LogiCORE IP Video Scaler v6.00.a

Product Guide

PG009 April 24, 2012

XILINX®

Table of Contents

Chapter 5: Constraining the Core

Chapter 6: Detailed Example Design

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating

Appendix C: Debugging

Appendix D: Application Software Development

Appendix E: C Model Reference

Appendix F: Additional Resources

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 filters the incoming video data stream using a separable polyphase H/V filter arrangement in order 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
  - Supports 4kx2k @ 24 Hz in supported high performance devices

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq™7000, Artix™-7, Virtex®-7, Kintex®-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Lite, AXI4-Stream(2)
Resources	See Table 2-8 through Table 2-14.
Provided with Core	
Documentation	Product Guide
Design Files	NGC netlist, Encrypted HDL
Example Design	Not Provided
Test Bench	Verilog (3)
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural, C Model(3)
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation(4)	Mentor Graphics ModelSim, Xilinx® ISim
Synthesis Tools	Xilinx Synthesis Technology (XST)
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG761 AXI Reference Guide.
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. For the supported versions of the tools, see the ISE Design Suite 14: 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.

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

$$Pix_{out}[x, y] = \sum_{HTaps\_1}^{i=0} \sum_{VTaps\_1}^{j=0} Pix_{in}[x - (HTaps/2) + i, y - (VTaps/2) + j] \times Coef[i, j]$$

Figure 1-1: **Generic Image Filtering Equation**

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 weightings applied to the set of input samples chosen to contribute to one output pixel, according to the scaling ratio.

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 may be used to provide a wider filter design toolset. More information about coefficients is located in Coefficients in Chapter 4.

A direct implementation of this equation suggests that a filter with VTaps x HTaps multiply operations per output are required. However, the Xilinx Video Scaler uses a separable filter, 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 summed intermediate result of the first stage is 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 1-1).

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

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 1-2).

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

Feature Summary

The Video Scaler core supports input and output image sizes up to 4096x4096, in YC4:2:0, YC4:2:2, YC4:4:4 and RGB chroma formats.

At compile time, using the configuration GUIs provided in the CORE Generator and EDK tools, you may select the number of taps (2-12) and phases (2-16, 32 or 64) used by the filter. While the size of the scaler implementation is greatly influenced by the number of taps and number of phases in each filter engine, for many cases the output image quality improves when using a large number of taps and phases.

The number of engines used to perform the scaling operations is also customizable. A greater number of engines allows the scaler throughput to increase proportionately. The size of the scaler implementation is also heavily influenced by the number of engines implemented.

The video data width (8, 10 and 12 bits) is also customizable. This also has an effect on the final implementation size.

Video is passed into the Video Scaler using the AXI4-Stream Video protocol. This format includes standard back-pressure signaling found in AXI4-Stream. This interconnect format is used for connecting to other IP blocks that support AX4-Stream.

In many cases, the Video Scaler core is set up as a preset standalone module with a fixed scale-factor, fixed coefficients, fixed filter size and other fixed variables. For this standalone module, de-select the AXI4-Lite option in the Core Generator GUI. Scaling parameters can all be fixed in the CORE Generator tool GUI.

In other cases, dynamic user control is required for changing various settings on a frame-by-frame basis. For these cases, the AXI4-Lite interface should be selected during generation.

Applications

- Broadcast Displays, Cameras, Switchers, and Video Servers
- LED Wall
- Multi-Panel Displays
- Digital Cinema
- 4Kx2K Projectors
- Post-processing block for image scaling
- Medical Endoscope
- Video Surveillance
- Consumer Displays
- Video Conferencing
- Machine Vision

Licensing

The Video Scaler core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

After installing the required Xilinx ISE software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Video Scaler core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Video Scaler core using the

example design and demonstration test bench provided with the core. In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The

core can be tested in the target device for a limited time before timing out (resetting to default values and the output video becoming black), at which time it can be reactivated by reconfiguring the device.

The timeout period for this core is set to approximately 8 hours for a 74.25 MHz clock. Using a faster or slower clock changes the timeout period proportionally. For example, using a 150 MHz clock results in a timeout period of approximately 4 hours.

To obtain a Full System Hardware Evaluation license, do the following:

1. Navigate to the product page for this core.
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full

The Full license key is available when you purchase the core and provides full access to all core functionality both in simulation and in hardware, including:

- Functional simulation support
- Full implementation support including place-and route-and bitstream generation
- Full functionality in the programmed device with no time outs

To obtain a Full license key, you must purchase a license for the core. Click on the "Order" link on the Xilinx.com IP core product page for information on purchasing a license for this core. After doing so, click the "How do I generate a license key to activate this core?" link on the Xilinx.com IP core product page for further instructions.

Simulation License

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator™ software.

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document.

Product Specification

Standards Compliance

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 UG761 AXI Reference Guide for additional information.

Performance

The following sections detail the performance characteristics of the Video Scaler core.

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 data is applies equally for all three of the clocks: `video_in_aclk`, `core_clk` and `video_out_aclk`.

The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors. To assist in making system-level and board-level decisions, Table 2-1 through Table 2-6 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: **Spartan-6 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	
								-2	-3
Yes	1920x1080	1920x1080	10	11x11	64	3(444)	1	175	195.5

Table 2-2: **Virtex-7 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

Table 2-3: **Virtex-6 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
Yes	1920x1080	1920x1080	10	11x11	64	3(444)	1	270	324

Table 2-4: **Kintex-7 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	332.5	370

Table 2-5: **Artix-7 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	181	222.5	247

Table 2-6: **Zynq-7000 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	280	320	394.5

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.

The latency may be approximated as:

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

The calculation does not take back-pressure exerted on the scaler into account.

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.

In all cases, it must be emphasized that the core is a spatial Video Scaler only. For every frame it consumes, it produces one scaled output frame (no more, no less).

When running from memory using the AXI4-Stream interface, there is greater flexibility to feed data into the scaler as needed. The throughput is dependent on the worst-case of the input and output image sizes:

- When up-scaling (output image larger than input image), the throughput is a function of output image size and the clock-frequencies used.
- When down-scaling (input image larger than output image), the throughput is a function of input image size and the clock-frequencies used.

In all cases, the number of engines affects overall throughput.

It is very important to ensure that the clock rate available supports worst-case conversions. This section includes detailed information and examples for worst-case scenarios.

Every user of the Xilinx Video Scaler should have a worst-case scenario in mind. The factors that may contribute to this scenario include:

- 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`, `video_in_aclk`, `video_out_aclk`: depends upon the selected device)

These factors may contribute to decisions made for configuring the scaler and its supporting system. For example, you may 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 an increased resource usage. He may also elect to change speed-grade or even device family dependent upon his findings.

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

To determine whether or not the scaler will meet 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 will include the Subject Image, and is of dimensions (**ActWidth** x **ActHeight**).
FPix	The input sample rate.
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 the system clock.
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.
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**).

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 will 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 may control the input feed using the AXI4-Stream back-pressure and handshaking by implementing an input frame buffer.

Live Video, page 16 and Memory Mode, page 23 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:

- `video_in_aclk`: Input Clock

  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 may safely be driven at any frequency above approximately 61 MHz. Once the memory mode scaler reaches the end of a frame, it will stop processing until the start of the next frame. So, faster clock rates are safe.

- `video_out_aclk`: Output Clock

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

  In both cases, bear in mind 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 may 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 video_out_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%. Bear in mind 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

  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 may 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 may be required in order 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 may 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 cycles per component per line.
- **CyclesPerOutputLine** – 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), and can be summarized as:

  For 4:4:4:

  ```
  CyclesPerOutputLine = Max(output_h_size,SubjWidth) +
  ProcessingOverheadPerComponent
  ```

  For 4:2:2 dual-engine:

CyclesPerOutputLine = Max(output_h_size,SubjWidth) + 2*ProcessingOverheadPerComponent

For 4:2:2 single-engine:

CyclesPerOutputLine = 2*Max(output_h_size,SubjWidth) + 3*ProcessingOverheadPerComponent

For 4:2:0:

CyclesPerOutputLine = 2*Max(output_h_size,SubjWidth) + 3*ProcessingOverheadPerComponent

For more details on the above estimations, continue reading. Otherwise, skip to the MaxVHoldsPerInputAperture bullet below.

The general calculation is:

```
CyclesPerOutputLine=(CompsPerEngine*Max(output_h_size,SubjWidth))+O
verHeadMult*ProcessingOverheadPerComponent
```

The CompsPerEngine and OverHeadMult values can be extracted from Table 2-7.

Table 2-7: **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 BRAM 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.

```
CyclesRequiredPerOutputLine=Max(output_h_size,SubjWidth)+Process
ingOverheadPerComponent
```

We modify this to include the chroma components. YC case is shown in this example.

CyclesRequiredPerOutputLine=2*Max(output_h_size,SubjWidth)+3*ProcessingOverhea dPerComponent

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

```
MaxVHoldsPerInputAperture = CEIL(Vertical scaling ratio)
```

where

```
vertical scaling ratio = output_v_size/input_v_size
```

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

- **MaxClksTakenPerVAperture** – This is the number of cycles it will take to generate **MaxVHoldsPerInputAperture** lines.

```
MaxClksTakenPerVAperture = 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' = FLineIn x MaxClksTakenPerVAperture
```

Also useful is the reciprocal relationship that defines the number of 'core_clk' cycles available before the next line is written into the input line buffer, for a predefined 'core_clk' frequency:

```
ClksAvailablePerLine = F'core_clk'/FLineIn
```

Within this number of cycles, all output lines that require the use of the current vertical filter aperture must be completely generated. If MaxClksTakenPerVAperture < ClksAvailablePerLine, then the desired conversion is possible using the current clock frequency, without the use of an input frame buffer.

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
MaxClksTakenPerVAperture = 3990 * 1 = 3990
MinF'core_clk' = 33750*3990 = 134.66 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 (video_out_aclk) = (1036800 / 0.016) * 1.1 = 71.28 MHz
```

Shrink-factor inputs:

$$hsf=2^{20} \times (1/1.0) = 0x100000$$
$$vsf=2^{20} \times (1/1.0) = 0x100000$$

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 2-7.

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
MaxClksTakenPerVAperture = 1750 * 2= 3500
MinF'core_clk' = 31500*3500 = 110.25 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 (video_out_aclk)= (480000 / 0.0152) * 1.1 = 34.65 MHz
```

Shrink-factor inputs:

$$hsf=2^{20} \times (1/1.25) = 0x0CCCCC$$
$$vsf=2^{20} \times (1/1.25) = 0x0CCCCC$$

Note: This case is easily possible with no input buffer in Spartan-6.

