LogiCORE IP Chroma Resampler v1.0

Product Guide

PG012 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: Constraining the Core

Chapter 5: Designing with the Core

Chapter 6: Detailed Example Design

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Debugging

Appendix C: Application Software Development

Appendix D: C Model Reference

Appendix E: Additional Resources

LogiCORE IP Chroma Resampler v1.0

Introduction

The Xilinx Chroma Resampler LogiCORE provides users with an easy-to-use IP block for converting between chroma sub-sampling formats.

Features

- Support for:
  - High-definition (1080p60) resolutions
  - Up to 4095 total scanlines and 4095 pixels per scanline
- Converts between YCbCr:
  - 4:4:4
  - 4:2:2
  - 4:2:0
- Supports both progressive and interlaced video
- Static, predefined, powers-of-two coefficients for low-footprint applications
- Configurable filters sizes with programmable filter coefficients for high performance applications
- Selectable processor interface
  - EDK pCore
  - General Purpose Processor
  - Constant Interface
- Support for 8-, 10-, or 12-bit input and output precision
- Xilinx Streaming Video Interface (XSVI) bus simplifies connecting to other video IP

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex®-7, Kintex™ -7, Virtex-6, and Spartan®-6
Supported User Interfaces	General Purpose Processor Interface, EDK pCore AXI4-Lite, Constant Interface
Resources	See Table 1-1 through Table 1-4.
Provided with Core	
Design Files	Netlist or EDK pCore, C Driver
Example Design	Not Provided
Test Bench	VHDL[2]
Constraints File	Not Provided
Simulation Model	Verilog and VHDL Structural Models[2]
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool v13.3 Platform Studio (XPS) v13.3
Simulation[3]	Mentor Graphics ModelSim, Xilinx ISim 13.3
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. HDL test bench and C Model available on the Chroma Resampler product page.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Overview

It is accepted that the human eye is not as receptive to chrominance (color) detail as luminance (brightness) detail. Using color-space conversion, it is possible to convert RGB into the YCbCr color space, where Y is Luminance information, and Cb and Cr are derived color difference signals. At normal viewing distances, there is no perceptible loss incurred by sampling the color difference signals (Cb and Cr) at a lower rate to provide a simple and effective video compression to reduce storage and transmission costs

The Chroma Resampler core converts between chroma sub-sampling formats of 4:4:4, 4:2:2, and 4:2:0. There are a total of six conversions available for the three supported sub-sampling formats. Conversion is achieved using a FIR filter approach. Some conversions require filtering in only the horizontal dimension, vertical dimension, or both. Interpolation operations are implemented using a two-phase polyphase FIR filter. Decimation operations are implemented using a low-pass FIR filter to suppress chroma aliasing.

Standards Compliance

The Chroma Resampler core uses AXI4-Lite interfaces and is compliant with 4:2:0 (MPEG-2, MPEG-4 Part 2, and H.264).

Feature Summary

The Chroma Resampler core converts between different Chroma sub-sampling formats. The supported formats are 4:4:4, 4:2:2, and 4:2:0. There are three different options for interpolating and decimating the video samples:

- Define a configurable filter with programmable coefficients for high-performance applications
- Use the pre-defined static filter with power-of-two coefficients for low-footprint applications.
- Replicate or drop pixels.

Licensing

The Chroma Resampler core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

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

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Chroma Resampler core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.) No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Chroma Resampler core using the example design and demonstration test bench provided with the core. In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (ceasing to function), at which time it can be reactivated by reconfiguring the device.

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

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

Full

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

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

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

This section details the performance information for various core configurations.

Maximum Frequencies

The maximum achievable clock frequency could vary and in most cases will be higher. The maximum achievable clock frequency and all resource counts may be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors. Maximum frequencies for the supported devices are:

- Virtex-7 Family: 250MHz
- Kintex-7 Family: 250MHz
- Virtex-6 Family: 250MHz
- Spartan-6 Family: 150 MHz

Latency

This section includes equations to calculate the latency of the core. NUM_H_TAPS is the number of horizontal filter taps. NUM_V_TAPS is the number of vertical filter taps. A delay of one line is equal to the number of video clock cycles between subsequent rising edges of the `hblank` signal.

4:2:2 to 4:4:4

The latency through the default filter is eight clock cycles. For non-default filters, the latency can be calculated according to the formula:

Latency = (2*NUM_H_TAPS) + 4 clock cycles

When using the replicate option, the latency is seven clock cycles.

4:4:4 to 4:2:2

The latency through the default filter is ten clock cycles. For non-default filters, the latency can be calculated according to the formula:

Latency = (NUM_H_TAPS + 7) clock cycles

When using the drop option, the latency is two clock cycles.

4:2:0 to 4:2:2

The latency through the default filter is 1 line + 10 clock cycles. For non-default filters, the latency can be calculated according to the formulas:

Vertical_Latency = (NUM_V_TAPS - 1) lines

Horizontal_Latency = (NUM_V_TAPS +8) clock cycles

When using the replicate option, the latency is 5 clock cycles.

4:2:2 to 4:2:0

The latency through the default filter is 1 line + 7 clock cycles. For non-default filters, the latency can be calculated according to the formulas:

Vertical_Latency = (NUM_V_TAPS/2) lines

Horizontal_Latency = (NUM_V_TAPS + 5) clock cycles

When using the drop option, the latency is 3 clock cycles.

4:2:0 to 4:4:4

The latency through the default filter is 1 line + 18 clock cycles. For non-default filters, the latency can be calculated according to the formulas:

Vertical_Latency = (NUM_V_TAPS - 1) lines

Horizontal_Latency = (NUM_V_TAPS + (2*NUM_H_TAPS) + 12) clock cycles

When using the replicate option, the latency is 12 clock cycles.

4:4:4 to 4:2:0

The latency through this default filter is 1 line + 17 clock cycles. For non-default filters, the latency can be calculated according to the formulas:

Vertical_Latency = (NUM_V_TAPS/2) lines

Horizontal_Latency = (NUM_H_TAPS + NUM_V_TAPS + 12) clock cycles

When using the drop option, the latency is equal to 5 clock cycles.

Throughput

For 4:4:4 data, this core processes inputs/outputs one YCbCr sample per clock cycle.

For 4:2:2 data, there is one Y sample per clock cycle. The Cb and Cr data are interleaved, and it takes a total of two clock cycles to input/output the chroma data for one sample.

For 4:2:0 data, there is one Y sample per clock cycle. For the lines with valid chroma data, the Cb and Cr data are interleaved, and it takes a total of two clock cycles to input/output the chroma data for one sample.

Resource Utilization

The information presented in Table 1-1 through Table 1-4 is a guideline to the resource utilization of the Chroma Resampler core for Kintex-7, Virtex-7, Virtex-6, and Spartan-6 FPGA families. The core does not use any dedicated I/O or clock resources. The design was tested using Xilinx ISE® software version 13.3 with default tool options.

For an accurate measure of device resource usage (for example, block RAMs, flip-flops, and LUTs) for a particular core instance, click **View Resource Utilization** in the CORE Generator interface after generating the core.

For each configuration, the resource usage and performance numbers were generated with the following parameters:

- pCore Interface
- 1920 x 1080 frame size
- Default filter size
- Progressive video
- 8-bit data

Table 1-1 details the resource usage for Virtex-7 devices. Table 1-2 details the resource usage for Kintex-7 devices. Table 1-3 details the resource usage for Virtex-6 devices. Table 1-4 details the resource usage for Spartan-6 devices.

Table 1-1: **Virtex-7 Resource Usage**[(1)]

Conversion	Filter Type	LUT6-FF pairs	LUTs	FFs	RAM16/8	DSP48A1	Clock Frequency (MHz)
4:2:2 to 4:4:4	User Defined	331	185	295	0/0	3	300
	Fixed Coefficient	322	211	321	0/0	0	458
	Drop/Replicate	179	66	180	0/0	0	388
4:2:2 to 4:4:4	User Defined	314	193	304	0/0	2	300
	Fixed Coefficient	320	204	311	0/0	0	370
	Drop/Replicate	182	89	194	0/0	0	450
4:2:2 to 4:2:0	User Defined	745	594	487	0/2	2	274
	Fixed Coefficient	746	595	500	0/2	0	291
	Drop/Replicate	221	133	198	0/0	0	422
4:2:0 to 4:2:2	User Defined	862	719	601	0/3	2	264
	Fixed Coefficient	890	726	629	0/3	0	300
	Drop/Replicate	358	243	293	0/1	0	300
4:4:4 to 4:2:0	User Defined	971	746	667	0/2	5	274
	Fixed Coefficient	957	764	706	0/2	0	317
	Drop/Replicate	274	156	260	0/0	0	414
4:2:0 to 4:4:4	User Defined	1,087	833	789	0/3	4	256
	Fixed Coefficient	1,084	876	829	0/3	0	309
	Drop/Replicate	425	283	372	0/1	0	309

1. Device, Package, Speed: XC7VX330T, FFG1157,C,-1 (ADVANCED 1.01 2011-09-26)

Table 1-2: **Kintex-7 Resource Usage**[1]

Conversion	Filter Type	LUT6-FF pairs	LUTs	FFs	RAM16/8	DSP48A1	Clock Frequency (MHz)
4:2:2 to 4:4:4	User Defined	313	199	295	0/0	3	303
	Fixed Coefficient	320	222	321	0/0	0	390
	Drop/Replicate	128	97	180	0/0	0	438
4:2:2 to 4:4:4	User Defined	320	183	304	0/0	2	303
	Fixed Coefficient	294	219	311	0/0	0	425
	Drop/Replicate	139	125	194	0/0	0	444
4:2:2 to 4:2:0	User Defined	740	593	487	0/2	2	262
	Fixed Coefficient	730	609	500	0/2	0	303
	Drop/Replicate	182	155	198	0/0	0	404
4:2:0 to 4:2:2	User Defined	878	695	601	0/3	2	275
	Fixed Coefficient	874	727	629	0/3	0	303
	Drop/Replicate	347	247	293	0/1	0	317
4:4:4 to 4:2:0	User Defined	937	753	667	0/2	5	275
	Fixed Coefficient	896	779	706	0/2	0	303
	Drop/Replicate	285	147	260	0/0	0	398
4:2:0 to 4:4:4	User Defined	1,039	831	789	0/3	4	268
	Fixed Coefficient	1,106	860	829	0/3	0	289
	Drop/Replicate	427	279	372	0/1	0	317

[1] Device, Package, Speed: XC7K70T, FBG484, C, -1 (ADVANCED 1.02 2011-09-26)

Table 1-3: **Virtex-6 Resource Usage[1]**

Conversion	Filter Type	LUT6-FF pairs	LUTs	FFs	RAM16/8	DSP48A1	Clock Frequency (MHz)
4:2:2 to 4:4:4	User Defined	345	187	295	0/0	3	262
	Fixed Coefficient	326	240	329	0/0	0	486
	Drop/Replicate	167	94	180	0/0	0	417
4:2:2 to 4:4:4	User Defined	317	188	304	0/0	2	262
	Fixed Coefficient	307	202	311	0/0	0	424
	Drop/Replicate	175	108	202	0/0	0	456
4:2:2 to 4:2:0	User Defined	761	614	485	0/2	2	262
	Fixed Coefficient	780	650	499	0/2	0	300
	Drop/Replicate	223	130	198	0/0	0	417
4:2:0 to 4:2:2	User Defined	876	726	599	0/3	2	262
	Fixed Coefficient	930	799	628	0/3	0	305
	Drop/Replicate	373	269	293	0/1	0	342
4:4:4 to 4:2:0	User Defined	960	762	665	0/2	5	262
	Fixed Coefficient	986	807	705	0/2	0	305
	Drop/Replicate	271	158	260	0/0	0	405
4:2:0 to 4:4:4	User Defined	1,034	854	787	0/3	4	262
	Fixed Coefficient	1,103	957	828	0/3	0	300
	Drop/Replicate	446	301	372	0/1	0	349

[1] Device, Package, Speed: XC6VLX75T, FF484, C, -1 (PRODUCTION 1.15 2011-09-26)

Table 1-4: **Spartan-6 Resource Usage**

Conversion	Filter Type	LUT6-FF pairs	LUTs	FFs	RAM16/8	DSP48A1	Clock Frequency (MHz)
4:2:2 to 4:4:4	User Defined	286	200	295	0/0	3	263
	Fixed Coefficient	294	219	321	0/0	0	293
	Drop/Replicate	183	120	182	0/0	0	316
4:2:2 to 4:4:4	User Defined	320	218	305	0/0	2	226
	Fixed Coefficient	292	224	312	0/0	0	271
	Drop/Replicate	167	95	194	0/0	0	316
4:2:2 to 4:2:0	User Defined	731	614	483	2/0	2	219
	Fixed Coefficient	731	614	497	2/0	0	219
	Drop/Replicate	198	129	198	0/0	0	271
4:2:0 to 4:2:2	User Defined	822	733	597	3/0	2	210
	Fixed Coefficient	883	766	625	3/0	0	219
	Drop/Replicate	347	254	293	1/0	0	249
4:4:4 to 4:2:0	User Defined	963	847	683	2/0	5	210
	Fixed Coefficient	912	760	703	2/0	0	219
	Drop/Replicate	236	169	260	0/0	0	271
4:2:0 to 4:4:4	User Defined	1,123	1,015	817	3/0	4	196
	Fixed Coefficient	1,213	1,046	858	3/0	0	196
	Drop/Replicate	389	298	372	1/0	0	256

1. Device, Package, Speed: XC6SLX4, CSG225, C, -2 (PRODUCTION 1.20 2011-09-26)

Chapter 2

Core Interfaces and Register Space

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

The Chroma Resampler core supports the following three processor interface options:

- Constant interface
- EDK pCore interface
- General Purpose Processor (GPP) interface

The EDK pCore and GPP interfaces provide the system designer with the ability to dynamically control filter coefficients, chroma parity, and frame size parameters within the core.

