Chroma Resampler v4.0

LogiCORE IP Product Guide

PG012 November 18, 2015

XILINX®

XILINX

Table of Contents

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 (1)
  - Supports 4kx2k @ 24 Hz in supported high performance devices

1. Performance on low power devices may be lower.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4-Lite, AXI4-Stream (2)
Resources	See Table 2-1 through Table 2-3.
Provided with Core	
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog
Constraints File	Provided in XDC
Simulation Models	Encrypted RTL, VHDL or Verilog Structural, C-Model
Supported Software Drivers (3)	Standalone
Tested Design Flows (4)	
Design Entry Tools	Vivado® Design Suite IP Integrator
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis Tools	Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the Vivado IP Catalog.
2. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG761 AXI Reference Guide [Ref 1].
3. Standalone driver details can be found in the SDK directory (*<install_directory>*/doc/usenglish/xilinx_drivers.htm). Linux OS and driver support information is available from //wiki.xilinx.com.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

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, only the 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 the Xilinx Vivado Design Suite. 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 and Ordering Information

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

For more information, 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

The Chroma Resampler core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* [Ref 1] 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 2-3 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):

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

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:

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

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

Table 2-1 through Table 2-3 were generated using Vivado® Design Suite with default tool options for characterization data. UltraScale™ results are expected to be similar to 7 series results.

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

Table 2-1: **Artix-7 FPGA and Zynq-7000 Device with Artix Based Programmable Logic Performance**

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	no	293	259	160	0/0	0	212
	Drop/Replicate	yes	2190	1946	1720	0/0	0	180
	Fixed Coefficient	no	450	396	305	0/0	0	212
	Fixed Coefficient	yes	2409	2071	1872	0/0	0	196
	User Defined	yes	2454	2137	1893	0/0	3	196
4:4:4 to 4:2:0	Drop/Replicate	no	339	300	189	0/0	0	219
	Drop/Replicate	yes	2347	2021	1805	0/1	0	188
	Fixed Coefficient	no	668	566	498	0/2	0	196
	Fixed Coefficient	yes	2685	2311	2086	0/2	0	188
	User Defined	yes	2753	2389	2124	0/2	5	196
4:2:2 to 4:4:4	Drop/Replicate	no	310	276	183	0/0	0	204
	Drop/Replicate	yes	2266	1970	1750	0/0	0	196
	Fixed Coefficient	no	435	391	298	0/0	0	219
	Fixed Coefficient	yes	2385	2102	1868	0/0	0	196
	User Defined	yes	2437	2104	1889	0/0	2	204

Table 2-1: **Artix-7 FPGA and Zynq-7000 Device with Artix Based Programmable Logic Performance**

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:2:2 to 4:2:0	Drop/Replicate	no	288	277	128	0/0	0	219
	Drop/Replicate	yes	2242	2001	1739	0/1	0	172
	Fixed Coefficient	no	451	398	291	0/2	0	212
	Fixed Coefficient	yes	2415	2118	1869	0/2	0	196
	User Defined	yes	2451	2133	1886	0/2	2	180
4:2:0 to 4:4:4	Drop/Replicate	no	379	342	262	0/1	0	196
	Drop/Replicate	yes	2527	2167	1942	0/2	0	164
	Fixed Coefficient	no	769	679	584	0/3	0	196
	Fixed Coefficient	yes	2945	2546	2247	0/4	0	188
	User Defined	yes	3019	2638	2307	0/4	4	188
4:2:0 to 4:2:2	Drop/Replicate	no	309	292	175	0/1	0	212
	Drop/Replicate	yes	2365	2053	1804	0/2	0	188
	Fixed Coefficient	no	510	446	353	0/3	0	204
	Fixed Coefficient	yes	2520	2199	1957	0/4	0	188
	User Defined	yes	2605	2237	1993	0/4	2	180
Device, Part, Speed: XC7A100T,FGG484, -1 (ADVANCED 1.05a 2012-08-31)								

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

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	no	289	266	160	0/0	0	288
	Drop/Replicate	yes	2315	2022	1721	0/0	0	281
	Fixed Coefficient	no	450	387	305	0/0	0	288
	Fixed Coefficient	yes	2511	2170	1875	0/0	0	296
	User Defined	yes	2504	2208	1893	0/0	3	258
4:4:4 to 4:2:0	Drop/Replicate	no	320	288	190	0/0	0	288
	Drop/Replicate	yes	2414	2094	1804	0/1	0	281
	Fixed Coefficient	no	646	549	498	0/2	0	296
	Fixed Coefficient	yes	2749	2400	2086	0/2	0	274
	User Defined	yes	2861	2462	2126	0/2	5	281

Table 2-2: **Virtex-7 FPGA Performance *(Cont'd)***

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:2:2 to 4:4:4	Drop/Replicate	no	302	270	183	0/0	0	288
	Drop/Replicate	yes	2331	2072	1754	0/0	0	281
	Fixed Coefficient	no	424	384	298	0/0	0	312
	Fixed Coefficient	yes	2486	2192	1868	0/0	0	288
	User Defined	yes	2495	2206	1890	0/0	2	242
4:2:2 to 4:2:0	Drop/Replicate	no	272	260	129	0/0	0	312
	Drop/Replicate	yes	2374	2075	1740	0/1	0	266
	Fixed Coefficient	no	416	367	291	0/2	0	288
	Fixed Coefficient	yes	2515	2185	1868	0/2	0	296
	User Defined	yes	2525	2184	1884	0/2	2	226
4:2:0 to 4:4:4	Drop/Replicate	no	391	334	262	0/1	0	296
	Drop/Replicate	yes	2599	2260	1941	0/2	0	274
	Fixed Coefficient	no	789	663	584	0/3	0	274
	Fixed Coefficient	yes	3050	2635	2246	0/4	0	288
	User Defined	yes	3145	2724	2303	0/4	4	234
4:2:0 to 4:2:2	Drop/Replicate	no	304	286	175	0/1	0	281
	Drop/Replicate	yes	2460	2150	1805	0/2	0	281
	Fixed Coefficient	no	493	441	353	0/3	0	288
	Fixed Coefficient	yes	2594	2281	1959	0/4	0	274
	User Defined	yes	2698	2353	1991	0/4	2	274
Device, Part, Speed: XC7V585T,FFG1157, -1 (ADVANCED 1.07b 2012-08-28)								

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

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:4:4 to 4:2:2	Drop/Replicate	no	296	263	160	0/0	0	288
	Drop/Replicate	yes	2286	2045	1721	0/0	0	258
	Fixed Coefficient	no	420	405	305	0/0	0	242
	Fixed Coefficient	yes	2440	2209	1875	0/0	0	258
	User Defined	yes	2483	2195	1893	0/0	3	266