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
MaxClksTakenPerVAperture = 3990 * 3 = 11970
MinF'core_clk' = 31500*11970 = 377.06 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=480/31500 = 0.0152
NumOutputPixels=1920x1080 = 2073600
MinFOut (video_out_aclk) =(2073600 / 0.0152) * 1.1 = 149.69 MHz
```

Shrink-factor inputs:

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

Note: Without an input frame buffer, this conversion will only work 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
MaxClksTakenPerVAperture = 2020 * 3 = 6060
MinF'core_clk' = 30000*6060 = 190.89 MHz
video_in_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=480/31500 = 0.0152
NumOutputPixels=1920x1080 = 2073600
MinFOut (video_out_aclk) = (2073600 / 0.0152) * 1.1 = 149.69 MHz
```

Shrink-factor inputs:

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

Note: For a dual-engine implementation, without an input frame buffer, this conversion will work 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
MaxClksTakenPerVAperture = 1750 * 1= 1750
MinF'core_clk' = 37680*1750 = 65.94 MHz
video_in_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=600/37680 = 0.0159
NumOutputPixels=640x480 = 307200
MinFOut (video_out_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 will work 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
MaxClksTakenPerVAperture = 3990 * 1 = 3990
MinF'core_clk' = 67500*3990 = 269.32 MHz
video_in_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=1080/67500 = 0.016
NumOutputPixels=1280x720 = 921600
MinFOut (video_out_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 will not work with or without frame buffers (see Memory Mode, page 23) 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
MaxClksTakenPerVAperture = 2020 * 1 = 3990
MinF'core_clk' = 67500*2020 = 136.35 MHz
```

```
video_in_aclk = frequency defined by live-mode input pixel clock.
SubjectImageTimeTaken=1080/67500 = 0.016
NumOutputPixels=1280x720 = 921600
MinFOut (video_out_aclk) =(921600 / 0.016) * 1.1 = 63.36 MHz
```

Shrink-factor inputs:

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

Note: This conversion will work 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
MaxClksTakenPerVAperture = 2710 * 1 = 2710
MinF'core_clk' = 45000*2710 = 121.95 MHz
video_in_aclk = frequency defined by live-mode input pixel clock. Typically 74.25
MHz.
SubjectImageTimeTaken=720/45000 = 0.016
NumOutputPixels=640x480 = 307200
MinFOut (video_out_aclk) =(307200 / 0.016) * 1.1 = 21.12 MHz
```

Shrink-factor inputs:

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

Note: This conversion will work 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
MaxClksTakenPerVAperture = 3990 * 1 = 3990
MinF'core_clk' = 45000*3990 = 179.55 MHz
video_in_aclk = Frequency defined by live-mode input pixel clock. Typically 74.25
MHz.
SubjectImageTimeTaken = 720/45000 = 0.016
```

```
NumOutputPixels=1920x540 = 1036800
MinFOut (video_out_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 will work in Virtex and Kintex devices, and in higher speed grade Spartan 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
MaxClksTakenPerVAperture = 2020 * 2 = 4040
MinF'core_clk' = 45000*4040 = 181.8 MHz
video_in_aclk = Frequency defined by live-mode input pixel clock. Typically 74.25
MHz.
SubjectImageTimeTaken=720/45000 = 0.016
NumOutputPixels=1920x1080 = 2073600
MinFOut (video_out_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 will work in Virtex and Kintex devices, and in higher speed grade Spartan 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 may read data from the memory at it's leisure. In order 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 may be provided as the internal phase-accumulator dictates, instead of the input timing signals.

The critical factors may 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:

