LogiCORE IP Chroma Resampler v2.00.a

Product Guide

PG012 April 24, 2012

XILINX®

XILINX®

Table of Contents

Chapter 5: Constraining the Core

Chapter 6: Detailed Example Design

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating

Appendix C: Debugging

Appendix D: Application Software Development

Appendix E: C Model Reference

Appendix F: Additional Resources

Introduction

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

Features

- 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
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, and 12-bits per color component input and output
- Built-in, optional bypass and test-pattern generator mode
- Built-in, optional throughput monitors
- Supports spatial resolutions from 32x32 up to 7680x7680
  - Supports 1080P60 in all supported device families
  - Supports 4kx2k @ 24 Hz in supported high performance devices

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

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C-Model available on the product page on Xilinx.com at www.xilinx.com/products/intellectual-property/EF-DI-CHROM-RESAMP.htm.
3. For the supported versions of the tools, see the ISE Design Suite 14: 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.

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.

The core can be configured and instantiated using CORE Generator or EDK tools. Core functionality can be controlled dynamically with an optional AXI4-Lite interface.

Applications

- Pre-processing block for image sensors

- Video surveillance
- Industrial imaging
- Video conferencing
- Machine vision
- Other imaging applications

Licensing

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The core may be generated using either the Xilinx ISE CORE Generator tool or Embedded Edition software (EDK). For full access to all core functionality in simulation and in hardware, you must purchase a license for the core. Please contact your local Xilinx sales representative for information on pricing and availability of Xilinx LogiCORE IP.

For more information, please visit the LogiCORE IP Chroma Resampler product page.

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

Product Specification

The Chroma Resampler core converts between chroma sub-sampling formats of 4:4:4, 4:2:2, and 4:2:0. This chapter details the standards, performance, resource utilization and interfaces.

Standards Compliance

The Chroma Resampler core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to "Video IP: AXI Feature Adoption" in UG761, *Xilinx AXI Reference Guide*, for additional information.

Performance

This section details the performance characteristics of the Chroma Resampler core.

Maximum Frequencies

This section contains typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. Refer to Table 2-1 through Table 2-6 for device-specific information.

Throughput

The Chroma Resampler core produces one output pixel per input sample.

The core supports bidirectional data throttling between its AXI4-Stream Slave and Master interfaces. If the slave side data source is not providing valid data samples (`s_axis_video_tvalid` is not asserted), the core cannot produce valid output samples after its internal buffers are depleted. Similarly, if the master side interface is not ready to accept valid data samples (`m_axis_video_tready` is not asserted) the core cannot accept valid input samples once its buffers become full.

If the master interface is able to provide valid samples (s_axis_video_tvalid is high) and the slave interface is ready to accept valid samples (m_axis_video_tready is high), typically the core can process one sample and produce one pixel per ACLK cycle.

However, at the end of each scan line the core flushes internal pipelines for a number of clock cycles equal to the latency of the core, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples. Also at the end of each frame the core flushes internal line buffers for the number of lines of latency, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples.

When the core is processing timed streaming video (which is typical for image sensors), the flushing periods coincide with the blanking periods therefore do not reduce the throughput of the system.

When the core is processing data from a video source which can always provide valid data, e.g. a frame buffer, the throughput of the core can be defined as follows (assuming a worst case latency of 18 clock cycles and 7 scan lines):

$$R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS + 7} \times \frac{COLS}{COLS + 18} \quad \textit{Equation 2-1}$$

In numeric terms, 1080P/60 represents an average data rate of 124.4 MPixels/second (1080 rows x 1920 columns x 60 frames / second), and a burst data rate of 148.5 MPixels/sec.

To ensure that the core can process 124.4 MPixels/second, it needs to operate minimally at:

$$f_{ACLK} = R_{MAX} \times \frac{ROWS + 7}{ROWS} \times \frac{COLS + 58}{COLS} = 124.4 \times \frac{1087}{1080} \times \frac{1938}{1920} = 126.4 \quad \textit{Equation 2-2}$$

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 EOL Signal pulses.

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.

Resource Utilization

For an accurate measure of the usage of primitives, slices, and CLBs for a particular instance, check the **Display Core Viewer after Generation** check box in the CORE Generator interface.

The information presented in Table 2-1 through Table 2-6 is a guide to the resource utilization and maximum clock frequency of the Chroma Resampler core for Zynq-7000, Artix-7, Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. This core does not use any dedicated I/O or CLK resources. The design was tested using ISE® Design Suite v14.1 with the default tool options for characterization data.

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

- 1920 x 1080 frame size
- Default filter size
- Progressive video
- 8-bit data
- Odd Chroma parity
- AXI4-Lite interface included

Table 2-1: **Zynq-7000 resource Usage**

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/ 8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2274	2147	1601	0/0	0	246
	Fixed Coefficient	2274	2147	1601	0/0	0	246
	User Defined	2259	2163	1618	0/0	3	230
4:4:4 to 4:2:0	Drop/Replicate	2233	2130	1529	0/1	0	230
	Fixed Coefficient	2513	2358	1808	0/2	0	246
	User Defined	2283	2138	1608	0/2	5	239
4:2:2 to 4:4:4	Drop/Replicate	2117	2040	1479	0/0	0	246
	Fixed Coefficient	2251	2155	1593	0/0	0	230
	User Defined	2266	2168	1614	0/0	2	255

Table 2-1: **Zynq-7000 resource Usage *(Cont'd)***

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:2:2 to 4:2:0	Drop/Replicate	2181	2093	1465	0/1	0	239
	Fixed Coefficient	2268	2139	1592	0/2	0	255
	User Defined	2283	2138	1608	0/2	2	239
4:2:0 to 4:4:4	Drop/Replicate	2368	2256	1666	0/2	0	239
	Fixed Coefficient	2798	2615	1971	0/4	0	239
	User Defined	2873	2723	2027	0/4	4	263
4:2:0 to 4:2:2	Drop/Replicate	2200	2101	1528	0/2	0	222
	Fixed Coefficient	2393	2227	1682	0/4	0	239
	User Defined	2430	2291	1717	0/4	2	239

Device: XC7Z030-1FFG676C (ADVANCED 1.01d 2012-04-02)

Table 2-2: **Artix-7 Resource Usage**

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2183	2066	1597	0/0	0	173
	Fixed Coefficient	2183	2066	1597	0/0	0	173
	User Defined	2222	2104	1618	0/0	3	173
4:4:4 to 4:2:0	Drop/Replicate	2169	2059	1529	0/1	0	173
	Fixed Coefficient	2441	2292	1808	0/2	0	164
	User Defined	2530	2339	1849	0/2	5	173
4:2:2 to 4:4:4	Drop/Replicate	2050	1944	1475	0/0	0	164
	Fixed Coefficient	2160	2066	1593	0/0	0	173
	User Defined	2211	2085	1614	0/0	2	173
4:2:2 to 4:2:0	Drop/Replicate	2134	2021	1465	0/1	0	173
	Fixed Coefficient	2187	2077	1595	0/2	0	164
	User Defined	2238	2098	1613	0/2	2	164
4:2:0 to 4:4:4	Drop/Replicate	2292	2150	1668	0/2	0	164
	Fixed Coefficient	2734	2513	1972	0/4	0	173
	User Defined	2803	2606	2032	0/4	4	164
4:2:0 to 4:2:2	Drop/Replicate	2175	2026	1530	0/2	0	164
	Fixed Coefficient	2309	2164	1684	0/4	0	173
	User Defined	2364	2219	1719	0/4	2	173

Device: XC7A100T-1FGG484C (ADVANCED 1.03j 2012-04-02)

Table 2-3: **Virtex-7 Resource Usage**

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2068	1995	1446	0/0	0	232
	Fixed Coefficient	2272	2149	1601	0/0	0	273
	User Defined	2284	2161	1618	0/0	3	253
4:4:4 to 4:2:0	Drop/Replicate	2244	2121	1529	0/1	0	253
	Fixed Coefficient	2506	2361	1808	0/2	0	242
	User Defined	2564	2443	1848	0/2	5	253
4:2:2 to 4:4:4	Drop/Replicate	2119	2026	1479	0/0	0	253
	Fixed Coefficient	2260	2164	1593	0/0	0	263
	User Defined	2273	2178	1615	0/0	2	253
4:2:2 to 4:2:0	Drop/Replicate	2182	2100	1465	0/1	0	232
	Fixed Coefficient	2272	2163	1592	0/2	0	222
	User Defined	2288	2167	1609	0/2	2	212
4:2:0 to 4:4:4	Drop/Replicate	2382	2249	1666	0/2	0	263
	Fixed Coefficient	2766	2623	1971	0/4	0	253
	User Defined	2889	2707	2028	0/4	4	247
4:2:0 to 4:2:2	Drop/Replicate	2208	2114	1531	0/2	0	263
	Fixed Coefficient	2368	2247	1685	0/4	0	253
	User Defined	2457	2318	1718	0/4	2	202

Device: XC7V585T-1FFG1157C (ADVANCED 1.04j 2012-04-02)

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

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2239	2157	1601	0/0	0	255
	Fixed Coefficient	2239	2157	1601	0/0	0	255
	User Defined	2268	2167	1618	0/0	3	255
4:4:4 to 4:2:0	Drop/Replicate	2226	2129	1529	0/1	0	263
	Fixed Coefficient	2518	2346	1808	0/2	0	255
	User Defined	2586	2435	1848	0/2	5	239
4:2:2 to 4:4:4	Drop/Replicate	2105	2029	1479	0/0	0	230
	Fixed Coefficient	2239	2162	1593	0/0	0	230
	User Defined	2262	2181	1615	0/0	2	246

Table 2-4: **Kintex-7 Resource Usage *(Cont'd)***

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:2:2 to 4:2:0	Drop/Replicate	2171	2100	1465	0/1	0	239
	Fixed Coefficient	2230	2155	1592	0/2	0	255
	User Defined	2288	2160	1609	0/2	2	246
4:2:0 to 4:4:4	Drop/Replicate	2388	2244	1666	0/2	0	239
	Fixed Coefficient	2779	2621	1971	0/4	0	255
	User Defined	2838	2712	2028	0/4	4	239
4:2:0 to 4:2:2	Drop/Replicate	2203	2120	1531	0/2	0	255
	Fixed Coefficient	2365	2248	1685	0/4	0	263
	User Defined	2437	2326	1718	0/4	2	239

Device: XC7K70T-1FBG484C (ADVANCED 1.04c 2012-04-02)