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

Conversion	Filter Type	AXI4-Lite	LUT-FF Pairs	Slice-LUTs	Slice-Registers	RAMB36 BWERs/ 18BWERs	DSP48E1s	Clock Frequency (MHz)
4:4:4 to 4:2:0	Drop/Replicate	no	309	290	190	0/0	0	288
	Drop/Replicate	yes	2403	2125	1804	0/1	0	274
	Fixed Coefficient	no	663	537	498	0/2	0	296
	Fixed Coefficient	yes	2694	2425	2086	0/2	0	258
	User Defined	yes	2829	2471	2126	0/2	5	266
4:2:2 to 4:4:4	Drop/Replicate	no	290	277	183	0/0	0	304
	Drop/Replicate	yes	2307	2070	1754	0/0	0	274
	Fixed Coefficient	no	420	392	298	0/0	0	288
	Fixed Coefficient	yes	2432	2200	1868	0/0	0	296
	User Defined	yes	2462	2217	1890	0/0	2	266
4:2:2 to 4:2:0	Drop/Replicate	no	271	260	129	0/0	0	258
	Drop/Replicate	yes	2300	2110	1740	0/1	0	242
	Fixed Coefficient	no	429	366	291	0/2	0	288
	Fixed Coefficient	yes	2469	2228	1868	0/2	0	274
	User Defined	yes	2486	2206	1884	0/2	2	266
4:2:0 to 4:4:4	Drop/Replicate	no	362	344	262	0/1	0	296
	Drop/Replicate	yes	2539	2296	1941	0/2	0	266
	Fixed Coefficient	no	785	666	584	0/3	0	312
	Fixed Coefficient	yes	3015	2635	2246	0/4	0	274
	User Defined	yes	2954	2718	2303	0/4	4	266
4:2:0 to 4:2:2	Drop/Replicate	no	305	285	175	0/1	0	304
	Drop/Replicate	yes	2387	2181	1805	0/2	0	281
	Fixed Coefficient	no	504	437	353	0/3	0	242
	Fixed Coefficient	yes	2567	2318	1959	0/4	0	274
	User Defined	yes	2650	2347	1991	0/4	2	274
Device, Part, Speed: XC7K70T,FBG484, -1 (ADVANCED 1.07b 2012-08-28)								

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.

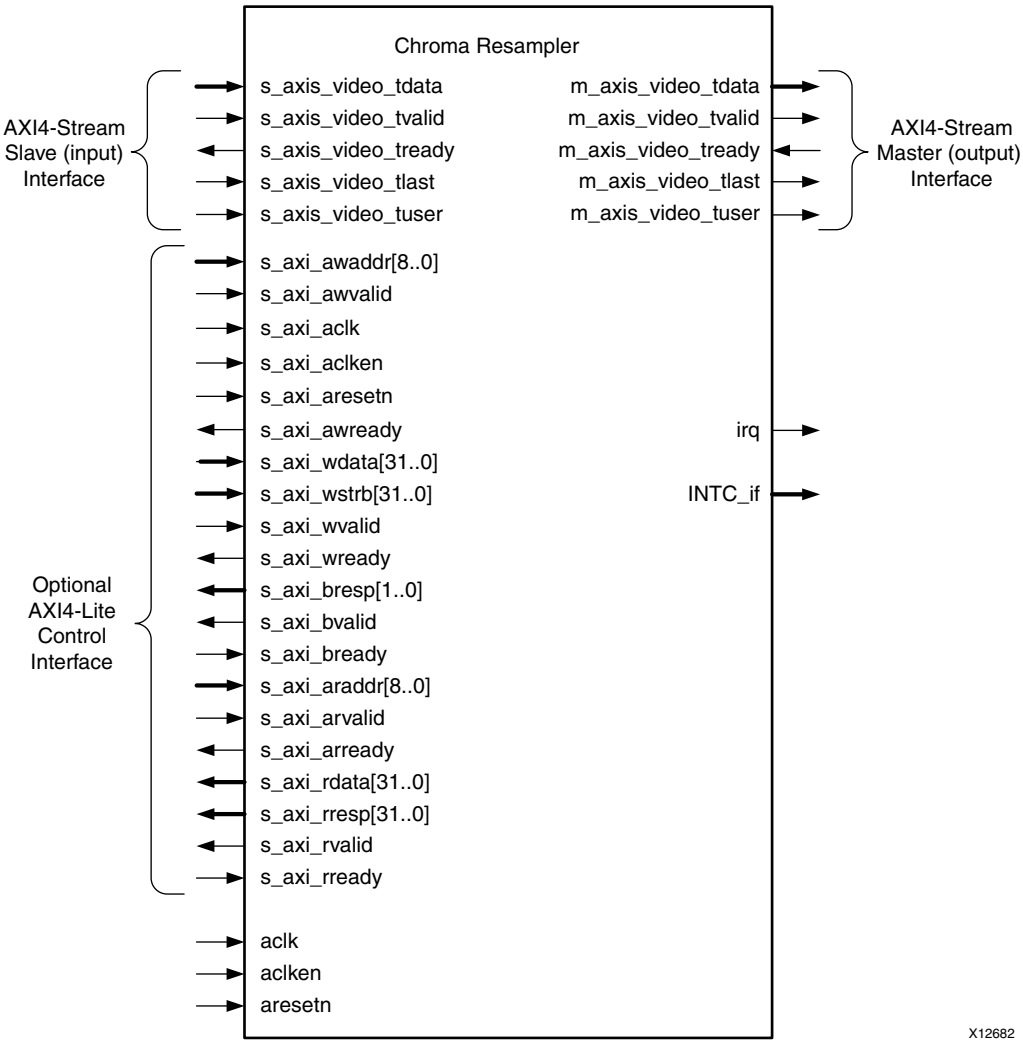

Figure 2-1: **Chroma Resampler Top-Level Signaling Interface**

Common Interface Signals

Table 2-4 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-4: **Common Interface Signals**

Signal Name	Direction	Width	Description
ACLK	In	1	Video Core Clock
ACLKEN	In	1	Video Core Active High Clock Enable
ARESETn	In	1	Video Core Active Low Synchronous Reset

Table 2-4: **Common Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
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 and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock, clock enable and reset pins: S_AXI_ACLK, S_AXI_ACLKEN and S_AXI_ARESETn. Refer to The Interrupt Subsystem for a description of the INTC_IF and IRQ pins.

