LogiCORE IP Color Correction Matrix v6.0

Product Guide for Vivado Design Suite

PG001 December 18, 2013

XILINX®

XILINX

Table of Contents

Introduction

The Xilinx LogiCORE™ IP Color Correction Matrix core is a 3 x 3 programmable coefficient matrix multiplier with offset compensation. This core can be used for color correction operations such as adjusting white balance, color cast, brightness, or contrast in an image.

Features

- Programmable matrix coefficients
- Independent clipping and clamping control
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, 12 and 16-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(3)
Resources	See Table 2-1 through Table 2-4.
Provided with Core	
Documentation	Product Guide
Design Files	Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog (4)
Constraints File	XDC
Simulation Models	Encrypted RTL, VHDL or Verilog Structural, C model (4)
Supported Software Drivers (2)	Standalone
Tested Design Flows	
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. 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.
3. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG761 AXI Reference Guide [Ref 1].
4. C model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-CCM.htm

Overview

There are many variations that cause difficulties in accurately reproducing color in imaging systems. These can include:

- Spectral characteristics of the optics (lens, filters)
- Lighting source variations like daylight, fluorescent, or tungsten
- Characteristics of the color filters of the sensor

The Color Correction Matrix IP core provides a method for correcting the image data for these variations. This fundamental block operates on either YUV or RGB data, and processing is “real-time” as a pre-processing hardware block.

As an example, following one of the three color channels through an imaging system from the original light source to the processed image helps understand the functionality of this core.

The blue color channel is a combination of the blue photons from the scene, multiplied by the relative response of the blue filter, multiplied by the relative response of the silicon to blue photons. However, the filter and silicon responses might be quite different from the response of the human eye, so blue to the sensor is quite different from blue to a human being.

This difference can be corrected and made to more closely match the blue that is acceptable to human vision. The Color Correction Matrix core multiplies the pixel values by some coefficient to strengthen or weaken it, creating an effective gain. At the same time a mixture of green or red can be added to the blue channel. To express this processing mathematically, the new blue (Bc) is related to the old blue (B), red (R), and green (G) according to:

$$Bc = K1 \times R + K2 \times G + K3 \times B$$

where K1, K2, and K3 are the weights for each of the mix of red, green, and blue to the new blue.

Extending this concept, a standard 3 x 3 matrix multiplication can be applied to each of the color channels in parallel simultaneously. This is a matrix operation where the weights define a color-correction matrix. In typical applications, color-correction also contains offset compensation to ensure black [0,0,0] levels are achieved.

$$\begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}$$ *Equation 1-1*

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}$$ *Equation 1-2*

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}$$ *Equation 1-3*

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_1 \\ O_2 \\ O_3 \end{bmatrix}$$ *Equation 1-4*

As shown in the matrix operation, the input pixels are transformed to a set of corrected output pixels. This can be a very useful function configured as a static application; however, the programmability of the coefficients and offset values allows this function to adapt to changing lighting conditions based on a separate control loop.

Feature Summary

The Color Correction Matrix core offers a 3x3 matrix multiplication for a variety of color correction applications. The coefficient matrix is fully programmable and includes offset compensation, and clipping and clamping of the output is also definable.

The core offers a processor interface for changing the matrix coefficients during run-time.

Applications

- Pre-processing block for image sensors
- Post-processing core for image data adjustment

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, visit the Color Correction Matrix product web page.

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

Product Specification

Standards

The Color Correction Matrix core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 1] for additional information.

Performance

The following sections detail the performance characteristics of the Color Correction Matrix 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 device, using a different version of Xilinx tools and other factors. See Table 2-1 through Table 2-4 for device-specific information.

Latency

The processing latency of the core is eight CLK cycles.

Throughput

The Color Correction Matrix core outputs one sample per clock cycle.

Resource Utilization

The information presented in Table 2-1 through Table 2-4 is a guide to the resource utilization and maximum clock frequency of the Color Correction Matrix core for all input/output width combinations for Virtex-7, Kintex-7, Artix-7 FPGA families and Zynq-7000 devices using Vivado Design Suite. UltraScale™ results are expected to be similar to 7 series results. The Xtreme DSP Slice count is always 9, regardless of parameterization, and this core does not use any dedicated I/O or CLK resources. The design was tested with the AXI4-Lite interface, INTC_IF and the Debug Features disabled. By default, the maximum number of pixels per scan line was set to 1920, active pixels per scan line was set to 1920.

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

INPUT Width	OUTPUT Width	FFs	Slice	LUT as Logic	LUT FF Pairs	DSPs	Clock Frequency (MHz)
8	8	223	101	211	270	9	275
	10	241	92	217	254	9	268
	12	243	117	215	292	9	290
	16	279	114	230	311	9	234
10	8	237	102	214	269	9	234
	10	255	105	222	285	9	278
	12	257	108	220	299	9	252
	16	309	122	242	327	9	238
12	8	251	99	218	282	9	238
	10	269	101	225	288	9	242
	12	271	106	223	294	9	245
	16	323	119	246	339	9	275
16	8	279	105	227	303	9	242
	10	297	120	234	318	9	245
	12	315	114	241	335	9	275
	16	351	122	255	351	9	268

Speedfile: 7K70T-1 FBG484 (PRODUCTION 1.08 2013-01-28)

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

INPUT Width	OUTPUT Width	FFs	Slice	LUT as Logic	LUT FF Pairs	DSPs	Clock Frequency (MHz)
8	8	207	89	190	240	9	194
	10	225	93	204	259	9	194
	12	259	110	228	300	9	164
	16	279	119	225	325	9	164
10	8	221	96	200	254	9	190
	10	239	106	207	277	9	186
	12	273	107	233	299	9	186
	16	293	123	229	329	9	164
12	8	251	104	222	290	9	168
	10	269	107	229	296	9	156
	12	271	110	218	305	9	160
	16	323	116	250	331	9	190
16	8	263	99	213	289	9	186
	10	297	115	239	322	9	160
	12	315	127	245	344	9	190
	16	351	146	258	382	9	186

Speedfile: 7A100T-1 FGG484 (PRODUCTION 1.07 2013-01-28)

Table 2-3: **Virtex-7 FGPA Performance**

INPUT Width	OUTPUT Width	FFs	Slice	LUT as Logic	LUT FF Pairs	DSPs	Clock Frequency (MHz)
8	8	207	89	194	246	9	268
	10	241	99	217	274	9	272
	12	259	108	224	292	9	245
	16	279	108	229	317	9	245
10	8	221	100	206	270	9	278
	10	239	98	212	274	9	278
	12	257	104	220	288	9	245
	16	309	121	242	343	9	245
12	8	251	103	218	280	9	238
	10	269	106	225	293	9	234
	12	287	116	232	319	9	260
	16	323	115	246	321	9	278

Table 2-3: **Virtex-7 FGPA Performance *(Cont'd)***

INPUT Width	OUTPUT Width	FFs	Slice	LUT as Logic	LUT FF Pairs	DSPs	Clock Frequency (MHz)
16	8	279	102	227	307	9	242
	10	297	121	234	320	9	242
	12	315	117	241	332	9	286
	16	351	127	255	369	9	282

