LogiCORE IP Color Filter Array Interpolation v4.0

Product Guide

PG002 October 19, 2011

XILINX®

Table of Contents

Appendix B: Migrating

Appendix C: Debugging

Appendix D: Application Software Development

Appendix E: C Model Reference

Appendix F: Additional Resources

LogiCORE IP Color Filter Array Interpolation v4.0 Product Guide

Introduction

The Xilinx Color Filter Array Interpolation LogiCORE™ IP provides an optimized hardware block to reconstruct 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
- Support for streaming or frame buffer processing
- Selectable processor interface
    - EDK pCore
      AXI interface based on AXI4-Lite specification
    - General Purpose Processor
    - Constant Interface
    - Transparent Interface
- 8-, 10-, and 12-bit input and output precision
- Automatic detection of timing parameters and timing signal polarities

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex®-7, Kintex®-7, Virtex-6, Spartan®-6
Supported User Interfaces	General Processor Interface, EDK pCore AXI4-Lite, Constant Interface, Transparent Interface
Resources	See Table 1-1 through Table 1-4.
Provided with Core	
Documentation	Product Specification
Design Files	Netlists, EDK pCore files, C drivers
Example Design	Not Provided
Test Bench	VHDL [2]
Constraints File	Not Provided
Simulation Models	VHDL, Verilog Structural, C Model [2]
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation[3]	Mentor Graphics ModelSim, Xilinx® ISim 13.3
Synthesis Tools	Xilinx Synthesis Technology (XST) 13.3
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C Model available on the product page on Xilinx.com at http://www.xilinx.com/products/ipcenter/EF-DI-CFA.htm
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Chapter 1

Overview

Images captured by a CMOS / CCD image sensor are monochrome in nature. To generate a color image, three primary colors (Red, Green, Blue or Cyan, Magenta, Yellow) 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 set of colored micro-lenses 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.

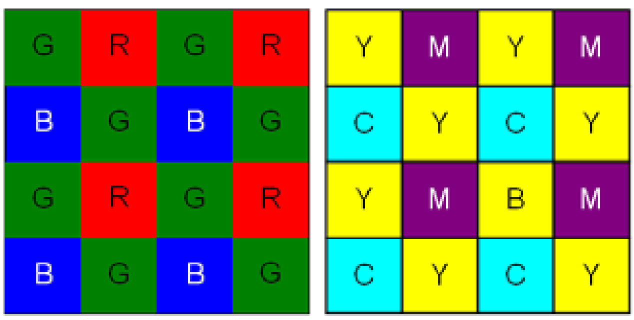


Figure 1-1: **RGB and CMY Bayer CFA Patterns**

The original data for each pixel only contains information about one color, depending 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 of the 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 could take place, which leads to aliasing between color channels.

Perfect recovery of the original signal is not 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 Bi-cubic Interpolation have been widely used for CFA demosaicing. However, simple methods usually compromise quality, and more elaborate methods require the use of an external frame buffer. The Xilinx Color Filter Array

Interpolation LogiCORE IP was designed to efficiently suppress interpolation artifacts, such as the zipper and color aliasing effects, by minimizing Chrominance Variances in a 5x5 neighborhood (Figure 1-2).

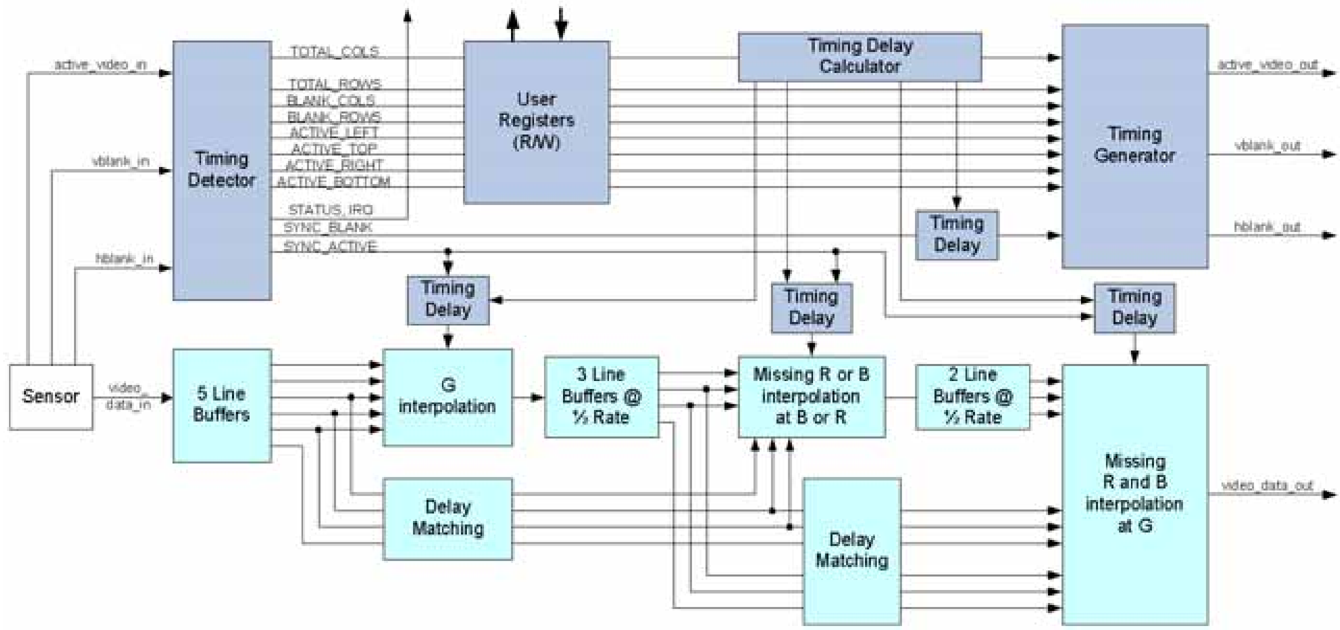


Figure 1-2: **Xilinx Color Filter Array Interpolation Block Diagram**

Image sensors that incorporate either Bayer RGB or CMY [Ref 1] Color Filters with all possible phase combinations are supported by the Xilinx Color Filter Array Interpolation LogiCORE IP.

The Xilinx Color Filter Array Interpolation LogiCORE IP also enables the user to couple the image sensor to downstream processing modules. The built-in timing detector module measures timing parameters of the input video stream, such as the total number of rows and columns, blank rows and columns, and makes the measurement results accessible through an EDK or general processor interface. A built-in, programmable timing generator module can create `hblank`, `vblank` and `active video` signals based on the user-provided parameters, and then use these signals to re-frame the input video data-stream. This module enables one to change the position of blanked regions as well as to crop the active area. However, the CFA Interpolation block cannot change the input/output image sizes, the input and output pixel clock rates, or the total image size.

Timing parameters illustrated in this figure are as follows:

TOTAL_ROWS = 64

TOTAL_COLS = 64

BLANK_ROWS = 20

ACTIVE_TOP = 10

ACTIVE_BOTTOM = 42

BLANK_POLARITY_IN = 3

The non-blanked horizontal area can be flush with the active area:

(ACTIVE_LEFT = BLANK_LEFT; ACTIVE_RIGHT= BLANK_RIGHT)

If the particular image sensor targeted does not provide the `active_video` signal, a signal driving the `active_video_in` port can be created as:

`active_video_in` = (`hblank_in` XNOR `hblank_polarity`) AND (`vblank_in` XNOR vblank_polarity)

where the `blank_polarity` signals designate whether the horizontal and vertical blanking signals are active high (1), or active low (0) as defined in Blanking Signal Polarities.

Standards Compliance

The Color Filter Array Interpolation core is compliant with the AXI4-Lite interconnect standard as defined in the AXI Reference Guide (UG761).

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 4096 columns by 4096 rows 8, 10, or 12 bits per pixel and supports the bandwidth necessary for High-definition (1080p60) resolutions.

You can generate the core as an EDK pCore (AXI4-Lite interconnect), a generic General Purpose Processor interface where all the user register connections are exposed as ports to the core, a constant interface where there is no processor connection and the values of the timing signals are known before you generate the core and with a transparent interface where there is no processor interface and the values of the timing signals do not need to be known before the core is generated.

Applications

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

Licensing

The Color Filter Array Interpolation core provides the following three licensing options:

- Simulation Only
- Full System Hardware Evaluation
- Full

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

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess core functionality with either the example design provided with the Color Filter Array Interpolation core, or alongside your own design and demonstrates the various interfaces to the core in simulation. (Functional simulation is supported by a dynamically generated HDL structural model.)

No action is required to obtain the Simulation Only Evaluation license key; it is provided by default with the Xilinx CORE Generator software.

Full System Hardware Evaluation

The Full System Hardware Evaluation license is available at no cost and lets you fully integrate the core into an FPGA design, place-and-route the design, evaluate timing, and perform functional simulation of the Color Filter Array Interpolation core using the example design and demonstration test bench provided with the core.

In addition, the license key lets you generate a bitstream from the placed and routed design, which can then be downloaded to a supported device and tested in hardware. The core can be tested in the target device for a limited time before timing out (resetting to default values and the output video becoming black), at which time it can be reactivated by reconfiguring the device.

The timeout period for this core is set to approximately 8 hours for a 74.25 MHz clock. Using a faster or slower clock changes the timeout period proportionally. For example, using a 150 MHz clock results in a timeout period of approximately 4 hours.

