LogiCORE IP Video Scaler v5.0

Product Guide

PG009 October 19, 2011

XILINX®

Table of Contents

Chapter 1: Overview

Chapter 2: Core Interfaces and Register Space

Chapter 3: Customizing and Generating the Core

Chapter 4: Designing with the Core

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

LogiCORE IP Video Scaler v5.0

Introduction

The Xilinx LogiCORE™ IP Video Scaler 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. System design is made easier through support of both streaming-video and frame buffer-based interfaces. This core is designed to connect via an AXI4-Lite interface.

The Video Scaler core allows the filter coefficients to be updated dynamically. It supports RGB/4:4:4, YUV4:2:2, and YUV4:2:0 color formats for 8, 10, or 12-bit video. The architecture takes advantage of the high-performance XtremeDSP™ slices.

The Video Scaler core may be fed with live video but also supports the option of a memory interface.

CORE Generator™ technology generates the core as either an AXI EDK pCore, a standalone netlist for a General Purpose Processor (GPP) or as a Constant (Fixed Mode) netlist. When generated as an EDK pCore, the processor interface is AXI4-Lite compliant.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex-7, Kintex-7 Virtex-6, Spartan-6
Supported User Interfaces	General Purpose Processor (GPP), EDK pCore AXI4-Lite, Constant
Resources	See Table 1-7 through Table 1-10.
Provided with Core	
Design Files	Netlist for GPP and Constant Interfaces Encrypted Source Code for EDK pCore
Example Design	Not Provided
Test Bench	VHDL[2]
Constraints File	Not Provided
Simulation Model	CORE Generator™ VHDL/Verilog Structural Models Bit-Accurate C Model[2]
Tested Design Tools	
Design Entry Tools	CORE Generator, Platform Studio (XPS)
Simulation[3]	Mentor Graphics ModelSim, Xilinx ISim
Synthesis Tools[3]	Xilinx Synthesis Technology (XST) 13.3
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete listing of supported devices, see the release notes for this core.
2. Test bench and C model available on the Video Scaler product page.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Chapter 1

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 user 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).

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

Standards Compliance

The Video Scaler core is compliant with the AXI4-Lite interconnect standard as defined in UG761, *AXI Reference Guide*.

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, the user 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 one of two interfaces selected in the configuration GUIs. The first option is to use the XSVI live video interface. Typically this should be used in a system where data is fed from a live source - the XSVI signals may be directly mapped to video signals that are found in most raster-scan video formats (HBlank, VBlank, Active Video). This interface includes no backwards flow control signalling.

The second option is the AXI4-Stream option. This option includes standard back-pressure signalling found in AXI4-Stream. This interconnect format is used for connecting to other IP blocks that support AX4-Stream. Largely, this interface is used when the source image originates from an external frame buffer in DDR memory.

The decision to use one interface type over another is dependent upon many factors, and is further exploited in Performance, page 10.

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, select Constant Mode implementation in the CORE Generator tool GUI. Scaling parameters can all be fixed in the CORE Generator tool GUI.

In some cases, dynamic user control is required for changing various settings on a frame-by-frame basis. For these cases, the processor interface is selected during generation. The first option is an EDK pCore interface that can be easily incorporated into an EDK project. Dynamic control of most scaler parameters is possible using AXI4-Lite. The second option is a General Purpose Processor interface. This option exposes the core's registers to the user. These exposed registers can be wrapped in an interface that is compliant with the systems processor.

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

Nomenclature

Table 1-1 defines terms used in this document.

Table 1-1: **Nomenclature**

Term	Definition
Scaler Aperture	The input data rectangle used to create the output data rectangle.
Filter Aperture	The group of contributory data used in a filter to generate one particular output. The number of elements in this group of data is the number of taps. We define the filter aperture size using the `num_h_taps` and `num_v_taps` parameters.
Coefficient Phase	Each tap is multiplied by a coefficient to make its contribution to the output pixel. The coefficients used are selected from a "phase" of `num_x_taps` coefficients. The phase selection is dependent upon the position of the output pixel in the input sampling grid space. For each dimension of the filter, each coefficient phase consists of `num_h_taps` or `num_v_taps` coefficients.
Channel	For scaler purposes, all monochromatic video streams, for example Y, Cb, Cr, R, G, B, are all considered separate channels.
Coefficient Phase Index	An index given that selects the coefficient phase applied to one filter aperture in a FIR. For an n-tap filter, this index points to n coefficients.

Table 1-1: **Nomenclature *(Cont'd)***

Term	Definition
Coefficient Bank	A group of coefficients that will be applied to one video component (Y or C) in one dimension (H or V) for a conversion of one frame. It includes all phases. For an n-tap, m-phase filter, a coefficient bank comprises nxm values. Each tap may be multiplied by any one of m coefficients assigned to it, selected by the phase index, which is applied to all taps.
Coefficient Set	A group of four coefficient banks (VY, VC, HY, HC). One full set should be written into the scaler before use.

Licensing

The Video Scaler provides three licensing options. 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 your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically-generated HDL structural model.)

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.

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 (ceasing to function), at which time it can be reactivated by reconfiguring the device.

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

Obtaining Your License Key

This section contains information about obtaining a simulation, full system hardware, and full license keys.

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.

Full System Hardware Evaluation License

1. Navigate to the product page for this core:

   www.xilinx.com/products/ipcenter/EF-DI-VID-SCALER.htm
2. Click Evaluate.
3. Follow the instructions to install the required Xilinx ISE software and IP Service Packs.

Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

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.

Performance

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

Maximum Frequency

The following are typical clock frequencies for the target devices:

- Virtex®-7 (-1) FPGA: 225 MHz
- Kintex™-7 (-1) FPGA: 225 MHz
- Virtex-6 (-1) FPGA: 225 MHz
- Spartan®-6 (-2) FPGA: 150 MHz

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_clk`, `clk` and `video_out_clk`.

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 1-2 through Table 1-5 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 1-2: **Performance Data for Virtex-7 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	FMAX (MHz)/ Speed Grade		
							-1	-2	-3
4x4	4	1	YC4:2:2	Live	8	2048x2048	263	329	365
12x12	4	1	YC4:2:2	Live	8	2048x2048	293	335	384
4x4	64	1	YC4:2:2	Live	8	2048x2048	283	335	375
4x4	4	1	YC4:2:2	Live	10	2048x2048	283	303	345
4x4	4	1	YC4:2:2	Live	8	512x512	273	345	355
4x4	4	2	YC4:2:2	Live	8	2048x2048	293	329	355
4x4	4	3	YC4:4:4/RGB	Live	12	2048x2048	273	329	329
4x4	4	1	YC4:2:0	Live	8	2048x2048	273	329	365
4x4	4	1	YC4:2:2	Memory	8	2048x2048	263	283	323

Table 1-3: **Performance Data for Kintex-7 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	FMAX (MHz)/ Speed Grade		
							-1	-2	-3
4x4	4	1	YC4:2:2	Live	8	2048x2048	272	320	386
12x12	4	1	YC4:2:2	Live	8	2048x2048	272	328	377
4x4	64	1	YC4:2:2	Live	8	2048x2048	272	337	328
4x4	4	1	YC4:2:2	Live	10	2048x2048	272	328	328
4x4	4	1	YC4:2:2	Live	8	512x512	295	328	361
4x4	4	2	YC4:2:2	Live	8	2048x2048	288	353	386
4x4	4	3	YC4:4:4/RGB	Live	12	2048x2048	272	328	386
4x4	4	1	YC4:2:0	Live	8	2048x2048	263	337	377
4x4	4	1	YC4:2:2	Memory	8	2048x2048	288	345	361

Table 1-4: **Performance Data for Virtex-6 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	FMAX (MHz)/ Speed Grade		
							-1	-2	-3
4x4	4	1	YC4:2:2	Live	8	2048x2048	272	320	386
12x12	4	1	YC4:2:2	Live	8	2048x2048	272	328	377
4x4	64	1	YC4:2:2	Live	8	2048x2048	272	337	328

Table 1-4: **Performance Data for Virtex-6 Devices *(Cont'd)***

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	F_{MAX} (MHz)/ Speed Grade		
							-1	-2	-3
4x4	4	1	YC4:2:2	Live	10	2048x2048	272	328	328
4x4	4	1	YC4:2:2	Live	8	512x512	295	328	361
4x4	4	2	YC4:2:2	Live	8	2048x2048	288	353	386
4x4	4	3	YC4:4:4/RGB	Live	12	2048x2048	272	328	386
4x4	4	1	YC4:2:0	Live	8	2048x2048	263	337	377
4x4	4	1	YC4:2:2	Memory	8	2048x2048	288	345	361

Table 1-5: **Performance Data for Spartan-6 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	F_{MAX} (MHz)/ Speed Grade		
							-1	-2	-3
4x4	4	1	YC4:2:2	Live	8	2048x2048	230	267	251
12x12	4	1	YC4:2:2	Live	8	2048x2048	175	205	195
4x4	64	1	YC4:2:2	Live	8	2048x2048	216	236	236
4x4	4	1	YC4:2:2	Live	10	2048x2048	210	257	216
4x4	4	1	YC4:2:2	Live	8	512x512	236	236	257
4x4	4	2	YC4:2:2	Live	8	2048x2048	205	195	205
4x4	4	3	YC4:4:4/RGB	Live	12	2048x2048	195	225	210
4x4	4	1	YC4:2:0	Live	8	2048x2048	236	236	236
4x4	4	1	YC4:2:2	Memory	8	2048x2048	225	251	230

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 with Live Video Data using the XSVI interface, throughput is limited to the frame rate at which the input video data arrives.

In contrast, when running from memory using the AXI4-Stream interface, there is much more flexibility to feed data into the scaler as needed. When in this free-flowing mode of operation, 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 `clk`, `video_in_clk`, `video_out_clk`: depends upon the selected device)

These factors may contribute to decisions made for configuring the scaler and its supporting system. For example, the user 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 frequency required to make the intended conversions possible.

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

- `video_in_clk`: 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 after it has received another pulse on its `vysnc_in` pin. So, faster clock rates are safe.

- `video_out_clk`: Output Clock

  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.

- `clk`: Core Clock

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

Definitions:

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'clk The 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.

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.
  - The user may not hold off the input stream.
  - The system must be able to cope with the constant flow of video data.
- **Memory mode:** The user may control the input feed using back-pressure/handshaking by implementing an input frame buffer.

Live Video Mode, page 15 and Memory Mode, page 20 detail some example cases that illustrate how to calculate the clock frequencies required to sustain the throughput required for given usage scenarios.

Live Video Mode

If no input frame buffer is used, and the timing of the input video format drives the scaler, then the number of '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))+
OverHeadMult*ProcessingOverheadPerComponent
```

The CompsPerEngine and OverHeadMult values can be extracted from Table 1-6.

Table 1-6: **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.

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)+Proces
singOverheadPerComponent
```

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

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

- **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 'clk' frequency required to achieve your goals according to this calculation:

```
MinF'clk' = FLineIn x MaxClksTakenPerVAperture
```

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

```
ClksAvailablePerLine = F'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'clk' = 33750*3990 = 134.66 MHz
video_in_clk = Frequency defined by live-mode input pixel clock.
Typically 74.25 MHz.
video_out_clk = Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.8 MHz
```

Shrink-factor inputs:

hsf=2^{20} x (1/1.0) = 0x100000

vsf=2^{20} x (1/1.0) = 0x100000

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 1-6.

Example 2: Up-scaling 640x480 60 Hz YC4:2:2 to 800x600
Assuming 30 kHz line rate
Vertical scale ratio = 1.25
Horizontal scale ratio = 1.25
FLineIn = 30000
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*800 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(600/480) = 2
MaxClksTakenPerVAperture = 1750 * 2= 3500
MinF'clk' = 30000*3500 = 105 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
Typically 74.25 MHz.
video_out_clk = Delivery of 1 frame of 800x600 pixels in 1/60 s:
Fmin = 31.7 MHz
```

Shrink-factor inputs:

hsf=2^{20} x (1/1.25) = 0x0CCCCC

vsf=2^{20} x (1/1.25) = 0x0CCCCC

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 30 kHz line rate
Vertical scale ratio = 3.0
Horizontal scale ratio = 2.2
FLineIn = 30000
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*1920 + 150 (approximately)
MaxVHoldsPerInputAperture =round_up(1080/480) = 3
MaxClksTakenPerVAperture = 3990 * 3 = 11970
MinF'clk' = 30000*11970 = 359.1 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
video_out_clk = Delivery of 1 frame of 1920x1080 pixels in 1/60 s:
Fmin = 136.9 MHz
```

Shrink-factor inputs:

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

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 30 kHz line rate
Vertical scale ratio = 3.0
Horizontal scale ratio = 2.2
FLineIn = 30000
Dual-engine implementation

```
CyclesPerOutputLine = 1*1920 + 2*50 (approximately)
MaxVHoldsPerInputAperture =round_up(1080/480) = 3
MaxClksTakenPerVAperture = 2020 * 3 = 6060
MinF'clk' = 30000*6060 = 181.8 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
video_out_clk = Delivery of 1 frame of 1920x1080 pixels in 1/60 s:
Fmin = 136.9 MHz
```

Shrink-factor inputs:

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

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 30 kHz line rate
Vertical scale ratio = 0.8
Horizontal scale ratio = 0.8
FLineIn = 30000
Single-engine implementation

```
CyclesRequiredPerOutputLine = 2*800 + 150 (approximately)
MaxVHoldsPerInputAperture = round_up(480/600) = 1
MaxClksTakenPerVAperture = 1750 * 1= 1750
MinF'clk' = 30000*1750 = 52.5 MHz
```

Shrink-factor inputs:

```
hsf=2^20 x (1/0.8) = 0x140000
vsf=2^20 x (1/0.8) = 0x140000
```

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'clk' = 67500*3990 = 269.32 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
video_out_clk = Delivery of 1 frame of 640x480 pixels in 1/60 s:
Fmin = 20.3 MHz
```

Shrink-factor inputs:

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

When using a single-engine, this conversion will not work with or without frame buffers (see Memory Mode, page 20) 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'clk' = 67500*2020 = 136.35 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
video_out_clk = Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.8 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
Typically 148.5 MHz.
video_out_clk = Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.8 MHz
```

Shrink-factor inputs:

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

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'clk' = 45000*2710 = 121.95 MHz
video_in_clk = frequency defined by live-mode input pixel clock.
Typically 74.25 MHz.
video_out_clk = Delivery of 1 frame of 640x480 pixels in 1/60 s:
Fmin = 20.3 MHz
```

Shrink-factor inputs:

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

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'clk' = 45000*3990 = 179.55 MHz
video_in_clk = Frequency defined by live-mode input pixel clock.
Typically 74.25 MHz.
video_out_clk = Delivery of 1 field of 1920x540 pixels in 1/60 s:
Fmin = 68.4 MHz
```

Shrink-factor inputs:

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

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'clk' = 45000*4040 = 181.8 MHz
video_in_clk = Delivery of 1 frame of 1280x720 pixels in 1/60 s:
Fmin = 60.8 MHz
video_out_clk = Delivery of 1 frame of 1920x1080 pixels in 1/60 s:
Fmin = 136.85 MHz
```

Shrink-factor inputs:

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

This conversion will work in Virtex and Kintex devices, and in higher speed grade Spartan devices.

Memory Mode

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 `clk` frequency according to this calculation:

