) = 0x0AAAAA
vsf=2^20 x (1/1.5) = 0x0AAAAA
```

Note: This conversion functions in Virtex and Kintex devices, and in higher speed grade Artix devices.

Memory Mode

"Memory Mode" refers to the situation where video input comes into the core via AXI4-Stream interface, having been passed through a large (usually external) frame-buffer. In this situation, the scaler can read data from the memory at it's leisure. To "hold-off" the incoming data, it de-asserts the s_axis_video_tready signal on the input AXI4-Stream Interface. This mode offers more freedom to scale the video stream as you pleases, dynamically.

Using an input frame buffer allows you to stretch the processing time over the entire frame period (utilizing the available blanking periods). New input lines can be provided as the internal phase-accumulator dictates, instead of the input timing signals.

The critical factors can be summarized as follows:

- **ProcessingOverheadPerLine** – The number of extraneous cycles needed by the scaler to complete the generation of one output line, in addition to the actual processing cycles. This is required due to filter latency and State-Machine initialization. For all cases in this document, this has been approximated as 50 cycles per component per line.
- **FrameProcessingOverhead** – The number of extraneous cycles needed by the scaler to complete the generation of one output frame, in addition to the actual processing cycles. This is required mainly due to vertical filter latency. For all cases in this document, this has been generally approximated as 10000 cycles per frame.
- **CyclesPerOutputFrame** – This is the number of cycles the scaler requires to generate one output frame, of multiple components. The final calculation depends upon the chroma format (and, for YC4:2:2 only, the filter configuration), and can be summarized as:

For 4:4:4:

```
CyclesPerOutputFrame =
Max [
       (output_h_size + ProcessingOverheadPerLine)*output_v_size,
       (input_h_size + ProcessingOverheadPerLine)*input_v_size
    ]
+ FrameProcessingOverhead
```

For 4:2:2 dual-engine:

```
CyclesPerOutputFrame =
Max [
       (output_h_size + (ProcessingOverheadPerLine*2))*output_v_size,
       (input_h_size + (ProcessingOverheadPerLine*2))*input_v_size
    ]
+ FrameProcessingOverhead
```

For 4:2:2 single-engine:

```
CyclesPerOutputFrame =
```

```
Max [
       ((output_h_size*2) + (ProcessingOverheadPerLine*3))*output_v_size,
       ((input_h_size*2) + (ProcessingOverheadPerLine*3))*input_v_size
    ]
+ FrameProcessingOverhead
```

For 4:2:0:

```
CyclesPerOutputFrame =
Max [
       ((output_h_size*2) + (ProcessingOverheadPerLine*3))*output_v_size,
       ((input_h_size*2) + (ProcessingOverheadPerLine*3))*input_v_size
    ]
+ FrameProcessingOverhead
```

It is then necessary to decide the minimum `core_clk` frequency according to this calculation (include 10% overhead):

```
MinF'core_clk' = FFrameIn x CyclesPerOutputFrame * 1.1
```

Example 11: Converting 720P YC4:2:2 to 1080i/60 (1920x540)

Vertical scale ratio = 0.75
Horizontal scale ratio = 1.5
FFrameIn = 60
Single-engine implementation.

```
CyclesPerOutputFrame = (1920*2 + 150)*540 + 10000 (approximately) = 2164600
MinF'core_clk' = 60 x 2164600 = 129.88 MHz
s_axis_video_aclk = Delivery of 1 frame of 1280x720 pixels in 1/60 s * 1.1:
Fmin = 60.83 MHz
m_axis_video_aclk = Delivery of 1 field of 1920x540 pixels in 1/60 s * 1.1: Fmin
= 68.43 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/1.5) = 0x0AAAAA
vsf=2^20 x (1/0.75) = 0x155555
```

This conversion is possible using Spartan-6 devices.

Note: See example 9 for contrasting conversion.

Example 12: Converting 720P/60 YC4:2:2 to 1080p/60
Vertical scale ratio = 1.5
Horizontal scale ratio = 1.5
FFrameIn = 60
Dual-engine implementation

```
CyclesPerOutputFrame = (1920*1 + 100)*1080 + 10000 (approx) = 2191600
MinF'core_clk' = 60 x 2191600= 131.5 MHz
s_axis_video_aclk - Delivery of 1 frame of 1280x720 pixels in 1/60 s * 1.1:
```

```
Fmin = 60.83 MHz
m_axis_video_aclk - Delivery of 1 frame of 1920x1080 pixels in 1/60 s * 1.1: Fmin
= 136.86 MHz
```

Shrink-factor inputs:

$$hsf=2^{20} \times (1/1.5) = 0x0AAAAA$$

$$vsf=2^{20} \times (1/1.5) = 0x0AAAAA$$

This conversion works in all devices, including Spartan-6 -2 speed grade devices.

Note: See example 10 for a contrasting conversion.

Clock, Enable, and Reset Considerations

Figure 3-5 shows a typical example of the Video Scaler in a system.

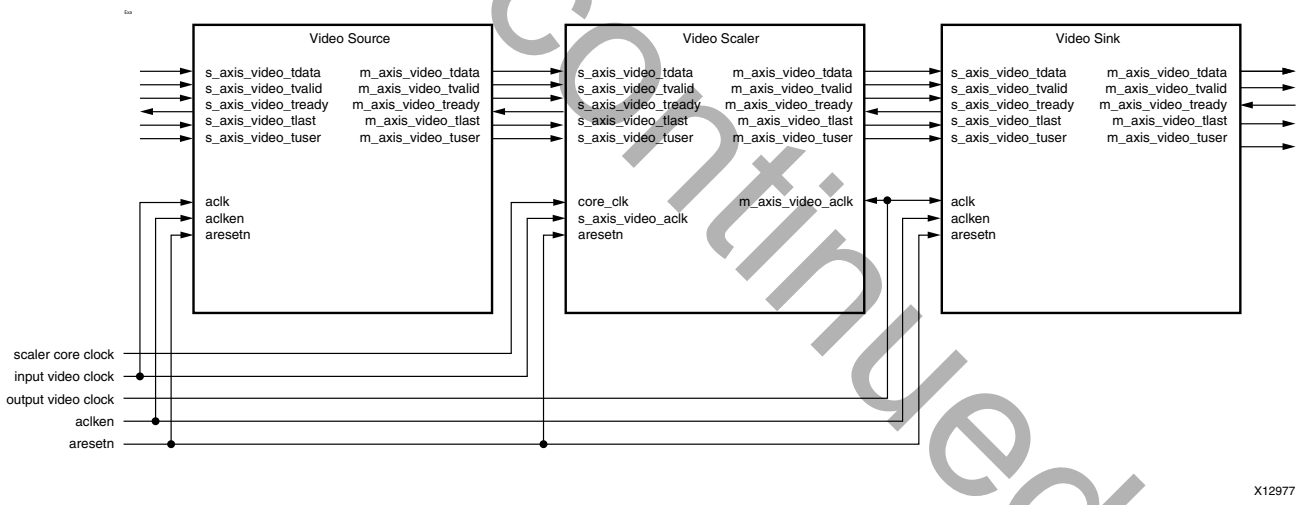

Figure 3-5: **Video Scaler Example**

In this case, the video source block works on a single clock domain, and sends video data out to the scaler on that clock domain. The scaler processes it using the `core_clk` domain, and passes data out on the scaler output clock domain. You can choose to make all of these clock domains independent, or to make two or all of these clock domains common.

The Video Scaler does not support any clock enable signals.

Scaler Aperture

This section explains how to define the scaler aperture using the appropriate dynamic control registers. The aperture is defined relative to the input timing signals.

Input Aperture Definition

It is vital to understand how to specify the scaler aperture properly. The scaler aperture is defined as the input data rectangle used to create the output data rectangle. The input values `aperture_start_line`, `aperture_end_line`, `aperture_start_pixel` and `aperture_end_pixel` need to be driven correctly.

For example, to scale from a rectangle of size 1280x720, set the input values as shown in Table 3-2.

Table 3-2: **Input Aperture: 720P**

Input	Value
aperture_start_pixel	0
aperture_end_pixel	1279
aperture_start_line	0
aperture_end_line	719

It is also important to understand how "line 0" and "pixel 0" are defined to ensure that these values are entered correctly. The pixel of active data (indicated by `s_axis_video_tvalid`=1, `s_axis_video_tready`=1) is defined as Line 0, pixel 0. An internal line counter is decoded to signal internally that the current line is indeed line 0. This line counter is reset on Start-of-frame (`s_axis_video_tuser` => 1), and increments end-of-line (`s_axis_video_tlast` => 1). Then the Scaler is at the edge of an aperture that is the same as the edge of the frame, it replicates the edge sample, for half the number of the filter taps. It does not do wrapping to the opposite edge, or fold back on previous pixels in the image.

Cropping

You can choose to select a small portion of the input image. To achieve this, set the `aperture_start_line`, `aperture_end_line`, `aperture_start_pixel` and `aperture_end_pixel` according to your requirements.

For example, from an input which is 720P, you may want to scale from a rectangle of size 80x60, starting at (pixel, line) = (20, 32). Set the values as shown in Table 3-3.

Table 3-3: **Input Aperture Values: Cropping**

Input	Value
aperture_start_pixel	20
aperture_end_pixel	99
aperture_start_line	32
aperture_end_line	91

Coefficients

This section describes the coefficients used by both the Vertical and Horizontal filter portions of the scaler, in terms of number, range, formatting and download procedures.

Coefficient Table

One single size-configurable, block RAM-based, Dual Port RAM block stores all H and V coefficients combined, and holds different coefficients for luma and chroma as desired.

This coefficient store can be populated with active coefficients as follows:

- Using the Coefficient Interface (see Coefficient Interface).
- By preloading using a .coe file
- Lanczos Coefficients option in the GUI

Coefficients that are preloaded using a .coe file remain in this memory until they are overwritten with coefficients loaded by the Coefficient Interface. Consequently, this is not possible when using Constant mode. Preloading with coefficients allows you an easy way of initializing the scaler from power-up.

You may want more than one coefficient set from which to choose. For example, it can be necessary to select different filter responses for different shrink factors. This is often true when down-scaling by different factors to eliminate aliasing artifacts. You can load (or preload using a .coe file) multiple coefficient sets.

The number of phases for each set can also vary, dependent upon the nature of the conversion, and how you have elected to generate and partition the coefficients. The maximum number of phases per set defines the size of the memory required to store them, and this can have an impact on resource usage. Careful selection of the parameters `max_phases` and `max_coef_sets` is paramount if optimal resource usage is important.

Each coefficient set is allocated an amount of space equal to 2^{max_phases}. Max_phases is a fixed parameter that is defined at compile time. However, it is not necessary for every set to have that many phases. The number of phases for each set can be different, provided you indicate how many phases there are in the current set being used, by setting the input register values `num_h_phases`, and `num_v_phases` accordingly. Without setting these correctly, invalid coefficients are selected.

Horizontal filter coefficients are stored in the lower half of the coefficient memory. Vertical filter coefficients are stored in the upper half of the coefficient memory. For each of the H and V sectors, luma coefficients occupy the lower half and chroma coefficients occupy the upper half. This method simplifies internal addressing. When the chroma format is set to

4:4:4., one set of coefficients is shared between all three channels (i.e., R, G, and B channels is scaled identically).

If you specify in the GUI that the Luma and Chroma filters share common coefficients, then there is no coefficient memory space available for chroma coefficients. In this case, you must not load chroma coefficients using the Coefficient interface, and must not specify chroma coefficients in the .coe file.

Similarly, if you have specified in the GUI that the Horizontal and Vertical filters share common coefficients, then there is no coefficient memory space available for Vertical coefficients. In this case, you must not load Vertical coefficients using the Coefficient interface, and must not specify Vertical coefficients in the .coe file.

Note: This option is only available if the number of horizontal taps is equal to the number of vertical taps.

Coefficient Interface

The scaler uses only one set of coefficients per frame period. To change to a different set of stored coefficients for the next frame, use the `h_coeff_set` and `v_coeff_set` dynamic register inputs.

You can load new coefficients into a different location in the coefficient store during some frame period **before** they are required. You can load a maximum of **one** coefficient set (including all of HY, HC, VY, VC components) per frame period. Subsequently, this coefficient set can be selected for use by controlling `h_coeff_set` and `v_coeff_set`.

Filter Coefficients can be loaded into the coefficient memory using the AXI4-Lite interface.

The 32-bit input word loaded via the `coef_data_in` register always holds two coefficients. The scaler supports 16-bit coefficient bit-widths. The word format is shown in Figure 3-6.

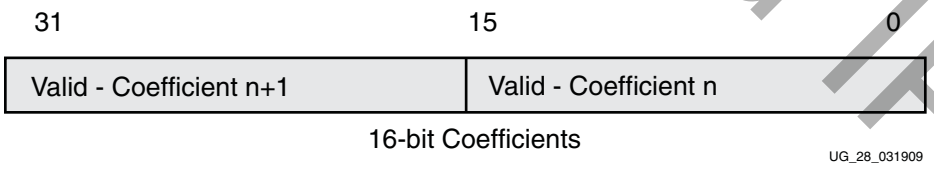

Figure 3-6: **Coefficient Write-Format on coef_data_in(31:0)**

Coefficients are written from the coefficient interface into a loading FIFO before being transferred into the main coefficient memory for use by the filters. Loading the FIFO must take place during the frame period before it is required. The transferal process from FIFO to coefficient memory takes place between the processing of 2 consecutive video frames, if a new coefficient set has indeed been loaded into the FIFO. Following SOF of every frame (`s_axis_video_tuser` = 1), `intr_coef_fifo_rdy` is asserted to indicate that the FIFO is ready to receive a coefficient. Following the delivery of the final coefficient of a set into

the scaler, `intr_coef_fifo_rdy` is driven low. The number of coefficient writes required for this to happen is dependent upon the number of taps and phases, and also on the settings of `Separate_hv_coefs` and `Separate_yc_coefs` parameters.

The guidelines are:

- The address `coef_set_addr` for all coefficients in one set must be written via the normal register interface.
- `coef_data_in` delivers two coefficients per 32-bit word. The lower word (bits 15:0) always holds the coefficient that is applied to the latest tap (the right-most or lowest). The word format is shown in Figure 3-6.
- All coefficients for one phase must be loaded sequentially via the `coef_data_in` register, starting with coef 0 and coef 1 [coef 0 is applied to the **newest** (right-most or lowest) input sample in the current filter aperture]. For an odd number of coefficients, the final upper 16 bits is ignored.
- All phases must be loaded sequentially starting at phase 0, and ending at phase (`max_phases-1`). This must always be observed, even if a particular set of coefficients has fewer active phases than `max_phases`.
- **For RGB/4:4:4**, when not sharing coefficients across H and V operations, for each dimension, one bank of coefficients must be loaded into the FIFO before they can be streamed into the coefficient memory. When sharing coefficients across H and V operations, it is only necessary to write coefficients for the H operation. This process is permitted to take as much time as desired by the system. This means that worst case, for a 12H-tap x 12V-tap 64-phase filter, you need to write 6 times per phase. If you have specified separate H and V coefficients, this is a total of **768** write operations per set.
- **For YC4:2:2 or YC4:2:0**, when not sharing coefficients across H and V operations or across Y and C operations, one bank of luma (Y) and chroma (C) coefficients must be loaded into the FIFO **for each dimension** before they can be streamed into the coefficient memory. When sharing coefficients across H and V operations, it is only necessary to write coefficients for the H operation. Also, when sharing coefficients across Y and C operations, it is only necessary to write coefficients for the Y operation. This process is permitted to take as much time as desired by the system. This means that worst case, for a 12H-tap x 12V-tap 64-phase filter, you need to write 6 times per phase. If you have specified separate H and V coefficients and separate Y and C coefficients, this is a total of **1536** write operations per set.
- Writing a new address to `coef_set_addr` resets the internal state-machine that oversees the coefficient loading procedure. An error condition is asserted if the loading procedure comes up less than **2 x** `max_phases*Max(num_h_taps, num_v_taps)` when `coef_set_addr` is updated.

Coefficient Readback

The Xilinx Video Scaler core also includes a coefficient readback feature. This is essentially the reverse of the write process, with the exception that it occurs for only one bank of coefficients at a time. The coefficient readback interface signals are shown in Table 3-4.

Table 3-4: **Coefficient Readback Registers and Interrupts**

Input	Description
coef_set_bank_rd_addr(15:8)	Coefficient set read-address
coef_set_bank_rd_addr(1:0)	Coefficient bank read-address. 00=HY 01=HC 10=VY 11=VC
coef_mem_rd_addr(15:8)	Coefficient phase read-address
coef_mem_rd_addr(3:0)	Coefficient tap read-address
coef_mem_output(15:0)	Coefficient readback output
intr_coef_mem_rdbk_rdy	Output flag indicating that the specified coefficient bank is ready for reading.

The basic steps for a coefficient readback are as follows:

1. Before changing the set and bank read address, set bit 3 of the Control register, `CoefMemRdEn`, to 0.
2. Using the `coef_set_bank_rd_addr`, provide a set number and bank number for the coefficient bank to read back.
3. Activate the new bank of coefficients by setting bit 3 of the Control register to 1. A Dual-Port RAM is then populated with that bank of coefficients.
4. Once the `intr_coef_mem_rdbk_rdy` interrupt has gone High, use `coef_mem_rd_addr` to provide the phase and tap number of the coefficient to read from that bank. The coefficient appears at `coef_mem_output`.

It is only possible to read back one bank of coefficients per frame period.

Coefficients can only be read from the Dual-Port RAM when control bit (3) is set High. However, it is only possible to populate it with a new coefficient bank when this bit is set Low.

Reading back coefficients do not cause image distortion, and can be executed during normal operation.

Examples of Coefficient Readback

The following C coding example shows how to read the coefficients from one bank, and print them out:

```
// Switch off coef_mem_rd_en bit in control register, by writing control reg bit 3.
  Scaler_WriteReg(Control_Reg_bit3, 0);

// Set up set and bank number to read
  Scaler_WriteReg(CoefSetBank_Reg, (SetNo<< 8) & BankNo);

// Switch on coef_mem_rd_en bit in control register.
  printf("Switch on coef_mem_rd_en bit in control register.\n");
  Scaler_WriteReg(Control_Reg_bit3, 1);

// Poll the intr_coef_mem_rdbk_rdy status register OR wait until at least 2 frames
will have passed:
  sleep(100);

// Now read the coefficients values for one bank:
  for (PhaseIndex = 0; PhaseIndex < PhaseNumberofPhases; PhaseIndex++)
  {
    printf("Phase %2d: ", PhaseIndex);
    for (TapIndex = 0; TapIndex < NumberOfTaps; TapIndex++)
    {
      // Write phase and tap number of desired coefficient into CoefMemRdAddr register
offset.
      CoefMemRdAddr = ((PhaseIndex & 0x000000ff)<<8) | TapIndex;
      Scaler_WriteReg(CoefMemRdAddr_Reg, CoefMemRdAddr);

      // Now read from the coefficient output register offset
      CoefValue = Scaler_ReadReg(CoeffMemOut_Reg);

      // Print coefficient values, handling negative values appropriately.
      if (CoefValue >32767)
         printf("%7d ", CoefValue-65536);
      else
         printf("%7d ", CoefValue);
    }
    printf("\n");
  }
```


Examples of Coefficient Set Generation and Loading

As mentioned, when data is fed in raster format, coefficient 0 is applied to the lowest tap in the aperture for the Vertical filter or for the right-most tap in the Horizontal filter. Following are a few examples of how to generate some coefficients and translate them into the correct format for downloading to the scaler.

Example 1: Num_h_taps = num_v_taps = 8; max_phases = 4

Table 3-5 shows a set of coefficients drawn from a sinc function.

Table 3-5: **Example 1 Decimal Coefficients**

Phase	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6	Tap 7
0	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000
1	-0.0600	0.0818	-0.1286	0.3001	0.9003	-0.1801	0.1000	-0.0693
2	-0.0909	0.1273	-0.2122	0.6366	0.6366	-0.2122	0.1273	-0.0909
3	-0.0693	0.1000	-0.1801	0.9003	0.3001	-0.1286	0.0818	-0.0600

In this example, a 32-point 1-D sinc function has been sub-sampled to generate four phases of eight coefficients each. Sub-sampling in this way usually results in a phases whose component coefficients rarely sum to 1.0 – this causes image distortion. The example MATLAB® m-code that follows shows how to normalize the phases to unity and how to express them as the 16-bit integers required by the hardware. For this process, `coef_width` = 16.

TIP: *Note that this is only pseudo code. Generation of actual coefficients is beyond the scope of this document.*

Refer to Answer Record 35262 for more information on coefficient generation for the video scaler.

```
% Subsample a Sinc function, and create 2D array
x=-(num_taps/2):1/num_phases:((num_taps/2)-1/num_phases);
coefs_2d=reshape(sinc(x), num_phases, num_taps)
format long

% Normalize each phase individually
for i=1:num_phases
   sum_phase = sum(coefs_2d(i,:));
   for j=1:num_taps
      norm_phases(i, j) = coefs_2d(i, j)/sum_phase;
   end
   % Check - Normalized values should sum to 1 in each phase
   norm_sum_phase = sum(norm_phases(i,:))
end