Speedfile: 7V585T-1 FFG1157 (ADVANCED 1.07b 2012-08-28)

Table 2-4: **Zynq-7000 Device Performance**

INPUT Width	OUTPUT Width	FFs	Slice	LUT as Logic	LUT FF Pairs	DSPs	Clock Frequency (MHz)
8	8	223	87	211	251	9	290
	10	225	95	208	270	9	290
	12	243	102	216	286	9	282
	16	279	111	229	312	9	242
10	8	221	103	205	257	9	286
	10	239	96	212	267	9	290
	12	257	106	220	293	9	234
	16	309	109	242	335	9	242
12	8	251	99	218	276	9	248
	10	253	104	217	285	9	234
	12	271	112	224	306	9	248
	16	323	115	246	332	9	282
16	8	263	103	218	291	9	290
	10	281	114	226	311	9	252
	12	315	124	241	343	9	290
	16	335	130	246	365	9	252

Speedfile: 7z030-1 ffg676 (PRELIMINARY 1.05 2013-01-28)

Core Interfaces and Register Space

Port Descriptions

The Color Correction Matrix 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 CCM core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the IRQ pin are present only when the core is configured through 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.

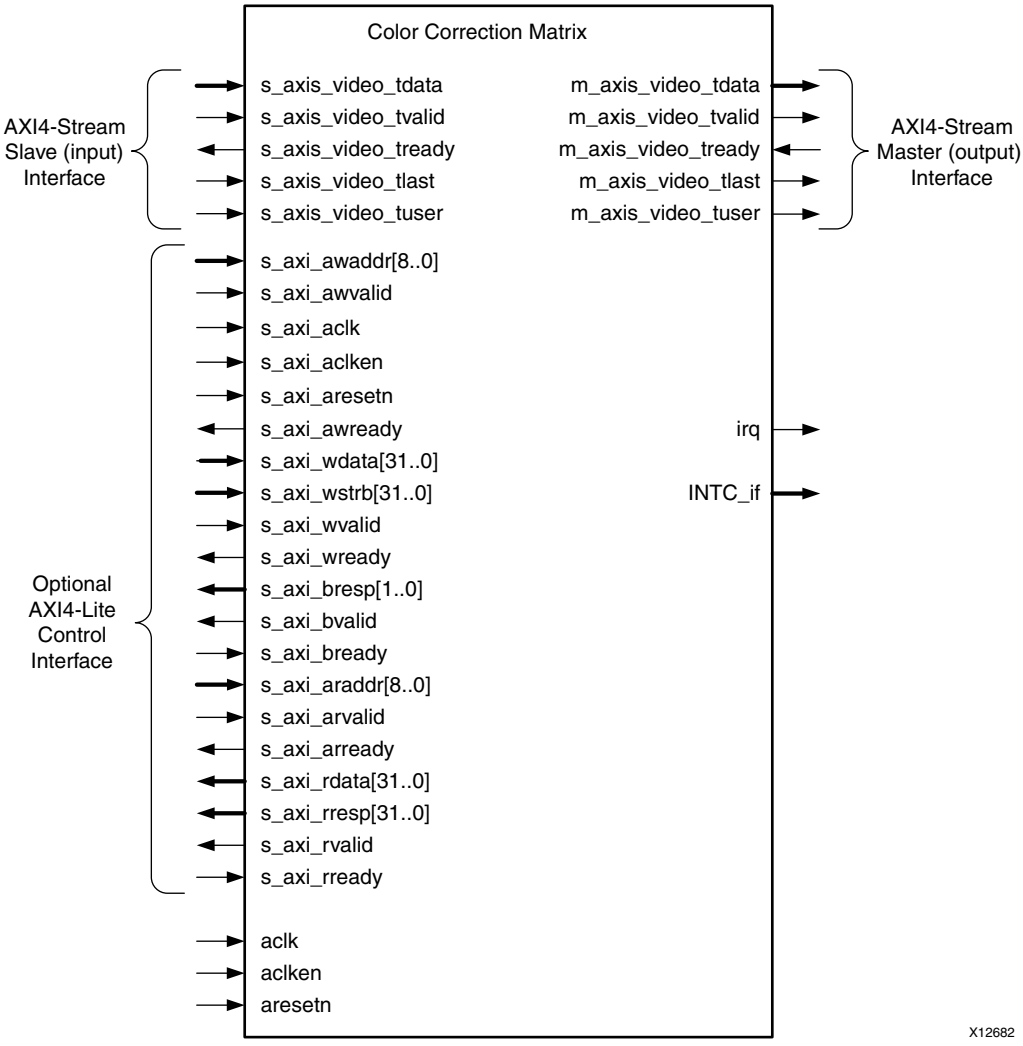

Figure 2-1: **CCM Core Top-Level Signaling Interface**

Common Interface Signals

Table 2-5 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-5: **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
INTC_IF	Out	6	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. See 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 CCM core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *Video IP: AXI Feature Adoption* section of the (UG761) *AXI Reference Guide* [Ref 1].

AXI4-Stream Signal Names and Descriptions

Table 2-6 describes the AXI4-Stream signal names and descriptions.

Table 2-6: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	24,32,40,48	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	24,32,40,48	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, 12, and 16 bit image data must be padded with zeros on the MSB to form an integer that is a multiple of 8 bits wide vector before connecting to `s_axis_video_tdata`. Padding does not affect the size of the core.

For example, when using RGB video, the RGB data on the CCM output `m_axis_video_tdata` is packed and padded to multiples of 8 bits as necessary, as shown for the RGB case in Figure 2-2. Zero padding the most significant bits is only necessary for 10, 12, and 16 bit wide data.

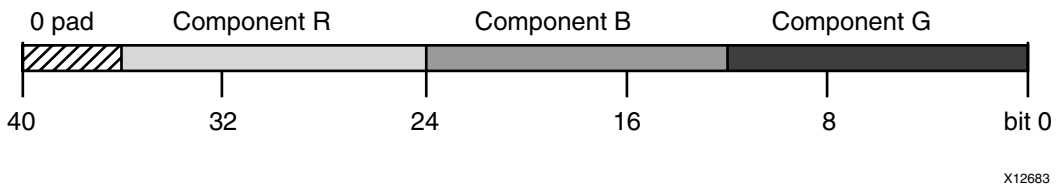

Figure 2-2: **RGB Data Encoding on m_axis_video_tdata**

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 2-3. 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 CCM core driving s_axis_video_tready) may change values until the transaction completes (s_axis_video_tready, s_axis_video_tvalid, ACLKEN 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.