ACLK

The AXI4-Stream interface must be synchronous to the core clock signal ACLK. All AXI4-Stream interface input signals are sampled on the rising edge of ACLK. All AXI4-Stream output signal changes occur after the rising edge of ACLK. The AXI4-Lite interface is unaffected by the ACLK signal.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to AXI4-Stream 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. The AXI4-Lite interface is unaffected by the ACLKEN signal.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to only AXI4-Stream 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. The ARESETn signal must be synchronous to the ACLK and must be held low for a minimum of 32 clock cycles of the slowest clock. The AXI4-Lite interface is unaffected by the ARESETn signal.

Data Interface

The Chroma Resampler receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *Vivado AXI Reference Guide* (UG1037), Video IP: AXI Feature Adoption section [Ref 1].

AXI4-Stream Signal Names and Descriptions

Table 2-5 describes the AXI4-Stream signal names and descriptions.

Table 2-5: **AXI4-Stream Data Interface Signal Descriptions**

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

Video Data

The AXI4-Stream interface specification restricts `TDATA` widths to integer multiples of 8 bits. Therefore, 10 and 12 bit 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.

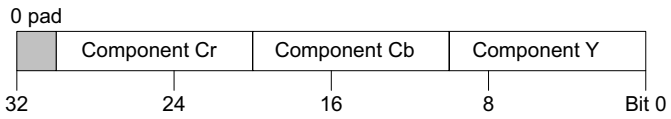

Figure 2-2: **YCbCr Data Encoding for 4:4:4 on m_axis_video_tdata**

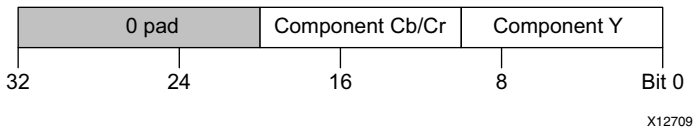

Figure 2-3: **YCbCr Data Encoding for 4:2:2 or 4:2:0 on m_axis_video_tdata**

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.

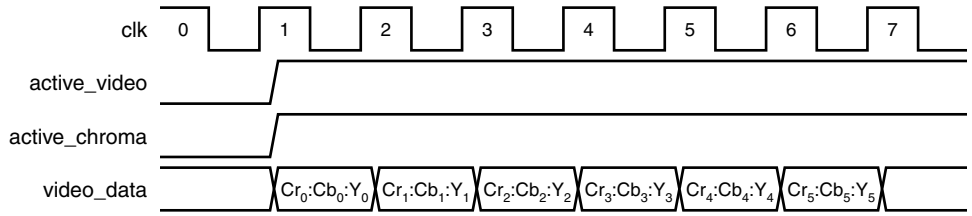

Figure 2-4: **YCbCr 4:4:4**

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.

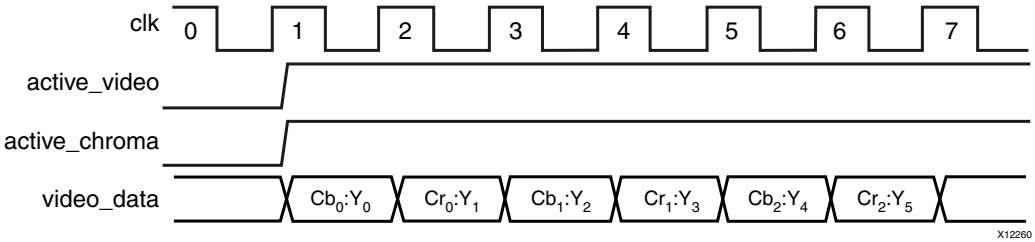

Figure 2-5: **YCbCr 4:2:2**

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.

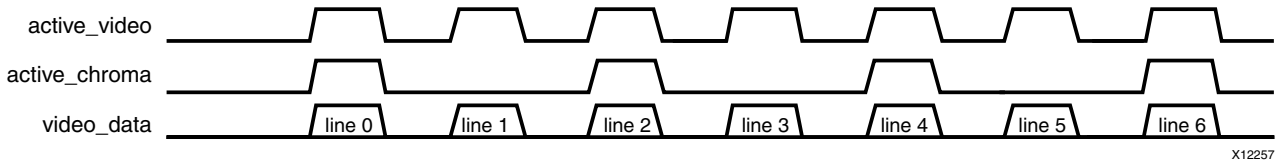

Figure 2-6: **YCbCr 4:2:0**

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 2-6. 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.

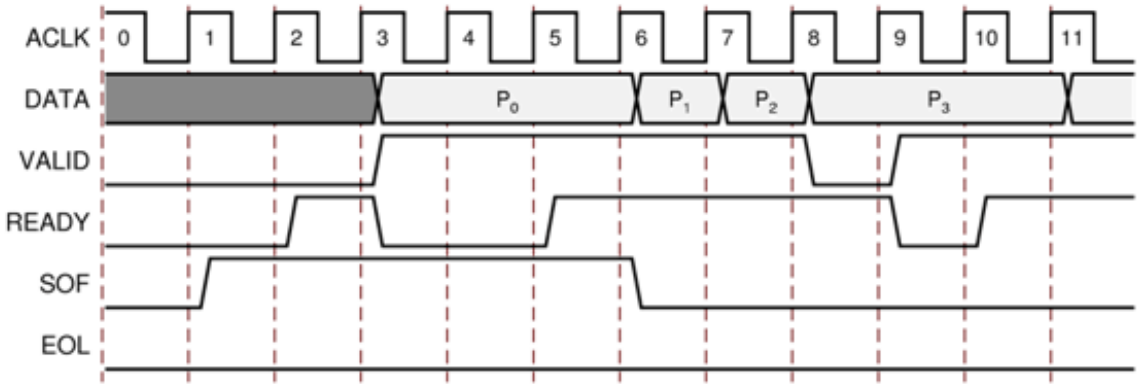

Figure 2-7: **Example of READY/VALID Handshake, Start of a New Frame**

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.

RECOMMENDED: *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.