```
MinF'clk' = FFrameIn x CyclesPerOutputFrame
```

Example 10: 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'clk' = 60 x 2164600 = **129.87 MHz**

video_in_clk = Delivery of 1 frame of 1280x720 pixels in 1/60 s: Fmin = **60.8 MHz**

video_out_clk = Delivery of 1 field of 1920x540 pixels in 1/60 s: Fmin = **68.4 MHz**

Shrink-factor inputs:

hsf=2^{20} x (1/1.5) = 0x0AAAAA

vsf=2^{20} x (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'clk' = 60 x 2191600= **131.5 MHz**

video_in_clk - Delivery of 1 frame of 1280x720 pixels in 1/60 s: Fmin = **136.85 MHz**

video_out_clk - Delivery of 1 frame of 1920x1080 pixels in 1/60 s: Fmin = **136.85 MHz**

Shrink-factor inputs:

hsf=2^{20} x (1/1.5) = 0x0AAAAA

vsf=2^{20} x (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 1-7 through Table 1-10 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 using pCore interface, add approximately 600 FFs and 850 LUTs (all families).

Table 1-7: **Resource Usage for Virtex-7 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM36/18	DSP48E1
4x4	4	1	YC4:2:2	Live	8	2048x2048	1884	1759	10/1	12
12x12	4	1	YC4:2:2	Live	8	2048x2048	2231	2788	21/2	28

Table 1-7: **Resource Usage for Virtex-7 Devices *(Cont'd)***

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM36/18	DSP48E1
4x4	64	1	YC4:2:2	Live	8	2048x2048	1916	1795	10/1	12
4x4	4	1	YC4:2:2	Live	10	2048x2048	1979	1834	11/6	12
4x4	4	1	YC4:2:2	Live	8	512x512	1841	1731	3/7	12
4x4	4	2	YC4:2:2	Live	8	2048x2048	2449	2687	7/7	24
4x4	4	3	YC4:4:4/ RGB	Live	12	2048x2048	1783	2094	9/10	28
4x4	4	1	YC4:2:0	Live	8	2048x2048	2160	1955	10/1	13
4x4	4	1	YC4:2:2	Memory	8	2048x2048	2026	1836	10/1	12

Table 1-8: **Resource Usage for Kintex-7 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM36/18	DSP48E1
4x4	4	1	YC4:2:2	Live	8	2048x2048	1839	1759	10/1	12
12x12	4	1	YC4:2:2	Live	8	2048x2048	2182	2778	21/2	28
4x4	64	1	YC4:2:2	Live	8	2048x2048	1878	1795	10/1	12
4x4	4	1	YC4:2:2	Live	10	2048x2048	1937	1834	11/6	12
4x4	4	1	YC4:2:2	Live	8	512x512	1822	1731	3/7	12
4x4	4	2	YC4:2:2	Live	8	2048x2048	2414	2687	7/7	24
4x4	4	3	YC4:4:4/ RGB	Live	12	2048x2048	1716	2094	9/10	28
4x4	4	1	YC4:2:0	Live	8	2048x2048	2116	1955	10/1	13
4x4	4	1	YC4:2:2	Memory	8	2048x2048	1992	1856	10/1	12

Table 1-9: **Resource Usage for Virtex-6 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM36/18	DSP48E1
4x4	4	1	YC4:2:2	Live	8	2048x2048	1205	1758	10/1	12
12x12	4	1	YC4:2:2	Live	8	2048x2048	1695	2787	21/2	28
4x4	64	1	YC4:2:2	Live	8	2048x2048	1241	1794	10/1	12
4x4	4	1	YC4:2:2	Live	10	2048x2048	1273	1834	11/6	12
4x4	4	1	YC4:2:2	Live	8	512x512	1178	1731	3/7	12

Table 1-9: **Resource Usage for Virtex-6 Devices *(Cont'd)***

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM36/18	DSP48E1
4x4	4	2	YC4:2:2	Live	8	2048x2048	1594	2665	7/7	24
4x4	4	3	YC4:4:4/ RGB	Live	12	2048x2048	1109	2083	9/10	28
4x4	4	1	YC4:2:0	Live	8	2048x2048	1394	1955	10/1	13
4x4	4	1	YC4:2:2	Memory	8	2048x2048	1310	1830	10/1	12

Table 1-10: **Resource Usage for Spartan-6 Devices**

Filter (HxV taps)	Max Phases	Engines	Chroma Format	Input Video Interface	Video Bitwidth	Max I/O Image Size (Pix x Lines)	LUTs	FFs	BRAM16/8	DSP48E1
4x4	4	1	YC4:2:2	Live	8	2048x2048	1247	1768	18/0	12
12x12	4	1	YC4:2:2	Live	8	2048x2048	1711	2796	37/1	28
4x4	64	1	YC4:2:2	Live	8	2048x2048	1259	1868	20/0	12
4x4	4	1	YC4:2:2	Live	10	2048x2048	1311	1844	25/0	12
4x4	4	1	YC4:2:2	Live	8	512x512	1173	1741	5/5	12
4x4	4	2	YC4:2:2	Live	8	2048x2048	1644	2697	18/0	24
4x4	4	3	YC4:4:4/ RGB	Live	12	2048x2048	1255	2077	24/0	28
4x4	4	1	YC4:2:0	Live	8	2048x2048	1419	1963	18/0	13
4x4	4	1	YC4:2:2	Memory	8	2048x2048	1337	1846	18/0	12

Chapter 2

Core Interfaces and Register Space

This chapter provides detailed descriptions for each interface. In addition, detailed information about configuration and control registers is included.

Port Descriptions

Core Interfaces

Control Interfaces

Processor interfaces provide the ability to dynamically control the parameters within the core. The Video Scaler core supports three processor interface options: Constant, ACI4-Lite pCore, or General Purpose Processor.

Constant Interface

The designer may elect to set up the Video Scaler in a fixed configuration. The settings, applied using the CORE Generator GUI, are not dynamic. They may not be changed during run-time. This applies to all control values for the core. When using a Constant mode implementation of the core, no processor interface is implemented.

AXI4-Lite pCore Interface

The designer may select AXI4-Lite option on the Video Scaler core if it is to be used in an EDK-based embedded system. The AXI4-Lite pCore interface creates a hardware peripheral that can be easily added to an AXI4-based EDK Project. When the core is connected to the system's AXI4-Lite interconnect, the system processor can easily access the core's registers and control the operation of the core.

General Purpose Processor (GPP) Interface

The General Purpose Processor interface option is selected when the core is to be used in a system that does not include an AXI4-compliant system processor. The General Purpose Processor interface exposes all of the core's control and status signals. This allows the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor.

Data Interface

Video data input may be fed into the Video Scaler core using either the XSVI interface or the AXI4-Stream interface.

The decision of which interface to use is dependent on the need to provide the input video data from a frame-buffer. Generally when upscaling vertically, buffering of the input data is required, although many factors, including clock-rate and worst-case scale-factors also affect this decision. See Throughput in Chapter 1 for more information on how to determine which interface is needed.

XSVI Input Interface

By selecting Live Mode in the CORE Generator GUI, the designer elects to supply video data into the core using an XSVI interface. This option is appropriate when maintaining compatibility with traditional video formats. Typically, this may be the case when feeding live video data into the core (not from an external memory or any AXI4-Stream component).

This interface does not include provision for back-pressure (although a non-XSVI back-pressure signal, `line_request`, is provided for optional use). Use of this interface when reading the video data from an external memory interface (for example, via AXI-VDMA) is not recommended.

AXI4-Stream Input Interface

By selecting Memory Mode in the CORE Generator GUI, the designer elects to supply video data into the core using an AXI4-Stream interface.

Xilinx recommends using this interface when supplying data from a frame-buffer (in external memory). AXI4-Stream includes provision for back-pressure as part of the AXI4-Stream standard.

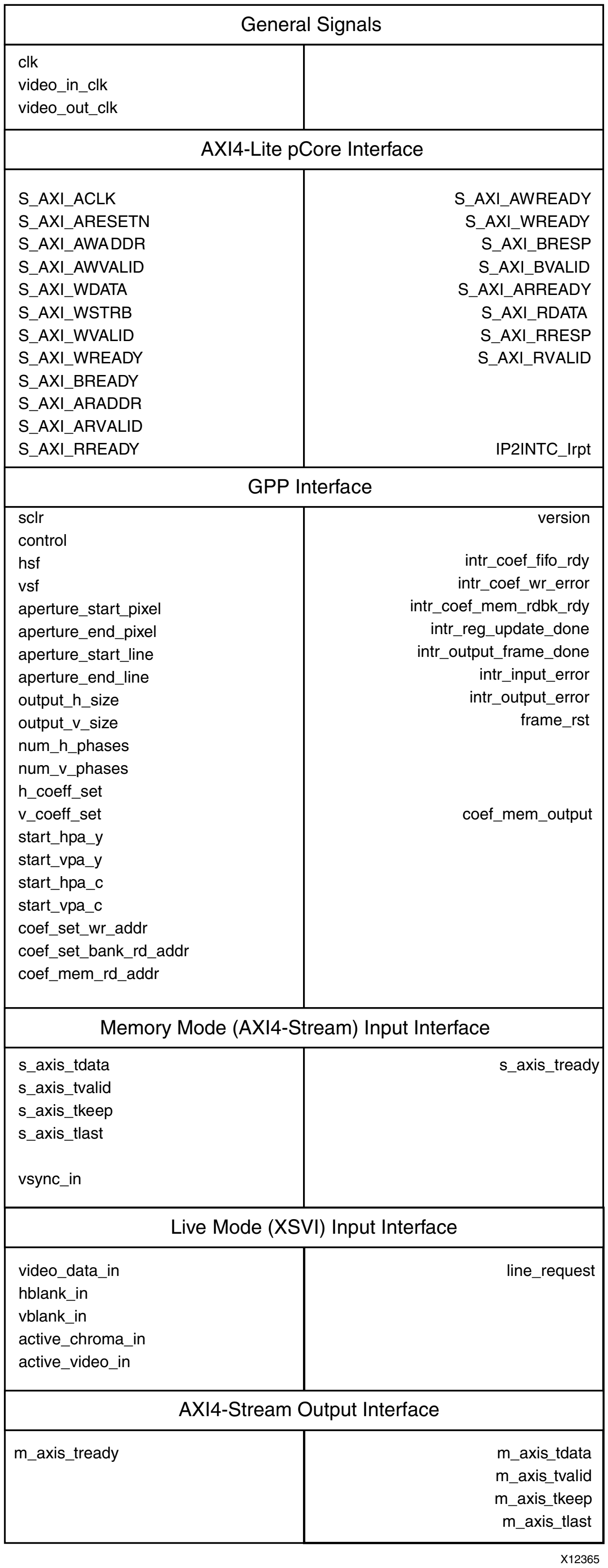
This interface option should also be selected when driving video data into the Video Scaler from any other AXI4-Stream-compliant IP block.

AXI4-Stream Output Interface

Video data emerges from the output of the core via an AXI4-Stream interface (XSVI is not an option for the output data interface).

Interface Diagram

Figure 2-1 includes all possible interface signals. The two processor-interface options (AXI4-Lite and GPP) are illustrated on the same diagram. These two interface options are mutually exclusive, and neither exist when Constant mode has been selected.



General Signals	
clk	
video_in_clk	
video_out_clk	

AXI4-Lite pCore Interface	
S_AXI_ACLK	S_AXI_AWREADY
S_AXI_ARESETN	S_AXI_WREADY
S_AXI_AWADDR	S_AXI_BRESP
S_AXI_AWVALID	S_AXI_BVALID
S_AXI_WDATA	S_AXI_ARREADY
S_AXI_WSTRB	S_AXI_RDATA
S_AXI_WVALID	S_AXI_RRESP
S_AXI_WREADY	S_AXI_RVALID
S_AXI_BREADY	
S_AXI_ARADDR	
S_AXI_ARVALID	
S_AXI_RREADY	IP2INTC_Irpt

GPP Interface	
sclr	version
control	
hsf	intr_coef_fifo_rdy
vsf	intr_coef_wr_error
aperture_start_pixel	intr_coef_mem_rdbk_rdy
aperture_end_pixel	intr_reg_update_done
aperture_start_line	intr_output_frame_done
aperture_end_line	intr_input_error
output_h_size	intr_output_error
output_v_size	frame_rst
num_h_phases	
num_v_phases	
h_coeff_set	
v_coeff_set	coef_mem_output
start_hpa_y	
start_vpa_y	
start_hpa_c	
start_vpa_c	
coef_set_wr_addr	
coef_set_bank_rd_addr	
coef_mem_rd_addr	

Memory Mode (AXI4-Stream) Input Interface	
s_axis_tdata	s_axis_tready
s_axis_tvalid	
s_axis_tkeep	
s_axis_tlast	
vsync_in	

Live Mode (XSVI) Input Interface	
video_data_in	line_request
hblank_in	
vblank_in	
active_chroma_in	
active_video_in	

AXI4-Stream Output Interface	
m_axis_tready	m_axis_tdata
	m_axis_tvalid
	m_axis_tkeep
	m_axis_tlast

Figure 2-1: **I/O Diagram**

Core Signal Names and Descriptions

General Signals

Regardless of the type of processor interface or video I/O interface used by the core, the Video Scaler uses the signaling shown in Table 2-1.

Table 2-1: **General Signals**

Signal Name	Direction	Width	Description
clk	In	1	Core clock
video_in_clk	In	1	Input pixel-rate clock
video_out_clk	In	1	Output pixel-rate clock

Control Interface Signals

Processor interfaces provide the system designer with the ability to dynamically control core parameters. The Video Scaler core supports two processor interface options:

- AXI4-Lite pCore Interface: As described in Table 2-2
- General Purpose Processor Interface: As described in Table 2-3

Table 2-2: **AXI4-Lite Control Bus Signals**

Signal Name	Direction	Width	Description
S_AXI_ACLK	In	1	AXI Clock
S_AXI_ARESETN	In	1	AXI Reset, active Low
IP2INTC_Irpt	Out	1	Interrupt request output
S_AXI_AWADDR	In	C_S_AXI_ ADDR_WIDTH	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.
S_AXI_AWVALID	In	1	AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available. 1 = Write address is valid. 0 = Write address is not valid.
S_AXI_AWREADY	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address. 1 = Ready to accept address. 0 = Not ready to accept address.
S_AXI_WDATA	In	C_S_AXI_ DATA_WIDTH	AXI4-Lite Write Data Bus
S_AXI_WSTRB	In	C_S_AXI_ DATA_WIDTH/8	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.

Table 2-2: **AXI4-Lite Control Bus Signals *(Cont'd)***

Signal Name	Direction	Width	Description
S_AXI_WVALID	In	1	AXI4-Lite Write Data Channel Write Data Valid. This signal indicates that valid write data and strobes are available. 1 = Write data/strobes are valid. 0 = Write data/strobes are not valid.
S_AXI_WREADY	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data. 1 = Ready to accept data. 0 = Not ready to accept data.
S_AXI_BRESP(2)	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer. 00b = OKAY - Normal access has been successful. 01b = EXOKAY - Not supported. 10b = SLVERR - Error. 11b = DECERR - Not supported.
S_AXI_BVALID	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. 1 = Response is valid. 0 = Response is not valid.
S_AXI_BREADY	In	1	AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response. 1 = Ready to receive response. 0 = Not ready to receive response
S_AXI_ARADDR	In	C_S_AXI_ ADDR_WIDTH	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction.
S_AXI_ARVALID	In	1	AXI4-Lite Read Address Channel Read Address Valid. 1 = Read address is valid. 0 = Read address is not valid.
S_AXI_ARREADY	Out	1	AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address. 1 = Ready to accept address. 0 = Not ready to accept address.
S_AXI_RDATA	Out	C_S_AXI_ DATA_WIDTH	AXI4-Lite Read Data Bus

Table 2-2: **AXI4-Lite Control Bus Signals *(Cont'd)***

Signal Name	Direction	Width	Description
S_AXI_RRESP(2)	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer. 00b = OKAY - Normal access has been successful. 01b = EXOKAY - Not supported. 10b = SLVERR - Error. 11b = DECERR - Not supported.
S_AXI_RVALID	Out	1	AXI4-Lite Read Data Channel Read Data Valid. This signal indicates that the required read data is available and the read transfer can complete. 1 = Read data is valid. 0 = Read data is not valid.
S_AXI_RREADY	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data. 1 = Ready to accept data. 0 = Not ready to accept data.

Table 2-3: **GPP Signals**

Signal Name	Direction	Width	Description
hsf	Input	24	Horizontal Shrink Factor Format 24.20, Range 12.0 (0xC00000) to 1/12 (0x015555) ***Note:*** Conceptually, this input value is the reciprocal of the horizontal scale factor: • hsf > 1.0 for horizontal downscaling cases • hsf < 1.0 for horizontal upscaling cases For example, when upscaling 640 to 1024 pixels, hsf = 0.625 (0x0A0000).
vsf	Input	24	Veritcal Shrink Factor Format 24.20, Range 12.0 (0xC00000) to 1/12 (0x015555) ***Note:*** Note: Conceptually, this input value is the reciprocal of the vertical scale factor: • vsf > 1.0 for vertical downscaling cases • vsf < 1.0 for vertical upscaling cases For example, when downscaling 1080 to 720 lines, vsf = 1.5 (0x180000)

Table 2-3: **GPP Signals *(Cont'd)***

Signal Name	Direction	Width	Description
aperture_start_pixel	Input	13	Location of first subject pixel in input line, relative to first active pixel in that line ***Note:*** When chroma format is is YC4:2:2 or YC4:2:0, an even number must be specified for this value.
aperture_end_pixel	Input	13	Location of final subject pixel in input line, relative to first active pixel in that line.
aperture_start_line	Input	13	Location of first subject line in input image, relative to first active line in that image
aperture_end_line	Input	13	Location of final subject line in input image, relative to first active line in that image
output_h_size	Input	13	Desired width of output rectangle (pixels).
output_v_vize	Input	13	Desired height of output image (lines)
num_h_phases	Input	7	Number of phases of coefficients in current horizontal filter set
num_v_phases	Input	7	Number of phases of coefficients in current vertical filter set
h_coeff_set	Input	4	Active coefficient set to use in horizontal filter operation
v_coeff_set	Input	4	Active coefficient set to use in vertical filter operation
start_hpa_y	Input	21	Fractional value used to initialize horizontal accumulator at rectangle left edge for luma
start_vpa_y	Input	21	Fractional value used to initialize vertical accumulator at rectangle top edge for luma
start_hpa_c	Input	21	Fractional value used to initialize horizontal accumulator at rectangle left edge for chroma
start_vpa_c	Input	21	Fractional value used to initialize vertical accumulator at rectangle top edge for chroma
control	Input	32	General control register
version	Input	32	Core HW version register
intr_output_frame_done	Output	1	Issued once per complete output frame
intr_input_error	Output	1	Issued if active_video_in is asserted before the scaler is ready to receive a new line
intr_output_error	Output	1	Issued if frame period completes before full output frame has been delivered
intr_reg_update_done	Output	1	Issued during Vertical blanking when the register values have been transferred to the active registers

Table 2-3: **GPP Signals *(Cont'd)***

Signal Name	Direction	Width	Description
intr_coef_wr_error	Output	1	Issued if coefficient is written into coefficient FIFO when the FIFO is not ready
intr_coef_fifo_rdy	Output	1	Issued when the coefficient FIFO is ready to receive a coefficient for the current set; stays low once a full set has been written into FIFO; sent high during Vertical blanking
intr_coef_mem_rdbk_rdy	Output	1	Issued when the output coefficient read-back FIFO has been fully populated with a bank of coefficients. This is cleared when bit 3 of the control register (addr 0) is set low. It is set high 2 frame-periods after the bit 3 of the control register has been set high, allowing time for the output coefficient FIFO to become populated with the requested bank.
frame_rst	Output	1	General purpose reset signal asserted for one line period during vertical blanking
coef_wr_en	Input	1	Write-enable for coefficient – active high
coef_data_in	Input	32	Coefficient input bus
coef_set_wr_addr	Input	4	Coefficient memory write address
coef_set_bank_rd_addr	Input	16	• bits[1:0]: Bank select: 00=HY; 01=HC; 10=VY; 11 =VC • bits[15:8]: Set select
coef_mem_rd_addr	Input	16	• bits[3:0]: Tap select • bits[15:8]: Phase select
coef_mem_output	Output	32	Coefficient output

Data Interface Signals

The Video Scaler core accepts video data via either of:

- An XSVI interface: As described in Table 2-4.
- AXI4-Stream interface: As described in Table 2-5.

The core output is delivered through another AXI4-Stream interface. See Table 2-6 AXI4-Stream Output Interface Signals.

Table 2-4: **Live-Mode (XSVI) Input Interface Signals**

Signal Name	Direction	Width	Description
active_video_in	Input	1	Write-enable to input data FIFO.
video_data_in	Input	Between 16 and 36, dependent on data width and chroma format	Video data input. When 4:2:2 or 4:2:0: • bits[(data_width-1):0]: Luma • bits[(2*data_width-1):data_width]: Chroma When 4:4:4: • bits[(data_width-1):0]: for example, R • bits[(2*data_width-1):data_width]: for example, G • bits[(3*data_width-1):2*data_width] : for example, B For 4:4:4, the 3 channels are treated identically.
vblank_in	Input	1	Vertical synchronization pulse . Must be High during V blanking period.
hblank_in	Input	1	Horizontal synchronization pulse. Must be High during H blanking period.
active_chroma_in	Input	1	Chroma input-line validation. • 4:4:4 and 4:2:2 operation: Set to '1' permanently. • 4:2:0 operation: Set to '1' for active chroma lines only.
line_request	Output	1	1 = Input data FIFO may accept another input line.

Table 2-5: **Memory-Mode (AXI4-Stream) Input Interface Signals**

Signal Name	Direction	Width	Description
s_axis_tdata	Input	16, 32, 64, 128, defined by S_AXIS_TDATA _WIDTH parameter	AXI4-Stream Video Data Input When 4:2:2 or 4:2:0: • bits[(data_width-1):0]: Luma • bits[(2*data_width-1):data_width]: Chroma When 4:4:4: • bits[(data_width-1):0]: for example, R • bits[(2*data_width-1):data_width]: for example, G • bits[(3*data_width-1):2*data_width]: for example, B For 4:4:4, the 3 channels are treated identically.
s_axis_tvalid	Input	1	AXI4-Stream tValid input signal. Indicates valid data on the s_axis_tdata bus.
s_axis_tlast	Input	1	AXI4-Stream tLast input signal. Coincides with the final pixel in a line on s_axis_tdata.
s_axis_tReady	Output	1	AXI4-Stream tReady output signal. High value indicates core is ready to receive data.
s_axis_tkeep	Input	2, 4, 8, 16, defined by S_AXIS_TDATA _WIDTH parameter	AXI4-Stream tKeep signal. Input should be driven to all ‘1’s.
vsync_in	Input		Vertical sync signal indicating that the next line at the input to the scaler will be the top line in the input frame

Table 2-6: **AXI4-Stream Output Interface Signals**

Signal Name	Direction	Width	Description
m_axis_tdata	Output	16, 32, 64, 128, defined by M_AXIS_TDATA _WIDTH parameter	Video data output. When 4:2:2 or 4:2:0: • bits[(data_width-1):0]: Luma • bits[(2*data_width-1):data_width]: Chroma When 4:4:4: • bits[(data_width-1):0]: for example, R • bits[(2*data_width-1):data_width]: for example, G • bits[(3*data_width-1):2*data_width]: for example, B For 4:4:4, the 3 channels are treated identically.
m_axis_tvalid	Output	1	AXI4-Stream tValid output signal. Indicates valid data on the m_axis_tdata bus.
m_axis_tlast	Output	1	AXI4-Stream tLast output signal. Coincides with the final pixel in a line on m_axis_tdata.
m_axis_tready	Input	1	AXI4-Stream tReady input signal. High value indicates that the downstream core is ready to receive data.
m_axis_tkeep	Output	2, 4, 8, 16, defined by M_AXIS_TDATA _WIDTH parameter	AXI4-Stream tKeep signal. Output will be driven to all ’1’s.

Register Space

The EDK pCore provides a memory-mapped interface for the programmable registers within the core, as described in Table 2-7.

Note: All registers default to 0x00000000 on power-up or software reset.

Table 2-7: **Video Scaler Registers Overview**

Address	Name	Read/Write	Description
0x0000	control	R/W	General control register
0x0004	status	R	General readable status register
0x0008	status_error	R	General readable status register for errors
0x000c	status_done	R/W	General read register for status done
0x0010	horz_shrink_factor	R/W	Horizontal Shrink Factor
0x0014	vert_shrink_factor	R/W	Vertical Shrink Factor

Table 2-7: **Video Scaler Registers Overview *(Cont'd)***

Address	Name	Read/Write	Description
0x0018	aperture_horz	R/W	aperture_start_pixel Location of first subject pixel in input line, relative to first active pixel in that line aperture_end_pixel Location of final subject pixel in input line, relative to first active pixel in that line
0x001c	aperture_vert	R/W	aperture_start_line Location of first subject line in input image, relative to first active line in that image aperture_end_line Location of final subject line in input image, relative to first active line in that image
0x0020	output_size	R/W	output_h_size Width of output image (pixels) output_v_size Height of the outuput image (lines)
0x0024	num_phases	R/W	num_h_phases Number of phases of coefficients in current horizontal filter set num_v_phases Number of phases of coefficients in current vertical filter set
0x0028	coeff_sets	R/W	hcoeffset Active coefficient set to use in horizontal filter operation vcoeffset Active coefficient set to use in vertical filter operation
0x002c	start_hpa_y	R/W	Fractional value used to initialize horizontal accumulator at rectangle left edge for luma
0x0030	start_hpa_c	R/W	Fractional value used to initialize vertical accumulator at rectangle top edge for luma
0x0034	start_vpa_y	R/W	Fractional value used to initialize horizontal accumulator at rectangle left edge for chroma
0x0038	start_vpa_c	R/W	Fractional value used to initialize vertical accumulator at rectangle top edge for chroma
0x003c	coef_write_set_addr	R/W	Coefficient set write address to indicate which coefficient bank to write
0x0040	coef_values	W	Coefficient values to write
0x0044	coef_set_bank_rd_addr	R/W	Set and bank number to be read
0x0048	coef_mem_rd_addr	R/W	Phase and tap number to be read

Table 2-7: **Video Scaler Registers Overview *(Cont'd)***

Address	Name	Read/Write	Description
0x004c	coef_mem_output	R	Coefficient readback output
0x00f0	Version	R	Core HW Version Register
0x0100	Software_Reset	W	Writing a SOFT_RESET value to this register resets the software registers and the Video Scaler IP core. The SOFT_RESET value is determined by EDK.
0x021C	GIER	R/W	Global Interrupt Enable Register
0x0220	ISR	R/W	Interrupt Status Register; read to determine the source of the interrupt, write to clear the interrupt
0x0228	IER	R/W	Interrupt Enable Register; 0 to mask out an interrupt, 1 to enable an interrupt

Table 2-8 through Table 2-32 describe the Video Scaler registers in more detail.

Table 2-8: **control Register**

0x0000								control																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																													enable		

Name	Bits	Description
Reserved	31:2	Reserved
Reg_Update_Enable	1	Register Update enable. This bit communicates to the IP core to take new values at the next frame vblank rising edge. The registers that utilize this bit are 0x0010 through 0x0038. Usage: This bit is cleared when the IP core next vblank happens.
Enable	0	Enable the Video Scaler core on the next video frame.

Table 2-9: **reserved Register**

0x0004								status																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																															C

Name	Bits	Description
Reserved	31:1	Reserved
Coef_write_rdy	0	If this bit is '1' then the Coeffs can be written into the core. Check at the beginning of a coeff transfer.

