LogiCORE IP Image Characterization v3.00a

Product Guide

PG015 December 18, 2012

XILINX®

XILINX

Table of Contents

SECTION IV: APPENDICES

SECTION I: SUMMARY

Introduction

The Xilinx Image Characterization LogiCORE™ IP calculates statistical information about each image frame passed to it in a video stream. This statistical data can be used by subsequent processing cores to develop sophisticated video analysis solutions including applications like face recognition and object detection. The statistics provided by this core include means and variances for luminance, chrominance, high and low frequencies, edges, and motion on both a global and block basis. The Image Characterization core supports 8-bit pixel data in YUV 4:2:2 or 4:2:0 as well as 8-bit motion data. The core is programmable through a comprehensive register interface for setting edge gains, high-pass gain, color selects (hue and saturation), and block size.

Features

- Global and Block Means and Variances for:
  - Luminance/Chrominance Content
  - Frequency Content
  - Edge Content
  - Motion Content
  - Color Content
- Global Histograms for:
  - Luminance
  - Chrominance
  - Hue
- Support for 8-bit, YUV 4:2:2 or YUV 4:2:0 data
- Support for 8-bit motion data
- Support for block sizes of 4x4, 8x8, 16x16, 32x32 or 64x64 pixels
- Supports 1920x1080p60 for block sizes of 8x8 or larger for Virtex and Kintex families.
- Support for streaming or frame buffer based processing
- AXI4-Lite support for interrupts and register access

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq ™-7000, Artix ™-7, Virtex®-7, Kintex®-7, Spartan®-6, Virtex®-6
Supported User Interfaces	AXI4-Stream (2), AXI4-Lite
Resources	See Table 2-1 through Table 2-10.
Provided with Core	
Design Files	ISE: Netlist Vivado: Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog (3)
Constraints File	Not Provided
Simulation Model	Verilog or VHDL Structural, C Model (3)
Supported S/W Driver(4)	Standalone
Tested Design Flows(5)	
Design Entry	ISE Design Suite v14.4 Vivado Design Suite v2012.4
Simulation	Mentor Graphics ModelSim, Vivado Simulator
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the release notes for this core.
2. Video protocol as defined in the Video IP: AXI Feature Adoption section of UG761 AXI Reference Guide [Ref 1].
3. C-Model available on the product page on Xilinx.com at http://www.xilinx.com/products/intellectual-property/EF-DI-IMG-CHAR.htm
4. Standalone driver details can be found in the EDK or SDK directory (<*install_directory*>/doc/usenglish/xilinx_drivers.htm).
5. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The Image Characterization LogiCORE IP is comprised of a collection of blocks that work together to calculate statistical data that can be used to describe an image in the analytics domain. These statistics are calculated on a global basis for the entire image as well as on a block basis which is implemented as a 2-D grid of NxM subdivisions of the image. The resulting image statistics are written to memory and can be read by another IP core or by software to implement complex analytics applications.

The main global and block statistical measures performed are:

- $\overline{x} = \frac{1}{M}\sum_{i=0}^{M-1} x_i$, the mean values

- $\sigma^2 = \frac{1}{M}\sum_{i=0}^{M-1} x_i^2 - \overline{x}^2$, the variance values

These statistical tools are applied to image content, including:

- Luminance/Chrominance Content
- Low Frequency Content
- High Frequency Content
- Color Content
- Edge Content
- Motion Content

In addition to the preceding statistics, histograms are also calculated on the global image. Histograms are generated for the Luminance, Chrominance (Cr and Cb), and Hue components.

The Image Characterization LogiCORE IP shares a number of similarities with the Image Statistics LogiCORE IP. At first glance the two IP cores may seem redundant, but the

practical uses for these IP cores are actually quite different. They are designed for use in different portions of the video processing pipeline.

The Image Statistics LogiCORE IP calculates a set of statistics for 16 user-defined zones. The particular statistics that the Image Statistics core gathers are optimized for use in control algorithms such as Auto-Focus or Auto-White balance.

In contrast, the Image Characterization core calculates a different set of statistics for a 2-D grid of NxN blocks. This results in a much finer-grained description of the image. The particular statistics that the Image Characterization core gathers have been optimized for use in analytics applications such as Video Surveillance or Road-Sign Recognition.

Feature Summary

The Image Characterization core calculates Mean and Variance statistics for the following characteristics of a frame:

- Luminance
- Chrominance
- High and Low Frequency
- Horizontal, Vertical and Diagonal Edges
- Motion
- Saturation

The statistics are calculated on a Global basis for the entire frame and on a block basis by dividing the frame into NxM blocks and calculating the statistics for each individual block. Block sizes of 4, 8, 16, 32, and 64 can be dynamically chosen depending upon the granularity of the video processing needed by the application.

The Image Characterization core also calculates Global Histograms for the following characteristics:

- Luminance
- Chrominance
- Hue

The Image Characterization core supports the processing of YUV 4:2:2 or YUV 4:2:0 video with 8-bit resolution per component. It also supports the processing of 8-bit Motion data.

When generating the Image Characterization core, you have the option of selecting the type of processor interface that is instantiated on the core. The first option is an EDK pCore

interface that can be easily incorporated into an EDK project. The second option is a Constant mode that uses the user-specified register defaults as the static implementation of the core.

Applications

- Video Surveillance
- Industrial Imaging
- Video Conferencing
- Machine Vision
- Automotive
- Other video applications requiring image analysis

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 Image Characterization product web page.

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

Product Specification

Standards

The Image Characterization 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 1] for additional information.

Performance

The following sections detail the performance characteristics of the Image Characterization v3.00a core.

Maximum Frequency

This section contains typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. See Table 2-1 through Table 2-10 for device-specific information.

Latency

The Image Characterization core outputs a data structure that contains the calculated global and block statistics for each frame that is processed. The block statistics are output as the core is processing the frame. The block size determines the latency for outputting the block statistics. For example, if a block size of 8 is used then the core can begin outputting block statistics once the 8th line has begun to be processed. The global statistics are output once the entire frame has been processed. The frame size will determine the latency of outputting the global statistics.

Throughput

The Image Characterization core outputs a data structure for each frame that is processed. The header and global data in the data structure consists of 1088 32-bit words. The size of the block data section of the data structure depends upon the size of the frame and block size. The block data consists of 14 32-bit words for each block in the frame. For example, for 1280x720 frame size and a block size of 16 there would be 3600 block in each frame. This would yield 50400 32-bit words for the block data for each frame. See the Image Characterization Data Structure section for more details.

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**.

Start by choosing the Family, Maximum Frame Size, and Minimum Block size of the core. If using the pCore Interface, add the corresponding resources. This gives an approximation of the resources that are used by the full core.

If any of the statistics values have been deselected, then the resources associated with that value can be subtracted from the resource calculation. Table 2-4 and Table 2-10 estimate the resources for each statistics value for Virtex-7. The same table can be used for the other families as the resource estimates are very similar. To use Table 2-4 or Table 2-10, locate the statistics value that has been deselected in the table and subtract the associated resources from the resource calculation for the full core. Note the special case when Hue Histogram is deselected, Saturation Mean/Variance is deselected and Color Select = 0. In that case, do not use the individual line items, but use the group calculation at the bottom of the table.

Resource Utilization using Vivado Design Environment

The information presented in Table 2-1 through Table 2-4 is a guide to the resource utilization and maximum clock frequency of the Image Characterization core for Virtex-7, Kintex-7, Artix-7 and Zynq-7000 FPGA families using Vivado tools.

Table 2-1: **Artix-7 (and Zynq-7000 Devices with Artix Based Fabric)**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	5120	5634	19/2	25	164
	16x16 block size	5080	5574	19/2	25	164
	64x64 block size	5011	5514	19/2	25	164

Table 2-1: **Artix-7 (and Zynq-7000 Devices with Artix Based Fabric) *(Cont'd)***

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1280x720 Maximum Frame Size	4x4 block size	5087	5599	17/3	25	164
	16x16 block size	5007	5539	17/3	25	156
	64x64 block size	4963	5478	17/3	25	156
720x480 Maximum Frame Size	4x4 block size	5065	5561	17/2	25	164
	16x16 block size	4994	5501	17/2	25	164
	64x64 block size	4923	5440	17/2	25	164
AXI-Lite Interface		1561	2137	0/0	1	-

Table 2-2: **Kintex-7 (and Zynq-7000 Devices with Artix Based Fabric)**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	5129	5634	19/2	25	212
	16x16 block size	5059	5574	19/2	25	204
	64x64 block size	5024	5514	19/2	25	204
1280x720 Maximum Frame Size	4x4 block size	5114	5599	17/3	25	212
	16x16 block size	5012	5539	17/3	25	212
	64x64 block size	4975	5478	17/3	25	204
720x480 Maximum Frame Size	4x4 block size	5063	5561	17/2	25	204
	16x16 block size	4983	5501	17/2	25	212
	64x64 block size	4884	5440	17/2	25	204
AXI-Lite Interface		1578	2137	0/0	1	-

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

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	5135	5634	19/2	25	204
	16x16 block size	5049	5574	19/2	25	212
	64x64 block size	4988	5514	19/2	25	204
1280x720 Maximum Frame Size	4x4 block size	5112	5599	17/3	25	204
	16x16 block size	4994	5539	17/3	25	212
	64x64 block size	4951	5478	17/3	25	212

Table 2-3: **Virtex-7 Resource Estimates *(Cont'd)***

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
720x480 Maximum Frame Size	4x4 block size	5027	5561	17/2	25	196
	16x16 block size	4953	5501	17/2	25	204
	64x64 block size	4867	5440	17/2	25	204
AXI-Lite Interface		1595	2137	0/0	1	-

Table 2-4: **Virtex-7 Resources Removed When an Item is Not Selected**

Feature	LUTs	FFs	Block RAMs	DSP48s
Y Histogram	193	175	1\0	0
U and V Histogram	172	192	1\0	0
Hue Histogram	1025	1145	1\0	0
Y Mean/Variance	175	130	1\0	3
U Mean/Variance	52	48	1\0	0
V Mean/Variance	52	48	1\0	0
Motion Mean/Variance	166	144	2\0	3
Frequency Mean/Variance	604	545	2\0	6
Edge Content Mean/Variance	665	751	1\2	3
Saturation Mean/Variance	110	97	1\0	0
Color Select = 4	19	72	2\0	0
Color Select = 0	356	322	4\0	0
Hue Histogram Saturation Mean/Variance Color Select = 0	1359	1477	6\0	0

Resource Utilization using ISE Design Environment

The information presented in Table 2-5 through Table 2-10 is a guide to the resource utilization and maximum clock frequency of the Image Characterization core for Virtex-7, Kintex-7, Artix-7, Zynq-7000, Virtex-6, and Spartan-6 FPGA families using ISE tools.

Table 2-5: **Artix-7 DSP Resource Estimates**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	6213	6188	18/4	35	132
	16x16 block size	5976	6117	18/4	35	156
	64x64 block size	5766	6050	18/4	35	148

Table 2-5: **Artix-7 DSP Resource Estimates *(Cont'd)***

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1280x720 Maximum Frame Size	4x4 block size	5997	6153	17/3	35	132
	16x16 block size	5834	6084	17/3	35	148
	64x64 block size	5889	6018	17/3	35	156
720x480 Maximum Frame Size	4x4 block size	5884	6108	17/2	35	156
	16x16 block size	5854	6038	17/2	35	156
	64x64 block size	5728	5973	17/2	35	156
AXI-Lite Interface		1524	1943	0	0	-

Table 2-6: **Kintex-7 DSP Resource Estimates**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	6069	6190	18/4	35	180
	16x16 block size	5869	6119	18/4	35	196
	64x64 block size	5944	6052	18/4	35	196
1280x720 Maximum Frame Size	4x4 block size	6135	6155	17/3	35	204
	16x16 block size	5804	6086	17/3	35	188
	64x64 block size	5720	6020	17/3	35	180
720x480 Maximum Frame Size	4x4 block size	5858	6110	17/2	35	180
	16x16 block size	5789	6040	17/2	35	180
	64x64 block size	5677	5975	17/2	35	188
AXI-Lite Interface		1768	1943	0/0	0	-

Table 2-7: **Virtex-7 Resource Estimates**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	6144	6190	18/4	35	188
	16x16 block size	5918	6119	18/4	35	188
	64x64 block size	5796	6052	18/4	35	156
1280x720 Maximum Frame Size	4x4 block size	5930	6155	17/3	35	188
	16x16 block size	5885	6086	17/3	35	180
	64x64 block size	5733	6020	17/3	35	164

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

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
720x480 Maximum Frame Size	4x4 block size	5935	6110	17/2	35	172
	16x16 block size	5927	6040	17/2	35	188
	64x64 block size	5787	5975	17/2	35	180
AXI-Lite Interface		1718	1943	0/0	0	-

Table 2-8: **Virtex-6 Resource Estimates**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	6267	6201	18/4	35	212
	16x16 block size	6013	6128	18/4	35	196
	64x64 block size	5939	6061	18/4	35	196
1280x720 Maximum Frame Size	4x4 block size	6196	6166	17/3	35	212
	16x16 block size	5899	6095	17/3	35	204
	64x64 block size	5872	6029	17/3	35	204
720x480 Maximum Frame Size	4x4 block size	5999	6121	17/2	35	172
	16x16 block size	5943	6049	17/2	35	204
	64x64 block size	5926	5984	17/2	35	204
AXI-Lite Interface		1524	1941	0/0	0	-

Table 2-9: **Spartan-6 Resource Estimates**

Feature		LUTs	FFs	Block RAMs 36/18	DSPs	MHz
1920x1080 Maximum Frame Size	4x4 block size	5995	6184	37/1	35	148
	16x16 block size	5815	6110	25/1	35	148
	64x64 block size	5724	6042	25/1	35	156
1280x720 Maximum Frame Size	4x4 block size	5889	6151	37/1	35	148
	16x16 block size	5768	6082	25/1	35	156
	64x64 block size	5663	6011	25/1	35	156
720x480 Maximum Frame Size	4x4 block size	5712	6101	20/3	35	156
	16x16 block size	5722	6034	20/3	35	125
	64x64 block size	5614	5969	20/3	35	156
AXI-Lite Interface		1858	1925	0/0	0	-

Table 2-10: **Virtex-7 Resources Removed When an Item is Not Selected**

Feature	LUTs	FFs	Block RAMs	DSP48s
Y Histogram	71	163	1\0	0
U and V Histogram	71	174	1\0	0
Hue Histogram	375	333	1\0	0
Y Mean/Variance	243	189	1\0	3
U Mean/Variance	247	247	1\0	3
V Mean/Variance	252	246	1\0	3
Motion Mean/Variance	181	201	1\2	3
Frequency Mean/Variance	529	651	2\0	6
Edge Content Mean/Variance	599	758	1\2	3
Saturation Mean/Variance	408	337	1\0	4
Color Select = 4	65	176	2\0	0
Color Select = 0	418	432	4\0	0
Hue Histogram Saturation Mean/Variance Color Select = 0	1974	1495	6\0	4

Core Interfaces and Register Space

Port Descriptions

The Image Characterization 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 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 via the GUI with an AXI4-Lite control interface. The `INTC_IF` interface is present only when the core is configured via the GUI with the INTC interface enabled. The `video_clk` pin is present only when the core is configured via the GUI with the Input Video Interface set to “Async_Video_Clock” mode

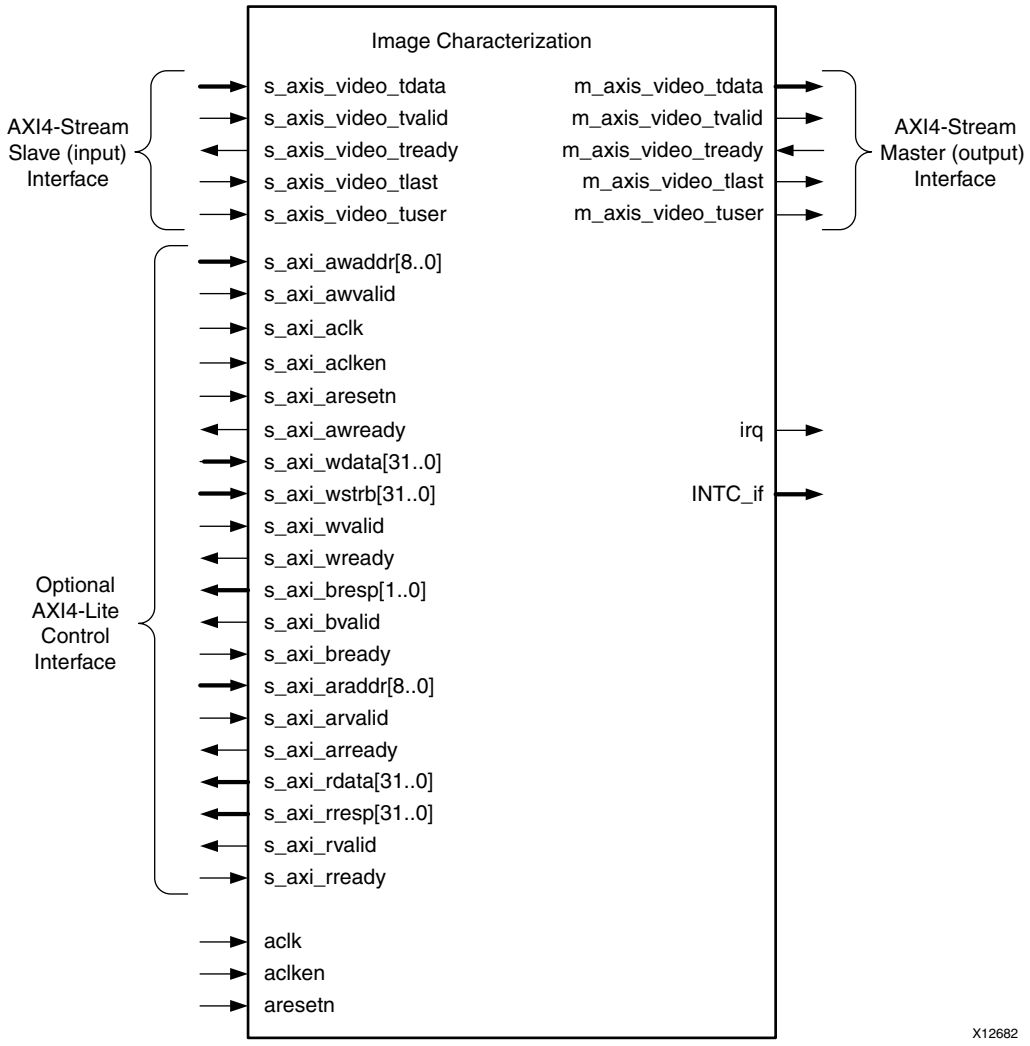

Figure 2-1: **Image Characterization Top-Level Signaling Interface**

Common Interface Signals

Table 2-11 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-11: **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	9	Optional External Interrupt Controller Interface. Available only when `INTC_IF` is selected on GUI.
IRQ	Out	1	Optional Interrupt Request Pin. Available only when AXI4-Liter interface is selected on GUI.

Table 2-11: **Common Interface Signals**

Signal Name	Direction	Width	Description
fsync_out	Out	1	Delayed output of the Start of Frame (SOF) signal from the AXI-4 Stream Slave input. Typically used to synchronize with the Object Segmentation Core.
buffer_ptr	Out	1	Output that specifies which output buffer the Image Characterization core is actively writing to. T ypically used to synchronize with the Object Segmentation Core.
video_clk	In	1	Clock associated with the AXI-4 Stream Slave input when the "Input_Video_Interface" is set to "Async_Video_Clock" mode via the 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.

