LogiCORE IP Color Filter Array Interpolation v6.00.a

Product Guide

PG002 July 25, 2012

XILINX®

XILINX

Table of Contents

SECTION I: SUMMARY

Introduction

The Xilinx LogiCORE™ IP Color Filter Array Interpolation core provides an optimized hardware block that reconstructs sub-sampled color data for images captured by a Bayer Color Filter Array image sensor. The color filter array overlaid on the silicon substrate enables CMOS or CCD image sensors to measure local light intensities that correspond to different wavelengths. However, the sensor measures the intensity of one principal color at any location. The Color Filter Array Interpolation LogiCORE IP provides an efficient and low-footprint solution to interpolate the missing color components for every pixel.

Features

- RGB and CMY Bayer image sensor support
- 5x5 interpolation aperture
- Low-footprint, high quality interpolation
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, and 12-bits per color component input and output
- Built-in, optional bypass and test-pattern generator mode
- Built-in, optional throughput monitors
- Supports spatial resolutions from 128x128 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)	Zynq-7000 (2), Artix-7, Virtex®-7, Kintex®-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Lite, AXI4-Stream(3)
Resources	See Table 2-1 through Table 2-8.
Provided with Core	
Design Files	ISE: NGC netlist, Encrypted HDL Vivado: Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog (4)
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural, C-Model (4)
Supported Software Drivers	Not Applicable
Tested Design Flows (6)	
Design Entry Tools	Integrated Software Environment (ISE), Vivado™ Design Suite (7), Platform Studio (XPS)
Simulation(5)	Mentor Graphics ModelSim, Xilinx® ISim
Synthesis Tools	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Supported in ISE Design Suite implementations only.
3. Video protocol as defined in the *Video IP: AXI Feature Adoption* section of UG761 AXI Reference Guide.
4. HDL test bench and C-Model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-CFA.htm
5. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.
6. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
7. Supports only 7 series devices.

Overview

Images captured by a CMOS/CCD image sensor are monochrome in nature. To generate a color image, three primary colors - typically Red, Green, and Blue - are required for each pixel. Before the invention of color image sensors, the color image was created by superimposing three identical images with three different primary colors. These images were captured by placing different color filters in front of the sensor, allowing a certain bandwidth of the visible light to pass through.

Kodak scientist Dr. Bryce Bayer realized that an image sensor with a Color Filter Array (CFA) pattern would allow the reconstruction of all the colors of a scene from a single image capture. The color filter array is manufactured as part of the image sensor as a layer laid over the phototransistors. Example CFA patterns are shown in Figure 1-1. These are called Bayer patterns and are used in most digital imaging systems.

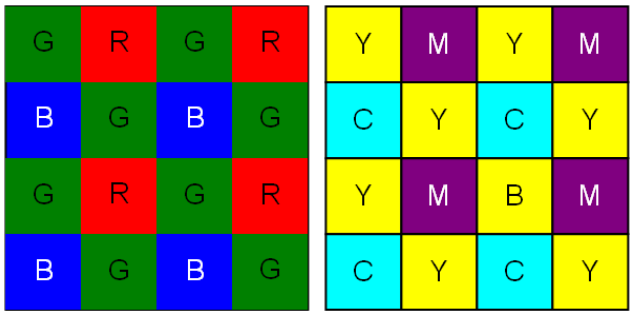

Figure 1-1: **RGB and CMY Bayer CFA Patterns**

The original data for each pixel contains information only about one color, based on which color filter is positioned over that pixel. However, information for all three primary colors is needed at each pixel to reconstruct a color image. Some missing information can be recreated from the information available in neighboring pixels. This process of recreating the missing color information is called color interpolation or demosaicing, and may require dedicated hardware to process the image data in real-time

There is no exact method to fully recover the missing information, as color channels have been physically sub-sampled by the CFA before proper low-pass filtering takes place, which may lead to aliasing between color channels.

Perfect recovery of the original signal may not be possible; however, the aliasing can be suppressed significantly by capitalizing on the temporal and spatial redundancies and structured nature of natural images/video sequences.

A variety of simple interpolation methods, such as Pixel Replication, Nearest Neighbor Interpolation, Bilinear Interpolation, and Bicubic Interpolation have been widely used for CFA demosaicing. Simple methods usually compromise quality, and more elaborate methods require the use of an external frame buffer. The CFA core was designed to efficiently suppress interpolation artifacts, such as the zipper and color aliasing effects, by minimizing Chrominance Variances in a 5x5 neighborhood, as illustrated in Figure 1-2.

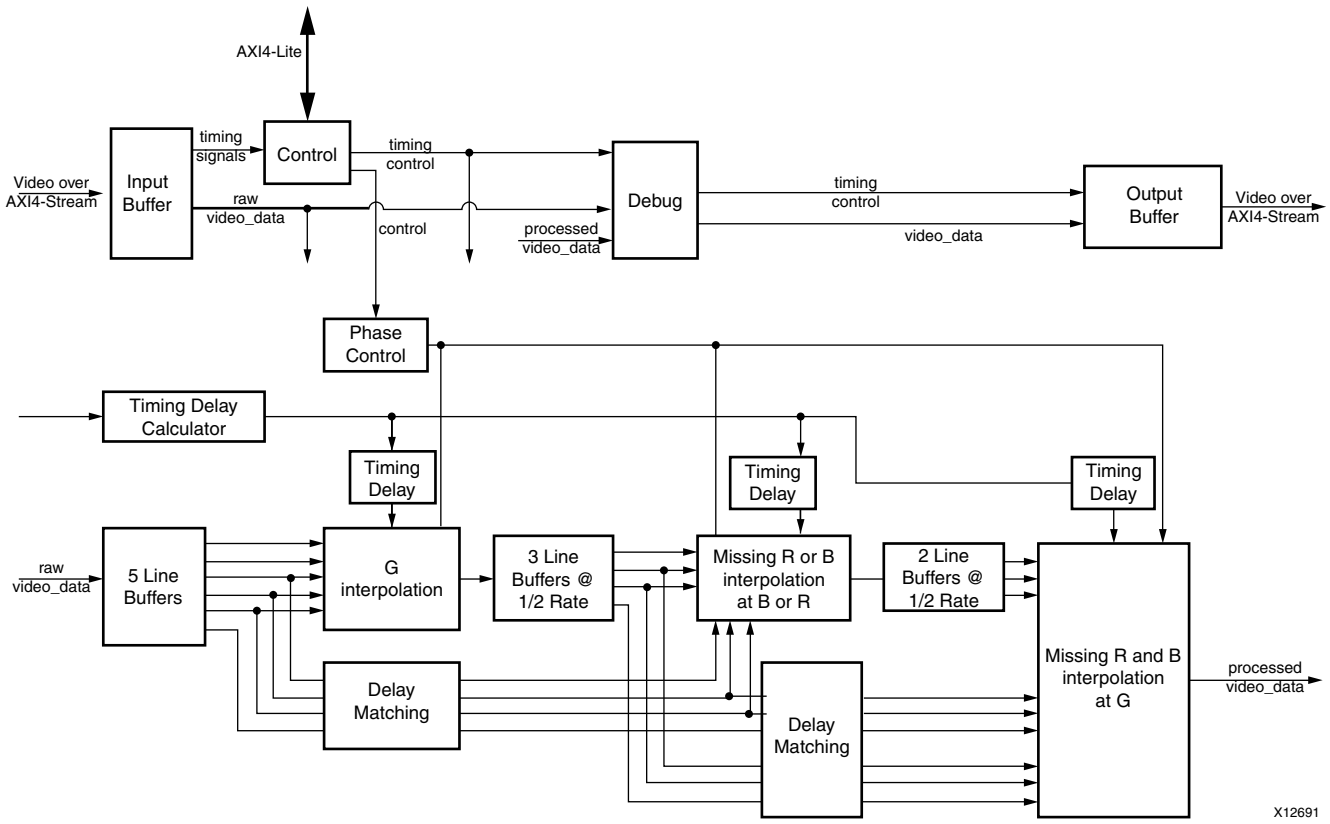

Figure 1-2: **Xilinx Color Filter Array Interpolation Block Diagram**

Image sensor data sheets specify whether the starting position, pixel(0,0) of the Bayer sampling grid is on a red-green or a blue-green line, and whether or not the first pixel is green. The Xilinx CFA LogiCORE IP supports all four possible Bayer phase combinations.