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

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

Full

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

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

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

Installing Your License File

The Simulation Only Evaluation license key is provided with the ISE CORE Generator system and does not require installation of an additional license file. For the Full System Hardware Evaluation license and the Full license, an email will be sent to you containing instructions for installing your license file. Additional details about IP license key installation can be found in the ISE Design Suite Installation, Licensing and Release Notes document. any section of the design labeled *DO NOT MODIFY*.

Ordering Information

The Color Filter Array Interpolation core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator system v13.1 or higher. The CORE Generator system is shipped with Xilinx ISE Design Suite development software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, please visit the Color Filter Array Interpolation product page.

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

Performance

The following sections detail the performance characteristics of the Color Filter Array Interpolation v3.0 core.

Maximum Frequencies

The following are 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.

- Virtex-7 FPGA: 303 MHz
- Kintex-7 FPGA: 303 MHz

- Virtex-6 FPGA: 225 MHz
- Spartan-6 FPGA: 150 MHz

Throughput

The Color Filter Array Interpolation core produces as much data as it consumes. If timing constraints are met, the throughput is equal to the rate at which video data is written into the core.

In numeric terms, 1080P / 60 YC4:2:2 represents an average data rate of 124.4 MPixels / sec, or a burst data rate of 148.5 MPixels / sec.

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 1-1 through Table 1-4 is a guide to the resource utilization of the Color Correction Matrix core for all input / output width combinations for Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. The Xtreme DSP Slice count is always 9, regardless of parameterization, and this core does not use any block RAMs, dedicated I / O, or CLK resources. The design was tested using ISE® v13.3 tools with default tool options for characterization data.

Table 1-1: **Spartan-6**

Data Width	Max Cols / Rows	LUT-FF Pairs	LUTs	FFs	RAM 16 / 8	DSP48A1	Fmax (MHz)
8	1023	5296	4949	3973	10 / 2	8	180
	2200	5659	5291	4149	26 / 0	8	173
10	1023	6107	5792	4578	12 / 1	8	173
	2200	6633	6176	4744	30 / 2	8	165
12	1023	5053	4519	5161	12 / 2	8	165
	2200	5357	4817	5380	36 / 0	8	159

1. SPEEDFILE: xc6slx100-3 fgg484

Table 1-2: **Virtex-7**

Data Width	Max Cols / Rows	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	1023	3817	3344	3868	3 / 6	8	300
	2200	4220	3617	4038	8 / 9	8	274
10	1023	4431	3990	4452	4 / 5	8	300
	2200	4717	4267	4622	11 / 8	8	274

Table 1-2: **Virtex-7 *(Cont'd)***

Data Width	Max Cols / Rows	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
12	1023	5203	4457	5037	5 / 4	8	291
	2200	5523	4705	5207	13 / 8	8	230

1. SPEEDFILE: xc7vx330t-2 ffg1157

Table 1-3: **Virtex-6**

Data Width	Max Cols / Rows	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	1023	3761	3402	3871	3 / 6	8	268
	2200	4150	3655	4039	11 / 8	8	283
10	1023	4481	3994	4455	4 / 5	8	268
	2200	4789	4450	4634	11 / 8	8	276
12	1023	4961	4497	5037	5 / 4	8	261
	2200	5072	4738	5207	13 / 8	8	283

1. SPEEDFILE: xc6vlx75t-2 ff484

Table 1-4: **Kintex-7**

Data Width	Max Cols / Rows	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48E1	Fmax (MHz)
8	1023	3883	3330	3868	3 / 6	8	295
	2200	3993	3659	4038	8 / 9	8	295
10	1023	4675	3897	4452	4 / 5	8	275
	2200	4800	4265	4622	11 / 8	8	282
12	1023	5213	4436	5037	5 / 4	8	289
	2200	5599	4637	5207	13 / 8	8	201

1. SPEEDFILE: xc7k70t-2 fbg676

Core Interfaces and Register Space

Port Descriptions

The Color Filter Array Interpolation core can be configured with four different interface options, each resulting in a slightly different set of ports. The core uses a set of signals that is common to all of the Xilinx Video IP cores called the Xilinx Streaming Video Interface (XSVI). The XSVI signals are common to all interface options and are shown in Figure 2-2 and described by Table 2-6.

Xilinx Streaming Video Interface

The Xilinx Streaming Video Interface (XSVI) is a set of signals common to all of the Xilinx video cores used to stream video data between IP cores. XSVI is also defined as an Embedded Development Kit (EDK) bus type so that the tool can automatically create input and output connections to the core. This definition is embedded in the pCORE interface provided with the IP, and it allows an easy way to cascade connections of Xilinx Video Cores. The Color Filter Array Interpolation core uses the following subset of the XSVI signals:

- `video_data`
- `vblank`
- `hblank`
- `active_video`

Other XSVI signals on the XSVI input bus, such as `video_clk`, `vsync`, `hsync`, `field_id`, and `active_chr` do not affect the function of this core.

Note: These signals are neither propagated, nor driven on the XSVI output of this core.

The following is an example EDK Microprocessor Peripheral Definition (.MPD) file definition. DWIDTH is the value you selected when you generated the IP in CORE Generator (i.e., 8, 10, or 12).

Input Side:

```
BUS_INTERFACE BUS = XSVI_CFA_IN, BUS_STD = XSVI, BUS_TYPE = TARGET

PORT active_video_in   = active_video,                 BUS = XSVI_CFA_IN, DIR = IN
PORT hblank_in         = hblank,                       BUS = XSVI_CFA_IN, DIR = IN
PORT vblank_in         = vblank,                       BUS = XSVI_CFA_IN, DIR = IN
PORT video_data_in     = video_data, VEC=[0:(DWIDTH-1)], BUS = XSVI_CFA_IN, DIR = IN
```

Output Side:

```
BUS_INTERFACE BUS = XSVI_CFA_OUT, BUS_STD = XSVI, BUS_TYPE = INITIATOR
```

```
PORT active_video_out = active_video,                    BUS = XSVI_CFA_OUT, DIR = OUT
PORT hblank_out       = hblank,                          BUS = XSVI_CFA_OUT, DIR = OUT
PORT vblank_out       = vblank,                          BUS = XSVI_CFA_OUT, DIR = OUT
PORT video_data_out   = video_data,VEC=[0:((DWIDTH*3)-1)],BUS = XSVI_CFA_OUT, DIR=OUT
```

The Color Filter Array Interpolation IP core is fully synchronous to the core clock, clk. Consequently, the input XSVI bus is expected to be synchronous to the input clock, clk. Similarly, to avoid clock resampling issues, the output XSVI bus for this IP is synchronous to the core clock, clk. The `video_clk` signals of the input and output XSVI buses are not used.

Constant Interface

Constant Interface, caters to those who want to interface to a particular image sensor with known, stationary timing parameters and Bayer Phase. Once the timing parameters are established and verified, typically by inserting a prototype CFA core instance with the EDK or General Purpose Processor interface into the user design, the timing parameters can be hard coded into a CFA core with a constant interface via the CFA core GUI. The processor interface and some of the timing detector module are trimmed from the design, leading to savings in FPGA logic resources. Since there is no processor interface generated, the core is not programmable, but can be reset, enabled, or disabled using the `sclr` and `ce` pins.

This interface does not provide additional programmability, the Constant Interface has no ports other than the Xilinx Streaming Video Interface, `clk`, `ce`, `sclr`, and `irq` signals. The Constant Interface Core Symbol is shown in Figure 2-2.

The Constant Interface option caters to those who want to interface to a particular image sensor with known, stationary timing parameters and Bayer Phase. Once the timing parameters are established and verified, typically by inserting a prototype CFA core instance with the EDK or General Purpose Processor interface into the user design, the timing parameters can be hard coded into a CFA core with a constant interface via the CFA core GUI. The processor interface and some of the timing detector module are trimmed from the design, leading to savings in FPGA logic resources. Since there is no processor interface generated, the core is not programmable, but can be reset, enabled, or disabled using the `sclr` and `ce` pins. The timing parameter values can be measured either by using a Color Filter Array Interpolation IP Core instance with a processor interface, or captured from the data-sheet of the image sensor. For more information on the definition of timing parameters, see Definition of Timing Parameters.

Transparent Interface

The Transparent Interface does not require any a-priori timing information from the image sensor used other than the maximum number of rows and columns (including blank rows and columns). The built-in timing detector feeds the measured timing parameters directly to the timing generator, as if the user connected the timing output ports of the General Purpose Processor Interface to the timing input ports, in a transparent manner. However, version 3.0 of the Color Filter Array core does not contain automatic Bayer Phase detection circuitry; therefore the Bayer Phase has to be supplied through the GUI in generation time. There is no processor interface of any kind generated, and the core is not programmable but can be reset, enabled / disabled using the `sclr` and `ce` pins.

General Purpose Processor Interface

The General Purpose Processor Interface exposes the timing registers as ports enabling developers designing a system with a user-defined bus to an arbitrary processor (Table 2-2). The function of the registers is identical to those described in Table 2-1.