Table 2-5: **Virtex-6 Resource Usage**

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2165	2056	1601	0/0	0	246
	Fixed Coefficient	2165	2056	1601	0/0	0	246
	User Defined	2175	2069	1618	0/0	3	239
4:4:4 to 4:2:0	Drop/Replicate	2084	2001	1530	0/1	0	246
	Fixed Coefficient	2381	2215	1808	0/2	0	255
	User Defined	2458	2285	1847	0/2	5	239
4:2:2 to 4:4:4	Drop/Replicate	2041	1926	1479	0/0	0	231
	Fixed Coefficient	2183	2087	1595	0/0	0	246
	User Defined	2196	2073	1614	0/0	2	231
4:2:2 to 4:2:0	Drop/Replicate	2040	1953	1466	0/1	0	239
	Fixed Coefficient	2171	2081	1592	0/2	0	262
	User Defined	2163	2074	1608	0/2	2	239
4:2:0 to 4:4:4	Drop/Replicate	2292	2161	1667	0/2	0	255
	Fixed Coefficient	2693	2542	1971	0/4	0	255
	User Defined	2738	2627	2034	0/4	4	246
4:2:0 to 4:2:2	Drop/Replicate	2122	2035	1529	0/2	0	239
	Fixed Coefficient	2262	2145	1682	0/4	0	255
	User Defined	2327	2205	1717	0/4	2	255

Device: XC6VLX75T-1FF484C (PRODUCTION 1.17 2012-04-02)

Table 2-6: **Spartan-6 Resource Usage**

Conversion	Filter Type	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB16BWERs/ 8BWERs	DSP48A1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	2190	2056	1607	0/0	0	164
	Fixed Coefficient	2190	2056	1607	0/0	0	164
	User Defined	2226	2085	1627	0/0	3	164
4:4:4 to 4:2:0	Drop/Replicate	2184	2043	1532	1/0	0	154
	Fixed Coefficient	2465	2290	1813	2/0	0	154
	User Defined	2544	2372	1856	2/0	5	154
4:2:2 to 4:4:4	Drop/Replicate	2055	1949	1487	0/0	0	164
	Fixed Coefficient	2211	2091	1603	0/0	0	154
	User Defined	2210	2086	1626	0/0	2	154
4:2:2 to 4:2:0	Drop/Replicate	2166	2052	1470	1/0	0	154
	Fixed Coefficient	2223	2073	1600	2/0	0	164
	User Defined	2217	2068	1615	2/0	2	148
4:2:0 to 4:4:4	Drop/Replicate	2314	2153	1673	2/0	0	164
	Fixed Coefficient	2746	2531	1982	4/0	0	164
	User Defined	2790	2624	2040	4/0	4	164
4:2:0 to 4:2:2	Drop/Replicate	2158	2037	1539	2/0	0	169
	Fixed Coefficient	2348	2195	1694	4/0	0	154
	User Defined	2380	2231	1727	4/0	2	154

Device: XC6SLX25-2FGG484C (PRODUCTION 1.21 2012-04-02)

Port Descriptions

The Chroma Resampler core uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the Chroma Resampler. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the `IRQ` pin are present only when the core is configured via the GUI with an AXI4-Lite control interface. The `INTC_IF` interface is present only when the core is configured via the GUI with the INTC interface enabled.

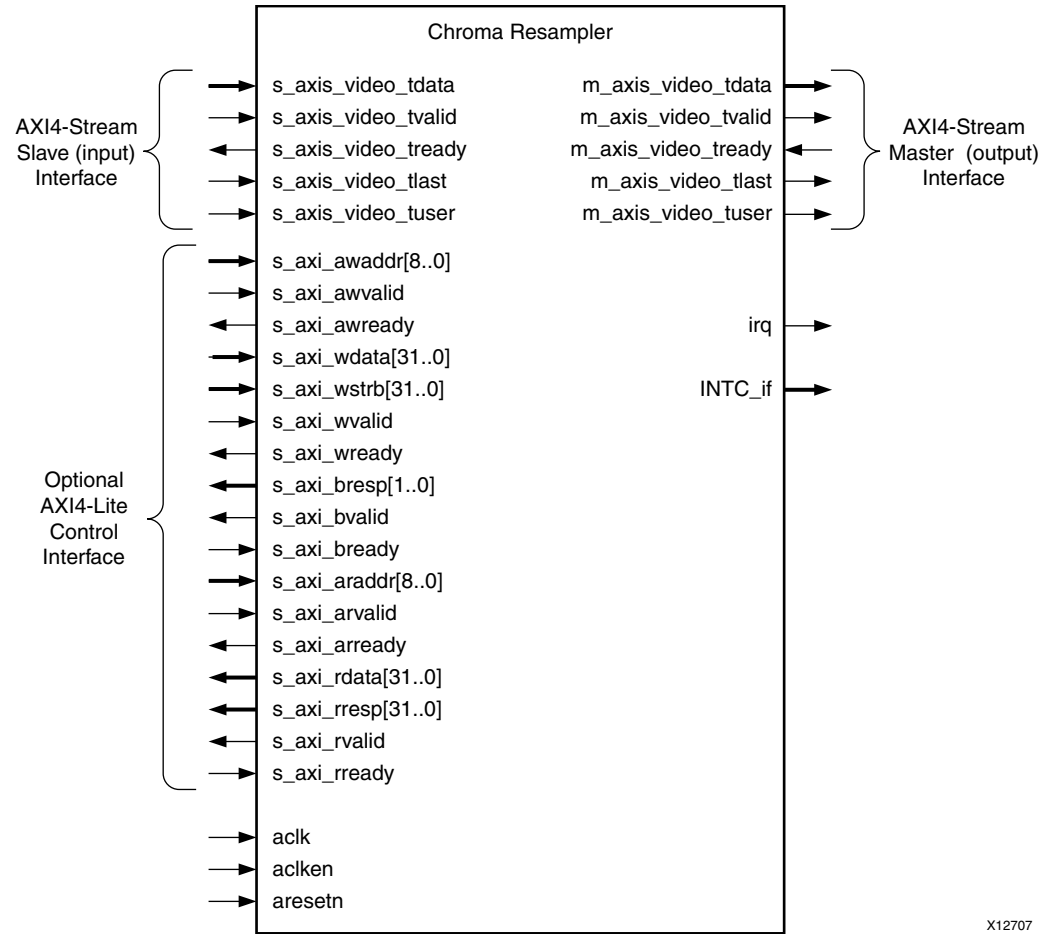

Figure 2-1: **Chroma Resampler Top-Level Signaling Interface**

Common Interface Signals

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

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

Signal Name	Direction	Width	Description
ACLK	In	1	Clock
ACLKEN	In	1	Clock Enable
ARESETn	In	1	Active low synchronous
INTC_IF	Out	9	Optional External Interrupt Controller Interface. Available only when INTC_IF is selected on GUI.
IRQ	Out	1	Optional Interrupt Request Pin. Available only when AXI4-Liter interface is selected on GUI.

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

ACLK

All signals, including the AXI4-Stream and AXI4-Lite component interfaces, must be synchronous to the core clock signal ACLK. All interface input signals are sampled on the rising edge of ACLK. All output signal changes occur after the rising edge of ACLK.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to both the AXI4-Stream and AXI4-Lite interfaces. Setting ACLKEN low (de-asserted) halts the operation of the core despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is de-asserted, core inputs are not sampled, except ARESETn, which supersedes ACLKEN.

ARESETn

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

Data Interface

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

AXI4-Stream Signal Names and Descriptions

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

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

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	16, 24, 32, 40	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal
s_axis_video_tready	Out	1	Input Ready
s_axis_video_tuser	In	1	Input Video Start Of Frame

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

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

Video Data

The AXI4-Stream interface specification restricts `TDATA` widths to integer multiples of 8 bits. Therefore, 10 and 12 bit sensor data must be padded with zeros on the MSB to form a 24-, 32-, or 40-bit wide vector before connecting to `s_axis_video_tdata`. Padding does not affect the size of the core.

Similarly, YCbCr data on the Chroma Resampler output `m_axis_video_tdata` is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2 and Figure 2-3. Zero padding the most significant bits is only necessary for 10- and 12-bit wide data.

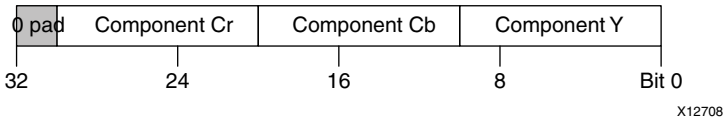

Figure 2-2: **YCbCr Data Encoding for 4:4:4 on m_axis_video_tdata**

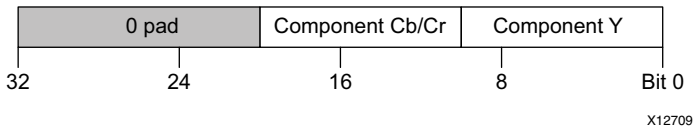

Figure 2-3: **YCbCr Data Encoding for 4:2:2 or 4:2:0 on m_axis_video_tdata**

YCbCr data is packed on the `video_data` bus as shown in Figure 2-4, Figure 2-5, and Figure 2-6. 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 2-4.

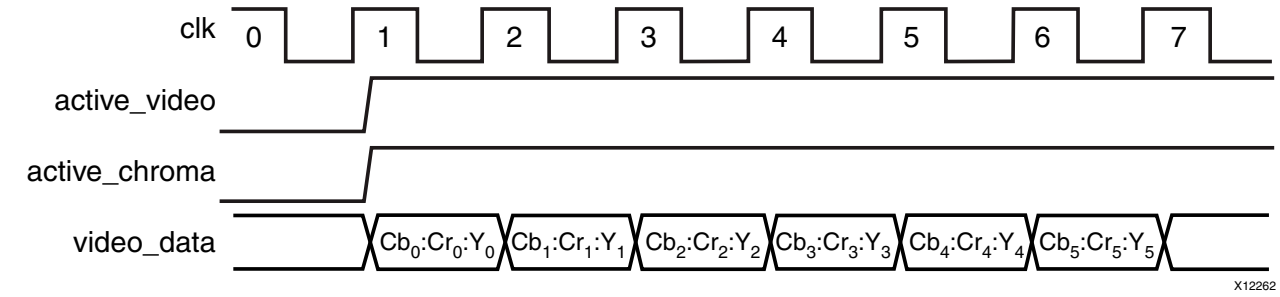

Figure 2-4: **YCbCr 4:4:4**

For 4:2:2, Cb and Cr are interleaved on the `video_data` bus. The first active video data sample contains Cb first, as shown in Figure 2-5.