Feature Summary

The Color Filter Array Interpolation core reconstructs a color image from an RGB or CMY Bayer filtered sensor using a 5x5 interpolation aperture. The core is capable of a maximum resolution of 7680 columns by 7680 rows with 8, 10, or 12 bits per pixel and supports the bandwidth necessary for High-definition (1080p60) resolutions in all Xilinx FPGA device families (performance on low power devices may be lower). Higher resolutions can be supported in Xilinx high-performance device families.

You can configure and instantiate the core from IP Catalog, CORE Generator or EDK tools. Core functionality may be controlled dynamically with an optional AXI4-Lite interface.

Applications

- Pre-processing block for image sensors
- Video surveillance
- Industrial imaging
- Video conferencing
- Machine vision
- Other imaging applications

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite/ISE 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 Filter Array Interpolation 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 Compliance

The Color Filter Array Interpolation 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 (UG761) *AXI Reference Guide* [Ref 3] for additional information.

Performance

The following sections detail the performance characteristics of the Color Filter Array Interpolation core.

Maximum Frequencies

The Resource Utilization tables contain typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. Refer to in Table 2-1 through Table 2-8 for device-specific information.

Latency

The processing latency of the core is shown in the follow equation:

Latency = 4 scan lines + 58 pixels

Throughput

The Color Filter Array Interpolation core produces one output pixel per input sample.

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

accept valid data samples (m_axis_video_tready is not asserted) the core cannot accept valid input samples once its buffers become full.

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

However, at the end of each scan line the core flushes internal pipelines for 58 clock cycles, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples. Also, at the end of each frame the core flushes internal line buffers for 4 scan lines, during which the s_axis_video_tready is de-asserted signaling that the core is not ready to process samples.

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

When the core is processing data from a video source which can always provide valid data, e.g. a frame buffer, the throughput of the core can be defined as follows:

$$R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS + 4} \times \frac{COLS}{COLS + 58} \quad \textit{Equation 2-1}$$

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

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

$$f_{ACLK} = R_{MAX} \times \frac{ROWS + 4}{ROWS} \times \frac{COLS + 58}{COLS} = 124.4 \times \frac{1084}{1080} \times \frac{1978}{1920} = 128.5 \quad \textit{Equation 2-2}$$

Resource Utilization

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

The information presented in Table 2-1 through Table 2-5 is a guide to the resource utilization and maximum clock frequency of the Color Filter Array Interpolation core for all input/output width combinations for Zynq-7000, Artix-7, Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. The Xtreme DSP Slice count is always 8, regardless of parameterization, and this core does not use any dedicated I/O or CLK resources. The design was tested using Xilinx design tools with default tool options for characterization data. 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 (and Zynq-7030, 7045 Devices with Kintex Based Fabric)**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3149	2877	3122	5 / 6	8	241
10	3922	3461	3678	7 / 5	8	235
12	4567	3916	4235	7 / 6	8	191

1. Speedfile: XC7K70T-1 FBG484 ADVANCED 1.05 2012-07-02

Table 2-2: **Artix-7(and Zynq-7010, 7020 Devices with Artix Based Fabric)**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3258	2849	3122	5 / 6	8	152
10	3920	3495	3678	7 / 5	8	165
12	4437	4023	4235	7 / 6	8	160

1. Speedfile: XC7A100T-1 FGG484 ADVANCED 1.04 2012-07-02

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

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3163	2864	3122	5 / 6	8	272
10	3820	3525	3678	7 / 5	8	244
12	4330	4012	4235	7 / 6	8	229

1. Speedfile: XC7V585T-1 FFG1157 Advanced 1.05 2012-07-02

Table 2-4: **Spartan-6**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 16 / 8	DSP48A1	Fmax (MHz)
8	3270	2891	3227	15 / 1	8	114
10	4005	3557	3804	17 / 2	8	116
12	4447	3839	4344	20 / 0	8	120

1. Speedfile: XC6SLX25-2 FGG484 Production 1.22 2012-07-02

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

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3218	2896	3122	5 / 6	8	198
10	3898	3494	3678	7 / 5	8	204
12	4469	4002	4235	7 / 6	8	193

1. Speedfile: XC6VLX75T-1 FF484 Production 1.17 2012-07-02

Table 2-6 through Table 2-8 were generated using Vivado Design Suite 2012.2 with default tool options for characterization data.

Table 2-6: **Kintex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3686	2924	3533	4 / 2	16	215
10	4297	3437	4157	6 / 1	16	231
12	4825	3897	4792	6 / 2	16	231

1. Speedfile: XC7K70T-1 FBG1157 Advanced 1.05 2012-07-02

Table 2-7: **Artix-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3700	2926	3533	4 / 2	16	149
10	4299	3432	4157	6 / 1	16	182
12	4861	3897	4792	6 / 2	16	182

1. Speedfile: XC7A100T-1 FGG484 Advanced 1.04 2012-07-02

Table 2-8: **Virtex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	3624	2925	3533	4 / 2	16	243
10	4300	3433	4157	6 / 1	16	243
12	4838	3896	4792	6 / 2	16	243

1. Speedfile: XC7V585T-1 FFG1157 Advanced 1.05 2012-07-02

Core Interfaces and Register Space

Port Descriptions

The Color Filter Array Interpolation 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 CFA 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 through the GUI with the INTC interface enabled.

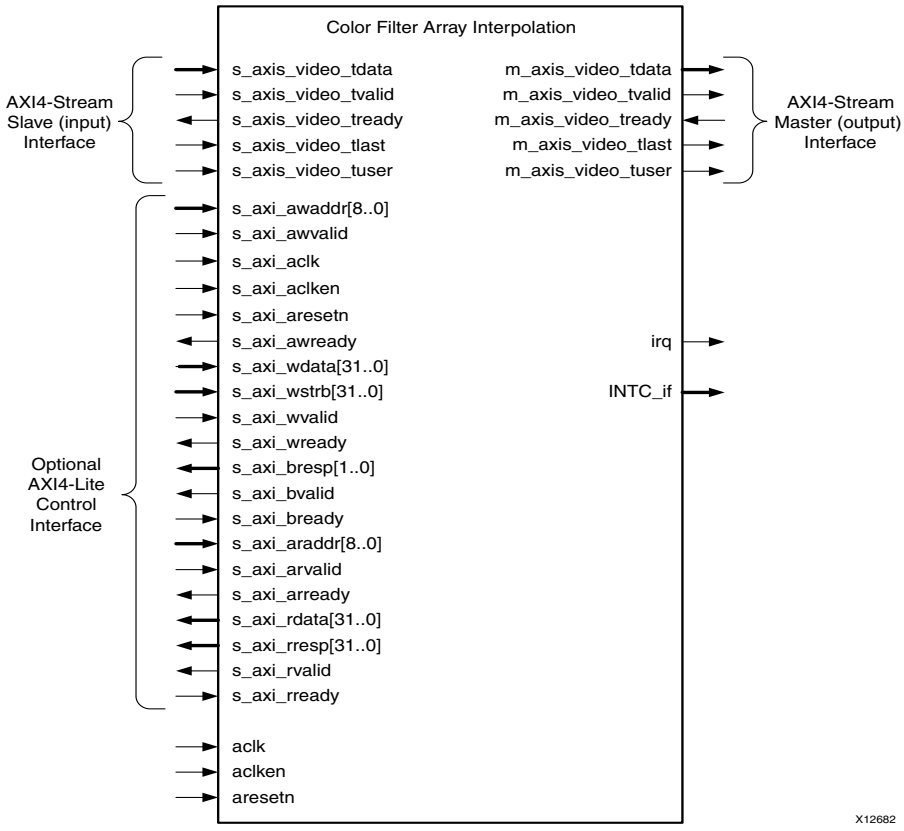

Figure 2-1: **CFA Core Top-Level Signaling Interface**

Common Interface Signals

Table 2-9 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-9: **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-Lite interface is selected on GUI.

The ACLK, ACLKEN and ARESETn signals are shared between the core and the AXI4-Stream data interfaces. The AXI4-Lite control interface has its own set of clock, clock enable and reset pins: S_AXI_ACLK, S_AXI_ACLKEN and S_AXI_ARESETn. Refer to The Interrupt Subsystem for a detailed 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 CFA 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 3].