% Translate real to integer values with precision defined by coef_width
int_phases = round(((2^(coef_width-2))*norm_phases))
```

This generates the 2D array of integer values shown (in hexadecimal form) in Table 3-6.

Table 3-6: **Example 1 Normalized Integer Coefficients**

Phase	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6	Tap 7
0	0x0000	0x0000	0x0000	0x0000	0x4000	0x0000	0x0000	0x0000
1	0xFBEF	0x058C	0xF749	0x1457	0x3D04	0xF3CC	0x06C8	0xFB4E
2	0xF9AF	0x08D8	0xF143	0x2C36	0x2C36	0xF143	0x08D8	0xF9AF
3	0xFB4E	0x06C8	0xF3CC	0x3D04	0x1457	0xF749	0x058C	0xFBEF

It remains to format these values for the scaler.

The 16-bit coefficients must be coupled into 32-bit values for delivery to the HW. The resulting coefficient file for download is shown in Table 3-7.

The coefficients must be downloaded in the following order:

1. Horizontal Luma (always required).
2. Horizontal Chroma (required if not sharing Y and C coefficients).
3. Vertical Luma (required if not sharing H and V coefficients).
4. Vertical Chroma (required if not sharing H and V coefficients, and also not sharing Y and C coefficients).

Table 3-7: **Example 1 Coefficient Set Download Format**

Horizontal Filter Coefficients for Luma				Horizontal Filter Coefficients for Chroma			
	Load Sequence Number	Value	Calculation Ph= Phase #, T= Tap #	Load Sequence Number	Value	Calculation Ph= Phase #, T= Tap #	
Phase 0	1	0x00000000	(Ph0 T1 << 16) | Ph0 T0	17	0x00000000	(Ph0 T1 << 16) | Ph0 T0	Phase 0
	2	0x00000000	(Ph0 T3 << 16) | Ph0 T2	18	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	3	0x00004000	(Ph0 T5 << 16) | Ph0 T4	19	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	4	0x00000000	(Ph0 T7 << 16) | Ph0 T6	20	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
Phase 1	5	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	21	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	Phase 1
	6	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	22	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	
	7	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	23	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	
	8	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	24	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	
Phase 2	9	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	25	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	Phase 2
	10	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	26	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	
	11	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	27	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	
	12	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	28	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	

Table 3-7: **Example 1 Coefficient Set Download Format *(Cont'd)***

Phase 3	13	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	29	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	**Phase 3**
	14	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	30	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	
	15	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	31	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	
	16	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	32	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	33	0x00000000	(Ph0 T1 << 16) | Ph0 T0	49	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	34	0x00000000	(Ph0 T3 << 16) | Ph0 T2	50	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	35	0x00004000	(Ph0 T5 << 16) | Ph0 T4	51	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	36	0x00000000	(Ph0 T7 << 16) | Ph0 T6	52	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
Phase 1	37	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	53	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	**Phase 1**
	38	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	54	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	
	39	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	55	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	
	40	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	56	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	
Phase 2	41	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	57	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	**Phase 2**
	42	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	58	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	
	43	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	59	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	
	44	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	60	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	

Table 3-7: **Example 1 Coefficient Set Download Format *(Cont'd)***

Phase 3	45	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	61	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	**Phase 3**
	46	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	62	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	
	47	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	63	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	
	48	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	64	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	

Example 2: Num_h_taps = num_v_taps = 8; max_phases = 5, 6, 7 or 8; num_h_phases = num_v_phases = 4

If the `max_phases` parameter is greater than the number of phases in the set being loaded, load default coefficients into the unused locations. Example 2 is an extended version of Example 1 to show this. Table 3-8 shows the same 4-phase coefficient set loaded into the scaler when `num_h_phases` = 4, `num_v_phases` = 4 and `max_phases` is greater than 4 (`max_phases` = 5, 6, 7 or 8, `num_h_taps` = 8, `num_v_taps` = 8).

Note that:

1. If `max_phases` is not equal to an integer power of 2, then the number of phases to be loaded is rounded up to the next integer power of 2. See Example 2 (Table 3-8). Unused phases should be loaded with zeros.
2. The number of values loaded per phase is ***not*** rounded to the nearest power of 2. See Example 3 (Table 3-11).

Table 3-8: **Example 2 Coefficient Set Download Format**

	Horizontal Filter Coefficients for Luma			Horizontal Filter Coefficients for Chroma			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	1	0x00000000	(Ph0 T1 << 16) | Ph0 T0	33	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	2	0x00000000	(Ph0 T3 << 16) | Ph0 T2	34	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	3	0x00004000	(Ph0 T5 << 16) | Ph0 T4	35	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	4	0x00000000	(Ph0 T7 << 16) | Ph0 T6	36	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
Phase 1	5	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	37	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	**Phase 1**
	6	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	38	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	
	7	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	39	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	
	8	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	40	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	

Table 3-8: **Example 2 Coefficient Set Download Format *(Cont'd)***

Phase 2	9	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	41	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0
	10	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	42	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2
	11	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	43	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4
	12	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	44	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6
Phase 3	13	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	45	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0
	14	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	46	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2
	15	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	47	0xF7491457	(Ph3 T5 << 16) | Ph3 T4
	16	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	48	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6
Phase 4	17	0x00000000	N/A Dummy coef	49	0x00000000	N/A Dummy coef
	18	0x00000000	N/A Dummy coef	50	0x00000000	N/A Dummy coef
	19	0x00000000	N/A Dummy coef	51	0x00000000	N/A Dummy coef
	20	0x00000000	N/A Dummy coef	52	0x00000000	N/A Dummy coef
Phase 5	21	0x00000000	N/A Dummy coef	53	0x00000000	N/A Dummy coef
	22	0x00000000	N/A Dummy coef	54	0x00000000	N/A Dummy coef
	23	0x00000000	N/A Dummy coef	55	0x00000000	N/A Dummy coef
	24	0x00000000	N/A Dummy coef	56	0x00000000	N/A Dummy coef
Phase 6	25	0x00000000	N/A Dummy coef	57	0x00000000	N/A Dummy coef
	26	0x00000000	N/A Dummy coef	58	0x00000000	N/A Dummy coef
	27	0x00000000	N/A Dummy coef	59	0x00000000	N/A Dummy coef
	28	0x00000000	N/A Dummy coef	60	0x00000000	N/A Dummy coef
Phase 7	29	0x00000000	N/A Dummy coef	61	0x00000000	N/A Dummy coef
	30	0x00000000	N/A Dummy coef	62	0x00000000	N/A Dummy coef
	31	0x00000000	N/A Dummy coef	63	0x00000000	N/A Dummy coef
	32	0x00000000	N/A Dummy coef	64	0x00000000	N/A Dummy coef
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**		
	Addr	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Addr**	**Value**	**Calculation** Ph= Phase #, T= Tap #
Phase 0	65	0x00000000	(Ph0 T1 << 16) | Ph0 T0	97	0x00000000	(Ph0 T1 << 16) | Ph0 T0
	66	0x00000000	(Ph0 T3 << 16) | Ph0 T2	98	0x00000000	(Ph0 T3 << 16) | Ph0 T2
	67	0x00004000	(Ph0 T5 << 16) | Ph0 T4	99	0x00004000	(Ph0 T5 << 16) | Ph0 T4
	68	0x00000000	(Ph0 T7 << 16) | Ph0 T6	100	0x00000000	(Ph0 T7 << 16) | Ph0 T6

Table 3-8: **Example 2 Coefficient Set Download Format *(Cont'd)***

Phase 1	69	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	101	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0
	70	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	102	0x1457F749	(Ph1 T3 << 16) | Ph1 T2
	71	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	103	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4
	72	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	104	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6
Phase 2	73	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	105	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0
	74	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	106	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2
	75	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	107	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4
	76	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	108	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6
Phase 3	77	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	109	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0
	78	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	110	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2
	79	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	111	0xF7491457	(Ph3 T5 << 16) | Ph3 T4
	80	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	112	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6
Phase 4	81	0x00000000	N/A Dummy coef	113	0x00000000	N/A Dummy coef
	82	0x00000000	N/A Dummy coef	114	0x00000000	N/A Dummy coef
	83	0x00000000	N/A Dummy coef	115	0x00000000	N/A Dummy coef
	84	0x00000000	N/A Dummy coef	116	0x00000000	N/A Dummy coef
Phase 5	85	0x00000000	N/A Dummy coef	117	0x00000000	N/A Dummy coef
	86	0x00000000	N/A Dummy coef	118	0x00000000	N/A Dummy coef
	87	0x00000000	N/A Dummy coef	119	0x00000000	N/A Dummy coef
	88	0x00000000	N/A Dummy coef	120	0x00000000	N/A Dummy coef
Phase 6	89	0x00000000	N/A Dummy coef	121	0x00000000	N/A Dummy coef
	90	0x00000000	N/A Dummy coef	122	0x00000000	N/A Dummy coef
	91	0x00000000	N/A Dummy coef	123	0x00000000	N/A Dummy coef
	91	0x00000000	N/A Dummy coef	124	0x00000000	N/A Dummy coef
Phase 7	93	0x00000000	N/A Dummy coef	125	0x00000000	N/A Dummy coef
	94	0x00000000	N/A Dummy coef	126	0x00000000	N/A Dummy coef
	95	0x00000000	N/A Dummy coef	127	0x00000000	N/A Dummy coef
	96	0x00000000	N/A Dummy coef	128	0x00000000	N/A Dummy coef

Example 3: Num_h_taps = 9; num_v_taps = 7; max_phases = num_h_phases = num_v_phases = 4

Now consider the case where the number of taps in the Horizontal dimension is different to that in the Vertical dimension. For this case, when loading the coefficients for the dimension for which the number of taps is **smaller**, each **phase** of coefficients must be padded with zeros up to the larger number of taps.

Example coefficients are shown in hexadecimal form in Table 3-9 (horizontal) and Table 3-10 (vertical).

Table 3-9: **Example 9-Tap Coefficients**

Phase	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6	Tap 7	Tap 8
0	0x0000	0x0000	0x0000	0x0000	0x4000	0x0000	0x0000	0x0000	0x0000
1	0xFFB1	0x0123	0x047C	0x10C6	0x3A26	0xF5F0	0x037D	0xFF0A	0x0046
2	0xFF84	0x01D1	0xF865	0x2490	0x2A42	0xF3D0	0x0490	0xFEB4	0x0060
3	0xFF9E	0x017E	0xF93F	0x3619	0x14D7	0xF846	0x0312	0xFF1B	0x0043

Table 3-10: **Example 7-Tap Coefficients**

Phase	Tap 0	Tap 1	Tap 2	Tap 3	Tap 4	Tap 5	Tap 6
0	0x0000	0x0000	0x0000	0x4000	0x0000	0x0000	0x0000
1	0x006D	0xFD69	0x0F04	0x3A81	0xF6FE	0x0204	0xFFA4
2	0x00B2	0xFB85	0x2160	0x2B58	0xF4E0	0x02B0	0xFF81
3	0x0097	0xFBE1	0x332B	0x1627	0xF8B1	0x01DF	0xFFA5

The resulting coefficient file for download is shown in Table 3-11.

Table 3-11: **Example 3 Coefficient Set Download Format**

	Horizontal Filter Coefficients for Luma			Horizontal Filter Coefficients for Chroma			
	Load Sequence Number	Value	Calculation Ph= Phase #, T= Tap #	Load Sequence Number	Value	Calculation Ph= Phase #, T= Tap #	
Phase 0	1	0x00000000	(Ph0 T1 << 16) | Ph0 T0	21	0x00000000	(Ph0 T1 << 16) | Ph0 T0	Phase 0
	2	0x00000000	(Ph0 T3 << 16) | Ph0 T2	22	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	3	0x00004000	(Ph0 T5 << 16) | Ph0 T4	23	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	4	0x00000000	(Ph0 T7 << 16) | Ph0 T6	24	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
	5	0x00000000	(0 << 16) | Ph0 T8	25	0x00000000	(0 << 16) | Ph0 T8	

Table 3-11: **Example 3 Coefficient Set Download Format *(Cont'd)***

Phase 1	6	0x0123FFB1	(Ph1 T1 << 16) | Ph1 T0	26	0x0123FFB1	(Ph1 T1 << 16) | Ph1 T0	**Phase 1**
	7	0x10C6047C	(Ph1 T1 << 16) | Ph1 T2	27	0x10C6047C	(Ph1 T1 << 16) | Ph1 T2	
	8	0XF5F03A26	(Ph1 T1 << 16) | Ph1 T4	28	0XF5F03A26	(Ph1 T1 << 16) | Ph1 T4	
	9	0XFF0A037D	(Ph1 T1 << 16) | Ph1 T6	29	0XFF0A037D	(Ph1 T1 << 16) | Ph1 T6	
	10	0x00000046	(0 << 16) | Ph1 T8	30	0x00000046	(0 << 16) | Ph1 T8	
Phase 2	11	0x01D1FF84	(Ph2 T1 << 16) | Ph2 T0	31	0x01D1FF84	(Ph2 T1 << 16) | Ph2 T0	**Phase 2**
	12	0x2490F865	(Ph2 T3 << 16) | Ph2 T2	32	0x2490F865	(Ph2 T3 << 16) | Ph2 T2	
	13	0XF3D02A2	(Ph2 T5 << 16) | Ph2 T4	33	0XF3D02A2	(Ph2 T5 << 16) | Ph2 T4	
	14	0XFEB40490	(Ph2 T7 << 16) | Ph2 T6	34	0XFEB40490	(Ph2 T7 << 16) | Ph2 T6	
	15	0x00000060	(0 << 16) | Ph2 T8	35	0x00000060	(0 << 16) | Ph2 T8	
Phase 3	16	0x017EFF9E	(Ph3 T1 << 16) | Ph3 T0	36	0x017EFF9E	(Ph3 T1 << 16) | Ph3 T0	**Phase 3**
	17	0x3619F93F	(Ph3 T3 << 16) | Ph3 T2	37	0x3619F93F	(Ph3 T3 << 16) | Ph3 T2	
	18	0XF84614D7	(Ph3 T1 << 16) | Ph3 T4	38	0XF84614D7	(Ph3 T1 << 16) | Ph3 T4	
	19	0XFF1B0312	(Ph3 T1 << 16) | Ph3 T6	39	0XFF1B0312	(Ph3 T1 << 16) | Ph3 T6	
	20	0x00000043	(0 << 16) | Ph3 T8	40	0x00000043	(0 << 16) | Ph3 T8	
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	41	0x00000000	(Ph0 T1 << 16) | Ph0 T0	61	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	42	0x40000000	(Ph0 T3 << 16) | Ph0 T2	62	0x40000000	(Ph0 T3 << 16) | Ph0 T2	
	43	0x00000000	(Ph0 T5 << 16) | Ph0 T4	63	0x00000000	(Ph0 T5 << 16) | Ph0 T4	
	44	0x00000000	(0 << 16) | Ph0 T6	64	0x00000000	(0 << 16) | Ph0 T6	
	45	0x00000000	N/A dummy coef	65	0x00000000	N/A dummy coef	

Table 3-11: **Example 3 Coefficient Set Download Format *(Cont'd)***

Phase 1	46	0XFD69006D	(Ph1 T1 << 16) | Ph1 T0	66	0XFD69006D	(Ph1 T1 << 16) | Ph1 T0	Phase 1
	47	0x3A810F04	(Ph1 T1 << 16) | Ph1 T2	67	0x3A810F04	(Ph1 T1 << 16) | Ph1 T2	
	48	0X0204F6FE	(Ph1 T1 << 16) | Ph1 T4	68	0X0204F6FE	(Ph1 T1 << 16) | Ph1 T4	
	49	0X0000FFA4	(0 << 16) | Ph1 T6	69	0X0000FFA4	(0 << 16) | Ph1 T6	
	50	0x00000000	N/A dummy coef	70	0x00000000	N/A dummy coef	
Phase 2	51	0XFB8500B2	(Ph2 T1 << 16) | Ph2 T0	71	0XFB8500B2	(Ph2 T1 << 16) | Ph2 T0	Phase 2
	52	0x2B582160	(Ph2 T3 << 16) | Ph2 T2	72	0x2B582160	(Ph2 T3 << 16) | Ph2 T2	
	53	0X02B0F4E0	(Ph2 T5 << 16) | Ph2 T4	73	0X02B0F4E0	(Ph2 T5 << 16) | Ph2 T4	
	54	0X0000FF81	(0 << 16) | Ph2 T6	74	0X0000FF81	(0 << 16) | Ph2 T6	
	55	0x00000000	N/A dummy coef	75	0x00000000	N/A dummy coef	
Phase 3	56	0XFBE10097	(Ph3 T1 << 16) | Ph3 T0	76	0XFBE10097	(Ph3 T1 << 16) | Ph3 T0	Phase 3
	57	0x1627332B	(Ph3 T3 << 16) | Ph3 T2	77	0x1627332B	(Ph3 T3 << 16) | Ph3 T2	
	58	0X01DFF8B1	(Ph3 T1 << 16) | Ph3 T4	78	0X01DFF8B1	(Ph3 T1 << 16) | Ph3 T4	
	59	0X0000FFA5	(0 << 16) | Ph3 T6	79	0X0000FFA5	(0 << 16) | Ph3 T6	
	50	0x00000000	N/A dummy coef	80	0x00000000	N/A dummy coef	

Coefficient Preloading Using a .coe File

To preload the scaler with coefficients (mandatory when in Fixed mode), you must specify, using the GUI, a .coe file that contains the coefficients you want to use. It is important that the .coe file specified is in the correct format. The coefficients specified in the .coe file become hard-coded into the hardware during synthesis.

Generating .coe Files

Generating .coe files can be accomplished by either generating the coefficient file from the C Model, by using the Lanczos coefficients option in the GUI or developing a custom set of coefficients. Developing a custom set of coefficients is a very complex and subjective operation, and is beyond the scope of this document. Refer to Answer Record 35262 for more information on generating video scaler coefficients.

Format for .coe Files

The guidelines for creating a .coe file are:

- Coefficients can be specified in either 16-bit binary form or signed decimal form.
- First line of a 16-bit binary file must be `memory_initialization_radix=2;`
- First line of a signed decimal file must be `memory_initialization_radix=10;`
- Second line of all .coe files must be `memory_initialization_vector=`
- All coefficient entries must end with a comma (",") except the final entry which must end with a semicolon ";".
- Final entry must have a carriage return at the end after the semicolon.
- All coefficient sets must be listed consecutively, starting with set 0.
- All sets in the file must be of equal size in terms of the number of coefficient entries.
- Number of coefficient entries in all sets depends upon:
  - Max_coef_sets
  - Max_phases
  - Max_taps (=max(num_h_taps, num_v_taps))
  - User setting for "Separate Y/C coefficients"
  - User setting for "Chroma_format"
  - User setting for "Separate H/V coefficients"

  The simplest method is to specify an intermediate value `num_banks`:

```
num_banks=4;

if (Separate H/V coefficients = 0) then

    num_banks := num_banks/2;

end;

if (Separate Y/C coefficients = 0) or (chroma_format=4:4:4)
then

    num_banks := num_banks/2;

end;
```

  Consequently, the number of entries in the .coe file can be defined as:

```
num_coefs_in_coe_file = max_coef_sets x num_banks x max_phases x max_taps
```

- Within each set, coefficient banks must be specified in the following order:

Table 3-12: **Ordering of Coefficients in .coe File for Different Coefficient Sharing Options**

Separate Y/C Coefficients	Separate H/V Coefficients	Bank Order in .coe File
True	True	HY, HC, VY, VC
True	False	H, V
False	True	Y, C
False	False	Single set only

- Within each bank, all phases must be listed consecutively, starting with phase 0, followed by phase 1, etc.
- The number of phases specified (per bank) in the .coe file must be equal to Max_Phases, even for filters that use fewer phases. Set all coefficients in unused phases to 0 (decimal) or 0000000000000000 (16b binary).
- Within each phase, all coefficients must be listed consecutively. The first specified coefficient for any phase represents the value applied to the newest (rightmost or lowest) tap in the aperture.

Table 3-13 shows an example of a .coe file with the following specification:

num_h_taps = num_v_taps = 12;

max_phases = 4;

max_coef_sets = 1;

Separate H/V Coefficients = False;

Separate Y/C Coefficients = False;

Both signed decimal and 16-bit binary forms are shown.

Table 3-13: **.coe File Example 1**

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
N/A	N/A	1	memory_initialization_radix=10;	memory_initialization_radix=2;
		2	memory_initialization_vector=	memory_initialization_vector=
0	0	3	0,	0000000000000000,
0	1	4	162,	0000000010100010,
0	2	5	0,	0000000000000000,
0	3	6	-1069,	1111101111010011,
0	4	7	0,	0000000000000000,
0	5	8	5199,	0001010001001111,

Table 3-13: **.coe File Example 1 *(Cont'd)***

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
0	6	9	8167,	0001111111100111,
0	7	10	4457,	0001000101101001,
0	8	11	0,	0000000000000000,
0	9	12	-616,	1111110110011000,
0	10	13	0,	0000000000000000,
0	11	14	85,	0000000001010101,
1	0	15	28,	0000000000011100,
1	1	16	155,	0000000010011011,
1	2	17	-186,	1111111101000110,
1	3	18	-1062,	1111101111001010,
1	4	19	960,	0000001111000000,
1	5	20	6311,	0001100010100111,
1	6	21	7842,	0001111010100010,
1	7	22	3246,	0000110010101110,
1	8	23	-538,	1111110111100110,
1	9	24	-518,	1111110111111010,
1	10	25	72,	0000000001001000,
1	11	26	73,	0000000001001001,
2	0	27	53,	0000000000110101,
2	1	28	125,	0000000001111101,
2	2	29	-366,	1111111010010010,
2	3	30	-890,	1111110010000110,
2	4	31	2060,	0000100000001100,
2	5	32	7209,	0001110000101001,
2	6	33	7209,	0001110000101001,
2	7	34	2060,	0000100000001100,
2	8	35	-890,	1111110010000110,
2	9	36	-366,	1111111010010010,
2	10	37	125,	0000000001111101,
2	11	38	53,	0000000000110101,
3	0	39	73,	0000000001001001,
3	1	40	72,	0000000001001000,
3	2	41	-518,	1111110111111010,

Table 3-13: **.coe File Example 1 *(Cont'd)***

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
3	3	42	-538,	1111110111100110,
3	4	43	3246,	0000110010101110,
3	5	44	7842,	0001111010100010,
3	6	45	6311,	0001100010100111,
3	7	46	960,	0000001111000000,
3	8	47	-1062,	1111101111001010,
3	9	48	-186,	1111111101000110,
3	10	49	155,	0000000010011011,
3	11	50	28;	0000000000011100;
3		51	""	""

Table 3-14 shows an example of a .coe file with the following specification:

num_h_taps = 12, num_v_taps = 12;

max_phases = 4;

max_coef_sets = 2;

Separate H/V Coefficients = True;

Separate Y/C Coefficients = True;

Just signed decimal form is shown. For clarity, the same coefficient values have been used for each bank. Be aware that these are not realistic coefficients. Also note that this list includes ellipses to show continuation, and that it does not include a complete set of coefficients.

Table 3-14: **.coe File Example 2**

Set	Bank	Phase	Tap	File line-number	Line Text
N/A				1	memory_initialization_radix=10;
				2	memory_initialization_vector=
0	0 (HY)	0	0	3	0,
0	0 (HY)	0	1	4	162,
0	0 (HY)	0	2	5	0,
0	0 (HY)	0	3	6	-1069,
0	0 (HY)	0	...	...	...
0	0 (HY)	1	0	15	28,

Table 3-14: **.coe File Example 2 *(Cont'd)***

0	0 (HY)	1	1	16	155,
0	0 (HY)	1	2	17	-186,
0	0 (HY)	…	…	…	…
0	0 (HY)	3	0	39	73,
0	0 (HY)	3	1	40	72,
0	0 (HY)	3	…	…	…
0	0 (HY)	3	11	50	28,
0	1 (HC)	0	0	51	0,
0	1 (HC)	0	1	52	162,
0	1 (HC)	0	2	53	0,
0	…	…	…	…	…
0	1 (HC)	3	0	87	73,
0	1 (HC)	3	1	88	72,
0	1 (HC)	3	…	…	…
0	1 (HC)	3	11	98	28,
0	2 (VY)	0	0	99	0,
0	2 (VY)	0	1	100	162,
0	2 (VY)	0	2	101	0,
0	…	…	…	…	…
0	2 (VY)	3	0	135	73,
0	2 (VY)	3	1	136	72,
0	2 (VY)	3	…	…	…
0	2 (VY)	3	11	146	28,
0	3 (VC)	0	0	147	0,
0	3 (VC)	0	1	148	162,
0	3 (VC)	0	2	149	0,
0	…	…	…	…	…
0	3 (VC)	3	0	183	73,
0	3 (VC)	3	1	184	72,
0	3 (VC)	3	…	…	…
0	3 (VC)	3	11	194	28,
1	0 (HY)	0	0	195	0,
1	0 (HY)	0	1	196	162,
1	0 (HY)	0	2	197	0,

Table 3-14: **.coe File Example 2 *(Cont'd)***

1	0 (HY)	...	...	...	...
1	0 (HY)	3	11	242	28
1	1 (HC)	0	0	243	0,
1	...	...	...	...	...
1	2 (VY)	0	0	291	0,
1	...	...	...	...	...
1	3 (VC)	3	0	375	73,
1	3 (VC)	3	1	376	72,
1	3 (VC)	3	...	...	...
1	3 (VC)	3	11	386	28;
-	-	-	-	387	""

Table 3-15 shows an example of a .coe file with the following specification:

num_h_taps = 4, num_v_taps = 3;

max_phases = 4;

max_coef_sets = 1;

Separate H/V Coefficients = True;

Separate Y/C Coefficients = False;

Just signed decimal form is shown.

Table 3-15: **.coe File Example 3**

Bank	Phase	Tap	File line-number	Line Text	Notes
N/A			1	memory_initialization_radix=10;	
			2	memory_initialization_vector=	
0 (H)	0	0	3	-104,	
0 (H)	0	1	4	1018,	
0 (H)	0	2	5	15364,	
0 (H)	0	3	6	106,	
0 (H)	1	0	7	-240,	
0 (H)	1	1	8	4793,	
0 (H)	1	2	9	12022,	
0 (H)	1	3	10	-191,	

Table 3-15: **.coe File Example 3 *(Cont'd)***

0 (H)	2	0	11	-282,	
0 (H)	2	1	12	8474,	
0 (H)	2	2	13	8474,	
0 (H)	2	3	14	-282,	
0 (H)	3	0	15	-191,	
0 (H)	3	1	16	12022,	
0 (H)	3	2	17	4793,	
0 (H)	3	3	18	-240,	
1 (V)	0	0	19	86,	
1 (V)	0	1	20	16212,	
1 (V)	0	2	21	86,	
1 (V)	-	-	22	0,	Padding value
1 (V)	1	0	23	512,	
1 (V)	1	1	24	16068,	
1 (V)	1	2	25	-197,	
1 (V)	-	-	26	0,	Padding value
1 (V)	2	0	27	1243,	
1 (V)	2	1	28	15539,	
1 (V)	2	2	29	-398,	
1 (V)	-	-	30	0,	Padding value
1 (V)	3	0	31	2829,	
1 (V)	3	1	32	14099,	
1 (V)	3	2	33	-544,	
1 (V)	-	-	34	0;	Padding value
-	-	-	35	""	

Lanczos Coefficients GUI Option

This option in the GUI generates the coefficients automatically for the Video Scaler core. These generated coefficients automatically update when the configurations for the Scaler is changed. The parameters on which the coefficient generation depends on the number of H and V Taps and Phases, and the selection made for separate Luma (Y) and Chroma (C) filters and the Horizontal (H) and Vertical (V) Filters. When the AXI4-Lite interface is selected, then Maximum number of Phases and Maximum number of Coefficient sets determine the

coefficients generated. This file is generated in the folder where the project is present. When the core is generated this .coe file generated is used to generate the .mif file.

IMPORTANT: *When the number of coefficient sets is more than one then the .coe file generated in this method has coefficients updated for one set. The remaining sets are populated with zeros.*

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 9]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 7]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 8]

Customizing and Generating the Core

This chapter includes information for using Xilinx tools to customize and generate the core using Vivado tools.

Vivado Integrated Design Environment (IDE)

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 5] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 7].

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 9] 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.

Interface

The Video Scaler core is configured through the Vivado Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time. The GUI consists of Interface tab, Parameters tab and Fixed Mode tab, used to configure the core in Constant (fixed) mode.

Figure 4-1 shows the Interface page and Figure 4-2 shows the Parameters page of the Video Scaler GUI.

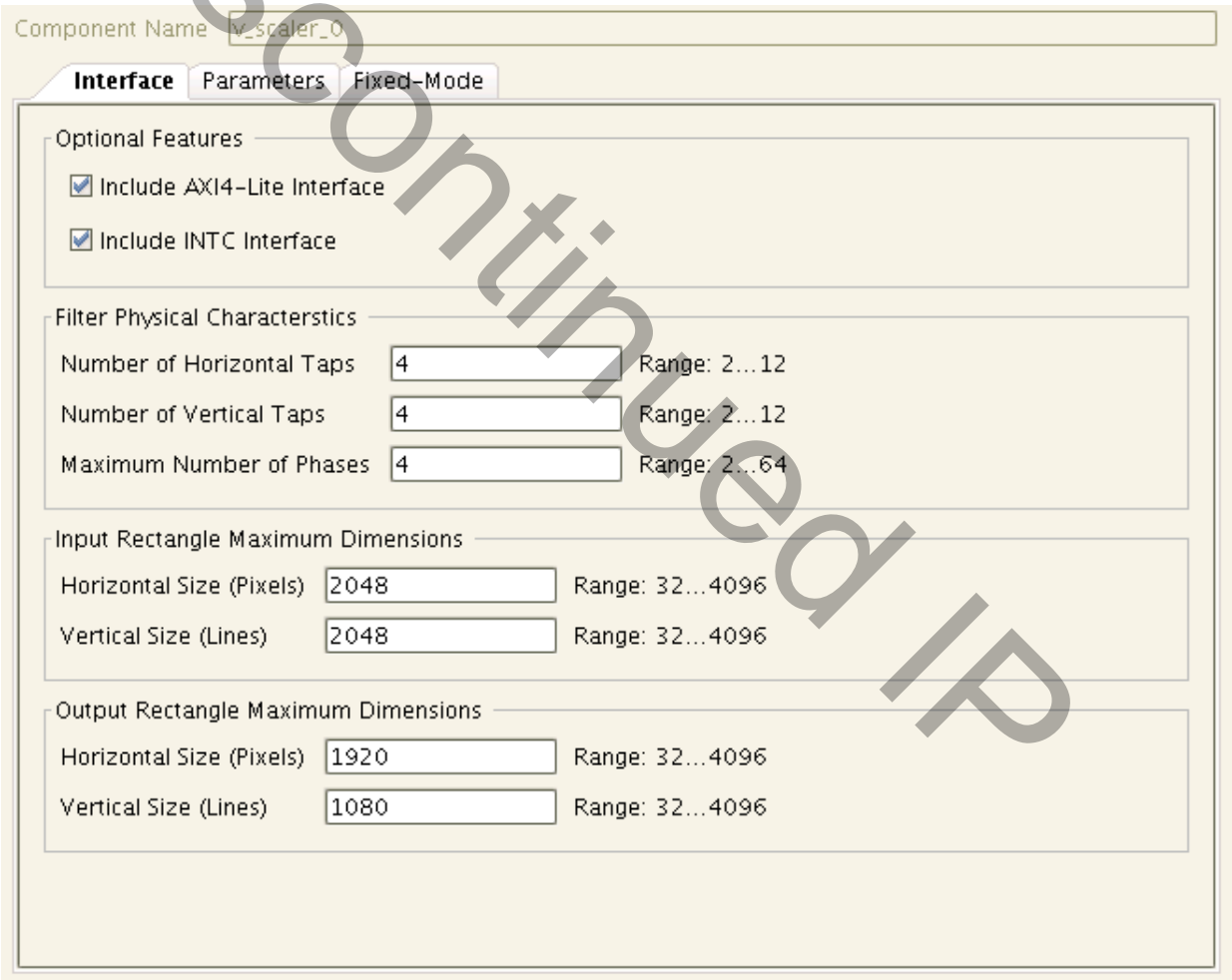

Figure 4-1: **Video Scaler Interface Screen**

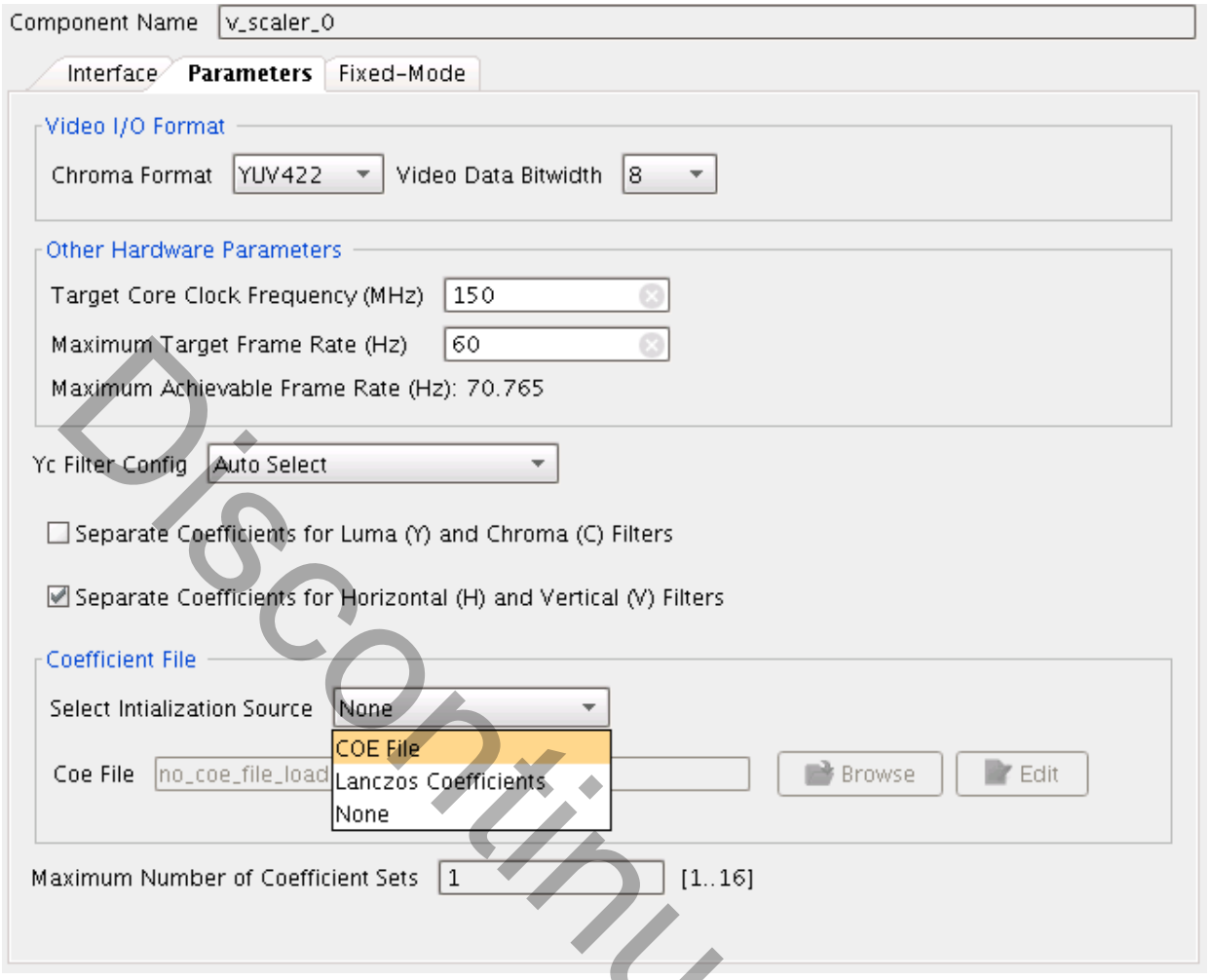

Figure 4-2: **Video Scaler Parameters Screen**

The first two pages display 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 "_". The name "v_scaler_v8_1" is not allowed.

Optional Features:

- **AXI4-Lite Register Interface**: When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, refer to Chapter 2, Common Interface Signals.

Number of Horizontal Taps: This represents the number of multipliers that can be used in the system for the horizontal filter, and can vary between 2 and 12 inclusive. You should be aware that increasing this number increases XtremeDSP slice usage.

Number of Vertical Taps: This represents the number of multipliers that can be used in the system for the vertical filter, and can vary between 2 and 12 inclusive. You should be aware that increasing this number increases XtremeDSP slice usage.

Input/output rectangle Maximum Frame Dimensions: These fields represent the maximum anticipated rectangle size on the input and output of the Video Scaler. The rectangle can vary between 32x32 through 4096x4096. These dimensions affect BRAM usage in the input and output line-buffers, and in the Vertical filter line-stores. They also have an effect on the calculation of the maximum frame-rate achievable when using the scaler core.

Max Number of Phases: This represents the maximum number of phases that the designer intends for a particular system. It can vary between 2 and 16 inclusive, but also can be set to 32 or 64. Setting this value high has two consequences: increased coefficient storage (block RAM), and increased time required to download each coefficient set. When the AXI4-Lite control interface is not selected, this parameter is greyed out - the number of H-phases and V-phases can be entered on page 2 of the GUI.

Video Data Bitwidth: 8, 10, or 12 bits. This specifies both the input and output video component bitwidths. When using IP Integrator, this parameter is automatically computed based on the Video Data Bitwidth of the video IP core connected to the slave AXI-Stream video interface.

Max Coef Sets: This represents the maximum number of sets of coefficients that can be stored internally to the scaler. It be set vary between 2 and 16. The coefficient set to be used during the scaling of the current frame is selected using the `h_coeff_set` and `v_coeff_set` controls. Increasing this value simply increases block RAM usage.

Chroma Format: Set this according to the chroma format required, either 422 (default), YUV420, YUV444, or RGB. Selecting YUV420 causes greater block RAM usage to align luma and chroma vertical apertures prior to the filters, and to realign the output lines after the filters. When using IP Integrator, this parameter is automatically computed based on the Chroma Format of the video IP core connected to the slave AXI-Stream video interface.

YC Filter Configuration: When running YUV420 or YUV422 data, the scaler can be configured to perform Y and C operations in parallel (two engines) or sequentially (one engine). Selecting **Auto** allows the tool to select whether to use single- or dual engines. The Parameters tab indicates the estimated maximum frame-rate achievable given you parameter settings. It makes this decision according to the specified desired frame rate. You can also manually select between the two options. When in YUV444/RGB mode, the scaler is implemented with three engines in parallel.

When the Chroma format is specified as YUV444/RGB, the triple-engine parallel architecture is always selected. Otherwise, selection between the YC Sequential or Parallel options can be achieved automatically (YC Filter Configuration = Auto Select) or manually in the GUI (see Figure 4-3).

The primary goal of selecting the correct architecture is to optimize resource usage for a worst-case operational scenario. When **Auto Select** is selected, the system establishes the worst case from the following input parameters:

- Input maximum rectangle size
- Output maximum rectangle size
- Target clock-frequency
- Desired frame rate

The pseudo-code calculation made the **Auto Select** option follows:

```
OverheadMultiplier := 1.15;
max_pixels := max(MaxHSizeIn, MaxHSizeOut);
max_lines := max(MaxVSizeIn, MaxVSizeOut);
max_frame_cycles := max_pixels * max_lines * OverHeadMultiplier;
MaxFrameRateOneComponent := (TgtFMax * 1000000)/max_frame_cycles;
if (TargetFrameRate <= MaxFrameRateOneComponent/2) then
   Use Single engine
else
   Use Dual engine
end if;
```

The Parameters tab shows the estimated maximum achievable frame rate given the above information using a similar calculation as shown in the sample. You are advised to take a look at this value, and can elect to force the GUI one way or the other. This is advisable in cases where, for example, an overhead per frame higher than 15% is needed. This overhead is intended as a general way of representing inactive periods in a frame (such as blanking), but also includes filter flushing time, state-machine initialization, and others.

Coefficient File Input: You can specify a .coe file to preload the coefficient store with coefficients or use the Lanczos Coefficients option. When using Constant mode, this is a necessary step. The .coe file format and the Lanczos Coefficient option are described in more detail in Coefficients in Chapter 3.

You can specify whether the same coefficients are used for Y and C filter operations.You can also specify whether the H and V operations use the same coefficients. This is only an option if the specified number of horizontal taps is equal to the specified number of vertical taps. Specifying the same coefficients in this way can make for a smaller implementation.

Coefficient File Input: You must specify a .coe file so that the coefficients are hard-coded into the netlist. This is described in more detail in Coefficients in Chapter 3.

Constant mode has the following restrictions:

- A single coefficient set must be specified using a .coe file; this is the only way to populate the coefficient memory.
- Coefficients can not be written to the core; the `coef_wr_addr` control is disabled.
- h_coeff_set or v_coeff_set cannot be specified; there is only one set of coefficients.
- start_hpa_y, start_hpa_c, start_vpa_y, start_vpa_c cannot be specified; they are set internally to zero.
- The control register is always set to "0x00000003," and fixed the scaler in active mode.

Fixed Mode

This option allows the core to be used without any dynamic control. When unchecking the **AXI4-Lite Register Interface** option on page one, the options on page three become active. For example, the Aperture settings that would otherwise be dynamically set using the AXI4-Lite Control interface are now set statically in page three. The Fixed Mode tab is shown in Figure 4-3.

In this mode, the coefficients are hard-coded into the netlist. You must provide the desired coefficients as an external .coe file, specifying this file in the GUI.

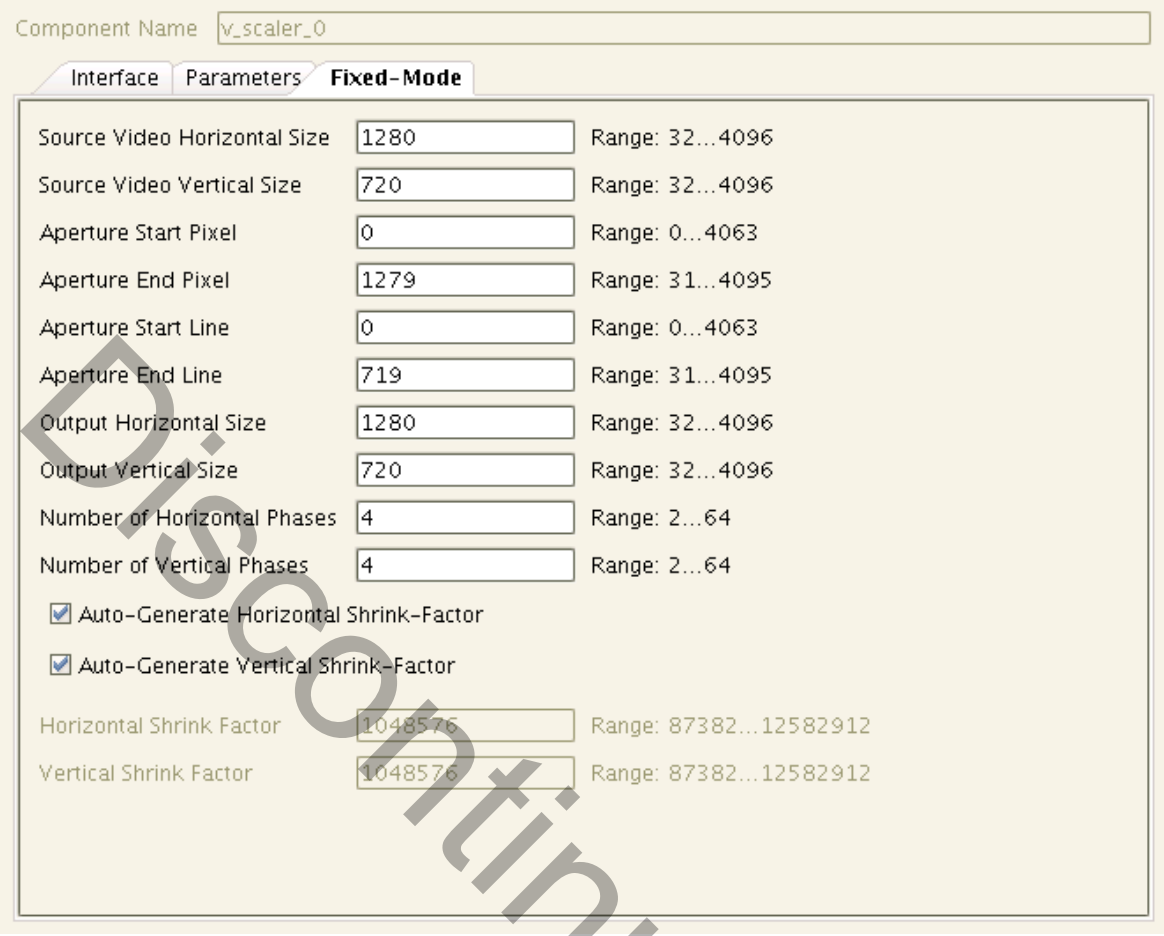

Figure 4-3: **Video Scaler Graphical User Interface for Fixed Mode**

Fixed-Mode GUI Parameters

Horizontal Scale Factor, Vertical Scale Factor: Specify, as unsigned integers, the 24-bit numbers that represent the desired fixed scale factors.

Aperture Start Pixel, Aperture End Pixel, Aperture Start Line, Aperture End Line: These parameters define the size and location of the input rectangle. They are explained in detail in Scaler Aperture in Chapter 3. The cropping feature is only available when using a Live Video data-source. In Memory mode, Aperture Start Pixel and Aperture Start Line are fixed at 0.

Output Horizontal Size, Output Vertical Size: These two parameters define the size of the output rectangle. They do not determine anything about the target video format. You must determine what do with the scaled rectangle that emerges from the scaler core.

Number of Horizontal/Vertical Phases: Non power-of-two numbers of phases are supported.

Output Generation

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

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

The only constraints required are clock frequency constraints for the video clock, `clk`, and the AXI4-Lite clock, `s_axi_aclk`. Paths between the four 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 constraints for this core.

Clock Frequencies

There are no constraints for this core.

Clock Management

There are no constraints for this core.

Clock Placement

There are no constraints for this core.

Banking

There are no constraints for this core.

Transceiver Placement

There are no constraints for this core.

I/O Standard and Placement

There are no constraints for this core.

Simulation

This section 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 8].

Synthesis and Implementation

This section contains information about synthesis and implementation in the Vivado Design Suite. For details about synthesis and implementation, see "Synthesizing IP" and "Implementing IP" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 5].

C Model Reference

This chapter introduces the bit-accurate C model for the Video Scaler core that has been developed primarily for system level modeling.

The C model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-VID-SCALER.htm.

Features

- Bit accurate with Video Scaler IP core
- Library module for the Video Scaler core function
- Available for 32 and 64-bit Windows and 32 and 64-bit Linux platforms
- Supports all features of the HW core that affect numerical results
- Designed for rapid integration into a larger system model
- Example application C code is provided to show how to use the function

The main features of the C model package are:

- **Bit-Accurate C Model**: Produces the same output data as the Video Scaler core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals.
- **Application Source Code**: Uses the model library function. This can be used as example code showing how to use the library function. However, it also serves these purposes:
  - Input .yuv file is processed by the application; 8-bit YUV422 format accepted.
  - Output .yuv file is generated by the application; 8-bit YUV422 format generated.
  - Report .txt file is generated for run time status and error messages.

Unpacking and Model Contents

To use the C model, the `v_scaler_v8_1_bitacc.zip` file must first be uncompressed. Once this is completed, the directory structure and files shown in Table 5-1 are available for use.

Table 5-1: **Directory Structure and Files of the Video Scaler v3.0Bit Accurate Model**

File Name	Contents
./doc	Documentation directory
README.txt	Release notes
pg009_v_scaler.pdf	*LogiCORE IP Video Scaler Product Guide*
v_scaler_v8_1_bitacc_cmodel.h	Model header file
v_ycrcb2rgb_v4_0_bitacc_cmodel.h	Color-space-converter model header file
yuv_utils.h	Header file declaring the YUV image / video container type and support functions including .yuv file I/O
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I/O functions.
video_utils.h	Header file declaring the generalized image / video container type, I/O and support functions
video_fio.h	Header file declaring support functions for test bench stimulus file I/O
run_bitacc_cmodel.c	Example code calling the C model
run_bitacc_cmodel.sh	Bash shell script that compiles and runs the model.
./lin64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms.
libIp_v_scaler_v8_1_bitacc_cmodel.so	Model shared object library
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Precompiled yCrCb-to RGB converter shared object file for lin64 compilation
run_bitacc_cmodel	64-bit Linux fixed configuration executable
./lin	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms.
libIp_v_scaler_v8_1_bitacc_cmodel.so	Model shared object library
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Precompiled yCrCb-to RGB converter shared object file for lin compilation
run_bitacc_cmodel	32-bit Linux fixed configuration executable

Table 5-1: **Directory Structure and Files of the Video Scaler v3.0Bit Accurate Model *(Cont'd)***

File Name	Contents
./nt64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_scaler_v8_1_bitacc_cmodel.lib	Precompiled library file for 64-bit Windows platforms compilation
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.lib	Precompiled utilities library file for 64-bit Windows platforms compilation
run_bitacc_cmodel.exe	64-bit Windows fixed configuration executable
./nt	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_v_scaler_v8_1_bitacc_cmodel.lib	Precompiled library file for 32-bit Windows platforms compilation
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.lib	Precompiled utilities library file for 32-bit Windows platforms compilation
run_bitacc_cmodel.exe	32-bit Windows fixed configuration executable
./examples	
video_in.yuv	Example YUV input file, resolution 1280Hx720V
video_in.hdr	Header file for video_in.yuv
video_in_128x128.yuv	Example YUV input file, resolution 128Hx128V
video_in_128x128.hdr	Header file for video_in_128x128.yuv
scaler_video.cfg	User-programmable configuration file containing control-register values for the core. This example gives out a scaled YUV file
scaler_coefs.cfg	User-programmable configuration file containing control-register values for the core. This example is configured to produce an example coefficient (.coe) file.

Software Requirements

The Video Scaler C models were compiled and tested with the software shown in Table 5-2.

Table 5-2: **Compilation Tools for Bit-Accurate C Models**

Platform	C Compiler
32/64-bit Linux	GCC 4.1.1
32/64-bit Windows	Microsoft Visual Studio 20058

Interface

The Video Scaler core function is a statically linked library. A higher-level software project can make function calls to this function:

```
int xilinx_ip_v_scaler_v8_1_bitacc_simulate(
struct xilinx_ip_v_scaler_v8_1_generics generics,
struct xilinx_ip_v_scaler_v8_1_inputs inputs,
struct xilinx_ip_v_scaler_v8_1_outputs* outputs).
```

Before using the model, the structures holding the inputs, generics and output of the Video Scaler instance must be defined:

```
struct xilinx_ip_v_scaler_v8_1_generics scaler_generics;
struct xilinx_ip_v_scaler_v8_1_inputs scaler_inputs;
struct xilinx_ip_v_scaler_v8_1_outputs* scaler_outputs;
```

The declarations of these structures are in the v_scaler_v8_1_bitacc_cmodel.h file.

Before making the function call, complete these steps:

1. Populate the **generics** structure:
   - num_h_taps - number of horizontal taps
   - num_v_taps - number of vertical taps
   - max_phases - maximum number of phases that are used in any scaling operation
   - max_coef_sets - maximum number of coefficient sets that are stored in the hardware (and delivered by the C-model in a .coe file)
   - Separate_YC_Coefs
     - 0: Y and C filter operations use common coefficients
     - 1: Y and C filter operations use separate coefficients
   - Separate_HV_Coefs
     - 0: H and V filter operations use common coefficients
     - 1: H and V filter operations use separate coefficients
   - UserCoefsEnabled
     - 0: Coefs generated by Model's internal automatic coef generator
     - 1: Coefficients taken from a user-defined coefs file.
   - init_coefs - four planes of pointers to coefficients (HY, HC, VY, VC). Each plane is a 2D array addressed by:
     - Tap number

- Phase Number

These coefficients are used to initialize the scaler when it is in constant (fixed) mode.

2. Populate the **inputs** structure to define the values of run time parameters:

   Note: This function processes one frame at a time.

   - video_in - Video structure described in Input and Output Video Structure, page 83.
   - aperture_start_pixel
   - aperture_end_pixel
   - aperture_start_line
   - aperture_end_line
   - hsf
   - vsf
   - num_h_phases
   - num_v_phases
   - SingleFrameCoefs - coefficients that are used to scale the next frame.

3. Populate the **outputs** structure.

   - video_out - Video structure described in Input and Output Video Structure, page 83.

   The video_in variable is not initialized because the initialization depends on the actual test image to be simulated. The next section describes the initialization of the video_in structure.

   Results are provided in the outputs structure, which contains the output video data in the form of type video_struct. After the outputs have been evaluated or saved, dynamically allocated memory for input and output video structures must be released. See Delete the Video Structure, page 86 for more information. Successful execution of all provided functions returns a value of 0. Otherwise, a non-zero error code indicates that problems were encountered during function calls.

Input and Output Video Structure

Input images or video streams can be provided to the Video Scaler reference model using the video_struct structure, defined in `video_utils.h`:

```
struct video_struct{
  int       frames, rows, cols, bits_per_component, mode;
  uint16*** data[5]; };
```

Table 5-3: **Member Variables of the Video Structure**

Member Variable	Designation
frames	Number of video/image frames in the data structure
rows	Number of rows per frame(1)
cols	Number of columns per frame(1)
bits_per_component	Number of bits per color channel / component(2)
mode	Contains information about the designation of data planes(2)
data	Set of 5 pointers to 3 dimensional arrays containing data for image planes(4)

1. Pertaining to the image plane with the most rows and columns, such as the luminance channel for yuv data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y,u and v can have different dimensions.
2. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
3. Named constants to be assigned to mode are listed in Table 5-4.
4. Data is in 16 bit unsigned integer format accessed as data[plane][frame][row][col].

Table 5-4: **Named Constants for Video Modes with Corresponding Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome, luminance only
FORMAT_RGB(1)	3	RGB image/video data
FORMAT_C444(1)	3	444 YUV, or YCrCb image/video data
FORMAT_C422(1)	3	422 format YUV video, (u,v chrominance channels horizontally sub-sampled)
FORMAT_C420(1)	3	420 format YUV video, (u,v sub-sampled both horizontally and vertically)
FORMAT_MONO_M	3	Monochrome (luminance) video with motion
FORMAT_RGBA	4	RGB image / video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with motion
FORMAT_C422_M	5	422 YUV video with motion
FORMAT_C444_M	5	444 YUV video with motion
FORMAT_RGBM	5	RGB video with motion

1. Supported by the Video Scaler core.

Initializing the Video Scaler Input Video Structure

The Video Scaler core assigns data to a video structure typically by reading from a .yuv video file. This file is described in the Model IO Files chapter below. The `yuv_util.h` and `video_util.h` header files packaged with the bit-accurate C models contain functions to

facilitate file I/O. The run_bitacc_cmodel example code uses these functions to read from the delivered YUV file.

YUV Image Files

The header yuv_utils.h declares functions which help access files in standard YUV format. It operates on images with three planes (Y, U and V). The following functions operate on arguments of type yuv8_video_struct, which is defined in `yuv_utils.h`:

```
int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv8(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

Exchanging data between yuv8_video_struct and general video_struct type frames/videos is facilitated by functions:

```
int copy_yuv8_to_video(struct yuv8_video_struct* yuv8_in,
struct video_struct* video_out );
int copy_video_to_yuv8( struct video_struct* video_in,
struct yuv8_video_struct* yuv8_out );
```

All image/video manipulation utility functions expect both input and output structures to be initialized (for example, pointing to a structure which has been allocated in memory) either as static or dynamic variables. Moreover, the input structure has to have the dynamically allocated container (data[] or y[],u[],v[]) structures already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and throw an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.

Working with video_struct Containers

Header file `video_utils.h` define functions to simplify access to video data in video_struct.

```
int video_planes_per_mode(int mode);
int video_rows_per_plane(struct video_struct* video, int plane);
int video_cols_per_plane(struct video_struct* video, int plane);
```

Function video_planes_per_mode returns the number of component planes defined by the mode variable, as described in Table 5-4, page 84. Functions video_rows_per_plane and video_cols_per_plane return the number of rows and columns in a given plane of the selected video structure. The example below demonstrates using these functions in conjunction to process all pixels within a video stream stored in variable in_video with the following construct:

```
for (int frame = 0; frame < in_video->frames; frame++) {
  for (int plane = 0; plane < video_planes_per_mode(in_video->mode); plane++) {
    for (int row = 0; row < rows_per_plane(in_video,plane); row++) {
```

```
        for (int col = 0; col < cols_per_plane(in_video,plane); col++) {
     // User defined pixel operations on
// in_video->data[plane][frame][row][col]
        }
      }
    }
}
```

Delete the Video Structure

Large arrays such as the video_in element in the video structure must be deleted to free up memory. The following example function is defined as part of the video_utils package.

```
void free_video_buff(struct video_struct* video )
{
   int plane, frame, row;
   if (video->data[0] != NULL) {
      for (plane = 0; plane <video_planes_per_mode(video->mode); plane++)
      {
         for (frame = 0; frame < video->frames; frame++) {
            for (row = 0; row<video_rows_per_plane(video,plane); row++) {
               free(video->data[plane][frame][row]);
            }
            free(video->data[plane][frame]);
         }
         free(video->data[plane]);
      }
   }
}
```

This function can be called as follows:

```
free_video_buff ((struct video_struct*) &manr_outputs.video_out);
```

C-Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided. This demonstrates the steps required to run the model.

After following the compilation instructions, run the example executable.

The executable takes the path/name of the input file and the path/name of the output file as parameters. If invoked with insufficient parameters, the following help message is printed:

```
Usage: run_bitacc_cmodel -c <cfg file> -t <test name> -y <YUV File> -h <hsize> -v
<vsize>\n");
```

```
      cfg file    :  Path/name of the scaler config (.cfg) file.
      test name   :  Name of test specified in .cfg file.
                     Use 'all' to generate all tests specified in .cfg file
      YUV file    :  Path/name of Input YUV File.
      hsize       :  Source YUV file Horizontal size.
      vsize       :  Source YUV file Vertical size.
     Example: run_bitacc_cmodel -c ./scaler.cfg -t test001 -y./video_in.yuv -h 1280
-v 720
```

Config File

During successful execution, the specified config file is parsed by the run_bitacc_cmodel example. This file specifies:

- Output YUV Filename
- Number of frames to be scaled
- Number of scaler taps and phases
- Input and output rectangle sizes
- Various other options. More details about the options can be found in Chapter 3, Designing with the Core.

An example of a config file is given in the delivered zip file.

Compiling the Video Scaler C-Model

Linux (32 or 64-bit)

Place the header file and C source file in a single directory. Then in that directory, compile using the GNU C Compiler:

```
gcc -m64 -x c++ ./run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_scaler_v8_1_bitacc_cmodel -Wl,-rpath,.

gcc -m32 -x c++ ./run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_scaler_v8_1_bitacc_cmodel -Wl,-rpath,.
```

Windows (32 or 64-bit)

Compile the precompiled library `v_scaler_v8_1_bitacc_cmodel.dll` and top-level demonstration code `run_bitacc_cmodel.c` with an ANSI C compliant compiler under Windows. This section includes an example using Microsoft Visual Studio.