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

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
video_in_aclk = Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.83 MHz
video_out_aclk = Delivery of 1 field of 1920x540 pixels in 1/60 s: Fmin = 68.43
MHz
```

Shrink-factor inputs:

$$hsf=2^{20} \times (1/1.5) = 0x0AAAAA$$
$$vsf=2^{20} \times (1/0.8) = 0x155555$$

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
video_in_aclk - Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.83 MHz
video_out_aclk - Delivery of 1 frame of 1920x1080 pixels in 1/60 s: 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 will work in all devices, including Spartan-6 -2 speed grade devices.

Note: See example 10 for a contrasting conversion.

Resource Utilization

Table 2-8 through Table 2-14 show the resource usage observed for a broad range of scaler configurations and devices. This post-PAR characterization data has been collated through automated implementation of each configuration. This data will vary between implementations, and is intended primarily as a guideline.

Note: When NOT using AXI4Lite interface, subtract approximately 400 FFs and 600 LUTs (all families).

Table 2-8: **Resource Usage for Spartan-6 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 16/8	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2600	2162	17/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2632	2231	18/5	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2664	2207	24/0	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3117	2405	27/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3523	3318	34/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2628	2205	17/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2575	2127	4/5	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2920	2127	24/0	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4871	2509	51/32	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1242	1037	16/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2052	1457	16/0	24

Table 2-9: **Resource Usage for Virtex-7 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2272	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2299	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2299	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2433	16/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	2462	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2342	10/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2216	3/6	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2905	1947	9/9	28

Table 2-9: **Resource Usage for Virtex-7 Devices *(Cont'd)***

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4691	2691	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1237	1065	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2027	1541	6/6	24

Table 2-10: **Resource Usage for Virtex-6 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2134	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2084	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2186	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2190	16/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	2464	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2100	10/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2074	3/6	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2905	2031	9/9	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4691	2532	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1228	1003	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2027	1344	6/6	24

Table 2-11: **Resource Usage for Kintex-7 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2223	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2317	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2314	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2419	16/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	2525	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2266	10/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2255	3/6	12

Table 2-11: **Resource Usage for Kintex-7 Devices *(Cont'd)***

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2906	2254	9/9	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4692	2842	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1228	994	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2031	1451	6/6	24

Table 2-12: **Resource Usage for Artix-7 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2116	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2103	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2132	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2305	16/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	2474	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2152	10/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2061	3/6	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2906	2063	9/9	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4692	2585	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1228	986	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2031	1387	6/6	24

Table 2-13: **Resource Usage for Kintex-7 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2223	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2317	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2314	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2419	16/0	20
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	2525	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2266	10/0	12

Table 2-13: **Resource Usage for Kintex-7 Devices *(Cont'd)***

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2255	3/6	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2906	2254	9/9	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4692	2842	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1228	994	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2031	1451	6/6	24

Table 2-14: **Resource Usage for Zynq-7000 Devices**

AXI4-Lite?	Max Input Rectangle Size (HxV)	Max Output Rectangle Size (HxV)	Bits per Pixel	Taps	Max Phases	Num Engines	Num Coef Sets	FFs	LUTs	BRAM 36/18	DSP48
Yes	1920x1080	1920x1080	8	4x4	4	1	1	2583	2291	10/0	12
Yes	1920x1080	1920x1080	10	4x4	4	1	1	2615	2290	10/6	12
Yes	1920x1080	1920x1080	12	4x4	4	1	1	2647	2297	11/5	12
Yes	1920x1080	1920x1080	8	8x8	4	1	1	3100	2431	16/0	203
Yes	1920x1080	1920x1080	8	11x11	4	1	1	3505	3517	20/0	26
Yes	1920x1080	1920x1080	8	4x4	64	1	1	2611	2300	10/0	12
Yes	512x1080	512x1080	8	4x4	4	1	1	2556	2224	3/6	12
Yes	1920x1080	1920x1080	8	4x4	4	3 (444)	1	2906	2236	9/9	28
Yes	1920x1080	1920x1080	10	11x11	64	3 (444)	1	4691	2790	42/2	70
No	1920x1080	1920x1080	8	4x4	4	1	1	1228	1003	9/0	12
No	1920x1080	1920x1080	8	4x4	4	2	1	2031	1428	6/6	24

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 an 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.

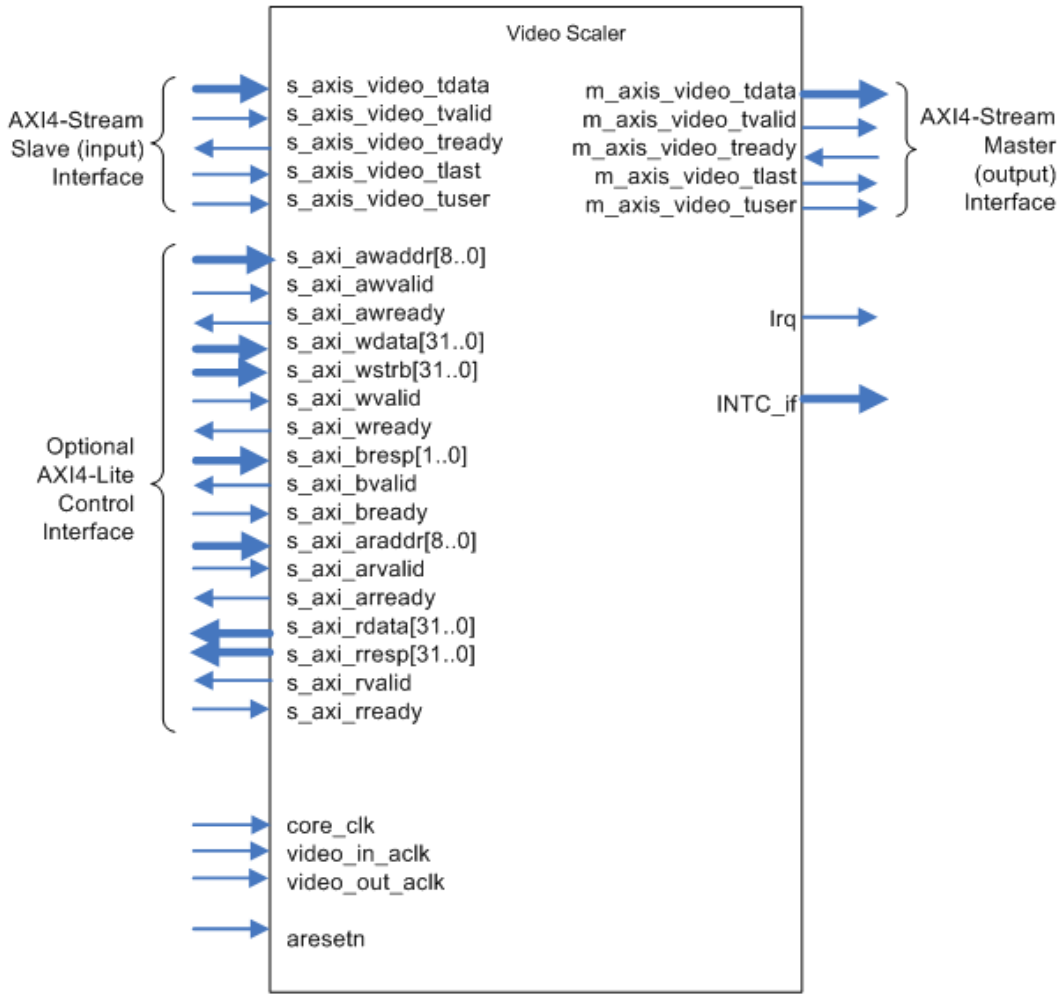

Figure 2-1: **Video Scaler Core Top-Level Signaling Interface**

Common Interface Signals

Table 2-15 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-15: **Common Interface Signals**

Signal Name	Direction	Width	Description
CORE_CLK	In	1	High-speed clock - used for internal processing, not for IO
VIDEO_IN_ACLK	In	1	Input clock - synchronous with incoming video data on s_axis_video AXI4-Stream interface. Also synchronous with AXI4-Lite interface
VIDEO_OUT_ACLK	In	1	Output clock - synchronous with outgoing video data on m_axis_video AXI4-Stream interface.
ARESETn	In	1	Active Low Synchronous reset
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

The CORE_CLK and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and the AXI4-Lite control interface. Refer to The Interrupt Subsystem for a description of the INTC_IF and IRQ pins.

VIDEO_IN_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 VIDEO_IN_ACLK.

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

VIDEO_OUT_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 VIDEO_OUT_ACLK.

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

CORE_CLK

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

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to both the AXI4-Stream and AXI4-Lite interfaces. When ARESETn is set to 0, the core resets at the next rising edge of ACLK.

Data Interface

The Video Scaler core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *AXI Reference Guide (UG761)*, Video IP: AXI Feature Adoption section.

AXI4-Stream Signal Names and Descriptions

Table 2-16 describes the AXI4-Stream signal names and descriptions.

Table 2-16: **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.