Fsync_Out

The fsync_out signal is a delayed output of the Start of Frame signal on the s_axis_video_tuser signal of the AXI-4 Stream Slave input. This signal is typically used with the Object Segmentation core for synchronization purposes.

Buffer_Ptr

The `buffer_ptr` pin is used to denote which output buffer the Image Characterization core is actively writing data. When `buffer_ptr` = '0', the Image Characterization core is writing data to the buffer specified in register "Image Characterization Start Address 0". When `buffer_ptr` = '1', the Image Characterization core is writing data to the buffer specified in register "Image Characterization Start Address 1". This signal is typically used with the Object Segmentation core for synchronization purposes.

Video_Clk

The `video_clk` is associated with the AXI-4 Stream Slave input when the "Input_Video_Interface" is set to "Async_Video_Clock" mode in the GUI. The `video_clk` is asynchronous to the core clock aclk.

Data Interface

The Image Characterization 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 *AXI Reference Guide* (UG761) [Ref 1].

AXI4-Stream Signal Names and Descriptions

Table 2-12 describes the AXI4-Stream signal names and descriptions.

Table 2-12: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	24	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_s2mm_tdata	Out	32	Output Data
m_axis_s2mm_tvalid	Out	1	Output Valid
m_axis_s2mm_tready	In	1	Output Ready
m_axis_s2mm_tlast	Out	1	Output End of Line
m_axis_s2mm_cmd_tdata	Out	72	Output Command Data
m_axis_s2mm_cmd_tvalid	Out	1	Output Command Valid
m_axis_s2mm_cmd_tready	In	1	Output Command Ready

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. The Image Characterization core supports YUV 4:2:2 and 4:2:0 video with the addition of 8-bit motion content data. Luma ("Y") is located on the bus at `s_axis_video_tdata`[7:0]. Chroma ("U/V") is located on the bus at `s_axis_video_tdata`[15:8]. Motion ("M") is located on the bus at `s_axis_video_tdata`[23:16].

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 2-2. During valid transfers, `DATA` only carries active video data. Blank periods and ancillary data packets are not transferred via the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

Once `s_axis_video_tvalid` is asserted, no interface signals (except the Image Characterization core driving `s_axis_video_tready`) may change value until the transaction completes (`s_axis_video_tready` and `s_axis_video_tvalid` `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.

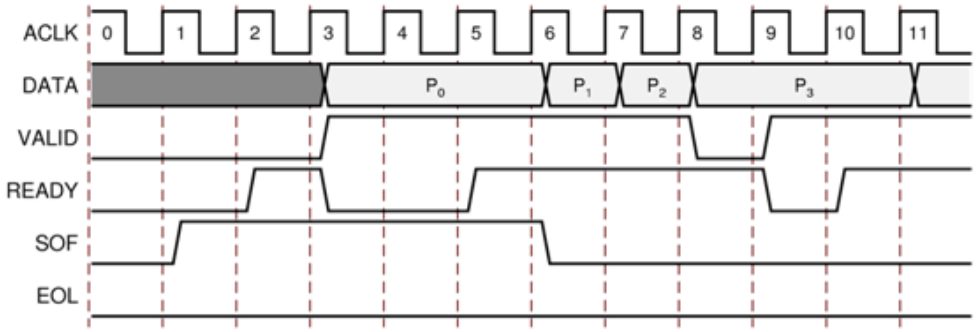

Figure 2-2: **Example of READY/VALID Handshake, Start of a New Frame**

Start of Frame Signals - 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 2-2. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The `SOF` signal may be asserted an arbitrary number of `ACLK` cycles before the first pixel value is presented on `DATA`, as long as a `VALID` is not asserted.

End of Line Signals - m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line signal, physically transmitted over the AXI4-Stream TLAST signal, marks the last pixel of a line. The EOL pulse is 1 valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-3.

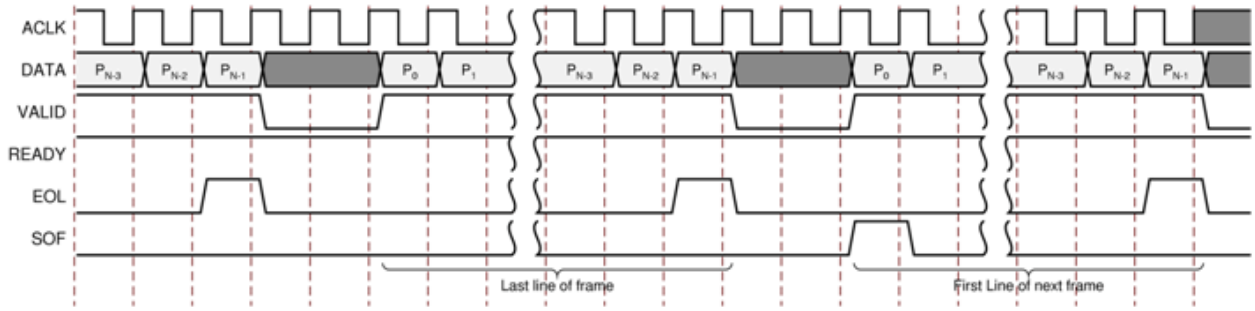

Figure 2-3: **Use of EOL and SOF Signals**

Control Interface

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

Constant Configuration

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

AXI4-Lite Interface

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

The Image Characterization core can be controlled via the AXI4-Lite interface using read and write transactions to the Image Characterization register space.

Table 2-13: **AXI4-Lite Interface Signals**

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

S_AXI_ACLK

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

S_AXI_ACLKEN

The S_AXI_ACLKEN pin is an active-high, synchronous clock-enable input for the AXI4-Lite interface. Setting S_AXI_ACLKEN low (de-asserted) halts the operation of the AXI4-Lite interface despite rising edges on the S_AXI_ACLK pin. AXI4-Lite interface states are

maintained, and AXI4-Lite interface output signal levels are held until S_AXI_ACLKEN is asserted again. When S_AXI_ACLKEN is de-asserted, AXI4-Lite interface inputs are not sampled, except S_AXI_ARESETn, which supersedes S_AXI_ACLKEN. The AXI4-Stream interfaces signals are not affected by the S_AXI_ACLKEN.

S_AXI_ARESETn

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

Register Space

The standardized Xilinx Video IP register space is partitioned into control-, timing-, and core specific registers. The Image Characterization core uses only one timing related register, ACTIVE_SIZE (0x0020), which allows specifying the input frame dimensions. The core has 19 core-specific registers for setting the buffer addresses, scaling values and the color selects.

Table 2-14: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	No	Power-on-Reset : 0x0	Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 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 2-14: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0010	VERSION	R	N/A	0x0300a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	SYSDEBUG0	R	N/A	0	0-31: Frame Throughput monitor (1)
0x0018	SYSDEBUG1	R	N/A	0	0-31: Line Throughput monitor (1)
0x001C	SYSDEBUG2	R	N/A	0	0-31: Pixel Throughput monitor (1)
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame
0x0100	Image Characterization Start Address 0	R/W	Yes	Specified via GUI	31-0: Starting Address for Image Characterization Buffer 0
0x0104	Image Characterization Start Address 1	R/W	Yes	Specified via GUI	31-0: Starting Address for Image Characterization Buffer 1
0x0108	Block Size	R/W	Yes	Specified via GUI	6-0: Block Size (4, 8, 16, 32, or 64)
0x010C	Total Number of Blocks	R/W	Yes	Specified via GUI	19-0: Total Number of Blocks (Horizontal x Vertical)
0x0110	Number of Blocks	R/W	Yes	Specified via GUI	9-0: Number of Blocks Wide 25-16: Number of Blocks High
0x0114	Global Y Width Scaling	R/W	Yes	Specified via GUI	16-0: Scale factor for Global Luma and Chroma Means & Variances Calculated: (1/ Num_Blocks_Wide)*65536
0x0118	Global Y Height Scaling	R/W	Yes	Specified via GUI	16-0: Scale factor for Global Luma and Chroma Means & Variances Calculated: (1/ Num_Blocks_High)*65536
0x011C	Block Y Scaling	R/W	Yes	Specified via GUI	16-0: Scale factor for Block Means & Variances. Calculated: (1/ Num_Block_Pixels)*65536

Table 2-14: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0120	Block C Scaling	R/W	Yes	Specified via GUI	16-0: Scale factor for Block Chroma based Means & Variances. Calculated: (2/ Num_Block_Pixels)*65536 for YUV 4:2:2 or (4/Num_Block_Pixels)*65536 for YUV 4:2:0
0x0124	High Frequency Gain	R/W	Yes	Specified via GUI	3-0: High Frequency Gain (Accepted values = 1, 2,4, 8)
0x0128	Edge Gain	R/W	Yes	Specified via GUI	3-0: Right Diagonal Edge Gain 11-8: Left Diagonal Edge Gain 19-16: Vertical Edge Gain 27-24: Horizontal Edge Gain (Accepted values = 0, 1, 2, 4, 8)
0x012C	Color Select #1	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum
0x0130	Color Select #2	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum
0x0134	Color Select #3	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum
0x0138	Color Select #4	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum
0x013C	Color Select #5	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum
0x0140	Color Select #6	R/W	Yes	Specified via GUI	7-0: Hue Minimum 15-8: Hue Maximum 23-16: Saturation Minimum 31-24: Saturation Maximum

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

CONTROL (0x0000) Register

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

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

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

STATUS (0x0004) Register

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

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

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

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

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

ERROR (0x0008) Register

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

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

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

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

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

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

IRQ_ENABLE (0x000C) Register

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

Version (0x0010) Register

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

hardware. See Table 2-14 for details.

SYSDEBUG0 (0x0014) Register

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

SYSDEBUG1 (0x0018) Register

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

SYSDEBUG2 (0x001C) Register

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

ACTIVE_SIZE (0x0020) Register

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

Block_Size (0x0108) Register

The Block_Size register encodes the size of the NxN block is used to overlay the image. Accepted block sizes (N) are 4, 8, 16, 32 or 64.

Total_Number_of_Blocks (0x010C) Register

The Total_Number_of_Blocks register encodes the total number of blocks that overlay the image. The total number of blocks is calculated as the Number of Horizontal Blocks x Number of Vertical Blocks.

Number_of_ Blocks (0x0110) Register

The Number_of_Blocks register encodes the number of Vertical blocks in each column and the number of Horizontal blocks in each row.

Global_Y_Width_Scaling (0x0114) Register

The Global_Y_Width_Scaling register encodes a scaling value that is used when calculating the global image statistics. The scaling value is typically calculated as (1/ Number_Horizontal_Blocks)*65536.

Global_Y_Height_Scaling (0x0118) Register

The Global_Y_Height_Scaling register encodes a scaling value that is used when calculating the global image statistics. The scaling value is typically calculated as (1/ Number_Vertical_Blocks)*65536.

Block_Y_Scaling (0x011C) Register

The Block_Y_Scaling register encodes a scaling value that is used when calculating the block image statistics for the Luma based components such as Y, Low Frequency, High Frequency, Edge Content and Motion. The scaling value is typically calculated as (1/ Number_Block_Pixels)*65536.

Block_C_Scaling (0x0120) Register

The Block_C_Scaling register encodes a scaling value that is used when calculating the block image statistics for the Chroma based components such as U, V and Saturation. The scaling value is typically calculated as (2/Number_Block_Pixels)*65536 when using YUV 4:2:2 data or as (4/Number_Block_Pixels)*65536 when using YUV 4:2:0 data.

High_Frequency_Gain (0x0124) Register

The High_Frequency_Gain register encodes the gain value for the high frequency data when calculating the high frequency mean and variance values. Accepted gain values are 1, 2, 4, or 8.

Edge_Gain (0x0128) Register

The Edge_Gain register encodes the gain values for the 4 edge components: Horizontal, Vertical, Right Diagonal and Left Diagonal. Each gain value has an accepted range of 0, 1, 2, 4, or 8.

Color_Select #1-8 (0x012C - 0x0148) Register

The Color_Selecct registers encode the minimum and maximum values for the Hue and Saturation thresholds that are used for color detection.

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 Image Characterization 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, as seen in Table 2-15.

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

Table 2-15: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	Pixel counter terminal count
3	Line counter terminal count
4	Slave Error
5	EOL Early
6	EOL Late
7	SOF Early
8	SOF Late

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected INTC_IF signals as edge triggered interrupt sources. The INTC IP provides functionality to mask (enable or disable), as well as identify individual interrupt

sources from software. Alternatively, for an external processor or MCU the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

Designing with the Core

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

Architecture

The Image Characterization core is implemented as four subsystems: YC Processing, Block Stats, Global Stats, and Histograms. These subsystems are connected as shown in Figure 3-1.

The details of each sub-system are discussed in the following sections.

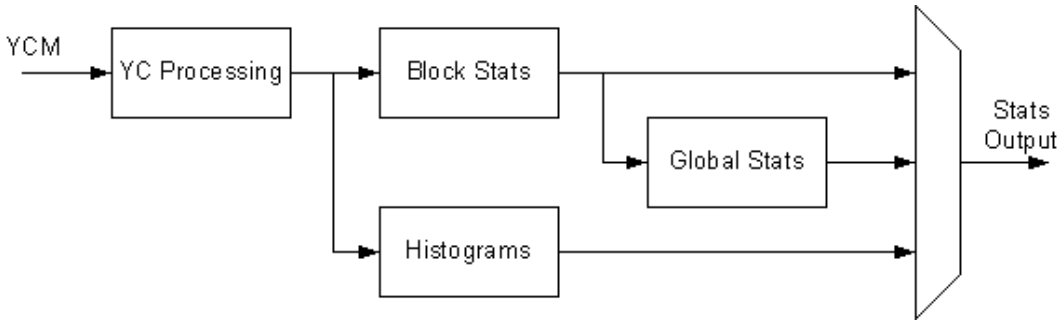

Figure 3-1: **Image Characterization Block Diagram**

YC Processing

The YC Processing subsystem receives the image data and calculates the following information for each pixel in the image:

- Frequency Content
- Edge Content
- Color Conversion

The Frequency, Edge, and Color calculations are implemented as separate processing pipelines as illustrated in Figure 3-2. The results are then passed to the Block Stats, Global Stats, and Histogram blocks which generate the various statistical data about the image.

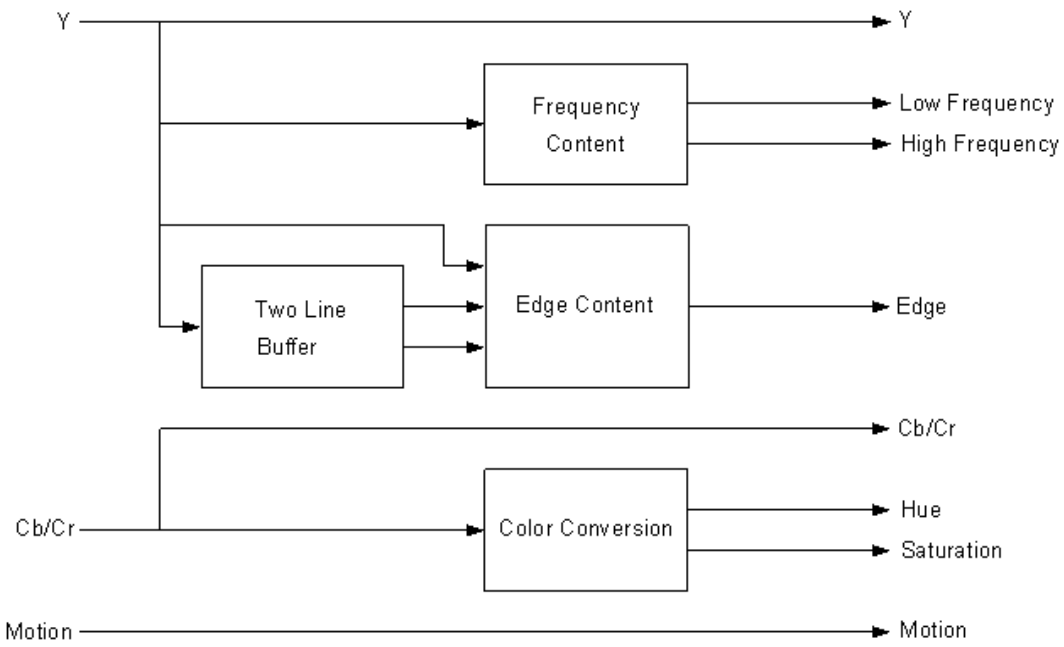

Figure 3-2: **YC Processing Block Diagram**

Frequency Content

The Frequency Content is calculated on only the Luminance or Y portion of the image. The frequency content that is calculated consists of the Low Frequency portion of the image and the High Frequency portion of the image.

The Low Frequency portion of the image is calculated by passing the data through a low pass filter to remove the high frequencies. As shown in Figure 3-3, the low pass filter is implemented as a 7-tap FIR filter with the following hard-coded filter coefficients: -1, 0, 9, 16, 9, 0, -1.

The High Frequency portion of the image is calculated by subtracting the low frequency portion of the image from the baseband image. The High Freq. Gain register allows you to multiply the High Frequency data by 1, 2, 4, or 8 before the value is finally clamped to the range 0 : 255.

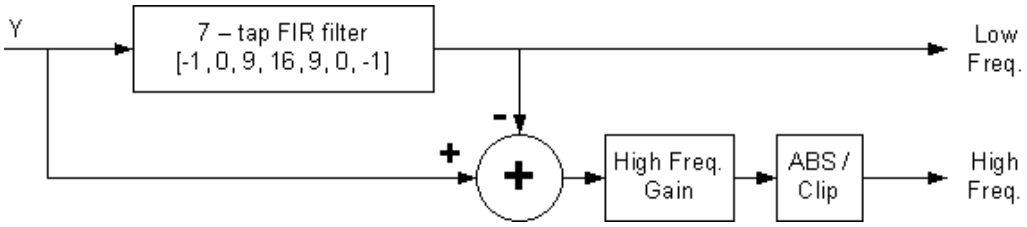

Figure 3-3: **Frequency Content Block Diagram**

Edge Content

The Edge Content is calculated on only the Y portion of the image. Four Sobel filters are used to look for horizontal, vertical, and diagonal edges in the image. As shown in Figure 3-4, each of the four components has a separate gain factor to allow for the emphasis of a particular type of edge. Valid gain values are 0, 1, 2, 4, and 8. The final result is a sum of the four edge components. This arrangement gives a good measure of the edges for any particular pixel in the image. Since 3x3 2D FIR filters are used to implement the Sobel filters, a line buffer capable of holding two lines of data is required, as shown in Figure 3-2. The size of the line buffer is based on the Maximum Frame Size parameter. See the EDK pCore GUI in Chapter 8 section for more details.