Double-buffering is also supported by the General Purpose Processor Interface; however the first set of registers, which are typically part of the bus decoding logic, have to be supplied by the user-defined bus interface. Values from this register bank (external to the CFA core) are copied over to the internal registers when `vblank_in` becomes inactive after the user committed the changes by setting bit 1 (`REG_UPDATE`) of the control input to '1'. Before the commit, the CFA core is using the values measured by the timing detector to generate output timing signals. The measured values can be accessed via dedicated timing outputs (see Figure 2-3).

Similarly, output port values reflect working register values actively used by the core. Working registers contain measurement data from the timing detector module until the user performs a successful register update which copies over input port values to the working registers.

EDK pCore Interface

Many imaging applications have an embedded processor that can dynamically control the parameters in the core. The developer can select an EDK pCore interface, which creates a pCore that can be added to an EDK project as a hardware peripheral. This pCore provides a memory-mapped interface for the programmable registers in the core, which are described in Table 2-1.

The EDK Interface generates additional AXI 4-Lite Bus interface ports besides the Xilinx Streaming Video Interface (XSVI), `clk`, `ce`, and `sclr` signals (Figure 2-4). The AXI 4-Lite bus signals are automatically connected when the generated pCore is inserted into an EDK project. For more information on these signals, see [Ref 3]. The XSVI is described in the Xilinx Streaming Video Interface section.

Table 2-1: **EDK pCore Interface Register Descriptions**

Address Offset	Read-Write	Name	Description
0x00000000	R/W	cfa_reg_00_control	General control register. Default value is 1.
0x00000004	R/W	cfa_reg_01_reset	Software reset register. Default value is 0.
0x00000008	R	cfa_reg_02_status	General status register.
0x0000000C	R/W	cfa_reg_03_interrupt_control	Interrupt control register
0x00000010	R/W	cfa_reg_04_active_left	User defined value for ACTIVE_LEFT(1)
0x00000014	R/W	cfa_reg_05_active_right	User defined value for ACTIVE_RIGHT(1)
0x00000018	R/W	cfa_reg_06_active_top	User defined value for ACTIVE_TOP(1)
0x0000001C	R/W	cfa_reg_07_active_bottom	User defined value for ACTIVE_BOTTOM(1)
0x00000020	R/W	cfa_reg_08_total_rows	User defined value for TOTAL_ROWS(1)
0x00000024	R/W	cfa_reg_09_total_cols	User defined value for TOTAL_COLS(2)
0x00000028	R/W	cfa_reg_10_blank_rows	User defined value for BLANK_ROWS(1)
0x0000002C	R/W	cfa_reg_11_blank_left	User defined value for BLANK_LEFT(1)

Table 2-1: **EDK pCore Interface Register Descriptions *(Cont'd)***

0x00000030	R/W	cfa_reg_12_blank_right	User defined value for BLANK_RIGHT[(1)]
0x00000034	R/W	cfa_reg_13_blank_polarity	User defined polarity values for Vertical (Bit 1) and Horizontal (Bit 0) Blanking. 0: indicates blanking (active low) signal 1: indicates valid video (active high) signal
0x00000038	R/W	cfa_reg_14_bayer_phase	User defined register to specify the Bayer grid. Bits 0 (bayer_phase_x) and 1 (bayer_phase_x) specify whether the top-left corner of the Bayer sampling grid starts with a Green, Red or Blue pixel.
0x0000003C	R	cfa_reg_15_active_left_r	ACTIVE_LEFT[(1)] value measured by the core
0x00000040	R	cfa_reg_16_active_right_r	ACTIVE_RIGHT[(1)] value measured by the core
0x00000044	R	cfa_reg_17_active_top_r	ACTIVE_TOP[(1)] value measured by the core
0x00000048	R	cfa_reg_18_active_bottom_r	ACTIVE_BOTTOM[(1)] value measured by the core
0x0000004C	R	cfa_reg_19_total_rows_r	TOTAL_ROWS[(2)] value measured by the core
0x00000050	R	cfa_reg_20_total_cols_r	TOTAL_COLS[(2)] value measured by the core
0x00000054	R	cfa_reg_21_blank_rows_r	BLANK_ROWS[(1)] value measured by the core
0x00000058	R	cfa_reg_22_blank_cols_r	BLANK_LEFT[(1)] value measured by the core
0x0000005C	R	cfa_reg_23_blank_cols_r	BLANK_RIGHT[(1)] value measured by the core
0x00000060	R	cfa_reg_24_blank_polarity_r	Measured blank polarity for Vertical (Bit 1) and Horizontal (Bit 0) Blanking. 0: indicates blanking (active low) signal 1: indicates valid video (active high) signal

1. Counting of rows and columns start from 0, that is, if the first pixel of the first line is active, both ACTIVE_LEFT and ACTIVE_TOP will be equal to 0.
2. Counting of total rows and columns starts from 1. For example, if rows 0 - 499 are non-blank, and 500-599 are blank, there are TOTAL_ROWS = 600 lines in the frame.

All of the Write registers are also readable, enabling the user to verify writes or read back current values. Default values of timing registers are defined in the Graphical User Interface (GUI).

Control Register

Table 2-2 contains the Control Register descriptions.

Table 2-2: **Control Register Description**

Bit	Name	Function
0	SW_ENABLE	Software Enable Register. '0' effectively disables the core halting further operations, which blocks the propagation of all video signals. The default value of SW enable is 1 (enabled).
1	REG_UPDATE	Host processor write done semaphore. '1' indicates the host processor has finished updating timing registers, which are ready to be copied over at the next V_SYNC signal. (See General EDK Programming Guidelines)
2	CLEAR_STAT	'1' clears flags in the status registers (clears interrupt source).

Software Reset Register

Table 2-3 contains the Software Reset Register descriptions.

Table 2-3: **Software Reset Register Description**

Bit	Name	Function
0	SW_RESET	Software Reset Register. The default value of SW_RESET is 0.

The core can be effectively reset in-system by asserting the software reset (bit 0), which returns the timing registers to their default values, specified through the GUI when the core is instantiated. The core outputs are also forced to 0 until the SW_RESET bit is deasserted.

Status Register

Table 2-4 provides the Status Register descriptions.

Table 2-4: **Status Register Descriptions**

Bit	Name	Function
0-6	-	Reserved
7	TIMING_LOCKED	'1' indicates that the timing module of the core has locked on the input timing signals and is generating stable output timing signals
8	VSYNC_DET	Vertical Sync detected
9	VSYNC_ERR	Vertical Sync error (TOTAL_ROWS larger than *MAX_ROWS* parameter)
10	HSYNC_ERR	Horizontal Sync error (*TOTAL_COLS* larger than *MAX_COLS* parameter)
11	VBLANK_CHG	VBLANK POLARITY changed since last vblank_in falling edge(1)
12	HBLANK_CHG	HBLANK POLARITY changed since last vblank_in falling edge(1)
13	TROWS_CHG	TOTAL_ROWS changed since last vblank_in falling edge(1)
14	TCOLS_CHG	TOTAL_COLS changed since last vblank_in falling edge(1)
15	BROWS_CHG	BLANK_ROWS changed since last vblank_in falling edge(1)
16	BCOLS_CHG	BLANK_COLS changed since last vblank_in falling edge(1)

1. Assumes that vblank_in is active high.

Interrupt Control Register

Table 2-5 provides the Control Register descriptions.

Table 2-5: **Interrupt Control Register Descriptions**

Bit	Name	Function
0	INT_EN	Enable / Disable Interrupts
1	CLR_SRC	Clear interrupt sources
2-7	-	Reserved
8	VSYNC_DET_INT	‘1’ enables rising VSYNC_DET to request interrupt
9	VSYNC_ERR_INT	‘1’ enables rising VSYNC_ERR to request interrupt
10	HSYNC_ERR_INT	‘1’ enables rising HSYNC_ERR to request interrupt

Table 2-5: **Interrupt Control Register Descriptions** ***(Cont'd)***

11	VBLANK_CHG_INT	'1' enables rising VBLANK_CHG to request interrupt
12	HBLANK_CHG_INT	'1' enables rising HBLANK_CHG to request interrupt
13	TROWS_CHG_INT	'1' enables rising TROWS_CHG to request interrupt
14	TCOLS_CHG_INT	'1' enables rising TCOLS_CHG to request interrupt
15	BROWS_CHG_INT	'1' enables rising BROWS_CHG to request interrupt
16	BCOLS_CHG_INT	'1' enables rising BCOLS_CHG to request interrupt

If multiple bits of the Interrupt Control Register are set to 1, the interrupt service routine has to determine the source of the interrupt by polling the Status Register. To facilitate subsequent interrupts by the same event, the interrupt service routine has to clear the interrupt source in the Status Register.

If multiple bits of the Interrupt Control Register are set to 1, the interrupt service routine has to determine the source of the interrupt by polling the Status Register. To facilitate subsequent interrupts by the same event, the interrupt service routine has to clear the interrupt source in the Status Register.

Timing Registers 0x0000000C - 0x00000028

Registers `ACTIVE_LEFT`, `ACTIVE_RIGHT`, `TOTAL_COLS`, and `BLANK_COLS` take unsigned integers smaller than generic core variable `MAX_CO`. For example, if `MAX_COLS` is defined as 1024, then the registers accept 10-bit unsigned integers.

Registers `ACTIVE_TOP`, `ACTIVE_BOTTOM`, `TOTAL_ROWS`, and `BLANK_ROWS` take unsigned integers smaller than generic core variable `MAX_ROWS`. For example, if `MAX_ROWS` is defined as 1024, then the registers accept 10-bit unsigned integers.