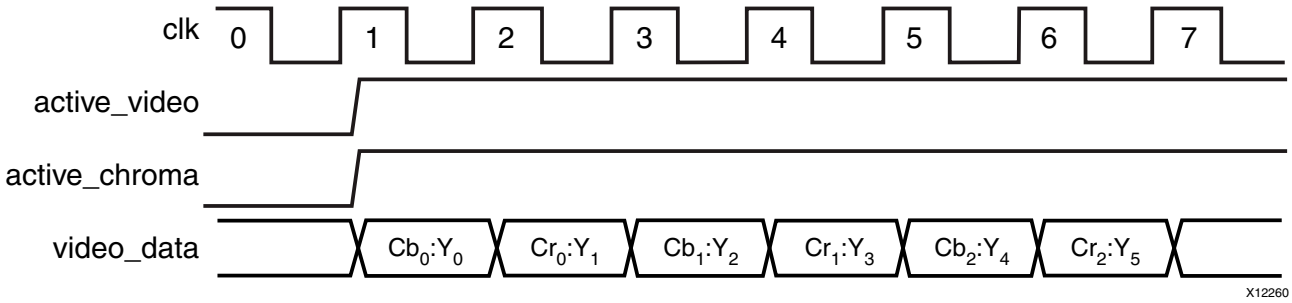

Figure 2-5: **YCbCr 4:2:2**

For 4:2:0, the format is similar to 4:2:2, except only the alternate lines have valid chroma, as shown in Figure 2-6. The `chroma_parity` register signals whether the first line has chroma information. Cb and Cr samples are interleaved as per 4:2:2.

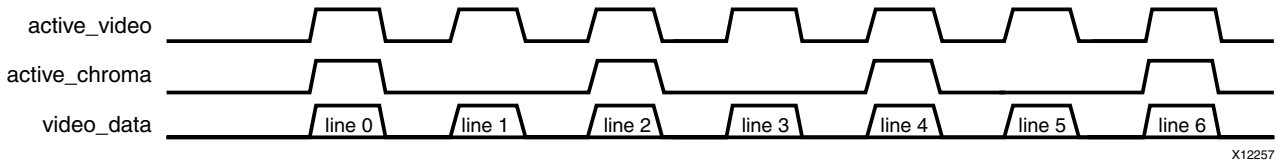

Figure 2-6: **YCbCr 4:2:0**

READY/VALID Handshake

A valid transfer occurs whenever `READY`, `VALID`, `ACLKEN`, and `ARESETn` are high at the rising edge of `ACLK`, as seen in Figure 2-9. During valid transfers, `DATA` only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

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

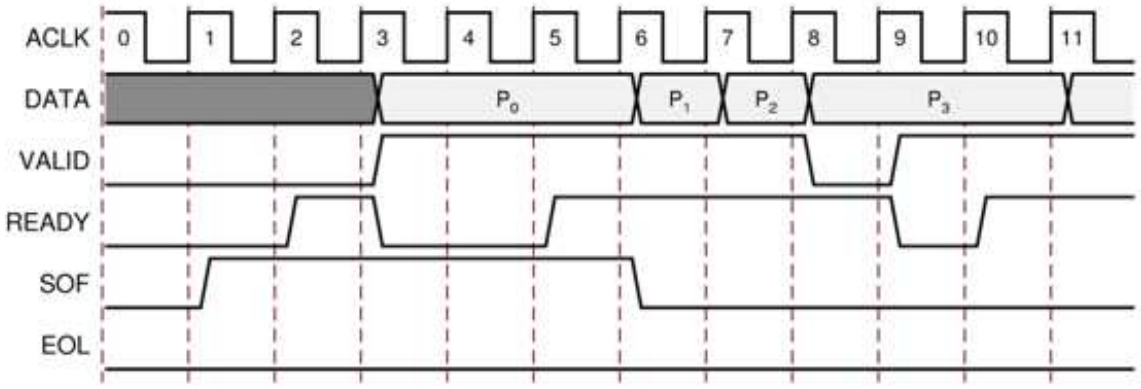

Figure 2-7: **Example of READY/VALID Handshake, Start of a New Frame**

Guidelines on Driving m_axis_video_tready

The m_axis_video_tready signal may be asserted before, during or after the cycle in which the Chroma Resampler asserted m_axis_video_tvalid. The assertion of m_axis_video_tready may be dependent on the value of m_axis_video_tvalid. A slave that can immediately accept data qualified by m_axis_video_tvalid, should pre-assert its m_axis_video_tready signal until data is received. Alternatively, m_axis_video_tready can be registered and driven the cycle following VALID assertion. It is recommended that the AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

Start of Frame Signals - m_axis_video_tuser, s_axis_video_tuser

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream TUSER0 signal, marks the first pixel of a video frame. The SOF pulse is 1 valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-7. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The SOF signal may be asserted an arbitrary number of ACLK cycles before the first pixel value is presented on DATA, as long as a VALID is not asserted.

End of Line Signals - m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is 1 valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-8.

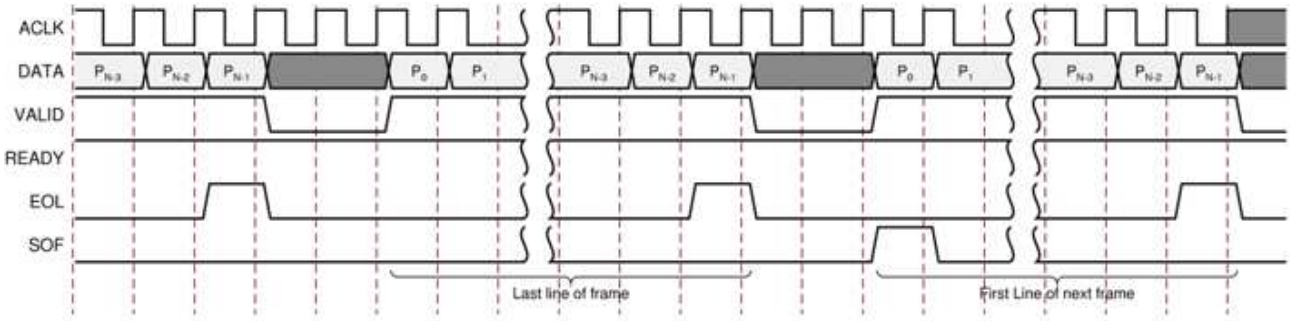

Figure 2-8: **Use of EOL and SOF Signals**

Control Interface

When configuring the core, the user has the option to add an AXI4-Lite register interface to dynamically control the behavior of the core. The AXI4-Lite slave interface facilitates integrating the core into a processor system, or along with other video or AXI4-Lite compliant IP, connected via AXI4-Lite interface to an AXI4-Lite master. In a static configuration with a fixed set of parameters (constant configuration), the core can be instantiated without the AXI4-Lite control interface, which reduces the core Slice footprint.

Constant Configuration

The constant configuration caters to users who will interface the core to a particular image sensor with a known, stationary resolution, field parity, and chroma parity. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame), and the field parity and chroma parity are hard coded into the core via the Chroma Resampler GUI. Since there is no AXI4-Lite interface, the core is not programmable, but can be reset, enabled, or disabled using the `ARESETn` and `ACLKEN` ports.

AXI4-Lite Interface

The AXI4-Lite interface allows a user to dynamically control parameters within the core. Core configuration can be accomplished using an AXI4-Stream master state machine, or an embedded ARM or soft system processor such as a MicroBlaze processor.

The Chroma Resampler can be controlled via the AXI4-Lite interface using read and write transactions to the Chroma Resampler register space.

Table 2-9: **AXI4-Lite Interface Signals**

Signal Name	Direction	Width	Description
s_axi_lite_awvalid	In	1	AXI4-Lite Write Address Channel Write Address Valid.
s_axi_lite_awread	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
s_axi_lite_awaddr	In	32	AXI4-Lite Write Address Bus
s_axi_lite_wvalid	In	1	AXI4-Lite Write Data Channel Write Data Valid.
s_axi_lite_wready	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
s_axi_lite_wdata	In	32	AXI4-Lite Write Data Bus
s_axi_lite_bresp	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
s_axi_lite_bvalid	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
s_axi_lite_bready	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
s_axi_lite_arvalid	In	1	AXI4-Lite Read Address Channel Read Address Valid
s_axi_lite_arready	Out	1	Ready. Indicates DMA is ready to accept the read address.
s_axi_lite_araddr	In	32	AXI4-Lite Read Address Bus
s_axi_lite_rvalid	Out	1	AXI4-Lite Read Data Channel Read Data Valid
s_axi_lite_rready	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
s_axi_lite_rdata	Out	32	AXI4-Lite Read Data Bus
s_axi_lite_rresp	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.

Register Space

The standardized Xilinx Video IP register space is partitioned into control-, timing-, and core specific registers. The Chroma Resampler uses two timing related registers:

- `ACTIVE_SIZE` (0x0020) allows specifying the input frame dimensions.
- `ENCODING` (0x0028) allows specifying the field parity and chroma parity.

The core has a set of core-specific registers that allows the resampling filter coefficient values to be specified.

Table 2-10: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	No	Power-on-Reset: 0x0	Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 4: BYPASS Bit 5: TEST_PATTERN(1) Bit 30: FRAME_SYNC_RESET (1: reset) Bit 31: SW_RESET (1: reset)
0x0004	STATUS	R/W	No	0	Bit 0: PROC_STARTED Bit 1: EOF Bit 16: SLAVE_ERROR
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to `STATUS` bits
0x0010	VERSION	R	N/A	0x0500a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	SYSDEBUG0	R	N/A	0	0-31: Frame Throughput monitor(1)
0x0018	SYSDEBUG1	R	N/A	0	0-31: Line Throughput monitor(1)
0x001C	SYSDEBUG2	R	N/A	0	0-31: Pixel Throughput monitor(1)
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame
0x0028	ENCODING	R/W	Yes	Specified via GUI	7: Field Parity 8: Chroma Parity All other bits reserved

Table 2-10: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0100	COEF00_HPHASE0	R/W	Yes	Pre-defined Fixed Coefficient Filter Values	Coefficients for Horizontal Filter Phase 0
0x0104	COEF01_HPHASE0				
0x0108	COEF02_HPHASE0				
0x010C	COEF03_HPHASE0				
0x0110	COEF04_HPHASE0				
0x0114	COEF05_HPHASE0				
0x0118	COEF06_HPHASE0				
0x011C	COEF07_HPHASE0				
0x0120	COEF08_HPHASE0				
0x0124	COEF09_HPHASE0				
0x0128	COEF10_HPHASE0				
0x012C	COEF11_HPHASE0				
0x0130	COEF12_HPHASE0				
0x0134	COEF13_HPHASE0				
0x0138	COEF14_HPHASE0				
0x013C	COEF15_HPHASE0				
0x0140	COEF16_HPHASE0				
0x0144	COEF17_HPHASE0				
0x0148	COEF18_HPHASE0				
0x014C	COEF19_HPHASE0				
0x0150	COEF20_HPHASE0				
0x0154	COEF21_HPHASE0				
0x0158	COEF22_HPHASE0				
0x015C	COEF23_HPHASE0				