Table 2-10: **status Register**

0x0008								status_error																				R			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Error_Code3								Error_Code2								Error_Code1								Error_Code0							

Name	Bits	Description
Error_Code3	31:24	Error codes to be defined
Error_Code2	23:16	Error codes to be defined
Error_Code1	15:8	Error codes to be defined
Error_Code0	7:0	Error codes to be defined

Table 2-11: **status_done Register**

0x000c								status_done																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved																															d

Name	Bits	Description
Reserved	31:24	Reserved
Reserved	23:16	Reserved
Reserved	15:8	Reserved
Reserved	7:1	Reserved
Done	0	Done bit can be polled by software for end for video scaler operation. Usage: This bit is cleared when any value is written to the register.

Table 2-12: **horizontal_shrink_factor Register**

0x0010								horz_shrink_factor																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved								hsf_int				hsf_frac																			

Name	Bits	Description
Reserved	31:24	Reserved
hsf_int	23:20	Horizontal Shrink Factor integer
hsf_frac	19:0	Horizontal Shrink Factor fractional

Table 2-13: **vsf Register**

<table>
<tr><td colspan="8">0x0014</td><td colspan="20">vert_shrink_factor</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">vsf_int</td><td colspan="20">vsf_frac</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:24</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">vsf_int</td><td colspan="4">23:20</td><td colspan="20">Vertical Shrink Factor integer</td></tr>
<tr><td colspan="8">vsf_frac</td><td colspan="4">19:0</td><td colspan="20">Vertical Shrink Factor fractional</td></tr>
</table>

Table 2-14: **aperture_horz Register**

<table>
<tr><td colspan="8">0x0018</td><td colspan="20">aperture_horz</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="5">Reserved</td><td colspan="11">aperture_end_pixel</td><td colspan="5">Reserved</td><td colspan="11">aperture_start_pixel</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:26</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">aperture_end_pixel</td><td colspan="4">28:16</td><td colspan="20">Location of last pixel in line</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">15:11</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">aperture_start_pixel</td><td colspan="4">12:0</td><td colspan="20">Location of first pixel in line</td></tr>
</table>

Table 2-15: **aperture_vert Register**

<table>
<tr><td colspan="8">0x001c</td><td colspan="20">aperture_vert</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="5">Reserved</td><td colspan="11">aperture_end_line</td><td colspan="5">Reserved</td><td colspan="11">aperture_start_line</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:27</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">aperture_end_line</td><td colspan="4">28:16</td><td colspan="20">Location of last line in active video</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">15:11</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">aperture_start_line</td><td colspan="4">12:0</td><td colspan="20">Location of first line in active video</td></tr>
</table>

Table 2-16: **output_size Register**

<table>
<tr><td colspan="8">0x0020</td><td colspan="20">output_size</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="5">Reserved</td><td colspan="11">output_v_size</td><td colspan="5">Reserved</td><td colspan="11">output_h_size</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:27</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">output_v_size</td><td colspan="4">28:16</td><td colspan="20">Number of lines in output image</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">15:11</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">output_h_size</td><td colspan="4">12:0</td><td colspan="20">Number of pixels in output image</td></tr>
</table>

Table 2-17: **num_phases Register**

<table>
<tr><td colspan="8">0x0024</td><td colspan="20">num_phases</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="17">Reserved</td><td colspan="7">num_v_phases</td><td></td><td colspan="7">num_h_phases</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:15</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">num_v_phases</td><td colspan="4">14:8</td><td colspan="20">Number of vertical phases</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">7</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">num_h_phases</td><td colspan="4">6:0</td><td colspan="20">Number of horizontal phases</td></tr>
</table>

Table 2-18: **coeff_sets Register**

<table>
<tr><td colspan="8">0x0028</td><td colspan="20">coeff_sets</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="24">Reserved</td><td colspan="4">vcoeffset</td><td colspan="4">hcoeffset</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:28</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">vcoeffset</td><td colspan="4">7:4</td><td colspan="20">Active vertical coefficient set</td></tr>
<tr><td colspan="8">hcoeffset</td><td colspan="4">3:0</td><td colspan="20">Active horizontal coefficient set</td></tr>
</table>

Table 2-19: **start_hpa_y Register**

0x002c								start_hpa_y																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved												start_hpa_y																			

Name	Bits	Description
Reserved	31:21	Reserved
start_hpa_y	20:0	Fractional value used to initialize horizontal accumulator for luma

Table 2-20: **start_vpa_y Register**

0x0030								start_hpa_c																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved												start_hpa_c																			

Name	Bits	Description
Reserved	31:21	Reserved
start_hpa_c	20:0	Fractional value used to initialize horizontal accumulator for chroma

Table 2-21: **start_hpa_c Register**

0x0034								start_vpa_y																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved												start_vpa_y																			

Name	Bits	Description
Reserved	31:21	Reserved
start_vpa_y	20:0	Fractional value used to initialize vertical accumulator for luma

Table 2-22: **start_vpa_c Register**

0x0038								start_vpa_c																				R/W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reserved												start_vpa_c																			

Name	Bits	Description
Reserved	31:21	Reserved
start_vpa_c	20:0	Fractional value used to initialize vertical accumulator for chroma

Table 2-23: **Coefficient_write_set_address Register**

<table>
<tr><td colspan="8">0x003c</td><td colspan="20">coef_write_set_addr</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="28">Reserved</td><td colspan="4">coef_wsa</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Reserved</td><td colspan="4">31:4</td><td colspan="20">Reserved</td></tr>
<tr><td colspan="8">coef_write_set_addr</td><td colspan="4">3:0</td><td colspan="20">Coefficient bank to write, address</td></tr>
</table>

Table 2-24: **coef_values Register**

<table>
<tr><td colspan="8">0x0040</td><td colspan="20">coef_values</td><td colspan="4">W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="16">coef_value_N+1</td><td colspan="16">coef_value_N</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">coef_value_N+1</td><td colspan="4">31:16</td><td colspan="20">Coefficient value N+1 where N is index for the coefficient set.
Usage: Each write to this register increments an internal counter by 2 to generate a coefficient set internal to the video scaler. LSB aligned for coefficients less than 16 bits.</td></tr>
<tr><td colspan="8">coef_value_N</td><td colspan="4">15:0</td><td colspan="20">Coefficient value N where N is index for the coefficient set.
Usage: Each write to this register increments an internal counter by 2 to generate a coefficient set internal to the video scaler. LSB aligned for coefficients less than 16 bits</td></tr>
</table>

Table 2-25: **Coefficient Set and Bank Read Address Register**

<table>
<tr><td colspan="8">0x0044</td><td colspan="16">coef_set_bank_rd_addr</td><td colspan="8">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="20">Reserved</td><td colspan="4">Set</td><td colspan="6">Reserved</td><td colspan="2">Bank</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Coeff Readback **Set**</td><td colspan="4">11:8</td><td colspan="20">Coefficient set to be read from the scaler</td></tr>
<tr><td colspan="8">Coeff Readback **Bank**</td><td colspan="4">1:0</td><td colspan="20">Coefficient bank to be read from scaler:
00=HY; 01=HC; 10=VY; 11=VC</td></tr>
</table>

Table 2-26: **Coefficient Phase and Tap Read Address Register**

<table>
<tr><td colspan="8">0x0048</td><td colspan="16">coef_mem_rd_addr</td><td colspan="8">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="18">Reserved</td><td colspan="6"><b>Phase</b></td><td colspan="4">Reserved</td><td colspan="4"><b>Tap</b></td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Coeff Readback <b>Phase</b></td><td colspan="4">13:8</td><td colspan="20">Coefficient phase to be read from the scaler</td></tr>
<tr><td colspan="8">Coeff Readback <b>Bank</b></td><td colspan="4">3:0</td><td colspan="20">Coefficient tap to be read from scaler</td></tr>
</table>

Table 2-27: **Coefficient Memory Readback Output Register**

<table>
<tr><td colspan="8">0x004c</td><td colspan="16">coef_mem_rd_addr</td><td colspan="8">R</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">Reserved</td><td colspan="16"><b>Coeff Readback Output</b></td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8"><b>Coeff Readback Output</b></td><td colspan="4">15:0</td><td colspan="20">Coefficient readout from the scaler</td></tr>
</table>

Table 2-28: **Version Register**

<table>
<tr><td colspan="8">0x00F0</td><td colspan="16">Version</td><td colspan="8">R</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="32"><b>HW Version</b></td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8"><b>HW Version</b></td><td colspan="4">31:0</td><td colspan="20">Hard-coded hardware version register</td></tr>
</table>

Table 2-29: **Software Reset Register**

<table>
<tr><td colspan="8">0x0100</td><td colspan="20">Software_Reset</td><td colspan="4">W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="31">Reserved</td><td>d</td></tr>
<tr><td colspan="8">Name</td><td colspan="4">Bits</td><td colspan="20">Description</td></tr>
<tr><td colspan="8">Soft_Reset_Value</td><td colspan="4">31:0</td><td colspan="20">Soft Reset to reset the registers and IP core, data Value provided by the EDK create peripheral utility</td></tr>
</table>

Table 2-30: **Global Interrupt Enable Register**

<table>
<tr><td colspan="8">0x021C</td><td colspan="20">Software_Reset</td><td colspan="4">W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="31">Reserved</td><td>d</td></tr>
</table>

Name	Bits	Description
GIER	31	Global Interrupt Enable Register. Active High
Reserved	30:0	Reserved

Table 2-31: **Interrupt Status Register**

<table>
<tr><td colspan="8">0x0220</td><td colspan="20">ISR</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="26">Reserved</td><td colspan="6">Int</td></tr>
</table>

Name	Bits	Description
Reserved	31:6	Reserved
intr_coef_mem_rdbk_rdy	6	**Level sensitive**: Output flag indicating that the specified coefficient bank is ready for reading.
intr_reg_update_done	5	**Level sensitive:** issued during Vertical blanking when the register values have been transferred to the active registers.
intr_coef_wr_error	4	**Rising edge sensitive:** issued if coefficient is written into coefficient FIFO when the FIFO is not ready.
intr_output_error	3	**Rising edge sensitive:** issued if frame period completes before full output frame has been delivered.
intr_input_error	2	**Rising edge sensitive:** issued if active_video_in is asserted before the scaler is ready to receive a new line.
intr_coef_fifo_rdy	1	**Level sensitive:** issued when the coefficient FIFO is ready to receive a coefficient for the current set. Stays low once a full set has been written into FIFO. Sent high during Vertical blanking.
intr_output_frame_done	0	**Rising edge sensitive:** issued once per complete output frame.

Table 2-32: **Interrupt Enable Register**

<table>
<tr><td colspan="8">0x0228</td><td colspan="20">IER</td><td colspan="4">R/W</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="26">Reserved</td><td colspan="6">Int</td></tr>
</table>

Name	Bits	Description
Reserved	31:6	Reserved
intr_coef_mem_rdbk_rdy	6	Mask or enable interrupt for intr_coef_mem_rdbk_rdy
intr_reg_update_done	5	Mask or Enable interrupt for intr_reg_update_done
intr_coef_wr_error	4	Mask or Enable interrupt for intr_coef_wr_error
intr_output_error	3	Mask or Enable interrupt for intr_output_error
intr_input_error	2	Mask or Enable interrupt for intr_input_error
intr_coef_fifo_rdy	1	Mask or Enable interrupt for intr_coef_fifo_rdy
intr_output_frame_done	0	Mask or Enable interrupt for intr_output_frame_done

Chapter 3

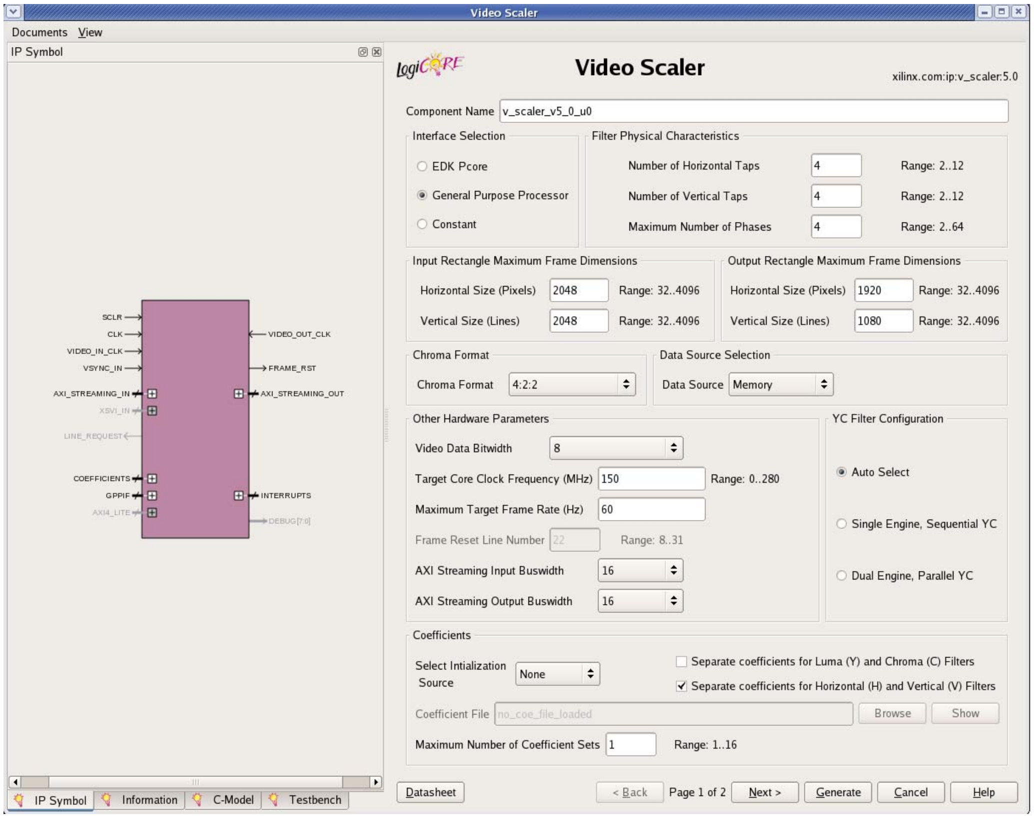
Customizing and Generating the Core

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

Graphical User Interface (GUI)

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 GUI main screen in GPP Mode.



Video Scaler

xilinx.com:ip:v_scaler:5.0

Component Name: v_scaler_v5_0_u0

Interface Selection
○ EDK Pcore
● General Purpose Processor
○ Constant

Filter Physical Characteristics
Number of Horizontal Taps: 4 Range: 2..12
Number of Vertical Taps: 4 Range: 2..12
Maximum Number of Phases: 4 Range: 2..64

Input Rectangle Maximum Frame Dimensions
Horizontal Size (Pixels): 2048 Range: 32..4096
Vertical Size (Lines): 2048 Range: 32..4096

Output Rectangle Maximum Frame Dimensions
Horizontal Size (Pixels): 1920 Range: 32..4096
Vertical Size (Lines): 1080 Range: 32..4096

Chroma Format
Chroma Format: 4:2:2

Data Source Selection
Data Source: Memory

Other Hardware Parameters
Video Data Bitwidth: 8
Target Core Clock Frequency (MHz): 150 Range: 0..280
Maximum Target Frame Rate (Hz): 60
Frame Reset Line Number: 22 Range: 8..31
AXI Streaming Input Buswidth: 16
AXI Streaming Output Buswidth: 16

YC Filter Configuration
● Auto Select
○ Single Engine, Sequential YC
○ Dual Engine, Parallel YC

Coefficients
Select Intialization Source: None
☐ Separate coefficients for Luma (Y) and Chroma (C) Filters
☑ Separate coefficients for Horizontal (H) and Vertical (V) Filters
Coefficient File: no_coe_file_loaded
Maximum Number of Coefficient Sets: 1 Range: 1..16

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 "_".

Interface Selection: The Video Scaler is generated with one of three interfaces:

- EDK pCore Interface: CORE Generator software generates the Video Scaler as a pCore that can be easily imported into an EDK project as a hardware peripheral. The core registers can then be programmed in real-time via a MicroBlaze processor and the AXI4-Lite interface. See AXI4-Lite pCore Interface in Chapter 2 for more information. When the EDK pCore is selected, the rest of the options are disabled (greyed out) and set to the default value. All modifications to the Video Scaler pCore are made with the EDK GUI.
- General Purpose Processor Interface: CORE Generator software generates a set of ports that can be used to program the Video Scaler. See General Purpose Processor (GPP) Interface in Chapter 2 for more information.
- Constant Interface: On CORE Generator GUI page 2, the user may enter fixed settings for the Video Scaler Parameters. See General Purpose Processor (GPP) Interface in Chapter 2 for more information. When the Constant Mode is selected, the options on GUI page 1 are disabled (greyed out) and set to the default value.

Num H 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.

Num V 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 (for pCore and GPP only): 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 (for pCore and GPP only)**:** 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.

Video Data Bitwidth: 8, 10, or 12 bits. This specifies both the input and output video bitwidths. This should not be confused with the AXI4-Stream bitwidths.

Max Coef Sets (for pCore and GPP only)**:** This represents the maximum number of sets of coefficients that may be stored internally to the scaler. It may 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.

Data Source Selection: The user may select how he intends to deliver video data to the core – Live or Memory data source.

Frame Reset Line Number (for Live-Video data-source only)**:** The user may set this value to move the position of the `frame_rst` output signal within the vertical blanking. It must be set such that `frame_rst` occurs while `vblank_in` is high.

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 the user's 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 user's 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 (TgtFrameRate <= 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.

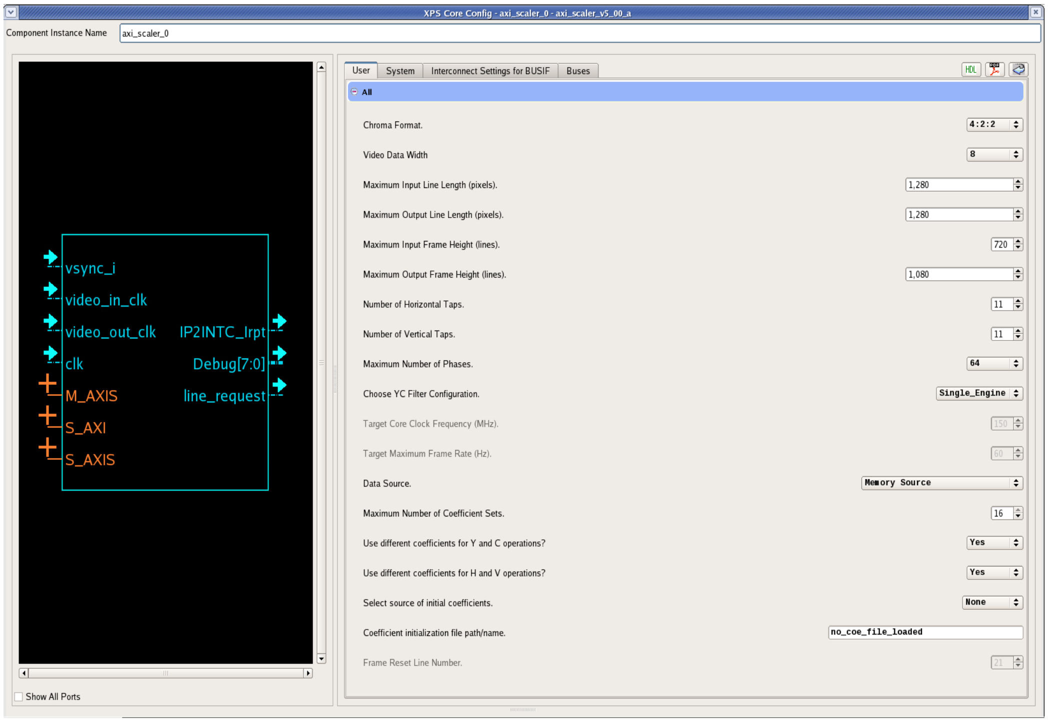
The user may specify whether the same coefficients are used for Y and C filter operations.The user may also specify whether the H and V operations use the same coefficients. This is only an option if the specified number of horizontal taps is equal to the specified number of vertical taps. Specifying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

AXI Stream Input/Output Buswidth: The data buses in these interfaces can be 16, 32, 64 or 128 bits wide. Input and output bus widths can be different, if necessary. Care should be taken to ensure that the AXI4-Stream buswidth is sufficient to accommodate all video data bits implied by the settings of Video Data Buswidth and chroma format.

pCore Interface

CORE Generator software may be configured to generate the scaler as a pCore to be built into an EDK project. In this case, CORE Generator creates un-synthesized encrypted VHDL source code. All options in the GUI are greyed-out in this case - the user must parameterize the scaler pCore in the EDK environment.

When in the EDK environment, the Video scaler GUI looks slightly different (see Figure 3-2), but offers the same options as the GPP CORE Generator GUI.



Component Instance Name: axi_scaler_0

Chroma Format.: 4:2:2
Video Data Width: 8
Maximum Input Line Length (pixels).: 1,280
Maximum Output Line Length (pixels).: 1,280
Maximum Input Frame Height (lines).: 720
Maximum Output Frame Height (lines).: 1,080
Number of Horizontal Taps.: 11
Number of Vertical Taps.: 11
Maximum Number of Phases.: 64
Choose YC Filter Configuration.: Single_Engine
Target Core Clock Frequency (MHz).: 150
Target Maximum Frame Rate (Hz).: 60
Data Source.: Memory Source
Maximum Number of Coefficient Sets.: 16
Use different coefficients for Y and C operations?: Yes
Use different coefficients for H and V operations?: Yes
Select source of initial coefficients.: None
Coefficient initialization file path/name.: no_coe_file_loaded
Frame Reset Line Number.: 21

□ Show All Ports

Figure 3-2: **Video Scaler EDK GUI**

The GPP interface ports that are described in General Purpose Processor (GPP) Interface in Chapter 2 exist in the wrapper but are driven by registers on the AXI. These unused ports are greyed-out in the CORE Generator symbol, and are replaced by the AXI interface.

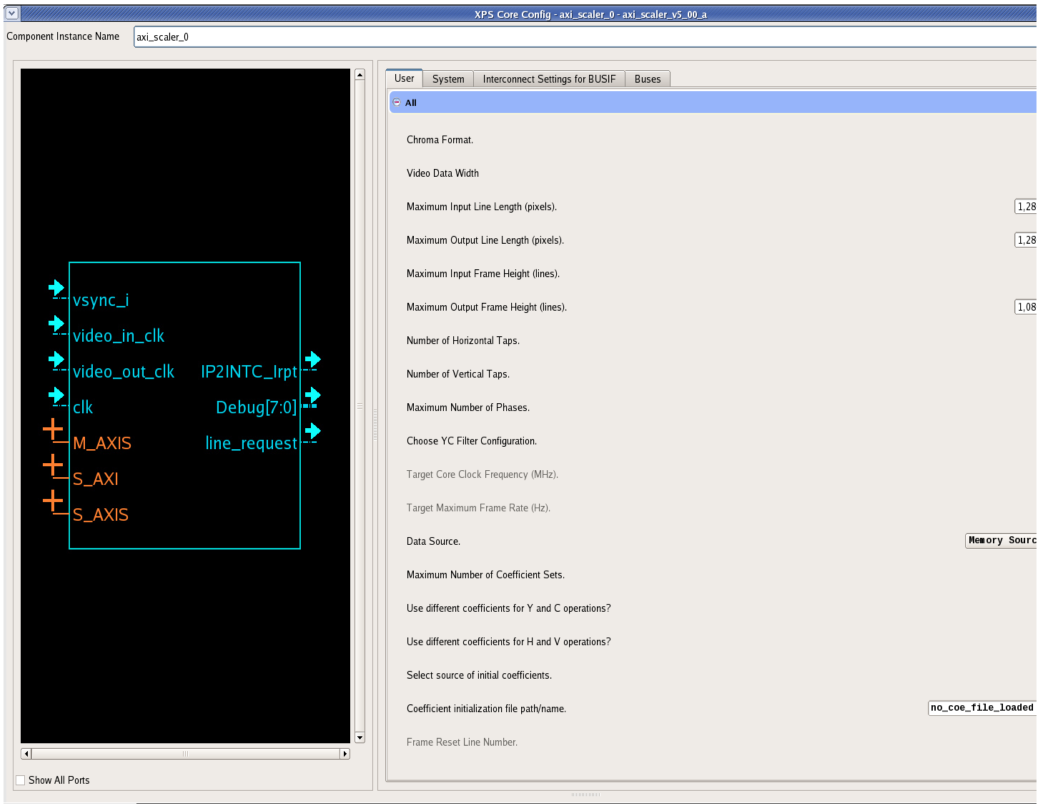
Fully verified MicroBlaze processor software driver source code is also provided by CORE Generator software for driving all of the control inputs. These are briefly described in Table 3-2.

GUI Parameters

Constant (Fixed Mode) Interface

This option generates a netlist whose scaling parameters are predetermined on page 2 of the CORE Generator GUI (Figure 3-4). This option removes the need for the user to control the inputs dynamically if a fixed-mode scaler is desired, and reduces resource usage. See Figure 3-3 and Figure 3-4.

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.



Component Instance Name: axi_scaler_0

Chroma Format.

Video Data Width

Maximum Input Line Length (pixels). 1,28

Maximum Output Line Length (pixels). 1,28

Maximum Input Frame Height (lines).

Maximum Output Frame Height (lines). 1,08

Number of Horizontal Taps.

Number of Vertical Taps.

Maximum Number of Phases.

Choose YC Filter Configuration.

Target Core Clock Frequency (MHz).

Target Maximum Frame Rate (Hz).

Data Source. Memory Sourc

Maximum Number of Coefficient Sets.

Use different coefficients for Y and C operations?

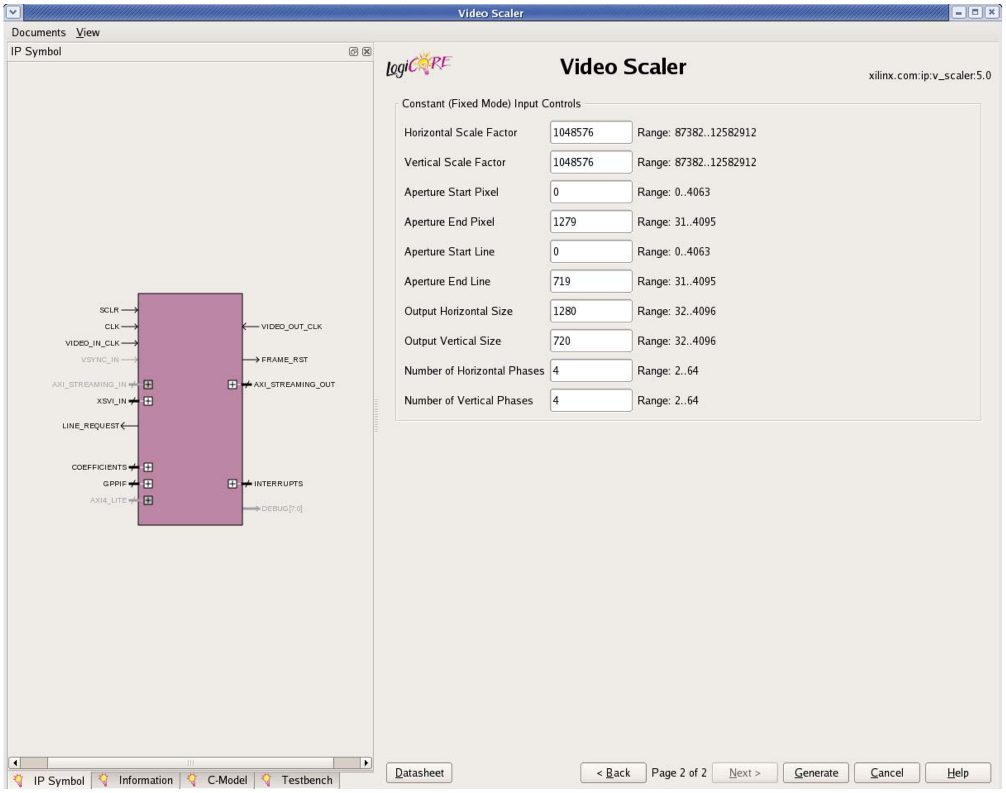
Use different coefficients for H and V operations?

Select source of initial coefficients.

Coefficient initialization file path/name. no_coe_file_loaded

Frame Reset Line Number.

Figure 3-3: **Video Scaler Graphical User Interface for Constant Mode (page 1)**



Video Scaler

xilinx.com:ip:v_scaler:5.0

Constant (Fixed Mode) Input Controls

Horizontal Scale Factor	1048576	Range: 87382..12582912
Vertical Scale Factor	1048576	Range: 87382..12582912
Aperture Start Pixel	0	Range: 0..4063
Aperture End Pixel	1279	Range: 31..4095
Aperture Start Line	0	Range: 0..4063
Aperture End Line	719	Range: 31..4095
Output Horizontal Size	1280	Range: 32..4096
Output Vertical Size	720	Range: 32..4096
Number of Horizontal Phases	4	Range: 2..64
Number of Vertical Phases	4	Range: 2..64

Figure 3-4: **Video Scaler Graphical User Interface for Constant Mode (page 2)**

Constant-Mode GUI Parameters

Horizontal Scale Factor, Vertical Scale Factor (for Constant Mode only)**:** Specify, as unsigned integers, the 24-bit numbers that represent the desired fixed scale factors. Calculation of these values is described as HSF, VSF in Control Values in Chapter 4.

Aperture Start Pixel, Aperture End Pixel, Aperture Start Line, Aperture End Line (for Constant Mode only)**:** See Control Values in Chapter 4. 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 Live Video data-source. In Memory mode, Aperture Start Pixel and Aperture Start Line are fixed at 0.

Output Horizontal Size, Output Vertical Size (for Constant Mode only)**:** 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 (for Constant Mode only)**:** Non power-of-two numbers of phases are supported.

Coefficient File Input (for Constant Mode only)**:** 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.

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
coefficient_file	no_coe_file_loaded	‘no_coe_file_loaded’, <valid coe file>
component_name	v_scaler_v5_0_u0	Not ‘v_scaler_v5_0’
data_source	Memory	‘Memory’, ‘Live_XSVI_Input’
data_width	8	8, 10, 12
frame_reset_line_number	22	8-31
horizontal_scale_factor	1048576	87382 - 12582912
init_coef_source	None	‘None’, ‘COE_File’
interface_selection	EDK_Pcore	‘EDK_Pcore’, ‘General_Purpose_Processor’, ‘Constant’
m_axis_tdata_width	16	16, 32, 64, 128
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

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

XCO Parameter	Default	Valid Values
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, 32, 64, 128
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 whether the interface selection is set to EDK pCore, General Purpose Processor or Constant. The output files are placed in the project directory.

EDK pCore Files

In contrast to GPP Mode and Constant Mode control interfaces, when you select this control interface option in CORE Generator, no netlist is created. Instead, a database is generated containing the necessary files for use in an EDK project. This database includes:

```
<component_name> -> drivers  -> scaler_v3_01_a     -> data     -> scaler_v2_1_0.mdd
                                                                  scaler_v2_1_0.tcl
                                                   -> example  -> example.c
                                                   -> src      -> Makefile
                                                                  xscaler.c
                                                                  xscaler.h
                                                                  xscaler_coefs.c
                                                                  xscaler_g.c
                                                                  xscaler_hw.h
                                                                  xscaler_intr.c
                                                                  xscaler_sinit.c
                 -> pcores   -> axi_scaler_v5_00_a -> data     -> scaler_v2_1_0.mpd
                                                                  scaler_v2_1_0.pao
                                                   -> hdl      -> vhdl->CoefsFIFO.vhd
                                                                      coefs.vhd
```

CoefRAM.vhd

CoefMemBlk.vhd
HeartBeater.vhd
HPhaseAccumulator.vhd
HWT.vhd
ImageXLib_arch.vhd
ImageXLib_utils.vhd
MemXLib_arch.vhd
MemXLib_utils.vhd
Scaler.vhd
Scaler_RTI.vhd
Scaler_wrap0.vhd
Scaler_wrap0_core.vhd
ScalerExternalSM.vhd
syncgen_core.vhd
user_logic.vhd
v_scaler_v5_0.vhd
xscaler.vhd
YCCheckSum.vhd

For use in an EDK project:

1. Copy the /drivers/scaler_v3_01_a sub-directory from the CORE Generator database to the /drivers directory in your EDK project repository.
2. Copy the /pcores/axi_scaler_v4_00_a sub-directory from the CORE Generator database to the /pcores directory in your EDK project repository.

All VHDL files are encrypted. ***Do not attempt to modify these files***.

Scaler Software Driver

All files provided by CORE Generator software under the drivers directory are tested SW drivers for the video scaler. They are unencrypted c-code which you may adapt for your own environment. This is intended for a memory-mapped system. The register map for the scaler registers is given in Chapter 2, Core Interfaces and Register Space.

File Details

<project directory>

This is the top-level directory. It contains .xco and other assorted files.

- `<component_name>.xco`: Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
- `<component_name>_flist.txt`: A text file listing all of the output files produced when the customized core was generated in the CORE Generator software.

<project directory>/<component_name>/pcores/axi_scaler_v5_00_a/data

This directory contains files that EDK uses to define the interface to the pCore.

< project directory>/<component_name>/pcores/axi_scaler_v5_00_a /hdl/vhdl

This directory contains the Hardware Description Language (HDL) files that implement the pCore.

< project directory>/<component_name>/drivers/scaler_v3_01_a/data

This directory contains files that Software Development Kit (SDK) uses to define the operation of the pCore's software driver.

< project directory>/<component_name>/drivers/ scaler_v3_01_a /doc/html/api

This directory contains HTML documentation files for the pCore's software driver.

< project directory>/<component_name>/drivers/ scaler_v3_01_a /src

This directory contains the source code of the pCore's software driver. The delivered files are listed in Table 3-2.

Table 3-2: **pCore Driver Files Delivered from CORE Generator**

File name	Description
\drivers\scaler_v3_01_a\example\example.c	Examples that demonstrate how to control the scaler core; Up-scaling and downscaling examples included
\drivers\scaler_v3_01_a\src\xscaler.h	Declaration of all driver functions and driver instance data structure definition
\drivers\scaler_v3_01_a\src\xscaler_hw.h	Register and bit definition of the scaler device
\drivers\scaler_v3_01_a\src\xscaler.c	Implementation of general driver functions
\drivers\scaler_v3_01_a\src\xscaler_intr.c	Implementation of the interrupt-related functions
\drivers\scaler_v3_01_a\src\xscaler_sinit.c	Implementation of the static initialization function
\drivers\scaler_v3_01_a\src\xscaler_g.c	Definition of scaler device list, with each element defining parameters for a scaler device, such as base address, vertical tap number, etc.
\drivers\scaler_v3_01_a\src\xscaler_coefs.c	Definition of all coefficients

General Purpose Processor and Constant Files

When the interface selection is set to General Purpose Processor or Constant, CORE Generator then outputs the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the `<project directory>`.

File Details

The CORE Generator software output consists of some or all of the files listed in Table 3-3.

Table 3-3: **CORE Generator Files for GPP or Constant Mode**

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo <component_name>.vho	The HDL templates for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation models for the core. They are used for functionally simulating the core.
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in the CORE Generator tool.
<component_name>.asy	IP symbol file.
<component_name>.gise <component_name>.xise	ISE software subproject files for use when including the core in ISE software designs.

Chapter 4

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).

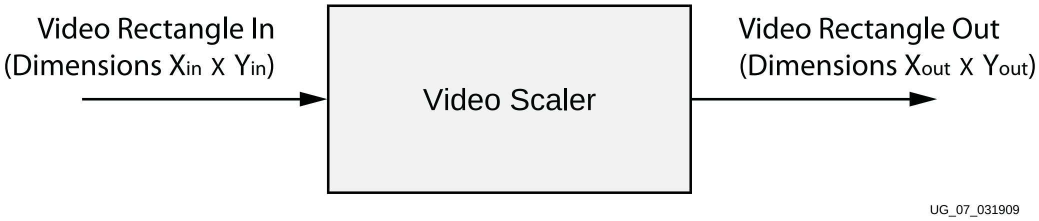


Figure 4-1: **High Level View of the Functionality**

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.

When using a live video input source, you have no control over the video timing signals. Hence, the specific requirements must allow for this. For example, when up-scaling by a factor of 2, two lines must be output for every input line. The scaler core clock-rate ('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.

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.

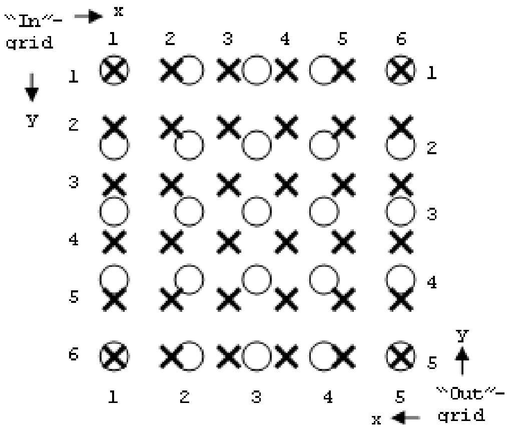


Figure 4-2: **5x5 Output Grid ("O") Super-imposed over 6x6 Input Grid ("X")**

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.

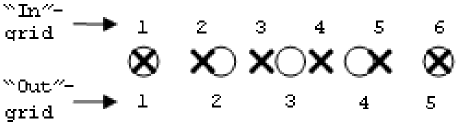


Figure 4-3: **Super-imposed Grids for 1 Dimension**

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.

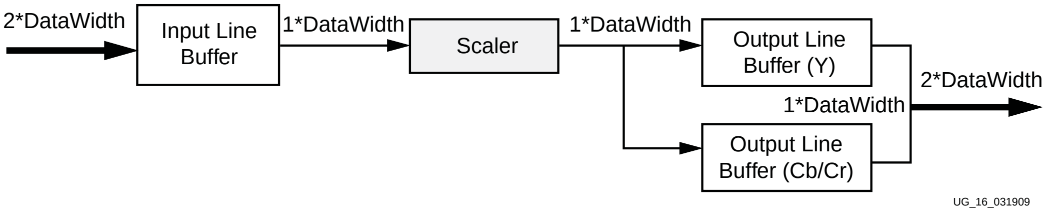


Figure 4-4: **Internal Data Path Bitwidths for Single-Engine YC Mode**

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.

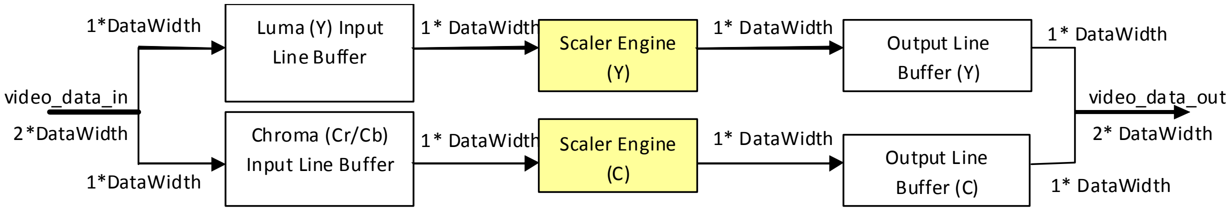


Figure 4-5: **Internal Data Path Bitwidths for Dual-Engine YC Mode**

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.

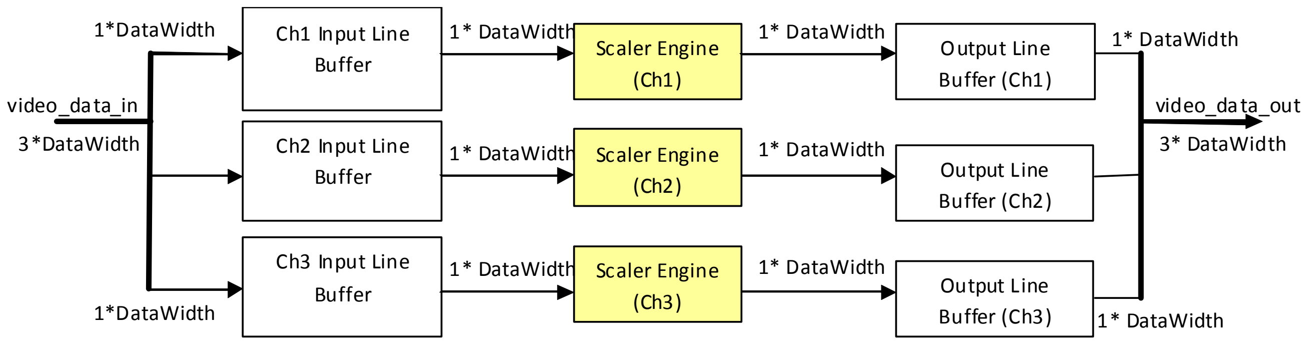


Figure 4-6: **Internal Data Path Bitwidths for Triple-Engine RGB/4:4:4 Architecture**

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

Data Source: Memory

When this mode is selected, data is transferred between external memory and the Video Scaler via AXI Interconnect and the AXI-VDMA using its AXI4-Stream ports, as shown in Figure 4-7.

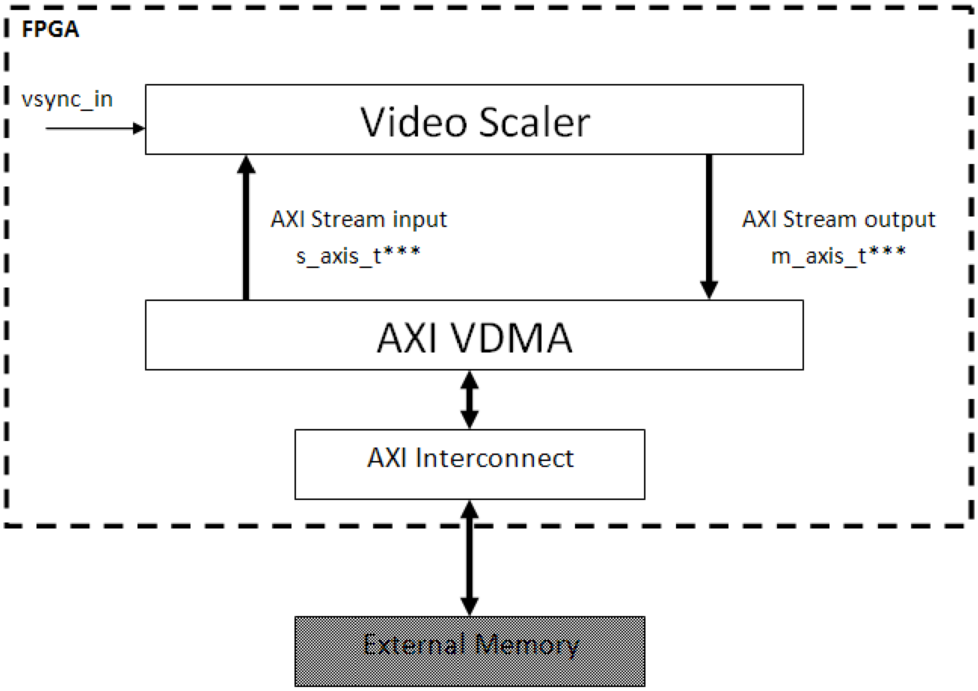


Figure 4-7: **Memory Source Use Model**

The user can alternatively elect to build an internal buffering solution. The size and nature of the internal buffer depend heavily upon the user's worst-case scaling requirements. The block RAM-based internal buffer block should ideally be constructed using the AXI4-Stream interface.

Data Source: Live

When this mode is selected, the scaler expects valid video data aligned with the `ACTIVE_VIDEO_IN` signal. Horizontal and Vertical synchronization signals must also be provided on the `hblank_in` and `vblank_in` pins. This usage is shown in Figure 4-8. The Live Mode may be selected in the “Data Source” drop-down box in the CORE Generator tool GUI. Note that the XSVI bus becomes active on the CORE Generator tool symbol on the left side of the GUI.

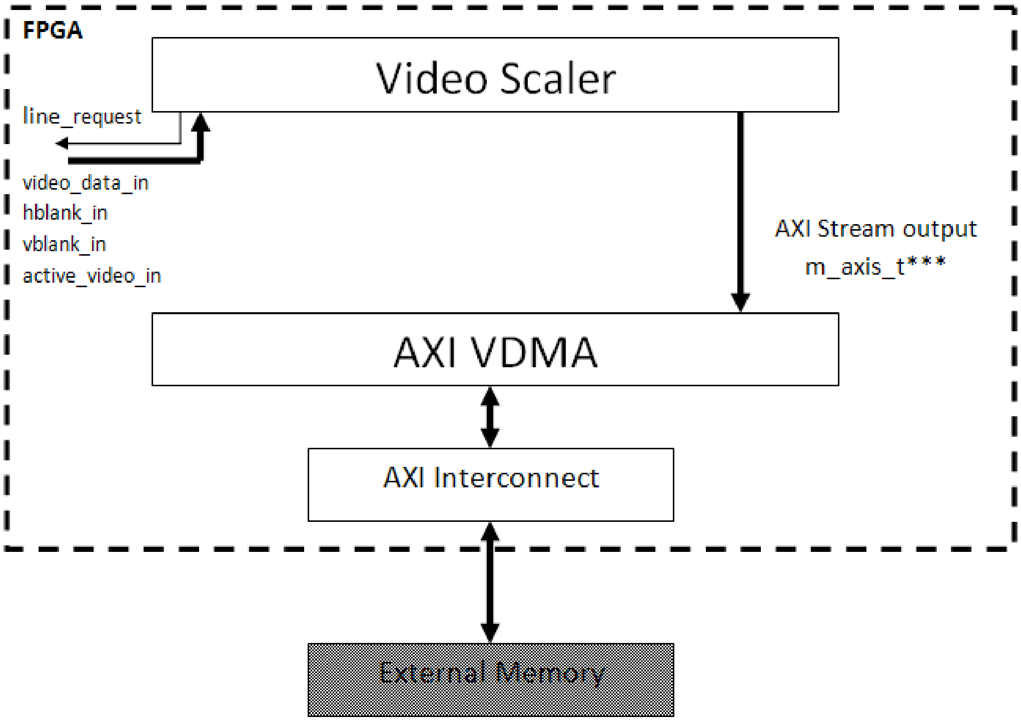


Figure 4-8: **Live Source Use Model**

Live Data Source – Input Control Signals and Timing

Valid video data is written into the input line-buffer, using `active_video_in`, shown in Figure 4-9. `active_video_in` must remain in a high state for the duration of the active input line.

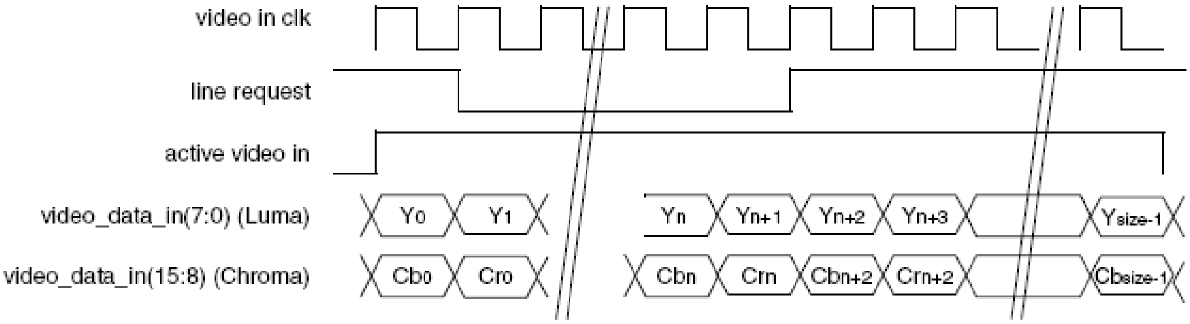


Figure 4-9: **Scaler 4:2:2 Input Timing**

An additional input, `active_chroma_in,` is required in the 4:2:0 case. This must be asserted high on all lines for 4:2:2, but only for alternate lines for 4:2:0, as shown in

Figure 4-10. There must be valid data at line 1 (counting from line 1; *not* line 0) and at every odd numbered line after line 1.

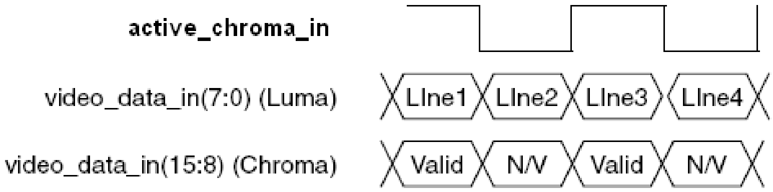


Figure 4-10: **Scaler 4:2:0 Input Chroma Validation**

Clocking

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

- `video_in_clk` handles the clocking of data into the core.
- `clk` is used internally to the core.
- `video_out_clk` is the clock that will be used to read video data out from the core.

Figure 4-11 shows the top level buffering, indicating the different clock domains, and the scope of the control state-machines.

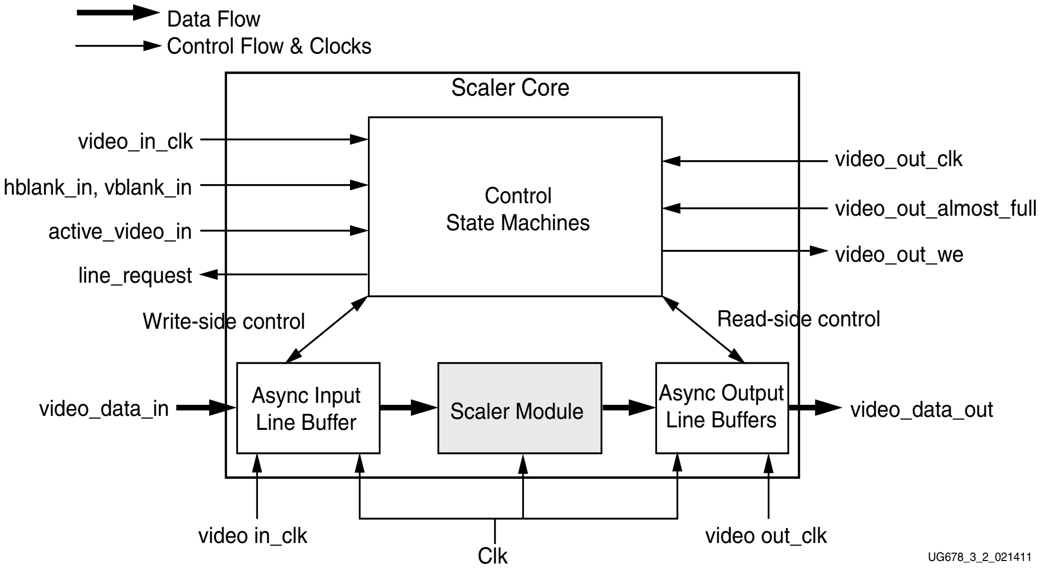


Figure 4-11: **Block Diagram Showing Clock Domains (Live Mode)**

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 1 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 4-11. The control state-machines monitor the I/O line buffers. They also monitor the current input and output line numbers.

Output Signals and Timing

When a line of data becomes available in the output buffer, and the video_out_full flag is low, the video_out_we flag is asserted as shown in Figure 4-12, and data is driven out. The target must deassert video_out_full when it is ready to accept the entire line.

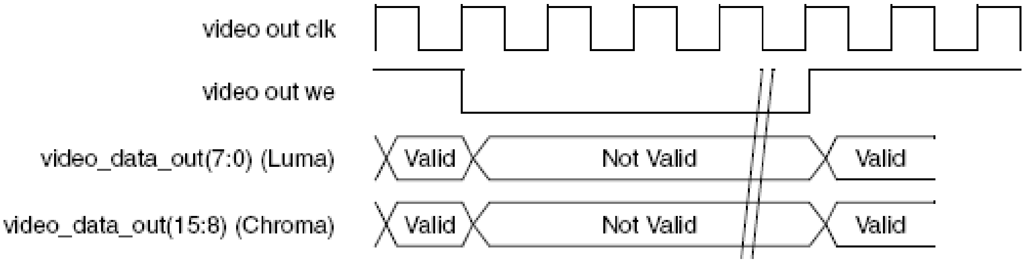


Figure 4-12: **Scaler Output Timing**

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.

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. Line 0 is defined as the first active line following a rising edge in active_video_in. An internal line counter is decoded to signal internally that the current line is indeed line 0. This line counter is reset on a falling edge of vblank_in. It increments on a rising edge of hblank_in.

One situation that needs to be avoided is the counter effectively starting at 1 instead of 0. This will cause no video output. The correct relationship between input hblank_in and vblank_in to avoid this situation is shown in Figure 4-13. **The falling edge of vblank_in occurs while hblank_in is still high.**

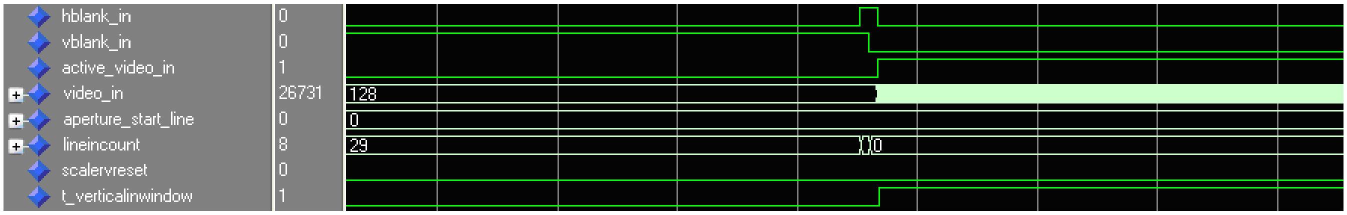


Figure 4-13: **Hblank_in at Falling Edge of VBlank_in**

Pixel 0 is defined as the first active pixel after the rising edge of active_video_in. This is indicated in Figure 4-14. The value 128 is used as the default value in video_data_in during blanking. In this example, the first pixel in the horizontal scaler aperture is the first active pixel in the input line.

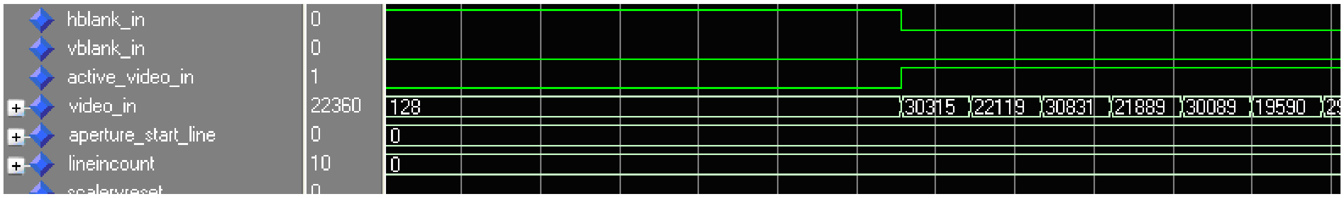


Figure 4-14: **Active_video_in in Relation to First Active Sample**

Cropping

When using "Live" mode, 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

Figure 4-15 shows the opening of an internal processing window signal (t_verticalwindow) with the preceding cropping settings. A similar operation occurs in the horizontal domain. A useful developer note is that if the largest input rectangle is cropped from the input, then this size may be used in deciding the `max_pixels_in_per_line` parameter. This may save block RAM usage in some cases.

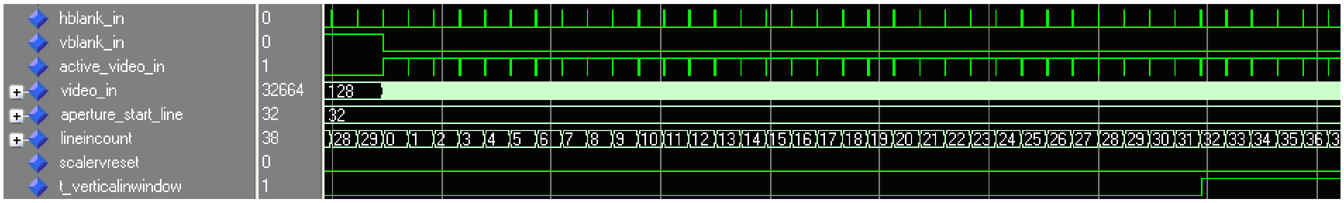


Figure 4-15: **Cropping from the Input Image**

When using "Memory" mode, cropping must be achieved by selecting the appropriate rectangular area from memory. `aperture_start_pixel` and `aperture_start_line` must be set to zero.

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 the user an easy way of initializing the scaler from power-up.

When using pCore or GPP interfaces, 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 by the phase accumulators.

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 will be scaled identically).

If the user specifies 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, the user must not load chroma coefficients using the Coefficient interface, and must not specify chroma coefficients in the .coe file.

Similarly, if the user has 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, the user 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 coefficient memory interface, as shown in Table 4-3.

Table 4-3: **Coefficient Loading Interface Signaling**

Input	Description
coef_data_in(31:0)	32-bit coefficient input bus
coef_wr_en	Coefficient write-enable
coef_set_wr_addr(3:0)	Coefficient set write address
intr_coef_fifo_rdy	Output flag indicating the readiness of the scaler to accept another coefficient.

The 32-bit input word always holds two coefficients. The scaler supports 16-bit coefficient bit-widths. The word format is shown in Figure 4-16.

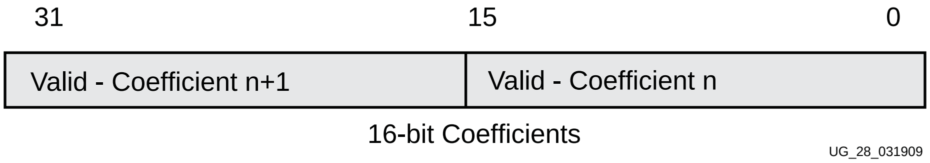
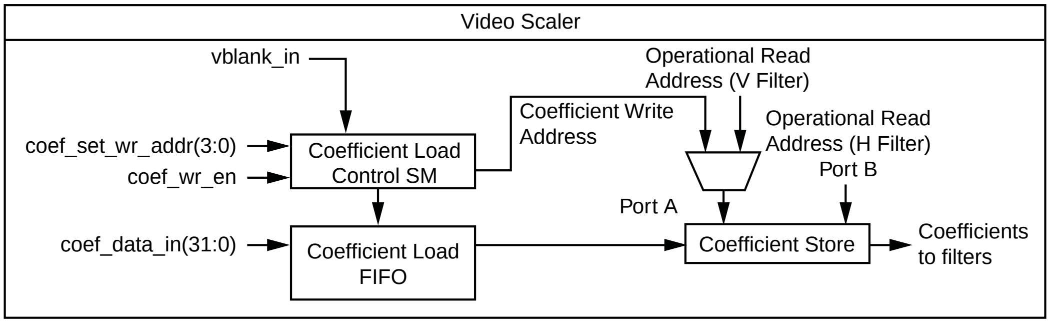


Figure 4-16: **Coefficient Write-Format on coef_data_in(31:0)**

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. Following vertical blanking, `intr_coef_fifo_rdy` will be driven High by the Video Scaler core. Following the delivery of the final coefficient of a set into the scaler, `intr_coef_fifo_rdy` will be driven Low.

An address-multiplexer is used to support the coefficient write interface as shown in Figure 4-17. The coefficient write-address is multiplexed with the coefficient read-address for the vertical filter to create the address for Port A on the dual-port coefficient RAM. Consequently, coefficients must be loaded into the coefficient stores when no active video scaling is occurring. It is only possible, therefore, to load the coefficients during the vertical blanking period. Since this would be an impossible burden on a processor, an external block RAM FIFO has been provided to which you load your coefficients during one frame period, as shown in Figure 4-17. Following a latency period after the positive transition of `vblank_in`, any new coefficient set is streamed into the internal coefficient store for use by the filter in the next frame.



UG678_7-3_081809

Figure 4-17: **Coefficient Loading Mechanism, Including External FIFO**

A waveform indicating the coefficient loading process is shown in Figure 4-18.

The coefficient memory interface is an asynchronous interface. A high level on the `coef_wr_en` signal is used to capture the coefficients delivered on `coef_data_in` as shown in Figure 4-18. An internal state-machine detects the 3rd 'clk' period when `coef_wr_en` is stable and high. At this point, the data is registered into the FIFO. Xilinx recommends that the high `coef_wr_en` pulse be no less than the equivalent of **6 'clk' periods** in duration. It is required that it also be low for a period no less than 6 'clk' periods between write operations.

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 4-16.
- All coefficients for one phase must be loaded sequentially via `coef_data_in`, 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]. See Figure 4-18. 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 user 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 the user has 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 user 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 the user has 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.

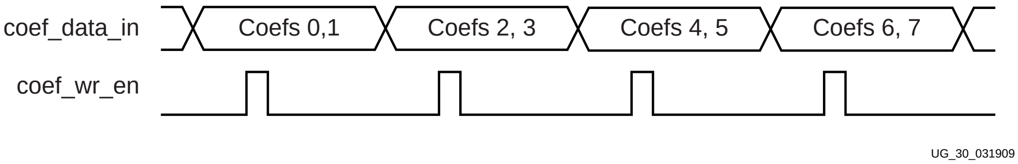


Figure 4-18: **Coefficient Loading Procedure – One Phase (8-tap Filter Shown)**

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 4-4.

Table 4-4: **Coefficient Readback Interface Signaling**

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

Table 4-4: **Coefficient Readback Interface Signaling**

Input	Description
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 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` three clock cycles later.

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. It is also important that the FrameRst pulse is allowed to occur at least once (it will occur once per frame) while control bit (3) is High.

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 4-5 shows a set of coefficients drawn from a sinc function.

Table 4-5: **Example 1 Decimal Coefficients**

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

In this example, a 32-point 1-D sinc function has been sub-sampled to generate four phases of eight coefficients each. Sub-sampling in this way usually results in a phases whose component coefficients rarely sum to 1.0 – this 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 and Filter Coefficient Calculations 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 4-6.

Table 4-6: **Example 1 Normalized Integer Coefficients**

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

It remains to format these values for the scaler.

The 16-bit coefficients must be coupled into 32-bit values for delivery to the HW. The resulting coefficient file for download is shown in Table 4-7.

The coefficients must be downloaded in the following order:

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

4. Vertical Chroma (required if not sharing H and V coefficients, and also not sharing Y and C coefficients)

Table 4-7: **Example 1 Coefficient Set Download Format**

	Horizontal Filter Coefficients for Luma			**Horizontal Filter Coefficients for Chroma**			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	1	0x00000000	(Ph0 T1 << 16) | Ph0 T0	17	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	2	0x00000000	(Ph0 T3 << 16) | Ph0 T2	18	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	3	0x00004000	(Ph0 T5 << 16) | Ph0 T4	19	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	4	0x00000000	(Ph0 T7 << 16) | Ph0 T6	20	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
Phase 1	5	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	21	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	**Phase 1**
	6	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	22	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	
	7	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	23	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	
	8	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	24	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	
Phase 2	9	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	25	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	**Phase 2**
	10	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	26	0x2C36F143	(Ph2 T3 << 16) | Ph2 T2	
	11	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	27	0xF1432C36	(Ph2 T5 << 16) | Ph2 T4	
	12	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	28	0xF9AF08D8	(Ph2 T7 << 16) | Ph2 T6	
Phase 3	13	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	29	0x06C8FB4E	(Ph3 T1 << 16) | Ph3 T0	**Phase 3**
	14	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	30	0x3D04F3CC	(Ph3 T3 << 16) | Ph3 T2	
	15	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	31	0xF7491457	(Ph3 T5 << 16) | Ph3 T4	
	16	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	32	0xFBEF058C	(Ph3 T7 << 16) | Ph3 T6	
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	33	0x00000000	(Ph0 T1 << 16) | Ph0 T0	49	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	34	0x00000000	(Ph0 T3 << 16) | Ph0 T2	50	0x00000000	(Ph0 T3 << 16) | Ph0 T2	
	35	0x00004000	(Ph0 T5 << 16) | Ph0 T4	51	0x00004000	(Ph0 T5 << 16) | Ph0 T4	
	36	0x00000000	(Ph0 T7 << 16) | Ph0 T6	52	0x00000000	(Ph0 T7 << 16) | Ph0 T6	
Phase 1	37	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	53	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	**Phase 1**
	38	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	54	0x1457F749	(Ph1 T3 << 16) | Ph1 T2	
	39	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	55	0xF3CC3D04	(Ph1 T5 << 16) | Ph1 T4	
	40	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	56	0xFB4E06C8	(Ph1 T7 << 16) | Ph1 T6	

Table 4-7: **Example 1 Coefficient Set Download Format *(Cont'd)***

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 4-8 shows the same 4-phase coefficient set loaded into the scaler when `num_h_phases` = 4, `num_v_phases` = 4 and `max_phases` is greater than 4 (`max_phases` = 5, 6, 7 or 8, `num_h_taps` = 8, `num_v_taps` = 8).

Note that:

1. If `max_phases` is not equal to an integer power of 2, then the number of phases to be loaded is rounded up to the next integer power of 2. See Example 2 (Table 4-8). Unused phases should be loaded with zeros.
2. The number of values loaded per phase is ***not*** rounded to the nearest power of 2. See Example 3 (Table 4-11).

Table 4-8: **Example 2 Coefficient Set Download Format**

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

Table 4-8: **Example 2 Coefficient Set Download Format *(Cont'd)***

Phase 2	9	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	41	0x08D8F9AF	(Ph2 T1 << 16) | Ph2 T0	**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	
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	
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**			
	Addr	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Addr**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	65	0x00000000	(Ph0 T1 << 16) | Ph0 T0	97	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**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	

Table 4-8: **Example 2 Coefficient Set Download Format *(Cont'd)***

Phase 1	69	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	101	0x058CFBEF	(Ph1 T1 << 16) | Ph1 T0	**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	
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 4-9 (horizontal) and Table 4-10 (vertical).

Table 4-9: **Example 9-Tap Coefficients**

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

Table 4-10: **Example 7-Tap Coefficients**

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

The resulting coefficient file for download is shown in Table 4-11.

Table 4-11: **Example 3 Coefficient Set Download Format**

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

Table 4-11: **Example 3 Coefficient Set Download Format *(Cont'd)***

Phase 3	16	0x017EFF9E	(Ph3 T1 << 16) | Ph3 T0	36	0x017EFF9E	(Ph3 T1 << 16) | Ph3 T0	**Phase 3**
	17	0x3619F93F	(Ph3 T3 << 16) | Ph3 T2	37	0x3619F93F	(Ph3 T3 << 16) | Ph3 T2	
	18	0XF84614D7	(Ph3 T1 << 16) | Ph3 T4	38	0XF84614D7	(Ph3 T1 << 16) | Ph3 T4	
	19	0XFF1B0312	(Ph3 T1 << 16) | Ph3 T6	39	0XFF1B0312	(Ph3 T1 << 16) | Ph3 T6	
	20	0x00000043	(0 << 16) | Ph3 T8	40	0x00000043	(0 << 16) | Ph3 T8	
	Vertical Filter Coefficients for Luma			**Vertical Filter Coefficients for Chroma**			
	Load Sequence Number	**Value**	**Calculation** Ph= Phase #, T= Tap #	**Load Sequence Number**	**Value**	**Calculation** Ph= Phase #, T= Tap #	
Phase 0	41	0x00000000	(Ph0 T1 << 16) | Ph0 T0	61	0x00000000	(Ph0 T1 << 16) | Ph0 T0	**Phase 0**
	42	0x40000000	(Ph0 T3 << 16) | Ph0 T2	62	0x40000000	(Ph0 T3 << 16) | Ph0 T2	
	43	0x00000000	(Ph0 T5 << 16) | Ph0 T4	63	0x00000000	(Ph0 T5 << 16) | Ph0 T4	
	44	0x00000000	(0 << 16) | Ph0 T6	64	0x00000000	(0 << 16) | Ph0 T6	
	45	0x00000000	N/A dummy coef	65	0x00000000	N/A dummy coef	
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 Constant mode), you must specify, using the CORE Generator GUI or the EDK 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 and Filter Coefficient Calculations for more information on generating video scaler coefficients.

Extracting Coefficients From xscaler_coefs.c File

The pCore 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. 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 4-12. This function may be used independently for all Horizontal, Vertical, Luma, and Chroma filter operations.

Table 4-12: **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
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 4-13: **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 4-14 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 4-14: **.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,

Table 4-14: **.coe File Example 1 *(Cont'd)***

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
0	9	12	-616,	1111110110011000,
0	10	13	0,	0000000000000000,
0	11	14	85,	0000000001010101,
1	0	15	28,	0000000000011100,
1	1	16	155,	0000000010011011,
1	2	17	-186,	1111111101000110,
1	3	18	-1062,	1111101111001010,
1	4	19	960,	0000001111000000,
1	5	20	6311,	0001100010100111,
1	6	21	7842,	0001111010100010,
1	7	22	3246,	0000110010101110,
1	8	23	-538,	1111110111100110,
1	9	24	-518,	1111110111111010,
1	10	25	72,	0000000001001000,
1	11	26	73,	0000000001001001,
2	0	27	53,	0000000000110101,
2	1	28	125,	0000000001111101,
2	2	29	-366,	1111111010010010,
2	3	30	-890,	1111110010000110,
2	4	31	2060,	0000100000001100,
2	5	32	7209,	0001110000101001,
2	6	33	7209,	0001110000101001,
2	7	34	2060,	0000100000001100,
2	8	35	-890,	1111110010000110,
2	9	36	-366,	1111111010010010,
2	10	37	125,	0000000001111101,
2	11	38	53,	0000000000110101,
3	0	39	73,	0000000001001001,
3	1	40	72,	0000000001001000,
3	2	41	-518,	1111110111111010,
3	3	42	-538,	1111110111100110,
3	4	43	3246,	0000110010101110,
3	5	44	7842,	0001111010100010,

Table 4-14: **.coe File Example 1 *(Cont’d)***

Phase	Tap	File Line-number	Line Text (Signed Decimal Form)	Line Text (16-bit Binary Form)
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 4-15 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 4-15: **.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,

Table 4-15: **.coe File Example 2 *(Cont'd)***

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

Table 4-15: **.coe File Example 2 *(Cont'd)***

1	3 (VC)	3	1	376	72,
1	3 (VC)	3	…	…	…
1	3 (VC)	3	11	386	28;
-	-	-	-	387	“”

Table 4-16 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 4-16: **.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,	

Table 4-16: **.coe File Example 3 *(Cont'd)***

1 (V)	0	2	21	86,	
1 (V)	-	-	22	0,	Padding value
1 (V)	1	0	23	512,	
1 (V)	1	1	24	16068,	
1 (V)	1	2	25	-197,	
1 (V)	-	-	26	0,	Padding value
1 (V)	2	0	27	1243,	
1 (V)	2	1	28	15539,	
1 (V)	2	2	29	-398,	
1 (V)	-	-	30	0,	Padding value
1 (V)	3	0	31	2829,	
1 (V)	3	1	32	14099,	
1 (V)	3	2	33	-544,	
1 (V)	-	-	34	0;	Padding value
-	-	-	35	“”	

Control Values

There follows a brief description of the function of the control values.

In GPP mode and pCore mode, these values are provided as dynamic inputs, and may be changed during runtime – the user inputs become active once per frame after completion of an output frame, using an internal active value capture register.

For the pCore version of the core, CORE Generator software provides the GPP core placed in a wrapper which allows you to parameterize the scaler core in EDK. The ports are driven by registers that sit on the AXI4-Lite. The address is decoded in the wrapper. A MicroBlaze™ processor software driver is provided in source-code form to drive these ports. Typical usage of the pCore is shown in Figure 4-19.

- **`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

- **`output_h_size`**, **`output_v_size`**

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

- **`hsf`**, **`vsf`**

These are the horizontal and vertical shrink-factors that must be supplied the user. They should be supplied as integers, and can typically be calculated as follows:

$$hsf = round([\frac{1 + aperture\_end\_pixel - aperture\_start\_pixel}{output\_h\_size}] * 2^{20})$$

and

$$vsf = round([\frac{1 + aperture\_end\_line - aperture\_start\_line}{output\_v\_size}] * 2^{20})$$

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 described below.

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

- **num_h_phases, num_v_phases**

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. Non power-of-two numbers of phases are supported.

- **coef_wr_addr, h_coeff_set, v_coeff_set**

In GPP and pCore interfaces, you may load coefficients. The scaler can store up to max_coef_sets coefficient sets internally. coef_wr_addr sets the set location of the set to which you intend to write. The set may subsequently be used by controlling the h_coeff_set and v_coeff_set values.

- **start_hpa_y, start_hpa_c, start_vpa_y, start_vpa_c**

These are the start-phase controls. Internally to the core, the scaler accumulates the 24-bit shrink-factor (hsf, vsf) to determine phase and filter aperture. 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 cosited. 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.

- **control**

The control register contains only two active bits. The default value for the control register during continuous operation is "0x3."

- **`bit 0`** is a general purpose enable. Activated/deactivated on a `vblank_in` basis, a value of 0 disables the scaler output.
- **`bit 1`** enables values on the other register inputs to become internally active on a `vblank_in` basis. A value of 0 prevents the active internal values from being changed.

Constant (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

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.
- 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.

General Purpose Processor (GPP) Interface

This interface type exposes all control ports to the user. You are responsible for driving these ports. Xilinx recommends that GPP mode be used only by experienced scaler users.

Figure 4-19 indicates how the EDK pCore is effectively a wrapper around the GPP mode core. This should be considered as an example of how you may choose to wrap the GPP mode core to suit any processor.

In GPP mode, the control values may be changed during runtime – the user input control values become active once per frame after completion of an output frame, using an internal active value capture register.

Interrupts

There are six interrupts:

1. **intr_output_frame_done** – Issued once per complete output frame.
2. **intr_reg_update_done** – Issued during Vertical blanking when the register values have been transferred to the active registers.
3. **intr_input_error** – Issued if `active_video_in` is asserted before the scaler is ready to receive a new line.
4. **intr_output_error** – Issued if frame period completes before full output frame has been delivered.
5. **intr_coef_wr_error** – Issued if coefficient is written into coefficient FIFO when the FIFO is not ready.
6. **intr_coef_fifo_rdy** – High when the coefficient FIFO is ready to receive a coefficient for the current set; stays low once a full set has been written into FIFO; sent high during Vertical blanking.
7. **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 **GPP mode**, all seven interrupts are active.

In **Constant mode**, only `intr_input_error`, `intr_output_error` and `intr_output_frame_done` are active.

Inside the **pCore** wrapper, an Interrupt Controller (Xilinx Interrupt Control LogiCORE™ (DS516)) collates these interrupts into one interrupt on the AXI4-Lite bus. The microprocessor must then read the interrupt status registers to establish the nature of the interrupt. The interrupt registers are defined in Chapter 2, Core Interfaces and Register Space. A generic n-peripheral system is shown in Figure 4-19. It shows the intended usage of interrupts in an EDK-based system. It also shows how the Xilinx Interrupt Controller is used internally to the pCore along with the scaler in GPP mode.

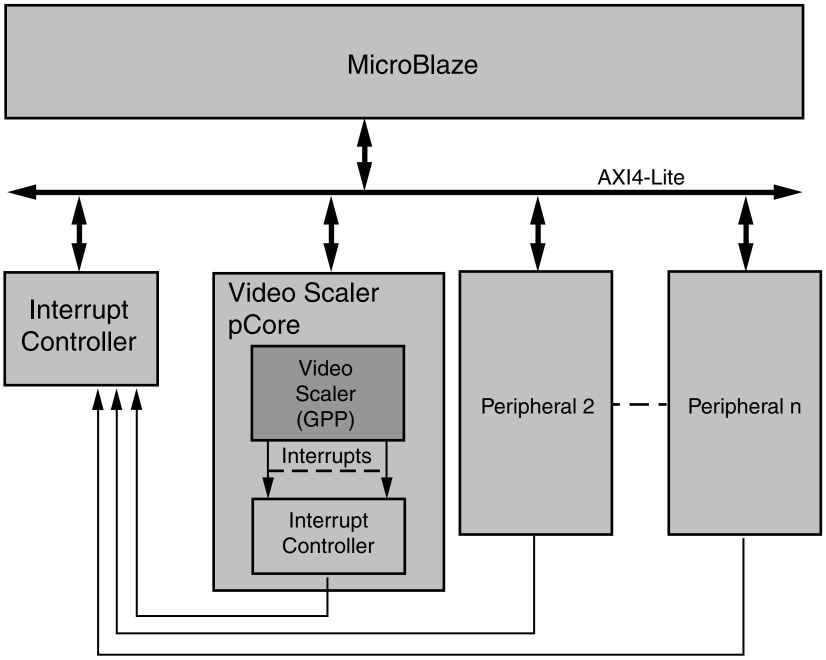


Figure 4-19: **Typical EDK-based System Showing Interrupt Structure**

Resets

The Video Scaler core has one reset (`sclr`) that is used for the entire core. In the GPP and Constant versions of the core (not EDK pCore), the signal is exposed to the user and is active High. For the pCore version, an internal software reset drives this signal (active Low).

Protocol Description

For the pCore version of the Video Scaler core, the register interface is compliant with the AXI4-Lite interface. The video output interface is compliant with AXI4-Stream protocol. In Memory mode, the input video interface is also compliant with AXI4-Stream protocol.

Evaluation Core Timeout

When generated with a Evaluation Hardware license, the core includes a timeout circuit that disables the core after a specific period of time. The timeout circuit can only be reset by reloading the FPGA bitstream. The timeout period for this core is set to approximately eight hours for a 75 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 four hours.

After the timeout period has expired, video output will no longer be available at the outputs of the core.

Chapter 5

Constraining the Core

This chapter contains applicable constraints for the Video Scaler core.

Required Constraints

There are no required constraints for the Video Scaler core.

Device, Package, and Speed Grade Selections

Device, package and speed grade should be selected according to the worst-case throughput scenario required by the user. Typically, this depends on scale factor and image size. For more information, see Performance in Chapter 1. This core is not characterized for lower power devices.

Clock Frequencies

The core clock (`clk`), the video input clock (`video_in_clk`) and the video output clock (`video_out_clk`) 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 1.

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 the Video Scaler core.

Banking

There are no specific banking requirements for the Video Scaler core.

Transceiver Placement

The Video Scaler includes no transceivers.

I/O Standard and Placement

There are no specific I/O standards or placement requirements for the Video Scaler core.

Chapter 6

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
- Inclusion of the video scaler in an EDK project
- Typical usage of video scaler in conjunction with other cores
- System level distribution of video timing and genlock signals

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 simplify the VFBC interface, and act as a SW-controllable processor peripheral.
- The Timebase Controller is a SW-configurable timing detector and generator block, which generates timing signals for distribution around the system. See PG016, *LogiCORE IP Timing Controller Product Guide* for more information.
- 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.

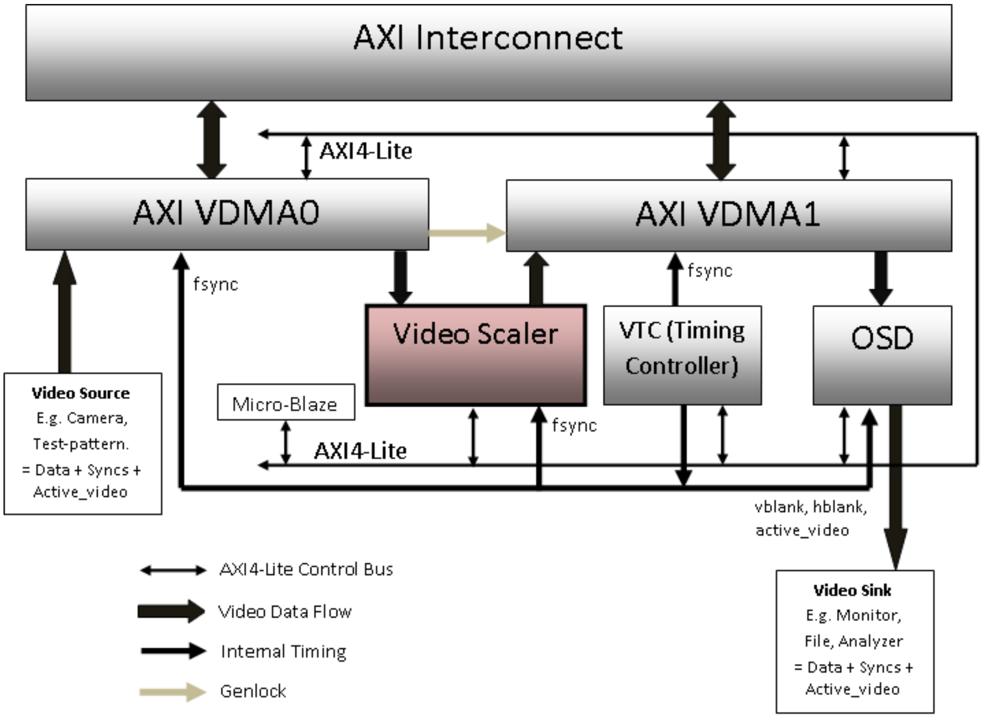


Figure 6-1: **Simplified System Diagram**

Control Buses

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

In the example in EDK MHS File Text, AXI_VDMA0 is sourced from an engineering test-pattern generator (not included in the MHS file below). In the example in EDK MHS File Text, data is passed between IP and AXI_VDMA0 using AXI4-Stream.

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.

In the example in EDK MHS File Text, data is passed between IP and AXI_VDMA0 using AXI4-Stream.

Video Scaler Configuration

The Video Scaler core is configured as follows:

- Single-engine 4:2:2
- 11Hx11V-taps
- 64 phases
- Shared YC coefficients
- Memory mode

The core uses a 148.5 MHz derivative of the 74.25 MHz input 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.

EDK MHS File Text

This section contains an example EDK MHS file insert for the system described.

Note: This is not a complete design, but gives some idea as to the construction of a Video Scaler system in EDK.