AXI4-Stream Signal Names and Descriptions

Table 2-10 describes the AXI4-Stream signal names and descriptions.

Table 2-10: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	8,16	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	Output Video Data
m_axis_video_tvalid	Out	1	Output Valid

Table 2-10: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
m_axis_video_tready	In	1	Output Ready
m_axis_video_tuser	Out	1	Output Video Start Of Frame
m_axis_video_tlast	Out	1	Output Video End Of Line

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit sensor data must be padded with zeros on the MSB to form a 16 bit wide vector before connecting to s_axis_video_tdata. Padding does not affect the size of the core.

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

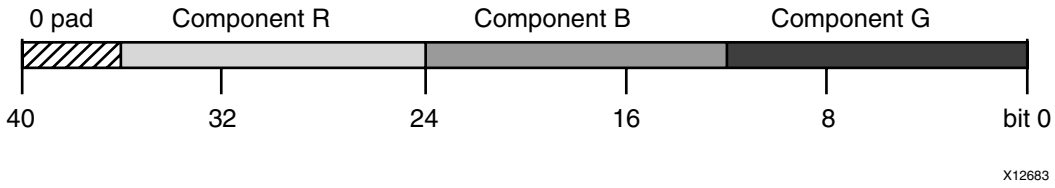

Figure 2-2: **12-bit RGB Data Encoding on m_axis_video_tdata**

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 through the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

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

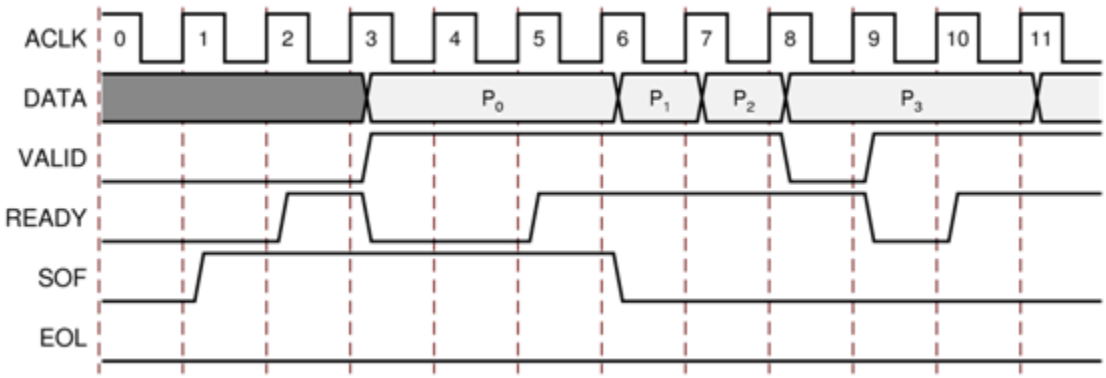

Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

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 CFA 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. It is recommended that the AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

Start of Frame Signals - m_axis_video_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. The SOF signal 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 (EOL) 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.

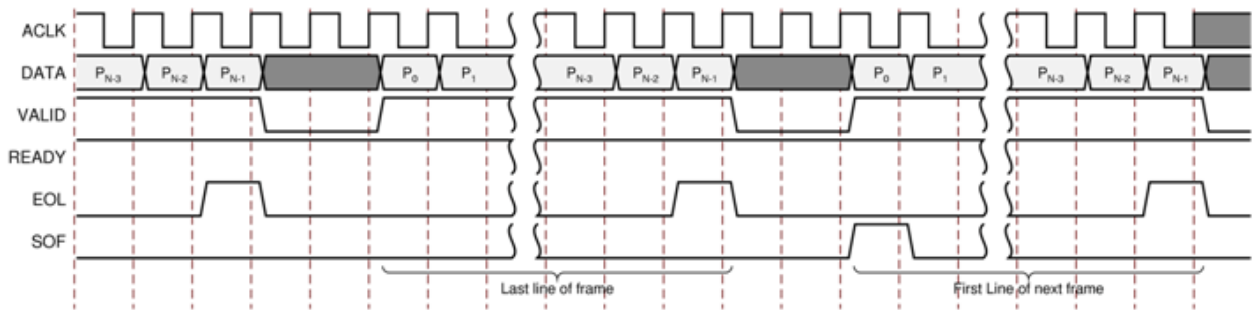

Figure 2-4: **Use of EOL and SOF Signals**

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 through AXI4-Lite interface to an AXI4-Lite master. In a static configuration with a fixed set of parameters (constant configuration), the core can be instantiated without the AXI4-Lite control interface, which reduces the core Slice footprint.

Constant Configuration

The constant configuration caters to users who interface the core to a particular image sensor with a known, stationary resolution and Bayer Phase. Typically, the starting phase of the Bayer pattern can be obtained from the sensor documentation. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame), and the Bayer phase is hard coded into the core through the 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 CFA core can be controlled through the AXI4-Lite interface using read and write transactions to the CFA register space.

Table 2-11: **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

Table 2-11: **AXI4-Lite Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
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 CFA core uses only one timing related register, ACTIVE_SIZE (0x0020), which allows specifying the input frame dimensions. Also, the core has only one core-specific register, BAYER_PHASE (0x0100) which allows specifying the phase of the Bayer grid over the image sensor sampling array, as described in BAYER_PHASE(0x0100) Register. Table 2-12 describes the register names.

Table 2-12: **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
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

Table 2-12: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
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	31-0: Frame Throughput monitor(1)
0x0018	SYSDEBUG1	R	N/A	0	31-0: Line Throughput monitor(1)
0x001C	SYSDEBUG2	R	N/A	0	31-0: Pixel Throughput monitor(1)
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per scan line 28-16: Number of Active Lines per Frame
0x0100	BAYER_PHASE	R/W	Yes	Specified via GUI	1-0: Bayer Sampling Grid starting position

1. Only available when the debugging features option is enabled when 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 CFA core ACTIVE_SIZE and BAYER_PHASE 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 are copied over to the active set at the end of the AXI4-Stream interface 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 CFA core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. Refer to Debugging Features 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 Debugging Features 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 software 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 through the ARESETn pin.

STATUS (0x0004) Register

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

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

Bit 0 of the STATUS register, PROC_STARTED, indicates that processing of a frame has commenced through 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 through 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 through 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 through 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 through 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 through 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. See Table 2-12 for details.

SYSDEBUG0 (0x0014) Register

The SYSDEBUG0, or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG registers can be useful to identify external memory / Frame buffer / or throughput bottlenecks in a video system. Refer to Debugging Features 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 Debugging Features 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 Debugging Features 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 128 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 128 to 7680. To avoid processing errors, the user should restrict values written to ACTIVE_SIZE to the range supported by the core instance.

BAYER_PHASE(0x0100) Register

Image sensor data sheets specify whether the starting position, pixel(0,0), of the Bayer sampling grid is on a red-green, or blue-green line, and whether the first pixel is green or not. The Xilinx CFA LogiCORE IP supports all four possible Bayer phase combinations. Bits 1:0 specify whether the top-left corner of the Bayer sampling grid starts with Green, Red, or Blue Pixel, according to Figure 2-5, which displays top-left corner of the imager sample matrix along with the Bayer Phase Register value combinations.