Bayer Phase Register

Bits 0 (`bayer_phase_x`) and 1 (`bayer_phase_y`) specify whether the top-left corner of the Bayer sampling grid starts with Green, Red, or Blue Pixel, according to Figure 2-1, which displays top-left corner of the imager sample matrix along with the Bayer Phase Register value combinations.

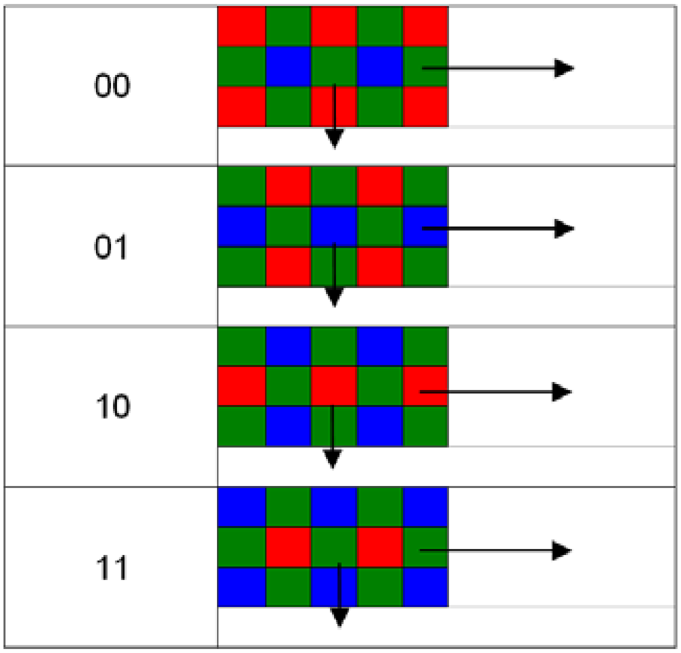


Figure 2-1: **Bayer Phase Register Combination Definitions**

Transparent Interface

This interface option is the easiest to use and is recommended for the user who is not interested in reading out or modifying the timing parameters. This interface does not require any timing information from the image sensor used. The built-in timing detector feeds the measured timing parameters directly to the timing generator, as if the user connected the timing output ports of the General Purpose Processor Interface to the timing input ports, in a transparent manner. The Transparent Interface has no ports other than the Xilinx Streaming Video Interface, clk, ce, sclr, and irq signals. The Constant Interface Core Symbol is shown in Figure 2-2.

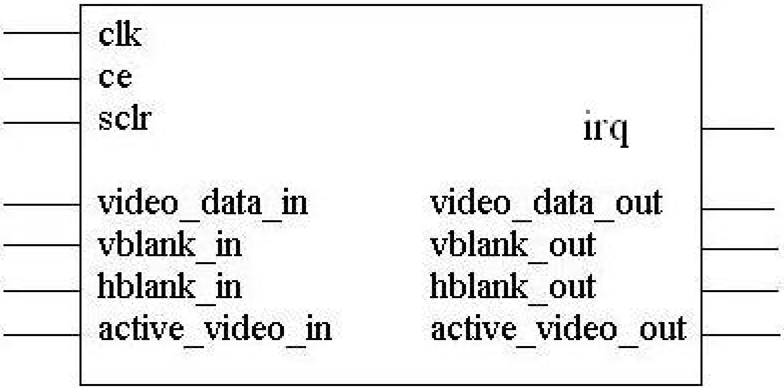


Figure 2-2: **Core Symbol for Constant and Transparent Interfaces**

Table 2-6: **Port Descriptions for the Constant and Transparent Interfaces**

Port Name	Port Width	Direction	Description
video_data_in	DWIDTH	IN	data input bus
hblank_in	1	IN	horizontal blanking input
vblank_in	1	IN	vertical blanking input
active_video_in	1	IN	active video signal input
video_data_out	3* DWIDTH	OUT	data output bus
hblank_out	1	OUT	horizontal blanking output
vblank_out	1	OUT	vertical blanking output
active_video_out	1	OUT	active video signal output
irq	1	OUT	interrupt request pin
clk	1	IN	rising-edge clock
ce	1	IN	clock enable (active high)
sclr	1	IN	synchronous clear – reset (active high)

- **video_data_in**: This is the sample input bus for Bayer patterned data. DWIDTH bits wide color values are expected in unsigned integer representation.
- **hblank_in**. This signal conveys information about the blank / non-blank regions of video scan lines.
- **vblank_in:** This signal conveys information about the blank / non-blank regions of video frames.
- **active_video_in**: This signal is high when valid data is presented at the input.
- **clk - clock:** Master clock in the design, synchronous to, or identical with video clk.
- **ce - clock enable**: Pulling `CE` low suspends all operations within the core. Outputs are held, no input signals are sampled, except for reset (`SCLR` takes precedence over `CE`).
- **sclr - synchronous clear:** Pulling `SCLR` high results in resetting all output pins to zero. Internal registers within the XtremeDSP™ slice and D-flip-flops are cleared. However, the core uses SRL16 / SRL32-based delay lines for `hblank`, `vblank`, and `active_video` generation, which are not cleared by `SCLR`. This may result in non-zero outputs after `SCLR` is deasserted, until the contents of SRL16 / SRL32s are flushed. Unwanted results can be avoided if `SCLR` is held active until SRL16 / SRL32s are flushed.
- **video_data_out:** This bus contains RGB output in the same order as `video_data_in`. Color values are represented as DWIDTH bits wide unsigned integers.

Bits	3DWIDTH-1:2DWIDTH	2DWIDTH-1:DWIDTH	DWIDTH-1:0
video data signals	Red / Magenta	Blue / Cyan	Green / Yellow

- **hblank_out, vblank_out and active_video_out:** The corresponding input signals are delayed so `active_video` and blanking outputs are in phase with the video data output, maintaining the integrity of the video stream. The `active_video_out` signal is high when valid data is presented at the output.

- **irq:** The Interrupt output pin can be used in a processor system to signal special conditions detected by the CFA core. For more information on interrupt subsystems, see Using the Interrupt Subsystem. For a complete list of events that can be monitored, see Interrupt Control Register.

General Purpose Processor Interface

Figure 2-3 shows the core pinout for the General Purpose Processor Interface; Table 2-7 provides descriptions for its pins in addition to the pins defined in Table 2-6.

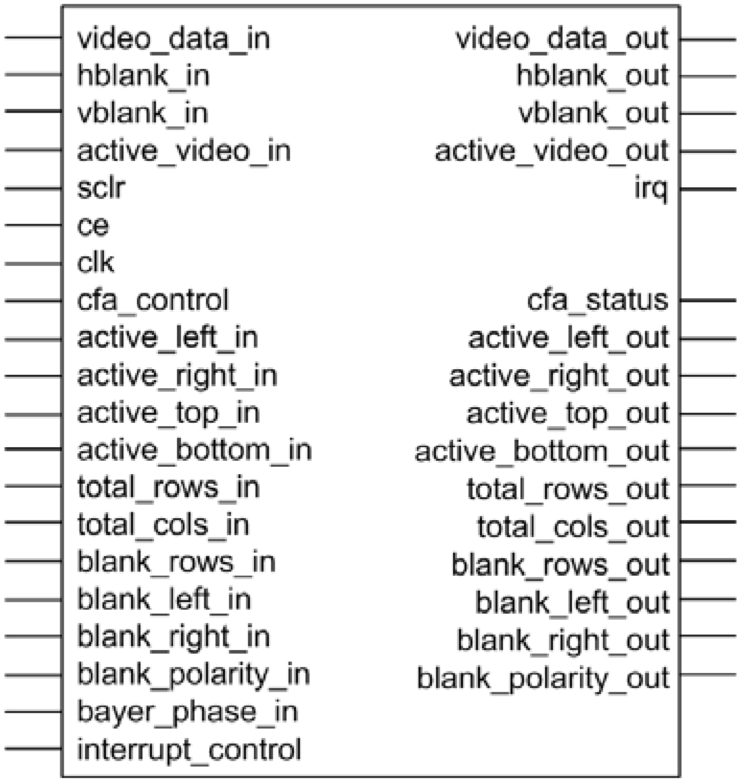


Figure 2-3: **Core Pinout for General Purpose Processor Interface**

Table 2-7: **Optional Pins for the General Purpose Processor Interface**

Port Name	Port Width	Direction	Description
control	4	IN	Bit 0: Software Enable Register Bit 1: Host processor write done semaphore Bit 2: Clear status registers (clears interrupt source) Bit 3: Reserved
status	18	OUT	Status Register
active_left_in	COLS_WIDTH	IN	User defined value for ACTIVE_LEFT(1)
active_right_in	COLS_WIDTH	IN	User defined value for ACTIVE_RIGHT(1)
active_top_in	ROWS_WIDTH	IN	User defined value for ACTIVE_TOP(1)
active_bottom_in	ROWS_WIDTH	IN	User defined value for ACTIVE_BOTTOM(1)