```
BEGIN axi_vdma
 PARAMETER INSTANCE = axi_vdma_0
 PARAMETER HW_VER = 4.00.a
 PARAMETER C_BASEADDR = 0x7e220000
 PARAMETER C_HIGHADDR = 0x7e22ffff
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MM2S_GENLOCK_MODE = 1
 PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
 PARAMETER C_MM2S_MAX_BURST_LENGTH = 256
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 256
 PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 16
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_INCLUDE_SG = 0
 PARAMETER C_S2MM_LINEBUFFER_THRESH = 2560
 PARAMETER C_INCLUDE_MM2S_SF = 1
 PARAMETER C_INCLUDE_S2MM_SF = 1
 PARAMETER C_MM2S_LINEBUFFER_THRESH = 1920
 PARAMETER C_FLUSH_ON_FSYNC = 1
 BUS_INTERFACE M_AXIS_MM2S = axi_vdma_0_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_S2MM = tpg_0_M_AXIS_S2MM
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_MM2S = axi4_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 PORT s2mm_fsync = xsvi2axi_0_fsync_out
 PORT s2mm_frame_ptr_out = axi_vdma_0_s2mm_frame_ptr_out
 PORT mm2s_frame_ptr_in = axi_vdma_0_s2mm_frame_ptr_out
 PORT mm2s_introut = axi_vdma_0_mm2s_introut
 PORT s2mm_introut = axi_vdma_0_s2mm_introut
 PORT s_axi_lite_aclk = clk_50_0000MHzMMCM0
 PORT m_axi_mm2s_aclk = clk_200_0000MHzMMCM0
 PORT m_axi_s2mm_aclk = clk_200_0000MHzMMCM0
 PORT s_axis_s2mm_aclk = vid_in_clk
 PORT m_axis_mm2s_aclk = clk_75_0000MHzMMCM0
 PORT s2mm_prmry_reset_out_n = XSVI2AXI_S2MM_RESET_OUT_N
END