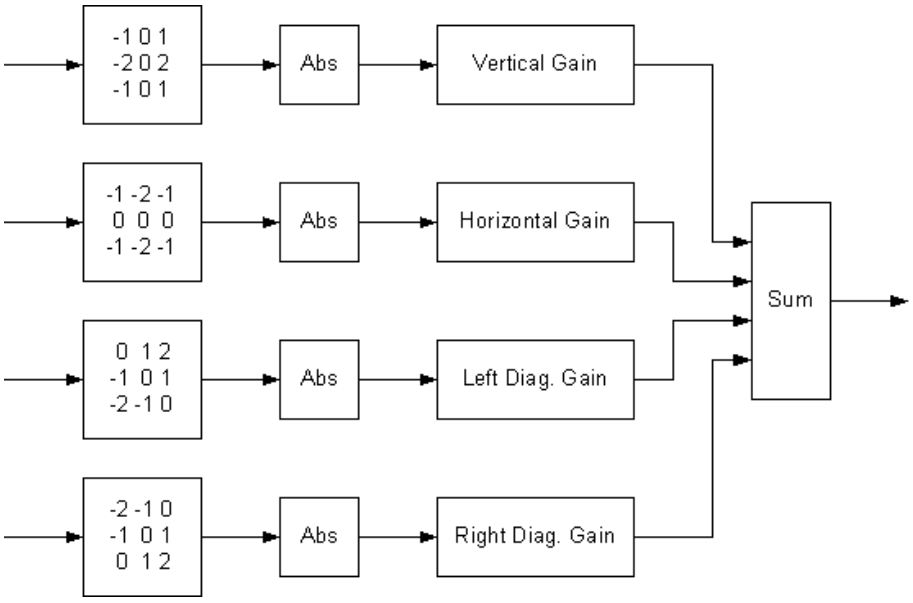

Figure 3-4: **Edge Content Block Diagram**

Color Conversion

The Color Content of the image is calculated from the Chrominance (C) portion of the image. For Chrominance, the color difference signals B-Y (Cb) and R-Y (Cr) are generated independently, but it is the color comprised by the combination of Cb and Cr that is of interest in characterization. To this effect, the magnitude (Saturation) and angle (Hue) of the Chrominance vector is calculated to provide the color content.

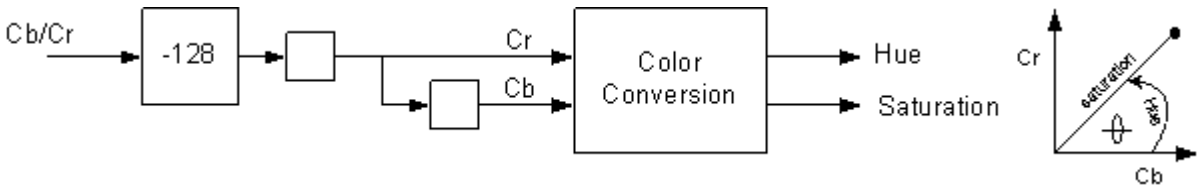

Figure 3-5: **Color Conversion**

$$Cr' = Cr - 128$$

$$Cb' = Cb - 128$$

$$Saturation = \sqrt{(Cr')^2 + (Cb')^2}$$

$$Hue = \frac{256 \times atan2(Cr',\ Cb')}{2\pi}$$

Equation 3-1

Block Statistics

The Block Statistics subsystem receives data from the YC Processing subsystem. It then subdivides the image into an HxV number of horizontal and vertical blocks respectively. In the example shown in Figure 3-6, the image is subdivided into an 8x5 grid of blocks.

Blocks are measured in pixels. Valid block sizes are 4x4, 8x8, 16x16, 32x32, and 64x64. You can specify the block size using the "Block_Size" register. Blocks are defined as starting at the upper left corner of the image, then moving left to right and top to bottom. If the image has a non-integer number of blocks, the partial blocks along the right and bottom edge of the image (the gray boxes in Figure 3-6) will be excluded from the video analytics analysis.

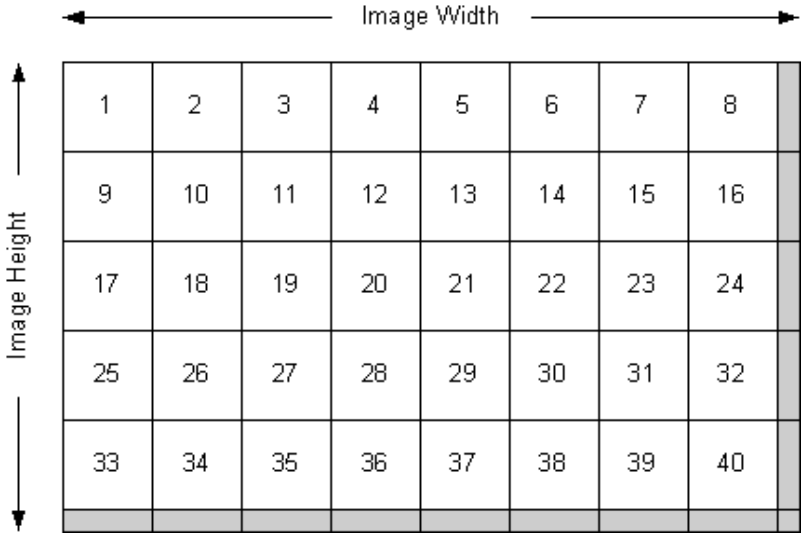

Figure 3-6: **Block Overlay of an Image Frame**

The following measurements are calculated for each block:

- Mean and Variance

- Y
- Cr
- Cb
- Low Frequency Content
- High Frequency Content
- Edge Content
- Motion Content
- Saturation

- Color Selection (x8)

A block diagram of the block statistics processing is illustrated in Figure 3-7. The mean and variance processing is implemented as eight independent processing pipelines.

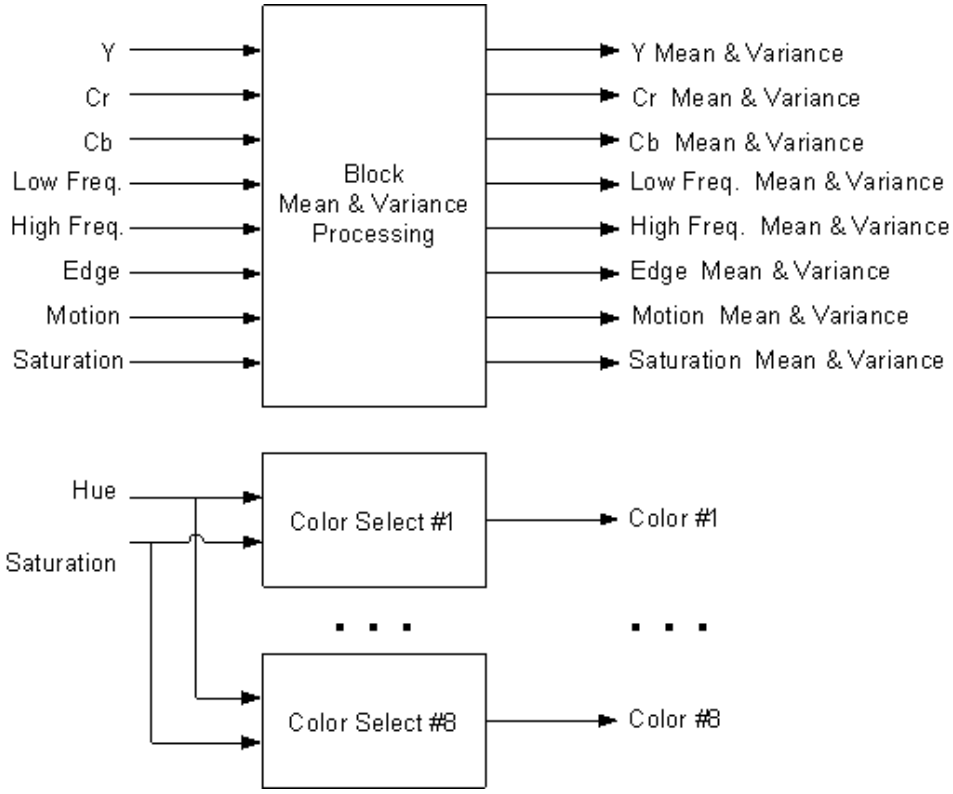

Figure 3-7: **Block Statistics Block Diagram**

The Mean, Variance, and Color Select calculations are each discussed in more detail in the following sections. Each measurement is implemented as an independent processing chain.

Block Mean

A block diagram of the block mean processing is illustrated in Figure 3-8. Block mean values are calculated by first scaling each pixel in the block by a "Block Scaling" factor. Once each pixel in the block has been scaled, it is summed in an accumulator.

The block scaling factor is set through the "Block_Y_Scaling" register for the Y-based data streams and through the "Block_C_Scaling" register for the C-based data streams. The Block_Y_Scaling is typically calculated as (1/Num_Block_Pixels)*65536. Num_Block_Pixels is the number of pixels in a block. The Block_C_Scaling is typically calculated as (2/Num_Block_Pixels)*65536 for 4:2:2 data or as (4/Num_Block_Pixels)*65536 for 4:2:0 data.

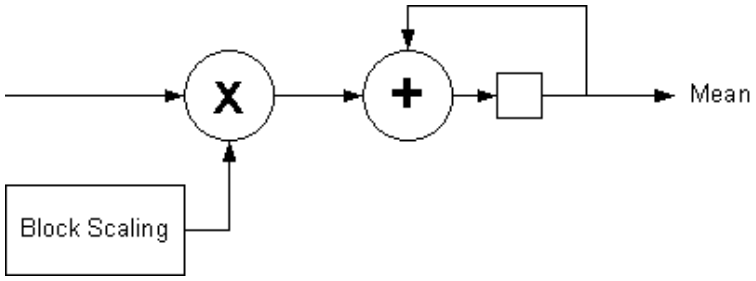

Figure 3-8: **Block Mean Block Diagram**

Block Variance

A block diagram of the block variance processing is illustrated in Figure 3-9. Block variance values are calculated by first squaring each pixel in the block and then scaling by the "Block Scaling" factor. Once each pixel has been scaled, it is summed in an accumulator. The last step is to subtract the square of the block mean value from the accumulated value.

The block scaling factor is set through the "Block_Y_Scaling" register for the Y-based data streams and through the "Block_C_Scaling" register for the C-based data streams. Typical calculations for the block scale factors are discussed in the Block Mean section.

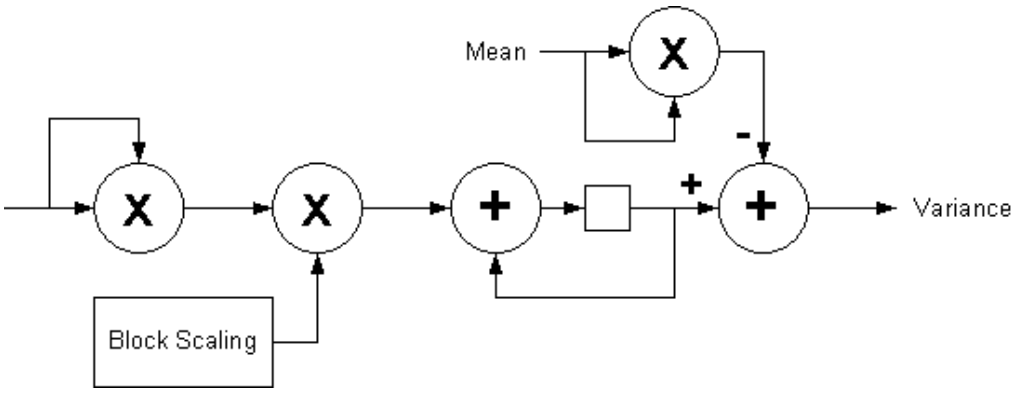

Figure 3-9: **Block Variance Block Diagram**

Color Select

Color Select uses the Hue and Saturation values for a pixel to detect if the pixel falls within a specified color range. If the pixel meets the specified color range, then the color is said to have been detected. There will be eight separate color selector circuits allowing for the detection of eight different colors.

Each color selector inputs a Hue Minimum and Maximum and a Saturation Minimum and Maximum. These values are set by using the Color Select #1-8 registers.

To match the specified color, the Hue and Saturation of the pixel must fall within both the Hue Thresholds and the Saturation Thresholds. For each pixel in the block that matches the specified color, a counter is incremented. The final value from the Color Select is the number of pixels in the block that matched the specified color.

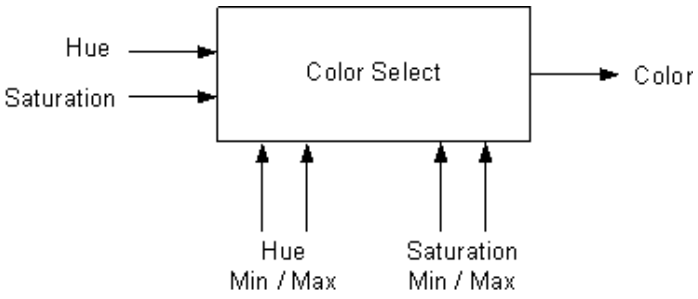

Figure 3-10: **Block Color Select Block Diagram**

Global Statistics

The Global Stats subsystem uses the Block mean and variance values to calculate Global means and variances of the full image. Global means and variances are calculated for the following values:

- Y
- Cr
- Cb
- Low Frequency Content
- High Frequency Content
- Edge Content
- Motion Content
- Saturation

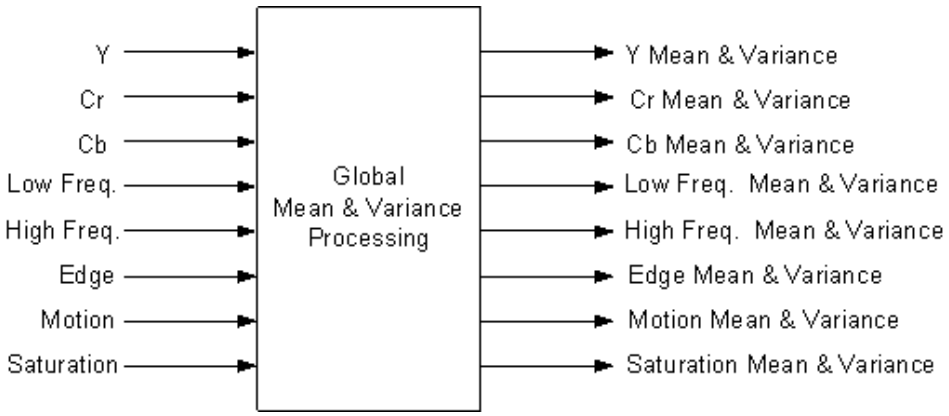

Figure 3-11: **Global Statistics Block Diagram**

To calculate a global mean or variance value, the value from each block is first scaled by the Global Width Scaling factor and the Global Height Scaling factor. After the scaling process, the value is summed by an accumulator for all of the blocks in the image. Figure 3-12 illustrates this process.

The Global Width Scaling factor is set through the "Global_Y_Width_Scaling" register (see Table 2-14). Typically this value is calculated by (1/Num_Blocks_Wide)*65536. The Global Height Scaling factor is set through the "Global_Y_Height_Scaling" register. Typically this value is calculated by (1/Num_Blocks_High)*65536.

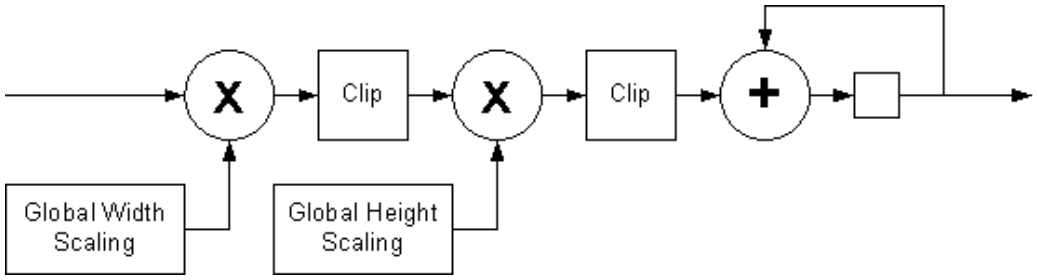

Figure 3-12: **Global Mean/Variance Block Diagram**

Histograms

The Histograms subsystem takes its input from the YC Processing subsystem. It calculates separate histograms over the entire frame for the following values:

- Y
- Cr
- Cb
- Hue

Since the Image Characterization core supports 8-bit data, each histogram contains 256 bins. The Histogram data is stored in memory starting with bin 0 and ending with bin 255.

Image Characterization Output

All of the statistics calculated by the Image Characterization core are written to external memory buffers. The data is written to two external memory buffers in a ping-pong fashion. The locations in memory of the two buffers are specified through the "Image_Characterization_Start_Addr0" and "Image_Characterization_Start_Addr1" registers. The statistics are written to memory in a data structure that is specified in the Image Characterization Data Structure section.

Image Characterization Output Order

The Image Characterization core writes the calculated statistics to memory in the following order:

1. Frame Header Start
2. Block Statistics
3. Global Statistics
4. Histograms
5. Frame Header Final

The first step is writing the Frame Header Start block. The Header contains a Struct_Valid value. The Frame Header Start block writes a value of 0x0001 to the Struct_Valid value. This denotes that a new data structure has been started, but is not completed and should not be used for processing.

When the Image Characterization core completes all of the block statistics for a particular block of the image, those values are written to memory. Once all of the Block Statistics have been written to memory and the Global Statistics have been calculated, the Global Statistics are written to memory. Next the Histograms are written to memory. The final step is to write the Frame Header Final block. The Frame Header Final block is the same as the Frame Header Start block except that a value of 0xFFFF is written to the Struct_Valid value to denote that the data structure has been completed and is now ready to be used for processing.

Image Characterization Data Structure

The Image Characterization Data Structure defines how the image characterization statistics are organized when written to external memory. The data structure is made up of three pieces which are located contiguously in memory:

- Frame Header (see Table 3-2)
- Global Stats & Histograms (see Table 3-3)
- Block Stats (see Table 3-4)

The first two pieces are static in size. Both contain PAD values that are used to pad the size of structure to be a multiple of 128 bytes.

The size of the Block Stats structure is dependent on the number of blocks in the processed image. There will be one instance of the Block Stats data structure for each block in the image. The Block Stats data structures are arranged contiguously in memory. The order of the blocks corresponds to traversing through the blocks from left to right and from top to bottom.

The values in the Statistics Data structure use the following bit widths:

- Mean – 8-bits
- Variance – 16-bits
- Histogram – 32-bits (21-bits actual)
- Color_Select – 16-bits (12-bits actual)
- PAD – 32-bits (0x0000)

Table 3-1: **Image Characterization Data Structure**

Byte 3	Byte 2	Byte 1	Byte 0
Frame Header (32 words)			
Global Stats (32 words)			
Histograms (1024 words)			
Block Stats - Block #1 (14 words)			
...			
Block Stats - Block # HxV (14 words)			

Table 3-2: **Image Characterization Data Structure Frame Header**

Byte 3	Byte 2	Byte 1	Byte 0
Struct_Valid			
Frame_Index			
PAD (x30)			

Table 3-3: **Image Characterization Data Structure Global Stats**

Byte 3	Byte 2	Byte 1	Byte 0
Low_Freq_Mean	V_mean	U_Mean	Y_Mean
Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
U_Var		Y_Var	

Table 3-3: **Image Characterization Data Structure Global Stats *(Cont'd)***

Byte 3	Byte 2	Byte 1	Byte 0
Low_Freq_Var		V_Var	
Edge_Var		High_Freq_Var	
Saturation_Var		Motion_Var	
PAD (x26)			
Y_Histogram (x256)			
U_Histogram (x256)			
V_Histogram (x256)			
Hue_Histogram (x256)			

Table 3-4: **Image Characterization Data Structure Block Stats**

Byte 3	Byte 2	Byte 1	Byte 0
Low_Freq_Mean	V_mean	U_Mean	Y_Mean
Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
U_Var		Y_Var	
Low_Freq_Var		V_Var	
Edge_Var		High_Freq_Var	
Saturation_Var		Motion_Var	
Color_Sel_2		Color_Sel_1	
Color_Sel_4		Color_Sel_3	
Color_Sel_6		Color_Sel_5	
Color_Sel_8		Color_Sel_7	
Reserved			
Reserved			
Reserved			
Reserved			

Note: The Block Stats repeats once for each block in the image. For example a 1280x720 image with block size 16 would result in 3600 contiguous instances of Block Stats data.