Port Descriptions

Table 2-1 contains general port information. A more detailed description of each port follows the table. This set of signals is common to all interface options.

Table 2-1: **Port Descriptions for the Signals Common to All Interfaces**

Port Name	Port Width[1]	Direction	Description
video_data_in	3*DATA_WIDTH or 2*DATA_WIDTH	IN	Data input bus
hblank_in	1	IN	Horizontal blanking input
vblank_in	1	IN	Vertical blanking input
active_video_in	1	IN	Active video signal input
field_id_in	1	IN	Field ID for interlaced video
clk	1	IN	Rising-edge clock
ce	1	IN	Clock enable (active high)
sclr	1	IN	Synchronous clear , reset (active High)
video_data_out	3*DATA_WIDTH or 2*DATA_WIDTH	OUT	Data output bus
hblank_out	1	OUT	Horizontal blanking output
vblank_out	1	OUT	Vertical blanking output
active_video_out	1	OUT	Active video signal output

Table 2-1: **Port Descriptions for the Signals Common to All Interfaces *(Cont'd)***

Port Name	Port Width[1]	Direction	Description
field_id_out	1	OUT	Field ID for interlaced video

1. If chroma format is 4:4:4, port width is 3*DATA_WIDTH. Otherwise, port width is 2*DATA_WIDTH. DATA_WIDTH represents the data width, (8, 10 or 12 bits) selected in the CORE Generator GUI.

- `video_data_in`: This bus contains unsigned luma and chroma input data in the format shown in Table 2-2.
  - For 4:2:2 and 4:2:0 Cb and Cr are interleaved on bit-fields (2*DATA_WIDTH -1 : DATA_WIDTH). On each line, Chroma sample Cb is expected first.
  - For 4:2:0, the chroma_parity signal/register specifies whether the first line of video contains chroma information or not.
  - See Figure 2-6, Figure 2-7, and Figure 2-8 for timing diagrams.

Table 2-2: **video_data_in Format**

	3*DATA_WIDTH-1 : 2*DATA_WIDTH	***2*DATA_WIDTH -1 : DATA_WIDTH***	***DATA_WIDTH -1 : 0***
YCbCr 4:4:4	Cb	Cr	Y
YCbCr 4:2:2		Cb/Cr	Y
YCbCr 4:2:0		Cb/Cr	Y

- `hblank_in`: The `hblank_in` signal conveys information about the blank/non-blank regions of video scan lines. When input video is horizontally blanked, `hblank_in` is expected High, and active_video_in is expected to be Low.
- `vblank_in`: The `vblank_in` signal conveys information about the blank/non-blank regions of video frames, and is used by the core to detect the end of a frame, when user registers can be copied to active registers to avoid visual tearing of the image. When input video is vertically blanked, `vblank_in` is expected to be High.
- `active_video_in`: The `active_video_in` signal is High when valid data is presented at the input.
- `field_id_in`: The `field_id_in` signal is only present for interlaced data. The signal should be set to 1 for the odd field and 0 for the even field.
- `clk`: Master clock in the design. It is synchronous to, or identical with video clock.
- `ce`: Pulling CE Low suspends all operations within the core. Outputs are held, and no input signals are sampled except for reset (SCLR takes precedence over CE).
- `sclr`: Pulling SCLR High resets all output pins to zero. Internal registers within the DSP48 slices and D-flip-flops are cleared. However, SRL16/SRL32-based delay lines are not cleared by SCLR. This may result in non-zero outputs after SCLR is deasserted, until the contents of SRL16/SRL32s are flushed. SRLs used are cleared only if SCLR is held high for the latency of the core.
- `video_data_out`: This bus contains the output data in the format shown in Table 2-3.
  - For 4:2:2 and 4:2:0, Cb and Cr are interleaved on bit-fields (2*DATA_WIDTH -1 : DATA_WIDTH). On each line, Chroma sample Cb is expected first.
  - For 4:2:0, the chroma_parity signal/register specifies whether the first line of video contains chroma information or not.
  - See Figure 2-6, Figure 2-7, and Figure 2-8 for timing diagrams.

Table 2-3: **video_data_out Format**

	3*DATA_WIDTH-1 : 2*DATA_WIDTH	**2*DATA_WIDTH -1 : DATA_WIDTH**	**DATA_WIDTH -1 : 0**
YCbCr 4:4:4	Cb	Cr	Y
YCbCr 4:2:2		Cb/Cr	Y
YCbCr 4:2:0		Cb/Cr	Y

- `hblank_out` and `vblank_out`: Corresponding input signals are delayed so blanking outputs are in phase with the video data output. This maintains the integrity of the video stream. `hblank_out` and `vblank_out` are High when the video output is horizontally blanked. Unwanted blanking inputs should be tied High, and corresponding outputs left unconnected, which will result in the trimming of any unused logic within the core.
- `field_id_out`: The `field_id_out` signal is only present for interlaced data. The signal is set to 1 for the odd field, and is set to 0 for the even field.

Constant Interface

The Constant Interface does not provide an option for the filter coefficients to be changed in system. There is no processor interface, and the core is not programmable, but it can be reset and enabled/disabled using the SCLR and CE ports. The Constant Interface consists of the common signals described in Table 2-1 and the additional signals shown in Table 2-4. The core symbol is shown in Figure 2-1.

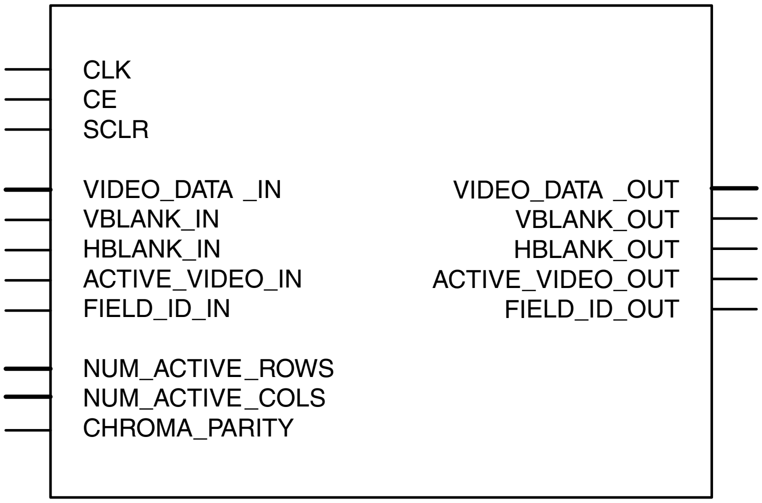


Figure 2-1: **Constant Interface Core Symbol**

Table 2-4: **Additional Port Descriptions for the Constant Interface**

Pin Name	Dir	Width	Description
num_active_cols	[1]	IN	Number of active columns
num_active_rows	[2]	IN	Number of active rows
chroma_parity	1	IN	For 4:2:0, specifies whether the first line of video contains chroma information (1) or not (0)

1. The port width for num_active_cols is the number of bits needed to represent the Maximum Number of Columns set in the GUI (log2(C_MAX_COLS+1).
2. The port width for num_active_rows is the number of bits needed to represent the Maximum Number of Rows set in the GUI (log2(C_MAX_ROWS+1).

EDK pCore Interface

There are multiple imaging applications that include an embedded processor that dynamically controls the parameters within an integrated system. The Chroma Resampler core can generate a pCore interface, which allows adding the core to an EDK project as a hardware peripheral.

The Chroma Resampler core, when configured as an EDK pCore, uses the AXI4-Lite Interface (Table 2-5) to interface to an embedded microprocessor or an AXI4-Lite bus master.

Refer to the AMBA AXI4 Interface Protocol web site (http://www.xilinx.com/ipcenter/axi4.htm) for more information on the AXI4 and AXI4-Lite interface signals.

Table 2-5: **AXI4-Lite Interface Pinout**

Pin Name	Dir	Width	Description
AXI Global System Signals[1]			
S_AXI_ARESETN	I	1	AXI Reset, active low
IP2INTC_Irpt	O	1	Interrupt request output
AXI Write Address Channel Signals[1]			
S_AXI_AWADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.
S_AXI_AWVALID	I	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	O	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.
AXI Write Data Channel Signals[1]			
S_AXI_WDATA	I	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Write Data Bus.
S_AXI_WSTRB	I	[C_S_AXI_DATA_WIDTH/8-1:0]	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.

Table 2-5: **AXI4-Lite Interface Pinout *(Cont'd)***

Pin Name	Dir	Width	Description
S_AXI_WVALID	I	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	O	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.
AXI Write Response Channel Signals[1]			
S_AXI_BRESP[2]	O	[1:0]	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	O	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. • 1 = Response is valid. • 0 = Response is not valid.
S_AXI_BREADY	I	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.
AXI Read Address Channel Signals[1]			
S_AXI_ARADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction
S_AXI_ARVALID	I	1	AXI4-Lite Read Address Channel Read Address Valid. • 1 = Read address is valid. • 0 = Read address is not valid.
S_AXI_ARREADY	O	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.
AXI Read Data Channel Signals[1]			
S_AXI_RDATA	O	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Read Data Bus.

Table 2-5: **AXI4-Lite Interface Pinout *(Cont'd)***

Pin Name	Dir	Width	Description
S_AXI_RRESP[2]	O	[1:0]	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	O	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	I	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.

1. The function and timing of these signals are defined in the AMBA AXI Protocol Version: 2.0 Specification .
2. For signals S_AXI_RRESP[1:0] and S_AXI_BRESP[1:0], the core does not generate the Decode Error ('11') response. Other responses like '00' (OKAY) and '10' (SLVERR) are generated by the core based upon certain conditions.

Figure 2-2 shows the core symbol for the EDK pCore interface.

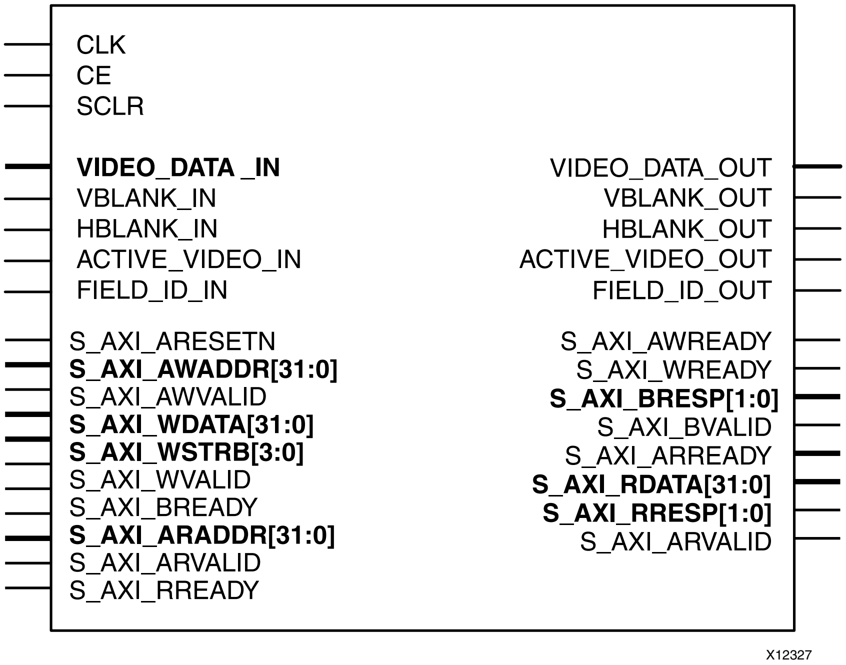


Figure 2-2: **EDK pCore Symbol**

Register Space

The memory-mapped interface allows dynamic updates of the programmable registers within the core, as described in Table 2-6.