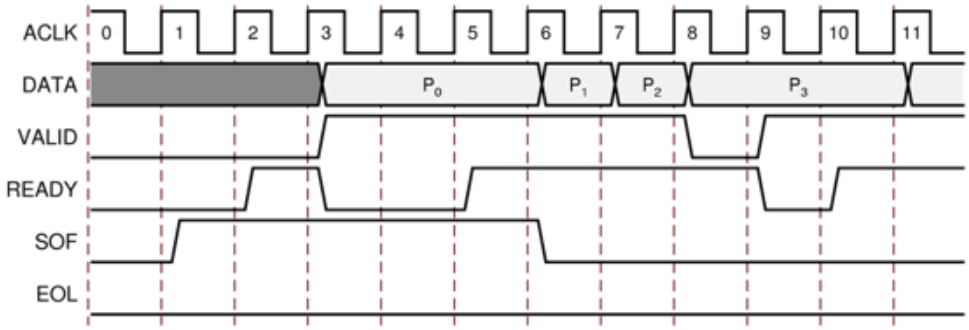

Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

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 CCM core 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_tuser0, s_axis_video_tuser0

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream TUSER0 signal, marks the first pixel of a video frame. The SOF pulse is 1 valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-3. 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 2-4.

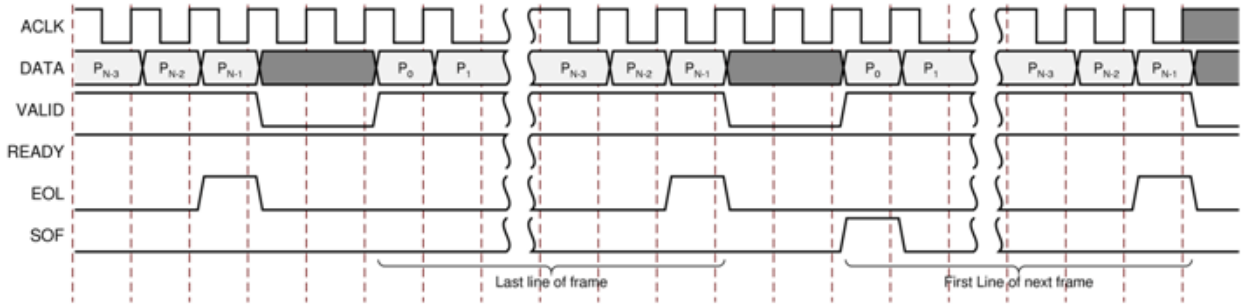

Figure 2-4: **Use of EOL and SOF Signals**

Control Interface

When configuring the core, you can 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 those who uses the CCM core in a single setup that does not need to change. In constant configuration, the image resolution, matrix coefficients, offsets and clip/clamp values are hard coded into the core via the CCM core 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 MicroBlaze.

The CCM core can be controlled with the AXI4-Lite interface using read and write transactions to the CCM register space.

Table 2-7: **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 to control-, timing-, and core specific registers. The CCM core uses only one timing related register, ACTIVE_SIZE (0x0020), which allows specifying the input frame dimensions. The core has 14 core-specific registers for setting the matrix coefficients, the offsets and clip/clamp values.

Table 2-8: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	N	Power-on-Reset : 0x0	Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 4: BYPASS(1) 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

Table 2-8: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to STATUS bits
0x0010	VERSION	R	N/A	0x0500a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	SYSDEBUG0	R	N/A	0	0-31: Frame Throughput monitor(1)
0x0018	SYSDEBUG1	R	N/A	0	0-31: Line Throughput monitor(1)
0x001C	SYSDEBUG2	R	N/A	0	0-31: Pixel Throughput monitor(1)
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame
0x0100	K11	R/W	Yes	Specified via GUI	17:0 - real numbers in the [-8 : 8] range, multiplied by 16384
0x0104	K12	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0108	K13	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x010C	K21	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0110	K22	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0114	K23	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0118	K31	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384

Table 2-8: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x011C	K32	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0120	K33	R/W	Yes	Specified via GUI	17-0: Matrix Coefficient Real numbers in the [-8 : 8] range, multiplied by 16384
0x0124	ROFFSET	R/W	Yes	Specified via GUI	16-0: Red Offset [-2Out_Data_Width : 2Out_Data_Width-1]
0x0128	GOFFSET	R/W	Yes	Specified via GUI	16-0: Green Offset [-2Out_Data_Width : 2Out_Data_Width-1]
0x012C	BOFFSET	R/W	Yes	Specified via GUI	16-0: Blue Offset [-2Out_Data_Width : 2Out_Data_Width-1]
0x0130	CLIP	R/W	Yes	Specified via GUI	15-0: Maximum Output [0 : 2Out_Data_Width-1]
0x0134	CLAMP	R/W	Yes	Specified via GUI	15-0: Minimum Output [0 : 2Out_Data_Width-1]

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 CCM core K*, *OFFSET, CLIP and CLAMP 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 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 CCM core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. Refer to Debug Tools in Appendix C 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, 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 Debug Tools in Appendix C 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 causes 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 resets the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

STATUS (0x0004) Register

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

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

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

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

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

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

IRQ_ENABLE (0x000C) Register

Any bits of the STATUS register can generate a host-processor interrupt request via the IRQ pin. The Interrupt Enable register facilitates selecting which bits of STATUS register asserts 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. Refer to Table 2-8 for more information.

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 Debug Tools in Appendix C 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 Debug Tools in Appendix C 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 Debug Tools in Appendix C for more information.

ACTIVE_SIZE (0x0020) Register

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

K11 - K33 (0x0100 - 0x0120) Registers

The K11 - K33 registers hold the matrix coefficients shown in Equation 1-1 and Equation 1-2. The coefficients are presented in 18.14 fixed point format. The 18-bit signed integer values are equivalent to real numbers in the [-8 : 8] range, multiplied by 16384.

ROFFSET (0x0124) Register

The ROFFSET register holds the O1 offset value shown in Equation1-1 and Equation 1-2. The offset value has a width of the output data width plus 1. It is a signed integer with a range of [-2Out_Data_Width : 2Out_Data_Width-1].

GOFFSET (0x0128) Register

The GOFFSET register holds the O2 offset value shown in Equation1-1 and Equation 1-2. The offset value has a width of the output data width plus 1. It is a signed integer with a range of [-2Out_Data_Width : 2Out_Data_Width-1].

BOFFSET (0x012C) Register

The BOFFSET register holds the O3 offset value shown in Equation1-1 and Equation 1-2. The offset value has a width of the output data width plus 1. It is a signed integer with a range of `[-2Out_Data_Width : 2Out_Data_Width-1]`.

Clip (0x0130) Register

The Clip register holds the maximum output data value. Output values greater than this value is replaced with this value. The Clip value has the same width as the output data width. It is an unsigned integer with a range of `[0 : 2Out_Data_Width-1]`.

Clamp (0x0134) Register

The Clamp register holds the minimum output data value. Output values smaller than this value is replaced with this value. The Clamp value has the same width as the output data width. It is an unsigned integer with a range of `[0 : 2Out_Data_Width-1]`.

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 CCM core detects AXI4-Stream framing errors, as well as the beginning and the end of frame processing.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (`IRQ`) is present. Events associated with bits of the `STATUS` register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (`IRQ_ENABLE`) are set. After 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 application can still benefit from the core signaling error and status events. By selecting **Enable INTC Port**, the core generates the optional `INTC_IF` port. This vector of signals gives parallel access to the individual interrupt sources, as seen in Table 2-9.