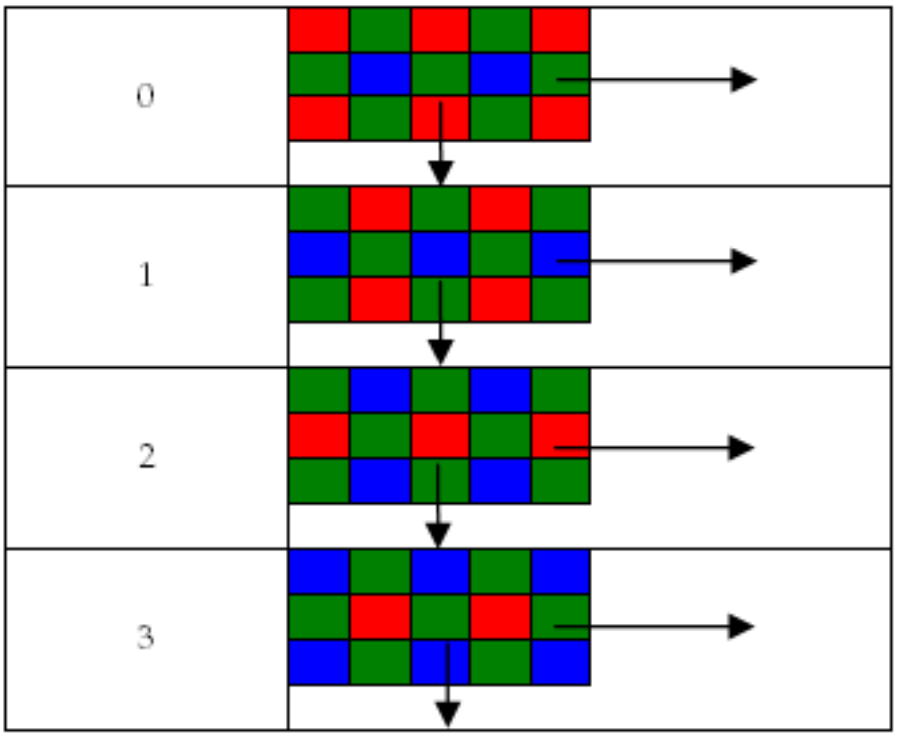

Figure 2-5: **Bayer Phase Register Combination Definitions**

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 CFA 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. Once set by the corresponding event, bits of the STATUS register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface the user can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional INTC_IF port. This vector of signals gives parallel access to the individual interrupt sources listed in Table 2-13.

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

Table 2-13: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	EOL Early
3	EOL Late
4	SOF Early
5	SOF Late

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

Designing with the Core

General Design Guidelines

The CFA core interpolates Bayer sub-sampled image sensor data to downstream processing modules that process RGB data. The core processes samples provided via an AXI4-Stream slave interface, outputs pixels via an AXI4-Stream master interface, and can be controlled via an optional AXI4-Lite interface. The CFA Interpolation block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate. It is recommended that the CFA core is 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 couples the sensor data interface to AXI4-Stream.

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

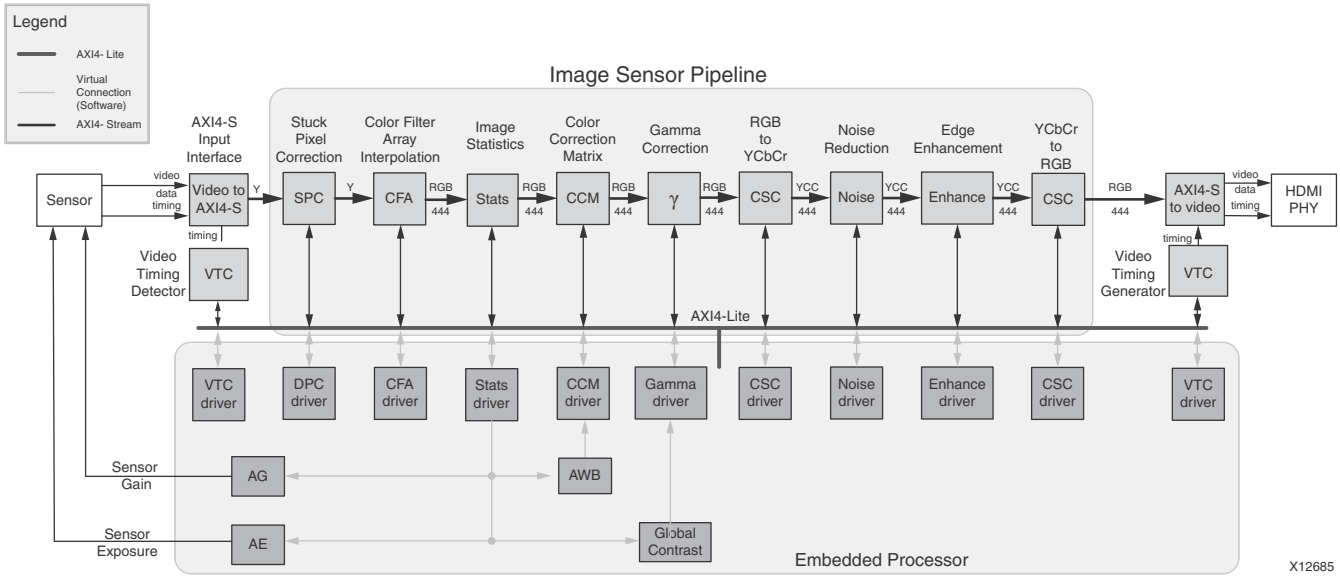

Figure 3-1: **Image Sensor Pipeline System with CFA Core**

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.

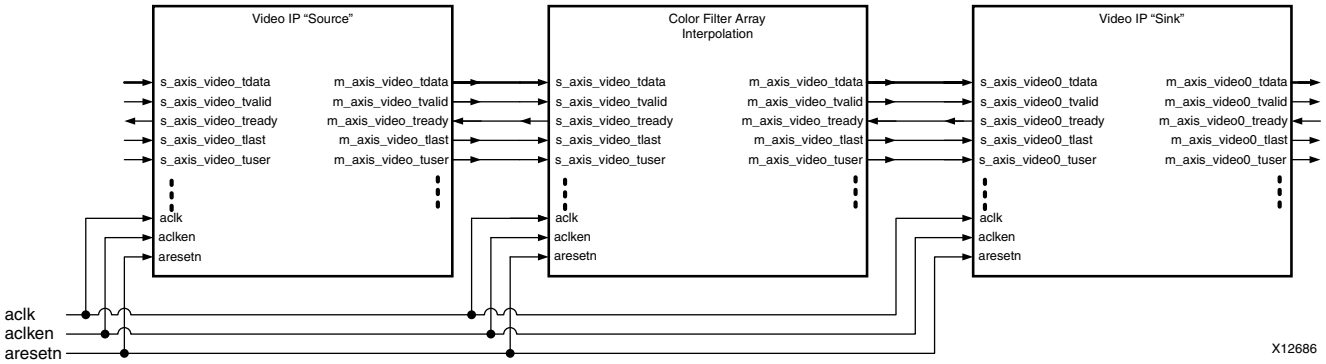

Figure 3-2: **Example of ACLK Routing in an ISP Processing Pipeline**

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

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

The ACLKEN pin facilitates:

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

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

Note: When two cores connected via AXI4-Stream interfaces, where only the master or the slave interface has an ACLKEN port, which is not permanently tied high, the two interfaces should be connected via the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 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 CFA core has two reset source: the ARESETn pin (hardware reset), and the software reset option provided via the AXI4-Lite control interface (when present).

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

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

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 Filter Array IP core must be configured for the actual image sensor frame-size and Bayer-phase to operate properly. To gather the frame size information from the image sensor, 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 image sensor 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 CFA, with the appropriate image dimensions. The Bayer-phase combination to be used can be derived from the sensor data sheet. If the data sheet is not available, the user first should identify whether the first pixel (pixel[0,0]) is green or not. If the green pixel location is chosen incorrectly, the resulting image shows a strong checkerboard pattern. Once the green pixel phase is established, the user should identify whether the first line contains Red or Blue pixels. If the Red or Blue pixel location is chosen incorrectly, the resulting image displays a Red-Blue color swap.

If the target system uses only one, stationary image sensor, with sensor aperture values fixed (no Pan-Tilt-Zoom, or cropping function by modifying sensor registers), the user may choose to consolidate the active-size and Bayer Phase values, and create a constant configuration by removing the AXI4-Lite interface. This option allows reducing the core Slice footprint.

Clock Domain Interaction

The ARESETn and ACLKEN input signals does 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 responds with an error if the core registers cannot be read or written within 128 S_AXI_ACLK clock cycles. The core registers cannot be read or written if the ARESETn signal is held low, if the ACLKEN signal is held low or if the ACLK signal is not connected or not running. If core register read does not complete, the AXI4-Lite read transaction 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 if the system needs to be reinitialized, it is recommended that pipelined Xilinx IP video cores are disabled/reset from system output towards the system input, and programmed/enabled from system input to system output. STATUS register bits allow system processors to identify the processing states of individual constituent cores, and successively disable a pipeline as one core after another is finished processing the last frame of data.

Error Propagation and Recovery

Parameterization and/or configuration registers define the dimensions of video frames video IP should process. Starting from a known state, based on these configuration settings the IP can predict when the beginning of the next frame is expected. Similarly, the IP can predict when the last pixel of each scan line is expected. SOF detected before it was expected (early), or SOF not present when it is expected (late), EOL detected before expected (early), or EOL not present when expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration.