The active video data always observes the following principles on an AXI4-Stream tData port:

Given a user-specified Video_Data_Width:

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 2-16.

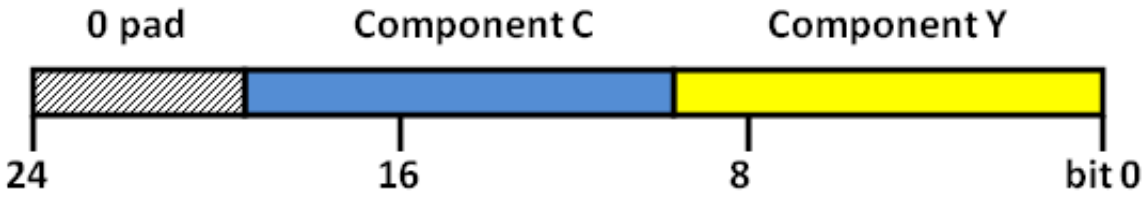

Figure 2-2: **RGB Data Embedding on m_axis_video_tdata**

Then insert the diagram in YUV_Embedding.PNG. This should be named "YUV Data embedding on TDATA"

For the RGB case,

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

An example showing 12-bit RGB data is shown in Figure 2-16.

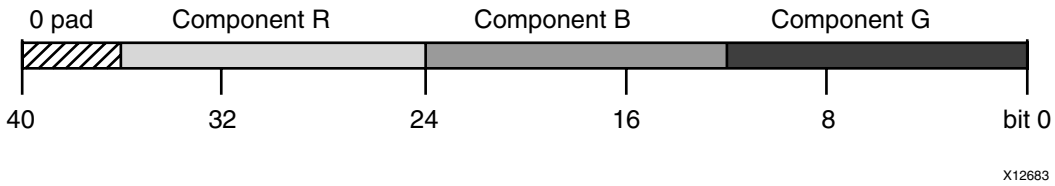

Figure 2-3: **RGB Data Embedding on TDATA**

READY/VALID Handshake

A valid transfer occurs whenever READY, TVALID, 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 tvalid is asserted, no interface signals (except the Video Scaler core driving tready) may change value until the transaction completes (tready, tvalid high on the rising edge of ACLK). Once asserted, tvalid may only be de-asserted after a transaction has completed. Transactions may 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.

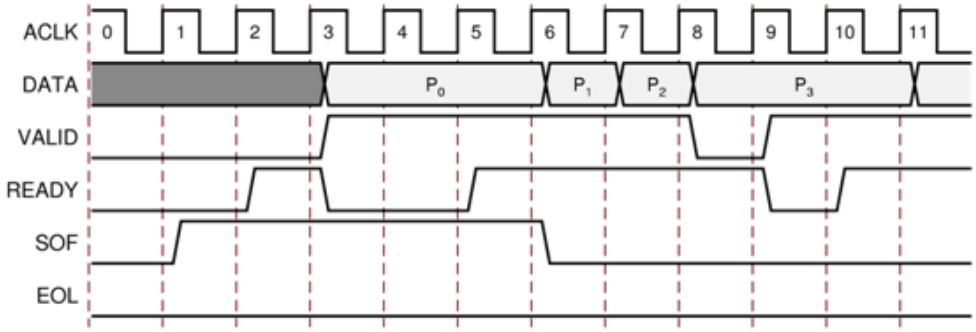

Figure 2-4: **Example of TREADY/TVALID Handshake, Start of a New Frame**

Guidelines on Driving tready into Master (Data Output) Interfaces

The tready signal may be asserted before, during or after the cycle in which the Video Scaler core asserted tvalid. The assertion of tready may be dependent on the value of tvalid. A slave that can immediately accept data qualified by tvalid, should pre-assert its tready signal until data is received. Alternatively, tready can be registered and driven the cycle following TVALID assertion. 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 may be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on TDATA, as long as a TVALID is not asserted.

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.

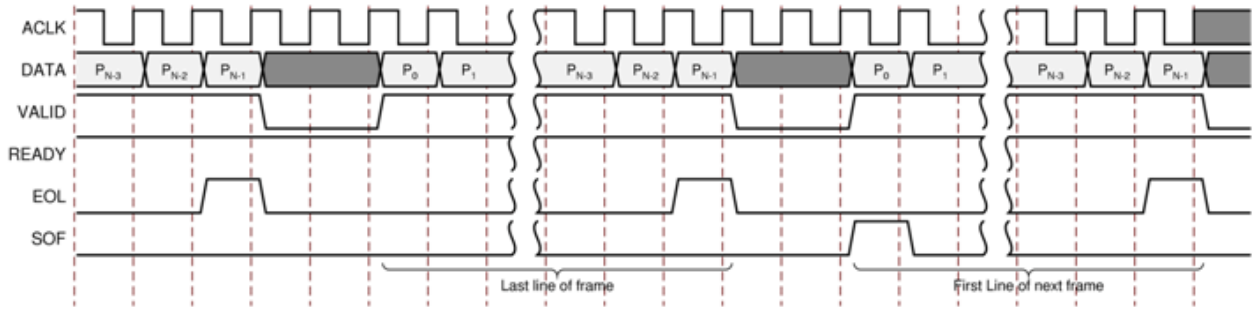

Figure 2-5: **Use of EOL and SOF Signals**

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The Video Scaler core uses only one timing related register, `ACTIVE_SIZE` (0x0020), which allows specifying the input frame dimensions. The core has thirteen core specific registers which allow you to dynamically control the operation of the core.

Table 2-17: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	Control	R/W	No	Pwr-on-Rst: 0x0	b0: C_ENABLE b1: REG_UPDATE b2: reserved b3: CoefMemRdEn b31: SW_RESET (1: reset)
0x0004	Status	R/W	No	0	b0: coef_fifo_rdy
0x0008	(Reserved)	-	-	-	-
0x000C	(Reserved)	-	-	-	-
0x0010	Version	R	No	0x0600a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR

Table 2-17: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0014	(Reserved)	-	-	-	-
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 2-17: **Register Names and Descriptions**

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 4, 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 will 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

All bits of the STATUS register can be used to request an interrupt from the host processor. In order 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 twill be set low. The user must not write further coefficients to the core until this bit goes high.

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 will assert 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 may wish to work this calculation backwards. For a desired scale-factor, you may wish to calculate the output size or the input size. This is application-dependent. Smooth zoom/shrink applications may take advantage of this approach, coupled with usage of 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 values provided by the user in the GUI. In order 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 4

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. The user 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 CORE Generator 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.

Active_Coefs (0x011c) Register

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

Usage of 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 may 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. In order 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 4.

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 may be read using this interface. The coefficient readback mechanism is described in detail in Coefficients in Chapter 4.

Fixed Mode

When using this mode, the values are fixed at compile time. The user system does not need to drive any of the parameters. The CORE Generator 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 may not be written to the core; the coefficient writing function is disabled.
- You may not specify h_coeff_set or v_coeff_set; there is only one set of coefficients.
- You may 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 user 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 user 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 2-18.

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

Table 2-18: **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.
5	intr_coef_mem_rdbk_rdy	Sent low after CoefMemRdEn (control register bit (3)) is written low. Two frames after CoefMemRdEn is written high, this signal is driven high again.

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 the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

Customizing and Generating the Core

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

Graphical User Interface

The Video Scaler core is configured through the CORE Generator Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time.

Figure 3-1 shows the main page of the Video Scaler Coregen GUI.

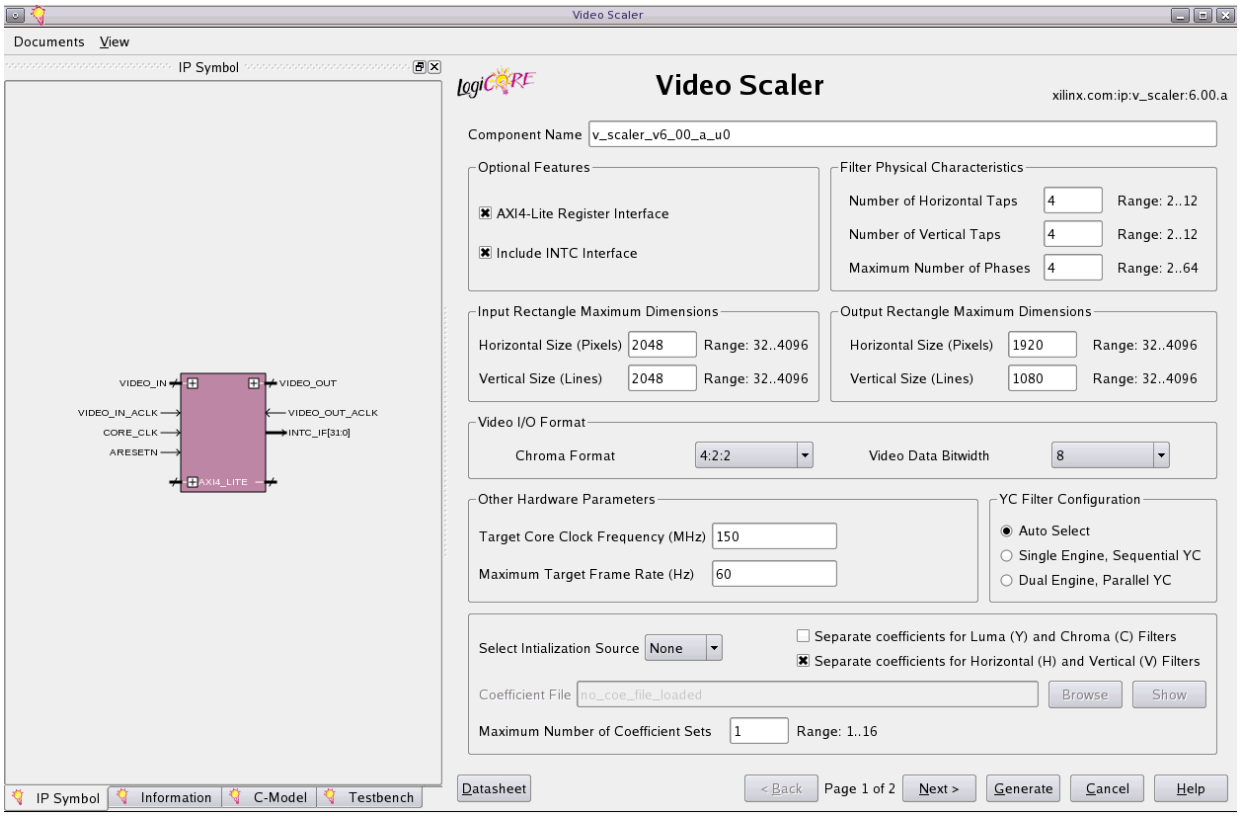

Figure 3-1: **Video Scaler Main Screen**

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

Component Name: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name "v_scaler_v6_00_a" is not allowed.

Optional Features:

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

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

Number of Vertical Taps: This represents the number of multipliers that may be used in the system for the vertical filter, and may vary between 2 and 12 inclusive. The user 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 may 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 may vary between 2 and 16 inclusive, but also may 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 may 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.

Max Coef Sets: This represents the maximum number of sets of coefficients that may 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 4:2:2 (default), 4:2:0, or 4:4:4. Selecting 4:2:0 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.

YC Filter Configuration: When running 4:2:0 or 4:2:2 data, the scaler may 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 "Information" tab indicates the estimated maximum frame-rate achievable given you parameter settings. It makes this decision according to the specified desired frame rate. The user may also manually select between the two options. When in 4:4:4/RGB mode, the scaler is implemented with three engines in parallel.

When the Chroma format is specified as 4:4:4, 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 CORE Generator tool GUI or the EDK GUI (see Figure 3-2).

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 GUI tries to establish 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 by the GUI for the Auto Select option is as 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 Information tab in the CORE Generator interface (not available in EDK GUI) shows the estimated maximum achievable frame rate given the above information using a similar calculation as shown in the sample. The user is advised to take a look at this value, and may 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: The user may specify a .coe file to preload the coefficient store with coefficients. When using Constant mode, this is a necessary step. The .coe file format is described in more detail in Coefficients in Chapter 4.

The user may specify whether the same coefficients are used for Y and C filter operations.The user may 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 may make for a smaller implementation.

Fixed Mode GUI

This option allows the core to be used without any dynamic control. When unchecking the **AXI4-Lite Register Interface** option on page 1 of the GUI, the options on page 2 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 2 of the GUI. Page 2 of the GUI is shown in Figure 3-2.