Table 2-10: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0160 0x0164 0x0168 0x016C 0x0170 0x0174 0x0178 0x017C 0x0180 0x0184 0x0188 0x018C 0x0190 0x0194 0x0198 0x019C 0x01A0 0x01A4 0x01A8 0x01AC 0x01B0 0x01B4 0x01B8 0x01BC	COEF00_HPHASE1 COEF01_HPHASE1 COEF02_HPHASE1 COEF03_HPHASE1 COEF04_HPHASE1 COEF05_HPHASE1 COEF06_HPHASE1 COEF07_HPHASE1 COEF08_HPHASE1 COEF09_HPHASE1 COEF10_HPHASE1 COEF11_HPHASE1 COEF12_HPHASE1 COEF13_HPHASE1 COEF14_HPHASE1 COEF15_HPHASE1 COEF16_HPHASE1 COEF17_HPHASE1 COEF18_HPHASE1 COEF19_HPHASE1 COEF20_HPHASE1 COEF21_HPHASE1 COEF22_HPHASE1 COEF23_HPHASE1	R/W	Yes	Pre-defined Fixed Coefficient Filter Values	Coefficients for Horizontal Filter Phase 1
0x01C0 0x01C4 0x01C8 0x01CC 0x01D0 0x01D4 0x01D8 0x01DC	COEF00_VPHASE0 COEF01_VPHASE0 COEF02_VPHASE0 COEF03_VPHASE0 COEF04_VPHASE0 COEF05_VPHASE0 COEF06_VPHASE0 COEF07_VPHASE0	R/W	Yes	Pre-defined Fixed Coefficient Filter Values	Coefficients for Vertical Filter Phase 0
0x01E0 0x01E4 0x01E8 0x01EC 0x01F0 0x01F4 0x01F8 0x01FC	COEF00_VPHASE1 COEF01_VPHASE1 COEF02_VPHASE1 COEF03_VPHASE1 COEF04_VPHASE1 COEF05_VPHASE1 COEF06_VPHASE1 COEF07_VPHASE1	R/W	Yes	Pre-defined Fixed Coefficient Filter Values	Coefficients for Vertical Filter Phase 0

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

CONTROL (0x0000) Register

Bit 0 of the CONTROL register, SW_ENABLE, facilitates enabling and disabling the core from software. Writing '0' to this bit effectively disables the core halting further operations, which blocks the propagation of all video signals. After Power up, or Global Reset, the SW_ENABLE defaults to 0 for the AXI4-Lite interface. Similar to the ACLKEN pin, the SW_ENABLE flag is not synchronized with the AXI4-Stream interfaces: Enabling or Disabling the core takes effect immediately, irrespective of the core processing status. Disabling the core for extended periods may lead to image tearing.

Bit 1 of the CONTROL register, REG_UPDATE is a write done semaphore for the host processor, which facilitates committing all user and timing register updates simultaneously. The Chroma Resampler ACTIVE_SIZE, ENCODING, and coefficient registers are double buffered. One set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers will get copied over to the active set at the end of the AXI4-Stream frame, if and only if REG_UPDATE is set. Setting REG_UPDATE to 0 before updating multiple register values, then setting REG_UPDATE to 1 when updates are completed ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

Bit 4 of the CONTROL register, BYPASS, switches the core to bypass mode if debug features are enabled. In bypass mode, the core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. Refer to Appendix C, Debugging for more information. If debug features were not included at instantiation, this flag has no effect on the operation of the core. Switching bypass mode on or off is not synchronized to frame processing, and therefore can lead to image tearing.

Bit 5 of the CONTROL register, TEST_PATTERN, switches the core to test-pattern generator mode if debug features are enabled. Refer to Appendix C, Debugging for more information. If debug features were not included at instantiation, this flag has no effect on the operation of the core. Switching test-pattern generator mode on or off is not synchronized to frame processing, therefore can lead to image tearing.

Bits 30 and 31 of the CONTROL register, FRAME_SYNC_RESET and SW_RESET facilitate software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress will cause image tearing. For applications where the soft-ware reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 will reset the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

STATUS (0x0004) Register

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

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

Bit 0 of the STATUS register, PROC_STARTED, indicates that processing of a frame has commenced via the AXI4-Stream interface.

Bit 1 of the STATUS register, End-of-frame (EOF), indicates that the processing of a frame has completed.

Bit 16 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred.

ERROR (0x0008) Register

Bit 16 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred. This bit can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS and ERROR registers remain set after an event associated with the particular ERROR register bit, even if the event condition is not present at the time the interrupt is serviced.

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

Bit 0 of the ERROR register, EOL_EARLY, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 1 of the ERROR register, EOL_LATE, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the last EOL signal surpassed the value programmed into the ACTIVE_SIZE register.

Bit 2 of the ERROR register, SOF_EARLY, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the latest and the preceding Start-Of-Frame (SOF) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 3 of the ERROR register, SOF_LATE, indicates an error during processing a video frame via the AXI4-Stream slave port. The number of pixels received between the last SOF signal surpassed the value programmed into the ACTIVE_SIZE register.

IRQ_ENABLE (0x000C) Register

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

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware. See Table 2-10 for details.

SYSDEBUG0 (0x0014) Register

The SYSDEBUG0, or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Appendix C, Debugging for more information.

SYSDEBUG1 (0x0018) Register

The SYSDEBUG1, or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Appendix C, Debugging for more information.

SYSDEBUG2 (0x001C) Register

The SYSDEBUG2, or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Appendix C, Debugging for more information.

ACTIVE_SIZE (0x0020) Register

The ACTIVE_SIZE register encodes the number of active pixels per scan line and the number of active scan lines per frame. The lower half-word (bits 12:0) encodes the number of active pixels per scan line. Supported values are between 32 and the value provided in the **Maximum number of pixels per scan line** field in the GUI. The upper half-word (bits 28:16) encodes the number of active lines per frame. Supported values are 32 to 7680. To

avoid processing errors, the user should restrict values written to ACTIVE_SIZE to the range supported by the core instance.

ENCODING (0x0028) Register

Bit 7 (FIELD_PARITY) indicates field parity (0: even/bottom field, 1: odd/top field) if interlaced video is used. The host processor is not expected to update this register value on a frame-by-frame basis. Instead, the core will toggle automatically after processing fields, using the programmed value as the initial value for the first field after the value was committed.

Bit 8 (CHROMA_PARITY) of the ENCODING register specifies whether the first line of video contains Chroma information (1) or not (0) if YCbCr 4:2:0 encoded video being processed.

The Interrupt Subsystem

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

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (IRQ) is present. Events associated with bits of the STATUS register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (IRQ_ENABLE) are set. Once set by the corresponding event, bits of the STATUS register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface the user can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional INTC_IF port. This vector of signals gives parallel access to the individual interrupt sources, as seen in Table 2-11.

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

Table 2-11: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	Pixel counter terminal count
3	Line counter terminal count

Table 2-11: **INTC_IF Signal Functions**

INTC_IF signal	Function
4	Slave error
5	EOL Early
6	EOL Late
7	SOF Early
8	SOF Late

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

Customizing and Generating the Core

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

CORE Generator GUI

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

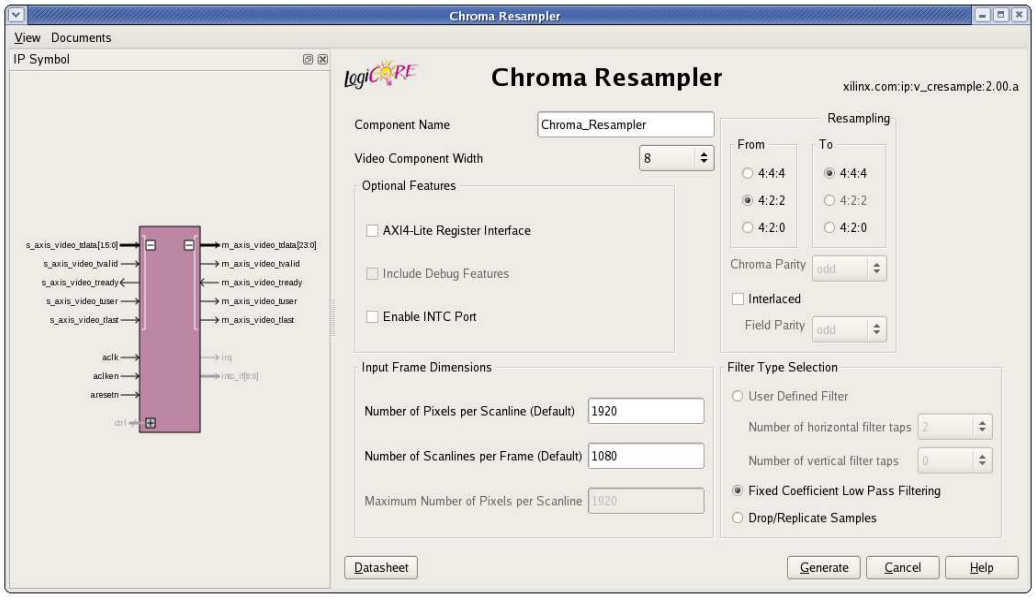

Figure 3-1: **Chroma Resampler CORE Generator GUI**

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

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name v_cresample_v2_00_a cannot be used as a component name.

- **Video Component Width**: Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.
- **Optional Features**:
  - **AXI4-Lite Register Interface**: When selected, the core will be generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, refer to Control Interface in Chapter 3.
  - **Include Debug Features**: When selected, the core will be generated with debugging features, which simplify system design, testing and debugging. For more information, refer to Appendix C, Debugging.

    Note: Debugging features are only available when the AXI4-Lite Register Interface is selected.
  - **INTC Interface**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, refer to The Interrupt Subsystem in Chapter 3.
- **Input Frame Dimensions**:
  - **Number of Active Pixels per Scan line**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the lower half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance processes.
  - **Number of Active Lines per Frame**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance processes.
  - **Maximum Number of Active Pixels Per Scan Line**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for Number of Active Pixels per Scan line, or the corresponding lower half-word of the ACTIVE_SIZE register must always be less than or equal to the value provided by Maximum Number of Active Pixels Per Scan line. Using a tight upper-bound results in optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field reflect the actual contents of the **Number of Active Pixels per Scan Line** field. In constant mode, the maximum number of pixels equals the active number of pixels.
- **Resampling**: Select the input and output chroma formats. The supported formats are 4:4:4, 4:2:2, and 4:2:0.
- **Chroma Parity**: For 4:2:0, select odd if the first line of video contains chroma information. Chroma parity is only used for 4:2:0 data.

- **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.
- **Field Parity**: For interlaced video, select odd if the odd (or top) field comes first. Select even if the even (or bottom) field comes first.
- **Filter Type Selection**:
  - **User Defined Filter**: Users can program the filter coefficients through the AXI4-Lite 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 24. 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, 13, 15, 17, 19, 21, 23
      - 4:2:2 to 4:4:4: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
      - 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, 13, 15, 17, 19, 21, 23
      - 4:2:0 to 4:4:4: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
    - **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 4.