General Design Guidelines

Image Characterization Control and Timing

The basic operation of the Image Characterization core is the same regardless of how the core is instantiated. The process begins with the `s_axis_video_tuser` signal going

active to denote that start of a frame. At this point, if the "Register Update Enable" (bit 1) of the Control register is set to '1', then all of the system registers are updated. This mechanism allows the registers of the core to be double buffered.

At the beginning of each video frame, the Image Characterization core also writes the "Frame Header Start" block to the memory buffer to signify that a new buffer has been started. Next, the core begins to process the incoming frame of video data. As the block statistics for each block finish processing, they are written out to external memory. This continues until the entire frame has been processed and all of the block statistics data have been written to the memory buffer.

As the block statistics data is written to memory, it is used to calculate the global statistics. Once all of the block data has been written to memory, the global data is written to memory followed by the histograms. When all of the statistics data has been written to memory, the core completes the process by writing the "Frame Header Final" block to denote that the entire statistics structure has been written to memory and is now ready to be used for further processing. At this point, the `frame_done` signal goes to '1' and remains there until the `s_axis_video_tuser` goes high to denote the start of a new frame.

Input AXI4-Stream Slave Interface

The Image Characterization core uses an AXI4-stream slave interface to input video data. Data on `s_axis_tdata` (TDATA) is only transferred when `s_axis_tready` (TREADY) and `s_axis_tvalid` (TVALID) are both asserted. The `s_axis_tlast` (TLAST) must be asserted high during the last TDATA transaction of each video line. Figure 3-13 illustrates the transfer of one line of data consisting of 8 data values.

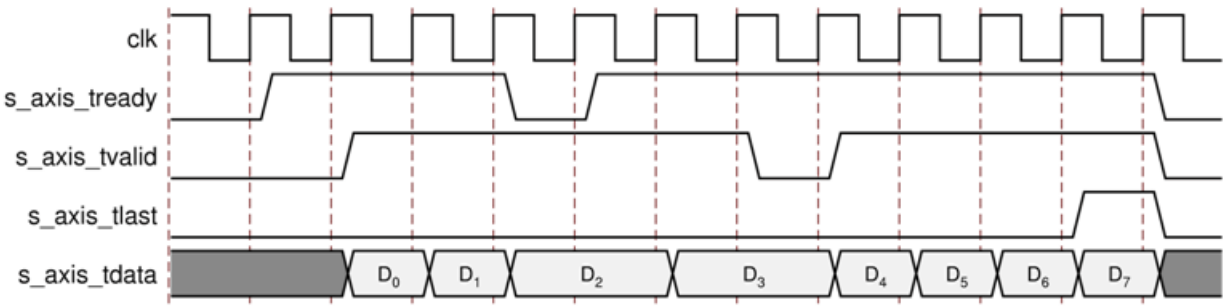

Figure 3-13: **Input AXI4-Stream Timing**

Video Input Bus

The Image Characterization core uses an AXI4 Stream interface to input a stream of video plus motion. This data bus uses a 24-bit word with Luma (Y), Chroma (C), and Motion (M)

(such as from the Xilinx Motion Adaptive Noise Reduction IP LogiCORE) placed. Any unused portion of the bus can be simply tied to a constant value.

s_axis_tdata[23:16]	s_axis_tdata[15:8]	s_axis_tdata[7:0]
Motion[7:0]	Chroma[7:0]	Luma[7:0]

Motion data is a magnitude measurement of how much a Luma pixel has changed between the current frame and the previous frame. The Xilinx Motion Adaptive Noise Reduction LogiCORE IP can be used to calculate the motion content of a video sequence and drive the YCM input to the Image Characterization core.

Chroma Formatting

The formatting of the chroma data can often be a source of confusion. The video formats supported by the Image Characterization core are YCbCr 4:2:2 and YCbCr 4:2:0. For both formats, the chroma components, Cb and Cr, are interleaved on a single 8-bit channel as illustrated in Figure 3-14. The Cb value is always placed before the corresponding Cr value.

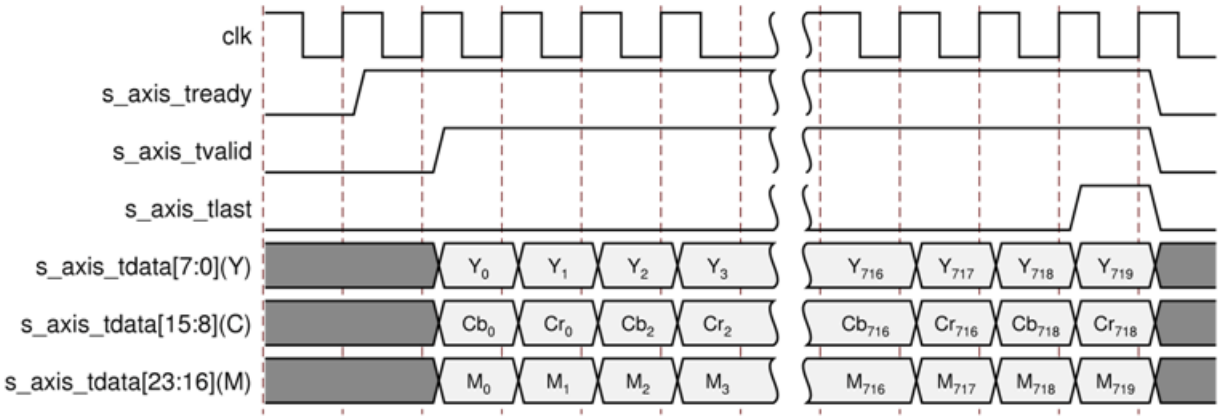

Figure 3-14: **4:2:2 or 4:2:0 Bus Format**

Figure 3-15 shows a high-level view of a YCbCr 4:2:0 video stream. For a YCbCr 4:2:0 video stream, the chroma channel is valid for one line and then invalid for the next line. This format of alternating lines of valid and invalid chroma data is repeated for the entire video frame. When processing YCbCr 4:2:0 video, the Image Characterization core uses the Chroma_Polarity bit in the Control register to determine the status of the Chroma data for the first line of each frame of video. If Chroma_Parity = 1, the chroma data is valid for the first line and invalid for the next line. If Chroma_Parity = 0, the chroma data is invalid for the

first line and valid for the next line. For Figure 3-15, Chroma_Parity = 1.

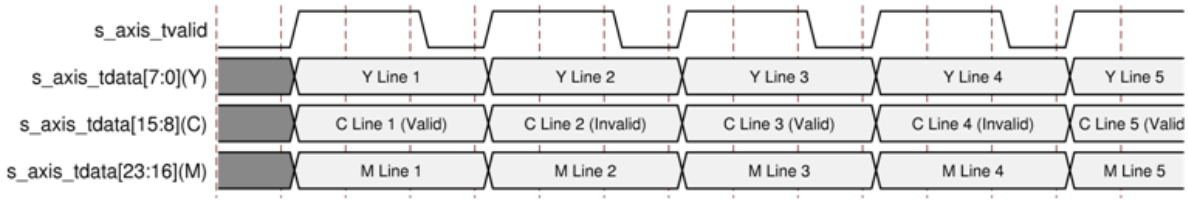

Figure 3-15: **4:2:2 or 4:2:0 High Level**

Video Input Clocking

The Video Input Interface has two modes: Async Video Clock and Sync Video Clock. Async Video Clock is use when the video input data has a clock that is asynchronous to the core's processing clock. In this mode a `video_clk` port is provided for the video input data's clock and a clk port is provided for the core's processing clock. The speed of the `video_clk` may be equal to or slower than the speed of the clk. When the video input data has a clock that is synchronous to the core's processing clock, the Sync Video Clock mode is recommended. In this case a separate `video_clk` is not needed so only the clk port is provided.

Example Case

This section provides a walk through of the process of initializing the Image Characterization core. This example could be for an application that looks for rectangular road signs that are either Yellow, Green, Red, or Blue.

- 1280x720 YUV 4:2:0 video with Motion
- Block size 16x16
- Emphasis on Horizontal Edges
- Colors
  - C1: Cb = 44, Cr = 142 (Yellow)
  - C2: Cb = 72, Cr = 58 (Green)
  - C3: Cb = 100, Cr = 212 (Red)
  - C4: Cb = 212, Cr = 114 (Blue)

Initializing the Image Characterization core can be broken down into the three steps. The first step is setting up the Buffer address registers so that the output Image Characterization data is written to the correct location in external memory. The second step is to initialize the core with the proper frame data so that it can properly calculate the statistics for each frame. The last step is to initialize the core so that it calculates the proper

statistics for the current application. These three steps will be discussed in the following sections.

Buffer Initialization

The Image Characterization core has two Buffer Start Address registers. These registers should be programmed with the base address of the two Image Characterization data buffers. The Image Characterization core will write the results of one frame of processing to one of the buffers and then switch to the other buffer for writing the results of the next frame. The core will continue to ping-pong between the buffers on each successive frame.

The Image Characterization data structure has a "Frame Index" value in the header portion of the data structure. The first frame that is processed uses the value programmed in the Image Characterization Start Frame Index register. Each successive frame will increment the frame index by a value of one.

Frame Data Initialization

The Image Characterization core has to be informed of the details of the frame size and the block size in order to properly process each frame. To initialize the frame data follow these steps:

1. Load the Active_Size register with the frame height and width of the frame to be processed.
    a. Frame Height = 720 lines
    b. Frame Width = 1280 pixels.
2. Load the Block Size register with the size of the block to be processed.
    a. Block Size = 16.
3. Load the Number of Blocks register with the number of blocks high and the number of blocks wide.
    a. Number of Block High = (720/16) = 45
    b. Number of Blocks Wide = (1280/16) = 80.
4. 4. Load the Total Number of Blocks register with the total number blocks.
    a. a. Total Number Blocks = 45 x 80 = 3600
5. Load the Global Scaling registers
    a. Global Y Width Scaling = (1/80)*65536 = 819
    b. Global Y Height Scaling = (1/45)*65536 = 1456
6. Load the Block Scaling registers
    a. Block Y Scaling = (1/(16x16))*65536 = 256

b. Block C Scaling = (4/(16x16))*65536 = 1024 for 4:2:0 format data

c. Block C Scaling = (2/(16x16))*65536 = 512 for 4:2:2 format data

Characterization Initialization

The first step to programming the Characterization registers is to calculate the Hue and Saturation values for the specified Cr and Cb values. See Color Conversion for the equations.

- C1: Cb = 44, Cr = 142
  - Cb1' = 44 - 128 = -84
  - Cr1' = 142 - 128 = 14
  - Sat1 = sqrt((-84)^2 + (14)^2) = 85
  - Hue1 = (256 * atan2(14, -84)) / 2π = 121
- C2: Cb = 72, Cr = 58
  - Cb2' = 72 - 128 = -56
  - Cr2' = 58 - 128 = -70
  - Sat2 = sqrt((-56)^2 + (-70)^2) = 90
  - Hue2 = (256 * atan2(-70, -56)) / 2π = -91
- C3: Cb = 100, Cr = 212
  - Cb3' = 100 - 128 = -28
  - Cr3' = 212 - 128 = 84
  - Sat3 = sqrt((-28)^2 + (84)^2) = 89
  - Hue3 = (256 * atan2(84, -28)) / 2π = 77
- C4: Cb = 212, Cr = 114
  - Cb4' = 212 - 128 = 84
  - Cr4' = 114 - 128 = -14
  - Sat4 = sqrt((84)^2+ (-14)^2) = 85
  - Hue4 = (256 * atan2(-14, 84)) / 2π = -7

When using the color selects in the Image Characterization core, a color range is defined by specifying a Saturation Maximum/Minimum and a Hue Maximum/Minimum. For this example margin of +/- 10 is added to the Saturation and Hue values to define the various color ranges. In reality, the size of the color range will be highly application specific and may need to be adjusted to maximize performance.

When initializing the Characterization registers following these steps:

1. Load the High Frequency Gain register
    a. High Frequency Gain register = 1
    b. The example statement did not mention emphasizing the High Frequency so the gain was set to 1.
2. Load the Edge Gain register
    a. Horizontal Edges Gain = 4
    b. Vertical Edges Gain = 1
    c. Left Diagonal Edge Gain = 1
    d. Right Diagonal Edge Gain = 1
    e. The example statement mentioned emphasizing Horizontal edges.
3. Load Colr Selects
    a. Color Select #1
        i. Saturation Max = 95
        ii. Saturation Min = 75
        iii. Hue Max = 131
        iv. Hue Min = 111
    b. Color Select #2
        i. Saturation Max = 100
        ii. Saturation Min = 80
        iii. Hue Max = -81
        iv. Hue Min = -101
    c. Color Select #3
        i. Saturation Max = 99
        ii. Saturation Min = 79
        iii. Hue Max = 87
        iv. Hue Min = 67
    d. Color Select #4
        i. Saturation Max = 95

ii. Saturation Min = 75

iii. Hue Max = 3

iv. Hue Min = -17

Use Model

Figure 3-16 shows an example of using the Image Characterization core in a larger system. In this setup, the Motion Adaptive Noise Reduction (MANR) LogiCORE IP calculates the motion in the video and drives the input video data bus of the Image Characterization core. The Image Characterization core writes the calculated statistics to memory via the AXI4 S2MM port. The statistics are then read by the Object Segmentation block for higher level analysis and processing. Such a system can be easily built using the building blocks provided by Xilinx (VDMA, Timing Controller, OSD, etc.).

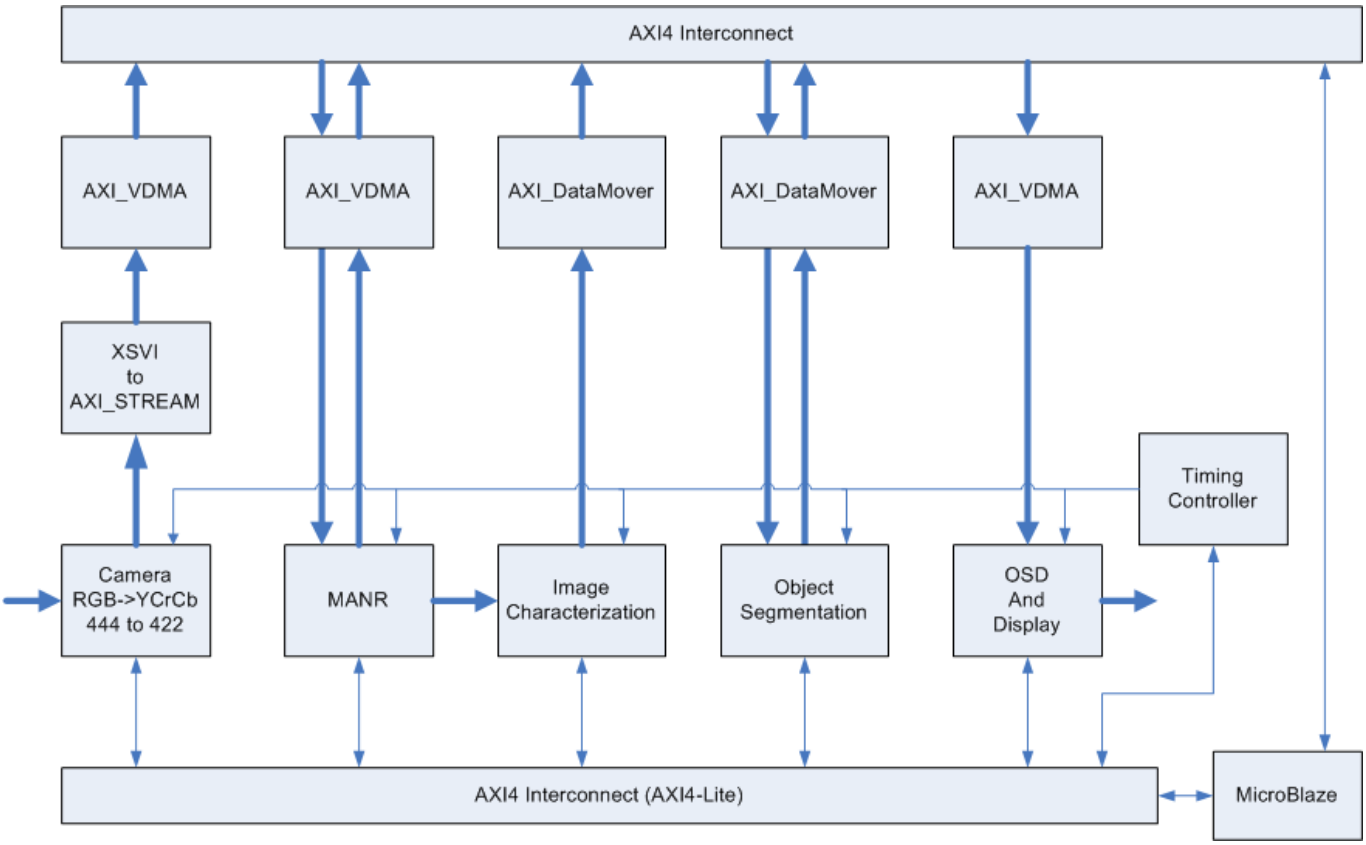

Figure 3-16: **Image Characterization Example Use Model**

Clocking

ACLK and Video_Clk

When configured in Sync Video Clock mode, there is only the aclk signal which is used to clock the video input interface as well as the core logic. When configured in Async Video Clock mode, the aclk signal is still present, but is only used to clock the core logic. An additional clock signal called `video_clk` is used to clock the video input interface. The Async Video Clock mode should be selected when the core clock rate is to be run at a higher rate than the input video clock rate.

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 IC 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 complete even if the video processing is stalled with `ARESETn`, `ACLKEN` or with the video clock not running.

Resets

ARESETn

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

IMPORTANT: *ARESETn is not 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.

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

S_AXI_ARESETn

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

System Considerations

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

If the target system uses only one setup of the IC core, you may choose to create a constant configuration by removing the AXI4-Lite interface. This reduces the core Slice footprint.

Clock Domain Interaction

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

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

Programming Sequence

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

Error Propagation and Recovery

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

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

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

C Model Reference

The LogiCORE IP Image Characterization v3.00a has a bit accurate C model for 32-bit and 64-bit Windows, as well as 32-bit and 64-bit Linux platforms. The model has an interface consisting of a set of C functions, which reside in a statically link library (shared library). Full details of the interface are provided in Interface. An example piece of C code is provided to show how to call the model.

The model is bit accurate because it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate because it does not model the core's latency or its interface signals.

The latest version of the model is available for download on the LogiCORE IP Image Characterization Web page at: http://www.xilinx.com/products/ipcenter/EF-DI-IMG-CHAR.htm

Unpacking and Model Contents

Unzip the `v_ic_v3_00_a_bitacc_model.zip` file, containing the bit accurate models for the Image Characterization IP Core. This creates the directory structure and files in Table 4-1.