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

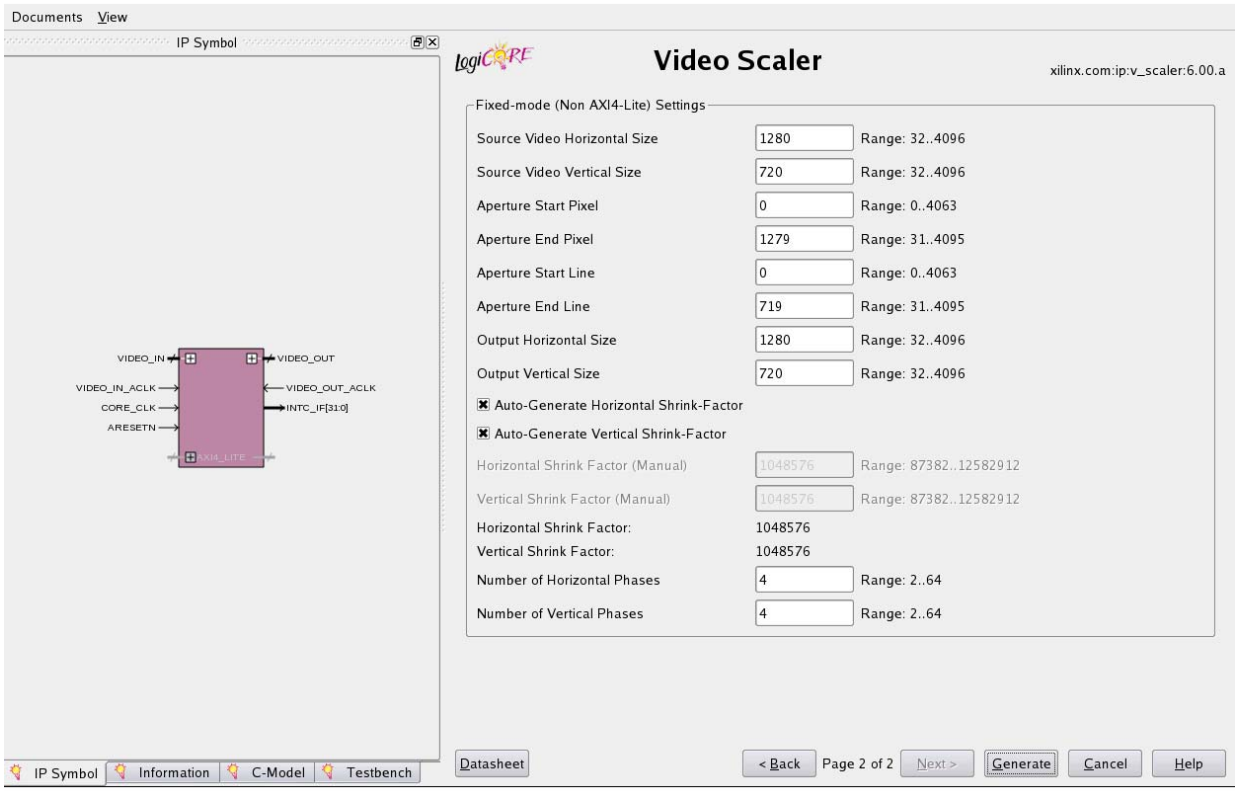

Figure 3-2: **Video Scaler Graphical User Interface for Constant Mode (page 2)**

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 4. 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. The user 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.

Coefficient File Input: The user 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 4.

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 may 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.

EDK GUI

Definitions of the EDK GUI controls are identical to the corresponding CORE Generator GUI functions. The EDK GUI is shown in Figure 3-3.

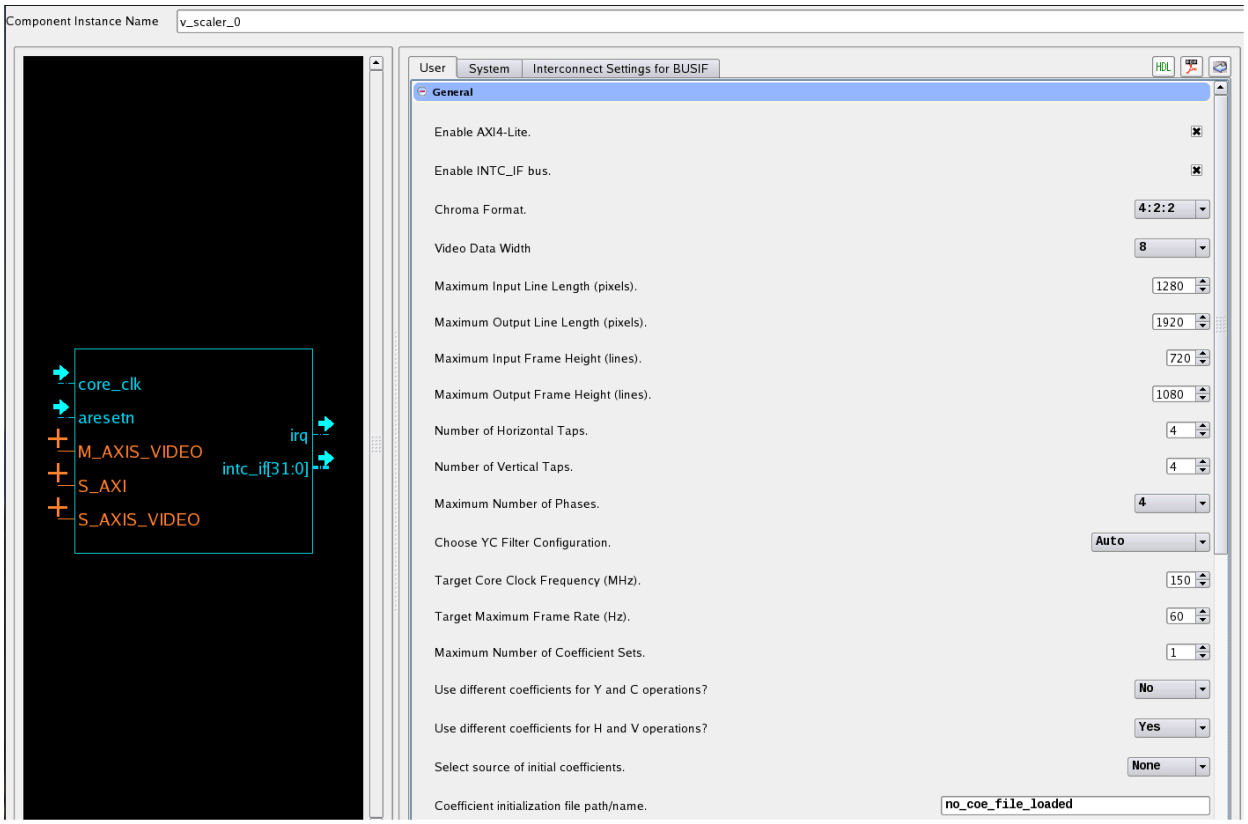

Figure 3-3: **Video Scaler EDK GUI**

Fully verified MicroBlaze processor software driver source code is also provided by CORE Generator software for driving all of the control inputs.

Parameter Values in the XCO File

Table 3-1 defines valid entries for the Xilinx CORE Generator (XCO) parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 3-1: **XCO Parameter Values**

XCO Parameter	Default	Valid Values
aperture_end_line	719	32 - 4095
aperture_end_pixel	1279	32 - 4095
aperture_start_line	0	0 - 4063
aperture_start_pixel	0	0 - 4063
chroma_format	4:2:2	4:2:0, 4:2:2, 4:4:4

Table 3-1: **XCO Parameter Values**

XCO Parameter	Default	Valid Values
coefficient_file	no_coe_file_loaded	'no_coe_file_loaded', <valid coe file>
component_name	v_scaler_v6_00_a_u0	Not 'v_scaler_v6_00_a'
data_width	8	8, 10, 12
horizontal_scale_factor	1048576	87382 - 12582912
init_coef_source	None	'None', 'COE_File'
HAS_AXI4_LITE	false	true, false
m_axis_tdata_width	16	16, 24, 32, 40
maximum_number_of_active_lines_per_input_frame	2048	32 - 4096
maximum_number_of_active_lines_per_output_frame	1080	32 - 4096
maximum_number_of_active_pixels_per_input_line	2048	32 - 4096
maximum_number_of_active_pixels_per_output_line	1920	32 - 4096
maximum_number_of_coefficient_sets	1	1 - 16
maximum_number_of_phases	4	2 – 16, 32, 64
number_of_horizontal_phases	4	2 – 16, 32, 64
number_of_horizontal_taps	4	2 – 12
number_of_vertical_phases	4	2 – 16, 32, 64
number_of_vertical_taps	4	2 – 12
output_horizontal_size	1280	32 - 4096
output_vertical_size	720	32 - 4096
s_axis_tdata_width	16	16, 24, 32, 40
separate_hv_coefs	true	true, false
separate_yc_coefs	false	true, false
target_core_clk_freq_mhz	150	0 - 550
target_max_frame_rate	60	0 - 120
vertical_scale_factor	1048576	87382 - 12582912
yc_filter_config	0	0, 1, 2

Output Generation

The output files generated from Xilinx CORE Generator for the Video Scaler core depend upon if the AXI4-Lite interface is selected. The output files are placed in the project directory.

File Details

The CORE Generator output consists of some or all the following files.

Table 3-2: **CORE Generator Output**

Name	Description
<component_name>.xco	CORE Generator input file containing the parameters used to generate a core.
<component_name>.ngc	Binary Xilinx implementation netlist files containing the information required to implement the module in a Xilinx (R) FPGA.
<component_name>.vho <component_name>.veo	Template files containing code that can be used as a model for instantiating
<component_name>.vhd <component_name>.v	Structural simulation model
/pg009_v_scaler.pdf /doc/v_scaler_v6_00_a_vinfo.	Core documents
<component_name>.asy	Graphical symbol information file. Used by the ISE tools and some third party tools to create a symbol representing the core.
<component_name>_xmdf.tcl	ISE Project Navigator interface file. ISE uses this file to determine how the files output by CORE Generator for the core can be integrated into your ISE project.
<component_name>.gise <component_name>.xise	ISE Project Navigator support files. These are generated files and should not be edited directly.
<component_name>_readme.txt	Readme file for the IP.
<component_name>_flist.txt	Text file listing all of the output files produced when a customized core was generated in the CORE Generator.

Designing with the Core

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

Basic Architecture

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 4-1).

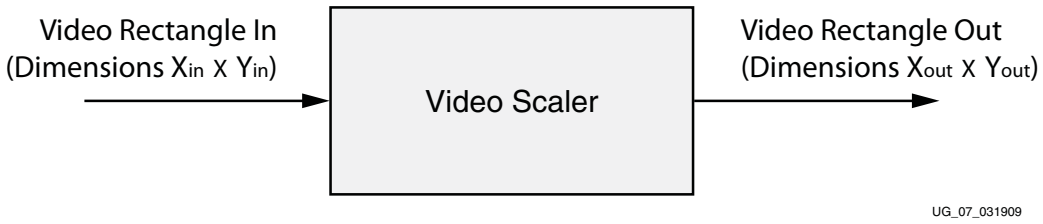

Figure 4-1: **High Level View of the Functionality**

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

The Xilinx Video Scaler makes few assumptions regarding the origin or the destination of the video data. The input could be fed in real-time from a live video feed, or it could be read from an 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, and
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, it's timing is specific, regular and defined. Over-zealous de-assertion of s_axis_video_tready will cause 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 in order to generate the desired scaled outputs. This naturally takes time. If the

amount of time taken approaches or exceeds the amount of time it takes to deliver the input data into the scaler in the first place, then these errors may occur. 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 Throughput in Chapter 2.

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.
- Generally, the output data does not conform to any standard. It is therefore not possible to feed the output directly to a display driver. Usually, a frame buffer is ultimately required to smooth the output data over an output frame period. The output video stream is described later.

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 4-2 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. The user may weight the coefficients to reflect no bias in either direction, and may even select a unity coefficient set. Output location (2, 2) is offset from the input grid in both vertical and horizontal dimensions. Coefficients may be chosen to reflect this, most likely showing some

bias towards input pixel (2, 2), etc. Filter characteristics may be built into the filter coefficients by appropriately applying anti-aliasing low-pass filters.

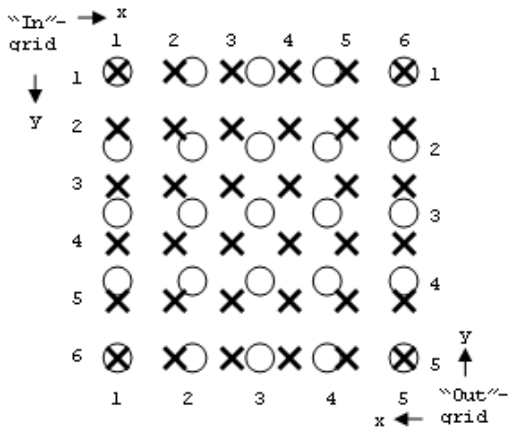

Figure 4-2: **5x5 Output Grid ("O") Super-imposed over 6x6 Input Grid ("X")**

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 will always fall 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 4-3. As illustrated in this figure, the five output pixels shown from left to right could have the phases 0, 1, 2, 3, 0.

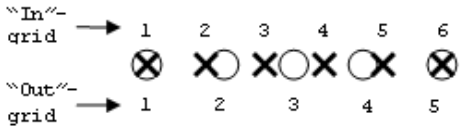

Figure 4-3: **Super-imposed Grids for 1 Dimension**

The examples in Figure 4-2 and Figure 4-3 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. Knowledge of this factor is required before using the scaler, and it is a direct input to the system. Usually it is defined by the system requirements at a higher level, and it may be different in H and V dimensions. A typical example is drawn from the broadcast industry, where some footage may 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).

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 4-4, as implemented for a 4:2:2 or 4:2:0 scaler. DataWidth may be set to 8, 10, or 12 bits.

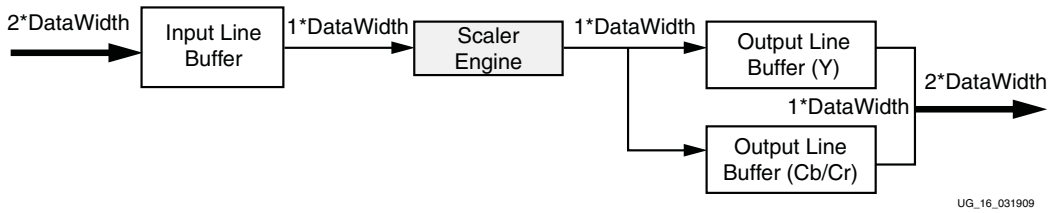

Figure 4-4: **Internal Data Path Bitwidths for Single-Engine YC Mode**

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 will be 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 4-5.

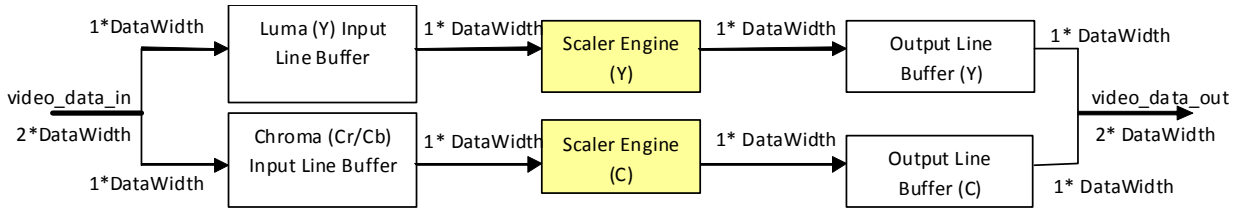

Figure 4-5: **Internal Data Path Bitwidths for Dual-Engine YC Mode**

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 4-6.

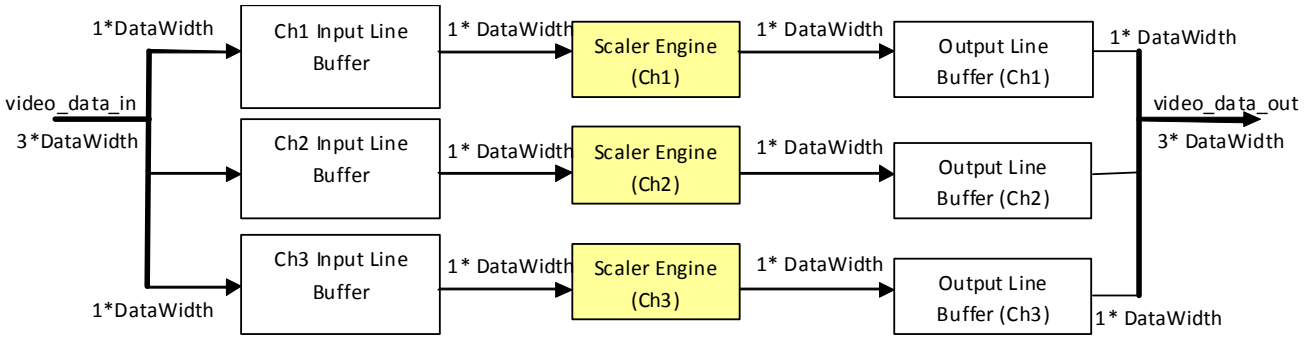

Figure 4-6: **Internal Data Path Bitwidths for Triple-Engine RGB/4:4:4 Architecture**

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

Clocking

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

- `video_in_aclk` handles the clocking of data into the core.
- `core_clk` is used internally to the core.
- `video_out_aclk` is the clock that will be used to read video data out from the core.

Figure 4-7 shows the top level buffering, indicating the different clock domains, and the scope of the control state-machines.

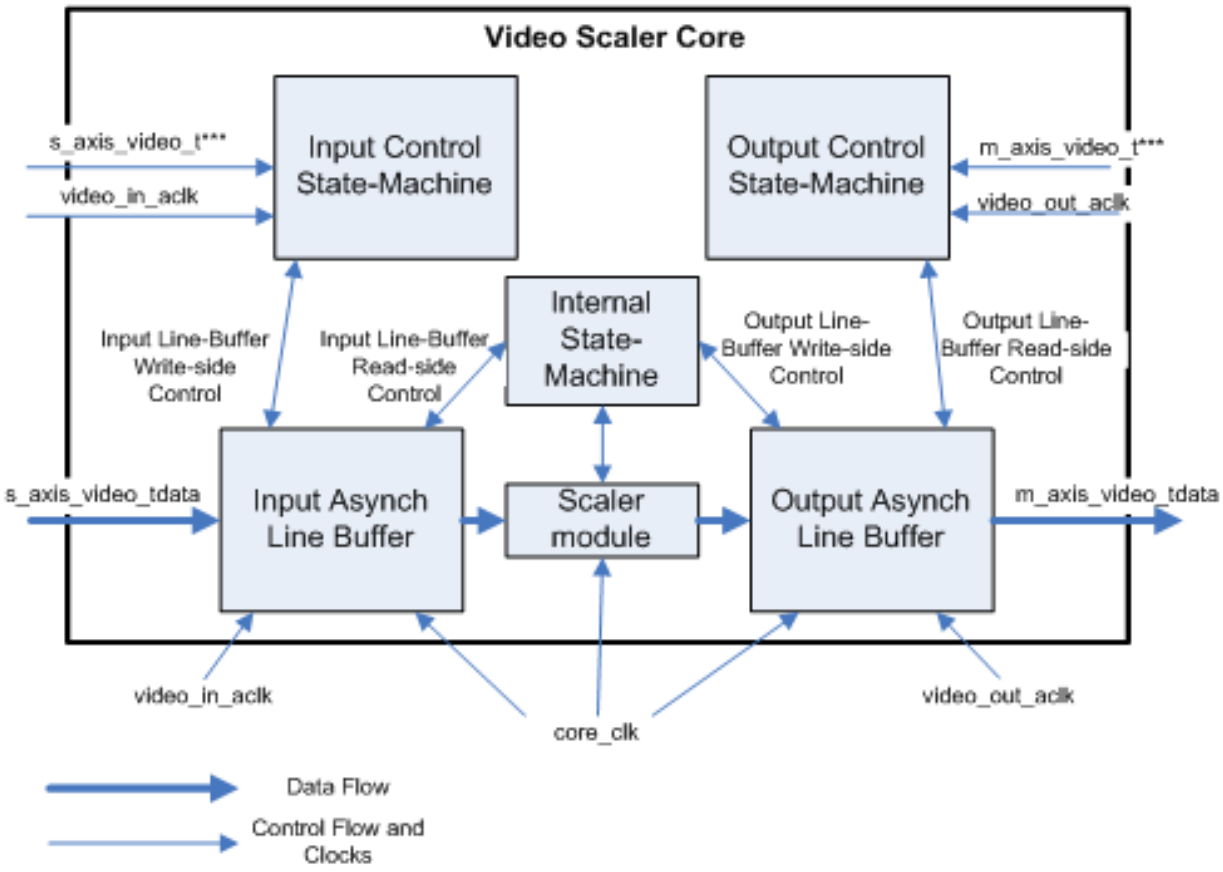

Figure 4-7: **Block Diagram Showing Clock Domains**

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 4-7. The control state-machines monitor the I/O line buffers. They also monitor the current input and output line numbers.

Clock, Enable, and Reset Considerations