- **Drop/Replicate Samples**: 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

Definitions of the EDK GUI controls are identical to the corresponding CORE Generator GUI functions described in CORE Generator GUI, page 31.

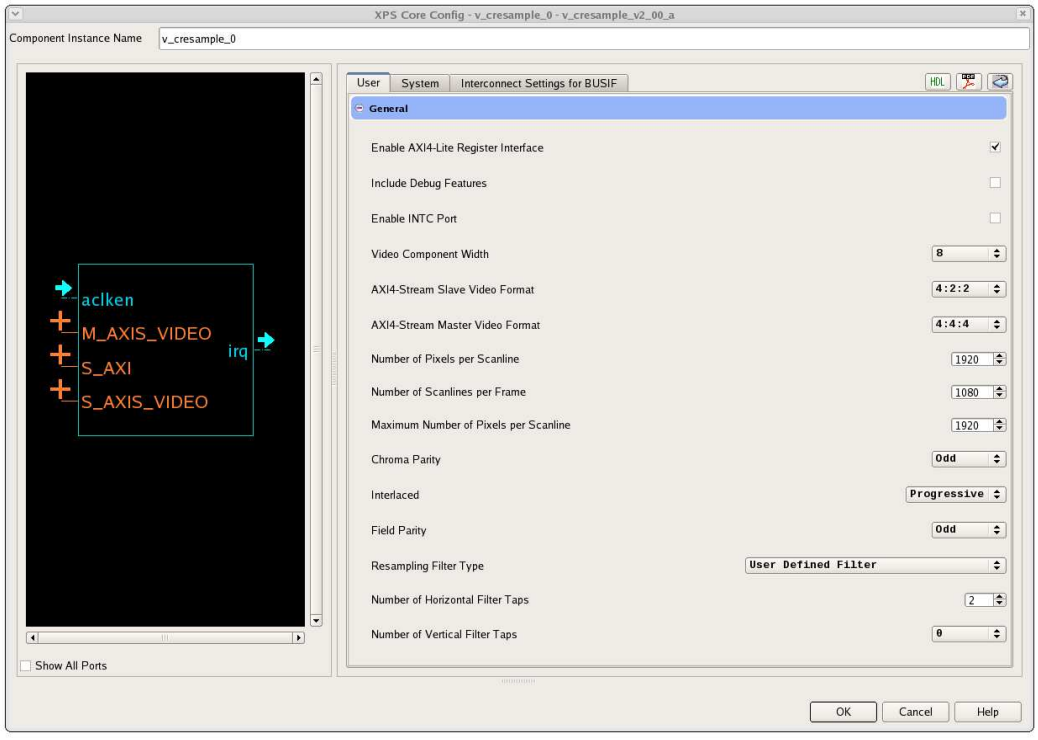

Figure 3-2: **EDK pCore GUI**

Parameter Values in the XCO File

Table 3-1 defines valid entries for the XCO parameters. Parameters are not case sensitive. 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	Default	Valid Values
component_name	chroma_resampler	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_cresample_v2_00_a" is not allowed.
s_axis_video_data_width	8	8, 10, 12
s_axis_video_format	2	3, 2, 1
m_axis_video_format	3	3, 2, 1
has_axi4_lite	false	true, false
has_intc_if	false	true, false
has_debug	false	true, false
active_cols	1920	32 – 7680
active_rows	1080	32 - 7680
max_cols	1920	32 - 7680
chroma_parity	odd	odd, even
interlaced	false	true, false
field_parity	odd	odd, even
convert_type	1	0, 1, 2
num_h_taps	2	0 – 24
num_v_taps	0	0 - 8

Output Generation

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 files listed in Table 3-2.

Table 3-2: **Files generated by CORE Generator**

Name	Description
<component_name>.xco	CORE Generator input file containing the parameters used to generate a core.
<component_name>.ngc	Binary Xilinx implementation netlist files containing the information required to implement the module in a Xilinx FPGA.
<component_name>.vho <component_name>.veo	Template files containing code that can be used as a model for instantiating.
<component_name>.vhd <component_name>.v	Structural simulation model.
/doc/pg012_v_cresample.pdf /doc/v_cresample_v2_00_a_vinfo.html	Core documents.
<component_name>.asy	Graphical symbol information file. Used by the ISE tools and some third party tools to create a symbol representing the core.
<component_name>_xmdf.tcl	ISE Project Navigator interface file. The ISE tool uses this file to determine how the files output by CORE Generator for the core can be integrated into the ISE project.
<component_name>.gise <component_name>.xise	ISE Project Navigator support files. These are generated files and should not be edited directly.
<component_name>_readme.txt	Readme file for the IP core.
<component_name>_flist.txt	Text file listing all of the output files produced when a customized core was generated in the CORE Generator.

Designing with the Core

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

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 4-1 introduces a graphical notation of sampling grid pixels.

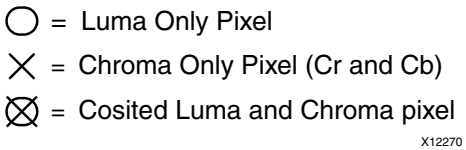

Figure 4-1: **Notation**

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 4-2.

Figure 4-2: **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 4-3.

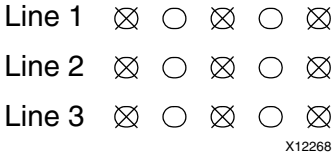

Figure 4-3: **4:2:2 Format**

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 4-4.

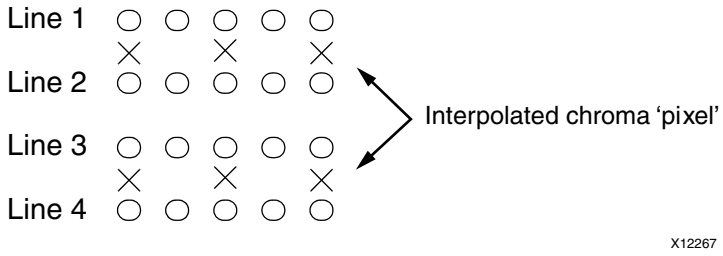

Figure 4-4: **: 4:2:0 Progressive Format**

The sampling positions of an interlaced picture are shown in Figure 4-5.

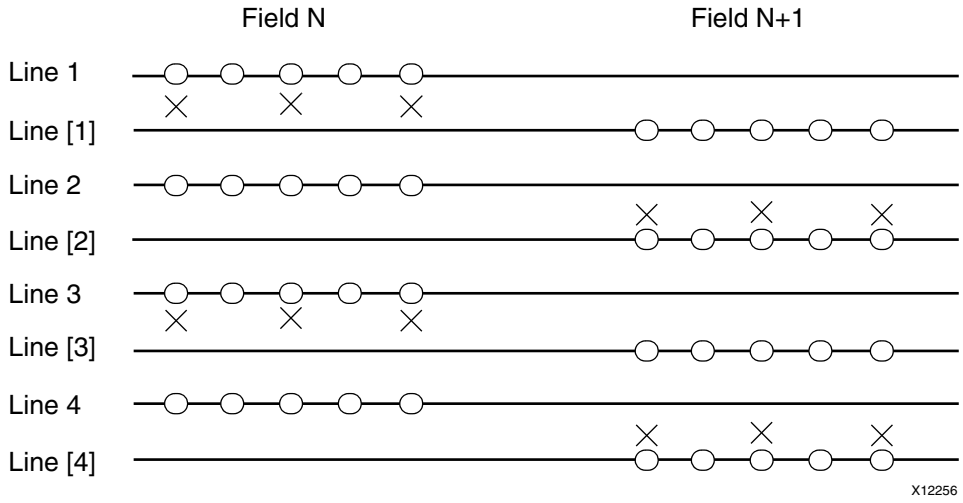

Figure 4-5: **4:2:0 Interlaced Format**

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 4-1 along with default filter information.

Table 4-1: **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 24. 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 $[\ ]_m^M$ represents rounding with clipping at M, and clamping at m.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_HPHASEp}_x \right]_0^{2^{DW}-1} \qquad \textit{Equation 4-1}$$

In phase 1, COEF00_HPHASE1 is the coefficient applied to the most recent input sample in the filter aperture. Figure 4-6 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.

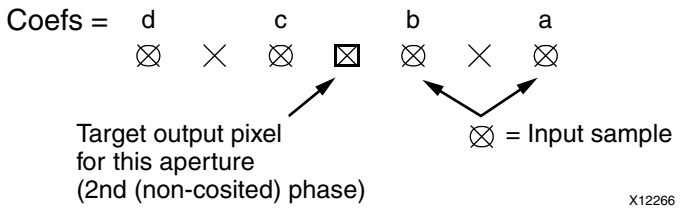

Figure 4-6: **4:2:2 to 4:4:4 Coefficient Configuration**

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 COEF*k*_HPHASE0 , 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*.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \quad \textit{Equation 4-2}$$

In phase 0, COEF00_HPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 4-7 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.

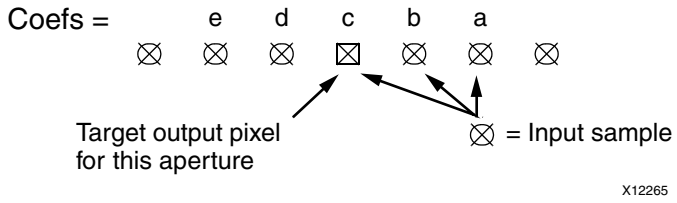

Figure 4-7: **4:4:4 to 4:2:2 Coefficient Configuration**

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_VPHASE*p*, where *k* is the coefficient index, p_y is the interpolation phase, $i_{x,y}$ are pixels from the input image, and $[\ ]_m^M$ represents rounding with clipping at *M*, and clamping at *m*.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_VPHASEp}_y \right]_0^{2^{DW}-1} \quad \textit{Equation 4-3}$$

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter Figure 4-8 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.

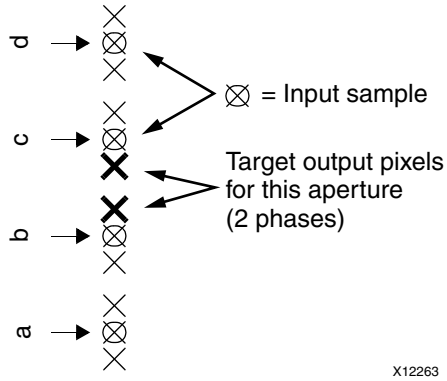

Figure 4-8: **4:2:0 to 4:2:2 Coefficient Configuration**

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.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_VPHASE0} \right]_0^{2^{DW}-1} \qquad \textit{Equation 4-4}$$

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 4-9 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.