Table 2-7: **Optional Pins for the General Purpose Processor Interface *(Cont'd)***

total_rows_in	COLS_WIDTH	IN	User defined value for TOTAL_ROWS[(1)]
total_cols_in	COLS_WIDTH	IN	User defined value for TOTAL_COLS[(1)]
blank_rows_in	ROWS_WIDTH	IN	User defined value for BLANK_ROWS[(1)]
blank_left_in	COLS_WIDTH	IN	User defined value for BLANK_LEFT[(1)]
blank_right_in	COLS_WIDTH	IN	User defined value for BLANK_RIGHT[(1)]
blank_polarity_in	2	IN	User defined input timing blank polarities for Vertical (Bit 1) and Horizontal (Bit 0) Blanking 0: indicates blanking (active low) signal 1: indicates valid video (active high) signal
bayer_phase	2	IN	See Bayer Phase Register
interrupt_control	17	IN	See section Using the interrupt subsystem
active_left_out	COLS_WIDTH	OUT	Input timing value measured for ACTIVE_LEFT*
active_right_out	COLS_WIDTH	OUT	Input timing value measured for ACTIVE_RIGHT[(1)]
active_top_out	ROWS_WIDTH	OUT	Input timing value measured for ACTIVE_TOP[(1)]
active_bottom_out	ROWS_WIDTH	OUT	Input timing value measured for ACTIVE_BOTTOM[(1)]
total_rows_out	COLS_WIDTH	OUT	Input timing value measured for TOTAL_ROWS[(1)]
total_cols_out	COLS_WIDTH	OUT	Input timing value measured for TOTAL_COLS[(1)]
blank_rows_out	ROWS_WIDTH	OUT	Input timing value measured for BLANK_ROWS[(1)]
blank_left_out	COLS_WIDTH	OUT	Input timing value measured for BLANK_LEFT[(1)]
blank_right_out	COLS_WIDTH	OUT	Input timing value measured for BLANK_RIGHT[(1)]
blank_polarity_out	2	OUT	User defined output timing blank polarities for Vertical (Bit 1) and Horizontal (Bit 0) Blanking 0: indicates blanking (active low) signal 1: indicates valid video (active high) signal

1. See Definition of Timing Parameters, page 26
2. See Blanking Signal Polarities, page 26

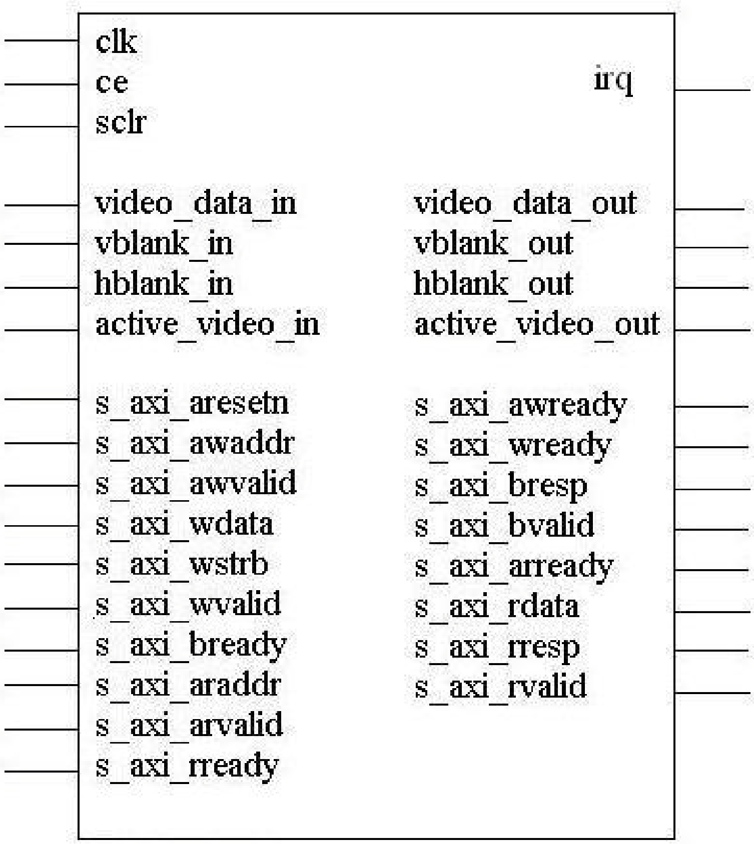


Figure 2-4: **Core Pinout for the EDK Processor Interface**

Chapter 3

Customizing and Generating the Core

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

Graphical User Interface (GUI)

The Xilinx Color Filter Array Interpolation core is easily configured to meet the developer's specific needs through the CORE Generator™ GUI. This section provides a quick reference to parameters that can be configured at generation time. Figure 3-1 shows the main Color Filter Array Interpolation screen.

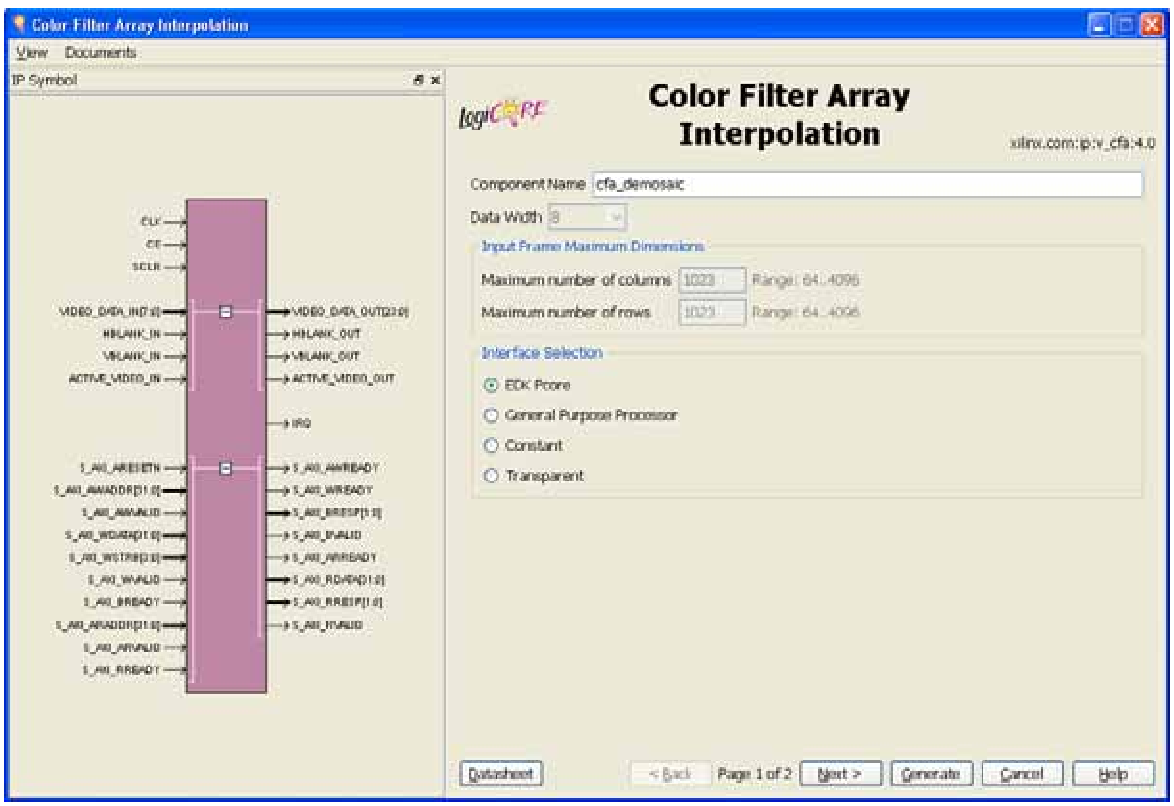


Figure 3-1: **Color Filter Array Interpolation Main Screen**

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 "_".

- **Data Width (DWIDTH):** Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.
- **Maximum Number of Columns (MAX_COLS)**: Specifies the maximum number of columns that can be processed by the core. Permitted values are from 128 to 4096. Specifying this value is necessary to establish the internal widths of counters and control-logic components as well as the depth of line buffers. Using a tight upper-bound on possible values of `TOTAL_COLS` results in optimal block RAM usage. However, feeding the configured CFA instance timing signals which violate the `MAX_COLS` constraint will lead to data-, and output timing signal corruption and is flagged by the status register.
- M**aximum Number of Rows (MAX_ROWS)**: Specifies the maximum number of rows that can be processed by the core. Permitted values are from 128 to 4096. Specifying this value is necessary to establish the internal widths of counters and control-logic components. Feeding the configured CFA instance timing signals which violate the `MAX_ROWS` constraint will lead to data-, and output timing signal corruption and is flagged by the status register.
- **Interface Selection:** As described in the previous sections, this option allows for the configuration of four different interfaces for the core.
  - **EDK pCore Interface**: CORE Generator software will generate a pCore which can be easily imported into an EDK project as a hardware peripheral. Internal timing measurement values can be read out, timing parameters used can be reprogrammed, and double-buffering is used to eliminate tearing of output images.
  - **General Purpose Processor Interface**: CORE Generator software will generate a set of ports to be used to program the core.
  - **Constant Interface**: Timing parameters provided on screen 2 of the GUI are constant, and therefore no programming is necessary. The timing detector circuitry is trimmed from the design, slightly reducing the slice-count for the core.
  - **Transparent Interface**: Timing parameters are measured automatically; therefore no programming other than setting the Bayer Phase is necessary.