Table 2-6: **EDK pCore Register Descriptions**

Address Offset	Register Name	Access Type	Default Value	Description	
0x00	cresample_reg00_control	R/W	0x1	Bit 0	Software Enable. • 0 = Not enabled • 1 = Enabled
				Bit 1	Host Processor Write Done Semaphore. • 0 = Host processor actively updating registers • 1 = Register update completed by host processor
0x04	cresample_reg01_reset	R/W	0x0	Bit 0	Software Reset. • 0 = Not reset • 1 = Reset

Table 2-6: **EDK pCore Register Descriptions *(Cont'd)***

Address Offset	Register Name	Access Type	Default Value	Description	
0x08	cresample_reg02_status	R	0x02030000	0-6	Reserved
				7	Timing Locked. 1 = Indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals
				16-31	Reserved
0x0C	cresample_reg03_num_active_cols	R/W	(1)	Number of active columns	
0x10	cresample_reg04_num_active_rows	R/W	(1)	Number of active rows	
0x14	cresample_reg05_chroma_parity	R/W	0x1	For 4:2:0, specifies whether the first line of video contains chroma information (1) or not (0).	
0x18	cresample_reg06_coef00_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x1C	cresample_reg07_coef01_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x20	cresample_reg08_coef02_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x24	cresample_reg09_coef03_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x28	cresample_reg10_coef04_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x2C	cresample_reg11_coef05_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x30	cresample_reg12_coef06_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x34	cresample_reg13_coef07_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x38	cresample_reg14_coef08_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x3C	cresample_reg15_coef09_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x40	cresample_reg16_coef10_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x44	cresample_reg17_coef11_hphase0	R/W	(2)	Coefficient for Horizontal Filter Phase 0	
0x48	cresample_reg18_coef00_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x4C	cresample_reg19_coef01_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x50	cresample_reg20_coef02_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x54	cresample_reg21_coef03_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x58	cresample_reg22_coef04_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x5C	cresample_reg23_coef05_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x60	cresample_reg24_coef06_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x64	cresample_reg25_coef07_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x68	cresample_reg26_coef08_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	
0x6C	cresample_reg27_coef09_hphase1	R/W	(2)	Coefficient for Horizontal Filter Phase 1	

Table 2-6: **EDK pCore Register Descriptions *(Cont'd)***

Address Offset	Register Name	Access Type	Default Value	Description
0x70	cresample_reg28_coef10_hphase1	R/W	[2]	Coefficient for Horizontal Filter Phase 1
0x74	cresample_reg29_coef11_hphase1	R/W	[2]	Coefficient for Horizontal Filter Phase 1
0x78	cresample_reg30_coef00_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x7C	cresample_reg31_coef01_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x80	cresample_reg32_coef02_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x84	cresample_reg33_coef03_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x88	cresample_reg34_coef04_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x8C	cresample_reg35_coef05_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x90	cresample_reg36_coef06_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x94	cresample_reg37_coef07_vphase0	R/W	[2]	Coefficient for Vertical Filter Phase 0
0x98	cresample_reg38_coef00_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0x9C	cresample_reg39_coef01_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xA0	cresample_reg40_coef02_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xA4	cresample_reg41_coef03_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xA8	cresample_reg42_coef04_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xAC	cresample_reg43_coef05_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xB0	cresample_reg44_coef06_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1
0xB4	cresample_reg45_coef07_vphase1	R/W	[2]	Coefficient for Vertical Filter Phase 1

1. cresample_reg03_num_active_cols and cresample_reg04_num_active_rows default to the maximum number of columns and maximum number of rows specified in the GUI.
2. The coefficients default to the pre-defined Fixed Coefficient filter values.

All of the registers are readable. This enables the system processor to verify writes and read current values contained within the registers.

This core supports a software enable/disable function. When disabled, the normal operation of the hardware is halted by blocking the propagation of all video signals. This function is controlled by bit 0 of the `cresample_reg00_control` register. The default value of Software Enable is 1 (enabled). Software Enable is sampled at the rising edge of `vblank_in`. While the general CE pin can be useful for data-throttling the core, the Software Enable register can enable/disable the core for an entire frame. This ensures no image tearing during the active portion of a video frame.

The in-system reset of the core is controlled by asserting `cresample_reg01_reset` (bit 0), which returns the coefficients, chroma parity, and frame size to their default values. The core control signals and output are forced to 0 until the software reset bit is deasserted.

The frame size, chroma_parity, and coefficient registers are double buffered in the hardware to ensure no image tearing if the coefficient values are modified in the active area of a frame. This double buffering provides system control that is more flexible and easier to use by decoupling the register updates from the blanking period. This allows the software a much larger window with which to update the parameter values. The updated values for the coefficient, chroma parity, and frame size registers are latched into the shadow registers immediately after writing, and the actual coefficients, chroma parity, and frame size used are stored in the working registers. The rising edge of `vblank_in` triggers the values from the shadow registers to be copied to the working registers when bit 1 of `cresample_reg00_control` is set to 1. This semaphore bit helps to prevent changing the coefficient, chroma parity, and frame size values mid-frame.

General Purpose Processor Interface

The General Purpose Processor Interface exposes the filter coefficients and frame size registers as ports (Table 2-6). The General Purpose Processor Interface is provided as an option to design a system with a user-defined bus interface (decoding logic and register banks) to an arbitrary processor.

The frame size, chroma parity, and coefficient registers are double buffered in the hardware to ensure no image tearing if the register values are modified in the active area of a frame. This double buffering provides system control that is more flexible and easier to use by decoupling the register updates from the blanking period. This allows the software a much larger window in which to update the parameter values. The updated values for the coefficient, chroma parity, and frame size registers are latched into the shadow registers immediately after writing, and the actual coefficients used are stored in the working register.

However, external registers (shadow registers) have to be supplied by the user-defined bus interface. Values from this register bank (external to the Chroma Resampler core) are copied over to the internal registers at the rising edge of `vblank_in` when bit 1 of the control port is set to 1.

The General Purpose Processor Interface core symbol is shown in Figure 2-3, and the ports are described in Table 2-7, page 24.

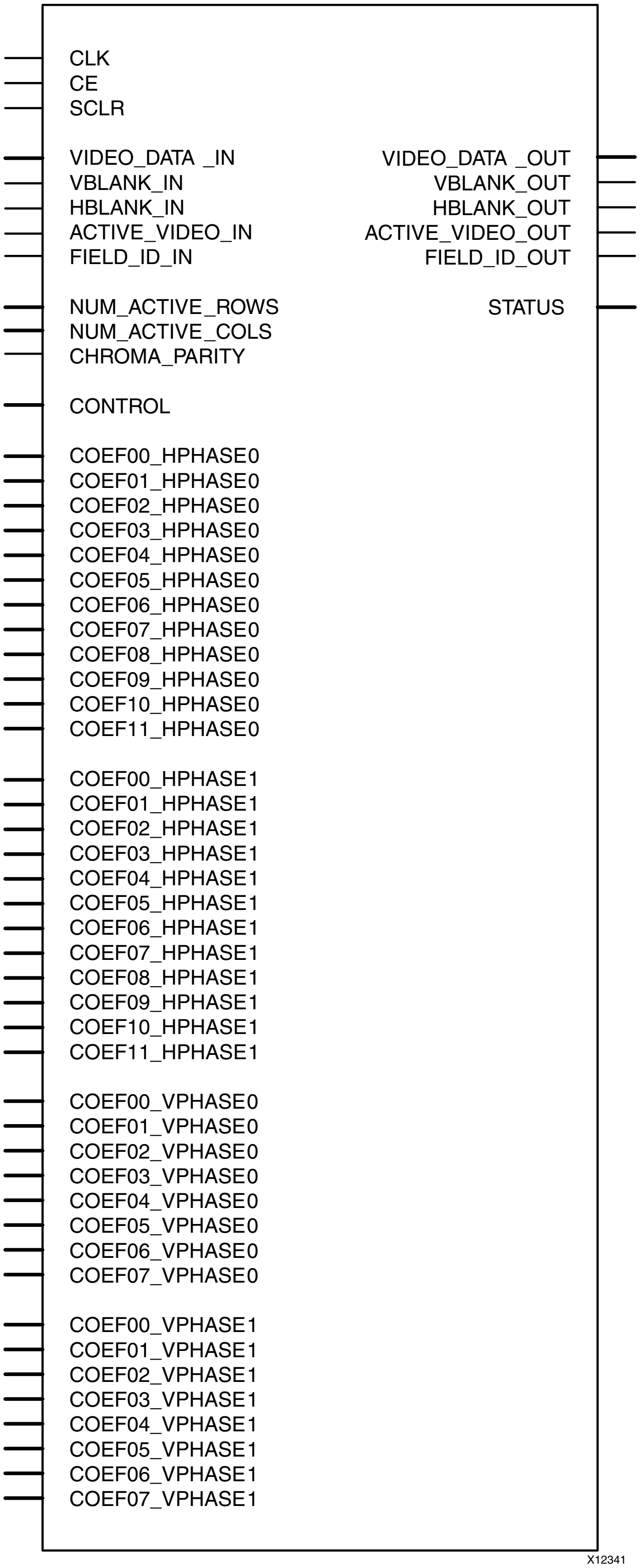


Figure 2-3: **Core Symbol for the General Purpose Processor Interface**

Table 2-7: **Additional Port Descriptions for the General Processor Interface**

Port Name	Port Width	Direction	Description	
control	2	IN	Bit 0	Software Enable. • 0 = Not enabled • 1 = Enabled
			Bit 1	Host Processor Write Done Semaphore. • 0 = Host processor actively updating registers • 1 = Register update completed by host processor
status	8	OUT	Bits 0-6	Reserved
			Bit 7	Timing Locked. 1 - Indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals
coef00_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef01_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef02_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef03_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef04_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef05_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef06_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef07_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef08_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef09_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef10_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef11_hphase0	16	IN	Coefficient for Horizontal Filter Phase 0	
coef00_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef01_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef02_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef03_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef04_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef05_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef06_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef07_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	
coef08_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1	

Table 2-7: **Additional Port Descriptions for the General Processor Interface *(Cont'd)***

Port Name	Port Width	Direction	Description
coef09_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1
coef10_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1
coef11_hphase1	16	IN	Coefficient for Horizontal Filter Phase 1
coef00_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef01_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef02_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef03_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef04_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef05_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef06_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef07_vphase0	16	IN	Coefficient for Vertical Filter Phase 0
coef00_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef01_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef02_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef03_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef04_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef05_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef06_vphase1	16	IN	Coefficient for Vertical Filter Phase 1
coef07_vphase1	16	IN	Coefficient for Vertical Filter Phase 1

1. The port width for num_active_cols is the number of bits needed to represent the Maximum Number of Columns set in the GUI (log2(C_MAX_COLS+1).
2. The port width for num_active_rows is the number of bits needed to represent the Maximum Number of Rows set in the GUI (log2(C_MAX_ROWS+1).

- `control`: This port gives access to the Software Enable and Register Update Enable bits.

  Software Enable (bit 0) can enable (1) or disable (0) core functionality, and is sampled at the rising edge of `vblank_in`. While the general CE pin can be useful for data-throttling the core, the Software Enable port can enable/disable the core for an entire frame. This ensures functionality is not frozen during the active portion of a video frame.

  Register updates can be completed by asserting bit 1 of the control port (write done semaphore). At the next rising edge of `vblank_in`, the core copies the user register contents to the active registers and starts using the updated values.
- `status`: Timing Lock (bit 7) indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals.
- `num_active_cols`: This port sets the number of active columns in a frame. The allowed values are 32 to 4095. The number of active columns must be less than or equal to the Maximum Number of Columns set in the GUI.

- num_active_rows: This port sets the number of active rows in a frame. The allowed values are 32 to 4095. The number of active rows must be less than or equal to the Maximum Number of Rows set in the GUI. For interlaced video, the number of rows must be the same for each field.
- chroma_parity: For 4:2:0, specifies whether the first line of video contains chroma information (1) or not (0).
- Coefficients: The coefficients for the interpolation and anti-aliasing filters can be programmed via these ports. One port is provided for each filter coefficient. Only the coefficient ports needed for the conversion and filter size selected are available.

Timing Diagrams

The propagation delay of the Chroma Resampler core is described in Latency in Chapter 1. The output timing signals (vblank_out, hblank_out, active_video_out, and field_id_out) are delayed appropriately so that the output video data is framed correctly by the timing signals.

The control signals vblank_out, hblank_out, active_video_out and field_id_out are driven by timing detector and generator modules within the core. The internal timing module assumes the following:

- active_video, hblank, and vblank signals are active High
- One vertical blanking period per frame
- A minimum vertical blanking period of three scanlines
- hblank_in is periodic throughout the frame
- One horizontal blanking period per scanlines
- A minimum horizontal blanking period equal to the horizontal latency of the core
- active_video_in is contiguous for each line
- The High portion of active_video_in should not extend across edges of either blanking signals.
- A minimum active frame size of 32 scanlines and 32 pixels per scanline
- A maximum total frame size of 4095 scanlines and 4095 pixels per scanline

Once the internal timing module has locked on the input timing signals and is generating stable output timing signals, the flag Timing_Lock (bit 7 of the output status port/register) is set to 1. While locking to the input timing signals, when Timing_Lock=0, the core cannot guarantee the correct video data output. Therefore, the data output, video_data_out, is set to zero until Timing_Lock is set, even though active_video_out may be High.

The timing module needs one full line (from one rising edge of hblank_in to the next rising edge) to detect the input timing signals. If timing is detected before the active portion of the frame (meaning there are two hblank_in pulses before active_video_in goes High), then the core will produce valid data for the first frame, as shown in Figure 2-4. If timing is not detected until after the active portion of the frame begins, then Timing_Lock will not go High until the next rising edge of vblank_in. The first frame of data will be invalid, and `video_data_out` will be set to 0. The next frame will output valid data, as shown in Figure 2-5.