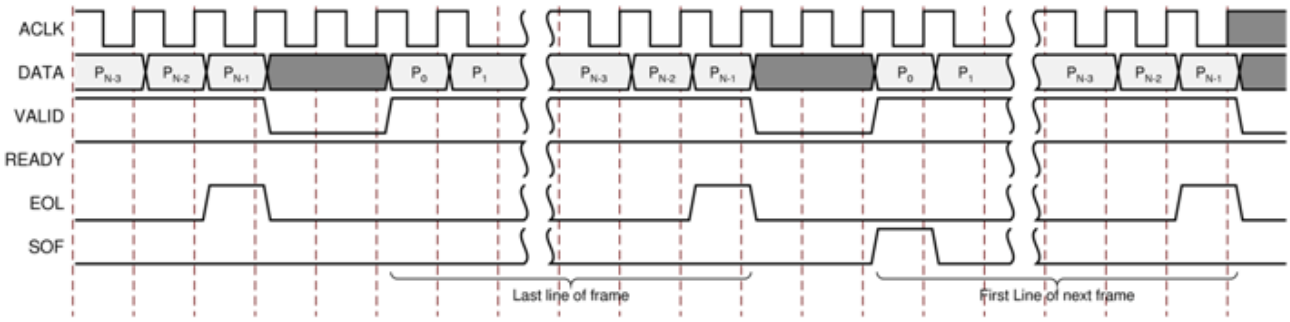

Figure 2-8: **Use of EOL and SOF Signals**

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 you 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 2-6: **AXI4-Lite Interface Signals**

Signal Name	Direction	Width	Description
s_axi_aclk	In	1	AXI4-Lite clock
s_axi_aclken	In	1	AXI4-Lite clock enable
s_axi_aresetn	In	1	AXI4-Lite synchronous Active Low reset
s_axi_awvalid	In	1	AXI4-Lite Write Address Channel Write Address Valid.
s_axi_awread	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
s_axi_awaddr	In	32	AXI4-Lite Write Address Bus
s_axi_wvalid	In	1	AXI4-Lite Write Data Channel Write Data Valid.
s_axi_wready	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
s_axi_wdata	In	32	AXI4-Lite Write Data Bus
s_axi_bresp	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
s_axi_bvalid	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
s_axi_bready	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
s_axi_arvalid	In	1	AXI4-Lite Read Address Channel Read Address Valid
s_axi_arready	Out	1	Ready. Indicates DMA is ready to accept the read address.
s_axi_araddr	In	32	AXI4-Lite Read Address Bus
s_axi_rvalid	Out	1	AXI4-Lite Read Data Channel Read Data Valid
s_axi_rready	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
s_axi_rdata	Out	32	AXI4-Lite Read Data Bus
s_axi_rresp	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.

S_AXI_ACLK

The AXI4-Lite interface must be synchronous to the S_AXI_ACLK clock signal. The AXI4-Lite interface input signals are sampled on the rising edge of ACLK. The AXI4-Lite output signal changes occur after the rising edge of ACLK. The AXI4-Stream interfaces signals are not affected by the S_AXI_ACLK.

S_AXI_ACLKEN

The S_AXI_ACLKEN pin is an active-high, synchronous clock-enable input for the AXI4-Lite interface. Setting S_AXI_ACLKEN low (de-asserted) halts the operation of the AXI4-Lite interface despite rising edges on the S_AXI_ACLK pin. AXI4-Lite interface states are maintained, and AXI4-Lite interface output signal levels are held until S_AXI_ACLKEN is asserted again. When S_AXI_ACLKEN is de-asserted, AXI4-Lite interface inputs are not sampled, except S_AXI_ARESETn, which supersedes S_AXI_ACLKEN. The AXI4-Stream interfaces signals are not affected by the S_AXI_ACLKEN.

S_AXI_ARESETn

The S_AXI_ARESETn pin is an active-low, synchronous reset input for the AXI4-Lite interface. S_AXI_ARESETn supersedes S_AXI_ACLKEN, and when set to 0, the core resets at the next rising edge of S_AXI_ACLK even if S_AXI_ACLKEN is de-asserted. The S_AXI_ARESETn signal must be synchronous to the S_AXI_ACLK and must be held low for a minimum of 32 clock cycles of the slowest clock. The S_AXI_ARESETn input is resynchronized to the ACLK clock domain. The AXI4-Stream interfaces and core signals are also reset by S_AXI_ARESETn.

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 2-7: **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	0x04000000	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 2-7: **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 2-7: **Register Names and Descriptions *(Cont'd)***

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

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.

IMPORTANT: *Bits of the STATUS register remain set until cleared.*

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. This is to facilitate identification of the interrupt source.

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

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

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 2-7 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 Scanline** 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 2-8.

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

Table 2-8: **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 2-8: **INTC_IF Signal Functions *(Cont'd)***

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

In a system integration tool, 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.

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 3-1 introduces a graphical notation of sampling grid pixels.

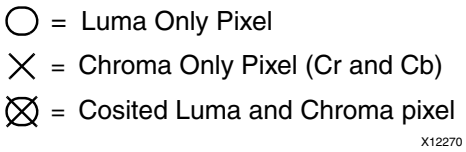

Figure 3-1: **Notation**

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 3-2.

Figure 3-2: **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 3-3.

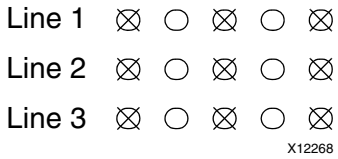

Figure 3-3: **4:2:2 Format**

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 3-4.

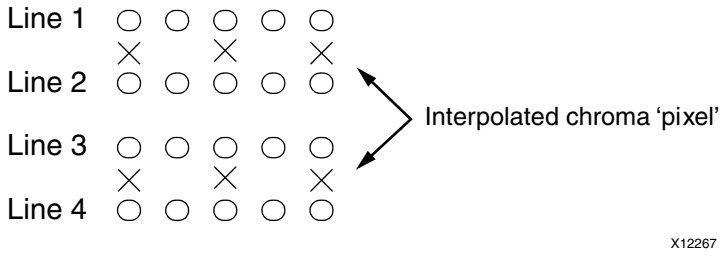

Figure 3-4: **: 4:2:0 Progressive Format**

The sampling positions of an interlaced picture are shown in Figure 3-5.

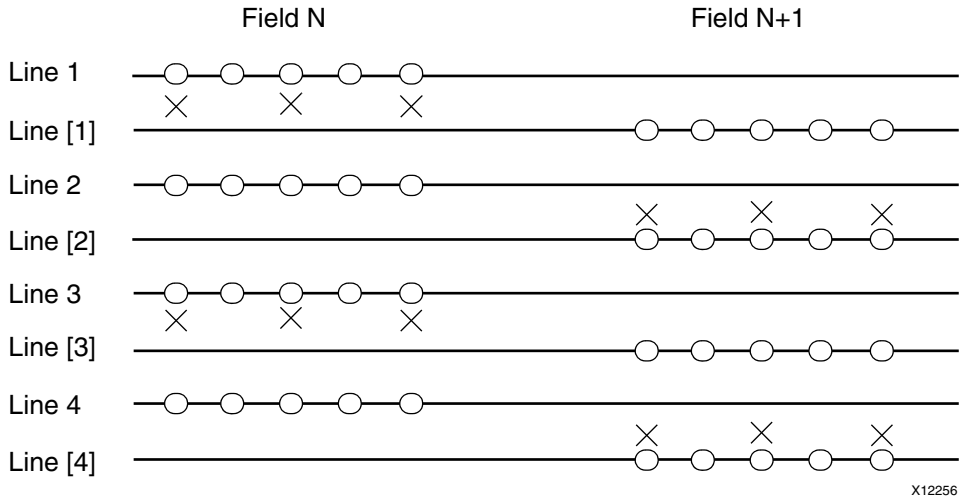

Figure 3-5: **4:2:0 Interlaced Format**

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 3-1 along with default filter information.