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

Table 2-9: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	Reserved
3	Reserved
4	Video over AXI4-Stream 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, you can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

Designing with the Core

General Design Guidelines

The Color Correction Matrix core is a 3x3 matrix multiplication with an additional offset.

The output values are:

$$Rc = K11 \times R + K12 \times G + K13 \times B + O1$$

$$Gc = K21 \times R + K22 \times G + K23 \times B + O2$$

$$Bc = K31 \times R + K32 \times G + K33 \times B + O3$$

In cases where the Input Data Width does not equal the Output Data Width, the data is scaled up or down accordingly. For example, if the Input Data Width =8, and Output Data Width=12, then the core scales the data up by a factor of 4. Meaning, with an identity matrix, an input of 1 gives an output of 4.

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.

RECOMMENDED: *The CCM core is designed to be used in conjunction with the Video In to AXI4-Stream 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 In to AXI4-Stream core formats the input video to the AXI4-Stream interface.

Typically, the Color Correction Matrix core is part of an Image Sensor Pipeline (ISP) System, as shown in Figure 3-1.

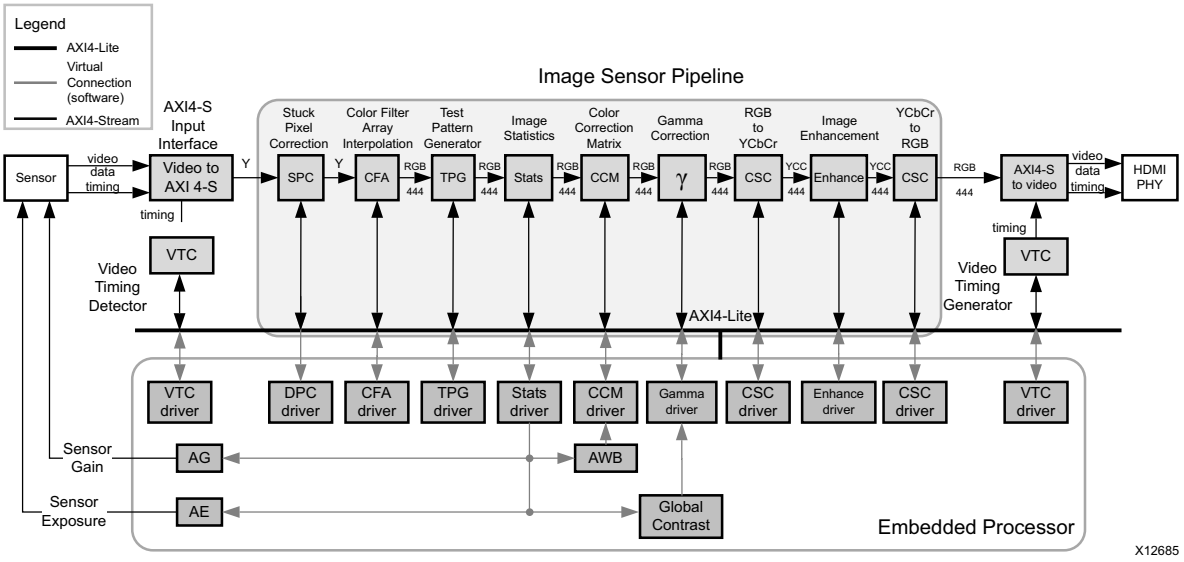

Figure 3-1: **Image Sensor Pipeline System with Color Correction Matrix Core**

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 3-2.

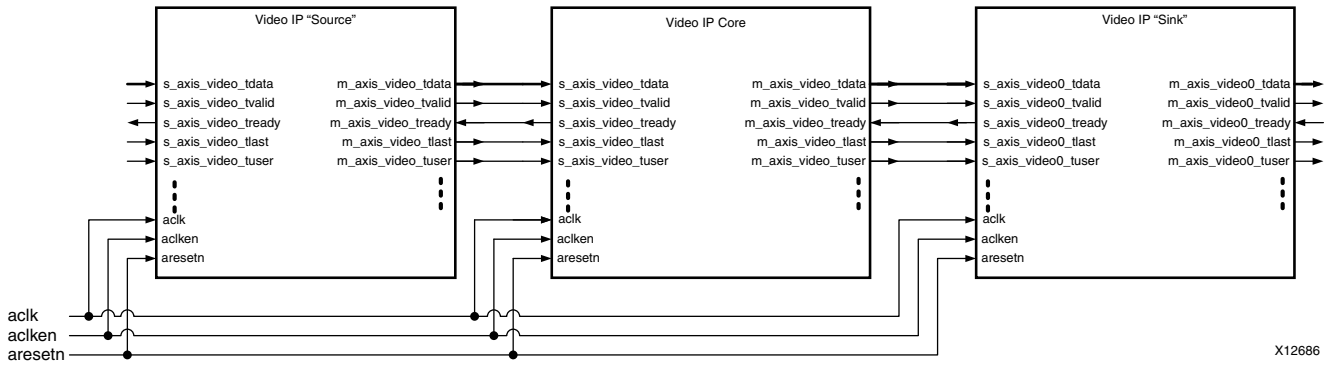

Figure 3-2: **Example of ACLK Routing in an ISP Processing Pipeline**

S_AXI_ACLK

The AXI4-Lite interface uses the A_AXI_ACLK pin as its clock source. The ACLK pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The Color Correction Matrix 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 Color Correction Matrix 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 Color Correction Matrix 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 Color Correction Matrix IP core must be configured for the actual video frame size to operate properly. To gather the frame size information from the video, 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 gathers the video 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 CCM, with the appropriate image dimensions.

If the target system uses only one configuration of the CCM (for example, does not ever need to be reprogrammed), 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 do 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 sends an error message 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 responds with **10** on the S_AXI_RRESP bus. Similarly, if a core register write

does not complete, the AXI4-Lite write transaction responds 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 the system needs to be reinitialized, pipelined video IP cores should be disabled and reset from system output towards the system input, and programmed and enabled from system input to system output. STATUS register bits allow system processors to identify the processing states of individual constituent cores, and successively disable a pipeline as one core after another is finished processing the last frame of data.

Error Propagation and Recovery

Parameterization and/or configuration registers define the dimensions of the video frames that video IP should process. Starting from a known state and based on these configuration settings, the IP can predict the expected beginning of 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, immediately terminating the previous frame. When SOF is detected late, the output SOF signal is generated according to the programmed values. Extra lines and 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, immediately terminating the previous line. 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

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* (UG810) [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 Color Correction Matrix core is easily configured to meet developers' specific needs before instantiation through the Vivado design tools Graphical User Interface (GUI). When you start to build the Color Correction Matrix core within the system, you are asked to set various parameters. This section provides a quick reference to the windows and parameters that can be configured at compile time.