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

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

C Model Reference

Installation and Directory Structure

This chapter contains information for installing the Color Filter Array C-Model, and describes the file contents and directory structure.

Software Requirements

The Color Filter Array v6.00.a C-Models were compiled and tested with the following software versions.

Table 4-1: **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 2005, 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_cfa_v6_00_a_bitacc_cmodel.so` and `libstlport.so.5.1` files are located is in your $LD_LIBRARY_PATH environment variable.

C-Model File Contents

Unzipping the `v_cfa_v6_00_a_bitacc_model.zip` file creates the following directory structures and files which are described in Table 4-2.

Table 4-2: **C-Model Files**

File	Description
/lin	Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Linux Platforms
libIp_v_cfa_v6_00_a_bitacc_cmodel.lib	Color Filter Array Interpolation v6.00.a model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by the Color Filter Array Interpolation and RGB to YCrCb object libraries (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_cfa_v6_00_a_bitacc_cmodel.lib	Color Filter Array Interpolation v6.00.a model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by the Color Filter Array Interpolation and RGB to YCrCb object libraries (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_cfa_v6_00_a_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_cfa_v6_00_a_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
pg002-v-cfa.pdf	*Color Filter Array Interpolation Core Product Guide*
v_cfa_v6_00_a_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.

Table 4-2: **C-Model Files *(Cont'd)***

File	Description
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, declared in the header file `v_cfa_v6_00_a_bitacc_cmodel.h`. Before using the model, the structures holding the inputs, generics and output of the CFA instance have to be defined, as illustrated below.

```
struct  xilinx_ip_v_cfa_v6_00_a_generics cfa_generics;
struct  xilinx_ip_v_cfa_v6_00_a_inputs   cfa_inputs;
struct  xilinx_ip_v_cfa_v6_00_a_outputs  cfa_outputs;
```

Declaration of the above structs are located in the `v_cfa_v6_00_a_bitacc_cmodel.h` file.

The only generic parameter the CFA v6.00.a IP Core bit accurate C-model takes is `DATA_WIDTH`, corresponding to the Core Generator **Data Width** parameter. Allowed values are 8, 10 and 12. Calling

```
xilinx_ip_v_cfa_v6_00_a_get_default_generics (&cfa_generics)
```

initializes the generics structure with the CFA GUI default `DATA_WIDTH` value (8).

The structure cfa_inputs defines the values of run-time parameters `BAYER_PHASE` and the actual input image. For the description of `BAYER_PHASE`, see Figure 2-5, page 26. For the description of the input structure, see CFA Input and Output Video Structure.

Calling `xilinx_ip_v_cfa_v6_00_a_get_default_inputs(&cfa_generics, &cfa_inputs)` initializes the `BAYER_PHASE` member of the input structure with the CFA GUI default value (3).

Note: The `video_in` variable is not initialized, as the initialization depends on the actual test image to be simulated. The next chapter describes the initialization of the video_in structure.

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

```
int xilinx_ip_v_cfa_v6_00_a_bitacc_simulate(
  struct xilinx_ip_v_cfa_v6_00_a_generics* generics,
  struct xilinx_ip_v_cfa_v6_00_a_outputs*  outputs).
```

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

After the outputs were evaluated and saved, dynamically allocated in memory for input and output video structures have to be released by calling function:

```
void xilinx_ip_v_cfa_v6_00_a_destroy(
  struct xilinx_ip_v_cfa_v6_00_a_inputs *input,
  struct xilinx_ip_v_cfa_v6_00_a_outputs *output).
```

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

CFA Input and Output Video Structure

Input images or video streams can be provided to the Color Filter Array v6.00.a 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 4-3 lists the video structure variables.

Table 4-3: **Member Variables of the Video Structure**

Member Variable	Designation
frames	Number of video/image frames in the data structure.
rows	Number of rows per frame[a]
cols	Number of columns per frame[a]
bits_per_component	Number of bits per color channel/component[b]
mode	Contains information about the designation of data planes[c]
data	Set of five pointers to three-dimensional arrays containing data for image planes.[d]

a. Pertaining to the image plane with most rows and columns, such as t6he luminance channel for y,u,v data. Frame dimensions are assumed constant through all frames of the video stream; however, different planes (such as y, u, and v) may have different dimensions.
b. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
c. Named constants to be assigned to mode are listed in Table 4-4.
d. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col]

Table 4-4 lists the named constants and the mode to be assigned.

Table 4-4: **Named Video Modes Constants with Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome- Luminance only
FORMAT_RGB	3	RGB image/video data
FORMAT_C444	3	444YUV, or YCrCb image/video data

Table 4-4: **Named Video Modes Constants with Planes and Representations *(Cont'd)***

Mode	Planes	Video Representation
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

Note: When using the C-model, the CFA core accepts FORMAT_RGB as input and FORMAT_RGB as output.

Initializing the CFA Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or sequence of images. 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 which 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 eight bits per pixel, and operates on images with three planes (R,G,B). Therefore, functions:

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

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.

Exchanging data between rgb8_video_struct and general video_struct type frames/videos is facilitated by 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 );
```

Note: All image / video manipulation utility functions expect both input and output structures initialized, (for example, pointing to a structure which has been allocated in memory), either as static or dynamic variables. Moreover, the input structure has to have the dynamically allocated container (data or 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 throw an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated the utility functions creates the appropriate container to hold results.

Binary Image/Video Files

The header `video_utils.h` declares functions which help load and save generalized video files in raw, un-compressed format. Functions

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

effectively serialize the `video_struct` structure. The corresponding file contains a small, plain text header defining "Mode", "Frames", "Rows", "Columns", and "Bits per 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 may differ within each frame as defined by the actual video mode selected.

Working with video_struct Containers

Header file `video_utils.h` define functions to simplify access to video data in video_struct.

```
int video_planes_per_mode(int mode);
int video_rows_per_plane(struct video_struct* video, int plane);
int video_cols_per_plane(struct video_struct* video, int plane);
```

Function video_planes_per_mode returns the number of component planes defined by the mode variable, as described in Table 4-4. Functions `video_rows_per_plane` and `video_cols_per_plane` return the number of rows and columns in a given plane of the selected video structure. The example below demonstrates all pixels within a video stream stored in variable `in_video`:

```
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]
      }
    }
  }
}
```

Destroying the Video Structure

Finally, the video structure must be destroyed to free up memory used to store the video structure.

C-Model Example Code

The example C demonstrator provided with the core, `run_bitacc_cmodel.c` demonstrates the steps required to run the C-model, by:

- Opening an example bmp file.
- Bayer sub-sampling the image.
- Increasing the color-component width to 10 or 12 bits as necessary by shifting 8 bit data derived from the bmp file and padding the LSBs with X-Y ramp bits.
- Running the CFA C-model.

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

```
Usage: run_bitacc_cmodel in_file out_file
          in_file     : path/name of the input  BMP file
          out_file    : path/name of the output BMP file
```

The C-model demonstrator `run_bitacc_cmodel.c` also generates stimuli and results text files which in turn can be processed by the example Verilog test-bench. During successful execution, two files with the extension 'txt', are created. The first file corresponds to the input bmp image, and has the same path and name as the input file, with extension '.txt'. The other file similarly corresponds to the output file. These files contain the inputs and outputs of the CFA algorithm in full precision, as the BMP format does not support color resolutions beyond eight bits per component.

Compiling with the CFA C-Model

Linux (32- and 64-bit)

To compile the example code, first ensure that the directory in which the files `libIp_v_cfa_v6_00_a_bitacc_cmodel.so` and `libstlport.so.5.1` are 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 ../parsers.c -o run_bitacc_cmodel -L.
-lIp_v_cfa_v6_00_a_bitacc_cmodel -Wl,-rpath,.

gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o run_bitacc_cmodel -L.
-lIp_v_cfa_v6_00_a_bitacc_cmodel -Wl,-rpath,.
```

Windows (32- and 64-bit)

Precompiled library `v_cfa_v6_00_a_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_cfa_v6_00_a_bitacc_cmodel.dll` file to the **Resource Files** folder of the project.
- The `run_bitacc_cmodel.c` file to the **Source Files** folder of the project.
- The `v_cfa_v6_00_a_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.