BEGIN axi_vtc
 PARAMETER INSTANCE = axi_vtc_1
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_BASEADDR = 0x7ee00000
 PARAMETER C_HIGHADDR = 0x7ee0ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE XSVI_OUT = axi_vtc_1_XSVI_OUT
 PORT video_clk_in = clk_75_0000MHzMMCM0
 PORT IP2INTC_Irpt = timebase_1_IP2INTC_Irpt
 PORT fsync = axi_vtc_1_fsync
END

BEGIN axi_scaler
 PARAMETER INSTANCE = axi_scaler_0
 PARAMETER HW_VER = 5.00.a
 PARAMETER C_SEPARATE_YC_COEFS = 1
 PARAMETER C_MAX_SAMPLES_OUT_PER_LINE = 1280
 PARAMETER C_MAX_PHASES = 64
 PARAMETER C_INIT_COEF_SOURCE = 1
 PARAMETER C_YC_FILTER_CONFIG = 1
 PARAMETER C_NUMBER_OF_H_TAPS = 11
 PARAMETER C_NUMBER_OF_V_TAPS = 11
 PARAMETER C_MAX_COEF_SETS = 16
 PARAMETER C_S_AXIS_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_TDATA_WIDTH = 16
 PARAMETER C_BASEADDR = 0x7c800000
 PARAMETER C_HIGHADDR = 0x7c80ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE S_AXIS = axi_vdma_0_M_AXIS_MM2S
 BUS_INTERFACE M_AXIS = axi_scaler_0_M_AXIS
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT video_in_clk = clk_75_0000MHzMMCM0
 PORT video_out_clk = clk_75_0000MHzMMCM0
 PORT clk = clk_150_0000MHzMMCM0
 PORT vsync_i = axi_vtc_1_fsync
 PORT IP2INTC_Irpt = axi_scaler_0_IP2INTC_Irpt
END