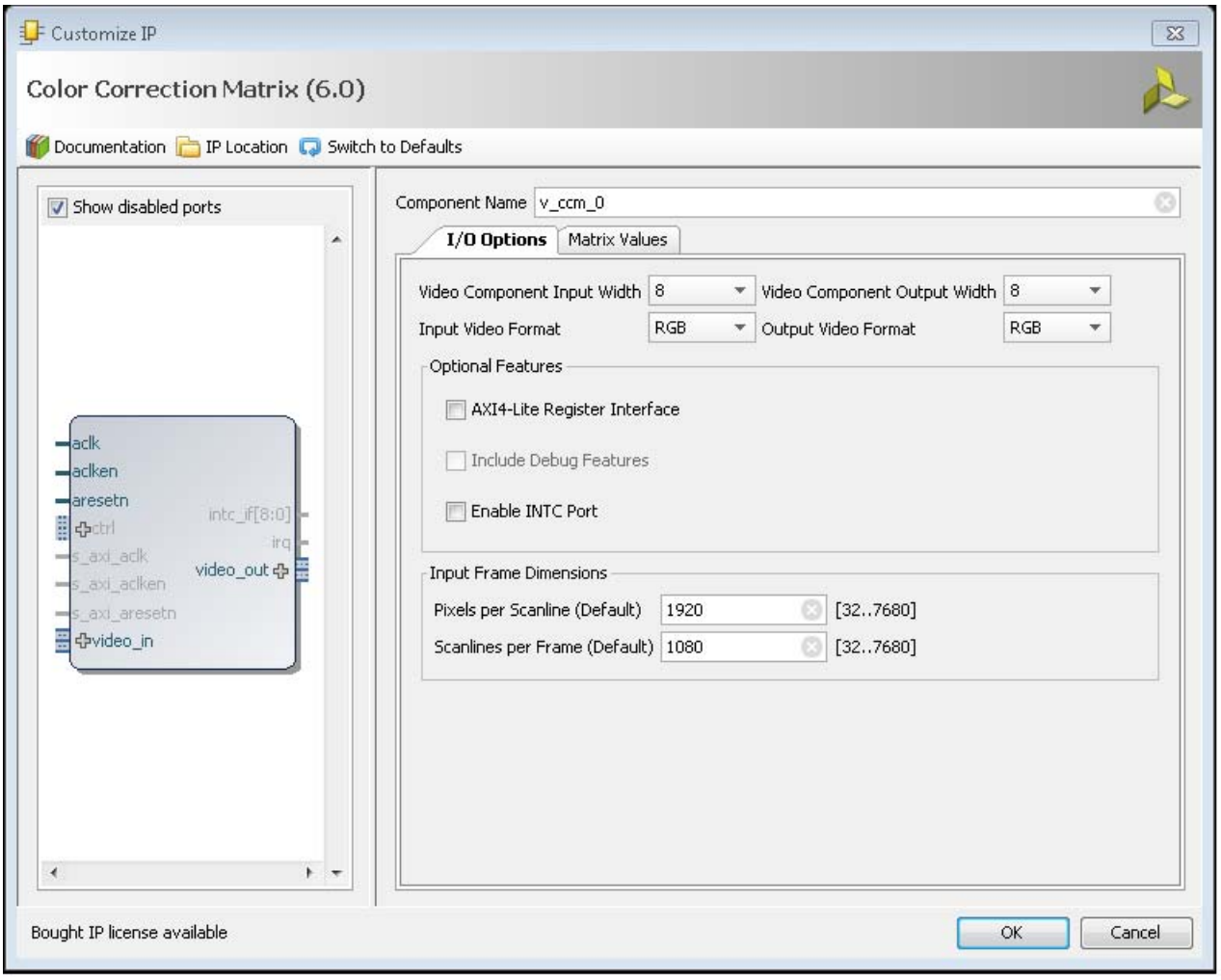

Figure 4-1: **IP Catalog Screen 1**

The first screen (Figure 4-1) shows a representation of the IP symbol on the left side, and the parameters on the right, 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 "_".

IMPORTANT: *The name v_ccm_v6_0 cannot be used as a component name.*

- **Input Data Width**: Specifies the bit width of the input color channel for each component.
  In cases where the Input Data Width does not equal the Output Data Width, the data is scaled up or down accordingly.
- **Output Data Width**: Specifies the bit width of the output color channel for each component.

In cases where the Input Data Width does not equal the Output Data Width, the data is scaled up or down accordingly.

- **Input Video Format**: Specifies the format of the input video. Valid selections are RGB and YUV 4:4:4.
- **Output Video Format**: Specifies the format of the output video. Valid selections are RGB and YUV 4:4:4.
- **Optional Features**:
  - **AXI4-Lite Register Interface**: When selected, the core is 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 Debugging Features**: When selected, the core is generated with debugging features, which simplify system design, testing and debugging. For more information, refer to Debug Tools in Appendix C.

IMPORTANT: *Debugging features are only available when the AXI4-Lite Register Interface is selected.*

  - **INTC Interface**: When selected, the core generates 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.

The second screen (Figure 4-2) also shows a representation of the IP symbol on the left side, but has a second set of parameters on the right, as described in this section.