The **Default Names screen (**Figure 3-2**)** allows for the definition of default timing, polarity, Bayer Phase and interrupt control values. For the Constant Interface, these values are permanent for the generated CFA instance. For the Transparent Interface, Timing Initialization values are discarded.

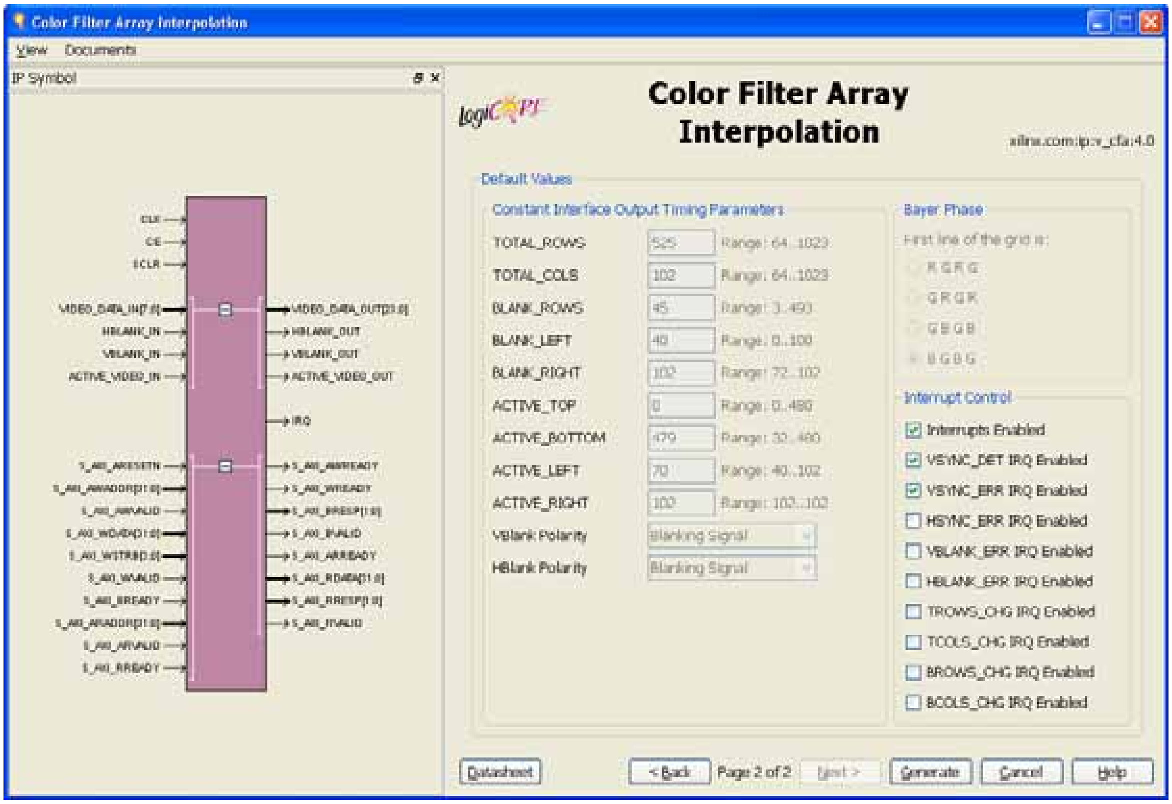


Figure 3-2: **Color Filter Array Interpolation, Default Status Screen**

- **Timing Initialization**: The timing initialization pane allows assigning default values for the output timing generator. This pane is only available when the Constant user interface is selected. For all other interface selections the IP core contains a timing detector module, which provides timing information for the output timing generator. This information is either directly driving the output timing generator (Transparent interface) or can be provided to a software driver, which can program the output timing generator. If the sensor-specific timing values have been established and are fixed for the core instance, the constant interface provides a way to save on resources by not instantiating a timing detector module, but using the established timing values provided though the CORE Generator GUI. For the definition of Timing Initialization generic parameters, see the preceding Definition of Timing Parameters section.
- **Bayer Phase**: Based on the data sheet of the particular image sensor used, and the particular register settings of the sensor, the user has to identify where the top-left corner of total area falls on the CFA matrix. For the first two samples, four combinations are possible. For RGB sensors, these are RG, GR, BG, GB. For CMY sensors the combinations are MY, YM, CY, YC.

Chapter 4

Designing with the Core

General Design Guidelines

The processor interfaces allow access to input timing information measured by the internal timing detector circuitry (Figure 1-2) and to control output timing signals by programming the built-in timing generator. From the edge transitions of the three input timing signals, the timing circuitry can measure:

- Blanking signal polarities
- Overall (total) frame dimensions
- The size and position of the non-blank area
- The size and position of the active area

Blanking Signal Polarities

Typical constituents of a video stream, blanking signals provide framing and blanking information that complements and formats image data provided via the `video_data_in` port. Image sensors provide this information by active high (data valid) signaling [Ref 1], or active low (blank) signaling.

The Xilinx Color Filter Array core is equipped with automatic detection of blanking signal polarity, based on the phase relations between the blanking and the active signals. The `active_video` input signal is assumed active high. If `active_video_in` is high during the logic high period of a blanking signal, that blanking signal is considered active high (valid signaling). If `active_video_in` is high during the logic low period of a blanking signal, the blanking signal is considered active low (blank signaling).

Note: The high portion of `active_video_in` should not extend across edges of either blanking signals.

The following definition of timing parameters assumes the `hblank_in` and `vblank_in` are driven by blanking signals, with logic high corresponding to blanked, logic low corresponding to non-blanked areas.

Definition of Timing Parameters

The periodic vblank, hblank, and `active_video` signals define the frame boundaries, as well as the blanked and active areas within a video stream. Edges of the vblank signal identify frame boundaries, and the blank / non-blank rows within frames. Edges of the hblank signal identify the blank / non-blank columns within frames, and also determine the total number of columns (`TOTAL_COLUMNS`) in the frame, which is the number of clock cycles between two rising (or falling) edges of hblank.

Note: The Color Filter Array core supports only `hblank_in` signals that are periodic through the entire frame time.

If the video stream signals were plotted line-by-line in a coordinate system scanning from left to right, top towards bottom, with the top-left corner identified by the falling edge of the `vblank` signal, the phase relationships between the vblank, hblank, and active video signals would define three rectangles: the total area containing the non-blank area, which contains the active area (Figure 4-1).

The timing parameters defining the sizes and positions of the total, non-blank and active areas can be defined as:

TOTAL_COLUMNS:	Defines the total number of columns, counting from 1, in a video frame. This is equal to the number of clk periods in a full hblank period.
TOTAL_ROWS:	Defines the total number of rows, counting from 1, in a video frame. This is equal to the number of hblank periods in a full vblank period.
BLANK_ROWS:	Defines the number of blank rows, counting from 1, in a video frame. This is equal to the number of hblank periods in the vertical blanking period.
BLANK_LEFT:	Defines the index, counting from 0, of the first non-blank column (on the left side of the active area).
NON_BLANK_COLUMNS	The number of clk periods, counting from 1, when hblank is inactive in a full hblank period.
BLANK_RIGHT:	Defines the index, counting from 0, of the first blank column on the right side of the active area. `BLANK_RIGHT=BLANK_LEFT+NON_BLANK_COLUMNS`
ACTIVE_TOP:	Defines the index of the first active row, where row 0 is at the beginning of the vertical non-blank period. The active area is typically smaller than the non-blank area, which for a typical sensor includes optically masked (inactive) pixels.
ACTIVE_LEFT	Defines the index, counting from 0, of the first active column. The active area is typically smaller than the non-blank area, which for a typical sensor includes optically masked (inactive) pixels.
ACTIVE_ROWS:	Number of `active_video` pulses in the vertical non-blanking period.
ACTIVE_COLUMNS:	Number of clock cycles between the rising and falling edges of `active_video`.
ACTIVE_RIGHT:	Defines the index, counting from 0, of the first non-active column on the right side of the active area. `ACTIVE_RIGHT=ACTIVE_LEFT+ACTIVE_COLUMNS`
ACTIVE_BOTTOM:	Defines the index of the first non-active row below the active area of the frame, where row 0 is at the beginning of the vertical non-blank period. `ACTIVE_BOTTOM = ACTIVE_TOP+ACTIVE_ROWS`