BEGIN axi_vdma
 PARAMETER INSTANCE = axi_vdma_1
 PARAMETER HW_VER = 3.01.a
 PARAMETER C_USE_FSYNC = 1
 PARAMETER C_MM2S_GENLOCK_MODE = 1
 PARAMETER C_MM2S_LINEBUFFER_DEPTH = 4096
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 4096
 PARAMETER C_MM2S_MAX_BURST_LENGTH = 256
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 256
 PARAMETER C_MM2S_LINEBUFFER_THRESH = 2560
 PARAMETER C_PRMRY_IS_ACLK_ASYNC = 1
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_M_AXIS_MM2S_TDATA_WIDTH = 16
 PARAMETER C_NUM_FSTORES = 5
 PARAMETER C_INCLUDE_SG = 0
 PARAMETER C_INCLUDE_MM2S_DRE = 1
 PARAMETER C_INCLUDE_S2MM_DRE = 1
 PARAMETER C_INCLUDE_MM2S_SF = 1
 PARAMETER C_INCLUDE_S2MM_SF = 1
 PARAMETER C_MM2S_GENLOCK_NUM_MASTERS = 2
 PARAMETER C_S2MM_LINEBUFFER_THRESH = 2560
 PARAMETER C_BASEADDR = 0x7e200000
 PARAMETER C_HIGHADDR = 0x7e20ffff
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_MM2S = axi4_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 BUS_INTERFACE M_AXIS_MM2S = axi_vdma_1_M_AXIS_MM2S
 BUS_INTERFACE S_AXIS_S2MM = axi_scaler_0_M_AXIS
 PORT s_axi_lite_aclk = clk_50_0000MHzMMCM0
 PORT m_axi_mm2s_aclk = clk_75_0000MHzMMCM0
 PORT m_axi_s2mm_aclk = clk_75_0000MHzMMCM0
 PORT mm2s_fsync = axi_vtc_1_fsync
 PORT s2mm_fsync = axi_vtc_1_fsync
 PORT s2mm_frame_ptr_out = axi_vdma_1_s2mm_frame_ptr_out
 PORT mm2s_frame_ptr_in = axi_vdma_0_s2mm_frame_ptr_out &
axi_vdma_1_s2mm_frame_ptr_out
 PORT mm2s_introut = axi_vdma_1_mm2s_introut
 PORT s2mm_introut = axi_vdma_1_s2mm_introut
 PORT s2mm_buffer_full = axi_vdma_1_s2mm_buffer_full
 PORT s2mm_buffer_almost_full = axi_vdma_1_s2mm_buffer_almost_full
END

BEGIN axi_osd
 PARAMETER INSTANCE = osd_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_LAYER1_TYPE = 1
 PARAMETER C_NUM_LAYERS = 2
 PARAMETER C_LAYER1_IMEM_SIZE = 96
 PARAMETER C_NUM_DATA_CHANNELS = 2
 PARAMETER C_ALPHA_CHANNEL_EN = 0
 PARAMETER C_OUTPUT_MODE = 1
 PARAMETER C_BASEADDR = 0x73a00000
 PARAMETER C_HIGHADDR = 0x73a0ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE S0_AXIS = axi_vdma_1_M_AXIS_MM2S
 BUS_INTERFACE XSVI_IN = axi_vtc_1_XSVI_OUT
 BUS_INTERFACE XSVI_OUT = osd_0_XSVI_OUT
 PORT clk = clk_75_0000MHzMMCM0
 PORT IP2INTC_Irpt = osd_0_IP2INTC_Irpt
END

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi4lite_0
 PARAMETER HW_VER = 1.03.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
 PORT INTERCONNECT_ACLK = clk_50_0000MHzMMCM0
END

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.20.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 0
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0xc0000000
 PARAMETER C_ICACHE_HIGHADDR = 0xcfffffff
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0xc0000000
 PARAMETER C_DCACHE_HIGHADDR = 0xcfffffff
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 PARAMETER C_INTERCONNECT_M_AXI_DC_AW_REGISTER = 0
 PARAMETER C_INTERCONNECT_M_AXI_DC_W_REGISTER = 0
 PARAMETER C_M_AXI_D_BUS_EXCEPTION = 1
 PARAMETER C_M_AXI_I_BUS_EXCEPTION = 1
 PARAMETER C_ILL_OPCODE_EXCEPTION = 1
 PARAMETER C_UNALIGNED_EXCEPTIONS = 1
 PARAMETER C_OPCODE_0x0_ILLEGAL = 1
 PARAMETER C_USE_STACK_PROTECTION = 1
 BUS_INTERFACE M_AXI_DP = axi4lite_0
 BUS_INTERFACE M_AXI_DC = axi4_0
 BUS_INTERFACE M_AXI_IC = axi4_0
 BUS_INTERFACE DEBUG = microblaze_0_debug
 BUS_INTERFACE DLMB = microblaze_0_dlmb
 BUS_INTERFACE ILMB = microblaze_0_ilmb
 PORT MB_RESET = proc_sys_reset_0_MB_Reset
 PORT CLK = clk_100_0000MHzMMCM0
 PORT INTERRUPT = microblaze_0_interrupt
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_ilmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzMMCM0
END

BEGIN lmb_v10
```

```
 PARAMETER INSTANCE = microblaze_0_dlmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzMMCM0
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_i_bram_ctrl
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = microblaze_0_ilmb
 BUS_INTERFACE BRAM_PORT =
microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_d_bram_ctrl
 PARAMETER HW_VER = 3.00.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = microblaze_0_dlmb
 BUS_INTERFACE BRAM_PORT =
microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN bram_block
 PARAMETER INSTANCE = microblaze_0_bram_block
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA =
microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
 BUS_INTERFACE PORTB =
microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT Ext_Reset_In = RESET
 PORT MB_Reset = proc_sys_reset_0_MB_Reset
 PORT Slowest_sync_clk = clk_50_0000MHzMMCM0
 PORT Interconnect_aresetn = proc_sys_reset_0_Interconnect_aresetn
 PORT Dcm_locked = proc_sys_reset_0_Dcm_locked
 PORT MB_Debug_Sys_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT BUS_STRUCT_RESET = proc_sys_reset_0_BUS_STRUCT_RESET
END

BEGIN axi_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x83000000
 PARAMETER C_HIGHADDR = 0x8300ffff
 PARAMETER C_INTERCONNECT_S_AXI_MASTERS = plbv46_axi_bridge_0.M_AXI
 BUS_INTERFACE S_AXI = axi_interconnect_0
 PORT RX = fpga_0_RS232_Uart_1_RX_pin
 PORT TX = fpga_0_RS232_Uart_1_TX_pin
 PORT Interrupt = RS232_Uart_1_Interrupt
 PORT S_AXI_ACLK = clk_100_0000MHzMMCM0
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 4.02.a
 PARAMETER C_CLKIN_FREQ = 200000000
 PARAMETER C_CLKOUT0_FREQ = 150000000
 PARAMETER C_CLKOUT0_GROUP = MMCM0
 PARAMETER C_CLKOUT1_FREQ = 200000000
 PARAMETER C_CLKOUT1_GROUP = MMCM0
 PARAMETER C_CLKOUT2_FREQ = 400000000
 PARAMETER C_CLKOUT2_GROUP = MMCM0
 PARAMETER C_CLKOUT3_FREQ = 400000000
 PARAMETER C_CLKOUT3_GROUP = MMCM0
 PARAMETER C_CLKOUT3_BUF = FALSE
 PARAMETER C_CLKOUT3_VARIABLE_PHASE = TRUE
 PARAMETER C_CLKOUT4_FREQ = 50000000
 PARAMETER C_CLKOUT4_GROUP = MMCM0
 PARAMETER C_CLKOUT5_FREQ = 75000000
 PARAMETER C_CLKOUT5_GROUP = MMCM0
 PARAMETER C_CLKOUT6_FREQ = 150000000
 PARAMETER C_CLKOUT6_GROUP = MMCM0
 PORT RST = RESET
 PORT CLKIN = CLK
 PORT CLKOUT0 = clk_100_0000MHzMMCM0
 PORT CLKOUT1 = clk_200_0000MHzMMCM0
 PORT CLKOUT2 = clk_400_0000MHzMMCM0
 PORT CLKOUT3 = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT CLKOUT4 = clk_50_0000MHzMMCM0
 PORT CLKOUT5 = clk_75_0000MHzMMCM0
 PORT CLKOUT6 = clk_150_0000MHzMMCM0
 PORT LOCKED = proc_sys_reset_0_Dcm_locked
 PORT PSCLK = clk_50_0000MHzMMCM0
 PORT PSEN = psen
 PORT PSINCDEC = psincdec
 PORT PSDONE = psdone
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_UART = 1
 PARAMETER C_BASEADDR = 0x74800000
 PARAMETER C_HIGHADDR = 0x7480ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_debug
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT Debug_SYS_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
END

BEGIN axi_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 1
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT TX = RS232_Uart_1_sout
 PORT RX = RS232_Uart_1_sin
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
END

BEGIN axi_v6_ddrx
 PARAMETER INSTANCE = DDR3_SDRAM
 PARAMETER HW_VER = 1.03.a
 PARAMETER C_MEM_PARTNO = MT4JSF6464HY-1G1
 PARAMETER C_INTERCONNECT_S_AXI_AR_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_AW_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_R_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_W_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_B_REGISTER = 1
 PARAMETER C_INTERCONNECT_S_AXI_READ_ACCEPTANCE = 8
 PARAMETER C_INTERCONNECT_S_AXI_WRITE_ACCEPTANCE = 8
 PARAMETER C_S_AXI_DATA_WIDTH = 128
 PARAMETER C_INTERCONNECT_S_AXI_MASTERS = microblaze_0.M_AXI_DC &
microblaze_0.M_AXI_IC & axi_vdma_0.M_AXI_MM2S & axi_vdma_0.M_AXI_S2MM &
axi_vdma_1.M_AXI_MM2S & axi_vdma_1.M_AXI_S2MM
 PARAMETER C_MMCM_EXT_LOC = MMCM_ADV_X0Y8
 PARAMETER C_S_AXI_BASEADDR = 0xc0000000
 PARAMETER C_S_AXI_HIGHADDR = 0xcfffffff
 BUS_INTERFACE S_AXI = axi4_0
 PORT ddr_ck_p = ddr_memory_clk
 PORT ddr_ck_n = ddr_memory_clk_n
 PORT ddr_cke = ddr_memory_cke
 PORT ddr_cs_n = ddr_memory_cs_n
 PORT ddr_odt = ddr_memory_odt
 PORT ddr_ras_n = ddr_memory_ras_n
 PORT ddr_cas_n = ddr_memory_cas_n
 PORT ddr_we_n = ddr_memory_we_n
 PORT ddr_dm = ddr_memory_dm
 PORT ddr_ba = ddr_memory_ba
 PORT ddr_addr = ddr_memory_addr
 PORT ddr_reset_n = ddr_memory_ddr3_rst
 PORT ddr_dq = ddr_memory_dq
 PORT ddr_dqs_p = ddr_memory_dqs
 PORT ddr_dqs_n = ddr_memory_dqs_n
 PORT clk = clk_200_0000MHzMMCM0
 PORT clk_ref = clk_200_0000MHzMMCM0
 PORT clk_mem = clk_400_0000MHzMMCM0
 PORT clk_rd_base = clk_400_0000MHzMMCM0_nobuf_varphase
 PORT PD_PSEN = psen
 PORT PD_PSINCDEC = psincdec
 PORT PD_PSDONE = psdone
END