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Customizing and Generating the Core

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

GUI

The Xilinx Color Filter Array Interpolation core is easily configured to meet the developer's specific needs through the Vivado design tools Graphical User Interface (GUI). This section provides a quick reference to parameters that can be configured at generation time.

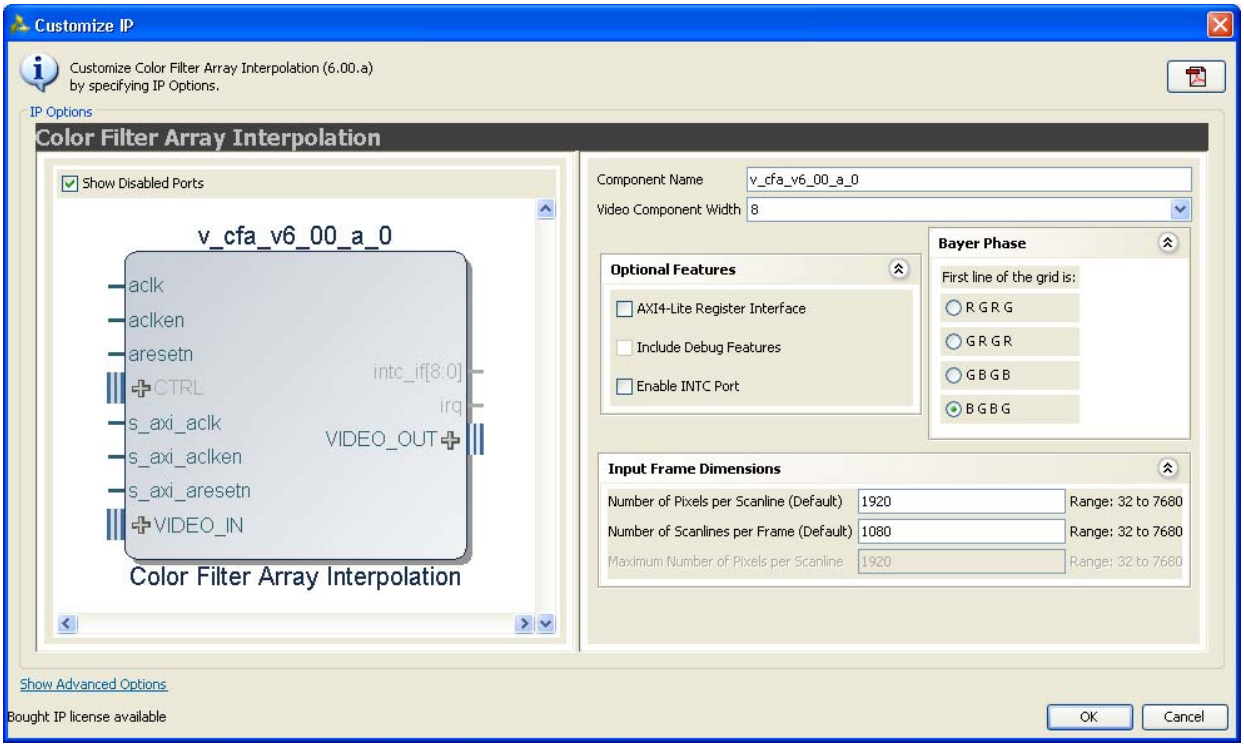

Figure 5-1: **Color Filter Array Interpolation Vivado IP Catalog GUI**

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

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name `v_cfa_v6_00_a` cannot be used as a component name.
- **Video Component Width:** Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.
- **Optional Features:**
  - **AXI4-Lite Register Interface:** When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, see 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, see Debugging Features in Appendix C.

    Note: 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, see The Interrupt Subsystem in Chapter 2.
- **Bayer Phase:** Specify whether the top-left corner of the Bayer sampling grid starts with Green, Red, or Blue Pixel. When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the `BAYER_PHASE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the interpolation supported by the core instance.
- **Input Frame Dimensions:**
  - **Number of Active Pixels per Scan line:** When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Active Lines per Frame:** When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the upper half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.
  - **Maximum Number of Active Pixels Per Scan line:** Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 128 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for **Number of Active Pixels per Scan line**, or the corresponding lower half-word of the `ACTIVE_SIZE` register must always be less than the value provided by **Maximum Number of Active Pixels Per Scan line**. Using a tight upper-bound results in

optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field are reflecting the actual contents of the **Number of Active Pixels per Scan line** field as for constant mode the maximum number of pixels equals the active number of pixels.

Output Generation

Vivado generates the files necessary to build the core and place those files in the `<project>/<project>.srcs/sources_1/ip/<core>` directory.

File Details

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

Name	Description
v_cfa_v6_00_a	Library directory and encrypted source files for the v_cfa_v6_00_a core
v_tc_v5_00_a	Library directory and encrypted source files for the helper core used with the v_cfa_v6_00_a
v_cfa_v6_00_a.veo	Verilog instantiation template
v_cfa_v6_00_a.vho	VHDL instantiation template
v_cfa_v6_00_a.xci	IP-XACT XML file describing which options were used to generate the core. An XCI file can also be used as a source file for Vivado.
v_cfa_v6_00_a.xml	IP-XACT XML file describing how the core is constructed so Vivado can properly build the core.

Constraining the Core

Required Constraints

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

Device, Package, and Speed Grade Selections

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

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for the Color Filter Array Interpolation core. See Maximum Frequencies in Chapter 2. The S_AXI_ACLK maximum frequency is the same as the ACLK maximum.

Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the ACLK clock signal, but must not be more than 128x faster than ACLK.

Clock Placement

There are no specific Clock placement requirements for the Color Filter Array Interpolation core.

Banking

There are no specific Banking rules for the Color Filter Array Interpolation core.

Transceiver Placement

There are no Transceiver Placement requirements for the Color Filter Array Interpolation core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Color Filter Array Interpolation core.

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core in the ISE Design Suite.

GUI

The Xilinx Color Filter Array Interpolation core is easily configured to meet the developer's specific needs through the CORE Generator™ or EDK GUIs. This section provides a quick reference to parameters that can be configured at generation time.

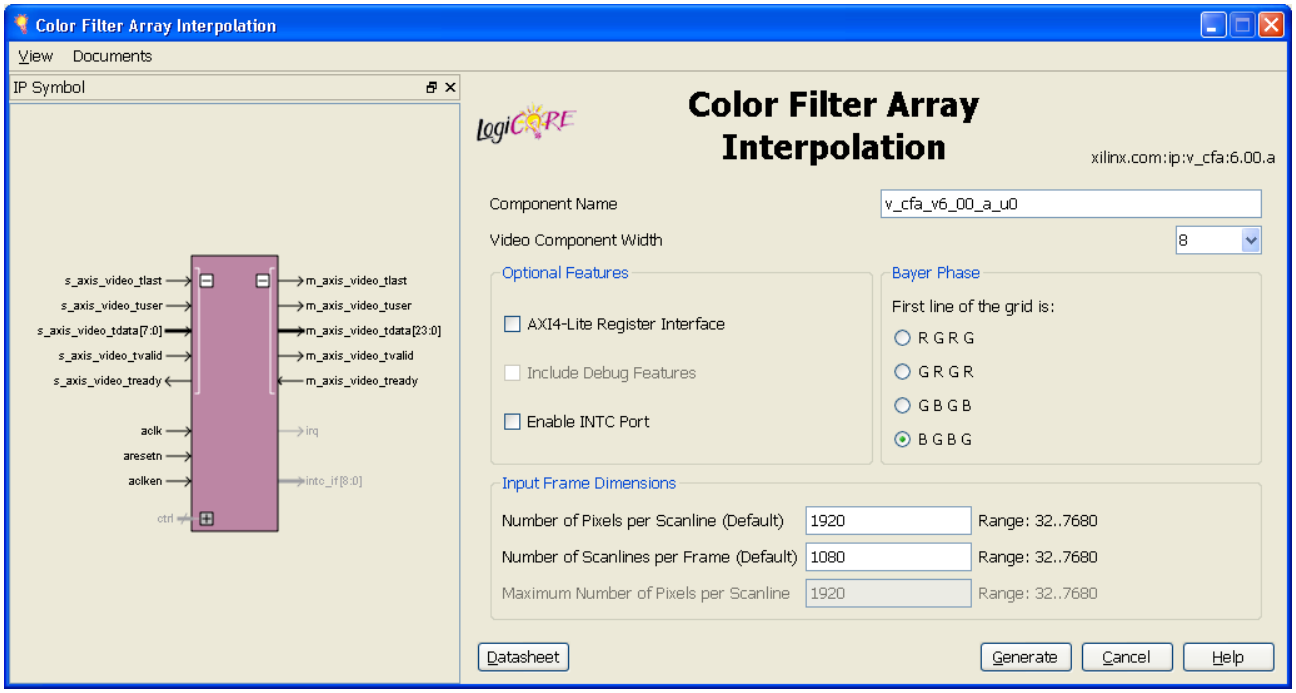

Figure 7-1: **Color Filter Array Interpolation CORE Generator GUI**

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

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed

from characters: a to z, 0 to 9 and "_". The name `v_cfa_v6_00_a` cannot be used as a component name.