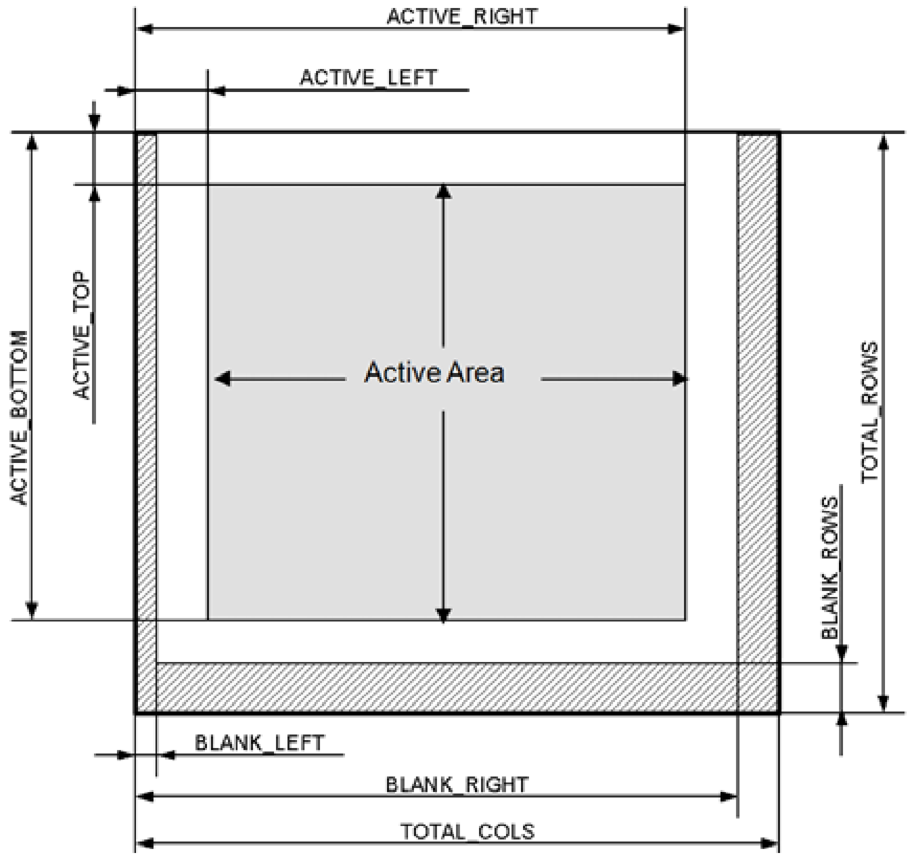


Figure 4-1: **Timing Parameters**

The logic high state of input signal `active_video_in` marks samples of `video_data_in` as valid. Although this signal could be used to mark an arbitrary region of the frame active, typical image sensors use this signal to designate a rectangle within the non-blank area as an active / valid area.

Note: The Color Filter Array Interpolation core only supports `active_video_in` signals that designate a rectangular area, are contiguous within one hblank period, and are periodic during the active region of the frame.

The top-left corner of the active area is defined by `ACTIVE_TOP` and `ACTIVE_LEFT`, which are the coordinates of the first sample marked active by `active_video_in` in the coordinate system defined by the blanking input signals. Similarly, `ACTIVE_BOTTOM` and `ACTIVE_RIGHT` are the coordinates of the last sample marked active by `active_video_in`. An example of horizontal timing and corresponding timing parameters is provided in Figure 4-2.

Timing Tolerances

Due to state-machine setup and reset constraints internal to the Xilinx Color Filter Array core, the following limitations must be observed when configuring the image sensor to be used in conjunction with the core:

- `BLANK_ROWS` > 2
- `ACTIVE_LEFT` > 3

- BLANK_LEFT <= ACTIVE_LEFT
- BLANK_LEFT + (TOTAL_COLS-BLANK_RIGHT) > 2
- ACTIVE_RIGHT < TOTAL_COLS - 5
- ACTIVE_RIGHT - ACTIVE_LEFT > 31
- BLANK_RIGHT> = ACTIVE_RIGHT
- ACTIVE_BOTTOM - ACTIVE_TOP > 31

Figure 4-2 shows an example in which these conditions are met.

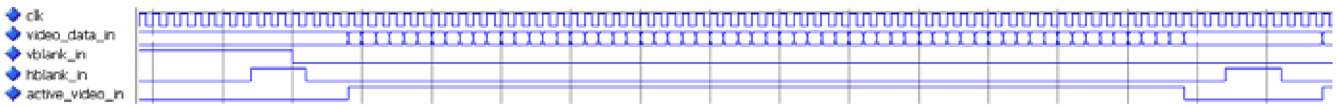


Figure 4-2: **Color Filter Array Interpolation Programming Flow Chart**

In this example, the timing parameters are as follows:

BLANK_LEFT =1

(CLK cycles between a falling edge of vblank_in and the next falling edge of hblank_in)

ACTIVE_LEFT = 4

(CLK cycles between a falling edge of vblank_in and the next rising edge of active_video_in)

ACTIVE_RIGHT = 63

(CLK cycles between a falling edge of vblank_in and the next falling edge of active_video_in)

BLANK_RIGHT = 66

(CLK cycles between a falling edge of vblank_in and the next rising edge of hblank_in)

TOTAL_COLS = 70

(CLK cycles between falling edges of hblank_in)

BLANK_POLARITY_IN = 0

(Both hblank and vblank signals in this example are active-low)

The propagation delay of the Color Filter Array Interpolation core depends on actual parameterization, but is at least four full line-times. Deasserting CE suspends processing, which may be useful for data-throttling to temporarily cease processing of a video stream to match the delay of other processing components.

The example in Figure 4-3 illustrates vertical timing for a very short video frame.

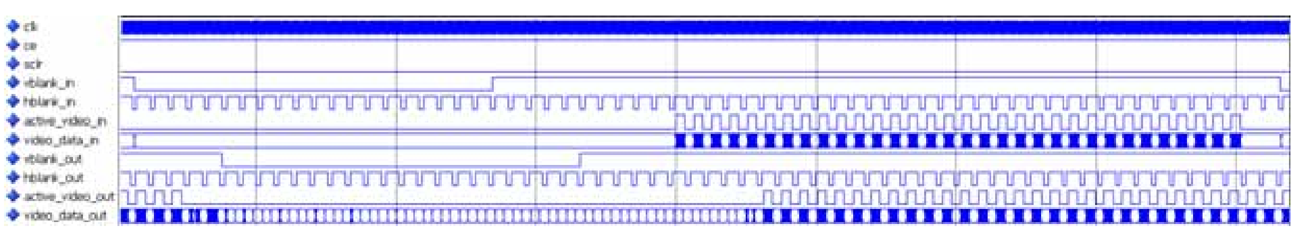
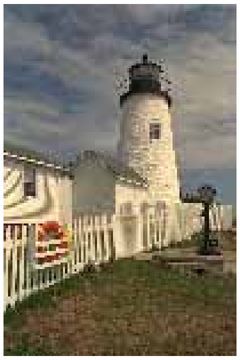


Figure 4-3: **Vertical Timing Example**

Quality Measures

Table 4-1 provides Peak Signal to Noise Ratio (PSNR) measurement results for typical test images using an 8-bit input data.

Table 4-1: **PSNR Results for Typical Test Images**

Image	PSNR [dB]
	34.051
	39.404
	33.736

Protocol Description

For the pCore version of the Color Filter Array Interpolation core, the register interface is compliant with the AXI4-Lite interface.

Constraining the Core

Required Constraints

The clk 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 frequency is the required frequency for the Color Filter Array Interpolation core. See Maximum Frequencies in Chapter 1.

Clock Management

There is only one clock for the Color Filter Array Interpolation core.

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 this top-level folder is described below:

- Expected

  Contains the pre-generated expected / golden data used by the testbench to compare actual output data.

- Stimuli

  Contains the pre-generated input data used by the testbench to stimulate the core (including register programming values).

- Results

  Actual output data will be written to a file in this folder.

- src

  Contains the .vhd & .xco files of the core.

- The .vhd file is a netlist generated using Coregen.

  You can regenerate a new netlist using the .xco file in Coregen.

- tb_src

  Contains the top-level testbench design.

  This directory also contains other packages used by the testbench.

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

Demonstration Test Bench

This demonstration test bench is provided as a simple introductory package that enables core users to observe the core generated by the CORE Generator tool operating in a waveform simulator. The user is encouraged to observe core-specific aspects in the waveform, make simple modifications to the test conditions, and observe the changes in the waveform.

Simulation

Simulation using ModelSim for Linux:

- From the console, Type "source run_mti.sh".

Simulation using ModelSim for Windows:

- Double-click on "run_mti.bat" file.

Simulation using iSim for Linux:

- Double-click on "run_isim.bat" file.

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the Object Segmentation core. Testing included the following:

- Register accesses
- Processing of multiple frames of data
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Object Segmentation core has been tested in a variety of hardware platforms 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

Migrating

Parameter Changes in the XCO File

There are no parameter changes in the XCO file.

Port Changes

Other than an AXI4-Lite interface in place of the PLB, there are no port changes.

Functionality Changes

There are no functionality changes to the core.

Special Considerations when Migrating to AXI

The Color Filter Array Interpolation core v4.0 changed from the PLB EDK pCore processor interface to the EDK pCore AXI4-Lite interface. As a result, all of the PLB-related connections have been replaced with an AXI4-Lite interface. This processor interface change does not change the functionality of the core other than an AXI4-Lite has to be used in place of the PLB. For more information about AXI4-Lite, see UG761 AXI Reference Guide.

Appendix C

Debugging

Consider the following:

- Are the input and output timing signals active_video, vblank, hblank connected?
- Is the video clock (clk) and reset (sclr) signals connected?
- Is bit 0 of the control register (BASEADDR + 0x00) set to '1'?
- Is bit 7 of the status register (BASEADDR + 0x08) set to '1'?
- Did you follow the Color Filter Array Interpolation Programming Flow Chart (Figure 2-2) to program the temporal, spatial and pixel age threshold registers?

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

Appendix D

Application Software Development

General EDK Programming Guidelines

All registers other than `control`, `status`, and `interrupt_control` registers are double-buffered to ensure no image tearing happens if values are modified in the active area of a frame. Updated values for timing registers are latched into shadow registers immediately after writing, and shadow register values are copied into the working registers when `vblank_in` becomes inactive. Double-buffering decouples register updates from the blanking period, allowing software a much larger window to update the parameter values without tearing.