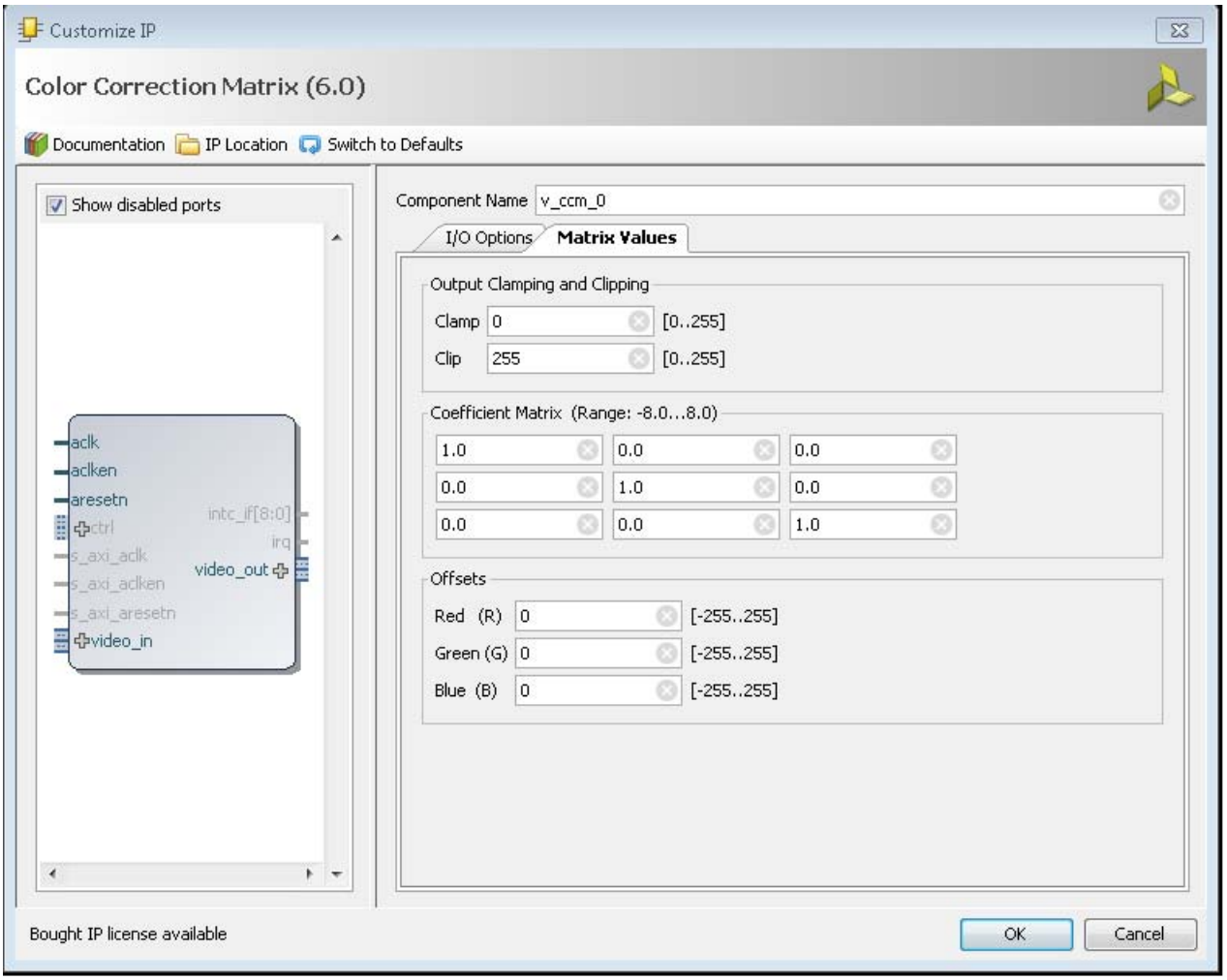

Figure 4-2: **IP Catalog Screen 2**

- **Frame Size**:
  - **Number of Columns**: When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the 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 is to process.
  - **Number of Lines**: When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the 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 is to process.
- **Output Clamping and Clipping**:
  - **Clamp**: Sets the minimum value of the output data. When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the GUI as the

default value for the Clamp register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the minimum value of the output data for the generated core instance.

- **Clip**: Sets the maximum value of the output data. When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the GUI as the default value for the Clip register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the maximum value of the output data for the generated core instance.

- **Coefficient Matrix**: Enter the floating-point coefficients ranging from [-8, 8] (K in Equation 1-1) by specifying the 18 bit coefficients with 14 fractional bits of the coefficient matrix. The entered values are the default used to initialize the core, and the values used when the core is reset. Enter the real valued coefficients as floating-point decimal values in the range [-8.0, 8.0] (K in Equation 1-1). When the core is generated, the floating-point decimal value is converted to an 18-bit vector with 14 fractional bits, which are used internally to the core.
- **Offsets**: Enter the offset coefficients (O in Equation 1-1). These signed coefficients have the same bit width as the output. Enter the offset values (O in Equation 1-1). These signed integer values must be in the range $[-2^{Out\_Data\_Width}, 2^{Out\_Data\_Width}-1]$, and are 1-bit wider than the Output Data Width.

Output Generation

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

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

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

Installation and Directory Structure

This chapter contains information for installing the Color Correction Matrix C model, and describes the file contents and directory structure.

The C model is available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-CCM.htm.

Software Requirements

The Color Correction Matrix v6.0 C models were compiled and tested with the following software versions.

Table 8-1: **Supported Systems and Software Requirements**

Platform	C-Compiler
Linux 32-bit and 64-bit	GCC 4.1.1
Windows 32-bit and 64-bit	Microsoft Visual Studio 2008 (Visual C++ 8.0)

Installation

The installation of the C model requires updates to the PATH variable, as described below.

Linux

Ensure that the directory in which the `libIp_v_ccm_v6_0_bitacc_cmodel.so` file is located in your $LD_LIBRARY_PATH environment variable.

C Model File Contents

Unzipping the v_ccm_v6_0_bitacc_model.zip file creates the following directory structures and files which are described in Table 8-2.

Table 8-2: **C Model Files**

File	Description
/lin	Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Linux Platforms
libIp_v_ccm_v6_0_bitacc_cmodel.lib	Color Correction Matrix v6.0 model shared object library (Linux platforms only)
run_bitacc_cmodel	Pre-compiled bit accurate executable for simulation on 32-bit Linux Platforms
/lin64	Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Linux Platforms
libIp_v_ccm_v6_0_bitacc_cmodel.lib	Color Correction Matrix v6.0 model shared object library (Linux platforms only)
run_bitacc_cmodel	Pre-compiled bit accurate executable for simulation on 32-bit Linux Platforms
/nt	Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Windows Platforms
libIp_v_ccm_v6_0_bitacc_cmodel.lib	Pre-compiled library file for win32 compilation
run_bitacc_cmodel.exe	Pre-compiled bit accurate executable for simulation on 32-bit Windows Platforms
/nt64	Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Windows Platforms
libIp_v_ccm_v6_0_bitacc_cmodel.lib	Pre-compiled library file for win32 compilation
run_bitacc_cmodel.exe	Pre-compiled bit accurate executable for simulation on 64-bit Windows Platforms
README.txt	Release notes
pg001-v-ccm.pdf	*Color Correction Matrix Product Guide*
v_ccm_v6_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I/O functions
video_utils.h	Header file declaring the generalized image / video container type, I/O and support functions.
Kodim19_128x192.bmp	128x192 sample test image of the Lighthouse image from the True-color Kodak test images
run_bittacc_cmodel.c	Example code calling the C model

Using the C Model

The bit accurate C model is accessed through a set of functions and data structures that are declared in the `v_ccm_v6_0_bitacc_cmodel.h` file.

Before using the model, the structures holding the inputs, generics, and output of the CCM instance must be defined:

```
struct  xilinx_ip_v_ccm_v6_0_generics ccm_generics;
struct  xilinx_ip_v_ccm_v6_0_inputs   ccm_inputs;
struct  xilinx_ip_v_ccm_v6_0_outputs  ccm_outputs;
```

The declaration of these structures is in the `v_ccm_v6_0_bitacc_cmodel.h` file.

Table 8-3 lists the generic parameters taken by the CCM v6.0 IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the GUI.

Table 8-3: **Model Generic Parameters and Default Values**

Generic Variable	Type	Default Value	Range	Description
IWIDTH	int	8	8,10,12, 16	Input data width
OWIDTH	int	8	8,10,12, 16	Output width
INPUT_VIDEO_FORMAT	int	2	1, 2	Input Video Format 1=YUV 4:4:4 2=RGB
OUTPUT_VIDEO_FORMAT	int	2	1 2	Output Video Format 1=YUV 4:4:4 2=RGB

Calling `xilinx_ip_v_ccm_v6_0_get_default_generics(&ccm_generics)` initializes the generics structure with the CCM GUI defaults, listed in Table 8-3.

Coefficients, offsets, clipping and clamping values can also be set dynamically through the pCore and General Purpose Processor interfaces. Consequently, these values are passed as inputs to the core, along with the actual test image, or video sequence (Table 8-4).