Table 3-1: **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 by 3 Horizontal Taps	
4:2:2 to 4:4:4	Horizontal Interpolation	2 Horizontal Taps	Only phase1 needed

Table 3-1: **Filter Summary *(Cont'd)***

Converter	Filter Configuration	Default FIR Size	Notes
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.

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

In phase 1, COEF00_HPHASE1 is the coefficient applied to the most recent input sample in the filter aperture. Figure 3-6 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.

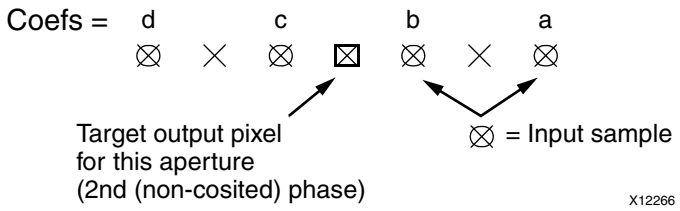

Figure 3-6: **4:2:2 to 4:4:4 Coefficient Configuration**

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*.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_HPHASE0} \right]_0^{2^{DW}-1} \qquad \textit{Equation 3-2}$$

In phase 0, COEF00_HPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 3-7 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.

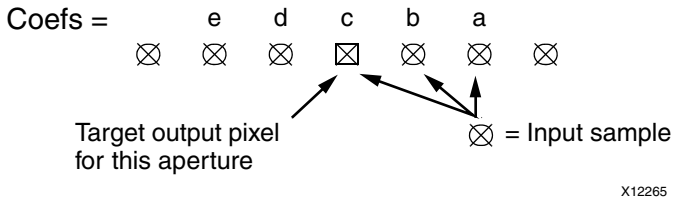

Figure 3-7: **4:4:4 to 4:2:2 Coefficient Configuration**

The default coefficients are [0.25 0.5 0.25].

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

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

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

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter Figure 3-8 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.

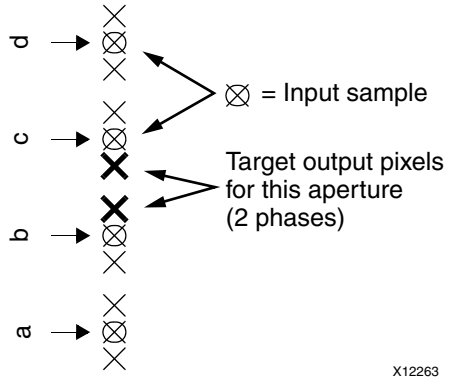

Figure 3-8: **4:2:0 to 4:2:2 Coefficient Configuration**

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*.

$$o_{x,y} = \left[\sum_{k=0}^{N_{taps}-1} i_{x-k,y}\text{COEFk\_VPHASE0} \right]_0^{2^{DW}-1} \quad \textit{Equation 3-4}$$

In phase 0, COEF00_VPHASE0 is the coefficient applied to the most recent input sample in the filter. Figure 3-9 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.

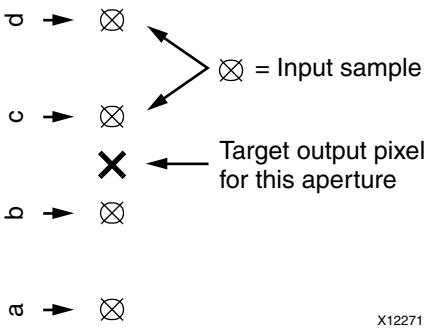

Figure 3-9: **4:2:2 to 4:2:0 Coefficient Configuration**

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

$$t_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} i_{x,y-k}\text{COEFk\_VPHASEp}_y \right]_0^{2^{DW}-1} \quad \textit{Equation 3-5}$$

Next, the values are filtered according to Equation 3-1. The resulting computation is shown in Equation 3-6.

$$o_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} t_{x-k,y} \text{COEFk\_HPHASE0} \right]_{0}^{2^{DW}-1} \quad \textit{Equation 3-6}$$

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 3-7.

$$t_{x,y} = \left[\sum_{k=0}^{N_{Htaps}-1} i_{x-k,y} \text{COEFk\_HPHASE0} \right]_{0}^{2^{DW}-1} \quad \textit{Equation 3-7}$$

Next, these values are filtered according to Equation 3-4. The resulting computation is shown in Equation 3-8.

$$o_{x,y} = \left[\sum_{k=0}^{N_{Vtaps}-1} t_{x,y-k}\text{COEFk\_VPHASE0} \right]_{0}^{2^{DW}-1} \quad \textit{Equation 3-8}$$

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 32.

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.

RECOMMENDED: *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

ACLK

The master and slave AXI4-Stream video interfaces use the `ACLK` clock signal as their shared clock reference, as shown in Figure 3-10.