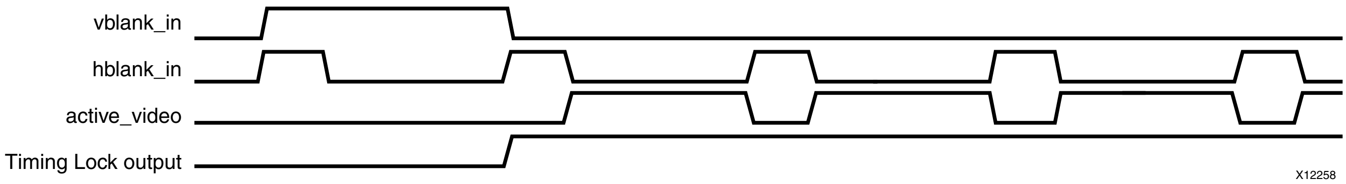


Figure 2-4: **Timing Lock Before Active Video**

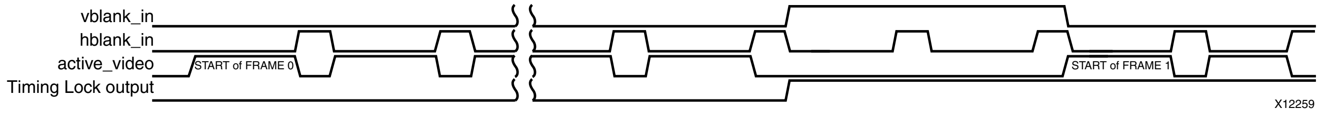


Figure 2-5: **Timing Lock After Active Video**

YCbCr data is packed on the `video_data` bus as shown in Figure 2-6, Figure 2-7, and Figure 2-8. For 4:4:4 chroma format, Y, Cb, and Cr are on a single bus and run at full sample rate, as shown in Figure 2-6.

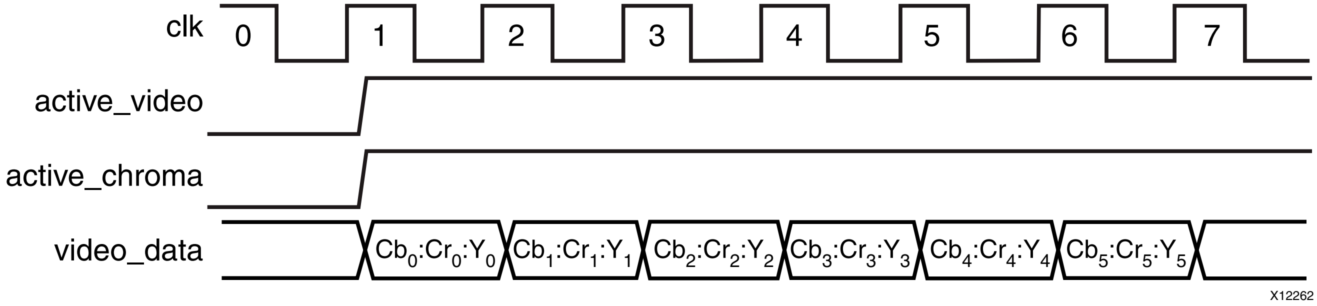


Figure 2-6: **YCbCr 4:4:4**

For 4:2:2, Cb and Cr are interleaved on bits [2*DATA_WIDTH-1 - DATA_WIDTH] of the `video_data` bus. The first active video data sample contains Cb first, as shown in Figure 2-7.

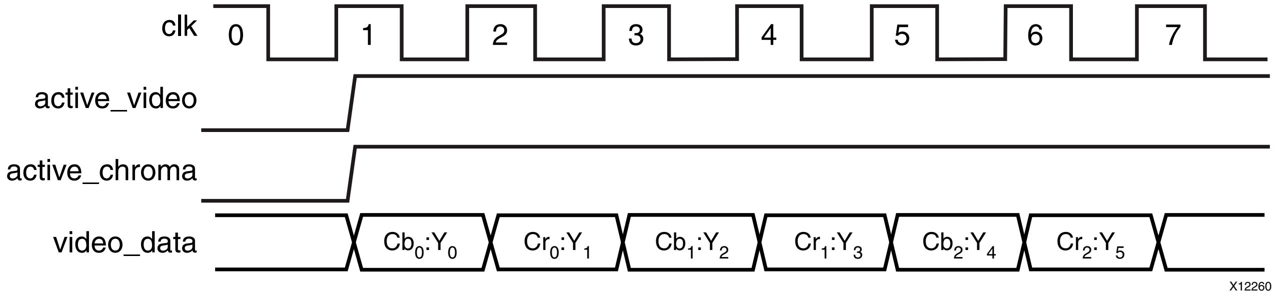


Figure 2-7: **YCbCr 4:2:2**

For 4:2:0, the format is similar to 4:2:2, except only the alternate lines have valid chroma, as shown in Figure 2-8. The `chroma_parity` port/register signals whether the first line has chroma information. Cb and Cr samples are interleaved as per 4:2:2.

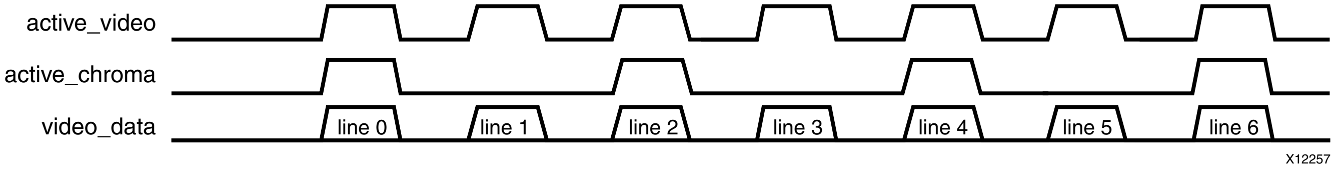


Figure 2-8: **YCbCr 4:2:0**

Deasserting CE suspends processing, which may be useful for data-throttling. This temporarily ceases the processing of a video stream to match the delay of other processing components.

When SCLR is asserted, all data and control signal outputs are forced to zero. If the input control signal was High at the time SCLR was asserted, the corresponding output control signal goes Low and stays Low until the next expected rising edge.

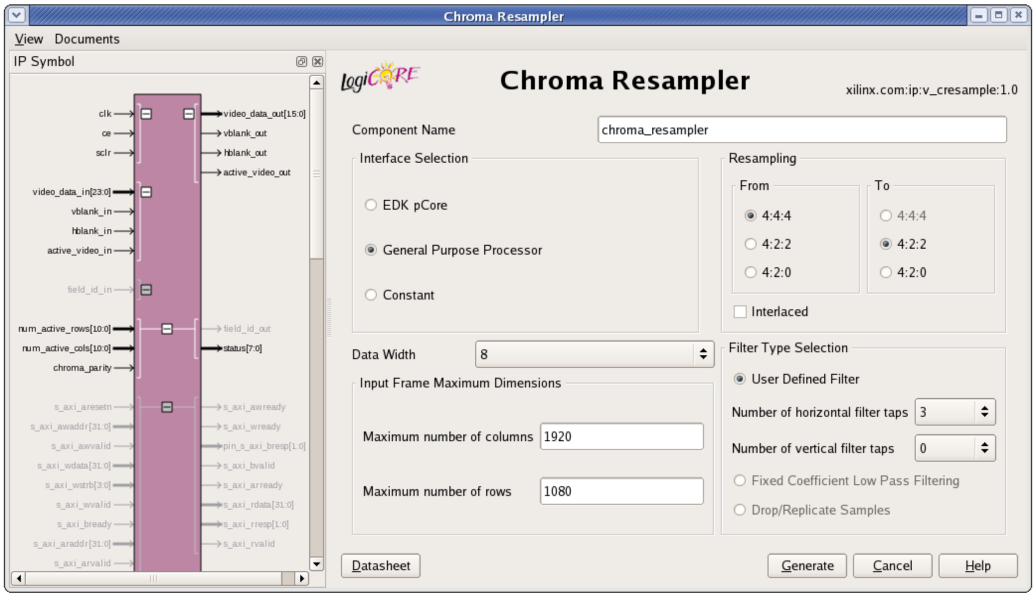
Chapter 3

Customizing and Generating the Core

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

GUI

The Chroma Resampler LogiCORE is easily configured to meet the developer's specific needs before instantiation through the CORE Generator GUI. This section provides a quick reference to the parameters that can be configured at generation time.



Chroma Resampler

xilinx.com:ip:v_cresample:1.0

Component Name: chroma_resampler

Interface Selection
- ○ EDK pCore
- ● General Purpose Processor
- ○ Constant

Resampling

From: ● 4:4:4 ○ 4:2:2 ○ 4:2:0

To: ○ 4:4:4 ● 4:2:2 ○ 4:2:0

□ Interlaced

Data Width: 8

Input Frame Maximum Dimensions
- Maximum number of columns: 1920
- Maximum number of rows: 1080

Filter Type Selection
- ● User Defined Filter
  - Number of horizontal filter taps: 3
  - Number of vertical filter taps: 0
- ○ Fixed Coefficient Low Pass Filtering
- ○ Drop/Replicate Samples

Figure 3-1: **Chroma Resampler GUI**

- 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: This option allows for the configuration of three different interfaces for the core.
  - EDK pCore Interface: CORE Generator will generate a pCore that can be easily imported into an EDK project as a hardware peripheral and coefficients can be programmed via registers. See Generating the EDK pCore, page 31.
  - General Purpose Processor Interface: CORE Generator will generate a set of ports to be used to program the core.
  - Constant Interface

- Resampling: Select the input and output chroma formats. The supported formats are 4:4:4, 4:2:2, and 4:2:0.
- Interlaced: This box should be checked for interlaced video. The default is progressive video. For interlaced video, it is assumed the number of rows is the same for each field. There is an input signal (`field_id_in`) to specify whether the odd or even field is first.
- Data Width: Specifies the bit width of the input channel. The allowed values are 8, 10, and 12.
- Maximum Number of Columns: The maximum number of active columns in the video input line. This value can be any integer from 32 to 4095.
- Maximum Number of Rows: The maximum number of active rows in the video input line. This value can be any integer from 32 to 4095. For interlaced video, it is assumed the number of rows is the same for each field.
- Filter Type Selection:
    - User Defined Filter: Users can program the filter coefficients through the GPP or EDK pCore interface (option not available with the Constant Interface). Filters are initialized with the coefficients used for the Fixed Coefficient Low Pass Filtering option.
        - Number of Horizontal Taps: The number of DSP48 multipliers that may be used in the system for the horizontal filter. Maximum is 12. The drop down menu will limit the number of taps to even or odd based on the conversion selected.

          Here is the possible number of horizontal taps based on conversion type:

            - 4:4:4 to 4:2:2: 3, 5, 7, 9, 11
            - 4:2:2 to 4:4:4: 2, 4, 6, 8, 10, 12
            - 4:2:2 to 4:2:0: 0 (vertical filter only)
            - 4:2:0 to 4:2:2: 0 (vertical filter only)
            - 4:4:4 to 4:2:0: 3, 5, 7, 9, 11
            - 4:2:0 to 4:4:4: 2, 4, 6, 8, 10, 12
        - Number of Vertical Taps: Number of DSP48 multipliers that can be used in the system for the vertical filter. Maximum is 8. The drop down menu will limit the number of taps to be even.

          Here is the possible number of vertical taps based on conversion type:

            - 4:4:4 to 4:2:2: 0 (horizontal filter only)
            - 4:2:2 to 4:4:4: 0 (horizontal filter only)
            - 4:2:2 to 4:2:0: 2, 4, 6, 8
            - 4:2:0 to 4:2:2: 2, 4, 6, 8
            - 4:4:4 to 4:2:0: 2, 4, 6, 8
            - 4:2:0 to 4:4:4: 2, 4, 6, 8
        - Fixed Coefficient Low Pass Filtering: Filters are pre-defined and not programmable. The filters use only power of two coefficients. So no DSP48s are necessary. Linear interpolation is employed for the low pass filters used for anti-aliasing and interpolation. The default coefficients are described in

the Implementation in Chapter 5. Only the EDK pCore and Constant Interfaces support this option.

- Drop/Replicate Samples: Only the EDK pCore and Constant Interfaces support this option.

  - Using the drop option results in down conversion with no filter. Some samples are passed directly to the output, but others are dropped entirely, as appropriate. This occurs on a line-by-line basis and on a pixel-by-pixel basis.

  - The replicate option is available in all up converters. It applies in both vertical and horizontal domains as appropriate. Using the replicate option results in up conversion with no filter. Replication of the previous input sample occurs instead.

Generating the EDK pCore

When the developer selects the EDK pCore interface, Xilinx CORE Generator creates a pCore and all support files that can be added to an EDK project as a hardware peripheral. This pCore provides a memory mapped interface for the programmable registers within the core and a complete device driver to enable rapid application development.