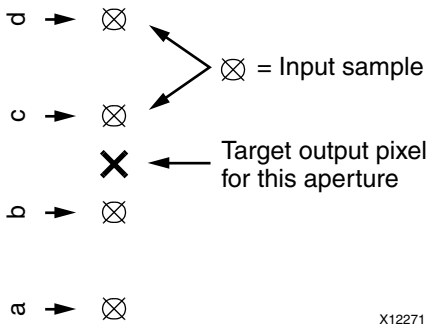

Figure 4-9: **4:2:2 to 4:2:0 Coefficient Configuration**

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 4-3. The resulting computation is shown in Equation 4-5.

$$t_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} i_{x,y-k} \text{COEFk\_VPHASEp}_y \right]_0^{2^{DW}-1} \quad \textit{Equation 4-5}$$

Next, the values are filtered according to Equation 4-1. The resulting computation is shown in Equation 4-6.

$$o_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} t_{x-k,y} \text{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \quad \textit{Equation 4-6}$$

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 4-7.

$$t_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} i_{x-k,y} \text{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \quad \textit{Equation 4-7}$$

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

$$o_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} t_{x,y-k} \text{COEFk\_VPHASE0} \right]_0^{2^{DW}-1} \quad \textit{Equation 4-8}$$

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.

Resampling Filters

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 24 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 39.

General Design Guidelines

The Chroma Resampler core converts between chroma sub-sampling formats of 4:4:4, 4:2:2, and 4:2:0. The core processes samples provided via an AXI4-Stream slave interface, outputs pixels via an AXI4-Stream master interface, and can be controlled via an optional AXI4-Lite

interface. The Chroma Resampler block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate. It is recommended that the Chroma Resampler is used in conjunction with the Video Input and Video Timing Controller cores. The Video Timing Controller core measures the timing parameters, such as number of active scan lines, number of active pixels per scan line of the image sensor. The Video Input core formats couples the sensor data interface to AXI4-Stream.

Clock, Enable, and Reset Considerations

This section details the clocking considerations when designing with the core.

ACLK

The master and slave AXI4-Stream video interfaces use the ACLK clock signal as their shared clock reference, as shown in Figure 4-10.

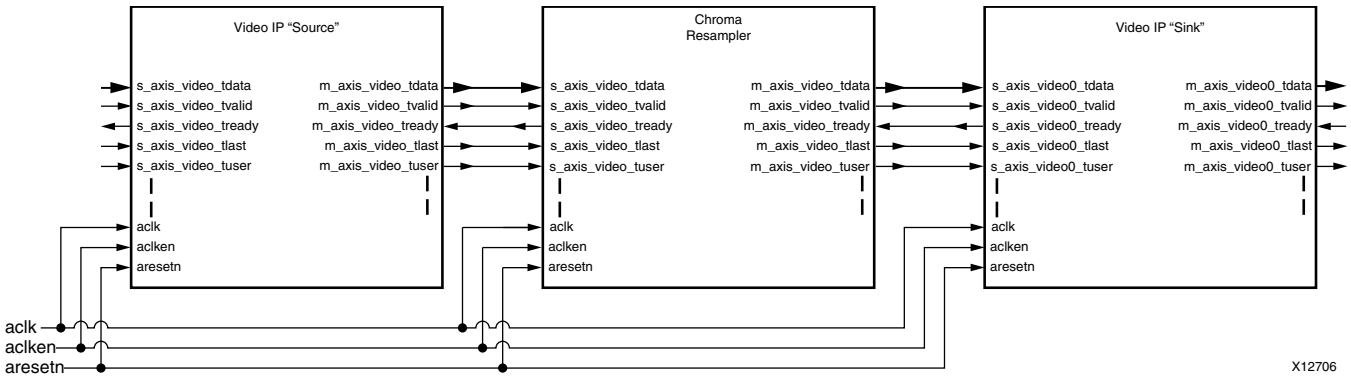

Figure 4-10: **Example of ACLK Routing in an ISP Processing Pipeline**

The ACLK pin is also shared between the AXI4-Lite and AXI4-Stream interfaces, the Chroma Resampler does not contain optional clock-domain crossing logic. If in the user system the AXI4-Lite Control interface clock (CLK_LITE) is different from the AXI4-Stream clock (CLK_STREAM), and

- ($F_{CLK\_STREAM} > F_{CLK\_LITE}$) then clock-domain crossing logic needs to be inserted in front of the AXI4-Lite Control interface and the Chroma Resampler can be clocked at the AXI4-Stream clock via ACLK,
- ($F_{CLK\_STREAM} < F_{CLK\_LITE}$) then clock-domain crossing logic needs to be inserted before the AXI4-Stream interface, and the Chroma Resampler needs to be clocked at the AXI4-Lite clock via the ACLK pin, as shown in Figure 4-11. Alternatively, if F_{CLK_LITE} greater than of the F_{MAX} of the Chroma Resampler, clock domain crossing logic can be inserted in front of the AXI4-Lite Control interface.

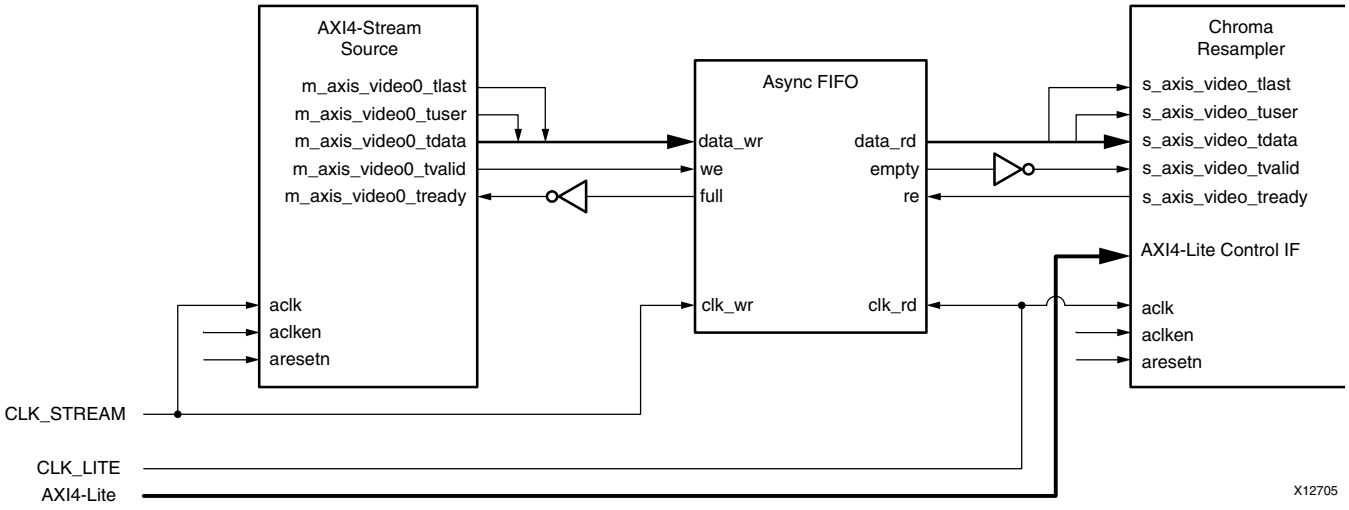

Figure 4-11: **Chroma Resampler Top-Level Signaling Interface**

In either case, Xilinx System Integrator tools, such as EDK, can automatically infer clock-domain crossing logic using the AXI interconnect core, when the tool detects that the master / slave side of AXI4 interfaces operate on different CLK rates. For manual instantiation of clock-domain crossing logic, HDL users can take advantage of the FIFO Generator IP core, as shown in Figure 4-11.

ACLKEN

The Chroma Resampler has two enable options: the ACLKEN pin (hardware clock enable), and the software reset option provided via the AXI4-Lite control interface (when present).

ACLKEN is by no means synchronized internally to AXI4-Stream frame processing therefore de-asserting ACLKEN for extended periods of time may lead to image tearing.

The ACLKEN pin facilitates:

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

Note: When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, to prevent transaction errors the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal (Figure 3-2).

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

ARESETn

The Chroma Resampler has two reset source: the ARESETn pin (hardware reset), and the software reset option provided via the AXI4-Lite control interface (when present).

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

The external reset pulse needs to be held for 32 ACLK cycles to reset the core.

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

System Considerations

When using the Chroma Resampler, it needs to be configured for the actual image frame-size to operate properly. To gather the frame size information from the image, it can be connected to the Video In to AXI4-Stream input and the Video Timing Controller. The timing detector logic in the Video Timing Controller will gather the image timing signals. The AXI4-Lite control interface on the Video Timing Controller allows the system processor to read out the measured frame dimensions, and program all downstream cores, such as the Chroma Resampler, with the appropriate image dimensions.

If the target system uses only fixed image sources with sensor aperture values fixed (no Pan-Tilt-Zoom, or cropping function), video format fixed (progressive vs interlaced, chroma parity, and field parity), and pre-defined resampling filters, the user may choose to create a constant configuration by removing the AXI4-Lite interface. This option allows reducing the core Slice footprint.

Programming Sequence

If processing parameters such as the image size needs to be changed on the fly, or the system needs to be reinitialized, it is recommended that pipelined Xilinx IP video cores are disabled/reset from system output towards the system input, and programmed/enabled from system input to system output. STATUS register bits allow system processors to identify the processing states of individual constituent cores, and successively disable a pipeline as one core after another is finished processing the last frame of data.

Error Propagation and Recovery

Parameterization and/or configuration registers define the dimensions of video frames video IP should process. Starting from a known state, based on these configuration settings the IP can predict when the beginning of the next frame is expected. Similarly, the IP can

predict when the last pixel of each scan line is expected. SOF detected before it was expected (early), or SOF not present when it is expected (late), EOL detected before expected (early), or EOL not present when expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration.

When SOF is detected early, the output SOF signal is generated early, terminating the previous frame immediately. When SOF is detected late, the output SOF signal is generated according to the programmed values. Extra lines / pixels from the previous frame are dropped until the input SOF is captured.

Similarly, when EOL is detected early, the output EOL signal is generated early, terminating the previous line immediately. When EOL is detected late, the output EOL signal is generated according to the programmed values. Extra pixels from the previous line are dropped until the input EOL is captured.

Constraining the Core

Required Constraints

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

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for the Chroma Resampler core. This core has not been characterized for use in low power devices.

Clock Frequencies

The pixel clock frequency is the required frequency for the Chroma Resampler core. See Maximum Frequencies in Chapter 2.

Clock Management

There is only one clock for the Chroma Resampler core.

Clock Placement

There are no specific Clock placement requirements for the Chroma Resampler core.

Banking

There are no specific Banking rules for the Chroma Resampler core.

Transceiver Placement

There are no Transceiver Placement requirements for the Chroma Resampler core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Chroma Resampler core.

Detailed Example Design

No example design is available at the time for 14.1.

Demonstration Test Bench

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

Test bench structure

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

- DUT

  The Chroma Resampler core instance under test.