Table 4-1: **Directory Structure and Files of the Image Characterization v3.00a Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
pg015_v_ic.pdf	LogiCORE IP Image Characterization Product Guide
v_ic_v3_00_a_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image/video container type and support functions
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions
yuv_utils.h	Header file declaring the YUV image/video container type and support functions
image_char_stats_utils.h	Header file declaring the Image Characterization Statistics container type and support functions
run_bitacc_cmodel.c	Example code calling the C model

Table 4-1: **Directory Structure and Files of the Image Characterization v3.00a Bit Accurate C Model**

example_420_512x512.yuv	Example YUV input file
ic_config_512.cfg	Example configuration file
/lin32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_ic_v3_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1.so.5.1	STL library, referenced by libIp_v_ic_v3_00_a_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_ic_v3_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_ic_v3_00_a_bitacc_cmodel.so
/win32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms
libIp_v_ic_v3_00_a_bitacc_cmodel.lib	Precompiled library file for Win32 compilation
/win64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms
libIp_v_ic_v3_00_a_0_bitacc_cmodel.lib	Precompiled library file for Win64 compilation

Installation

For Linux, make sure the following files are in a directory that is in your $LD_LIBRARY_PATH environment variable:

- libIp_v_ic_v3_00_a_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Image Characterization v3.00a C models were compiled and tested with the software listed in Table 4-2.

Table 4-2: **Compilation Tools for the Bit Accurate C Models**

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

Interface

The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file `v_ic_v3_00_a_bitacc_cmodel.h`

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

```
struct  xilinx_ip_v_ic_v3_00_a_generics ic_generics;
struct  xilinx_ip_v_ic_v3_00_a_inputs   ic_inputs;
struct  xilinx_ip_v_ic_v3_00_a_outputs  ic_outputs
```

The declaration of the above structures are in the `v_ic_v3_00_a_bitacc_cmodel.h` file.

Calling `xilinx_ip_v_ic_v3_00_a_get_default_generics` (and `ic_generics`) initializes the generics structure with the default values for each element of the structure. The generics defaults are:

```
int has_axi4_lite;     // Has AXI4-Lite Processor Interface
int has_intc_if;       // HAS INTC_IF debug port
int has_debug;         // Has Debug circuit
int frames;            // Number of Frames
int max_frame_width;   // Maximum Frame Width
int max_frame_height;  // Maximum Frame Height
int min_block_size;    // Minimum Block size
int chroma_format;     // Chroma Format: 0=4:2:0, 1=4:2:2
int chroma_filter;     // Input Chroma Filter
int chroma_parity;     // Chroma Polarity for 4:2:0 video (default = 1)
int stats_start_addr0; // Address #1 for Stats Output Buffer
int stats_start_addr1; // Address #2 for Stats Output Buffer
int frame_width;       // Frame Width
int frame_height;      // Frame Height
block_size = 8;     // Block Size (4,8,16,32,64)
int num_total_blocks;  // Number of total blocks
int num_blocks_wide;   // Number of blocks horizontal
int num_blocks_high;   // Number of blocks vertical
gyw_scale = 410;    // Global Y Width scaling
gyh_scale = 728;    // Global Y Height scaling
by_scale = 1024;    // Block Y scaling
bc_scale = 4096;    // Block C scaling
int hf_gain;           // High Frequency Gain
int edge_h_gain;       // Edge Gain Horizontal
int edge_v_gain;       // Edge Gain Vertical
int edge_r_gain;       // Edge Gain Right Diagonal
int edge_l_gain;       // Edge Gain Left Diagonal
int cs1_hue_min;       // Color Select Thresholds #1
int cs1_hue_max;
int cs1_sat_min;
int cs1_sat_max;
int cs2_hue_min;       // Color Select Thresholds #2
int cs2_hue_max;
int cs2_sat_min;
```

```
int cs2_sat_max;
int cs3_hue_min;       // Color Select Thresholds #3
int cs3_hue_max;
int cs3_sat_min;
int cs3_sat_max;
int cs4_hue_min;       // Color Select Thresholds #4
int cs4_hue_max;
int cs4_sat_min;
int cs4_sat_max;
int cs5_hue_min;       // Color Select Thresholds #5
int cs5_hue_max;
int cs5_sat_min;
int cs5_sat_max;
int cs6_hue_min;       // Color Select Thresholds #6
int cs6_hue_max;
int cs6_sat_min;
int cs6_sat_max;
int cs7_hue_min;       // Color Select Thresholds #7
int cs7_hue_max;
int cs7_sat_min;
int cs7_sat_max;
int cs8_hue_min;       // Color Select Thresholds #8
int cs8_hue_max;
int cs8_sat_min;
int cs8_sat_max;
use_y_mean_var = 1;    // Calculate Y Mean and Variance Values
use_u_mean_var = 1;    // Calculate U Mean and Variance Values
use_v_mean_var = 1;    // Calculate V Mean and Variance Values
use_mot_mean_var = 1;  // Calculate Motion Mean and Variance Values
use_freq_mean_var = 1; // Calculate Frequency Mean and Variance Values
use_edge_mean_var = 1; // Calculate Edge Mean and Variance Values
use_sat_mean_var = 1;  // Calculate Saturation Mean and Variance Values
num_color_selects = 8; // Number of Color Selects (0, 4 or 8)
use_y_histogram = 1;   // Calculate Y Histogram
use_u_histogram = 1;   // Calculate U Histogram
use_v_histogram = 1;   // Calculate V Histogram
use_hue_histogram = 1; // Calculate Hue Histogram
```

The structure `ic_inputs` defines the values of the input image. For a description of the input structure, see Image Characterization Video Input Structure.

The structure `ic_outputs` defines the values of the output image characterization statistics. For a description of the output structure, see Image Characterization Statistics Output Structure.

Note: The `video_in` and `video_out` variables are not initialized, as the initialization depends on the actual test to be simulated. The next chapters describe the initialization of the `video_in` and `video_out` structures.

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

```
int xilinx_ip_v_ic_v3_00_a_bitacc_simulate(
      struct xilinx_ip_v_ic_v3_00_a_generics* generics,
      struct xilinx_ip_v_ic_v3_00_a_inputs*   inputs,
      struct xilinx_ip_v_ic_v3_00_a_outputs*  outputs).
```

Results are provided in the outputs structure. After the outputs are evaluated and saved, dynamically allocated memory for input and output video structures must be released by calling the function:

```
void xilinx_ip_v_ic_v3_00_a_destroy(
       struct xilinx_ip_v_ic_v3_00_a_inputs *input,
       struct xilinx_ip_v_ic_v3_00_a_outputs *output)
```

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

Image Characterization Video Input Structure

Input images or video streams can be provided to the Image Characterization v3.00a reference model using the general purpose `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: **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*
Cols	Number of columns per frame*
Bit_per_component	Number of bits per color channel/component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
Mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table 4-4. Modes FORMAT_C420_M and FORMAT_C422_M are supported for this core.
Data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

*Pertaining to the image plane with the most rows and columns, such as the luminance channel for YUV data. Frame dimensions are assumed constant through all frames of the video stream, however, different planes, such as Y,U and V can have different dimensions.

Table 4-4: **Named Constants for Video Modes With Corresponding Planes and Representations (1)**

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

1. Modes FORMAT_C420_M and FORMAT_C422_M are supported for this core

Initializing the Image Characterization Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `yuv_utils.h` and `video_utils.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O.

The Image Characterization reference model expects to input a video structure with a mode of FORMAT_C420_M or FORMAT_C422_M with the video data mapped into the video structure as shown in Table 4-5.

Table 4-5: **Video Structure Mapping**

Video Structure Plane	Video Data
0	Y
1	U
2	V
3	Motion

Binary Image/Video Files

The header `video_utils.h` declares functions that load and save generalized video files in raw, uncompressed format. The following functions serialize the `video_struct` structure:

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

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

Working With Video_struct Containers

The header file `video_utils.h` defines functions to simplify access to video data in `video_struct`.

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

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 following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in variable `in_video`, with the following construct:

```
for (int frame = 0; frame < in_video->frames; frame++) {
  for (int plane = 0; plane < video_planes_per_mode(in_video->mode); plane++) {
    for (int row = 0; row < rows_per_plane(in_video,plane); row++) {
      for (int col = 0; col < cols_per_plane(in_video,plane); col++) {
   // User defined pixel operations on
// in_video->data[plane][frame][row][col]
      }
```

YUV Binary Image/Video Files

The header `yuv_utils.h` file declares functions that help load and save YUV video files in raw, uncompressed format. The following functions serialize the `yuv8_video_struct` and `yuv_video_struct` structures:

```
int read_yuv8(FILE* infile, struct yuv8_video_struct* yuv_video);
int write_yuv8(FILE* outfile, struct yuv8_video_struct* yuv_video);
int read_yuv(FILE* infile, struct yuv_video_struct* yuv_video);
int write_yuv(FILE* outfile, struct yuv_video_struct* yuv_video);
```

The YUV8 functions are used with 8-bit YUV data; the YUV functions are used with 16-bit YUV data. The corresponding file contains binary data, 16-bits or 8-bits per component, stored one frame after the other. Each frame is stored in planer format starting with the Y plane, followed by the U plane, and ending with the V plane.

Working With Yuv_video_struct Containers

The header file `yuv_utils.h` defines functions to simplify access to video data in `yuv_video_struct` and `yuv8_video_struct`.

```
int alloc_yuv8_frame_buff(struct yuv8_video_struct* yuv8video );
int alloc_yuv_frame_buff(struct yuv_video_struct* yuv_video );
void free_yuv_frame_buff(struct yuv_video_struct* yuv_video );
int copy_yuv8_to_video(struct yuv8_video_struct* yuv_in,
                       struct video_struct* video_out );
int copy_yuv_to_video(struct yuv_video_struct*  yuv_in,
                      struct video_struct* yideo_out );
int copy_video_to_yuv8(struct video_struct* video_in,
                       struct yuv8_video_struct* yuv_out );
int copy_video_to_yuv(struct video_struct* video_in,
                      struct yuv_video_struct*  yuv_out  );
```

The `alloc_yuv*_frame_buff` and `free_yuv_frame_buff` functions can be used to dynamically allocate and clear `yuv*_video_struct` structures. The copy functions can be used to copy YUV data between `yuv*_video_struct` and `video_struct` structures. These routines are important because the Image Characterization core only accepts `video_structs`.

Image Characterization Statistics Output Structure

The Image Characterization v3.00a reference model outputs a set of image characterization statistics for each frame that is processed. The output statistics are provided using the `image_char_stats_struct` structure, defined in the `image_char_stats_utils.h` file:

```
struct image_char_stats_struct
{
  int   frames;                           // Number of frames
  int   num_blocks_wide;                  // Number of blocks wide
  int   num_blocks_high;                  // Number of blocks high
  int*  frame_index;                      // Frame Index for each frame
  struct global_stats_struct** global;    // Global stats
  struct block_stats_struct*** block;     // Block stats
  int** y_histogram;                      // Y Histogram
  int** u_histogram;                      // U Histogram
  int** v_histogram;                      // V Histogram
  int** hue_histogram;                    // Hue Histogram
};

struct global_stats_struct
{
  uint8   y_mean;      // Y mean
  uint8   u_mean;      // U mean
  uint8   v_mean;      // V mean
  uint8   m_mean;      // Motion mean
  uint8   e_mean;      // Edge mean
  uint8   lp_mean;     // Low Frequency mean
  uint8   hp_mean;     // High Frequency mean
  uint8   sat_mean;    // Saturation mean
```

```
  uint16  y_var;       // Y variance
  uint16  u_var;       // U variance
  uint16  v_var;       // V ariance
  uint16  m_var;       // Motion variance
  uint16  e_var;       // Edge variance
  uint16  lp_var;      // Low Frequency variance
  uint16  hp_var;      // High Frequency variance
  uint16  sat_var;     // Saturation variance
};

struct block_stats_struct
{
  uint8   y_mean;       // Y mean
  uint8   u_mean;       // U mean
  uint8   v_mean;       // V mean
  uint8   m_mean;       // Motion mean
  uint8   e_mean;       // Edge mean
  uint8   lp_mean;      // Low Frequency mean
  uint8   hp_mean;      // High Frequency mean
  uint8   sat_mean;     // Saturation mean
  uint16  y_var;        // Y variance
  uint16  u_var;        // U variance
  uint16  v_var;        // V ariance
  uint16  m_var;        // Motion variance
  uint16  e_var;        // Edge variance
  uint16  lp_var;       // Low Frequency variance
  uint16  hp_var;       // High Frequency variance
  uint16  sat_var;      // Saturation variance
  uint16  color_sel[8]; // Color Select (x8)
};
```

The `image_char_stats_struct` can hold the results of multiple processed frames. The number of frames in the structure is specified in the frames element of the structure. The `num_blocks_wide` and `num_blocks_high` elements denote the width and height of the 2-D grid of block statistics that are stored for each frame of statistics. The `frame_index` is an array with one value per frame; it holds the index values of each frame. The global element is an array of `global_stats_structs`; there is one structure per frame. It holds the global statistics as defined in the `global_stats_struct`. The block element is a 3-D grid of `block_stats_structs`. The first dimension is based on the number of frames, the second dimension is based on the `num_blocks_high`, and the third dimension is based on the `num_blocks_wide`. Each point of the grid is an instance of the `block_stats_struct` that holds the block statistics for each block of each frame. The `y_histogram`, `u_histogram`, `v_histogram` and `hue_histogram` are each 2-D arrays. The first dimension is based on the frames, and the second dimension is an array of 256 elements. They each hold a corresponding 256 bin histogram for each frame.

Working With Image_char_stats_struct Containers

The header file `image_char_stats_utils.h` defines functions to simplify the use of the image characterization statistics structures.

```
int alloc_ic_stats_buff(struct image_char_stats_struct* ic_stats);
void free_ic_stats_buff(struct image_char_stats_struct* ic_stats);
```

```
int write_ic_stats(FILE *output_fid, struct image_char_stats_struct *stats);
```

The `alloc_ic_stats_buff` function can be used to dynamically create an `image_char_stats_struct`. The frame, `num_blocks_wide` and `num_blocks_high`, elements of the structure must be specified before calling this routine. The `free_ic_stats_buff` function can be used to destroy an `image_char_stats_struct`.

The `write_ic_stats` function writes the `image_char_stats_struct` to a text file. Each frame of statistics is written to the file in this order:

1. Structure header
2. Global statistics
3. Histograms (Y, U, V and Hue)
4. Block statistics (each column of each row)

The output matches the format of the output of the Image Characterization core. See Image Characterization Output in Chapter 3 for more information.

C Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided and has these characteristics:

- Contains an example of how to write an application that makes a function call to the Image Characterization C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information for use by the model.
- Writes the Image Characterization statistics to an output file.

After following the compilation instructions in this chapter, you should run the example executable. If invoked with insufficient parameters, the following help message is generated:

```
Usage: run_bitacc_cmodel in_file config_file out_file

   in_file        :  Path/name of the input file - must be .yuv.
   config_file    :  Path/name of the configuration file - must be .cfg.
   out_file       :  Path/name of the output IC Stats file.
```

Config File Format

During successful execution, the specified config file is parsed by the run_bitacc_cmodel example. In this file, you must specify:

- Input video format
- Image Characterization initialization parameters
- Image Characterization statistics that are generated

The example config file provided in the zip file provides more information on the formatting of this file.

Initializing the Image Characterization Input Video Structure

In the example code wrapper, data is assigned to a video structure by reading from a .yuv video file. The `yuv_util.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O. The run_bitacc_cmodel example code uses these functions to read from the delivered YUV file.

YUV Image Files

The header `yuv_utils.h` file declares functions that help access files in standard YUV format. It operates on images with three planes (Y, U and V). The following functions operate on arguments of type yuv8_video_struct, which is defined in `yuv_utils.h`.

```
int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video);
int read_yuv8(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

Exchanging data between `yuv8_video_struct` and general `video_struct` type frames/videos is facilitated by functions:

```
int copy_yuv8_to_video(struct yuv8_video_struct* yuv8_in,
                       struct video_struct* video_out );
int copy_video_to_yuv8( struct video_struct* video_in,
                        struct yuv8_video_struct* yuv8_out );
```

Note: All image/video manipulation utility functions expect both input and output structures to be initialized. For example, pointing to a structure to which memory has been allocated, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data[] or y[],u[],v[]) structures already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and generate 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 create the appropriate container to hold results.

C Model Example I/O Files

Input Files

- <**in_filename>.yuv** (for example, `video_in.yuv`, `video_in_128x128.yuv`).
  - Standard 8-bit YUV file format. Entire Y plane followed by entire Cb plane, followed by entire Cr plane.
  - Can be viewed in a YUV player, such as YUVPlayer.
  - No header.
- <**config_file>.cfg** (for example, `ic_config_512.cfg`).
  - Image Characterization configuration.

Output Files

- <**output_filename>.txt** (for example, `ic_stats_out.txt`).
  - Image Characterization statistics.

Compiling Image Characterization C Model With Example Wrapper

Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip file, as shown in this example:

```
setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
```

2. Copy these files from the /lin (for 32-bit) or from the /lin64 (for 64-bit) directory to the root directory:

   `libstlport.so.5.1`

   `libIp_v_ic_v3_00_a_bitacc_cmodel.so`

3. In the root directory, compile with the GNU C Compiler using this command:

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_ic_v3_00_a_bitacc_cmodel -Wl,-rpath,.
```

4. This command creates the executable run_bitacc_cmodel, which can be run with this command:

```
./run_bitacc_cmodel example_420_512x512.yuv ic_config_512.cfg
ic_stats_out.txt
```

Windows (32-bit and 64-bit)

Precompiled `library v_ic_v3_00_a_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. The following is an example using Microsoft Visual Studio. In Visual Studio create a new, empty Win32 Console Application project. To existing items, add:

- **libIp_v_ic_v3_00_a_bitacc_cmodel.lib** to the "Resource Files" folder of the project
- **run_bitacc_cmodel.c** to the "Source Files" folder of the project
- **v_ic_v3_00_a_bitacc_cmodel.h** to "Header Files" folder of the project
- **yuv_utils.h** to the "Header Files" folder of the project
- **rgb_utils.h** to the "Header Files" folder of the project
- **video_utils.h** to the "Header Files" folder of the project
- **image_char_stats_utils.h** to the "Header Files" folder of the project

After the project is created and populated, it must be compiled and linked (built) to create a Win32 or Win 64 executable. To perform the build step, select **Build Solution** from the Build menu. An executable matching the project name is created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" was selected in the "Configuration Manager" in the Build menu.

Running the Delivered Executables

Included in the zip file are precompiled executable files for use with Win32, Win64, Linux32, and Linux64 platforms.

Linux (32-bit and 64-bit)

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip file, as shown in this example:

```
setenv LD_LIBRARY_PATH <unzipped_c_model_dir>:${LD_LIBRARY_PATH}
```

2. Copy these files from the /lin or /lin64 directory to the root directory:

   `libstlport.so.5.1`

   `libIp_v_ic_v3_00_a_bitacc_cmodel.so`

   `run_bitacc_cmodel`

3. Execute the model:

```
./run_bitacc_cmodel example_420_512x512.yuv ic_config_512.cfg
ic_stats_out.txt
```

Windows (32-bit and 64-bit)

1. Copy `run_bitacc_cmodel.exe` from the /win32 or /win64 directory to the root directory:
2. Execute the model:

```
run_bitacc_cmodel example_420_512x512.yuv ic_config_512.cfg
ic_stats_out.txt
```

Image Characterization Statistics Output

This appendix provides information on the image characterization statistics output. For additional information, see Image Characterization Output in Chapter 3.

An output file can contain multiple sets of image characterization statistics data. The Frame Header of the next frame follows directly after the Block Statistics of the last block of the previous frame.