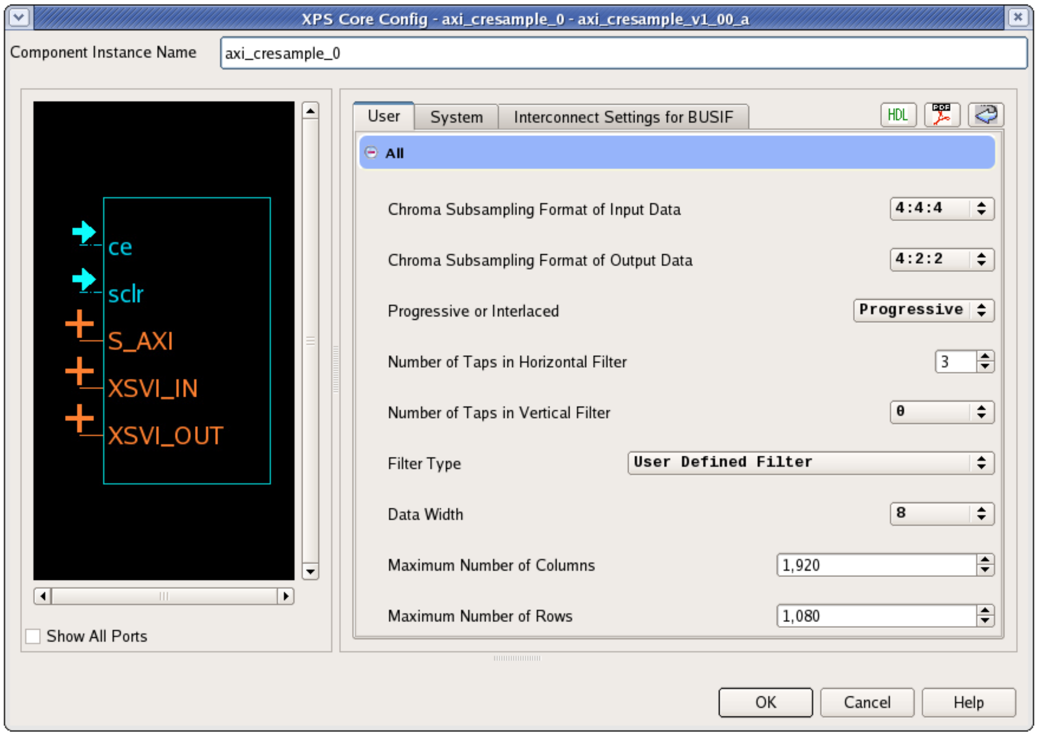
Xilinx CORE Generator will place all EDK pCore source files in the "pcores" subdirectory located in the core output directory. The core output directory is given the same name as the component. For example, if the component name is set to "chroma_resampler," then the EDK pCore source files will be located in the following directory:

```
<coregen project directory>/chroma_resampler/pcores/
axi_cresample_v1_00_a
```

The pCore should be copied to the user's `<EDK_Project>/pcores` directory or to a user pCores repository.

Parameter Modification in CORE Generator

EDK pCore parameters found in `<coregen project directory>/chroma_resampler/pcores/axi_cresample_v1_00_a/data/cresample_v2_1_0.mpd` cannot be modified in the Xilinx CORE Generator tool. Parameters shown on the CORE Generator GUI will be disabled if the EDK pCore (AXI4-Lite) Interface is selected. It is recommended that all parameter changes be made with the Chroma Resampler pCore GUI in the EDK environment, as shown in Figure 3-2.



XPS Core Config - axi_cresample_0 - axi_cresample_v1_00_a

Component Instance Name: axi_cresample_0

Chroma Subsampling Format of Input Data: 4:4:4

Chroma Subsampling Format of Output Data: 4:2:2

Progressive or Interlaced: Progressive

Number of Taps in Horizontal Filter: 3

Number of Taps in Vertical Filter: 0

Filter Type: User Defined Filter

Data Width: 8

Maximum Number of Columns: 1,920

Maximum Number of Rows: 1,080

Figure 3-2: **EDK pCore GUI**

Parameter Values in the XCO File

Table 3-1 defines valid entries for the XCO parameters. Parameters are not case sensitive. Default values are displayed in bold. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator tool GUI to configure the core and perform range and parameter value checking.

Table 3-1: **XCO Parameters**

XCO Parameter	Valid Values
component_name	ASCII text using characters: a..z, 0..9 and "_" starting with a letter
chroma_format_in	1, 2, **3**
chroma_format_out	1, **2**, 3
convert type	**0**, 1, 2
data_width	**8**, 10, 12
interface	0, **1**, 2
interlaced	true, **false**
max_cols	32 – 4095 (default = **1920**)
max_rows	32 – 4095 (default = **1080**)
num_h_taps	0 – 12 (default = **3**)
num_v_taps	0 – 8 (default = **0**)

Output Generation

The output files generated from the Xilinx CORE Generator software for the core depend upon whether the interface selection is set to EDK pCore or General Purpose Processor/ Constant. The output files are placed in the project directory.

EDK pCore Files

When the interface selection is set to EDK pCore, CORE Generator will output the core as a pCore that can be easily incorporated into an EDK project. The pCore output consists of a hardware pCore and a software driver. The pCore has the following structure in the <Component_Name> directory:

- drivers
  - cresample_v1_00_a
    - data
    - build
    - example
    - src
- pcores
  - axi_cresample_v1_00_a
    - data
    - hdl
      - vhdl

File Details

This section details the files contained in each directory.

<project directory>

This is the top-level directory. It contains xco and other assorted files, as shown in Table 3-2.

Table 3-2: **<project directory> Files for EDK pCore Interfaces**

Name	Description
<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_cresample_v1_00_a/data

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

< project directory>/<component_name>/pcores/axi_cresample_v1_00_a/hdl/ vhdl

This directory contains the HDL files that implement the pCore.

< project directory>/<component_name>/drivers/cresample_v1_00_a/data

This directory contains files that SDK uses to define the operation of the pCore's software driver.

< project directory>/<component_name>/drivers/cresample_v1_00_a/example

This directory contains some example code using the pCore's software driver.

< project directory>/<component_name>/drivers/cresample_v1_00_a/src

This directory contains the source code of the pCore's software driver.

Table 3-3: **src Files for EDK pCore Interfaces**

Name	Description
cresample.c	Provides the API access to all features of the device driver.
cresample.h	Provides the API access to all features of the device driver.

General Purpose Processor and Constant Interface Files

When the interface selection is set to General Purpose Processor, CORE Generator will output 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 `<project directory>`.

File Details

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

Table 3-4: **<project directory> Files for GPP and Constant Interfaces**

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 template for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation model for the core. It is 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.

Chapter 4

Constraining the Core

This chapter details any applicable constraints for the Chroma Resampler core.

Required Constraints

The `clk` pin should be constrained at the maximum pixel clock rate desired for the video stream.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. This core has not been characterized for use in Lower Power devices.

Clock Frequencies

There are no specific clock frequency requirements for this core other than the Maximum Frequency discussed in the Performance section.

Clock Management

There is only one clock for this core. For pCore users, the AXI interconnect handles the cross-clock domain crossing.

Clock Placement

There are no specific clock placement requirements for this core.

Banking

There are no specific banking rules for this core.

Transceiver Placement

There are no transceivers used in this core.

I/O Standard and Placement

There are no specific I/O standard or placement requirements.

Chapter 5

Designing with the Core

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

Sub-sampled Video Formats

The sub-sampling scheme is commonly expressed as a three part ratio J:a:b (for example, 4:2:2), that describes the number of luminance and chrominance samples in a conceptual region that is J pixels wide, and 2 pixels high. The parts are (in their respective order):

- J: Horizontal sampling reference (width of the conceptual region). This is usually 4.
- a: Number of chrominance samples (Cr, Cb) in the first row of J pixels.
- b: Number of (additional) chrominance samples (Cr, Cb) in the second row of J pixels.

To illustrate the most common sub-sampling schemes, Figure 5-1 introduces a graphical notation of sampling grid pixels.

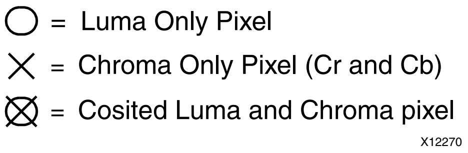

○ = Luma Only Pixel
✕ = Chroma Only Pixel (Cr and Cb)
⊗ = Cosited Luma and Chroma pixel
X12270

Figure 5-1: **Notation**

4:4:4

Similar to RGB, the 4:4:4 format is used for image capture and display purposes. Cb and Cr channels are sampled at the same rate as luminance. Hence, all pixel locations have luma and chroma data co-sited, as shown in Figure 5-2.

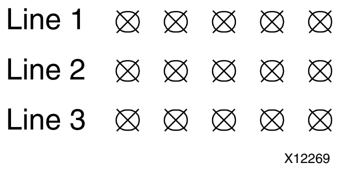


Figure 5-2: **4:4:4 Format**

4:2:2

This format contains horizontally sub-sampled chroma. For every two luma samples, there is an associated pair of Cb and Cr samples. The sub-sampled chroma locations are co-sited with alternate luma samples, as shown in Figure 5-3.

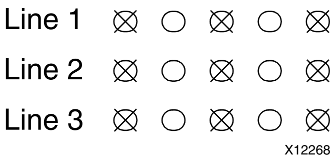


Figure 5-3: **4:2:2 Format**

4:2:0 (MPEG2, MPEG-4 Part 2 and H.264)

The version of 4:2:0 that is used for MPEG2, MPEG-4 Part 2 and H.264 encoding contains horizontally and vertically sub-sampled chroma. Additionally, the chroma sampling locations are not co-sited with the luma pixels. In fact, vertical interpolation is used to create the chroma samples, and their effective location puts them directly between alternate pairs of original scanlines. Horizontal chroma positions are co-sited with alternate luma samples.

The sampling positions of a progressive picture are shown in Figure 5-4.

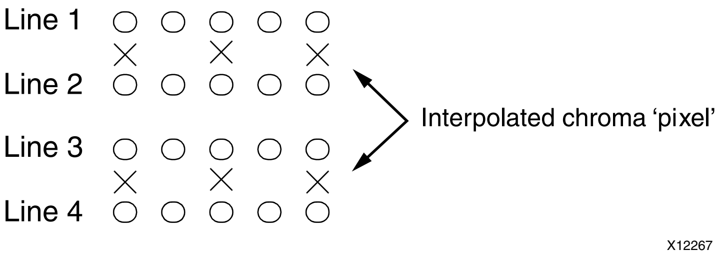


Figure 5-4: **: 4:2:0 Progressive Format**

The sampling positions of an interlaced picture are shown in Figure 5-5.

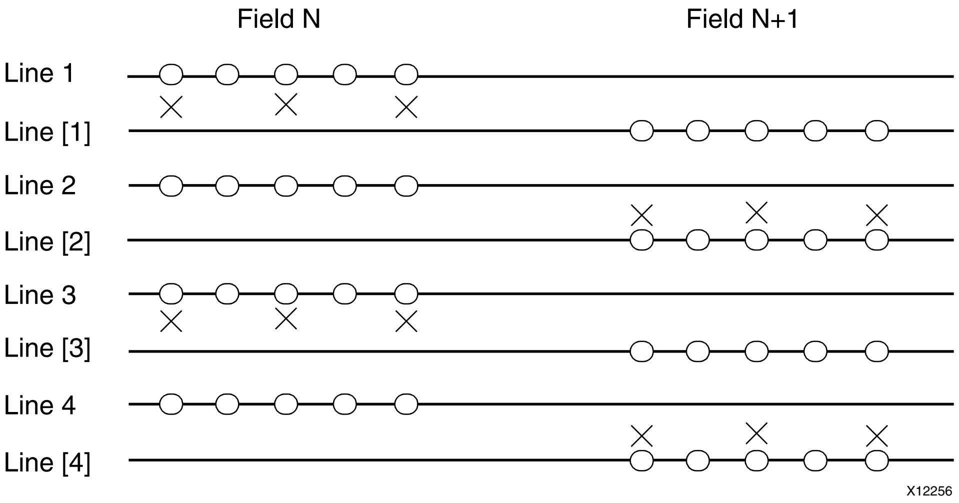


Figure 5-5: **4:2:0 Interlaced Format**

Implementation

Between the three supported sub-sampling formats (4:4:4, 4:2:2, 4:2:0), there are six conversions available. Conversion is achieved using a FIR filter approach. Some require filtering in only the horizontal dimension or only in the vertical dimension, and in some cases in both the horizontal and the vertical dimensions. These are detailed in Table 5-1 along with default filter information.

Table 5-1: **Filter Summary**

Converter	Filter Configuration	Default FIR Size	Notes
4:4:4 to 4:2:2	Horizontal anti-aliasing	3 Horizontal Taps	
4:4:4 to 4:2:0	Separable 2D anti-aliasing	2 Vertical Taps x 3 Horizontal Taps	
4:2:2 to 4:4:4	Horizontal Interpolation	2 Horizontal Taps	Only phase1 needed
4:2:2 to 4:2:0	Vertical anti-aliasing	2 Vertical Taps	2 phases
4:2:0 to 4:4:4	Separable 2D Interpolation	2 Horizontal Taps by 2 Vertical Taps	
4:2:0 to 4:2:2	Vertical Interpolation	2 Horizontal Taps by 2 Vertical Taps	2 phases

Three implementation options are offered for each conversion operation:

- DSP48 based filter with programmable coefficients and programmable number of taps. The maximum number of vertical taps is 8. The maximum number of horizontal taps is 12. 2D filters must be separable. Coefficients are in the range [-2, 2), represented in 16-bit signed, fixed-point format with 2 integer bits and 14 fractional bits.
- Pre-defined fixed coefficient, non-programmable filter with power of two coefficients (using only shifts and additions for filtering therefore no DSP48s are used). Default

coefficients implement linear interpolation for the interpolation and anti-aliasing low pass filters.