- axi4lite_mst

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

- axi4s_video_mst

  The AXI4-Stream master module, which opens the stimuli TXT file and initiates AXI4-Stream transactions to provide stimuli data for the core

- axi4s_video_slv

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

- ce_gen

  Programmable Clock Enable (`ACLKEN`) generator

Running the Simulation

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

Directory and File Contents

The directory structure underneath the top-level folder is:

- expected:
  Contains the pre-generated expected/golden data used by the 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 simulation files and the XCO CORE Generator parameterization file of the core instance. The VHD file is a netlist generated using CORE Generator. The XCO file can be used to regenerate a new netlist using CORE Generator.

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

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

- isim_wave.wcfg:
  Waveform configuration for ISIM
- mti_wave.do:
  Waveform configuration for ModelSim

- run_isim.bat :
  Runscript for iSim in Windows
- run_isim.sh:
  Runscript for iSim in Linux
- run_mti.bat:
  Runscript for ModelSim in Windows
- run_mti.sh:
  Runscript for ModelSim in Linux

Appendix A

Verification, Compliance, and Interoperability

Simulation

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

- Register accesses
- Processing multiple frames of data
- AXI4-Stream bidirectional data-throttling tests
- Testing detection, and recovery from various AXI4-Stream framing error scenarios
- Testing different `ACLKEN` and `ARESETn` assertion scenarios
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Chroma Resampler core has been validated in hardware at Xilinx to represent a variety of parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Lite interconnect and various other peripherals. The software for the test system included pre-generated input and output data along with live video stream. The MicroBlaze processor was responsible for:
  - Initializing the appropriate input and output buffers
  - Initializing the Chroma Resampler core
  - Launching the test
  - Comparing the output of the core against the expected results

- Reporting the Pass/Fail status of the test and any errors that were found

Interoperability

The core slave (input) and master (output) AXI4-Stream interface can work directly with any Xilinx Video core that generates or consumes YCbCr 4:4:4, 4:2:2, or 4:2:0 data.

Appendix B

Migrating

From version v1.0 to v2.00.a of the Chroma Resampler, the following significant changes took place:

- XSVI interfaces were replaced by AXI4-Stream interfaces.
- Since AXI4-Stream does not carry video timing data, the timing detector and timing generator modules were trimmed.
- The pCore and General Purpose Processor and Constant modes became obsolete and were removed.
- Native support for EDK have been added - the Chroma Resampler appears in the EDK IP Catalog.
- Debugging features have been added.
- The AXI4-Lite control interface register map is standardized between Xilinx video cores.
- For YCbCr 4:4:4 video format, the order of Cb and Cr has been swapped in the video data bus. See Figure 2-2, page 18.

Because of the complex nature of these changes, replacing a v1.0 version of the core in a customer design is not trivial. An existing EDK pCore or Constant Chroma Resampler instance can be converted from XSVI to AXI4-Stream using components from XAPP521 (v1.0), *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol*.

A v1.0 pCore instance in EDK can be replaced from v2.00.a directly from the EDK IP Catalog. However, the application software needs to be updated for the changed functionality and addresses of the `IRQ_ENABLE`, `STATUS`, `ERROR`, timing, and coefficient registers. Consider replacing a legacy Chroma Resampler pCore from EDK with a v2.00.a instance without AXI4-Lite interface to save resources.

For an ISE design using the General Purpose Processor interface, all of the following steps might be necessary:

- Timing detection, generation using the Video Timing Controller Core
- Replacing XSVI interfaces with conversion modules described in XAPP521 or trying to use the Video In to AXI4-Stream core
- Updating the Chroma Resampler instance to v2.00.a with or without AXI4-Lite interface

The INTC interface and debug functionality are new features for v2.00.a. When migrating an existing design, these functions may be disabled.

Debugging

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

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

1. Bringing up the AXI4-Lite Interface
2. Bringing up the AXI4-Stream Interfaces
    - (Optional) Balancing throughput

Once the core is working as expected, the user may consider 'hardening' the configuration by replacing the Chroma Resampler with an instance where GUI default values are set to the established `ACTIVE_SIZE`, `FIELD_PARITY` and `CHROMA_PARITY` values, but the AXI4-Lite interface is disabled. This configuration reduces the core slice footprint.

Bringing up the AXI4-Lite Interface

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

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

Symptom	Solution
Readback value for the VERSION_REGISTER is different from expected default values	Does the core receive `ACLK`? Is the core enabled? Set `ACLKEN`=1 Is the core in reset? Set `ARESETn`=1. The address maps between software and hardware could get out of sync. Regenerate addresses in EDK, make sure the MHS file in EDK and the xparameters.h in the SDK project are up to date.
Readback values from values are stuck, and cannot be overwritten.	Is the target address writable?
The interface is unreliable. Subsequent reads from the same address return different value, readback values differ from values written to the same address.	Clock domain crossing issues between the host processor and the peripheral. HDL users need to make sure the AXI4-Lite Master is in the same clock domain as the AXI4-Lite Slave. If not, proper clock-domain crossing logic, such as asynchronous FIFOs need to be inserted.

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

Bringing up the AXI4-Stream Interfaces

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

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

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope™ tool, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 1 of the ERROR register reads back set.	Bit 1 of the ERROR register, EOL_LATE, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope analyzer, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 2 or Bit 3 of the ERROR register reads back set.	Bit 2 of the ERROR register, SOF_EARLY, and bit 3 of the ERROR register SOF_LATE indicate the number of pixels received between the latest and the preceding Start-Of-Frame (SOF) differ from the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using the ChipScope analyzer, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the Chroma Resampler keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically. Is m_axis_video_tready low? If so, the Chroma Resampler cannot send data downstream, and the internal FIFOs are full.
m_axis_video_tvalid stuck low, the downstream core is not receiving data	1. No data is generated during the first two lines of processing. 2. If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (s_axis_video_tlast) End-of-line signal. Check the ERROR register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the ERROR register.
Generated EOL signal (m_axis_video_tlast) signal misplaced.	Check the ERROR register.

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

Symptom	Solution
Data samples lost between Upstream core and the Chroma Resampler. Inconsistent EOL and/or SOF periods received.	1. Are the Master and Slave AXI4-Stream interfaces in the same clock domain? 2. Is proper clock-domain crossing logic instantiated between the upstream core and the Chroma Resampler (Asynchronous FIFO)? 3. Did the design meet timing? 4. Is the frequency of the clock source driving the Chroma Resampler `ACLK` pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the Chroma Resampler. Inconsistent EOL and/or SOF periods received.	1. Are the Master and Slave AXI4-Stream interfaces in the same clock domain? 2. Is proper clock-domain crossing logic instantiated between the upstream core and the Chroma Resampler (Asynchronous FIFO)? 3. Did the design meet timing? 4. Is the frequency of the clock source driving the Chroma Resampler `ACLK` pin lower than the reported Fmax reached?

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

Debugging Features

The Chroma Resampler is equipped with optional debugging features which aim to accelerate system bring-up, optimize memory and data-path architecture and reduce time to market. The optional debug features can be turned on/off via the **Include Debug Features** checkbox on the GUI when an AXI4-Lite interface is present. Turning off debug features reduces the core Slice footprint.

Built in Test-Pattern Generator

The optional built-in test-pattern generator facilitates to temporarily feed the output AXI4-Stream master interface with a predefined pattern.

Flag TEST_PATTERN (bit 5 of the CONTROL register) can turn test-pattern generation on (1) or off, when the core instance Debugging Features were enabled at generation. Within the IP this switch controls multiplexers in the AXI4-Stream path, switching between the regular core processing output and the test-pattern generator. When enabled, a set of counters generate 256 scan-lines of color-bars, each color bar 64 pixels wide, repetitively cycling through Black, Red, Green, Yellow, Blue, Magenta, Cyan, and White colors till the end of

each scan-line. After the Color-Bars segment, the rest of the frame is filled with a monochrome horizontal and vertical ramp.

Starting a system with all processing cores set to test-pattern mode, then by turning test-pattern generation off from the system output towards the system input allows successive bring-up and parameterization of subsequent cores.

Core Bypass Option

The bypass option facilitates establishing a straight through connection between input (AXI4-Stream slave) and output (AXI4-Stream master) interfaces bypassing any processing functionality.

The BYPASS flag (bit 4 of the CONTROL register) turns bypass on (1) or off (0) when the core instance debugging features were enabled at generation. Within the IP core, this switch controls multiplexers in the AXI4-Stream path.

In bypass mode, the Chroma Resampler core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. In bypass mode, YCbCr 4:4:4 to 4:2:2 (or 4:2:0) conversion passes the Y and Cb components of the YCbCr 4:4:4 input to the output. For 4:2:2 (or 4:2:0) to 4:4:4 conversion, the input is passed to the output and the top Cr output component is set to zero. Starting a system with all processing cores set to bypass, then by turning bypass off from the system input towards the system output allows verification of subsequent cores with known good stimuli.

Throughput Monitors

Throughput monitors enable the user to monitor processing performance within the core. This information can be used to help debug frame-buffer bandwidth limitation issues, and if possible, allow video application software to balance memory pathways.

Often times video systems, with multi-port access to a shared external memory, have different processing islands. For example a pre-processing sub-system working in the input video clock domain may clean up, transform, and write a video stream, or multiple video streams, to memory. The processing sub-system may read the frames out, process, scale, encode, then write frames back to the frame buffer, in a separate processing clock domain. Finally, the output sub-system may format the data and read out frames locked to an external clock.

Typically, access to external memory using a multi-port memory controller involves arbitration between competing streams. However, to maximize the throughput of the system, different memory ports may need different specific priorities. To fine tune the arbitration and dynamically balance frame rates, it is beneficial to have access to throughput information measured in different video data paths.

The SYSDEBUG0 (0x0014), or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG1 (0x0018), or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG2 (0x001C), or Pixel Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset.

Priorities of memory access points can be modified by the application software dynamically to equalize frame, or partial frame rates.

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 D

Application Software Development

This chapter contains information on programming the Chroma Resampler.

Programmer's Guide

The software API is provided to allow easy access to the Chroma Resampler AXI4-Lite registers defined in Table 3-1. To utilize the API functions, the following two header files must be included in the user C code:

```
#include "cresample.h"
#include "xparameters.h"
```

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

For examples on API function calls and integration into a user application, the drivers subdirectory of the pCore contains a file, `example.c`, in the `cresample_v2_00_a/examples` subfolder. This file is a sample C program that demonstrates how to use the Chroma Resampler pCore API.