The following is an example of the image characterization statistics for one image frame.

```
### Frame #1 Header ###
ffffffff     # Frame Valid
00000001     # Frame Index
00000000     # Frame Header Padding (30 lines)
…
00000000
### Frame #1 Global Statistics ###
957d8395     # Low Frequency Mean = 0x95, V Mean = 0x7D,
             # U Mean = 0x83, Y Mean = 0x95
07001604     # Saturation Mean = 0x07, Motion Mean = 0x00,
             # Edge Mean = 0x16, High Frequency Mean = 0x04
0036007c     # U Variance = 0x0036, Y Variance = 0x007c
0077003a     # Low Frequency Variance = 0x0077, V Variance = 0x003a
02f1001d     # Edge Variance = 0x02f1, High Frequency Variance = 0x001d
00040000     # Saturation Variance = 0x0004, Motion Variance = 0x0000
00000000     # Global Statistics Padding (26 lines)
```

```
…
00000000
### Frame #1 Y Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 U Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 V Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 Hue Histogram ###
00000e71     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 Block #0 ([0][0]) ###
bc6f92bc     # Low Frequency Mean = 0x95, V Mean = 0x7D,
             # U Mean = 0x83, Y Mean = 0x
18000000     # Saturation Mean = 0x07, Motion Mean = 0x00,
             # Edge Mean = 0x16, High Frequency Mean = 0x
009100ac     # U Variance = 0x0036, Y Variance = 0x
00ac0046     # Low Frequency Variance = 0x0077, V Variance = 0x
00000000     # Edge Variance = 0x02f1, High Frequency Variance = 0x
000d0000     # Saturation Variance = 0x0004, Motion Variance = 0x
00000000     # Color Select 2 = 0x0000, Color Select 1 = 0x0000
00100010     # Color Select 4 = 0x0010, Color Select 3 = 0x0010
00000000     # Color Select 6 = 0x0000, Color Select 5 = 0x0000
00000000     # Color Select 8 = 0x0000, Color Select 7 = 0x0000
00000000     # Reserved
00000000     # Reserved
00000000     # Reserved
00000000     # Reserved
### Frame #1 Block #1 ([0][1]) ###
bd7091bd
17000000
004800a2
00a70046
00000001
00030000
00000000
```

```
00100010
00000000
00000000
00000000
00000000
00000000
00000000
…
# Frames #2 - #4094 follow the same format at Frame #0
…
### Frame #1 Block #4095 ([63][63]) ###
80817f80
02001307
0028002d
00400069
015a0020
00010000
0003000d
00100000
00000000
00000010
00000000
00000000
00000000
00000000
```

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

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

GUI

The Xilinx Image Characterization LogiCORE IP is easily configured to meet the developer's specific needs through the Vivado graphical user interface (GUI). This section provides a quick reference to the parameters that can be configured at generation time. The GUI is shown in Figure 5-1 through Figure 5-4.

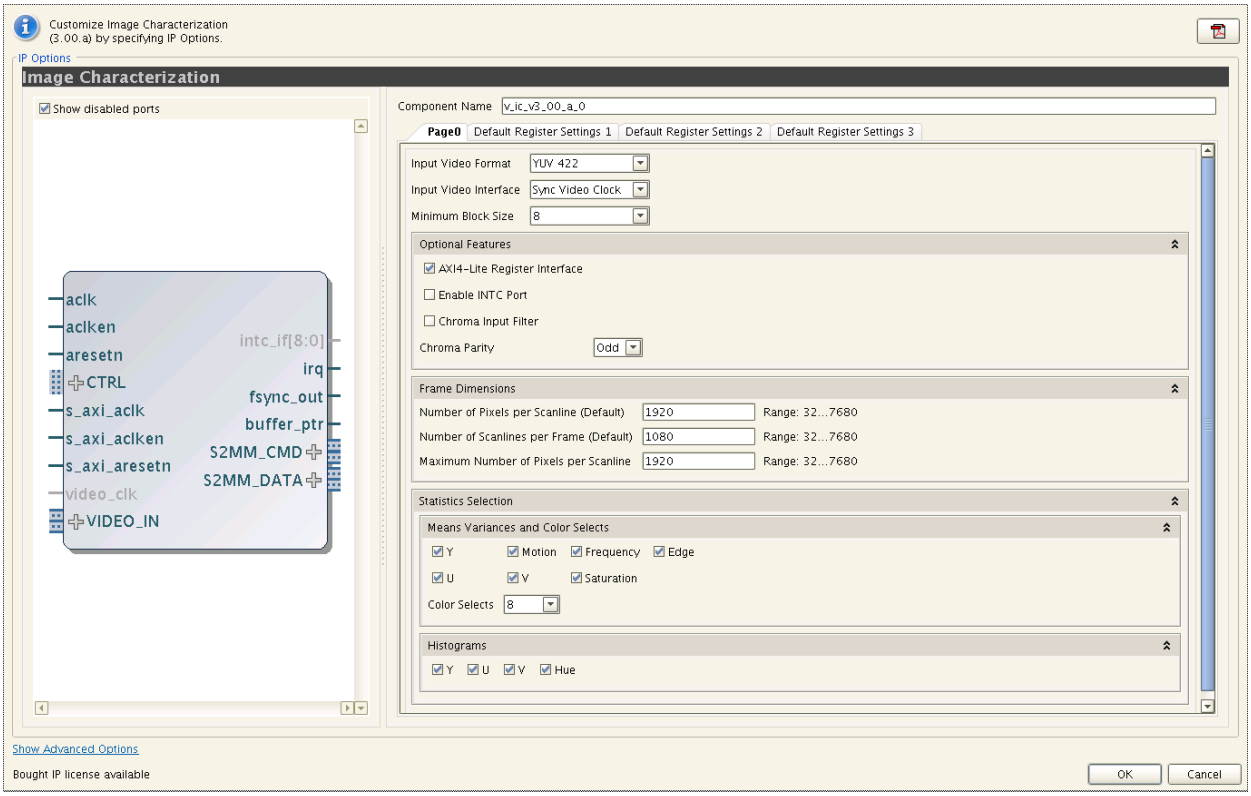

Figure 5-1: **Image Characterization Vivado IP Catalog - Main Window**

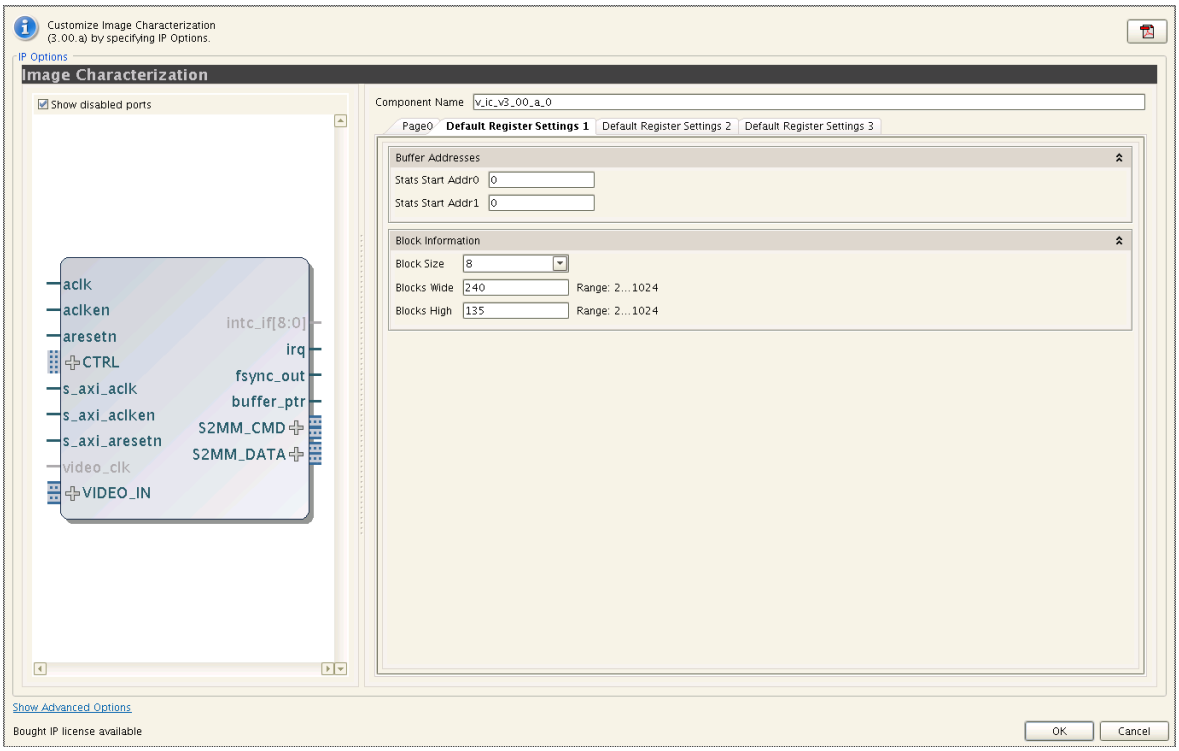

Figure 5-2: **Image Characterization Vivado IP Catalog - Default Registers (Page 1)**

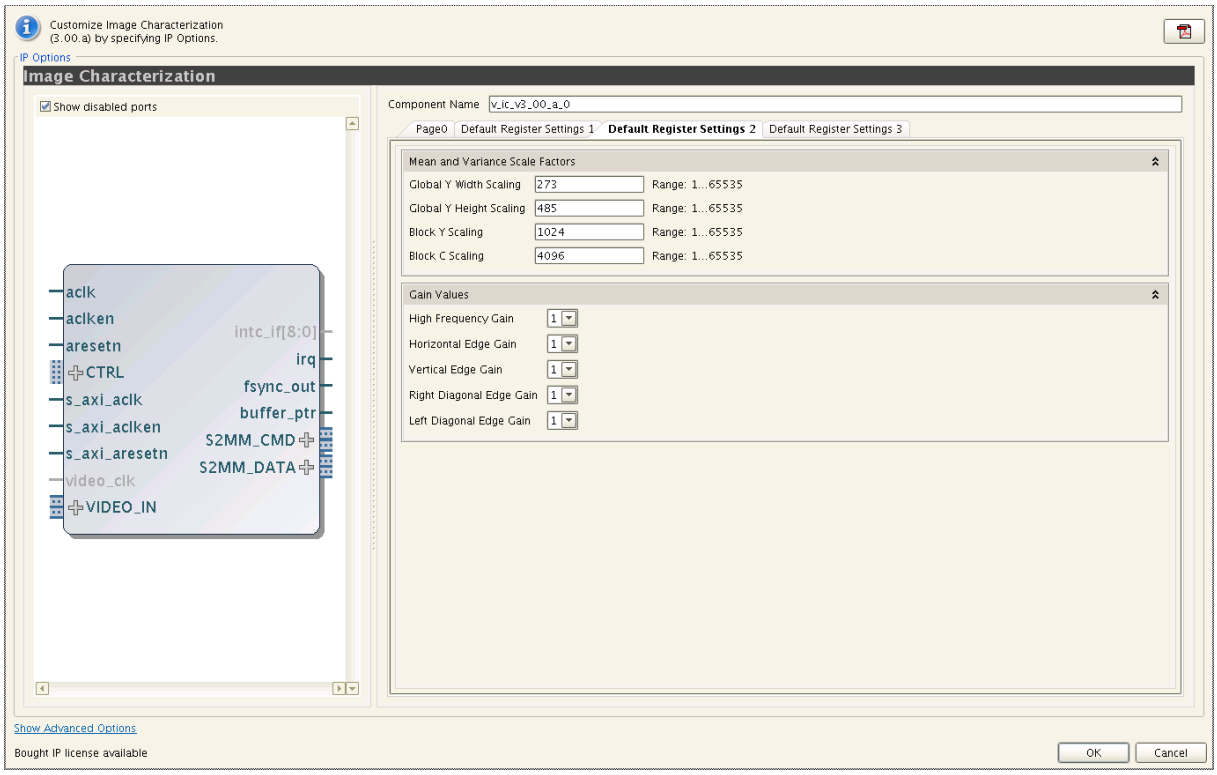

Figure 5-3: **Image Characterization Vivado IP Catalog - Default Registers (Page 2)**

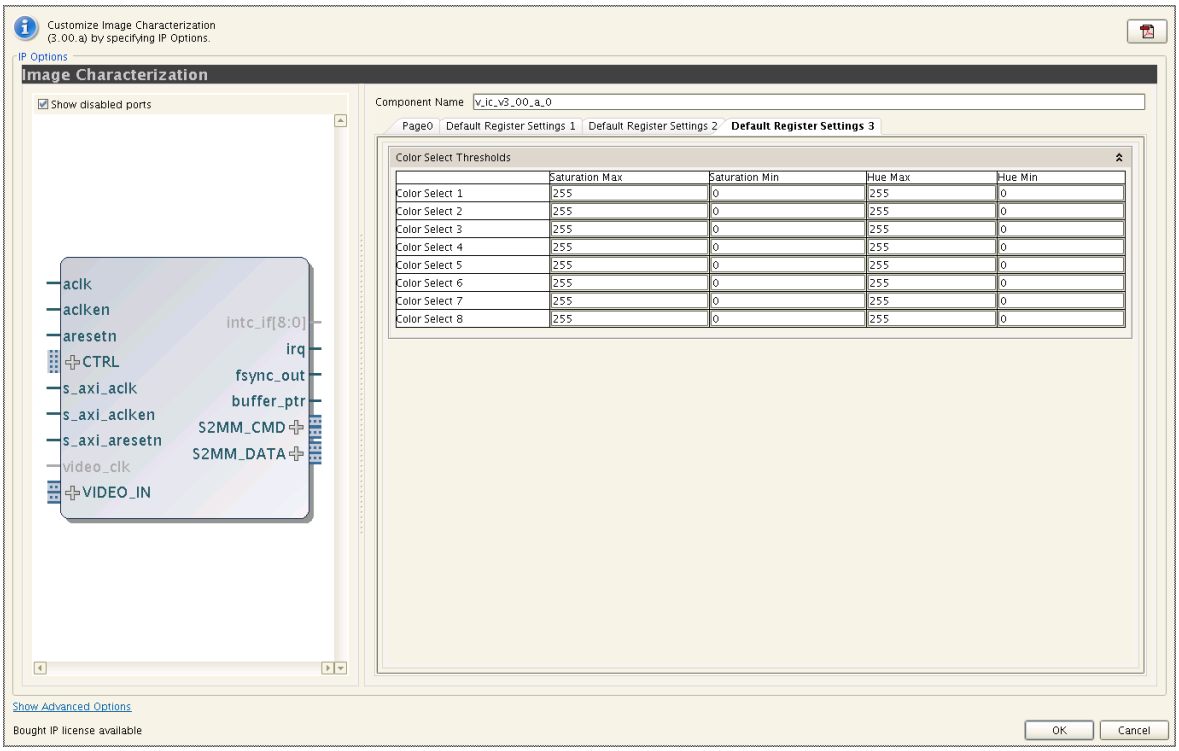

Figure 5-4: **Image Characterization Vivado IP Catalog - Default Registers (Page 3)**

Figure 5-1 displays the GUI parameters that affect the instantiation of the Image Characterization core. The selections on this page will affect the amount of resources that are used when the core is generated.

- **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 "_". Note: The name "v_ic_v3_00_a" is not allowed.
- **Input Video Format**: Sets the expected video format. The valid choices are YUV 4:2:0 and YUV 4:2:2. The selection is used to determine the Chroma format and does not affect resource utilization.
- **Input Video Interface**: Selects the clocking mode of the input video interface.
  - **Async Video Clock**: Select Async Video Clock when the video stream has a clock that is asynchronous to the Image Characterization core's processing clock. The video clock must be less than or equal to the rate of the core clock. The video_clk is a required input in this mode.
  - **Sync Video Clock**: Select Sync Video Clock when the video stream has a clock that is synchronous to the Image Characterization core's processing clock.
- **Minimum Block Size**: Sets the minimum block size that the core can use. Valid choices are 4, 8, 16, 32, and 64. The smaller the block size, the more resources that are used.
- **Optional Features**

- **AXI4-Lite Register Interface**: When selected, the core will be generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters.
- **Enable INTC Port**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions.
- **Chroma Input Filter**: When selected, the core will instantiate a 3-tap FIR filter to low pass filter the input Chroma data.
- **Chroma Parity**: Select odd if the first line of video contains chroma information. Chroma parity is only used for YUV 4:2:0 data.

- **Frame Dimensions**:
  - **Number of Pixels per Scanline (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the lower half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance processes.
  - **Number of Scanlines per Frame (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance processes.
  - **Maximum Number of Pixels per Scanline**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for Active Pixels per Scan line, or the corresponding lower half-word of the ACTIVE_SIZE register must always be less than or equal to the value provided by Maximum Number of Pixels per Scanline. Using a tight upper-bound results in optimal block RAM usage.
- **Statistics Selection**: Selected items will appear in the Image Characterization's statistics data structure. Unselected items will be replaced with a zero in the statistics data structure. Logic associated with an item is not instantiated when the item is unselected. Resources can be conserved by deselecting unneeded items.
  - **Means/Variances and Color Selects**
    - **Y**: When selected, global Y mean and variance values as well as block Y mean and variance values are included in the Image Characterization's statistics data structure.
    - **Motion**: When selected, global Motion mean and variance values as well as block Motion mean and variance values are included in the Image Characterization's statistics data structure.

- **Frequency**: When selected, global Frequency mean and variance values as well as block Frequency mean and variance values are included in the Image Characterization's statistics data structure. The frequency values include Low Frequency and High Frequency.
- **Edge**: When selected, global Edge Content mean and variance values as well as block Edge Content mean and variance values are included in the Image Characterization's statistics data structure.
- **U**: When selected, global U mean and variance values as well as block U mean and variance values are included in the Image Characterization's statistics data structure.
- **V**: When selected, global V mean and variance values as well as block V mean and variance values are included in the Image Characterization's statistics data structure.
- **Saturation**: When selected, global Saturation mean and variance values as well as block Saturation mean and variance values are included in the Image Characterization's statistics data structure.
- **Color Selects**: Sets the number of Color Select values that will be included in the Image Characterization's statistics data structure. Valid choices are 0, 4 and 8.

◦ **Histograms**

- **Y**: When selected, the Y Histogram is included in the Image Characterization's statistics data structure.
- **U**: When selected, the U Histogram is included in the Image Characterization's statistics data structure.
- **V**: When selected, the V Histogram is included in the Image Characterization's statistics data structure.
- **Hue**: When selected, the Hue Histogram is included in the Image Characterization's statistics data structure.

Figure 5-2 through Figure 5-4. displays the GUI parameters that affect the default register values of the Image Characterization core. When the AXI4-Lite interface is present, these values set the default values of the corresponding registers in the AXI4-Lite interface. When the AXI4-Lite interface is not present, these values are set permanently in the core.

- **Buffer Addresses**:
  - **Stats Start Addr0**: Specifies the Start Address of the first external memory buffer that will hold the Image Characterization data structure.
  - **Stats Start Addr1**: Specifies the Start Address of the second external memory buffer that will hold the Image Characterization data structure.
- **Block Information**:

- **Block Size**: Specifies the size of the block to be use during the Block Statistics calculations. Valid selections are 4, 8, 16, 32, or 64.
- **Blocks Wide:** Specifies the number of blocks in the horizontal direction. Calculated as Number of Pixels per Scanline / Block Size.
- **Blocks High**: Specifies the number of blocks in the vertical direction. Calculated as Number of Scanlines per Frame / Block Size.