- The simplest, lowest footprint solution is to simply drop (decimation) or replicate (interpolation) samples. For down sampling, some samples are passed directly to the output, but others are dropped entirely as appropriate. For up converters, replication of the previous input sample occurs.

Convert 4:2:2 to 4:4:4

This conversion is a 1:2 horizontal interpolation operation, implemented using a two-phase polyphase FIR filter. One of the two output pixels is co-sited with one of the input sample. The ideal output is achieved simply by replicating this input sample. Therefore, for phase 0, no coefficients are needed because the input sample is replicated.

In order to evaluate output pixel $o_{x,y}$, the FIR filter in the core convolves COEFk_HPHASEp_x , where k is the coefficient index, $i_{x,y}$ are pixels from the input image, p is the interpolation phase (0 or 1, depending on x) and $[\ ]_{\mathrm{m}}^{\mathrm{M}}$ represents rounding with clipping at M, and clamping at m.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\mathrm{COEFk\_HPHASEp}_x\right]_0^{2^{DW}-1} \qquad \textit{Equation 5-1}$$

In phase 1, COEF00_HPHASE1 is the coefficient applied to the most recent input sample in the filter aperture. Figure 5-6 illustrates coefficient use for a four tap filter example, with simplified nomenclature a= COEF00_HPHASE1, b= COEF01_HPHASE1, c= COEF02_HPHASE1, and d= COEF03_HPHASE1.

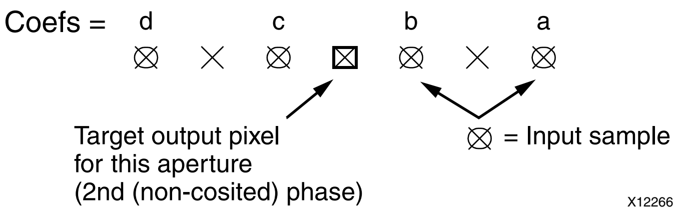


Figure 5-6: **4:2:2 to 4:4:4 Coefficient Configuration**

For the default two-tap polyphase filter, for the second phase, the default coefficients are [0.5 0.5].

Convert 4:4:4 to 4:2:2

This conversion is a horizontal 2:1 decimation operation, implemented using a low-pass FIR filter to suppress chroma aliasing. In order to evaluate output pixel $o_{x,y}$, the FIR filter in the core convolves COEFk_HPHASE0 , where k is the coefficient index, $i_{x,y}$ are pixels from the input image, and $[\ ]_{\mathrm{m}}^{\mathrm{M}}$ represents rounding with clipping at M, and clamping at m.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\mathrm{COEFk\_HPHASE0}\right]_0^{2^{DW}-1} \qquad \textit{Equation 5-2}$$

In phase 0, COEF00_HPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 5-7 illustrates coefficient use for a 5 tap filter example, with simplified nomenclature a= COEF00_HPHASE0, b= COEF01_HPHASE0, c= COEF02_HPHASE0, d= COEF03_HPHASE0, and e= COEF04_HPHASE0.

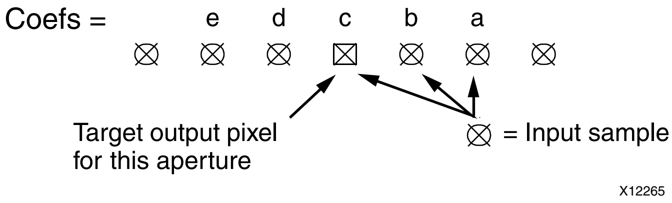


Figure 5-7: **4:4:4 to 4:2:2 Coefficient Configuration**

The default coefficients are [0.25 0.5 0.25].

Convert 4:2:0 to 4:2:2

This conversion is a 1:2 vertical interpolation operation, implemented using a 2-phase polyphase FIR filter. In order to evaluate output pixel $o_{x,y}$, the FIR filter in the core convolves COEFk_VPHASEp, where k is the coefficient index, p_y is the interpolation phase, $i_{x,y}$ are pixels from the input image, and $[\ ]_{\mathrm{m}}^{\mathrm{M}}$ represents rounding with clipping at M, and clamping at m.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y} \mathrm{COEFk\_VPHASEp}_y \right]_0^{2^{DW}-1} \qquad \textit{Equation 5-3}$$

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter Figure 5-8 illustrates coefficient use for a four tap filter example, with simplified nomenclature a= COEF00_VPHASE0, b= COEF01_VPHASE0, c= COEF02_VPHASE0, and d= COEF03_VPHASE0.

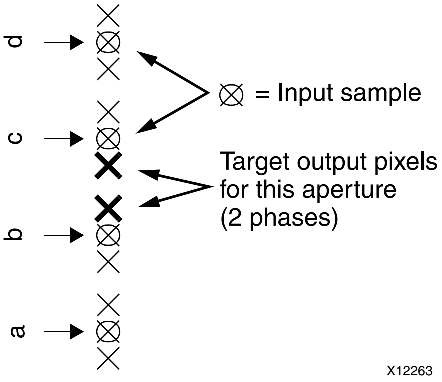


Figure 5-8: **4:2:0 to 4:2:2 Coefficient Configuration**

For progressive video, the default coefficients for phase 0 are [0.25 0.75], for phase 1 are [0.75 0.25].

For interlaced video, the default coefficients

- For the odd field, phase 0 defaults are [3/8 5/8], for phase1 are [7/8 1/8].

- For the even field, phase 0 defaults are [1/8 7/8], for phase1 are [5/8 3/8].

For the even field of interlaced data, the coefficients for phase 0 and phase 1 are swapped, and the filter coefficients for each filter are reversed.

Convert 4:2:2 to 4:2:0

This conversion is a vertical 2:1 decimation operation, implemented using a low-pass FIR filter to suppress chroma aliasing. In order to evaluate output pixel $o_{x,y}$, the FIR filter in the core convolves COEF*k*_VPHASE0 , where *k* is the coefficient index, $i_{x,y}$ are pixels from the input image, and $[\ ]_m^M$ represents rounding with clipping at *M*, and clamping at *m*.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y} \text{COEFk\_VPHASE0} \right]_0^{2^{DW}-1}$$

Equation 5-4

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 5-9 illustrates coefficient use for a four tap filter example, with simplified nomenclature a= COEF00_VPHASE0, b= COEF01_VPHASE0, c= COEF02_VPHASE0, and d= COEF03_VPHASE0.

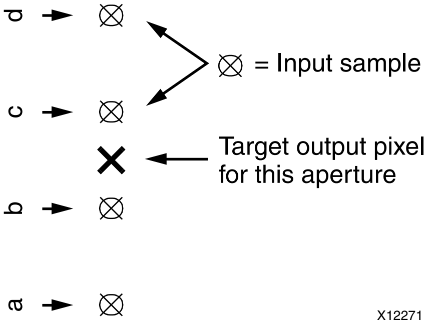


Figure 5-9: **4:2:2 to 4:2:0 Coefficient Configuration**

For progressive video, the default coefficients are [0.5 0.5]. For interlaced video, the default coefficients are [0.25 0.75] for the odd field. For the even field, the default coefficients are reversed: [0.75 0.25].

Convert 4:2:0 to 4:4:4

This conversion performs interpolation both vertically and horizontally. This is equivalent to a 2D separable filter implemented by cascading the 4:2:0 to 4:2:2 block and the 4:2:2 to 4:4:4 block. Quantized vertical filter results are filtered by the horizontal filter, which in turn quantizes results back to the [0 - 2^{DW}-1] range.

Intermediate 4:2:2 chroma values are computed using Equation 5-3. The resulting computation is shown in Equation 5-5.

$$t_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} i_{x,y-k} \text{COEFk\_VPHASEp}_y \right]_0^{2^{DW}-1}$$

Equation 5-5

Next, the values are filtered according to Equation 5-1. The resulting computation is shown in Equation 5-6.

$$o_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} t_{x-k,y} \mathrm{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \qquad \text{Equation 5-6}$$

Default coefficients are the same as defined in Convert 4:2:0 to 4:2:2 and Convert 4:2:2 to 4:4:4.

For the default two-tap polyphase filter, for the second phase, the default horizontal phase 1 coefficients are [0.5 0.5].

For progressive video, the default vertical coefficients for phase 0 are [0.25 0.75], for phase 1 are [0.75 0.25].

For interlaced video, the default vertical coefficients

- For the odd field, phase 0 defaults are [3/8 5/8], for phase1 are [7/8 1/8].
- For the even field, phase 0 defaults are [1/8 7/8], for phase1 are [5/8 3/8].

For the even field of interlaced data, the coefficients for phase 0 and phase 1 are swapped, and the filter coefficients for each filter are reversed.

Convert 4:4:4 to 4:2:0

This conversion performs decimation by 2 both vertically and horizontally. This is equivalent to a 2D separable filter implemented by cascading the 4:4:4 to 4:2:2 block and the 4:2:2 to 4:2:0 block. Quantized horizontal filter results are filtered by the vertical filter, which in turn quantizes results back to the [0 - 2DW-1] range.

Intermediate 4:2:2 chroma values are computed using . The resulting computation is shown in Equation 5-7.

$$t_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} i_{x-k,y} \mathrm{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \qquad \text{Equation 5-7}$$

Next, these values are filtered according to Equation 5-4. The resulting computation is shown in Equation 5-8.

$$o_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} t_{x,y-k} \mathrm{COEFk\_VPHASE0} \right]_0^{2^{DW}-1} \qquad \text{Equation 5-8}$$

Default coefficients are the same as defined in Convert 4:4:4 to 4:2:2 and Convert 4:2:2 to 4:2:0.

The default horizontal coefficients are [0.25 0.5 0.25].

For progressive video, the default vertical coefficients are [0.5 0.5]. For interlaced video, the default vertical coefficients are [0.25 0.75] for the odd field. For the even field, the default vertical coefficients are reversed: [0.75 0.25].

Computation Bit Width Growth

Full precision (DATA_WIDTH+16+log2(N_{Taps}) bits) is maintained during the FIR convolution operation.

FIR filter outputs are rounded to DATA_WIDTH bits by adding half an output LSB in the full precision domain prior to truncation. Clipping and clamping of the output data prevents overflows and underflows. Data is clipped and clamped at 2^{DATA_WIDTH} - 1 and 0.

Edge Padding

The edge pixels of images are replicated prior to filtering to avoid image artifacts.

General Design Guidelines

The upsampling and downsampling performed during the chroma format conversion is implemented with low pass filters for the interpolation and anti-aliasing.

The Chroma Resampler core offers a horizontal filter with a maximum of 12 taps and two phases, as well as a vertical filter with a maximum of eight taps and two phases. For conversions requiring up/down sampling in both horizontal and vertical directions, 2D separable filters are offered.

The number of taps used is defined in the GUI. The GUI will limit the number of taps to be even or odd depending on the preferred filter length for each conversion type. Only a subset of the coefficients will be used depending on the conversion type and filter size selected.

Each coefficient has 16 bits: 2 integer bits (one sign bit) and 14 fractional bits. The sign bit is the MSB. For example, a coefficient with a value of 1 is represented with this bit vector [0100000000000000].

The coefficients should sum to exactly 1 to achieve unity gain. If they sum to less than 1, some loss of dynamic range is observed. The valid range of coefficient values is [-2,2).

The default filter coefficients are defined in Implementation, page 38.

Clocking

The Gamma Correction core has one clock (`clk`) that is used to clock the entire core. This includes the AXI interface and the core logic.

Resets

The Gamma Correction core has one reset (`sclr`) that is used for the entire core. The reset is active High.

Protocol Description

For the pCore version of the Gamma Correction core, the register interface is compliant with the AXI4-Lite interface.

Chapter 6

Detailed Example Design

This chapter includes information about the Chroma Resampler test bench.

Demonstration Test Bench

A demonstration test bench is provided as a simple introductory package that enables core users to observe the core generated by CORE Generator operating in a waveform simulator. The user is encouraged to modify the test bench to match the timing of the targeted application and verify the input and output waveforms.