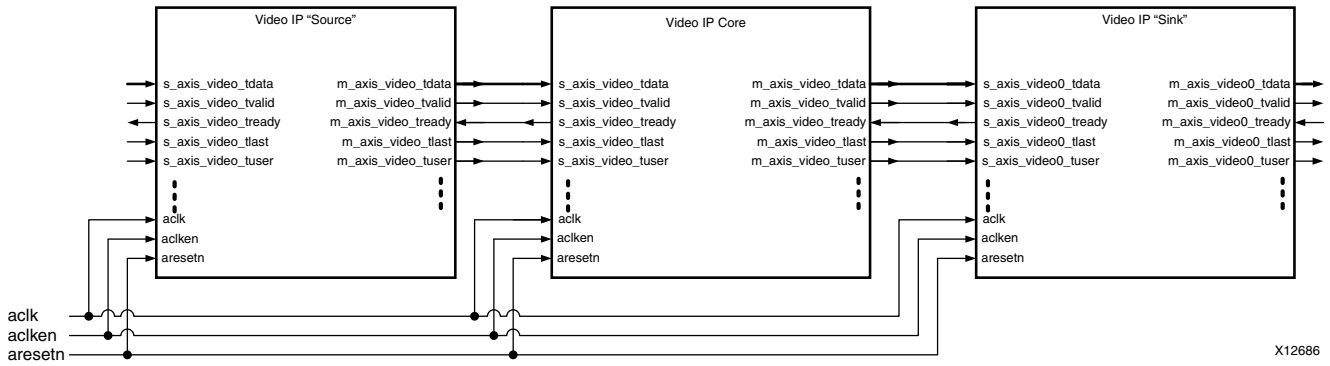

Figure 3-10: **Example of ACLK Routing in an ISP Processing Pipeline**

S_AXI_ACLK

The AXI4-Lite interface uses the S_AXI_ACLK pin as its clock source. The ACLK pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The Chroma Resampler core contains clock-domain crossing logic between the ACLK (AXI4-Stream and Video Processing) and S_AXI_ACLK (AXI4-Lite) clock domains. The core automatically ensures that the AXI4-Lite transactions completes even if the video processing is stalled with ARESETn, ACLKEN or with the video clock not running.

ACLKEN

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

ACLKEN may not be 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

IMPORTANT: *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 2-2).*

IMPORTANT: *When two cores connected through 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 through the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).

S_AXI_ACLKEN

The S_AXI_ACLKEN is the clock enable signal for the AXI4-Lite interface only. Driving this signal Low only affects the AXI4-Lite interface and does not halt the video processing in the ACLK clock domain.

ARESETn

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

IMPORTANT: *ARESETn is not synchronized internally to AXI4-Stream frame processing. Deasserting ARESETn while a frame is being process leads to image tearing.*

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

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

S_AXI_ARESETn

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

System Considerations

The Chroma Resampler must 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 one configuration of the Chroma Resampler (i.e. does not need to be reprogrammed ever), you may choose to create a constant configuration by removing the AXI4-Lite interface. This reduces the core Slice footprint.

Clock Domain Interaction

The ARESETn and ACLKEN input signals will not reset or halt the AXI4-Lite interface. This allows the video processing to be reset or halted separately from the AXI4-Lite interface without disrupting AXI4-Lite transactions.

The AXI4-Lite interface will respond with an error if the core registers cannot be read or written within 128 S_AXI_ACLK clock cycles. The core registers cannot be read or written if the ARESETn signal is held low, if the ACLKEN signal is held low, or if the ACLK signal is not connected or not running. If core register read does not complete, the AXI4-Lite read transaction will respond with **10** on the S_AXI_RRESP bus. Similarly, if a core register write does not complete, the AXI4-Lite write transaction will respond with **10** on the S_AXI_BRESP bus. The S_AXI_ARESETn input signal resets the entire core.

Programming Sequence

If processing parameters such as the image size need to be changed on the fly, or if 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 finishes 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 to expect the next frame. 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.

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite environment.

Vivado Integrated Design Environment (IDE)

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click on the selected IP or select the Customize IP command from the toolbar or popup menu.

For details, see the sections, "Working with IP" and "Customizing IP for the Design" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the "Working with the Vivado IDE" section in the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 5].

If you are customizing and generating the core in the Vivado IP Integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994) [Ref 7] for detailed information. IP Integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value you can run the `validate_bd_design` command in the Tcl console.

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Interface

The Chroma Resampler LogiCORE IP is easily configured to meet the developer's specific needs through the Vivado IP Catalog interface. This section provides a quick reference to the parameters that can be configured at generation time.

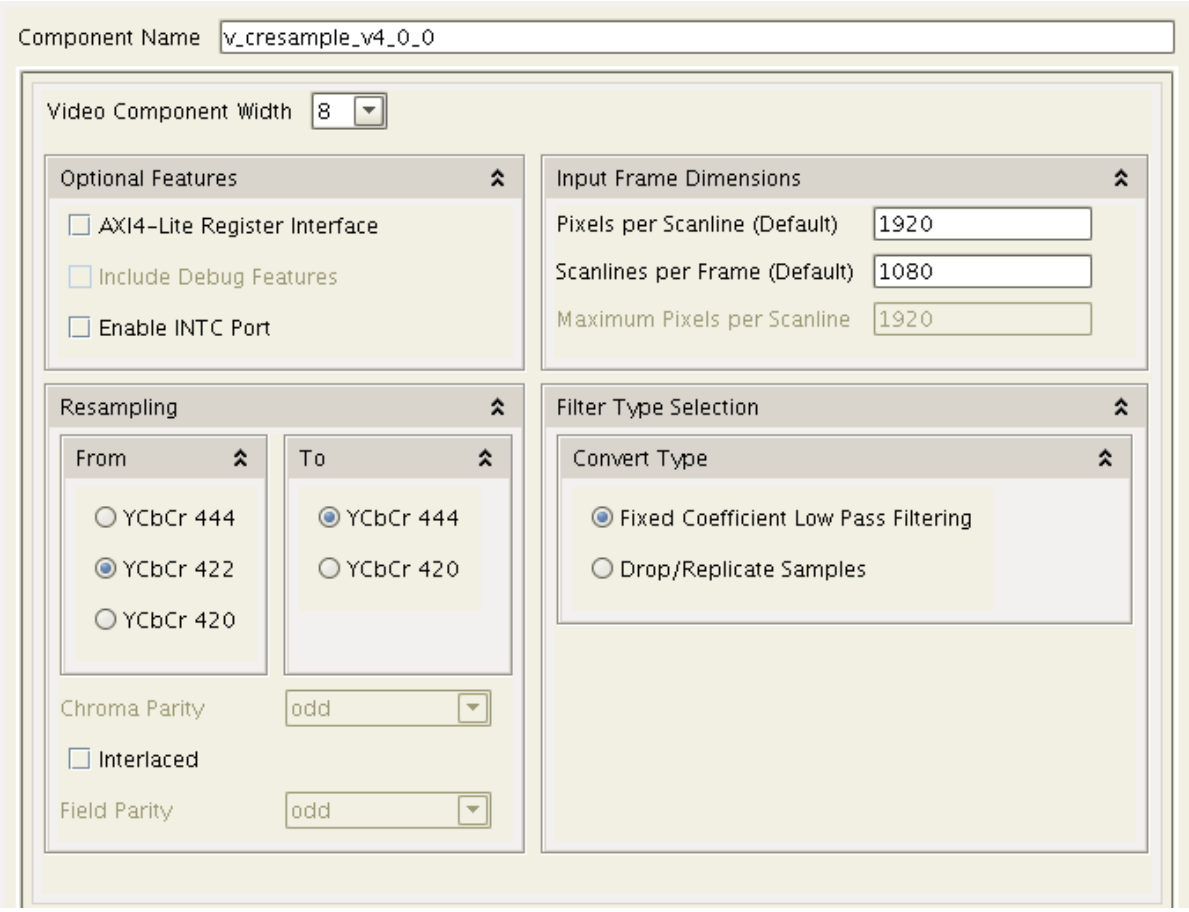

Figure 4-1: **Chroma Resampler Vivado IP Catalog GUI**

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_v4_0` 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. When using IP Integrator, this parameter is automatically computed based on the Video Component Width of the video IP core connected to the slave AXI-Stream video interface.
- **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 2.