BEGIN axi_intc
 PARAMETER INSTANCE = microblaze_0_intc
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x41200000
 PARAMETER C_HIGHADDR = 0x4120ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT IRQ = microblaze_0_interrupt
 PORT S_AXI_ACLK = clk_50_0000MHzMMCM0
 PORT INTR = osd_0_IP2INTC_Irpt & axi_vdma_0_mm2s_introut &
axi_vdma_0_s2mm_introut & timebase_1_IP2INTC_Irpt &
axi_scaler_0_IP2INTC_Irpt & axi_vdma_1_mm2s_introut &
axi_vdma_1_s2mm_introut
END
```

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

The use of the parameters aperture_start_pixel, aperture_end_pixel, aperture_start_line, and aperture_end_line is limited to the Live Mode operation of the core. These examples show the use of these parameters with the scaler in Live Mode.

When in Memory Mode (as in the example design) cropping an area of the image in memory must be achieved by controlling a memory controller block such as the AXI-VDMA, but using aperture pixel/line start/end values such as are shown in these examples, for addressing the memory at the exact locations.

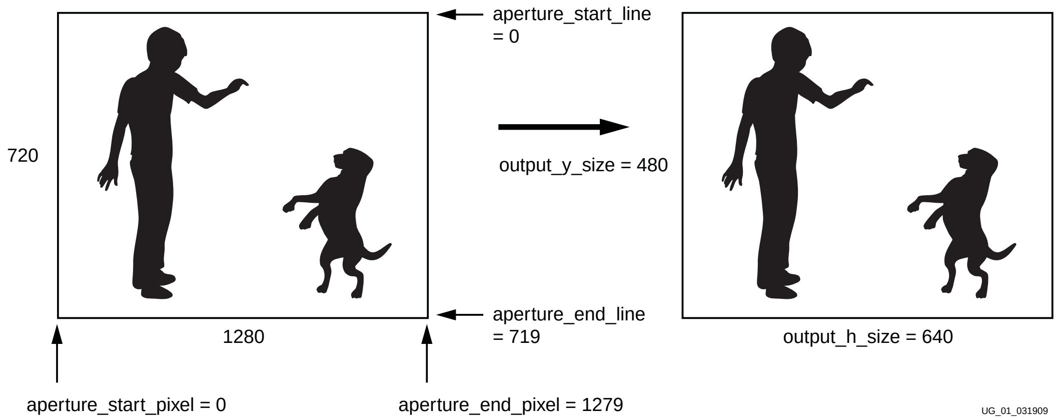


Figure 6-2: **Format Down-scaling. Example 720p to 640x480, HSF = 2^{20} x 1280/640; VSF = 2^{20} x 720/480**

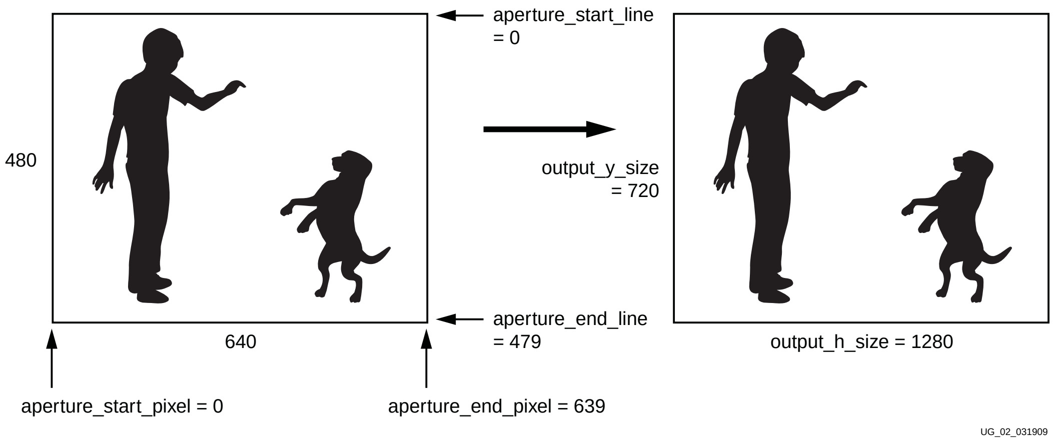


Figure 6-3: **Format Up-scaling. Example 640x480 to 720p, HSF = 2^{20} x 640/1280; 2^{20} x VSF = 480/720**

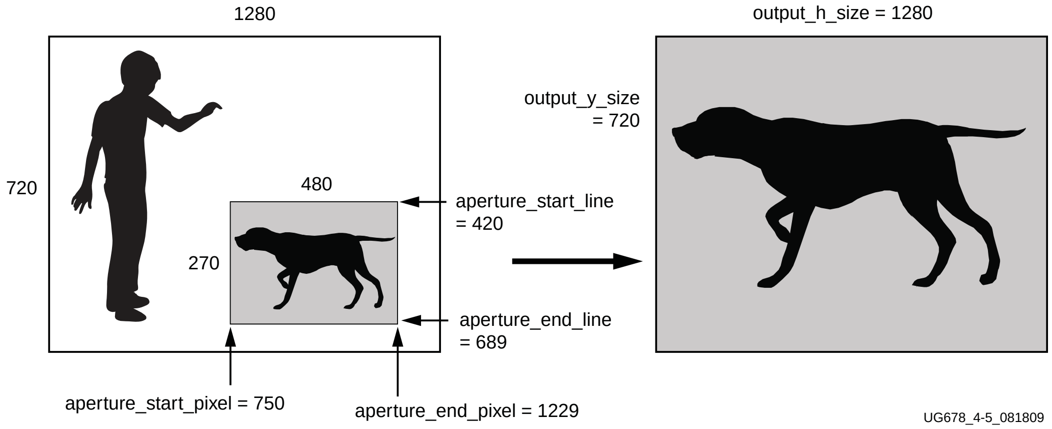


Figure 6-4: **Zoom (Up-scaling), HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

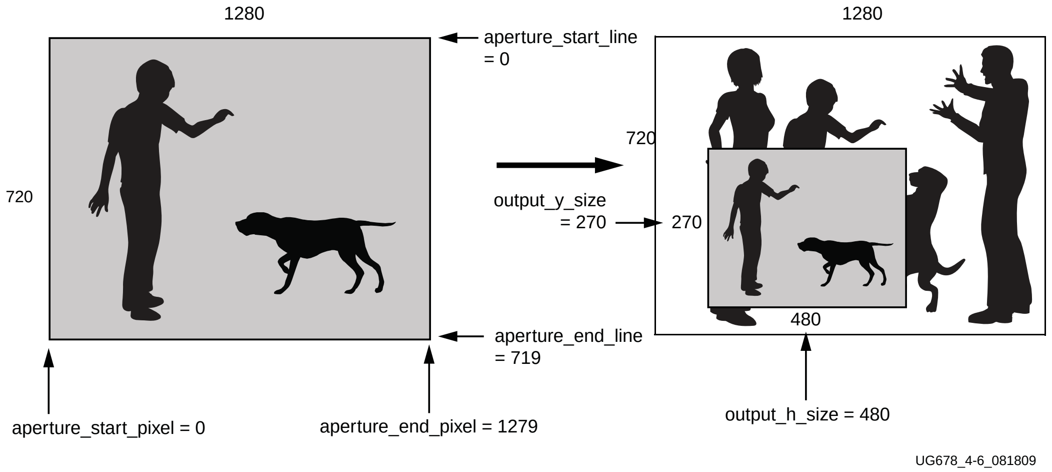


Figure 6-5: **Shrink (Down-scaling). Example for Picture-in-Picture (PinP), HSF = 2^{20} x 1280/480; VSF = 2^{20} x 720/270**

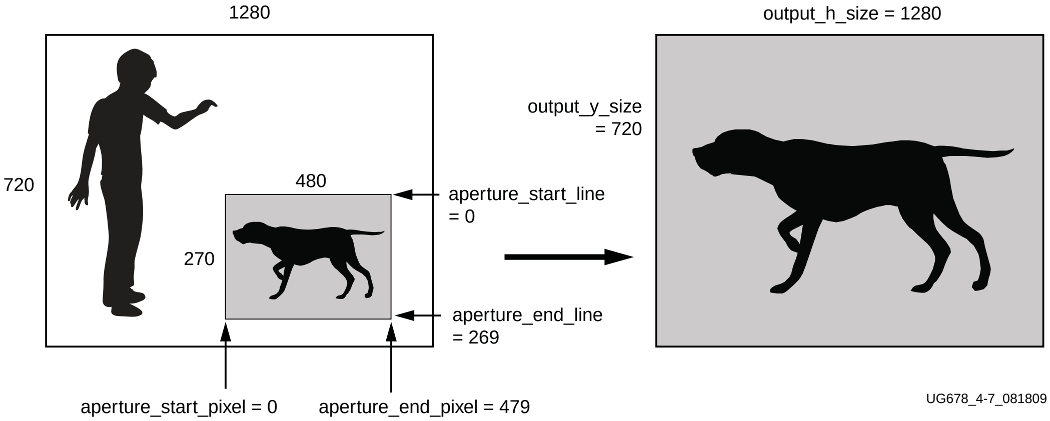


Figure 6-6: **Zoom (Up-scaling) reading from External Memory, HSF = 2^{20} x 480/1280; VSF = 2^{20} x 270/720**

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

This appendix describes migrating from older versions of the IP to the current IP release.

Migrating to the EDK pCore AXI4-Lite Interface

The Video Scaler v4.0 changed from the PLB processor interface to the EDK pCore AXI4-Lite interface. As a result, all of the PLB-related connections have been replaced with an AXI4-Lite interface. For more information, see UG761, *Xilinx AXI Reference Guide*.

Migrating to the AXI4-Stream Interface

The Video Scaler v5.0 core changed from the Video Frame Buffer Controller (VFBC) native interfaces to the AXI4-Stream interfaces. As a result, all of the VFBC-related connections have been replaced with AXI4-Stream connections. For more information, see UG761, *Xilinx AXI Reference Guide*.

Parameter Changes in the XCO File

The XCO file now includes AXI4-Stream IO data widths: `s_axis_tdata_width` and `m_axis_tdata_width`.

Port Changes

The Video Scaler v5.0 core now includes AXI4-Stream IO interface signals. It no longer includes VDMA IO signals.

Functionality Changes

There are no functional changes that have been added the core for v5.0.

Appendix C

Debugging

Some general debugging tips are as follows:

- Verify that the Version register can be read properly. See Table 2-7, 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.
- When using Memory mode, verify that the `vsync_in` input is being properly driven.
- When using Live mode, verify that the `vblank_in`, `hblank_in` and `active_video` inputs are being properly driven.
- Verify that the output interface is not holding off permanently. The `m_axis_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_tvalid` output. If this is occurring, check the data output.
- If `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 has run out of time. Upon receiving a `vblank/vsync`, it has not yet completed the scaling of the previous frame. In this case, revisit the calculations of clock frequency, scale-factor and others. If using Y/C with a single engine, then using two engines may remedy this issue.
- 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.

Application Software Development

Introduction

This appendix provides a description of how to program and control the data flow for the video scaler hardware 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.

Video Scaler Flow Diagram

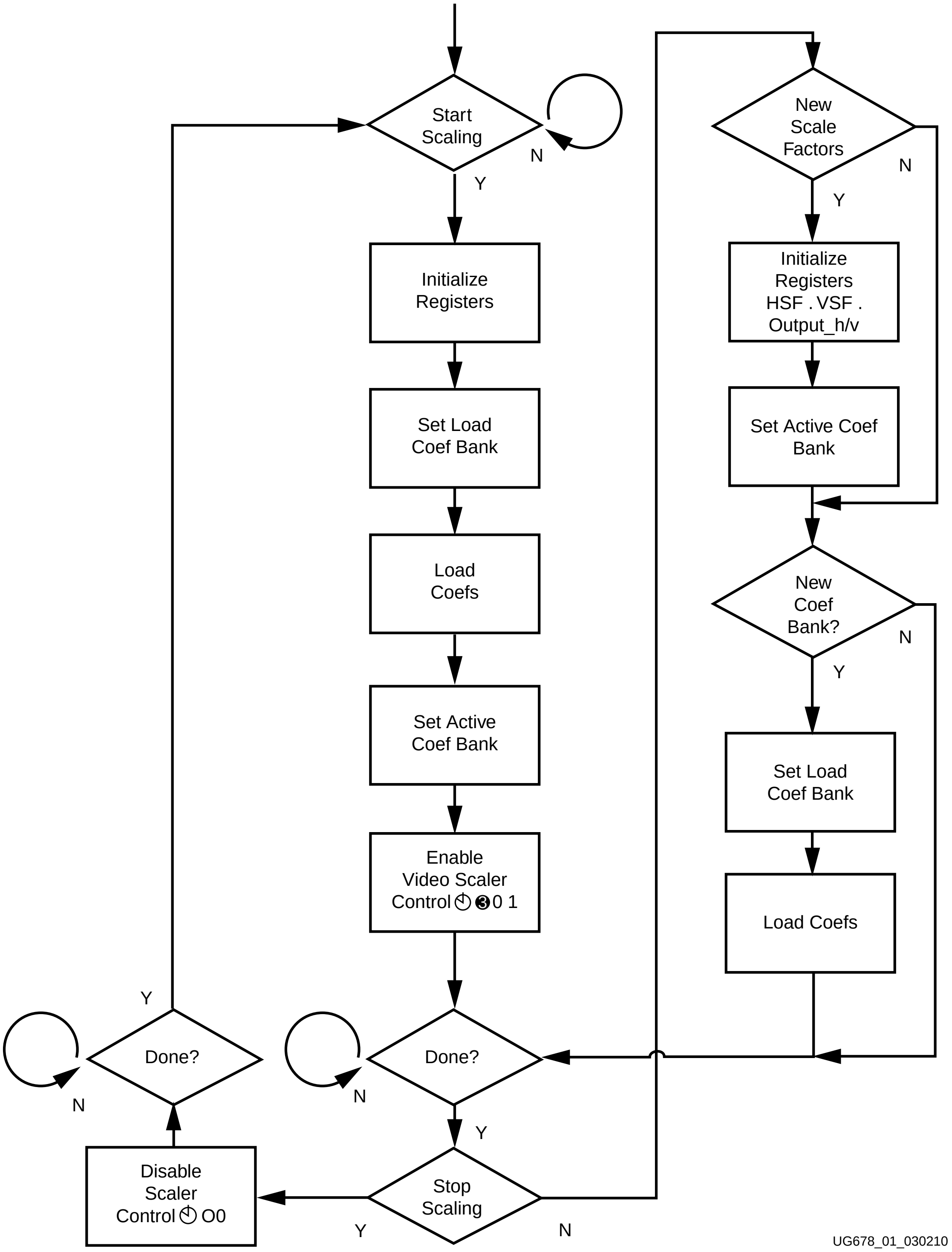


Figure D-0: **Video Scaler Flow Chart**

System Timing Diagram

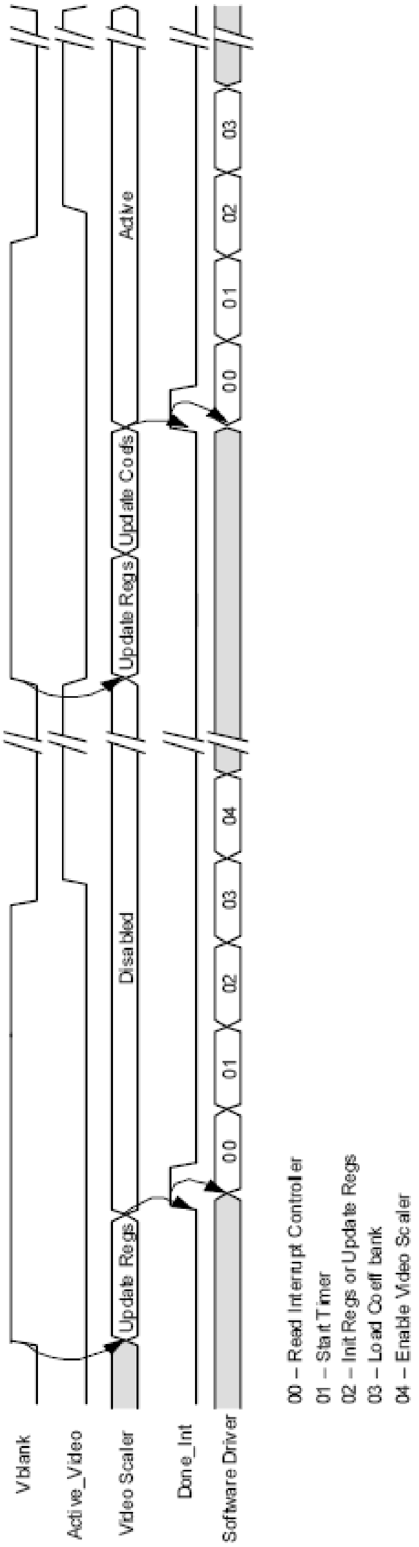


Figure D-0: **System Timing Diagram**

Proposed API function calls

The following functions are proposed for LO, 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

L2 API Function Calls

#define XScaler_Zoom(InstancePtr, zoom_factor_h, zoom_factor_v, starting_aperture_h, ending_aperture_h, starting_aperture_v, ending_aperture_v, output_h, output_v, num_of_frames)

- In a zoom operation the input image size is changing on a frame basis and the output resolution is fixed.
- Calls `XScaler_CalcScaleFactors`, `XScaler_SetScalerValues` every frame to perform the zoom function. Prior to beginning the zoom operation, you will have to preload the coeff banks you would like to use for the duration and decide when to transition to a new coefficient bank; example 4 coeff banks for 200 frames switch bank every 50 frames.

#define XScaler_DownSize(InstancePtr, downsize_factor_h, downsize_factor_v, num_of_frames)

- In a downsize operation, the input image size is not changing on a frame basis and the output resolution is changing.
- Calls `XScaler_CalcScaleFactors`, `XScaler_SetScalerValues` every frame to perform the downsize function. Prior to beginning the downsize operation, you will have to preload the coeff banks you would like to use for the duration and decide when to transition to a new coefficient bank; example 4 coeff banks for 200 frames switch bank every 50 frames.

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
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

Appendix E

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_v5_0 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 v5.0 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_v5_0_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_v5_0_bitacc_cmodel.h	Model header file

Table E-1: **Directory structure and files of the Video Scaler v3.0Bit Accurate Model**

File Name	Contents
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_v5_0_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_v5_0_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_v5_0_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_v5_0_bitacc_cmodel.so
run_bitacc_cmodel	32-bit Linux fixed configuration executable

Table E-1: **Directory structure and files of the Video Scaler v3.0Bit Accurate Model**

File Name	Contents
./nt64	Directory containing Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_scaler_v5_0_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_v5_0_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 2005 (Visual C++ 8.0)

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_v5_0_bitacc_simulate(
struct xilinx_ip_v_scaler_v5_0_generics generics,
struct xilinx_ip_v_scaler_v5_0_inputs inputs,
struct xilinx_ip_v_scaler_v5_0_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_v5_0_generics scaler_generics;
struct xilinx_ip_v_scaler_v5_0_inputs scaler_inputs;
struct xilinx_ip_v_scaler_v5_0_outputs* scaler_outputs;
```

The declarations of these structures are in the v_scaler_v5_0_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 121.
   - 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 121.

   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 124 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 122. 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)

For 64-bit Linux, cd into the /lin64 directory. From there, run the command:

```
gcc -m64 -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_scaler_v5_0_bitacc_cmodel -Wl,-rpath,.
```

When using 32-bit linux, cd into the /lin directory, and run with the '-m32' switch:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_scaler_v5_0_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.

References

These documents provide supplemental material useful with this product guide:

- 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 terms of the Xilinx Core License Agreement and can be generated using the Xilinx CORE Generator system. The CORE Generator system is shipped with Xilinx ISE Design Suite 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.

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.

Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had 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.