The latest version of the test bench is available for download on the Chroma Resampler product page at:

http://www.xilinx.com/products/intellectual-property/EF-DI-CHROM-RESAMP.htm

Directory and File Contents

The directory structure within the top-level folder is described below:

- Expected: Contains the expected/golden data generated using the C model used by the test bench to compare actual output data.
- Stimuli: Contains the pre-generated input data used by the test bench to stimulate the core (including register programming values).
- Results: Actual output data will be written to a file in this folder.
- src: Contains the .vhd & .xco files of the core.
    - The .vhd file is a netlist generated using CORE Generator.
    - You can regenerate a new netlist using the .xco file in CORE Generator.
- tb_src: Contains the top-level test bench design.
    - This directory also contains other packages used by the test bench.
    - A .cfg file is included which contains filter coefficients for updating the GPP ports.
- `isim_wave.wcfg`: Waveform configuration file for iSim
- `mti_wave.do`: Waveform configuration file for ModelSim
- `run_isim.bat`: Runscript for iSim in Windows OS
- `run_isim.sh`: Runscript for iSim in Linux OS
- `run_mti.bat`: Runscript for ModelSim in Windows OS
- `run_mti.sh`: Runscript for ModelSim in Linux OS

Operating Instructions

Simulation using ModelSim for Linux :

- From the console, type **`source run_mti.sh`**.

Simulation using ModelSim for Windows :

- Double click on `run_mti.bat` file.

Simulation using iSim for Linux :

- From the console, type **`source run_isim.sh`**.

Simulation using iSim for Windows :

- Double click on `run_isim.bat` file.

Verification, Compliance, and Interoperability

This appendix includes information about core verification and testing.

Simulation

A highly parameterizable test bench used to test the Chroma Resampler core. Testing includes the following:

- Register accesses
- Testing all six possible conversions
- Testing all three interpolation/decimation methods
- Testing both progressive and interlaced data
- Testing of various filter sizes
- Testing of various frame sizes
- Testing of various data widths

Hardware Testing

The Chroma Resampler core has been tested in a variety of hardware platforms at Xilinx to represent a variety of parameterizations. A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-LITE Interface, and various other peripherals.

Appendix B

Debugging

This appendix contains debugging information and resources.

Evaluation Core Timeout

The Chroma Resampler hardware evaluation core times out after approximately eight hours of operation. The output is driven to zero. This results in a dark-green screen for YUV color systems.

Appendix C

Application Software Development

An example of the programming flow for updating the register is shown in Figure C-1.

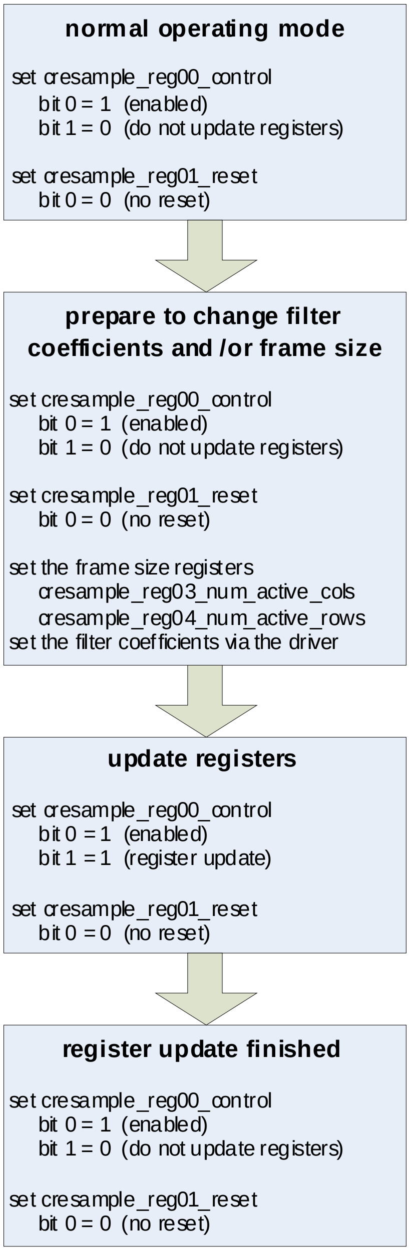

normal operating mode
set cresample_reg00_control
bit 0 = 1 (enabled)
bit 1 = 0 (do not update registers)
set cresample_reg01_reset
bit 0 = 0 (no reset)
prepare to change filter coefficients and /or frame size
set cresample_reg00_control
bit 0 = 1 (enabled)
bit 1 = 0 (do not update registers)
set cresample_reg01_reset
bit 0 = 0 (no reset)
set the frame size registers
cresample_reg03_num_active_cols
cresample_reg04_num_active_rows
set the filter coefficients via the driver
update registers
set cresample_reg00_control
bit 0 =1 (enabled)
bit 1 =1 (register update)
set cresample_reg01_reset
bit 0 =0 (no reset)
register update finished
set cresample_reg00_control
bit 0 = 1 (enabled)
bit 1 = 0 (do not update registers)
set cresample_reg01_reset
bit 0 = 0 (no reset)

Figure C-1: **Programming Flow Chart**

Device Drivers

The software API is provided to allow easy access to the pCore's registers defined in Table 2-6, page 19.

To use the API functions provided, the following two header files must be included in the user C code:

```
#include “cresample.h”
#include “xparameters.h”
```

The hardware settings, including the base address of the Chroma Resampler core, are defined in the `xparameters.h` file. The `cresample.h` file contains the macro function definitions for controlling the Chroma Resampler pCore.

For API function calls and user application integration examples, see the `example.c` file in the `cresample_v1_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the Chroma Resampler pCore API.

The functions included in the C driver (`cresample.c` and `cresample.h`) generated for the EDK pCore API are as follows:

CRESAMPLE_Enable(uint32 BaseAddress);

- This macro enables a Chroma Resampler instance.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

CRESAMPLE_Disable(uint32 BaseAddress);

- This macro disables a Chroma Resampler instance.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

CRESAMPLE_Reset(uint32 BaseAddress);

- This macro resets a Chroma Resampler instance. This reset affects the core immediately, and may cause image tearing. Reset affects the coefficient and frame size registers, forces `video_data_out` to 0, and forces timing signal outputs to their reset state until CRESAMPLE_ClearReset() is called.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

CRESAMPLE_ClearReset(uint32 BaseAddress);

- This macro clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

CRESAMPLE_RegUpdateEnable(uint32 BaseAddress);

- Calling RegUpdateEnable causes the Chroma Resampler to start using the updated register values on the next rising edge of `vblank_in`. Manually disable the register update after a sufficient amount of time to prevent continuous updates.
- This function only works when the Chroma Resampler core is enabled.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

CRESAMPLE_RegUpdateDisable(uint32 BaseAddress);

- Disabling the Register Update prevents the Chroma Resampler gain registers from updating. It is recommended that the Register Update be disabled while writing to the registers in the core, until the write operation is complete. While disabled, writes to the registers are stored, but do not affect the core's behavior.
- This function only works when the Chroma Resampler core is enabled.

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).

Configuring the Filter Coefficients

To simplify updating the filter coefficient registers, the following macros can be used. Filter coefficients are 16-bit signed numbers with 14 fractional bits. The drivers expect the coefficients to be converted to integer form. This means a coefficient value of 1, which is [0100000000000000], is fed to the driver as 16384.

configure_422_to_444(uint32 BaseAddress, NUM_H_TAPS, int *coef_phase1_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_H_TAPS is the number of horizontal filter coefficients defined in the GUI.
- coef_phase1_vector contains NUM_H_TAPS integer coefficients quantized to 16-bit fixed point format.
- There is no need to define phase 0 coefficients. Phase 0 always replicates the existing pixel.

configure_444_to_422(uint32 BaseAddress, int NUM_H_TAPS, int *coef_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_H_TAPS is the number of horizontal filter coefficients defined in the GUI.
- coef_vector contains NUM_H_TAPS integer coefficients quantized to 16-bit fixed point format.

configure_420_to_422(uint32 BaseAddress, int NUM_V_TAPS, int *coef_phase0_vector, int *coef_phase1_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_V_TAPS is the number of vertical filter coefficients defined in the GUI.
- coef_phase0_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format. These are the Phase 0 Vertical filter coefficients.
- coef_phase1_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format. These are the Phase 1 Vertical filter coefficients.

configure_422_to_420(uint32 BaseAddress, int NUM_V_TAPS, int *coef_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_V_TAPS is the number of vertical filter coefficients defined in the GUI.
- coef_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format.

configure_420_to_444(uint32 BaseAddress, int NUM_H_TAPS, int NUM_V_TAPS, int *hcoef_phase1_vector, int *vcoef_phase0_vector, int *vcoef_phase1_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_H_TAPS is the number of horizontal filter coefficients defined in the GUI.
- NUM_V_TAPS is the number of vertical filter coefficients defined in.the GUI
- hcoef_phase1_vector contains NUM_H_TAPS integer coefficients quantized to 16-bit fixed point format. There is no need to define horizontal phase 0 coefficients. Horizontal phase 0 always replicates the existing pixel.
- vcoef_phase0_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format. These are the Phase 0 Vertical filter coefficients.
- vcoef_phase1_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format. These are the Phase 1 Vertical filter coefficients.

configure_444_to_420(uint32 BaseAddress, int NUM_H_TAPS, int NUM_V_TAPS, int *hcoef_vector, int *vcoef_vector)

- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- NUM_H_TAPS is the number of horizontal filter coefficients defined in the GUI.
- NUM_V_TAPS is the number of vertical filter coefficients defined in the GUI.
- hcoef_vector contains NUM_H_TAPS integer coefficients quantized to 16-bit fixed point format.
- vcoef_vector contains NUM_V_TAPS integer coefficients quantized to 16-bit fixed point format.

Reading and Writing pCore Registers

This section contains macros for reading and writing to pCore registers.

CRESAMPLE_ReadReg(uint32 BaseAddress, uint32 RegOffset)

Each software register defined in Table 2-5, page 16 has a constant defined in `cresample.h` that is set to the offset for that register. Reading a value from a register uses the base address and offset for the register:

```
Xuint32 value = CRESAMPLE_ReadReg(XPAR_CRESAMPLE_0_BASEADDR,
CRESAMPLE_REG06_COEF00_HPHASE0);
```

This macro returns the 32-bit unsigned integer value of the register. The details of this macro are as follows:

- Read the given register.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from xparameters.h).
- RegOffset is the register offset of the register (defined in Table 7).

CRESAMPLE_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)

To write to a register, use the CRESAMPLE_WriteReg() function using the base address of the Chroma Resampler pCore instance (from `xparameters.h`), the offset of the desired register, and the data to write.

```
CRESAMPLE_WriteReg(XPAR_CRESAMPLE_0_BASEADDR,
CRESAMPLE_REG06_COEF00_HPHASE0, 16384);
```

The details of this macro are as follows:

- Write the given register.
- BaseAddress is the Xilinx EDK base address of the Chroma Resampler core (from `xparameters.h`).
- RegOffset is the register offset of the register (defined in Table 2-5, page 16).
- Data is the 32-bit value to write to the register.

Appendix D

C Model Reference

The Chroma Resampler core has a bit accurate C model designed for system modeling.

Features

- Bit-accurate with the Chroma Resampler v1.0 core
- Statically linked library (.lib for Windows)
- Dynamically linked library (.so for Linux)
- Available for 32-bit and 64-bit Windows platforms and 32-bit and 64-bit Linux platforms
- Supports all features of the Chroma Resampler core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code showing how to use the function is provided
- Example application C code wrapper file supports 8-bit YUV and BIN

Overview

The Chroma Resampler core has a bit-accurate C model for 32-bit and 64-bit Windows platforms and 32-bit and 64-bit Linux platforms. The model's interface consists of a set of C functions residing in a statically linked library (shared library).

See Using the C Model, page 56 for full details of the interface. A C code example of how to call the model is provided in C Model Example Code, page 61.

The model is bit accurate, as it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate, and it does not model the core's latency or its interface signals.

The latest version of the model is available for download on the Chroma Resampler product page at:

http://www.xilinx.com/products/intellectual-property/EF-DI-CHROM-RESAMP.htm

User Instructions

Unpacking and Model Contents

Unzip the `v_cresample_v1_0_bitacc_model.zip` file, containing the bit accurate model for the Chroma Resampler core. This produces the directory structure and files shown in Table D-1.

Table D-1: **Directory Structure and Files of Bit-Accurate Model**

File Name	Contents
README.txt	Release Notes
pg012_v_cresample.pdf	Chroma Resampler Product Guide
v_cresample_v1_0_bitacc_cmodel.h	Model header file
parsers.h	Header file for reading configuration file
yuv_utils.h	Header file declaring the YUV (.yuv) image file I/O functions
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions
run_bitacc_cmodel.c	Example code calling the C model
parsers.c	Code for reading configuration file
cresample.cfg	Configuration file containing the core parameter settings
input_image.yuv	Sample test image
input_image.hdr	Sample test image header file
/lin32	Precompiled bit-accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_cresample_v1_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_cresample_v1_0_bitacc_cmodel.so
/lin64	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_cresample_v1_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_cresample_v1_0_bitacc_cmodel.so
/nt32	Precompiled bit-accurate ANSI C reference model for simulation on 32-bit Windows platforms
libIp_v_cresample_v1_0_bitacc_cmodel.lib	Precompiled library file for nt32 compilation

Table D-1: **Directory Structure and Files of Bit-Accurate Model *(Cont'd)***

File Name	Contents
/nt64	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Windows platforms
libIp_v_cresample_v1_0_bitacc_cmodel.lib	Precompiled library file for nt64 compilation

Installation

For Linux systems, ensure that `libIp_v_cresample_v1_0_bitacc_cmodel.so` and `libstlport.so.5.1` are located is in the $LD_LIBRARY_PATH environment variable.