- **Video Component Width:** Specifies the bit width of input samples. Permitted values are 8, 10, and 12 bits.
- **Optional Features:**
  - **AXI4-Lite Register Interface:** When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. For more information, see 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, see Debugging Features in Appendix C.

    Note: 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, see The Interrupt Subsystem in Chapter 2.
- **Bayer Phase:** Specify whether the top-left corner of the Bayer sampling grid starts with Green, Red, or Blue Pixel. When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the `BAYER_PHASE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the interpolation supported by the core instance.
- **Input Frame Dimensions:**
  - **Number of Active Pixels per Scan line:** When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Active Lines per Frame:** When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the upper half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.
  - **Maximum Number of Active Pixels Per Scan line:** Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 128 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for **Number of Active Pixels per Scan line**, or the corresponding lower half-word of the `ACTIVE_SIZE` register must always be less than the value provided by **Maximum Number of Active Pixels Per Scan line**. Using a tight upper-bound results in optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field reflect the actual contents of the **Number**

of Active Pixels per Scan line field because the maximum number of pixels equals the active number of pixels in constant mode.

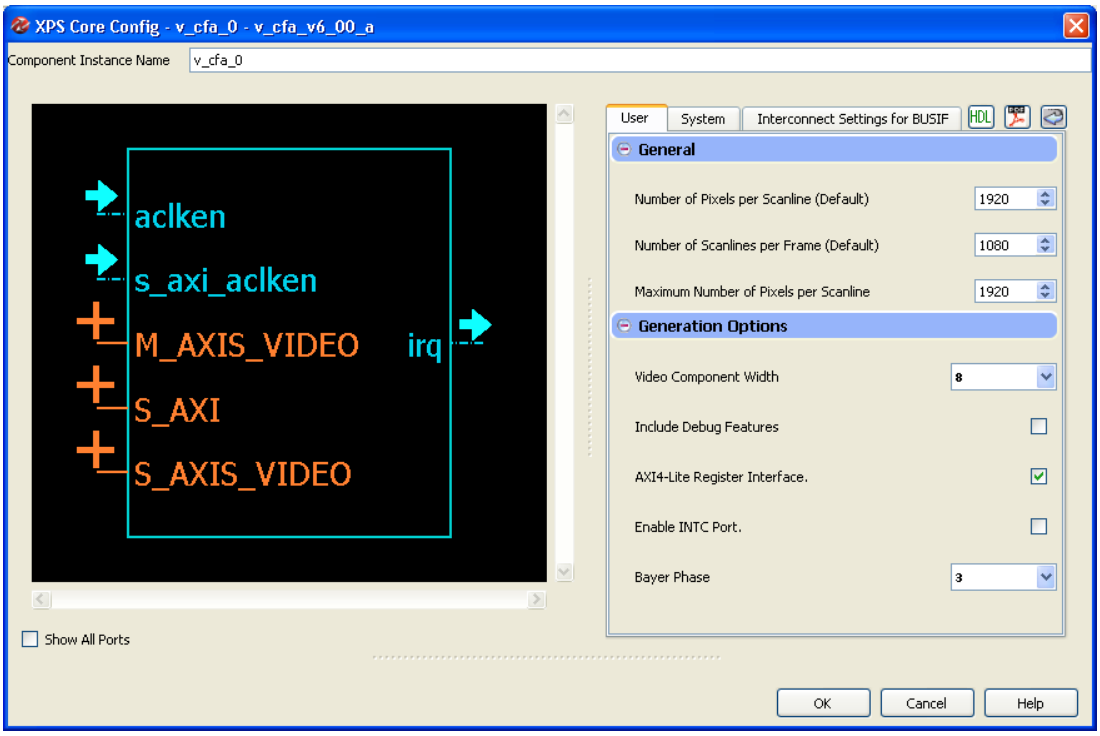

Figure 7-2: **Color Filter Array Interpolation, EDK GUI Screen**

Definitions of the EDK GUI controls are identical to the corresponding CORE Generator GUI functions, except for the Bayer Phase selection. For the definition of Bayer Phase encodings, see BAYER_PHASE(0x0100) Register in Chapter 2.

Parameter Values in the XCO File

Table 7-1 defines valid entries for the XCO parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 7-1: **XCO Parameters**

XCO Parameter	Default Values
active_rows	1080
bayer_phase	3
component_name	v_cfa_v6_00_a_u0
data_width	8

Table 7-1: **XCO Parameters *(Cont'd)***

XCO Parameter	Default Values
has_axi4_lite	false
has_debug	false
has_intc_if	false
max_cols	1920

Output Generation

CORE Generator outputs the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the <project directory>.

File Details

The CORE Generator output consists of some or all the files listed in Table 7-2.

Table 7-2: **CORE Generator Output Files**

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo	The HDL template for instantiating the core.
<component_name>.vho	
<component_name>.v	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>.vhd	
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.

Constraining the Core

Required Constraints

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

Device, Package, and Speed Grade Selections

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

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for the Color Filter Array Interpolation core. See Maximum Frequencies in Chapter 2. The S_AXI_ACLK maximum frequency is the same as the ACLK maximum.

Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the ACLK clock signal, but must not be more than 128x faster than ACLK.

Clock Placement

There are no specific Clock placement requirements for the Color Filter Array Interpolation core.

Banking

There are no specific Banking rules for the Color Filter Array Interpolation core.

Transceiver Placement

There are no Transceiver Placement requirements for the Color Filter Array Interpolation core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Color Filter Array Interpolation core.

Detailed Example Design

Directory and File Contents

The directory structure underneath the top-level folder is:

- expected:
  Contains the pre-generated expected/golden data used by the test bench to compare actual output data.
- stimuli:
  Contains the pre-generated input data used by the test bench to stimulate the core (including register programming values).
- Results:
  Actual output data is written to a file in this folder.
- Src:
  Contains the .vhd simulation files and the .xco CORE Generator parameterization file of the core instance. The .vhd file is a netlist generated using CORE Generator. The .xco file can be used to regenerate a new netlist using CORE Generator.

The available core C-model can be used to generate stimuli and expected results for any user bmp image. For more information, see Chapter 4, C Model Reference.

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

- isim_wave.wcfg:
  Waveform configuration for ISIM
- mti_wave.do:
  Waveform configuration for ModelSim
- run_isim.bat:
  Runscript for iSim in Windows
- run_isim.sh:
  Runscript for iSim in Linux
- run_mti.bat:
  Runscript for ModelSim in Windows
- run_mti.sh:
  Runscript for ModelSim in Linux

Example Design

No example design is available at the time for the LogiCORE IP Color Filter Array Interpolation v6.00.a core.

Demonstration Test Bench

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

Test Bench Structure

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

- DUT
  The CFA core instance under test.
- axi4lite_mst
  The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- axi4s_video_mst
  The AXI4-Stream master module, which opens the stimuli txt file and initiates AXI4-Stream transactions to provide stimuli data for the core
- axi4s_video_slv
  The AXI4-Stream slave module, which opens the result txt file and verifies AXI4-Stream transactions from the core
- ce_gen
  Programmable Clock Enable (ACLKEN) generator

Implementation

WRITER NOTE: No corresponding section in original document.

Simulation

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

SECTION IV: APPENDICES

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the Color Filter Array Interpolation 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 Filter Array Interpolation 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 Filer Array Interpolation 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

Quality Measures

Table A-1 provides Peak Signal to Noise Ratio (PSNR) measurement results for typical test images using an 8-bit input data.