- Mean and Variance Scale Factors
  - **Global Y Width Scaling**: Specifies the width scale factor for Global Luma and Chroma Means & Variances. Calculated as (1/Blocks Wide)*65536
  - **Global Y Height Scaling**: Specifies the height scale factor for Global Luma and Chroma Means & Variances. Calculated as (1/Blocks High)*65536
  - **Block Y Scaling**: Specifies the scale factor for Block Luma Means & Variances. Calculated as (1/Num_Block_Pixels)*65536
  - **Block C Scaling**: Specifies the scale factor for Block Chroma Means & Variances. Calculated as (2/Num_Block_Pixels)*65536 for YUV 4:2:2 or as (4/Num_Block_Pixels)*65536 for YUV 4:2:0.
- Gain Values
  - **High Frequency Gain**: Specifies the gain value for the High Frequency component. Valid selections are 1, 2, 4 or 8.
  - **Horizontal Edge Gain**: Specifies the gain value for the Horizontal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Vertical Edge Gain**: Specifies the gain value for the Vertical edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Right Diagonal Edge Gain:** Specifies the gain value for the Right Diagonal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Left Diagonal Edge Gain**: Specifies the gain value for the Left Diagonal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
- **Color Select Thresholds**
  - **Color Select 1 – 8**
    - **Saturation Max**: Specifies the maximum value for the Saturation thresholds.
    - **Saturation Min**: Specifies the minimum value for the Saturation thresholds.
    - **Hue Max**: Specifies the maximum value for the Hue thresholds.
    - **Hue Min**: Specifies the minimum value for the Hue thresholds.

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 Vivado tools software output consists of some or all the files shown in Table 5-1.

Table 5-1: **Output Files**

Name	Description
v_ic_v3_00_a	Library directory for the v_ ic_v3_00_a core
v_ ic_v3_00_a.veo	Verilog instantiation template
v_ ic_v3_00_a.vho	VHDL instantiation template
v_ ic_v3_00_a.xci	IP-XACT XML file describes which options were used to generate the core. An XCI file can also be used as a source file.
v_ ic_v3_00_a.xml	IP-XACT XML file describes how the core is constructed to build the core.

Constraining the Core

This chapter contains information about constraining the core in the Vivado™ Design Suite environment.

Required Constraints

The ACLK pin should be constrained at the desired pixel clock rate for your video stream. The S_AXI_ACLK pin should be constrained at the frequency of the AXI4-Lite subsystem. In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