Table 8-4: **Core Generic Parameters and Default Values**

Input Variable	Type	Default Value	Range[(1)]	Description
video_in	video_struct	null	N/A	Container to hold input image or video data.[(3)]
coeffs	double[3][3]	identity[(2)]	[-8 to 8]	3x3 matrix of floating point numbers
offsets	double[3]	zeros[(2)]	-2^{OWIDTH} *to* $2^{OWIDTH}-1$	Offsets applied to the output color channels

Table 8-4: **Core Generic Parameters and Default Values**

Input Variable	Type	Default Value	Range(1)	Description
CLAMP	int	0	0 *to* $2^{OWIDTH} - 1$	Clamping value for outputs
CLIP	int	$2^{OWIDTH} - 1$	0 *to* $2^{OWIDTH} - 1$	Clipping value for outputs

1. OWIDTH is the output data width of each color component
2. For a detailed description of inputs and other generic parameters, see Core Interfaces and Register Space.
3. For the description of the input structure, see Initializing the CCM Input Video Structure.

The structure `ccm_inputs` defines the values of run time parameters and the actual input image. Calling

```
xilinx_ip_v_ccm_v6_0_get_default_inputs(&ccm_generics, &ccm_inputs)
```

initializes the input structure with the CCM GUI default values (see Table 8-4).

The `video_in` variable is not initialized because the initialization depends on the actual test image to be simulated. Initializing the CCM Input Video Structure describes the initialization of the `video_in` structure.

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

```
int xilinx_ip_v_ccm_v6_0_bitacc_simulate(
struct xilinx_ip_v_ccm_v6_0_generics* generics,
struct xilinx_ip_v_ccm_v6_0_inputs*   inputs,
struct xilinx_ip_v_ccm_v6_0_outputs*  outputs).
```

Results are included in the outputs structure, which contains only one member, type `video_struct`. After the outputs are evaluated and saved, dynamically allocated memory for input and output video structures must be released by calling this function:

```
void xilinx_ip_v_ccm_v6_0_destroy(
struct xilinx_ip_v_ccm_v6_0_inputs *input,
struct xilinx_ip_v_ccm_v6_0_outputs *output).
```

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

CCM Input and Output Video Structure

Input images or video streams can be provided to the CCM v6.0 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: **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. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can have different dimensions.
cols	Number of columns per frame. Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream. However different planes, such as y, u and v can 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 8-6.
data	Set of five pointers to three dimensional arrays containing image plane data. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

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

Mode(1)	Planes	Video Representation
FORMAT_MONO	1	Monochrome – Luminance only
FORMAT_RGB	3	RGB image/video data
FORMAT_C444	3	444 YUV, or YCrCb image/video data
FORMAT_C422	3	422 format YUV video, (u, v chrominance channels horizontally sub-sampled)
FORMAT_C420	3	420 format YUV video, (u, v sub-sampled both horizontally and vertically)
FORMAT_MONO_M	3	Monochrome (Luminance) video with Motion
FORMAT_RGBA	4	RGB image/video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with Motion
FORMAT_C422_M	5	422 YUV video with Motion
FORMAT_C444_M	5	444 YUV video with Motion
FORMAT_RGBM	5	RGB video with Motion

1. The Color Correction Matrix core supports Modes FORMAT_RGB and FORMAT_C444.

Initializing the CCM 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 `bmp_util.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O.

Bitmap Image Files

The header `bmp_utils.h` declares functions that help access files in Windows Bitmap format (http://en.wikipedia.org/wiki/BMP_file_format). However, this format limits color depth to a maximum of 8-bits per pixel, and operates on images with three planes (R,G,B). Consequently, the following functions operate on arguments type `rgb8_video_struct`, which is defined in `rgb_utils.h`. Also, both functions support only true-color, non-indexed formats with 24-bits per pixel.

```
int write_bmp(FILE *outfile, struct rgb8_video_struct *rgb8_video);
int read_bmp(FILE *infile, struct rgb8_video_struct *rgb8_video);
```

Exchanging data between `rgb8_video_struct` and general `video_struct` type frames/videos is facilitated by these functions:

```
int copy_rgb8_to_video(struct rgb8_video_struct* rgb8_in,
                       struct video_struct* video_out );
int copy_video_to_rgb8(struct video_struct* video_in,
                       struct rgb8_video_struct* rgb8_out );
```

IMPORTANT: *All image/video manipulation utility functions expect both input and output structures to be initialized; for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data or r, g, b) 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 issue an error if the output container size does not match. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.*

Binary Image/Video Files

The `video_utils.h` header file declares functions that help load and save generalized video files in raw, uncompressed format.

```
int read_video( FILE* infile,  struct video_struct* in_video);
int write_video(FILE* outfile, struct video_struct* out_video);
```

These functions serialize the video_struct structure. The corresponding file contains a small, plain text header defining, "Mode", "Frames", "Rows", "Columns", and "Bits per Pixel". The plain text header is followed by binary data, 16-bits per component in scan line continuous format. Subsequent frames contain as many component planes as defined by the video mode value selected. Also, the size (rows, columns) of component planes can differ within each frame as defined by the actual video mode selected.

Working with Video_struct Containers

The `video_utils.h` header file defines 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);
```

The `video_planes_per_mode` function returns the number of component planes defined by the mode variable, as described in Table 8-6. The `video_rows_per_plane` and `video_cols_per_plane` functions return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in the `in_video` variable:

```
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 to demonstrate the steps required to run the model. After following the compilation instructions, run the example executable. The executable takes the path/name of the input file and the path of the output as parameters. If invoked with insufficient parameters, this help message is issued:

```
Usage: run_bitacc_cmodel in_file out_path
in_file     : path/name of the input file (YUV file)
out_path : path to to the output files
```

During successful execution, two directories is created at the location specified by the out_path command line parameter. The first directory is the "expected" directory. This directory contains a BMP file that corresponds to the output of the first frame that was processed. This directory also contains a txt file called `golden_1.txt`. This txt file contains the output of the model in a format that can be directly used with the demonstration test bench. The second directory that is created is the "stimuli" directory. This directory contains a txt file called `stimuli_1.txt`. This txt file contains the input of the model in a format that can be directly used with the demonstration test bench.

Compiling with the CCM C Model

Linux (32- and 64-bit)

To compile the example code, first ensure that the directory in which the file `libIp_v_ccm_v6_0_bitacc_cmodel.so` is located is present in your `$LD_LIBRARY_PATH` environment variable. These shared libraries are referenced during the compilation and linking process. Then cd into the directory where the header files, library files and `run_bitacc_cmodel.c` were unpacked. The libraries and header files are referenced during the compilation and linking process.

Place the header file and C source file in a single directory. Then in that directory, compile using the GNU C Compiler:

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c -o run_bitacc_cmodel -L.
-lIp_v_ccm_v6_0_bitacc_cmodel -Wl,-rpath,.
```

```
gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../gen_stim.c -o run_bitacc_cmodel -L.
-lIp_v_ccm_v6_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32- and 64-bit)

Precompiled library `v_ccm_v6_0_bitacc_cmodel.dll`, and top level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. Here an example is presented using Microsoft Visual Studio.