Figure 4-8 shows a typical example of the Video Scaler in a system.

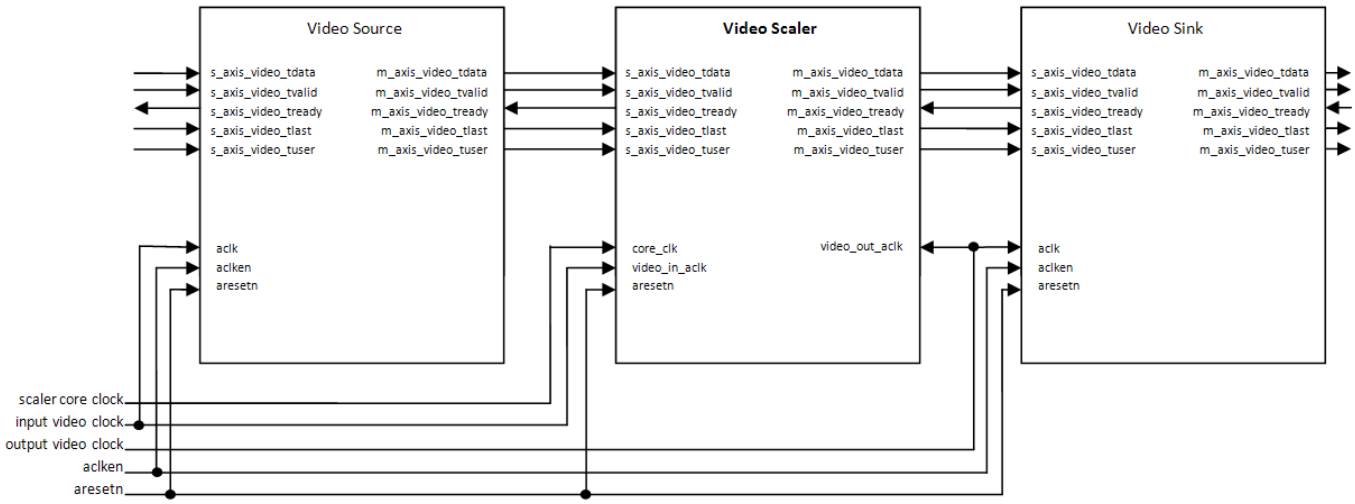

Figure 4-8: **Video Scaler Example**

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. The user may 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.

AResetn

The Video Scaler core has two reset sources: the ARESETn pin (hardware reset), and the software reset option provided via the AXI4-Lite control interface (when present).

Note: ARESETn is by no means synchronized internally to AXI4-Stream frame processing, therefore de-asserting ARESETn while a frame is being process will lead to image tearing. The external reset pulse needs to be held for 32 ACLK cycles to reset the core.

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.

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 4-1.

Table 4-1: **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).

Cropping

You may 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 4-2.

Table 4-2: **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 may be populated with active coefficients as follows:

- Using the Coefficient Interface (see Coefficient Interface).
- By preloading using a .coe file

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 may 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. The user may load (or preload using a .coe file) multiple coefficient sets.

The number of phases for each set may 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 may 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 may 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 will be 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 will be shared between all three channels (i.e., R, G, and B channels will be scaled identically).

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

Filter Coefficients may 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 4-9.

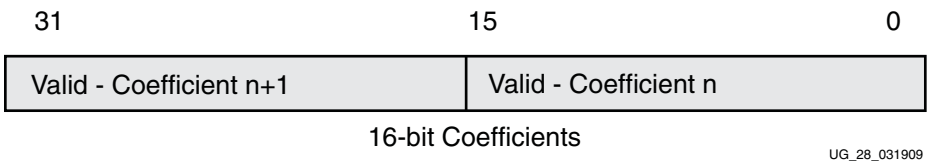

Figure 4-9: **Coefficient Write-Format on coef_data_in(31:0)**

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 very quickly during the next vertical blanking period between the processing of 2 consecutive video frames, but only 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 will be 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 will be 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 as follows:

- 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 will be applied to the latest tap (that is, spatially speaking, the right-most or lowest). The word format is shown in Figure 4-9.
- 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 will be 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 4-3.

Table 4-3: **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 will appear at `coef_mem_output`.

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

Coefficients may 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 will not cause image distortion, and can be executed during normal operation.

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 4-4 shows a set of coefficients drawn from a sinc function.

Table 4-4: **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 will cause 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. 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 4-5.

Table 4-5: **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 4-6.

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 Y and C coefficients)

Table 4-6: **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	

Table 4-6: **Example 1 Coefficient Set Download Format *(Cont'd)***

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	
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	

Table 4-6: **Example 1 Coefficient Set Download Format *(Cont'd)***

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	
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 4-7 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 4-7). 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 4-10).

Table 4-7: **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	
Phase 2	9	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	41	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	Phase 2
	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	Phase 3
	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	Phase 4
	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	Phase 5
	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	Phase 6
	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	

Table 4-7: **Example 2 Coefficient Set Download Format *(Cont'd)***

	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 7	29	0x00000000	N/A Dummy coef	61	0x00000000	N/A Dummy coef	Phase 7
	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	
Phase 0	65	0x00000000	(Ph0 T1 << 16) | Ph0 T0	97	0x00000000	(Ph0 T1 << 16) | Ph0 T0	Phase 0
	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	
Phase 1	69	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	101	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	Phase 1
	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	Phase 2
	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	Phase 3
	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	Phase 4
	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	Phase 5
	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	

Table 4-7: **Example 2 Coefficient Set Download Format *(Cont'd)***

Phase 6	89	0x00000000	N/A Dummy coef	121	0x00000000	N/A Dummy coef	Phase 6
	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	Phase 7
	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 4-8 (horizontal) and Table 4-9 (vertical).

Table 4-8: **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 4-9: **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 4-10.

Table 4-10: **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	
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	

Table 4-10: **Example 3 Coefficient Set Download Format *(Cont'd)***

	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	
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 extracting coefficients from a file provided with the core (Extracting Coefficients From xscaler_coefs.c File) 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.

Extracting Coefficients From xscaler_coefs.c File

The EDK version of the video scaler includes a software driver. The coefficients are included in this driver in the `xscaler_coefs.c` file. The pCore version of the core can be generated by selecting "EDK pCore" in the CORE Generator GUI, found in the EDK XPS GUI. Coefficients from this file can be extracted manually; however, it is important to know the format of this file.

All coefficients required for any conversion are provided with the SW Driver. The filename is `xscaler_coefs.c`. You may modify this file, and the driver code that reads the coefficients from it, as you see fit.

The file defines 19 "bins" of coefficients. You must select which bin to use according to your application. In the delivered driver, the file `xscaler.c` includes a function called `XScaler_CoeffBinOffset`, which assesses the scaling requirements specified by you (for example, input/output rectangle sizes) and calculates which bin of coefficients is required. In this driver, the bins have been allocated as per Table 4-11. This function may be used independently for all Horizontal, Vertical, Luma, and Chroma filter operations.

Table 4-11: **Coefficient "Binning" in SW Driver (xscaler_coefs.c)**

Bin #	SF=input_size /output_size	Comments
1	SF<1	All up-scaling cases
1+**Ceil**((output_size*16)/input_size) (bins 2 to 17) For example: • Down-scaling 1920 to 1440: use bin 13 • Down-scaling 1080 to 1000 : Use bin 16 • Down-scaling 1080 to 144 : Use bin 4	1<SF<16 (All down-scaling cases)	General down-scaling coefficients Down-scaling filter coefficients include anti-aliasing characteristics that differ according to scale-factor

Table 4-11: **Coefficient "Binning" in SW Driver (xscaler_coefs.c)**

Bin #	SF=input_size /output_size	Comments
18	N/A	Unity coefficient in center tap
19	1920/1280 (1080/720)	Example user-specific case for HD down scaling conversion

Within each "bin," four further levels of granularity can be observed. In order of decreasing size of granularity, these levels are:

- Number of taps defined
- Number of phases defined
- Phase number (one line in file)
- Tap number (one element of each line), newest (right-most or lowest) first

For example, the first set of coefficients, defined for two taps and ***two phases***, is given as:

```
// bin # 1; num_taps = 2; num_phases = 2
1018, 15366,
8192, 8192
```

The second set of coefficients, defined for two taps and ***three phases,*** is given immediately afterwards as:

```
/* bin # 1; num_taps = 2; num_phases = 3 */
1018, 15366,
5852, 10532,
10532, 5852,
```

And so forth.

Format for .coe Files

The guidelines for creating a .coe file are as follows:

- Coefficients may 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 4-12: **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 4-13 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 4-13: **.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,
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,

Table 4-13: **.coe File Example 1 *(Cont'd)***

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
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,
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 4-14 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's sake, 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 4-14: **.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,
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,

Table 4-14: **.coe File Example 2 *(Cont'd)***

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,
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 4-15 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 4-15: **.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,	
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,	

Table 4-15: **.coe File Example 3 *(Cont'd)***

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	""	

Constraining the Core

This chapter contains applicable constraints for the Video Scaler core.

Required Constraints

When using multiple clock domains in the Video Scaler and the AXI4Lite interface is enabled, control data passes from the AXI4Lite interface to the domain in which it being used. The AXI4Lite interface operates using the video_in_aclk domain. Internal hardware ensures a clean transfer between this clock domain and the core_clk and video_out_aclk domains, but for these cases, care should be taken to ensure that the paths that cross between these domains are ignored in order to allow the design to meet timing constraints.

Refer to Xilinx documentation for guidelines on how to constrain properly.

Device, Package, and Speed Grade Selections

There are no Device, Package or Speed Grade requirements for this core. This core has not been characterized for use in low power devices.

Clock Frequencies

The core clock (`core_clk`), the video input clock (`video_in_aclk`) and the video output clock (`video_out_aclk`) all need to be constrained to the frequency at which the user expects to run. Calculation of the frequencies to which these clocks must be constrained is outlined in Performance in Chapter 2.

Clock Management

The scaler contains no clock managers, DCMs, PLLs, or other clocking modules. All clocks must be driven into the Video Scaler core from an appropriate source.

Clock Placement

There are no specific clock placement requirements for this core.

Banking

There are no specific banking requirements for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

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

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.

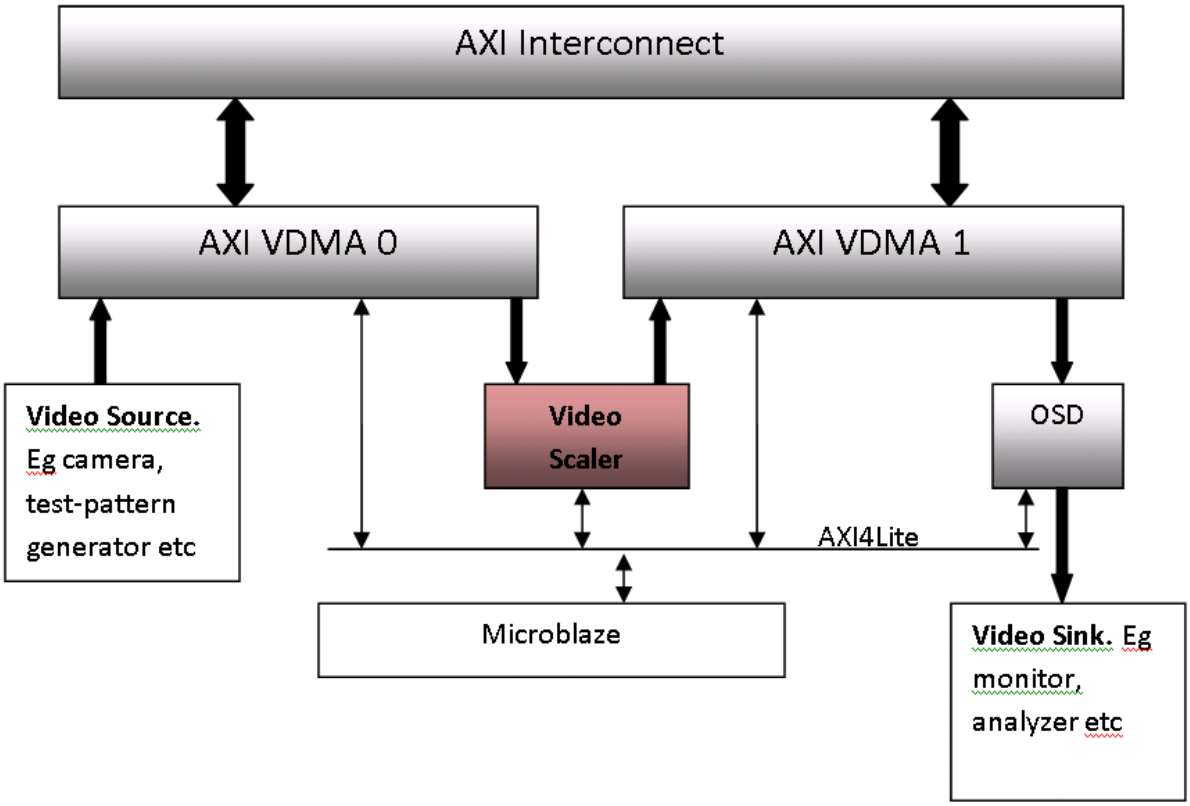

Figure 6-1: **Simplified System Diagram**

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. In order 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.

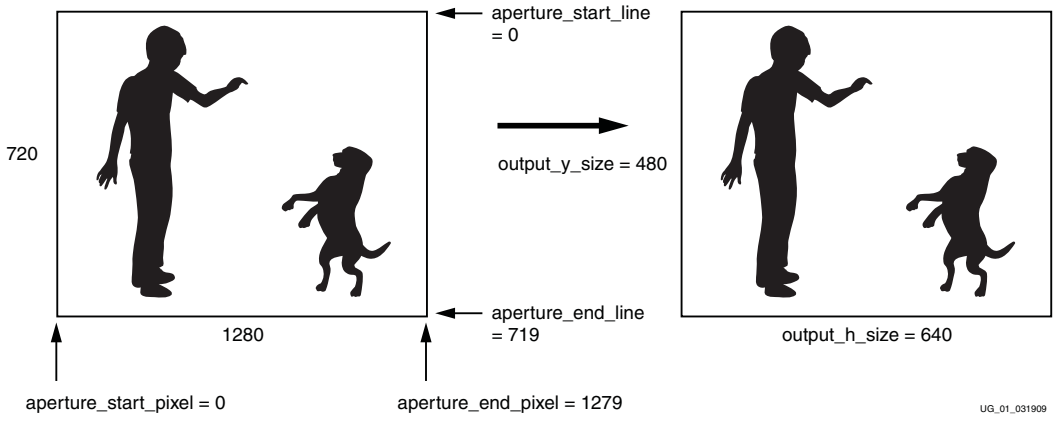

Figure 6-2: **Format Down-scaling. Example 720p to 640x480, HSF = 2^{20} x 1280/640; VSF = 2^{20} x 720/480**

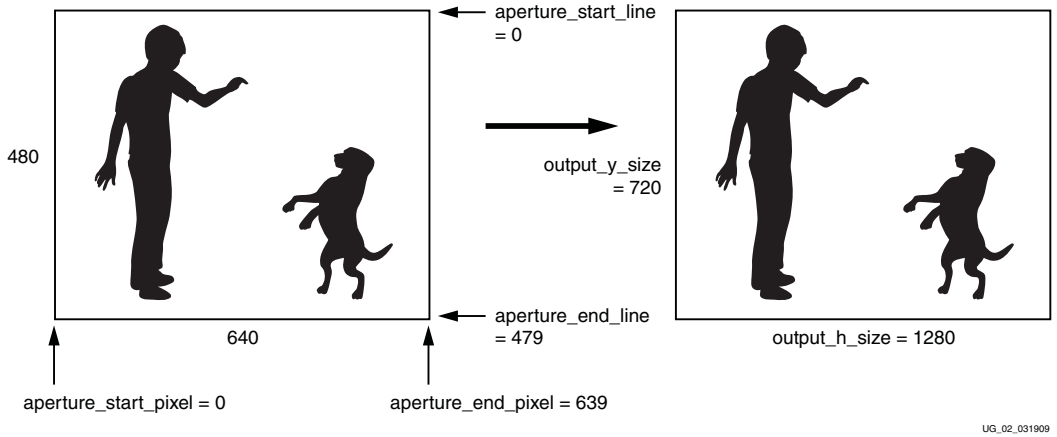

Figure 6-3: **Format Up-scaling. Example 640x480 to 720p, HSF = 2^{20} x 640/1280; 2^{20} x VSF = 480/720**

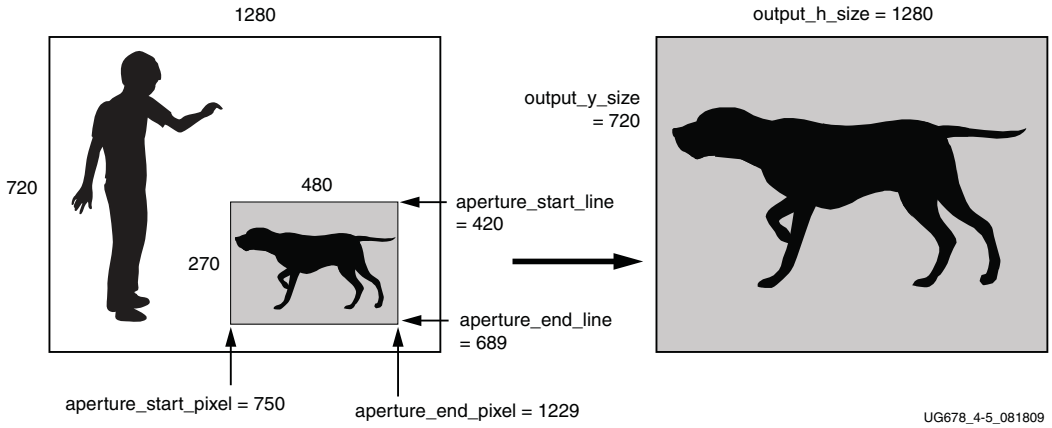

Figure 6-4: **Zoom (Up-scaling), HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

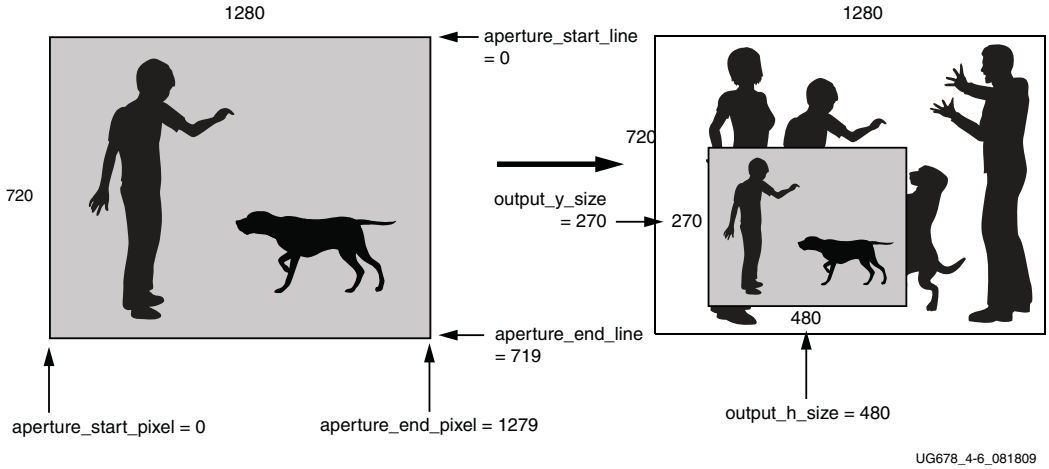

Figure 6-5: **Shrink (Down-scaling). Example for Picture-in-Picture (PinP), HSF = 2^{20} x 1280/480; VSF = 2^{20} x 720/270**

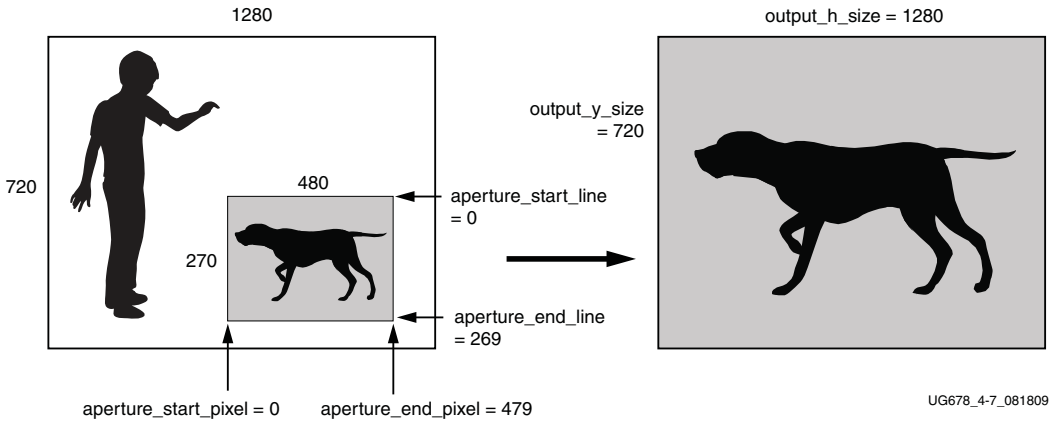

Figure 6-6: **Zoom (Up-scaling) reading from External Memory, HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

Demonstration Test Bench

A demonstration test bench is provided which enables core users to observe core behavior in a typical use scenario. The user is encouraged to make simple modifications to the test conditions and observe the changes in the waveform.

Test Bench Structure

The top-level entity, `tb_main.v`, instantiates the following modules:

- DUT

  The Video Scaler 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 opens the stimuli txt file and initiates AXI4-Stream transactions to provide stimuli data for the core

- axi4s_video_slv

  The AXI4-Stream slave module, which opens the result txt file and verifies AXI4-Stream transactions from the core

- ce_gen

  Programmable Clock Enable (`ACLKEN`) generator

Running the Simulation

- Simulation using ModelSim for Linux:
  From the console, Type "source run_mti.sh".
- Simulation using iSim for Linux:
  From the console, Type "source run_isim.sh".
- Simulation using ModelSim for Windows:
  Double-click on "run_mti.bat" file.

- Simulation using iSim:
  Double-click on "run_isim.bat" file.

Directory and File Contents

The directory structure underneath the top-level folder is:

- expected:
  Contains the pre-generated expected/golden data used by the testbench to compare actual output data.
- stimuli:
  Contains the pre-generated input data used by the testbench to stimulate the core (including register programming values).
- Results:
  Actual output data will be written to a file in this folder.
- Src:
  Contains the .vhd simulation files and the .xco CORE Generator parameterization file of the core instance. The .vhd file is a netlist generated using CORE Generator. The .xco file can be used to regenerate a new netlist using CORE Generator.

The available core C-model can be used to generate stimuli and expected results for any user bmp image. For more information, refer to Appendix E, C Model Reference.

The top-level directory contains packages and Verilog modules used by the test bench, as well as:

- isim_wave.wcfg:
  Waveform configuration for ISIM
- mti_wave.do:
  Waveform configuration for ModelSim
- run_isim.bat :
  Runscript for iSim in Windows
- run_isim.sh:
  Runscript for iSim in Linux
- run_mti.bat:
  Runscript for ModelSim in Windows
- run_mti.sh:
  Runscript for ModelSim in Linux

Appendix A

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 in order 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 in order to validate new core versions during development, and also to guard against EDK tools regressions.

Appendix B

Migrating

From v5.0 to v6.00.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.
    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 may 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 v6.00.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.

Appendix C

Debugging

Some general debugging tips are as follows:

- Verify that the Version register can be read properly. See Table 2-17, page 35 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 in order to introduce known data into the video stream.

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 may 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.

Bringing up the AXI4-Lite Interface

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.	Is the ACLK pin connected? In EDK, verify the ACLK pin connection in the system.mpd file. Does the core receive ACLK? The ACLK pin is shared by the AXI4-Lite and AXI4-Stream interfaces. The VERSION_REGISTER readout issue may be indicative of the core not receiving video clock, suggesting an upstream problem in the AXI4-Stream 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? Is ACLKEN connected to vcc? In EDK, verify that signal ACLKEN is connected in **system.mpd** to either net_vcc or to a designated clock enable signal.
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? ARESETn should be connected to vcc for the core not to be in reset. In EDK, verify that signal ARESETn is connected in **system.mpd** as 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.

If the AXI4-Lite communication is healthy, but the data seems corrupted, the next step is to verify the configuration for the Video Scaler core. Please check that all dynamic registers have been set correctly.

Bringing up the AXI4-Stream Interface

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

Table C-2: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Slave (input-side) s_axis_video_tready output(s) stuck low; the upstream core cannot send data.	During initialization, the core drives its slave s_axis_video_tready signals low. Afterwards, the core should assert tready automatically. Is the Master m_axis_video_tready signal being driven into the core High? If not, the Scaler core cannot send data downstream, and the internal FIFOs fill up, ultimately causing s_axis_video_tready to be deasserted.
Master (output-side) m_axis_video_tready signal stuck low; the downstream core is not receiving data.	Is the upstream m_axis_video_tready signal high on the Master (output) AXI4-Stream interface? Is the Video Scaler core actually receiving data at its input interface?
Data samples lost between Upstream core and the Video Scaler core, or between Video Scaler core and downstream core. Inconsistent EOL and or SOF periods received by downstream core.	• Check that data and clocks on all AXI4Stream interfaces are synchronous as expected. If not, is proper clock-domain crossing logic (Asynchronous FIFO) instantiated appropriately? • Did the design meet timing? • Is the frequency of the clock source driving all of the clock pins lower than the reported Fmax reached?

Interfacing to Third-Party IP

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 will correctly identify 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.

Application Software Development

Introduction

This appendix provides a description of how to program and control the data flow for the video scaler hardware EDK pCore. The information is sufficient for the development of a software driver (API) for use in application software for applications such as video conferencing and video analytics.

Note: A software driver is provided with the pCore so that you do not have to develop a software API as described here.

Conventions

Reserved locations in the registers will be ignored by the hardware and can be written by software with any value. Therefore the software does not need to zero or mask bits.

Unused coefficients should be set to zero. The number of taps is a compile time parameter for the IP core and needs to be known by the programmer to be able to load the coefficient tables correctly.

Filter Coefficient Calculations

The values for the filter coefficients can be calculated with any standard digital filter tool. MATLAB® software provides a tool box for establishing the filter coefficients once the cutoff frequency is known from the scale factor. It should be noted that sharp cutoff frequencies are generally not desired in image processing due to the ringing generated at sharp transitions (artifacts). Additionally allowing some amount of aliasing can be subjectively preferred in side-by-side comparisons. The MATLAB software FIR1 function can be used as a starting point for deriving coefficient values.

Xilinx provides a C-Model that generates coefficients. Contact Xilinx support for information on how to obtain this C-Model. Refer to the Video Scaler Product Page for information about accessing the C-Model.

API Function Calls

The following functions are proposed for L0, L1, L2 API.

L0 API Function Calls

#define XScaler_Enable(InstancePtr)

#define XScaler_Disable(InstancePtr)

#define XScaler_Reset(InstancePtr)

#define XScaler_GetStatus(InstancePtr)

#define XScaler_CheckDone(InstancePtr)

#define XScaler_SetHoriShrinkFactor(InstancePtr, Integer, Fractional)

#define XScaler_GetHoriShrinkFactor(InstancePtr)

#define XScaler_SetVertShrinkFactor(InstancePtr, Integer, Fractional)

#define XScaler_GetVertShrinkFactor(InstancePtr)

#define XScaler_SetHoriAperture(InstancePtr, FirstPixel, LastPixel)

#define XScaler_GetHoriAperture(InstancePtr)

#define XScaler_SetVertAperture(InstancePtr, FirstLine, LastLine)

#define XScaler_GetVertAperture(InstancePtr)

#define XScaler_SetOutputSize(InstancePtr, Lines, Pixels)

#define XScaler_GetOutputSize(InstancePtr)

#define XScaler_SetNumPhases(InstancePtr, Vert, Hori)

#define XScaler_GetNumPhases(InstancePtr)

#define XScaler_SetCoeffSet(InstancePtr, Vert, Hori)

#define XScaler_GetCoeffSet(InstancePtr)

#define XScaler_SetHoriAccuLuma(InstancePtr, Fraction)

#define XScaler_GetHoriAccuLuma(InstancePtr)

#define XScaler_SetVertAccuLuma(InstancePtr, Fraction)

#define XScaler_GetVertAccuLuma(InstancePtr)

#define XScaler_SetHoriAccuChroma(InstancePtr, Fraction)

#define XScaler_GetHoriAccuChroma(InstancePtr)

#define XScaler_SetVertAccuChroma(InstancePtr, Fraction)

#define XScaler_GetVertAccuChroma(InstancePtr)

#define XScaler_SetWriteCoeffBankAddr(InstancePtr, Address)

#define XScaler_GetWriteCoeffBankAddr(InstancePtr)

#define XScaler_SetCoefValue(InstancePtr, NPlus1, N)

#define XScaler_GetCoefValue(InstancePtr)

L1 API Function Calls

#define XScaler_CalcCoeffs(coeffs, scale, taps, phases, coeff_precision)

- software function, no registers written

#define XScaler_WriteCoeffValues(InstancePtr, coeffs, coeff_bank)

- sets coef_write_set_addr and writes consecutively coef_values

#define XScaler_CalcScaleFactors(InstancePtr, hsv, vsf, input_h, input_v, output_h, output_v)

- software function, no registers written

#define XScaler_SetActiveCoeffBank(InstancePtr, coeff_bank)

- sets active register coeff_sets

#define XScaler_SetScalerValues(InstancePtr, reg_data_structure)

- This is the main video scaler function call utilized in a frame basis when the shrink factor is changing every frame such as zooming applications.

The mandatory registers that need to change for a new shrink factor are:

- horz_shrink_factor
- vert_shrink_factor
- output_size

Optionally these registers may also need to be modified depending on the input resolution and user preference:

- aperture_horz
- aperture_vert
- num_phases
- coeff_sets
- start_hpa_y
- start_vpa_y
- start_hpa_c
- start_vpa_c

Example Settings

The following examples illustrate settings for different scale factors.

Pass Through

Table D-1 is an example of pass through of a 1280 x 720 resolution image.

Table D-1: **Pass Through Register Settings**

Address	Name	Decimal Value
0x0000	control	03
0x0010	hsf	1048576
0x0014	vsf	1048576
0x0018	aperture_start_pixel	0
0x0018	aperture_end_pixel	1279
0x001c	aperture_start_line	0
0x001c	aperture_end_line	719
0x0020	Output_h_size	1280
0x0020	Output_v_size	720
0x0024	num_h_phases	4
0x0024	num_v_phases	4
0x0028	h_coeff_set	0
0x0028	v_coeff_set	0
0x002c	start_hpa _y	0
0x0030	start_hpa_c	0
0x0034	start_vpa_y	0

Table D-1: **Pass Through Register Settings**

Address	Name	Decimal Value
0x0038	start_vpa_c	0
0x003c	Coef_set_write_addr	0
0x0040	Coef_values	See Coefficients in Chapter 4

Down Sample by 2 in Both Horizontal and Vertical Dimensions

Table D-2 is an example of scaling down a 1280 x 720 resolution image by a factor of 2 horizontally and vertically to 640x 360.

Table D-2: **Down Sample Register Settings**

Address	Name	Decimal VAlue
0x0000	control	07
0x0010	hsf	2097152
0x0014	vsf	2097152
0x0018	aperture_start_pixel	0
0x0018	aperture_end_pixel	1279
0x001c	aperture_start_line	0
0x001c	aperture_end_line	719
0x0020	Output_h_size	640
0x0020	Output_v_size	360
0x0024	num_h_phases	4
0x0024	num_v_phases	4
0x0028	h_coeff_set	0
0x0028	v_coeff_set	0
0x002c	start_hpa_y	0
0x0030	start_hpa_c	0
0x0034	start_vpa_y	0
0x0038	start_vpa_c	0
0x003c	Coef_set_write_addr	0
0x0040	Coef_values	See Coefficients in Chapter 4

C Model Reference

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

Features

- Bit accurate with v_scaler_v6_00_a 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 v6.00.a 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_v6_00_a_bitacc.zip` file must first be uncompressed. Once this is completed, the directory structure and files shown in Table E-1 are available for use.

Table E-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_v6_00_a_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_v6_00_a_bitacc_cmodel.so	Model shared object library
libIp_v_utils_v1_0_bitacc_cmodel.so	Utilities shared object library
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Precompiled yCrCb-to RGB converter shared object file for lin64 compilation
libstlport.so.5.1	STL library, referenced by libIp_v_scaler_v6_00_a_bitacc_cmodel.so
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_v6_00_a_bitacc_cmodel.so	Model shared object library
libIp_v_utils_v1_0_bitacc_cmodel.so	Utilities shared object library
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Precompiled yCrCb-to RGB converter shared object file for lin compilation
libstlport.so.5.1	STL library, referenced by libIp_v_scaler_v6_00_a_bitacc_cmodel.so
run_bitacc_cmodel	32-bit Linux fixed configuration executable
./nt64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.

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

File Name	Contents
libIp_v_scaler_v6_00_a_bitacc_cmodel.lib	Precompiled library file for 64-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.lib	Precompiled utilities 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
stlport.5.1.dll	STL library
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_v6_00_a_bitacc_cmodel.lib	Precompiled library file for 32-bit Windows platforms compilation
libIp_v_utils_v1_0_bitacc_cmodel.lib	Precompiled utilities 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
stlport.5.1.dll	STL library
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 following software shown in Table E-2.

Table E-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_v6_00_a_bitacc_simulate(
struct xilinx_ip_v_scaler_v6_00_a_generics generics,
struct xilinx_ip_v_scaler_v6_00_a_inputs inputs,
struct xilinx_ip_v_scaler_v6_00_a_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_v6_00_a_generics scaler_generics;
struct xilinx_ip_v_scaler_v6_00_a_inputs scaler_inputs;
struct xilinx_ip_v_scaler_v6_00_a_outputs* scaler_outputs;
```