Table A-1: **PSNR Results for Typical Test Images**

Image	PSNR [dB]
	34.051
	39.404
	33.736

Interoperability

The core slave (input) AXI4-Stream interface can work directly with the Video In to AXI4-Stream or Defective Pixel Correction cores. The core master (output) RGB interface can work directly with any Xilinx Video core which consumes RGB data.

Migrating

From version v4.0 to v5.00.a of the CFA 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 CFA 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 v5.00.a to v6.00.a of the CFA core, the following changes took place:

- The core originally had `aclk`, `aclken` and `aresetn` to control both the AXI4-Stream and AXI4-Lite interfaces. Separate clock, clock enable and reset pins now control the 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 AXI4-Stream domains.

Because of the complex nature of these changes, replacing a v4.0 version of the core in a customer design is not trivial. An existing EDK pCore, Transparent, or Constant CFA instance can be converted from XSVI to AXI4-Stream, using the Video In to AXI4-Stream core or components from XAPP521 (v1.0), *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol* [Ref 4].

A v4.0 pCore instance in EDK can be replaced from v6.00.a directly from the EDK IP Catalog. However, the application software needs to be updated for the changed functionality and addresses of the `IRQ_ENABLE`, `STATUS`, `ERROR`, and `BAYER_PHASE` registers. Consider replacing a legacy CFA pCore from EDK with a v6.00.a instance without AXI4-Lite interface to save resources.

If the user design explicitly used the timing detector or generator functionality of the CFA core, consider adding the Video Timing Controller core to migrate the functionality.

An ISE design using the General Purpose Processor interface, all of the above steps might be necessary:

- Timing detection, generation using the Video Timing Controller Core
- Replacing XSVI interfaces with conversion modules described in XAPP521 or try using the Video In to AXI4-Stream core
- Updating the CFA core instance to v6.00.a with or without AXI4-Lite interface

The INTC interface and debug functionality are new features for v6.00.a. When migrating an existing design, these functions may be disabled.

Debugging

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

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

1. Bring up the AXI4-Lite interface
2. Bring up the AXI4-Stream interfaces
3. Finding the right Bayer Phase value
    - (Optional) Balancing throughput

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

Bringing up the AXI4-Lite Interface

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

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

Symptom	Solution
Readback from the Version Register via 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? In EDK, verify that the S_AXI_ACLK and ACLK pin connections in the system.mhs file. The `VERSION_REGISTER` readout issue may be indicative of the core not receiving the AXI4-Lite interface.
Readback from the Version Register via 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`? In EDK, verify that signal `ACLKEN` is connected in the **`system.mhs`** 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 via 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. In EDK, verify that the S_AXI_ARESETn and ARESETn signals are connected in the **`system.mhs`** 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 EDK/SDK 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.

Troubleshooting the AXI4-Stream Interfaces

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

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

Symptom	Solution
Bit 0 of the `ERROR` register reads back set.	Bit 0 of the `ERROR` register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, 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 Chipscope, 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 Chipscope, measure the number EOL pulses between subsequent SOF pulses.

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

Symptom	Solution
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the CFA 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 CFA core cannot send data downstream, and the internal FIFOs are full.
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 the CFA 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 the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA `ACLK` pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the CFA 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 the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA `ACLK` pin lower than the reported Fmax reached?

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

Debugging Features

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

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 CFA core processing function is bypassed, and the core repeats AXI4-Stream input samples on its output. In bypass mode sensor samples are presented via the Green component output, while Red and Blue component outputs are set to zero.

Starting a system with all processing cores set to bypass, then by turning bypass off from the system input towards the system output allows verification of subsequent cores with known good stimuli.

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 data paths.

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.

Finding the Right Bayer Phase Value

Table C-3 lists the Bayer Phase values.

Table C-3: **Bayer Phases Values**

Symptom	Solution
Strong checkerboard pattern on output data.	The Green channel was not identified correctly. Set `BAYER_PHASE` to `BAYER_PHASE-2`.
Red-blue color swap on output data.	The Red channel was not identified correctly. Set `BAYER_PHASE` to `3-BAYER_PHASE`.
Image seems correct (no checkerboard pattern), but Green channel is swapped with Blue.	Make sure that bit-slices of `m_axis_video_tdata` are interpreted according to Figure 2-4.

See Solution Centers in Appendix E for information helpful to the debugging progress.

Interfacing to Third-Party IP

Table C-4 describes how to troubleshoot third-party interfaces.

Table C-4: **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. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.
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. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.

Application Software Development

Programmers Guide

The software API is provided to allow easy access to the CFA AXI4-Lite registers defined in Table 2-9. To utilize the API functions, the following two header files must be included in the user C code:

```
#include "cfa.h"
#include "xparameters.h"
```

The hardware settings of your system, including the base address of your CFA core, are defined in the `xparameters.h` file. The `cfa.h` file contains the macro function definitions for controlling the CFA 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 `cfa_v6_00_a0_a/example` subfolder. This file is a sample C program that demonstrates how to use the CFA pCore API.

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

Function Name and Parameterization	Description
CFA_Enable (uint32 BaseAddress)	Enables a CFA instance.
CFA_Disable (uint32 BaseAddress)	Disables a CFA instance.
CFA_Reset (uint32 BaseAddress)	Immediately resets a CFA instance. The core stays in reset until the RESET flag is cleared.
CFA_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.
CFA_FSync_Reset (uint32 BaseAddress)	Resets a CFA instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.
CFA_ReadReg (uint32 BaseAddress, uint32 RegOffset)	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-12).

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

Function Name and Parameterization	Description
CFA_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)	Write the register selected by RegOffset (defined in Table 2-12. Data is the 32-bit value to write to the register.
CFA_RegUpdateEnable (uint32 BaseAddress)	Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.
CFA_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. `CFA_Reset()` and `CFA_FSync_Reset ()` reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling `CFA_Reset()`, or setting bit 30 of the `CONTROL` register to 1 causes image tearing. After calling `CFA_Reset()`, the core remains in reset until `CFA_ClearReset()` is called.

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

Note: Calling `CFA_FSync_Reset()` does not guarantee prompt, or real-time resetting 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

Registers `BAYER_PHASE` and `ACTIVE_SIZE` are double-buffered to ensure no image tearing happens if values are modified during frame processing. Values from the AXI4-Liter 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 CFA core starts using the updated `ACTIVE_SIZE` and `BAYER_PHASE` 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 2-12 has a constant that is defined in cfa.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 CFA drivers which may change register locations does not affect the application dependent on the CFA driver.

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

Constant Name Definition	Value	Target Register
CFA_CONTROL	0x0000	CONTROL
CFA_STATUS	0x0004	STATUS
CFA_ERROR	0x0008	ERROR
CFA_IRQ_ENABLE	0x000C	IRQ_ENABLE
CFA_VERSION	0x0010	VERSION
CFA_SYSDEBUG0	0x0014	SYSDEBUG0
CFA_SYSDEBUG1	0x0018	SYSDEBUG1
CFA_SYSDEBUG2	0x001C	SYSDEBUG2
CFA_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
CFA_BAYER_PHASE	0x0100	BAYER_PHASE

Appendix E

Additional Resources

Xilinx Resources

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

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

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

http://www.xilinx.com/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.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

1. Eastman Kodak Company: *KAC – 1310, 1280 x 1024 SXGA CMOS Image Sensor Technical Data*.
2. Aptina MT9P031: *1/2.5-Inch 5Mp Digital Image Sensor Features*.
3. UG761 AXI Reference Guide.
4. XAPP521 (v1.0), *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol*.

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the *IP Release Notes Guide* (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The Color Filter Array Interpolation v6.00.a core is provided under the Xilinx Core License Agreement and can be generated using the Xilinx® CORE Generator™ system and EDK software. The CORE Generator system is shipped with Xilinx ISE® Design Suite software. The CORE Generator system and EDK are shipped with the Xilinx ISE Embedded Edition Design software.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release of Product Guide, replacing DS722 and UG802.
4/24/2012	2.0	Updated for core version. Added Zynq-7000 devices, added AXI4-Stream interfaces, deprecated GPP interface.
07/25/2012	3.0	Updated for core version. Added Vivado information.

Notice of Disclaimer