Software Requirements

The Chroma Resampler C models were compiled and tested with the software shown in Table D-2.

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

Platform	C Compiler
32-bit and 64-bit Linux	GCC 4.1.1
32-bit and 64-bit Windows	Microsoft Visual Studio 2005

Using the C Model

The bit-accurate C model is accessed through a set of functions and data structures declared in the header file `v_cresample_v1_0_bitacc_cmodel.h`.

Before using the model, the structures holding the inputs, generics and output of the Chroma Resampler instance have to be defined:

```
struct  xilinx_ip_v_cresample_v1_0_generics cresample_generics;
struct  xilinx_ip_v_cresample_v1_0_inputs   cresample_inputs;
struct  xilinx_ip_v_cresample_v1_0_outputs  cresample_outputs;
```

Declaration of these structs can be found in `v_cresample_v1_0_bitacc_cmodel.h`.

The generic parameters and default values are listed in Table D-3. For an actual instance of the core, these parameters can only be set during generation through the CORE Generator interface.

Table D-3: **Model Generic Parameters and Default Values**

Generic Variable	Type	Default Value	Range	Description
CHROMA_FORMAT_IN	Int	3	1, 2, 3	1=4:2:0, 2 = 4:2:2, 3=4:4:4
CHROMA_FORMAT_OUT	Int	2	1, 2, 3	1=4:2:0, 2 = 4:2:2, 3=4:4:4
INTERLACED	Int	0	0, 1	0 = progressive, 1 = interlaced

Table D-3: **Model Generic Parameters and Default Values**

Generic Variable	Type	Default Value	Range	Description
NUM_H_TAPS	Int	3	0 to 12	Allowed values depend on conversion • 4:4:4 to 4:2:2: 3, 5, 7, 9, 11 • 4:2:2 to 4:4:4: 2, 4, 6, 8, 10, 12 • 4:2:2 to 4:2:0: 0 (vertical filter only) • 4:2:0 to 4:2:2: 0 (vertical filter only) • 4:4:4 to 4:2:0: 3, 5, 7, 9, 11 • 4:2:0 to 4:4:4: 2, 4, 6, 8, 10, 12
NUM_V_TAPS	Int	0	0 to 8	Allowed values depend on conversion • 4:4:4 to 4:2:2: 0 (horizontal filter only) • 4:2:2 to 4:4:4: 0 (horizontal filter only) • 4:2:2 to 4:2:0: 2, 4, 6, 8 • 4:2:0 to 4:2:2: 2, 4, 6, 8 • 4:4:4 to 4:2:0: 2, 4, 6, 8 • 4:2:0 to 4:4:4: 2, 4, 6, 8
CONVERT_TYPE	Int	0	0, 1, 2	0 = User Defined Filter 1 = Fixed Coefficient Filter 2 = Drop/Replicate
DATA_WIDTH	Int	8	8,10,12	Data width of each component Y, Cb, Cr

Calling xilinx_ip_v_cresample_v1_0_get_default_generics(&cresample_generics) initializes the generics structure with the defaults, listed in Table D-3.

Filter coefficients can also be set dynamically through the pCore and General Purpose Processor interfaces; therefore this value is passed as an input to the core, along with the actual test image, or video sequence, as shown in Table D-4.

Table D-4: **Core Generic Parameters and Default Values**

Input Variable	Type	Default Value	Range	Description
video_in	video_struct	null	N/A	Container to hold input image or video data[1].
field_id	Int	1	0,1	For interlaced video. 1 = odd field first 0 = even field first
coefs_hphase0	float	0	[-2,2)	Array of coefficients used for phase 0 of the horizontal filter. Coefficient values should be quantized to 16 bits (14 fractional bits).

Table D-4: **Core Generic Parameters and Default Values *(Cont'd)***

Input Variable	Type	Default Value	Range	Description
coefs_hphase1	float	0	[-2,2)	Array of coefficients used for phase 1 of the horizontal filter. Coefficient values should be quantized to 16 bits (14 fractional bits).
coefs_vphase0	float	0	[-2,2)	Array of coefficients used for phase 0 of the vertical filter. Coefficient values should be quantized to 16 bits (14 fractional bits).
coefs_vphase1	float	0	[-2,2)	Array of coefficients used for phase 1 of the vertical filter. Coefficient values should be quantized to 16 bits (14 fractional bits).

1. For the description of the input structure, see Initializing the Chroma Resampler input video structure, page 60.

The structure cresample_inputs defines the values of run-time parameters and the actual input image.

Calling xilinx_ip_v_cresample_v1_0_get_default_inputs(&noise_generics, &noise_inputs) initializes the input structure with the default values, as described in Table D-4.

Note: The video_in variable is not initialized, because the initialization depends on the actual test image to be simulated. Chroma Resampler Input and Output Video Structure, page 58 describes the initialization of the video_in structure.

After the inputs are defined the model can be simulated by calling the function:

```
int xilinx_ip_v_cresample_v1_0_bitacc_simulate(
  struct xilinx_ip_v_cresample_v1_0_generics* generics,
  struct xilinx_ip_v_cresample_v1_0_inputs*   inputs,
  struct xilinx_ip_v_cresample_v1_0_outputs*  outputs).
```

Results are provided in the outputs structure, which contains only one member of type video_struct.

After the outputs are evaluated and/or saved, dynamically allocated memory for input and output video structures are released by calling the function

```
void xilinx_ip_v_cresample_v1_0_destroy(
  struct xilinx_ip_v_cresample_v1_0_inputs *input,
  struct xilinx_ip_v_cresample_v1_0_outputs *output).
```

Successful execution of all provided functions, except for the destroy function, return value 0. Otherwise, a non-zero error code indicates that problems were encountered during function calls.

Chroma Resampler Input and Output Video Structure

Input images or video streams can be provided to the Chroma Resampler 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 D-5 details the variables of the video structure.

Table D-5: **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. This variable pertains 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.
cols	Number of columns per frame. This variable pertains 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.
bits_per_component	Number of bits per color channel / component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table D-6.
data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

Table D-6 details the modes and representations.

Table D-6: **Named Video Modes with Corresponding Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome – Luminance only.
FORMAT_RGB	3	RGB image/video data
FORMAT_C444	3	4:4:4 YUV, or YCrCb image/video data
FORMAT_C422	3	4:2:2 format YUV video (u,v chrominance channels horizontally sub-sampled)
FORMAT_C420	3	4:2:0 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	4:2:0 YUV video with Motion
FORMAT_C422_M	5	4:2:2 YUV video with Motion

Table D-6: **Named Video Modes with Corresponding Planes and Representations**

Mode	Planes	Video Representation
FORMAT_C444_M	5	4:4:4 YUV video with Motion
FORMAT_RGBM	5	RGB video with Motion

The Chroma Resampler C model supports the following modes:

- FORMAT_C444
- FORMAT_C422
- FORMAT_C420

Initializing the Chroma Resampler input video structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `yuv_utils.h` and `video_utils.h` header files packaged with the bit-accurate C models contain functions to facilitate file I/O.

YUV Image/Video Files

The header `yuv_utils.h` declares functions that help access files in standard YUV format. It operates on images with 3 planes (Y, U, and V). Functions int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video); and int read_yuv8(FILE *infile, struct yuv8_video_struct *yuv8_video); operate on arguments of type yuv8_video_struct, which is defined in `yuv_utils.h`.

Exchanging data between yuv8_video_struct and general video_struct type frames/videos is facilitated by the following 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 initialized either as static or dynamic variables (for example, pointing to a structure which has been allocated in memory). 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 the results.

Binary Image/Video Files

The header `video_utils.h` declares functions that help load and save generalized video files in raw, uncompressed format (BIN files). Functions int read_video(FILE* infile, struct video_struct* in_video); and int write_video(FILE* outfile, struct video_struct* out_video); effectively serialize the video_struct structure. The corresponding file contains a small, plain text header defining, “Mode”, “Frames”, “Rows”, “Columns”, and “Bits per Component”. The plain text header is followed by binary data that is 16 bits per component in scan line continuous format. Subsequent frames contain as many component planes as defined by the video mode value selected. In addition, the size (rows, columns) of component planes may differ within each frame as defined by the actual video mode selected.

Working with video_struct Containers

Header file `video_utils.h` defines the following 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 D-6, page 59. 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 the 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]
      }
    }
  }
}
```

C Model Example Code

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

After following the compilation instructions, run the example executable. The executable takes the path to the input file, the path to the output file, and the configuration file name as parameters. If invoked with insufficient parameters, the following help message is printed:

```
Usage: run_bitacc_cmodel config_file
config_file    : path/name of the configuration file
```

During successful execution, the corresponding YUV or BIN output file is created.

Example Code Configuration File

The example code reads a configuration file which defines all the generic and input variables. An example configuration file is given in the zip file.

```
#######################################################################
```

```
#
#  cresample.cfg: Chroma Resampler example configuration file
#
######################################################################

# Generic variables
CSET CHROMA_FORMAT_IN = 3                          # allowed values: 3=444, 2=422, 1=420
CSET CHROMA_FORMAT_OUT = 2                         # allowed values: 3=444, 2=422, 1=420
CSET INTERLACED=false                              # false=progressive, true=interlaced
CSET NUM_H_TAPS=3                                  # number of horizontal taps
                                                   # see product guide for allowed values
CSET NUM_V_TAPS=1                                  # number of vertical taps
                                                   # see product guide for allowed values
CSET CONVERT_TYPE=0                                # 2=Drop/Replicate
                                                   # 1=Fixed Coefficient Filter
                                                   # 0=User Defined Filter
CSET DATA_WIDTH=8                                  # allowed values: 8, 10, 12

# Input variables
CSET FIELD_ID=1                                    # 0=even field first, 1=odd field first

# Input Image/Video
CSET INPUT_FILE_NAME = input_image.yuv             # name of input file
                                                   # with extension (.yuv or .bin)
CSET NUMBER_OF_FRAMES = 1;                         # number of frames
CSET NUMBER_OF_COLS = 64;                          # number of columns
CSET NUMBER_OF_ROWS = 64;                          # number of rows

# Filter Coefficients
# supported range of [-2 to 2) -
# quantized to 16-bit values with 14 fractional bits
# coefficient values not defined here will default to 0
# extra coefficients defined here will not be used
# (for example, coef05_hphase0 will not be used if num_h_taps=3)
CSET COEF00_HPHASE0 = 0.25;
CSET COEF01_HPHASE0 = 0.5;
CSET COEF02_HPHASE0 = 0.25;
```

All the variables are set with a line beginning with the keyword “CSET”.

For the generic variables, there is a one-to-one mapping between the generic variables in the configuration file and the generic variables in Table D-4, page 57. Any generic variables that are not set will use the default value.

The example code will create the input video_in by reading in a YUV or BIN file. The configuration file must specify the input image file name, the number of frames, the number of columns, and the number of rows. The input image chroma format must match the generic variable CHROMA_FORMAT_IN. The example code only processes 8-bit YUV and BIN input files.

Filter Coefficients can be defined in the configuration file. The coefficients have an allowed range of [-2,2). The coefficients will be quantized to 16-bit values with 14 fractional bits. Any undefined coefficients will default to 0. Any unnecessary extra coefficients that are defined will not be used. For example, COEF05_HPHASE0 will be unused when NUM_H_TAPS=3.

Compiling the Chroma Resampler C Model with Example Wrapper

Linux (32 and 64-bit)

To compile the example code, perform the following steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip-file:

   ```
   setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
   ```

2. Copy the following files from the /lin32 or /lin64 directory to the root directory:

   ```
   libstlport.so.5.1
   libIp_v_cresample_v1_0_bitacc_cmodel.so
   ```

3. Then in the root directory, compile using the GNU C Compiler using the following command:

   ```
   gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_cresample_v1_0_bitacc_cmodel -Wl,-rpath,.

   gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_cresample_v1_0_bitacc_cmodel -Wl,-rpath,.
   ```

Windows (32 and 64-bit)

Precompiled library `v_cresample_v1_0_bitacc_cmodel.lib` and top level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. This section presents an example using Microsoft Visual Studio.

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

- `libIp_v_cresample_v1_0_bitacc_cmodel.lib` to the Resource Files folder of the project
- `run_bitacc_cmodel.c` and `parsers.c` to the Source Files folder of the project
- `v_cresample_v1_0_bitacc_cmodel.h` to Header Files folder of the project

Once the project has been created and populated, it needs to be compiled and built in order to create an executable. To perform the build step, choose Build Solution from the Build menu. An executable matching the project name has been 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.

In order to ease modifying and debugging the top-level demonstrator using the built-in debugging environment of Visual Studio, the top-level command-line parameters can be specified through the Project Property pages. In the Solution Explorer pane, right click the project name, and select Properties from the context menu. Select Debugging on the left pane of the Property Pages dialog box. Enter the paths and filenames to the input and output images into the Command Arguments field.

Appendix E

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 user guide:

- *Xilinx AXI Reference Guide*
- *AMBA AXI4 Interface Protocol*

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 Chroma Resampler core is provided under 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 Chroma Resampler 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.