- **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.

IMPORTANT: *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 2.

- **Input Frame Dimensions**:
  - **Pixels per Scanline**: 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.
  - **Scanlines 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 Pixels Per Scanline**: 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 **Pixels per Scanline**, or the corresponding lower half-word of the `ACTIVE_SIZE` register must always be less than or equal to the value provided by **Maximum Pixels Per Scanline**. 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 **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 Implementation in Chapter 3.
  - **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.

Output Generation

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

Constraining the Core

Required Constraints

The only constraints required are clock frequency constraints for the video clock, `clk`, and the AXI4-Lite clock, `s_axi_aclk`. Paths between the two clock domains should be constrained with a `max_delay` constraint and use the `datapathonly` flag, causing setup and hold checks to be ignored for signals that cross clock domains. These constraints are provided in the XDC constraints file included with the core.

Simulation

This chapter contains information about simulating IP in the Vivado® Design Suite environment. For comprehensive information about Vivado simulation components, as well as information about using supported third party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 6].

Synthesis and Implementation

For details about synthesis and implementation, see "Synthesizing IP" and "Implementing IP" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

C Model Reference

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

Features

- Bit-accurate with the Chroma Resampler v4.0 core
- Statically linked library (`.lib` for Windows)
- Dynamically linked library (`.so` for Linux)
- Available for 32-bit and 64-bit Windows platforms and 32-bit and 64-bit Linux platforms
- Supports all features of the Chroma Resampler core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code showing how to use the function is provided
- Example application C code wrapper file supports 8-bit YUV and BIN

Overview

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

See Using the C Model, page 53 for full details of the interface. A C code example of how to call the model is provided in C Model Example Code, page 59.

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.

User Instructions

Unpacking and Model Contents

Unzip the `v_cresample_v4_0_bitacc_model.zip` file, containing the bit-accurate model for the Chroma Resampler core. This produces the directory structure and files shown in Table 8-1.

Table 8-1: **Directory Structure and Files of Bit-Accurate Model**

File Name	Contents
`v_cresample_v4_0_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
files included in the lin.zip file	Precompiled bit-accurate ANSI C reference model for simulation on 32-bit Linux platforms
`libIp_v_cresample_v4_0_bitacc_cmodel.so`	Model shared object library
files included in the lin64.zip file	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Linux platforms
`libIp_v_cresample_v4_0_bitacc_cmodel.so`	Model shared object library
files included in the nt.zip file	Precompiled bit-accurate ANSI C reference model for simulation on 32-bit Windows platforms
`libIp_v_cresample_v4_0_bitacc_cmodel.dll` `lib_Ip_v_cresample_v4_0_bitacc_cmodel.lib`	Precompiled library file for nt32 compilation
files included in the nt.zip file	Precompiled bit-accurate ANSI C reference model for simulation on 64-bit Windows platforms
`libIp_v_cresample_v4_0_bitacc_cmodel.dll` `lib_Ip_v_cresample_v4_0_bitacc_cmodel.lib`	Precompiled library file for nt64 compilation

Installation

For Linux systems, ensure that `libIp_v_cresample_v4_0_bitacc_cmodel.so` is included in the `$LD_LIBRARY_PATH` environment variable.

Software Requirements

The Chroma Resampler C models were compiled and tested with the software shown in Table 8-2.

Table 8-2: **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 2008

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_v4_0_bitacc_cmodel.h`.

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

```
struct  xilinx_ip_v_cresample_v4_0_generics cresample_generics;
struct  xilinx_ip_v_cresample_v4_0_inputs   cresample_inputs;
struct  xilinx_ip_v_cresample_v4_0_outputs  cresample_outputs;
```

Declaration of these structs can be found in `v_cresample_v4_0_bitacc_cmodel.h`.

The generic parameters and default values are listed in Table 8-3. For an actual instance of the core, these parameters can only be set during generation through the GUI.

Table 8-3: **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

Table 8-3: **Model Generic Parameters and Default Values *(Cont'd)***

Generic Variable	Type	Default Value	Range	Description
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_v4_0_get_default_generics(&cresample_generics)
```

initializes the generics structure with the defaults, listed in Table 8-3.

Filter coefficients can also be set dynamically through the AXI4-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 8-4.

Table 8-4: **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 57.

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

Calling

```
xilinx_ip_v_cresample_v4_0_get_default_inputs(&cresample_generics,
&cresample_inputs)
```

initializes the input structure with the default values, as described in Table 8-4.

IMPORTANT: *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 56 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_v4_0_bitacc_simulate(
  struct xilinx_ip_v_cresample_v4_0_generics* generics,
  struct xilinx_ip_v_cresample_v4_0_inputs*   inputs,
  struct xilinx_ip_v_cresample_v4_0_outputs*  outputs).
```

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

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

```
void xilinx_ip_v_cresample_v4_0_destroy(
  struct xilinx_ip_v_cresample_v4_0_inputs *input,
  struct xilinx_ip_v_cresample_v4_0_outputs *output).
```

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

Chroma Resampler Input and Output Video Structure

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

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

Table 8-5 details the variables of the video structure.

Table 8-5: **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 8-6 details the modes and representations.