In Visual Studio, create a new, empty Win32 Console Application project. As existing items, add:

- `libIp_v_scaler_v8_1_bitacc_cmodel.dll` to the "Resource Files" folder of the project
- `libIp_v_ycrcb2rgb_bitacc_model.dll` to the "Resource Files" folder of the project
- `run_bitacc_cmodel.c` to the "Source Files" folder of the project
- `v_scaler_v8_1_bitacc_cmodel.h` to "Header Files" folder of the project
- `v_ycrcb2rgb_v4_0_bitacc_cmodel.h` to the "Header Files" folder of the project
- `yuv_utils.h` to the "Header Files" folder of the project
- `rgb_utils.h` to the "Header Files" folder of the project
- `video_utils.h` to the "Header Files" folder of the project
- `xscaler_coefs.h` to the "Header Files" folder of the project

Once the project has been created and populated, it needs to be compiled and linked to create a win32 executable. To perform the build step, choose **Build Solution** from the Build menu. An executable matching the project name is created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" has been selected in the "Configuration Manager" under the Build menu.

Model IO Files

Input file

- `<input_filename>.yuv`
  - Standard 8-bit yuv file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player.
  - No header.

Output Files

- `<output_filename>.yuv`
  - Standard 8-bit 4:2:2 yuv file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player.

Detailed Example Design

This chapter provides an example system that includes the Video Scaler core. Important system-level aspects when designing with the video scaler are highlighted, including:

- Video scaler usage with the Xilinx AXI-VDMA block
- Typical usage of video scaler in conjunction with other cores

Example System General Configuration

The system input and output is expected to be no larger than 720P (1280Hx720V), with a maximum pixel frequency of 74.25 MHz, with equivalent clocks.

- MicroBlaze processor controls scale factors according to user input
- The system can upscale or downscale
- When down scaling, the full input image is scaled down and placed in the center of a black 720P background and displayed
- When upscaling, the center of the 720P input image is cropped from memory and upscaled to 720P, and displayed as a full 720P image on the output
- Operational clock frequencies are derived from the input clock

Figure 6-1 shows a typical example of the video scaler in memory mode incorporated into a larger system. Here are the essential details:

- The Xilinx AXI Video Direct Memory Access (AXI-VDMA) blocks allow fast DMA access to video frames in memory.
- The On-Screen Display (OSD) block aligns the data read from memory with the timing signals and presents it as a standard-format video data stream. It also alpha-blends multiple layers of information (for example, text or other video data). See PG010, *LogiCORE IP On-Screen Display Product Guide* for more information.

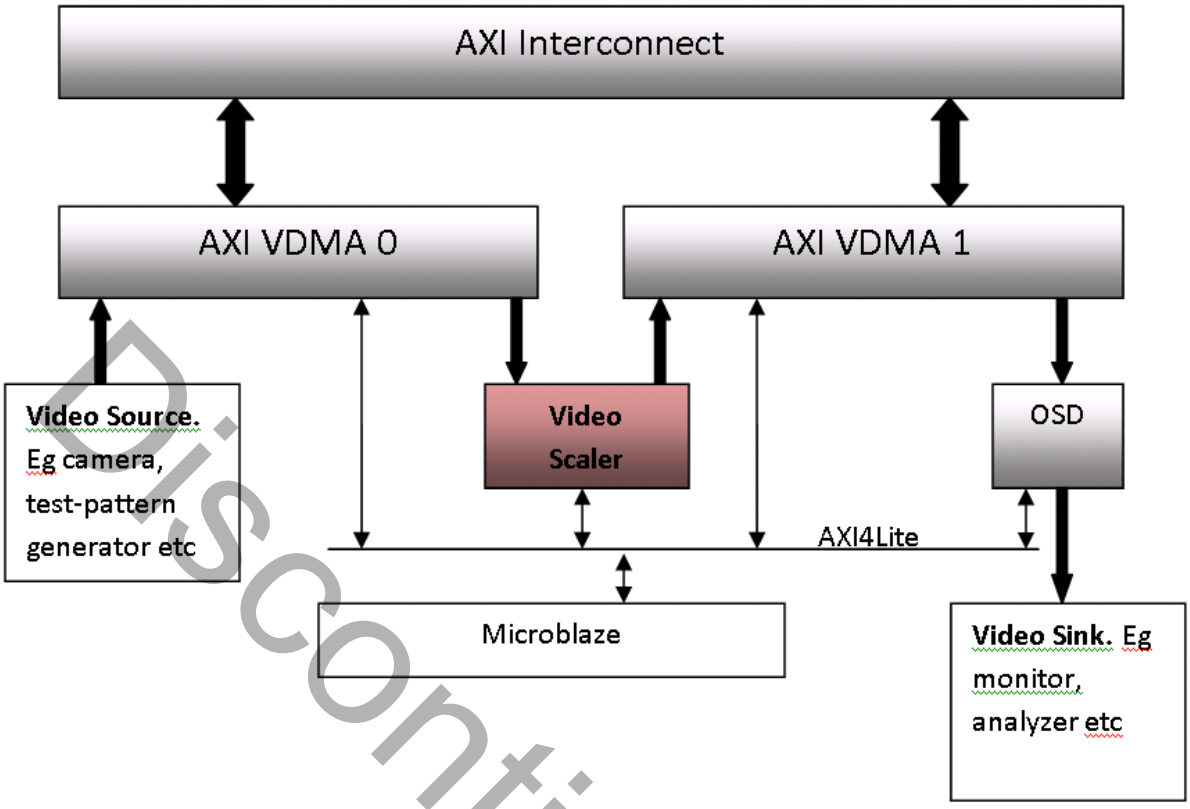

Figure 6-1: **Simplified System Diagram**

Control Buses

In this example, MicroBlaze is configured to use the AXI4-Lite bus. The AXI-VDMAs, Video Scaler, and OSD use AXI4-Lite.

AXI_VDMA0 Configuration

AXI_VDMA0 is used bi-directionally. The input side takes data from the source domain and writes frames of data into DDR memory. The read side reads data (on a separate clock domain and separate video timing domain) and feeds it to the scaler.

The system operates using a Genlock mechanism. A rotational 5-frame buffer is defined in the external memory. Using the Genlock bus, AXI_VDMA0 tells AXI_VDMA1 which of the five frame locations is being written to avoid R/W collisions.

AXI_VDMA1 Configuration

AXI_VDMA1 is used bi-directionally. The input side takes data from the scaler output and writes scaled frames of data into DDR memory. The read side reads data and feeds it to the OSD.

AXI_VDMA1 is a Genlock slave to AXI_VDMA0.

Video Scaler Configuration

To be able to support smooth shrink/zoom functions and downscaling, the scaler should be configured with a large number of taps and phases. To support 720P/60 YC operation, the core needs to be configured with a single YC engine, and run at a `core_clk` rate of 148.5 MHz (2x input pixel clock).

Cropping from Memory

Controlling the AXI_VDMA dynamically (for example, from a MicroBlaze processor or other processor) allows you to request any rectangle from any where in the image in memory, and change the position and dimensions of this rectangle on a frame-by frame basis.

OSD Configuration

The OSD is configured for two layers. The first layer is video data read from AXI_VDMA1. The second layer is text overlay.

Use Cases

Using systems such as the example system given above as the scaling section of a video system, it is possible to scale in many different ways. Examples of such use cases are shown in Figure 6-2 through Figure 6-6. These examples show particular variations of the following scaler parameters:

- `aperture_start_line`
- `aperture_end_line`
- `aperture_start_pixel`

- `aperture_end_pixel`
- `output_h_size`
- `output_v_size`
- `hsf`
- `vsf`

These examples show the use of the aperture_start_pixel, aperture_end_pixel, aperture_start_line, aperture_end_line parameters.

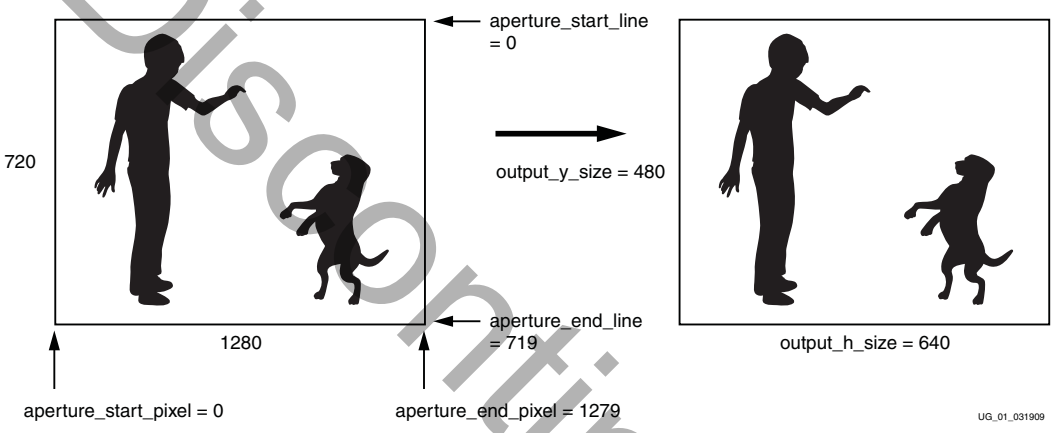

Figure 6-2: **Format Down-scaling. Example 720p to 640x480, HSF = 2^{20} x 1280/640; VSF = 2^{20} x 720/480**

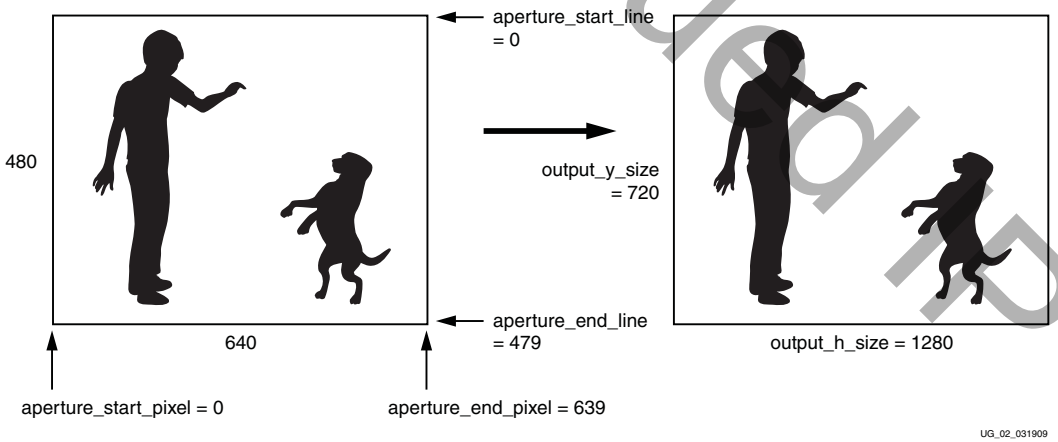

Figure 6-3: **Format Up-scaling. Example 640x480 to 720p, HSF = 2^{20} x 640/1280; 2^{20} x VSF = 480/720**

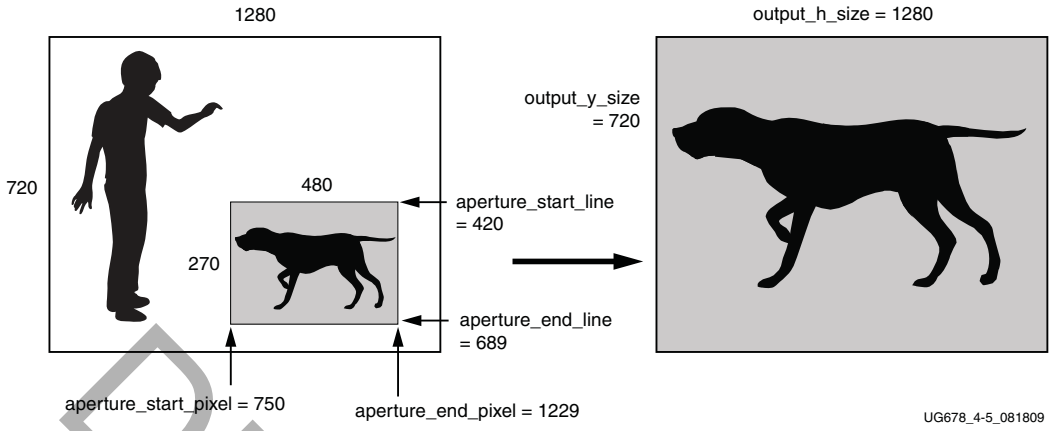

Figure 6-4: **Zoom (Up-scaling), HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

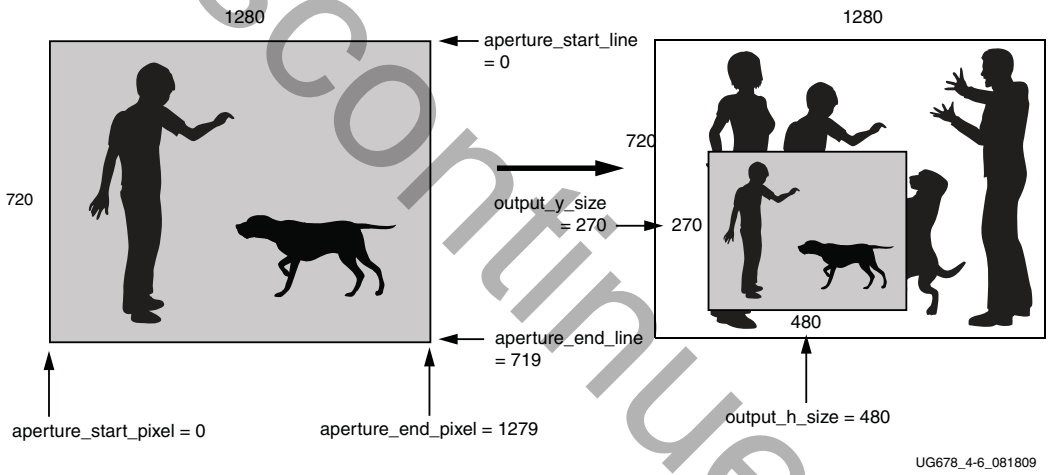

Figure 6-5: **Shrink (Down-scaling). Example for Picture-in-Picture (PinP), HSF = 2^{20} x 1280/480; VSF = 2^{20} x 720/270**