Table D-1: **Chroma Resampler Driver Function Definitions**

Function name and parameterization	Description
`CRESAMPLE_Enable (uint32 BaseAddress)`	Enables a Chroma Resampler instance.
`CRESAMPLE_Disable (uint32 BaseAddress)`	Disables a Chroma Resampler instance.
`CRESAMPLE_Reset (uint32 BaseAddress)`	Immediately resets a Chroma Resampler instance. The core stays in reset until the RESET flag is cleared.
`CRESAMPLE_ClearReset (uint32 BaseAddress)`	Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
`CRESAMPLE_FSync_Reset (uint32 BaseAddress)`	Resets a Chroma Resampler instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.
`CRESAMPLE_ReadReg (uint32 BaseAddress, uint32 RegOffset)`	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 3-4).

Table D-1: **Chroma Resampler Driver Function Definitions**

Function name and parameterization	Description
`CRESAMPLE_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)`	Write the register selected by RegOffset (defined in Table 3-4. Data is the 32-bit value to write to the register.
`CRESAMPLE_RegUpdateEnable (uint32 BaseAddress)`	Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.
`CRESAMPLE_RegUpdateDisable (uint32 BaseAddress)`	Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.

Software Reset

Software reset reinitializes registers of the AXI4-Lite control interface to their initial value, resets FIFOs, forces `m_axis_video_tvalid` and `s_axis_video_tready` to 0. `CRESAMPLE_Reset()` and `CRESAMPLE_FSync_Reset ()` reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling `CRESAMPLE_Reset()`, or setting bit 30 of the `CONTROL` register to 1 will cause image tearing. After calling `CRESAMPLE_Reset()`, the core remains in reset until `CRESAMPLE_ClearReset()` is called.

Calling `CRESAMPLE_FSync_Reset()` automates this reset process by waiting until the core finishes processing the current frame, then asserting the reset signal internally, keeping the core in reset only for 32 `ACLK` cycles, then deasserting the signal automatically. After calling `CRESAMPLE_FSync_Reset()`, it is not necessary to call `CRESAMPLE_ClearReset()` for the core to return to normal operating mode.

Note: Calling `CRESAMPLE_FSync_Reset()` does not guarantee prompt, or real-time resetting of the core. If the AXI4-Stream communication is halted mid frame, the core will not reset until the upstream core finishes sending the current frame or starts a new frame.

Double Buffering

Coefficient registers, `ACTIVE_SIZE`, and `ENCODING` are double-buffered to ensure no image tearing happens if values are modified during frame processing. Values from the AXI4-Lite interface are latched into processor registers immediately after writing, and processor register values are copied into the active register set at the Start Of Frame (SOF) signal. Double-buffering decouples AXI4-Lite register updates from the AXI4-Stream processing, allowing software a large window of opportunity to update processing parameter values without image tearing.

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the `RegUpdateEnable()` and `RegUpdateDisable()` functions allows synchronous commitment of register changes. The Chroma Resampler core will start using the updated coefficient, `ACTIVE_SIZE`, `FIELD_PARITY`, and `CHROMA_PARITY` values only if the `REGUPDATE` flag of the `CONTROL`

register is set (1), after the next Start-Of-Frame signal (`s_axis_video_tuser`) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 3-4 has a constant that is defined in `cresample.h` which is set to the offset for that register listed in Table D-2. It is recommended that the application software uses the predefined register names instead of register values when accessing core registers, so future updates to the Chroma Resampler drivers which may change register locations will not affect the application dependent on the Chroma Resampler driver.

Table D-2: **Predefined Constants Defined in cresample.h**

Constant Name Definition	Value	Target Register
CRESAMPLE_CONTROL	0x0000	CONTROL
CRESAMPLE_STATUS	0x0004	STATUS
CRESAMPLE_ERROR	0x0008	ERROR
CRESAMPLE_IRQ_ENABLE	0x000C	IRQ_ENABLE
CRESAMPLE_VERSION	0x0010	VERSION
CRESAMPLE_SYSDEBUG0	0x0014	SYSDEBUG0
CRESAMPLE_SYSDEBUG1	0x0018	SYSDEBUG1
CRESAMPLE_SYSDEBUG2	0x001C	SYSDEBUG2
CRESAMPLE_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
CRESAMPLE_ENCODING	0x0028	ENCODING

C Model Reference

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

Features

- Bit-accurate with the Chroma Resampler v2.00.a 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 69 for full details of the interface. A C code example of how to call the model is provided in C Model Example Code, page 75.

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_v2_00_a_bitacc_model.zip` file, containing the bit accurate model for the Chroma Resampler core. This produces the directory structure and files shown in Table E-1.

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

File Name	Contents
README.txt	Release Notes
doc/pg012_v_cresample.pdf	Chroma Resampler Product Guide
v_cresample_v2_00_a_bitacc_cmodel.h	Model header file
parsers.h	Header file for reading configuration file
video_utils.h video_fio.h yuv_utils.h rgb_utils.h bmp_utils.h	Header files 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
/examples	Example input files used by C model
cresample.cfg	Sample 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_v2_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_cresample_v2_00_a_bitacc_cmodel.so
/lin64	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_cresample_v2_00_a_bitacc_cmodel.so	Model shared object library

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

File Name	Contents
libstlport.so.5.1	STL library, referenced by libIp_v_cresample_v2_00_a_bitacc_cmodel.so
/nt32	Precompiled bit-accurate ANSI C reference model for simulation on 32-bit Windows platforms
libIp_v_cresample_v2_00_a_bitacc_cmodel.dll lib_Ip_v_cresample_v2_00_a_bitacc_cmodel.lib stlport.5.1.dll	Precompiled library file for nt32 compilation
/nt64	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Windows platforms
libIp_v_cresample_v2_00_a_bitacc_cmodel.dll lib_Ip_v_cresample_v2_00_a_bitacc_cmodel.lib stlport.5.1.dll	Precompiled library file for nt64 compilation

Installation

For Linux systems, ensure that `libIp_v_cresample_v2_00_a_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 E-2.

Table E-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_v2_00_a_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_v2_00_a_generics cresample_generics;
struct  xilinx_ip_v_cresample_v2_00_a_inputs   cresample_inputs;
struct  xilinx_ip_v_cresample_v2_00_a_outputs  cresample_outputs;
```

Declaration of these structs can be found in `v_cresample_v2_00_a_bitacc_cmodel.h`.

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

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

Generic Variable	Type	Default Value	Range	Description
S_AXIS_VIDEO_FORMAT	Int	2	1, 2, 3	1=4:2:0, 2 = 4:2:2, 3=4:4:4
M_AXIS_VIDEO_FORMAT	Int	3	1, 2, 3	1=4:2:0, 2 = 4:2:2, 3=4:4:4
INTERLACED	Int	0	0, 1	0 = progressive, 1 = interlaced
NUM_H_TAPS	Int	2	0 to 24	Allowed values depend on conversion • 4:4:4 to 4:2:2: 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23 • 4:2:2 to 4:4:4: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24 • 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, 13, 15, 17, 19, 21, 23 • 4:2:0 to 4:4:4: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
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	1	0, 1, 2	0 = User Defined Filter 1 = Fixed Coefficient Filter 2 = Drop/Replicate
S_AXIS_VIDEO_DATA_WIDTH	Int	8	8,10,12	Data width of each component Y, Cb, Cr
ACTIVE_COLS	Int	1920	32 to 7680	Number of pixels per scan line
ACTIVE_ROWS	Int	1080	32 to 7680	Number of scan lines per frame
FIELD_PARITY	Int	odd	odd, even	• Odd/top field • Even/bottom field
CHROMA_PARITY	Int	odd	odd, even	• Chroma information on odd/first line • Even lines

Calling xilinx_ip_v_cresample_v2_00_a_get_default_generics(&cresample_generics) initializes the generics structure with the defaults, listed in Table E-3.

Filter coefficients can also be set dynamically through theAXI4-Lite interface; therefore this value is passed as an input to the core, along with the actual test image, or video sequence, as shown in Table E-4.

Table E-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).
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).
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 73.

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

Calling xilinx_ip_v_cresample_v2_00_a_get_default_inputs(&cresample_generics, &cresample_inputs) initializes the input structure with the default values, as described in Table E-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 72 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_v2_00_a_bitacc_simulate(
  struct xilinx_ip_v_cresample_v2_00_a_generics* generics,
  struct xilinx_ip_v_cresample_v2_00_a_inputs*   inputs,
  struct xilinx_ip_v_cresample_v2_00_a_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_v2_00_a_destroy(
  struct xilinx_ip_v_cresample_v2_00_a_inputs *input,
  struct xilinx_ip_v_cresample_v2_00_a_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 E-5 details the variables of the video structure.

Table E-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 E-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 E-6 details the modes and representations.

Table E-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
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 E-6, page 73. 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 file_dir config_file
file_dir : path to the location of the input/output files
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 S_AXIS_VIDEO DATA_WIDTH=8;     # allowed values: 8, 10, 12
CSET ACTIVE_COLS=720;               # allowed values: 32-7680
CSET ACTIVE_ROWS=480;               # allowed values: 32-7680
CSET S_AXIS_VIDEO_FORMAT = 3;       # allowed values: 3=4:4:4, 2=4:2:2, 1=4:2:0
CSET M_AXIS_VIDEO_FORMAT = 2;       # allowed values: 3=4:4:4, 2=4:2:2, 1=4:2:0
CSET INTERLACED=false;              # false=progressive, true=interlaced
CSET FIELD_PARITY=odd;              # odd=odd/top field, even=even/bottom field
CSET CONVERT_TYPE=1;                # 2=Drop/Replicate, 1=Fixed Coefficient Filter, 0=User
Defined Filter
CSET NUM_H_TAPS=3;                # number of horizontal taps, see product guide for allowed
values
CSET NUM_V_TAPS=0;                 # number of vertical taps, see product guide for allowed
values

# Input Image/Video
CSET INPUT_FILE_NAME               = Zoneplate_720x480.yuv;      # name of input file with
extension (.yuv or .bin)
CSET OUTPUT_FILE_NAME             = Zoneplate_720x480_out.yuv;  # name of output file with
same extension as input file
CSET NUMBER_OF_FRAMES              = 1;                          # number of frames
CSET NUMBER_OF_COLS                = 720;                        # number of columns
CSET NUMBER_OF_ROWS                = 480;                        # 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 E-4, page 71. 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 S_AXIS_VIDEO_FORMAT. 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_v2_00_a_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_v2_00_a_bitacc_cmodel -Wl,-rpath,.

   gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o run_bitacc_cmodel -L.
   -lIp_v_cresample_v2_00_a_bitacc_cmodel -Wl,-rpath,.
   ```

Windows (32 and 64-bit)

Precompiled library `v_cresample_v2_00_a_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_v2_00_a_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_v2_00_a_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 F

Additional Resources

Xilinx Resources

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

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

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

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the Video and Imaging Resources page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des.

References

These documents provide supplemental material useful with this 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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.
04/24/2012	2.0	Updated core to v2.00.a and ISE Design Suite v14.1. Updated the C model parameters in Table E-3. Replaced XSVI interfaces with AXI4-Stream interfaces. Added native support for EDK.

Notice of Disclaimer