Table 8-6: **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

Table 8-6: **Named Video Modes with Corresponding Planes and Representations**

Mode	Planes	Video Representation
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 8-6, page 56. 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 8-4, page 55. 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 file from the `/lin32` or `/lin64` directory to the root directory:

   ```
   libIp_v_cresample_v4_0_bitacc_cmodel.so
   ```

3. 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_v4_0_bitacc_cmodel -Wl,-rpath,.

   gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o run_bitacc_cmodel -L.
   -lIp_v_cresample_v4_0_bitacc_cmodel -Wl,-rpath,.
   ```

Windows (32 and 64-bit)

Precompiled library `v_cresample_v4_0_bitacc_cmodel.lib` and top level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. This section presents an example using Microsoft Visual Studio.

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

- `libIp_v_cresample_v4_0_bitacc_cmodel.lib` to the Resource Files folder of the project
- `run_bitacc_cmodel.c` and `parsers.c` to the Source Files folder of the project
- `v_cresample_v4_0_bitacc_cmodel.h` to Header Files folder of the project

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

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

Detailed Example Design

No example design is available at this time. For a comprehensive listing of Video and Imaging application notes, white papers, related IP cores including the most recent reference designs available, see the Video and Imaging Resources page at www.xilinx.com/esp/video/refdes_listing.htm.

Test Bench

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

Demonstration Test Bench

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

Directory and File Contents

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

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

Test Bench Structure

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

It instantiates the following modules:

- `DUT`

  The <IP> core instance under test.

- `axi4lite_mst`

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

- `axi4s_video_mst`

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

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

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

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

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

- `axi4s_video_slv`

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

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

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

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

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

- `ce_gen`

  Programmable Clock Enable (`ACLKEN`) generator.

Verification, Compliance, and Interoperability

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. The AXI4-Stream interfaces must be compliant to the AXI4-Stream Video Protocol as described in Video IP: AXI Feature Adoption section of the *AXI Reference Guide* [Ref 1].

Migrating and Upgrading

This appendix contains information about migrating from an ISE design to the Vivado Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading their IP core, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migration to Vivado Design Suite, see *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 2].

Upgrading in Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

Parameter Changes

There are no parameter changes.

Port Changes

There are no port changes.

Other Changes

From version v3.01.a to v4.0 of the Chroma Resampler, no significant change took place.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the Chroma Resampler, the Xilinx Support web page (Xilinx Support web page) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support Web Case.

Documentation

This product guide is the main document associated with the Chroma Resampler. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Answer Records for the Chroma Resampler Core

AR 54518

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

1. Open a WebCase by selecting the WebCase link located under Support Quick Links.

- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.

Note: Access to WebCase is not available in all cases. Please login to the WebCase tool to see your specific support options.

Debug Tools

There are many tools available to address Chroma Resampler core design issues. It is important to know which tools are useful for debugging various situations.

Vivado Lab Tools

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

The Vivado lab tools logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Reference Boards

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

- 7 series evaluation boards
  - KC705
  - KC724

C Model Reference

See *C Model Reference* in this guide for tips and instructions for using the provided C model files to debug your design.

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

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.

Flag `BYPASS` (bit 4 of the `CONTROL` register) can turn bypass 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.

In bypass mode the core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output.

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.

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, Green, Blue, Cyan, Red, Yellow, Magenta, 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.

Throughput Monitors

Throughput monitors enable monitoring 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 multiport 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 multiport 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 datapaths.

The SYSDEBUG0 (0x0014) (or Frame Throughput Monitor) 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 black screen for RGB color systems and in a dark-green screen for YUV color systems.

Interface Debug

AXI4-Lite Interfaces

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

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

Symptom	Solution
Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Are the S_AXI_ACLK and ACLK pins connected? The VERSION_REGISTER readout issue may be indicative of the core not receiving the AXI4-Lite interface.
Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core enabled? Is s_axi_aclken connected to vcc? Verify that signal ACLKEN is connected to either net_vcc or to a designated clock enable signal.

Table C-1: **Troubleshooting the AXI4-Lite Interface *(Cont'd)***

Symptom	Solution
Readback from the Version Register through the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core in reset? S_AXI_ARESETn and ARESETn should be connected to vcc for the core not to be in reset. Verify that the S_AXI_ARESETn and ARESETn signals are connected to either net_vcc or to a designated reset signal.
Readback value for the VERSION_REGISTER is different from expected default values	The core and/or the driver in a legacy project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared.

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

AXI4-Stream Interfaces

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

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

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Vivado Lab Tools, 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 Vivado Lab Tools, 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 Vivado Lab Tools, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the core keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically. Is m_axis_video_tready low? If so, the core cannot send data downstream, and the internal FIFOs are full.

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

Symptom	Solution
m_axis_video_tvalid stuck low, the downstream core is not receiving data	• No data is generated during the first two lines of processing. • If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (`s_axis_video_tlast`) End-of-line signal. Check the `ERROR` register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the `ERROR` register.
Generated EOL signal (`m_axis_video_tlast`) signal misplaced.	Check the `ERROR` register.
Data samples lost between Upstream core and this core. Inconsistent EOL and/or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the `ACLK` pin lower than the reported Fmax reached?
Data samples lost between Downstream core and this core. Inconsistent EOL and/or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and this core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the `ACLK` pin lower than the reported Fmax reached?

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

Other Interfaces

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

Table C-3: **Troubleshooting Third-Party Interfaces**

Symptom	Solution
Severe color distortion or color-swap when interfacing to third-party video IP.	Verify that the color component logical addressing on the AXI4-Stream TDATA signal is in according to *Data Interface* in Chapter 2. If misaligned: In HDL, break up the TDATA vector to constituent components and manually connect the slave and master interface sides.
Severe color distortion or color-swap when processing video written to external memory using the AXI-VDMA core.	Unless the particular software driver was developed with the AXI4-Stream TDATA signal color component assignments described in *Data Interface* in Chapter 2 in mind, there are no guarantees that the software correctly identifies bits corresponding to color components. Verify that the color component logical addressing TDATA is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned: In HDL, break up the TDATA vector to constituent components, and manually connect the slave and master interface sides.

Additional Resources

Xilinx Resources

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

http://Xilinx Support web page.

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

http://www.xilinx.com/company/terms.htm.

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:

1. *Vivado AXI Reference Guide* (UG1037)
2. *ISE to Vivado Design Suite Migration Guide* (UG911)
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
5. *Vivado Design Suite User Guide: Getting Started* (UG910)
6. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
7. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
8. AMBA AXI4 Interface Protocol

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/18/2015	4.0	Added UltraScale+ support.
10/01/2014	4.0	Removed Application Software Development appendix.
12/18/2013	4.0	Added UltraScale Architecture support.
10/02/2013	4.0	Synchronized document version with core version. Updated Constraints.
03/20/2013	4.0	Updated for core version v4.0. Updated Debugging appendix. Removed ISE chapters.
10/16/2012	3.1	Updated for core version. Updated tools to ISE 14.3 and Vivado 2012.3, added Vivado GUI update. Added Vivado test bench.
07/25/2012	3.0	Updated for core version. Added Vivado information.
04/24/2012	2.0	Updated core to v2.00.a and ISE Design Suite v14.1. Updated the C model parameters in Table 8-3. Replaced XSVI interfaces with AXI4-Stream interfaces. Added native support for EDK.
10/19/2011	1.0	Initial Xilinx release.

Notice of Disclaimer