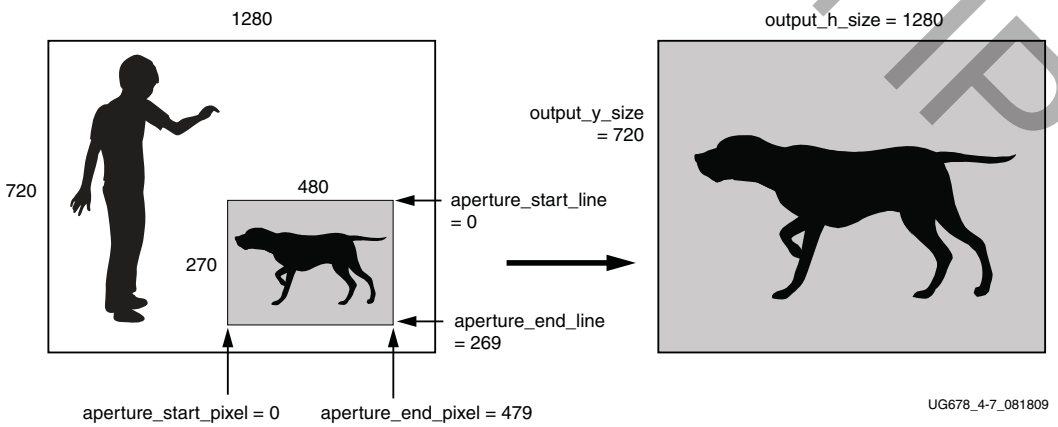

Figure 6-6: **Zoom (Up-scaling) reading from External Memory, HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

Test Bench

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

Demonstration Test Bench

A demonstration test bench is provided with the core which enables you to observe core behavior in a typical scenario. This test bench is generated together with the core in Vivado Design Suite. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

Directory and File Contents

The following files are expected to be generated in the in the demonstration test bench output directory:

- `axi4lite_mst.v`
- `axi4s_video_mst.v`
- `axi4s_video_slv.v`
- `ce_generator.v`
- `tb_<IP_instance_name>.v`

Test Bench Structure

The top-level entity is **`tb_<IP_instance_name>`**.

It instantiates the following modules:

- `DUT`

  The <IP> core instance under test.

- `axi4lite_mst`

The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- `axi4s_video_mst`

  The AXI4-Stream master module, which generates ramp data and initiates AXI4-Stream transactions to provide video stimuli for the core and can also be used to open stimuli files generated from the reference C models and convert them into corresponding AXI4-Stream transactions.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the stimuli file name and directory path
  `define STIMULI_FILE_NAME<path><filename>.`

  b. Comment-out/remove the following line:
  `MST.is_ramp_gen(`C_ACTIVE_ROWS, `C_ACTIVE_COLS, 2);`
  and replace with the following line:
  `MST.use_file(`STIMULI_FILE_NAME);`

  For information on how to generate stimuli files, see *Chapter 4, C Model Reference*.

- `axi4s_video_slv`

  The AXI4-Stream slave module, which acts as a passive slave to provide handshake signals for the AXI4-Stream transactions from the core output, can be used to open the data files generated from the reference C model and verify the output from the core.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the golden file name and directory path
  `define GOLDEN_FILE_NAME "<path><filename>".`

  b. Comment out the following line:
  `SLV.is_passive;`
  and replace with the following line:
  `SLV.use_file(`GOLDEN_FILE_NAME);`

  For information on how to generate golden files, see *Chapter 4, C Model Reference*.

- `ce_gen`

  Programmable Clock Enable (`ACLKEN`) generator.

Verification, Compliance, and Interoperability

This appendix contains details about verification and testing used for the Video Scaler core.

Simulation

A parameterizable test bench was used to test the Video Scaler core. Testing included the following:

- Register accessing
- Processing of multiple frames of data
- Various frame sizes
- Various scale-factors - up and down-scaling in both dimensions.
- Various coefficient sets
- Various filter configurations (number of taps, phases, engines)
- Both Memory mode and Live Mode

Hardware Testing

The Video Scaler core has been tested in a variety of hardware platforms at Xilinx to represent a variety of parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Interface and various other peripherals, as described in Chapter 6, Detailed Example Design.
- The software for the test system included input frames embedded in the source-code. The checksums of the scaled images were also pre-calculated and included in the SW. The frames, resident in external memory, are read by the AXI_VDMA, scaled by the scaler and the result is passed back to memory. SW then accesses the scaled frame in

memory and calculates the checksum of the scaled frame. This matches the pre-calculated checksum.

- Various configurations were implemented in this way. The C model was used to create the expected Checksums and generate the stimulus C-Code frame-data that is compiled into the software.
- Pass/Fail status is reported by the software.

In addition, the Video Scaler has been more regularly tested using an automated validation flow. Primarily, this instantiates the core to read registers back, validating the core's Version register and proving that it has been implemented in the design. This has been run regularly to validate new core versions during development, and also to guard against EDK tools regressions.

Migrating

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 4].

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.

Parameter Changes

There are no parameter changes.

Port Changes

There are no port changes.

Other Changes

From v6.00.a to v7.00.a of the Video Scaler core, the following significant changes took place:

1. AXI4-Stream Interface usage was updated to be compliant with the AXI-4 Stream Video Protocol as documented in *Video IP: AXI Feature Adoption* section of the UG761 AXI Reference Guide [Ref 3].

a. It no longer supports the AXI4-Stream tKeep usage.

b. It now uses the AXI4-Stream tUser pin as a Start-of-Frame indicator.

2. The GPP control option has been removed. The core can now only be controlled dynamically by using the AXI4-Lite interface.

3. The XSVI interface option has been removed. In v5.0 of the core, "Live-Mode" operation exclusively used XSVI. "Memory Mode" exclusively used the native AXI-VDMA interface format. These two modes of operation both now use the AXI4-Stream interface protocol.

4. Core Feature Changes

   a. The v7.00.a core includes a feature which allows the Fixed-Mode scale-factors to be optionally calculated by the GUI according to the user-specified input and output sizes.

   b. External sync signals are no longer required by this core. Previously, vsync, vblank, hblank, active_video were needed by the core. These signals are not embedded in the AXI4-Stream interfaces.

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 Scaler, the Xilinx Support web page (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 Scaler. 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 Scaler Core

AR 54540

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, navigate to the Xilinx Support web page.

1. 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 Scaler design issues. It is important to know which tools are useful for debugging various situations.

Example Design

The Video Scaler is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found in *Chapter 6, Example Design for the Vivado™ Design Suite*.

Core Wizard

The Image Edge Enhancement core is equipped with optional debugging features which aim to accelerate system bring-up, optimize memory and data-path architecture, and reduce time to market. The optional debug features can be turned on and off using the **Include**

Debug Features checkbox on the GUI when an AXI4-Lite interface is present. Turning off debug features reduces the core footprint.

Vivado Design Suite Debug Feature

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

The Vivado lab tools 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)
- See *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Reference Boards

Various Xilinx development boards support Video Scaler. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series evaluation boards
  - KC705
  - KC724

C-Model Reference

Please see *C Model Reference in Chapter 5* in this guide for tips and instructions for using the provided C-Model files to debug your design.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- Vivado Synthesis
- Vivado Implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

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 ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool 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-Lite Interfaces

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

Table C-1: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Are the S_AXI_ACLK and ACLK pins connected? In EDK, verify that the S_AXI_ACLK and ACLK pin connections in the system.mhs file. The VERSION_REGISTER readout issue can be indicative of the core not receiving the AXI4-Lite interface.
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core enabled?
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core in reset? S_AXI_ARESETn and ARESETn should be connected to vcc for the core not to be in reset. In EDK, verify that the S_AXI_ARESETn and ARESETn signals are connected in the **system.mhs** to either net_vcc or to a designated reset signal.
Readback value for the VERSION_REGISTER is different from expected default values	The core and/or the driver in a legacy EDK/SDK project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared.

Assuming the AXI4-Lite interface works, the second step is to bring up the AXI4-Stream interfaces.

AXI4-Stream Interfaces

Table C-2 describes how to troubleshoot the AXI4-Stream interface.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 1 of the ERROR register reads back set.	Bit 1 of the ERROR register, EOL_LATE, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.

Table C-2: **Troubleshooting AXI4-Stream Interface *(Cont'd)***

Symptom	Solution
Bit 2 or Bit 3 of the ERROR register reads back set.	Bit 2 of the ERROR register, SOF_EARLY, and bit 3 of the ERROR register SOF_LATE indicate the number of pixels received between the latest and the preceding Start-Of-Frame (SOF) differ from the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the core keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically. Is m_axis_video_tready low? If so, the core cannot send data downstream, and the internal FIFOs are full.
m_axis_video_tvalid stuck low, the downstream core is not receiving data	• No data is generated during the first two lines of processing. • If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (s_axis_video_tlast) End-of-line signal. Check the ERROR register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the ERROR register.
Generated EOL signal (m_axis_video_tlast) signal misplaced.	Check the ERROR register.
Data samples lost between Upstream core and the core. Inconsistent EOL and/or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and the core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the core. Inconsistent EOL and/or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and the core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the ACLK pin lower than the reported Fmax reached?

If the AXI4-Stream communication is healthy, but the data seems corrupted, the next step is to find the correct configuration for this core.

Other Interfaces

Table C-3 describes how to troubleshoot third-party interfaces.

Table C-3: **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 according to Data Interface in Chapter 2. If misaligned: In HDL, break up the TDATA vector to constituent components and manually connect the slave and master interface sides. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.
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 described in Data Interface in Chapter 2 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. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.

General Debugging Tips

Some general debugging tips are as follows:

- Verify that the Version register can be read properly. See Table 2-9, page 22 for register definitions.
- Verify the other resisters can be read properly. Do they match your expectations?
- Verify that bits 0 and 1 of the core's Control register are both set to “1”. Bit 0 is the Core Enable bit. Bit 1 is the Register Update Enable bit.
- Verify that the output interface is not holding off permanently. The m_axis_video_tready signal must be High for any data to come out of the core.
- Verify that the Video Scaler core is toggling its m_axis_video_tvalid output. If this is occurring, check the data output.
- If m_axis_video_tvalid is toggling, but the data is zero, this usually means that the coefficients are all 0. Perhaps the coefficients did not get loaded. Alternatively, the input data is all zero.
- If the bottom lines of the image are static or corrupted, this can mean that the scaler is configured incorrectly, likely expecting more input lines than you have provided it in the frame. In this case, revisit the calculations of clock frequency, scale-factor aperture and output sizes.

- If using AXI-VDMAs and external memory, check that the addresses for the scaler input/output frame buffers are correct.
- If using AXI-VDMAs and external memory, use XMD (or other utility) to check the actual data in memory before and/or after scaling.
- Attach a static pattern-generator to introduce known data into the video stream.

RECOMMENDED: *It is recommended to prototype the system with the AXI4-Stream interface enabled, so status and error detection, reset, and dynamic size programming can be used during debugging.*

The following steps are recommended to bring-up/debug the core in a video/imaging system:

1. Bring up the AXI4-Lite interface
2. Bring up the AXI4-Stream interfaces
3. Finding the right Noise-Reduction value

Once the core is working as expected, you can consider 'hardening' the configuration by replacing the core with an instance where GUI default values are set to the established dynamic values that work correctly for the application, but the AXI4-Lite interface is disabled. This configuration reduces the core slice footprint, and reduces SW burden. It could potentially lead to the removal of any SW element if a fixed-mode system is ultimately desired.

Additional Resources and Legal Notices

Xilinx Resources

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

http:/

http://Xilinx Support web page/documentation/sw_manuals/glossary.pdf.

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 product guide:

1. AXI - Video and Imaging Documentation
2. For more information on Lanczos resampling, refer to this Wikipedia page: http://en.wikipedia.org/wiki/Lanczos_resampling
3. *Vivado AXI Reference Guide* (UG1037)
4. *ISE to Vivado Design Suite Migration Guide* (UG911)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *Vivado Design Suite User Guide: Getting Started* (UG910)
8. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
9. *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	8.1	Added UltraScale™+ support.
10/01/2014	8.1	Removed Application Software Development appendix.
04/02/2014	8.1	Updated for Lanczos Coefficients GUI option.
12/18/2013	8.1	Added UltraScale Architecture support.
10/02/2013	8.1	Updated Constraints, Migration, and Test Bench chapters.
06/19/2013	8.1	Synchronized document version with core version. Updated calculations for clock frequencies and clocking examples. Updated Constraints.
03/20/2013	3.2	Updated for core version. Updated Debugging Appendix. Removed ISE chapters.
10/16/2012	3.1	Updated for core version.
07/25/2012	3.0	Updated for core version. Added Vivado information.
04/24/2012	2.0	Updated for core version. Added Zynq-7000 devices, added AXI4-Stream interfaces, deprecated GPP interface.
10/19/2011	1.0	Initial Xilinx release of Product Guide, replacing DS840 and UG805.

Please Read: Important Legal Notices