The declarations of these structures are in the v_scaler_v6_00_a_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 will be used in any scaling operation
   - max_coef_sets - maximum number of coefficient sets that will be stored in the hardware (and delivered by the C-model in a .coe file)
   - Separate_YC_Coefs
     - 0: Y and C filter operations will use common coefficients
     - 1: Y and C filter operations will use separate coefficients
   - Separate_HV_Coefs
     - 0: H and V filter operations will use common coefficients
     - 1: H and V filter operations will 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 106.
   - 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 106.

   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 109 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 E-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 may 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 E-4.
4. Data is in 16 bit unsigned integer format accessed as data[plane][frame][row][col].

Table E-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 will 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 E-4, page 107. 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 cfg_file
cfg_file: path/name of the input  config file
```

Config File

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

- Input YUV filename
- Output YUV Filename
- Number of frames to be scaled
- Number of scaler taps and phases
- Input and output frame sizes
- Various other options. More details about the options can be found in Chapter 4, Designing with the Core.

An example of a config file is given in the delivered zip file. The text is shown below:

```
# ################################################
# scaler.cfg: Scaler model example config file
# ################################################
# Note: In this file, ALL inactive lines must be preceded by "# "
# (including 1 whitespace before any subsequent character).
#
# ################################################
# Syntax:
#     InputFile <file.yuv> <Input file HSize> <Input file VSize>
#
InputFile ./video_in.yuv 1280 720
#
# Note: Currently, only 4:2:2 YUV formats are accepted by this program.
#
# ################################################
# Optionally generate output YUV file.
# Syntax:
#     OutputFile <file.yuv>
#
# EnableYUVOutputGeneration 0 switches this option OFF
# EnableYUVOutputGeneration 1 switches this option ON
# IMPORTANT: GENERATED YUV OUTPUT FILES MUST BE OF A CONSISTANT RESOLUTION.
# OTHERWISE XILINX RECOMMENDS SETTING EnableYUVOutputGeneration to 0
#
EnableYUVOutputGeneration 1
OutputFile ./video_out.yuv
#
# Note: Currently, only 4:2:2 YUV formats are generated by this program.
#
# ################################################
#
# a. frames                     - Number of frames to be read/processed
# b. aperture_start_pixel       - Cropping left edge
# c. aperture_end_pixel         - Cropping right edge
# d. aperture_start_line        - Cropping top edge
# e. aperture_end_line          - Cropping bottom edge
# f. output_h_size              - Desired horizontal size of output Luma rectangle
```

```
# g. output_v_size               - Desired horizontal size of output Luma rectangle
# h. num_h_taps                  - Number of taps in the Y/C horizontal filter
# i. num_v_taps                  - Number of taps in the Y/C vertical  filter
# j. num_h_phases                - Number of phases desired in the current Y/C
horizontal filter coefficients
# k. num_v_phases             - Number of phases desired in the current Y/C vertical
filter coefficients
# l. Separate_YC_Coefs           - 1=Separate; 0 = Shared
# m. Separate_HV_Coefs.          - 1=Separate; 0 = Shared. IMPORTANT:
Separate_HV_Coefs may only be set to 0 if num_h_taps==num_v_taps and
num_h_phases==num_v_phases.
# n. Enable ramp override.       - 1=ramp, 0=video
# o. Enable user-defined coefs   - 1=enabled; 0=disabled
# p. User coefficient file.      - Provide path/filename of user coefficient file
#
# Each line represents one test.
# Each test will be appended upon the previous one.
# Within each test, for multiple frame tests, each frame will be appended upon the
previous frame.
#
# ################################################
#
# Test information is written to the Repo directory (defined below).
# Please create this directory manually.
# If user coefficients are defined, place the .coe file in this location.
RepoDir .\test
# a     b     c     d    e      f      g    h    i    j    k    l    m    n    o    p
  5     0    1279   0   719    1280   720   8    8    4    4    1    1    0    0
user_coefs.coe
#
# Keep this comment on last line.

Note that it contains the input and output .yuv file names, and the individual
parameter settings. For the input file case, the resolution is given after the file
name
Every line beginning with '# ' is ignored - beware that this requires a whitespace
after the '#'.
```

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_v6_00_a_bitacc_cmodel -Wl,-rpath,.

gcc -m32 -x c++ ./run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_scaler_v6_00_a_bitacc_cmodel -Wl,-rpath,.
```

Windows (32 or 64-bit)

Compile the precompiled library `v_scaler_v3_0_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_v3_0_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_v3_0_bitacc_cmodel.h` to "Header Files" folder of the project
- `v_ycrcb2rgb_v3_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 in order 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.
  - May 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.
  - May be viewed in a YUV player.

Appendix F

Additional Resources

Xilinx Resources

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

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

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

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

- UG761 AXI Reference Guide
- AXI - Video and Imaging Documentation
- For more information on Lanczos resampling, refer to this Wikipedia page: http://en.wikipedia.org/wiki/Lanczos_resampling

Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The Video Scaler core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator. The CORE Generator system is shipped with Xilinx ISE Design Suite development software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the Video Scaler product page at: http://www.xilinx.com/products/intellectual-property/EF-DI-VID-SCALER.htm.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release of Product Guide, replacing DS840 and UG805.
04/24/2012	2.0	Updated for core version. Added Zynq-7000 devices, added AXI4-Stream interfaces, deprecated GPP interface.

Notice of Disclaimer

been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.