After startup / reset, output timing register values (`reg_04` - `reg_13`), and internal registers controlling the output timing generator are constantly updated with values measured by the timing detector (`reg_15` - `reg_24`). If the input timing changes (e.g., as a consequence of reprogramming the image sensor), the CFA core automatically adjusts its timing, which is reflected by the timing register values. However, when the user writes to any of registers `reg_04` - `reg_13`, the core stops automatically updating `reg_04` - `reg_13`, and retains the user-provided values. For register values not modified by the user, the core retains the values in effect at the time of the first register write. User provided values are not affecting output timing generation until the changes are committed (`REG_UPDATE` bit set to '1', `vblank_in` transitions to inactive). Subsequent changes in input timing signals will not automatically change the output timing registers (`reg_04` - `reg_13`) signals until the core is reset.

Programmer's Guide

The software API is provided to allow easy access to the CFA pCore's registers defined in Table 2-1. 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_v4_00_a/example` subfolder. This file is a sample C program that demonstrates how to use the CFA pCore API.

EDK pCore API Functions

This section describes the functions included in the C driver (`cfa.c` and `cfa.h`) generated for the EDK pCore API.

CFA_Enable(uint32 BaseAddress);

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

CFA_Disable(uint32 BaseAddress);

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

CFA_Reset(uint32 BaseAddress);

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

CFA_ClearReset(uint32 BaseAddress);

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

Reading and Writing pCore Registers

Each software register defined in Table 2-1 has a constant defined in `cfa.h` that is set to the offset for that register.

Reading a value from a register uses the base address and offset for the register:

```
Xuint32 value = CFA_ReadReg(XPAR_CFA_0_BASEADDR,
CFA_REG04_ACTIVE_LEFT);
```

This macro returns the 32-bit unsigned integer value of the register. The definition of this macro is:

CFA_ReadReg(uint32 BaseAddress, uint32 RegOffset)

- Read the given register.
- BaseAddress is the Xilinx EDK base address of the CFA core (from `xparameters.h`).
- RegOffset is the register offset of the register (defined in Table 2-1).

To write to a register, use the CFA_WriteReg() function using the base address of the CFA pCore instance (from `xparameters.h`), the offset of the desired register, and the data to write. For example:

CFA_WriteReg(`XPAR_CFA_0_BASEADDR, CFA_REG04_ACTIVE_LEFT, 70`);

The definition of this macro is:

CFA_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)

- Write the given register.
- BaseAddress is the Xilinx EDK base address of the CFA core (from `xparameters.h`).
- RegOffset is the register offset of the register (defined in Table 2-1).

- Data is the 32-bit value to write to the register.

CFA_RegUpdateEnable(uint32 BaseAddress);

- Updating timing register values, calling RegUpdateEnable causes the CFA to start using the updated table to update on the next rising edge of `VBlank_in`. This action causes the new values written to the inactive look-up table to become the active look-up table when the `VBlank_in` rising edge occurs. The user must manually disable the register update after a sufficient amount of time to prevent continuous updates.
- This function only works when the CFA core is enabled.
- BaseAddress is the Xilinx EDK base address of the CFA core (from `xparameters.h`)

CFA_RegUpdateDisable(uint32 BaseAddress);

- When using a double-buffered interface, disabling the Register Update prevents the CFA correction look-up table from updating. Xilinx recommends disabling the Register Update while writing to the inactive look-up table in the CFA correction core until the write operation is complete. While disabled, writes to the inactive look up table are stored, but do not affect the core's behavior.
- This function only works when the CFA core is enabled.

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

Figure D-1 provides a software flow diagram for updating registers during the operation of the core.

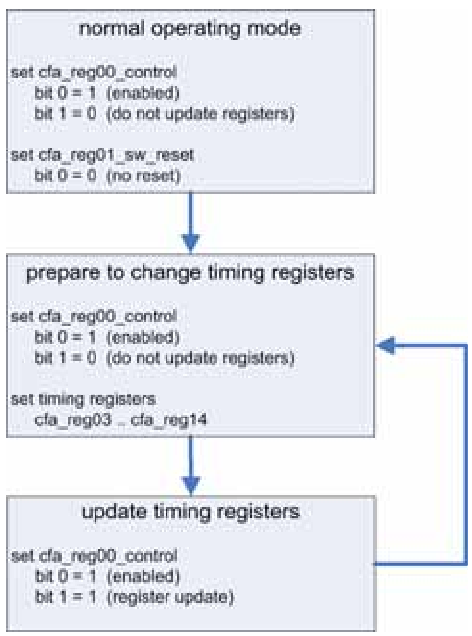


Figure D-1: **Color Filter Array Interpolator Programming Flow Chart**

Using the Interrupt Subsystem

The Color Filter Array core can signal several exceptional events to the host processor using the `irq` output.

Bits 8-16 of the status register can request an interrupt if the interrupt enable bit corresponding to the particular status bit is set to '1'.

For example, if TOTAL_COLS, established by the timing detector circuitry or entered dynamically through a processor interface, gets larger than MAX_COLS, bit 10 of the status register is set to '1'. If bit 10 of the Interrupt Enable register is also set (='1'), and the general interrupt enable flag (Interrupt Enable Register, bit 0) is also set (='1'), then the event sets the irq output to '1' as well.

For the complete list of interrupt events, see Status Register in Chapter 2.

Once the interrupt is serviced by the host processor, the processor should identify the interrupt source by polling the status register, then pulsing the clear-status flag (Bit 2 of the control register). Individual interrupts sources can be masked using the Interrupt Enable Register.

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 v4.0 C models were compiled and tested with the following software versions.

Table E-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 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_v4_0_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_v4_0_bitacc_model.zip file creates the following directory structures and files which are described inTable E-2.

Table E-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_v4_0_bitacc_cmodel.lib	Color Filter Array Interpolation v4.0 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_v4_0_bitacc_cmodel.lib	Color Filter Array Interpolation v4.0 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_v4_0_bitacc_cmodel.lib	Pre-compiled library file for win32 compilation (Windows platforms only)
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_v4_0_bitacc_cmodel.lib	Pre-compiled library file for win32 compilation (Windows platforms only)
run_bitacc_cmodel.exe	Pre-compiled bit accurate executable for simulation on 64-bit Windows Platforms
README.txt	Release notes
pg002_v_cfa.pdf	The Color Filter Array Interpolation Core Product Guide
v_cfa_v4_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I/O functions
video_utils.h	Header file declaring the generalized image / video container type, I/O and support functions.
Kodim19_128x192.bmp	128x192 sample test image of the Lighthouse image from the True-color Kodak test images
run_bittacc_cmodel.c	Example code calling the C model

Using the C-Model

The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file `v_cfa_v4_0_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_v4_0_generics cfa_generics;
struct  xilinx_ip_v_cfa_v4_0_inputs   cfa_inputs;
struct  xilinx_ip_v_cfa_v4_0_outputs  cfa_outputs;
```

Declaration of the above structs are located in the `v_cfa_v4_0_bitacc_cmodel.h` file.

The only generic parameter the CFA v4.0 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_v4_0_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`, please see Figure 4-2, page 18. For the description of the input structure, see CFA Input and Output Video Structure.

Calling `xilinx_ip_v_cfa_v4_0_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_v4_0_bitacc_simulate(
  struct xilinx_ip_v_cfa_v4_0_generics* generics,
  struct xilinx_ip_v_cfa_v4_0_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_v4_0_destroy(
  struct xilinx_ip_v_cfa_v4_0_inputs *input,
  struct xilinx_ip_v_cfa_v4_0_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 v4.0 reference model using the video_struct structure, defined in `video_utils.h`:

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

Table E-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 E-4.
d. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col]

Table E-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
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 8 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: 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 will create 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 E-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

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

After following the compilation instructions, run the example executable.

The executable takes the path / name of the input file and the path / 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
```

During successful execution, two other files with the extension 'bin', are created. The first file corresponds to the input bmp image, and has the same path and name as the input file, with extension '.bin'. 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 8 bits per component. The structure of.bin files are detailed in the section Binary Image / Video Files.

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_v4_0_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_v4_0_bitacc_cmodel -Wl,-rpath,.

gcc  -m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
run_bitacc_cmodel -L. -lIp_v_cfa_v4_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32- and 64-bit)

Precompiled library v_cfa_v4_0_bitacc_cmodel.dll, and top level demonstration code run_bitacc_cmodel.c should be compiled with an ANSI C compliant compiler under Windows. Here an example is presented using Microsoft Visual Studio.

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

- llibIpv_cfa_v4_0_bitacc_cmodel.dll to the "Resource Files" folder of the project
- run_bitacc_cmodel.c to the "Source Files" folder of the project
- v_cfa_v4_0_bitacc_cmodel.h header files to "Header Files" folder of the project (optional)

After the project has been created and populated, it needs to be compiled and linked (built) to create a win32 executable. To perform the build step, choose **Build Solution** from the Build menu. An executable matching the project name has been created either in the Debug or Release subdirectories under the project location based on whether **Debug** or **Release** has been selected in the **Configuration Manage**r under the Build menu.

Appendix F

Additional Resources

Xilinx Resources

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

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

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

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

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.

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 v4.0 core is provided under the Xilinx Core License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.

Notice of Disclaimer