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

- The `llibIpv_ccm_v6_0_bitacc_cmodel.dll` file to the "Resource Files" folder of the project
- The `run_bitacc_cmodel.c` and `gen_stim.c` files to the "Source Files" folder of the project
- The `v_ccm_v6_0_bitacc_cmodel.h` header files to "Header Files" folder of the project (optional)

After the project has been created and populated, it needs to be compiled and linked (built) to create a win32 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.

Detailed Example Design

No example design is available at the time for the LogiCORE IP Color Correction Matrix v6.0 core.

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 Color Correction Matrix 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 Color Correction Matrix 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 Color Correction Matrix 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) AXI4 Stream interface can be configured in either RGB or YUV 4:4:4 format. This interface can work directly with any video core that produces the same format. The core master (output) interface can be configured in either RGB or YUV 4:4:4 format. It can work directly with any video core the consumes the same format. The AXI4-Stream interfaces need to be compliant to the AXI4-Stream Video Protocol as described in the *Video IP: AXI Feature Adoption* section of the (UG761) *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 v3.0 to v4.00.a of the CCM core the following significant changes took place:

- XSVI interfaces were replaced by AXI4-Stream interfaces.
- Since AXI4-Stream does not carry video timing data, the timing detector and timing generator modules were trimmed.

- The pCore, General Purpose Processor and Transparent modes became obsolete and were removed.
- Native support for EDK have been added - the CCM core appears in the EDK IP Catalog.
- Debugging features have been added.
- The AXI4-Lite control interface register map is standardized between Xilinx video cores.

From v4.00.a to v5.00.a of the CCM core, the following changes took place:

- The core originally had `aclk`, `aclken` and `aresetn` to control both the Video over AXI4-Stream and AXI4-Lite interfaces. Separate clock, clock enable and reset pins now control the Video over AXI4-Stream and the AXI4-Lite interfaces with clock domain crossing logic added to the core to handle the dissimilar clock domains between the AXI4-Lite and Video over AXI4-Stream domains.

From v5.00.a to v5.01a of the CCM core, the following changes took place:

- Increased the Coefficient range from [-4 : 4] to [-8 : 8].
- Added demonstration test bench for Vivado tools.
- Added support for Vivado 2012.03 and ISE and EDK 14.3 tools.

From v5.01.a to v6.0 of the CCM core, the following changes took place:

- Corrected the Matrix conversion mapping order for RGB to YUV and YUV to RGB conversion. Now the mapping order is uniform across all the four conversions.
- Removed ISE support.

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 Color Correction Matrix, the Xilinx Support web page (www.xilinx.com/support) 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 Color Correction Matrix. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). 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 Color Correction Matrix Core

AR 54519

Contacting Technical Support

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

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

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 Color Correction Matrix 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 Color Correction Matrix. 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 Color Correction Matrix 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.

Application Software Development

Programmer Guide

The software API is provided to allow easy access to the CCM AXI4-Lite registers defined in Table 2-5. To utilize the API functions, the following two header files must be included in the user C code:

```
#include "ccm.h"
#include "xparameters.h"
```

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

For examples on API function calls and integration into a user application, the drivers subdirectory of the pCore contains a file, `example.c`, in the `ccm_v6_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the CCM pCore API.

Table D-1: **CCM Driver Function Definitions**

Function Name and Parameterization	Description
CCM_Enable (uint32 BaseAddress)	Enables a CCM instance.
CCM_Disable (uint32 BaseAddress)	Disables a CCM instance.
CCM_Reset (uint32 BaseAddress)	Immediately resets a CCM instance. The core stays in reset until the RESET flag is cleared.
CCM_ClearReset (uint32 BaseAddress)	Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
CCM_FSync_Reset (uint32 BaseAddress)	Resets a CCM instance on the next SOF signal.
CCM_ReadReg (uint32 BaseAddress, uint32 RegOffset)	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-8).

Table D-1: **CCM Driver Function Definitions *(Cont'd)***

Function Name and Parameterization	Description
CCM_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)	Write the register selected by RegOffset (defined in Table 2-8. Data is the 32-bit value to write to the register.
CCM_RegUpdateEnable (uint32 BaseAddress)	Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.
CCM_RegUpdateDisable (uint32 BaseAddress)	Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.

Software Reset

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

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

IMPORTANT: *Calling CCM_FSync_Reset() does not guarantee prompt, or real-time reseting of the core. If the AXI4-Stream communication is halted mid frame, the core does not reset until the upstream core finishes sending the current frame or starts a new frame.*

Double Buffering

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

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the `RegUpdateEnable()` and `RegUpdateDisable()` functions allows synchronous commitment of register changes. The CCM core starts using the updated `ACTIVE_SIZE` and core-specific values only if the

REGUPDATE flag of the CONTROL register is set (1), after the next Start-Of-Frame signal (s_axis_video_tuser0) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 2-8 has a constant that is defined in ccm.h which is set to the offset for that register listed in Table D-2. It is recommended that the application software uses the predefined register names instead of register values when accessing core registers, so future updates to the CCM drivers which may change register locations does not affect the application dependent on the CCM driver.

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

Constant Name Definition	Value	Target Register
CCM_CONTROL	0x0000	CONTROL
CCM_STATUS	0x0004	STATUS
CCM_ERROR	0x0008	ERROR
CCM_IRQ_ENABLE	0x000C	IRQ_ENABLE
CCM_VERSION	0x0010	VERSION
CCM_SYSDEBUG0	0x0014	SYSDEBUG0
CCM_SYSDEBUG1	0x0018	SYSDEBUG1
CCM_SYSDEBUG2	0x001C	SYSDEBUG2
CCM_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
CCM_K11	0x0100	MATRIX COEFFICIENT
CCM_K12	0x0104	MATRIX COEFFICIENT
CCM_K13	0x0108	MATRIX COEFFICIENT
CCM_K21	0x010C	MATRIX COEFFICIENT
CCM_K22	0x0110	MATRIX COEFFICIENT
CCM_K23	0x0114	MATRIX COEFFICIENT
CCM_K31	0x0118	MATRIX COEFFICIENT
CCM_K32	0x011C	MATRIX COEFFICIENT
CCM_K33	0x0120	MATRIX COEFFICIENT
CCM_ROFFSET	0x0124	RED OFFSET
CCM_GOFFSET	0x0128	GREEN OFFSET
CCM_BOFFSET	0x012C	BLUE OFFSET
CCM_CLIP	0x0130	CLIP
CCM_CLAMP	0x0134	CLAMP

Additional Resources

Xilinx Resources

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

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

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

http://www.xilinx.com/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. *AXI Reference Guide* (UG761)
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)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release of Product Guide, replacing DS720 and UG830.
4/24/2012	2.0	Updated for core version. Added Zynq-7000 devices, added AXI4-Stream interfaces, deprecated GPP interface. Added support for ISE v14.3 and Vivado v2012.3
07/25/2012	3.0	Updated for core version. Added Vivado information.
10/16/2012	3.1	Updated for core version. Added Vivado test bench.
03/20/2013	3.2	Updated for core version v6.0. Removed ISE chapters.
10/02/2013	6.0	Updated to synch doc version and core version. Updated Constraints chapter. Updated Test Bench chapter.
12/18/2013	6.0	Added UltraScale Architecture support.

Notice of Disclaimer