XDC

```
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/
*SYNC2PROCCLK_I*/data_sync_reg[0]*"] -datapath_only 2
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/
*SYNC2VIDCLK_I*/data_sync_reg[0]*"] -datapath_only 2
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. For a complete listing of supported devices, see the release notes for this core. For a complete listing of supported devices, see the release notes for this core.

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for this core. See Chapter 2, Maximum Frequency. 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 this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Detailed Example Design

No example design is available at the time for the LogiCORE IP Image Characterization v3.00a core.

Demonstration Test Bench

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

Generating the Test Bench

1. After customizing the IP, right-click on the core instance in **Sources** pane and select **Generate Output Products** (Figure 7-1).

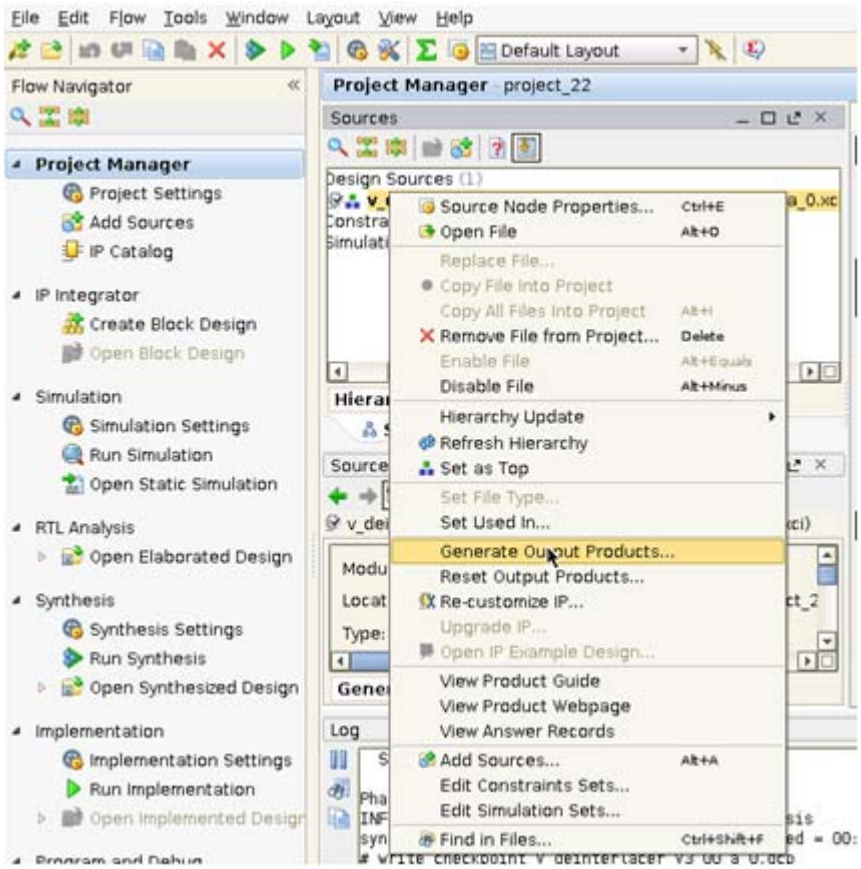

Figure 7-1: **Sources Pane**

A pop-up window prompts you to select items to generate.

2. Click on **Test Bench** and make sure **Action: Generate** is selected.

   The demo test bench package will be generated in the following directory (Figure 7-2):

   ```
   <PROJ_DIR>/<PROJ_NAME>.srcs/sources_1/ip/<IP_INSTANCE_NAME>/<IP_INSTANCE_NAME>/
   demo_tb/
   ```

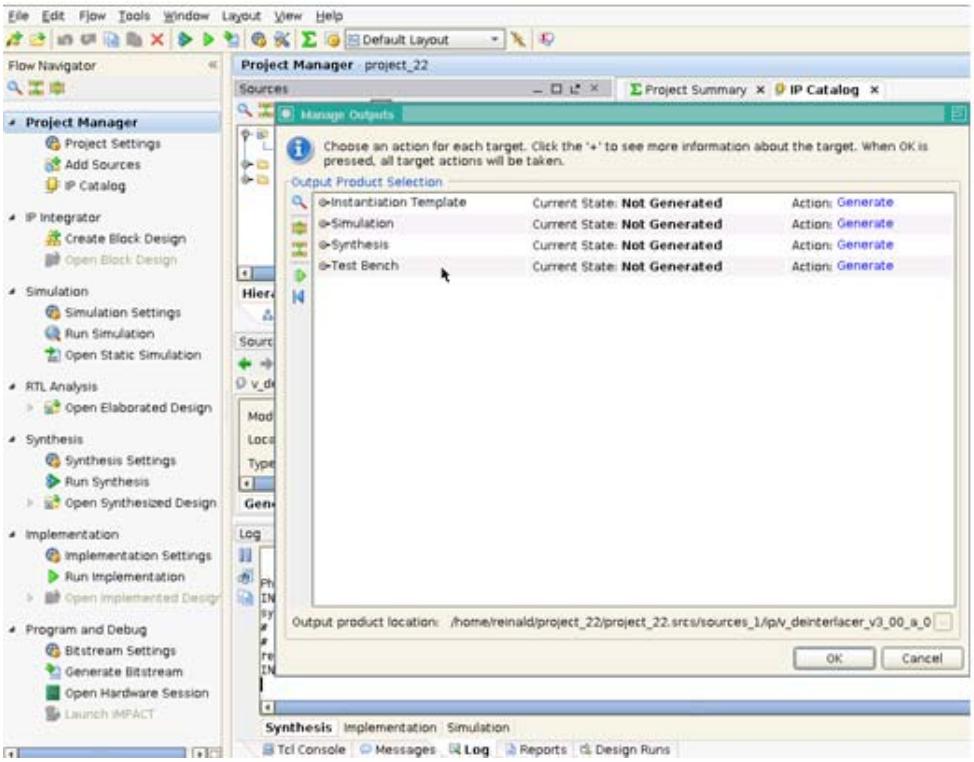

Figure 7-2: **Test Bench Generation**

Directory and File Contents

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

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

Test Bench Structure

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

It instantiates the following modules:

- `DUT`

  The <IP> core instance under test.

- `axi4lite_mst`

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

- `axi4s_video_mst`

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

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

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

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

  For information on how to generate stimuli files, refer to C Model Reference.

- `axi4s_video_slv`

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

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

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

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

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

- `ce_gen`

  Programmable Clock Enable (`ACLKEN`) generator.

Running the Simulation

There are two ways to run the demonstration test bench, after successfully generating the core output:

Option 1: Launch Simulation from the Vivado Design Suite GUI

This runs the test bench with the AXI4-Stream Master producing ramp data as stimuli, and AXI4-Stream Slave set to passive mode.

- Click **Simulation Settings** in the Flow Navigation window, change Simulation top module name to **tb_<IP_instance_name>**.
- Click **Run Simulation**. XSIM launches and you should be able to see the signals.
- You can also choose Modelsim for simulation by going to **Project Settings** and selecting Modelsim as the Target Simulator (Figure 7-3).

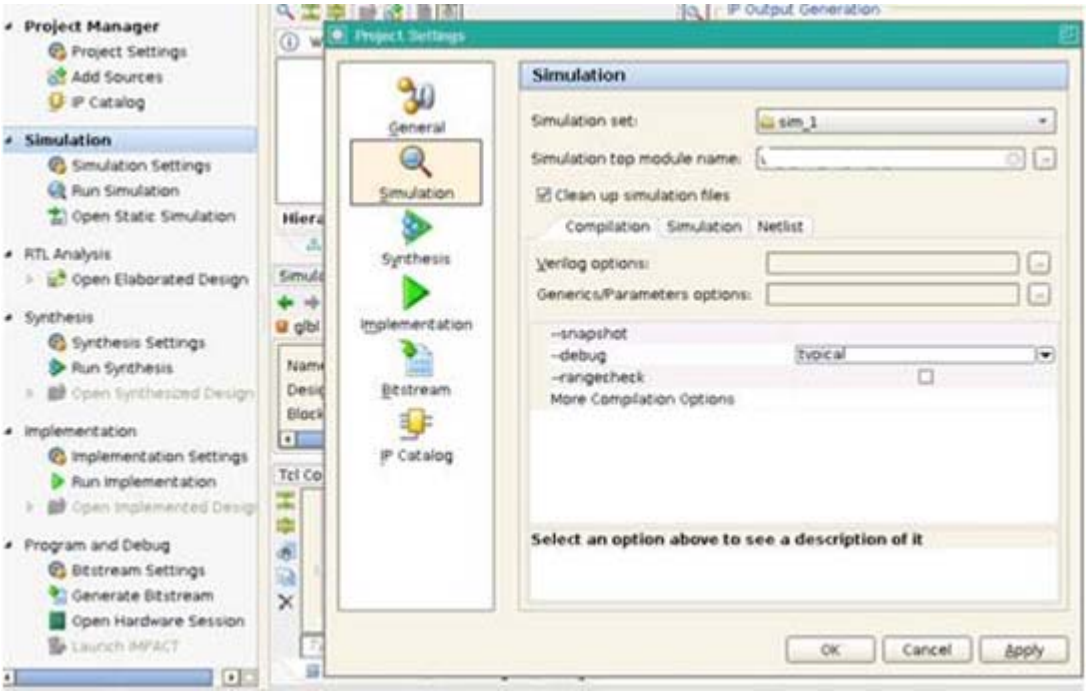

Figure 7-3: **Simulation GUI**

Option 2: Manually Compile and Run Simulation from Your Environment

- Add the generated test bench files to a new simulation set, along with the customized IP. For information on the location of generated test bench files, refer to Generating the Test Bench.
- Set up the environment variables for Xilinx libraries.
- Compile the generated IP.
- Compile the test bench files.
- Run the simulation.

RECOMMENDED: *To enable viewing a full-frame transaction, change the default simulation time from* ***1000 ns*** *to* ***all****.*

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 about using Xilinx tools to customize and generate the core in the ISE® Design Suite environment.

GUI

The Xilinx Image Characterization LogiCORE IP is easily configured to meet the developer's specific needs through the Vivado graphical user interface (GUI). This section provides a quick reference to the parameters that can be configured at generation time. The GUI is shown in Figure 8-1 through Figure 8-4.

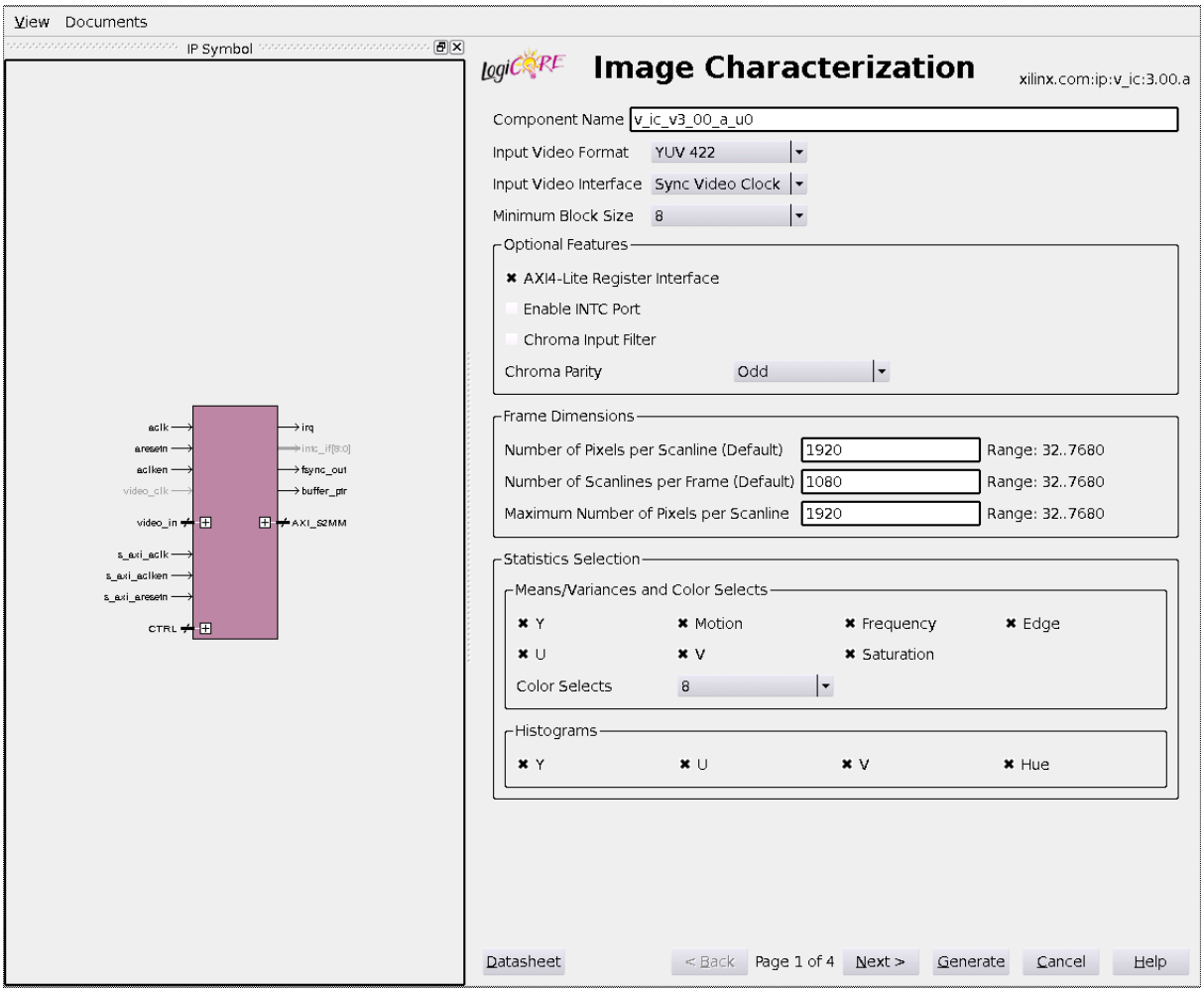

Figure 8-1: **Image Characterization CORE Generator GUI**

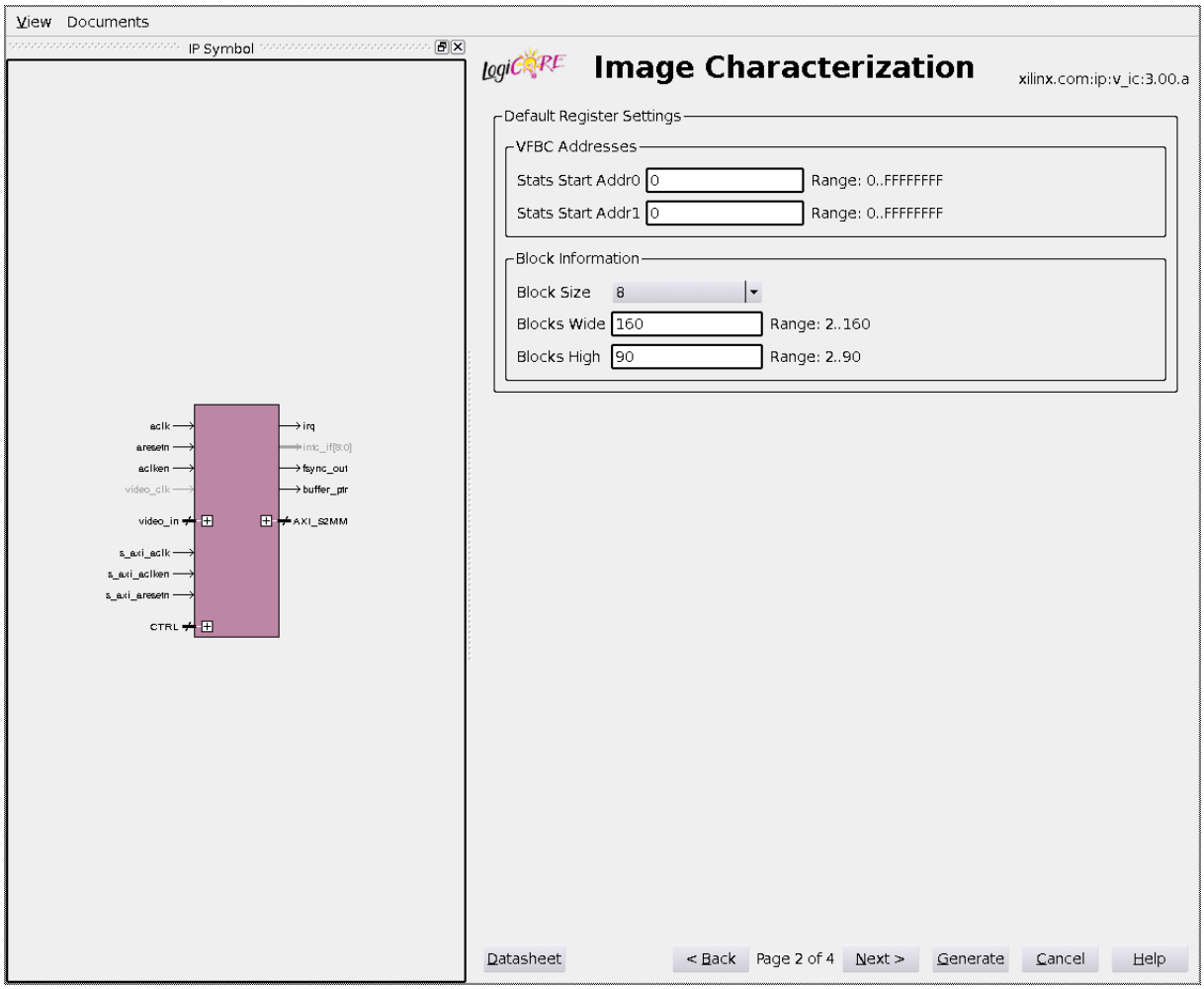

Figure 8-2: **Image Characterization CORE Generator GUI - Default Registers (Page 1)**

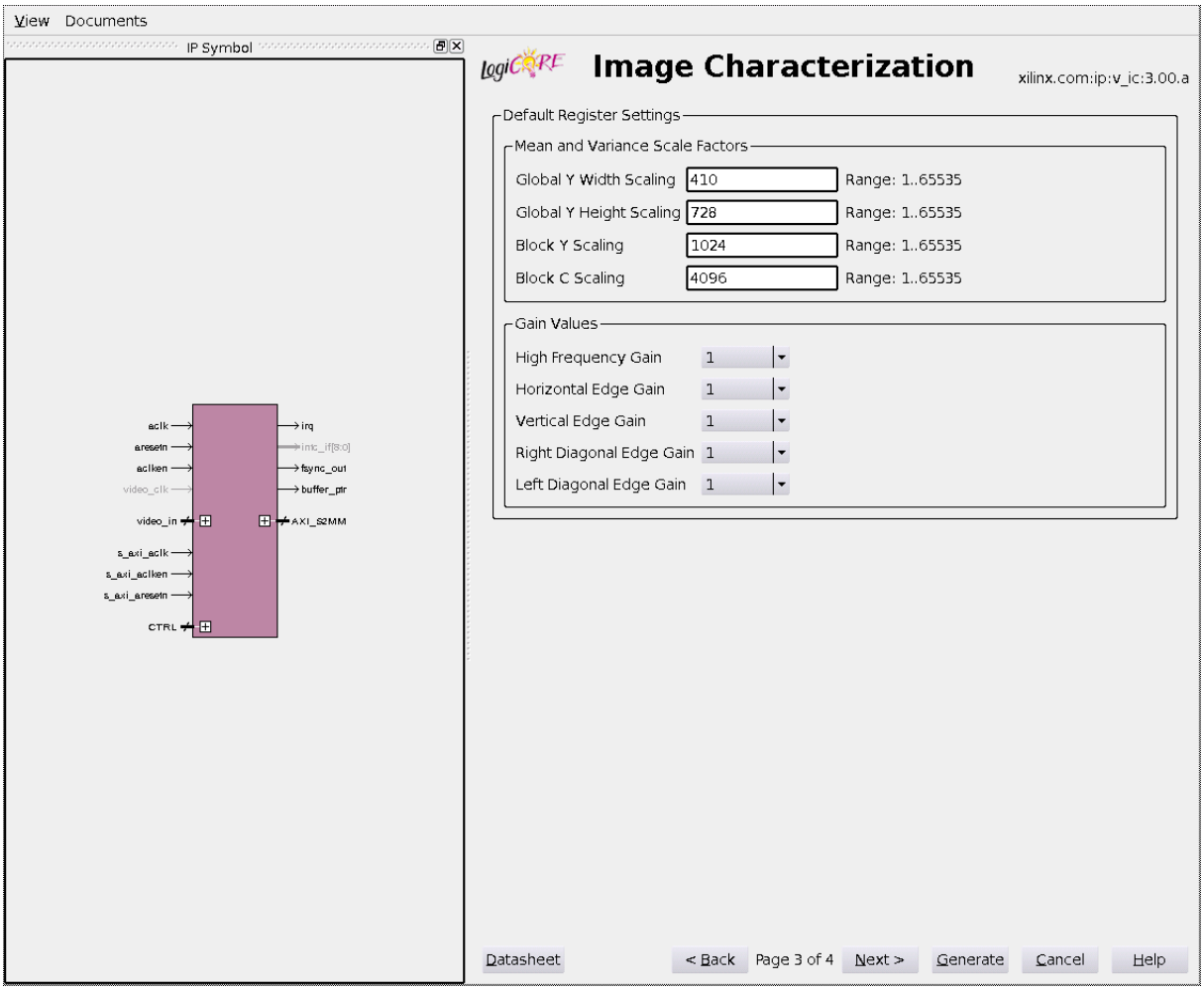

Figure 8-3: **Image Characterization CORE Generator GUI - Default Registers (Page 2)**

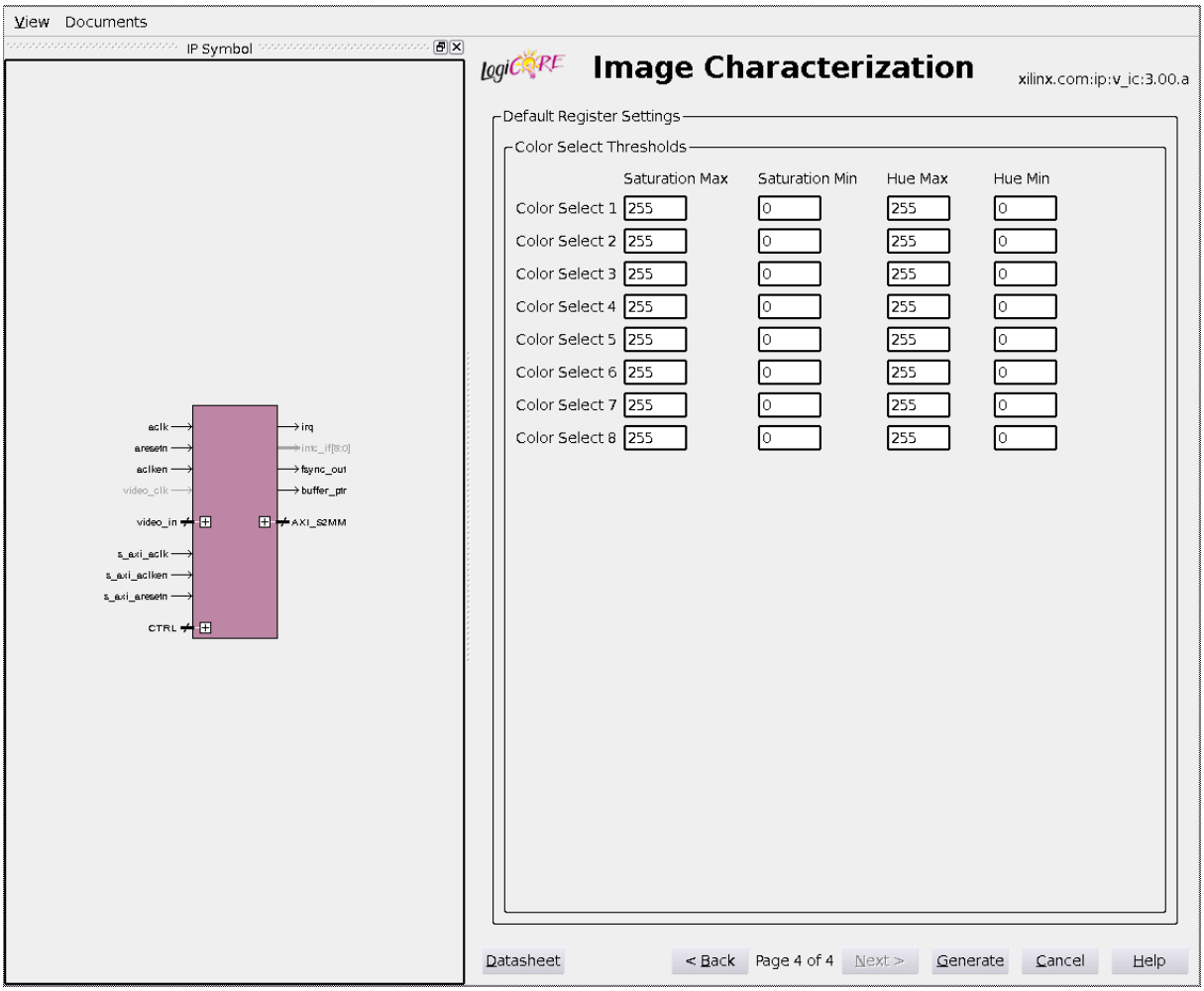

Figure 8-4: **Image Characterization CORE Generator GUI - Default Registers (Page 3)**

Figure 8-1 displays the GUI parameters that affect the instantiation of the Image Characterization core. The selections on this page will affect the amount of resources that are used when the core is generated.

- **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 "_". Note: The name "v_ic_v3_00_a" is not allowed.
- **Input Video Format**: Sets the expected video format. The valid choices are YUV 4:2:0 and YUV 4:2:2. The selection is used to determine the Chroma format and does not affect resource utilization.
- **Input Video Interface**: Selects the clocking mode of the input video interface.
  - **Async Video Clock**: Select Async Video Clock when the video stream has a clock that is asynchronous to the Image Characterization core's processing clock. The video clock must be less than or equal to the rate of the core clock. The video_clk is a required input in this mode.

  - **Sync Video Clock**: Select Sync Video Clock when the video stream has a clock that is synchronous to the Image Characterization core's processing clock.

- **Minimum Block Size**: Sets the minimum block size that the core can use. Valid choices are 4, 8, 16, 32, and 64. The smaller the block size, the more resources that are used.
- **Optional Features**
  - **AXI4-Lite Register Interface**: When selected, the core will be generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters.
  - **Enable INTC Port**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions.
  - **Chroma Input Filter**: When selected, the core will instantiate a 3-tap FIR filter to low pass filter the input Chroma data.
  - **Chroma Parity**: Select odd if the first line of video contains chroma information. Chroma parity is only used for YUV 4:2:0 data.
- **Frame Dimensions**:
  - **Number of Pixels per Scanline (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the lower half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance processes.
  - **Number of Scanlines per Frame (Default)**: When the AXI4-Lite control interface is enabled, the generated core will use the value specified in the CORE Generator GUI as the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance processes.
  - **Maximum Number of Pixels per Scanline**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for Active Pixels per Scan line, or the corresponding lower half-word of the ACTIVE_SIZE register must always be less than or equal to the value provided by Maximum Number of Pixels per Scanline. Using a tight upper-bound results in optimal block RAM usage.
- **Statistics Selection**: Selected items will appear in the Image Characterization's statistics data structure. Unselected items will be replaced with a zero in the statistics data structure. Logic associated with an item is not instantiated when the item is unselected. Resources can be conserved by deselecting unneeded items.
  - **Means/Variances and Color Selects**

- **Y**: When selected, global Y mean and variance values as well as block Y mean and variance values are included in the Image Characterization's statistics data structure.
- **Motion**: When selected, global Motion mean and variance values as well as block Motion mean and variance values are included in the Image Characterization's statistics data structure.
- **Frequency**: When selected, global Frequency mean and variance values as well as block Frequency mean and variance values are included in the Image Characterization's statistics data structure. The frequency values include Low Frequency and High Frequency.
- **Edge**: When selected, global Edge Content mean and variance values as well as block Edge Content mean and variance values are included in the Image Characterization's statistics data structure.
- **U**: When selected, global U mean and variance values as well as block U mean and variance values are included in the Image Characterization's statistics data structure.
- **V**: When selected, global V mean and variance values as well as block V mean and variance values are included in the Image Characterization's statistics data structure.
- **Saturation**: When selected, global Saturation mean and variance values as well as block Saturation mean and variance values are included in the Image Characterization's statistics data structure.
- **Color Selects**: Sets the number of Color Select values that will be included in the Image Characterization's statistics data structure. Valid choices are 0, 4 and 8.

◦ **Histograms**

- **Y**: When selected, the Y Histogram is included in the Image Characterization's statistics data structure.
- **U**: When selected, the U Histogram is included in the Image Characterization's statistics data structure.
- **V**: When selected, the V Histogram is included in the Image Characterization's statistics data structure.
- **Hue**: When selected, the Hue Histogram is included in the Image Characterization's statistics data structure.

Figure 8-2 through Figure 8-4. displays the GUI parameters that affect the default register values of the Image Characterization core. When the AXI4-Lite interface is present, these values set the default values of the corresponding registers in the AXI4-Lite interface. When the AXI4-Lite interface is not present, these values are set permanently in the core.

• **Buffer Addresses**:

- **Stats Start Addr0**: Specifies the Start Address of the first external memory buffer that will hold the Image Characterization data structure.
- **Stats Start Addr1**: Specifies the Start Address of the second external memory buffer that will hold the Image Characterization data structure.

- **Block Information**:
  - **Block Size**: Specifies the size of the block to be use during the Block Statistics calculations. Valid selections are 4, 8, 16, 32, or 64.
  - **Blocks Wide:** Specifies the number of blocks in the horizontal direction. Calculated as Number of Pixels per Scanline / Block Size.
  - **Blocks High**: Specifies the number of blocks in the vertical direction. Calculated as Number of Scanlines per Frame / Block Size.
- Mean and Variance Scale Factors
  - **Global Y Width Scaling**: Specifies the width scale factor for Global Luma and Chroma Means & Variances. Calculated as (1/Blocks Wide)*65536
  - **Global Y Height Scaling**: Specifies the height scale factor for Global Luma and Chroma Means & Variances. Calculated as (1/Blocks High)*65536
  - **Block Y Scaling**: Specifies the scale factor for Block Luma Means & Variances. Calculated as (1/Num_Block_Pixels)*65536
  - **Block C Scaling**: Specifies the scale factor for Block Chroma Means & Variances. Calculated as (2/Num_Block_Pixels)*65536 for YUV 4:2:2 or as (4/Num_Block_Pixels)*65536 for YUV 4:2:0.
- Gain Values
  - **High Frequency Gain**: Specifies the gain value for the High Frequency component. Valid selections are 1, 2, 4 or 8.
  - **Horizontal Edge Gain**: Specifies the gain value for the Horizontal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Vertical Edge Gain**: Specifies the gain value for the Vertical edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Right Diagonal Edge Gain:** Specifies the gain value for the Right Diagonal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
  - **Left Diagonal Edge Gain**: Specifies the gain value for the Left Diagonal edge portion of the Edge Content component. Valid selections are 0, 1, 2, 4, or 8.
- **Color Select Thresholds**
  - **Color Select 1 – 8**
    - **Saturation Max**: Specifies the maximum value for the Saturation thresholds.
    - **Saturation Min**: Specifies the minimum value for the Saturation thresholds.

- **Hue Max**: Specifies the maximum value for the Hue thresholds.
- **Hue Min**: Specifies the minimum value for the Hue thresholds.

EDK pCore GUI

When the Xilinx Image Characterization LogiCORE IP is generated from EDK software, it is generated with each option set to the default value. All customizations of an Image Characterization pCore are done with the EDK pCore GUI. Figure 8-5 illustrates the EDK pCore GUI for the Image Characterization pCore. All of the options in the EDK pCore GUI for the Image Characterization core correspond to the same options in the CORE Generator GUI.

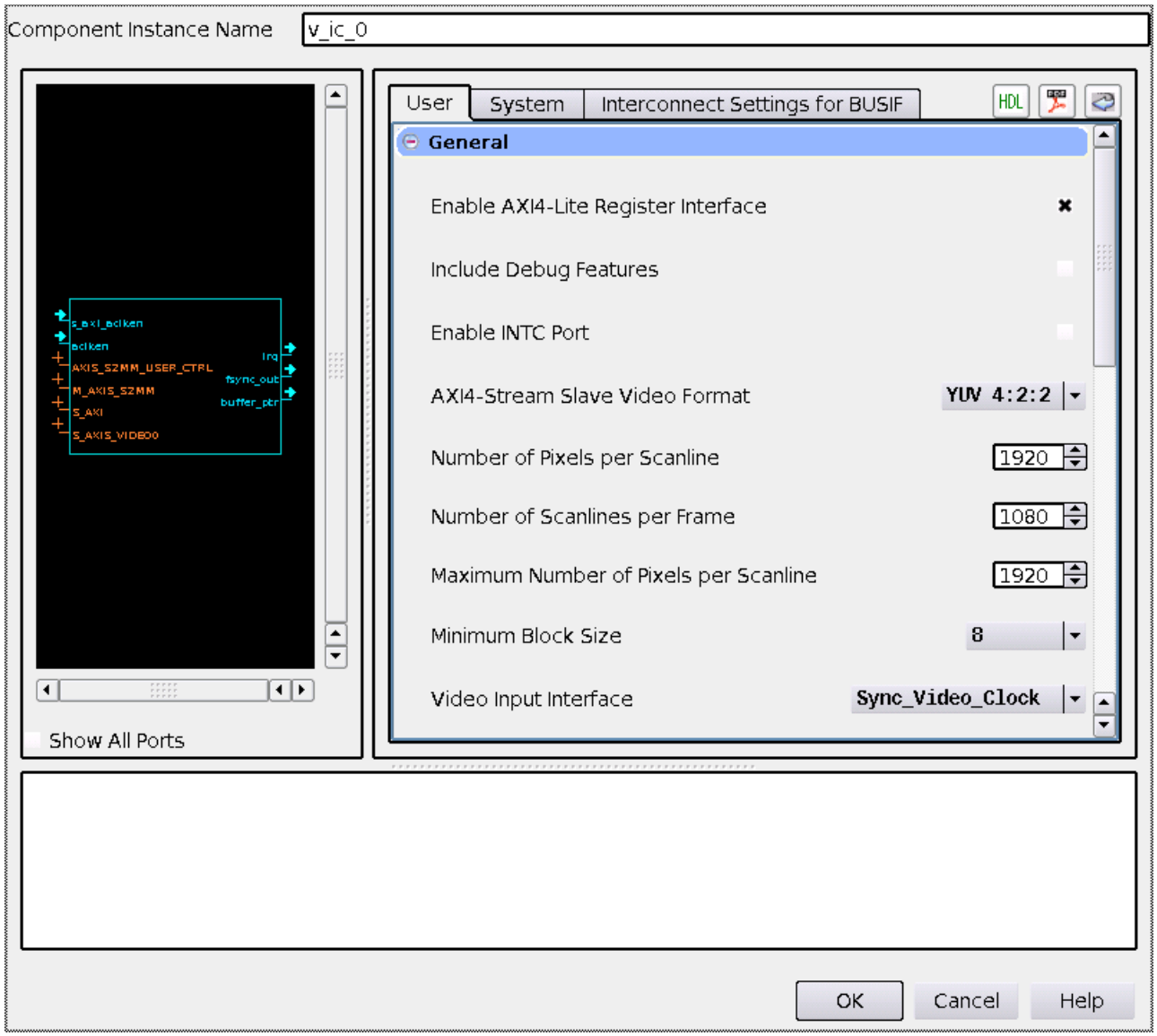

Figure 8-5: **Image Characterization pCore GUI**

Parameter Values in the XCO File

Table 8-1 defines valid entries for the Xilinx CORE Generator (XCO) parameters. Xilinx recommends that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator GUI to configure the core and perform range and parameter value checks. The XCO parameters help in define the interface in other design tools.

Table 8-1: **XCO Parameters**

XCO Parameter	Default	Valid Values
component_name	v_ic_v3_00_a_u0	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_ic_v2_0" is not allowed.
c_s_axis_video_format	"YUV_420"	"YUV_420", "YUV_422"
c_input_video_interface	"Sync_Video_Clock"	"Sync_Video_Clock", "Async_Video_Clock"
c_min_block_size	8	4, 8, 16, 32, 64
c_has_axi4_lite	false	false, true
c_has_intc_if	false	false, true
c_chroma_filter	false	false, true
c_chroma_parity	Odd	Odd, Even
c_active_cols	1920	32..7680
c_active_rows	1080	32..7680
c_max_cols	1920	32..7680
c_use_y_mean_var	true	false, true
c_use_mot_mean_var	true	false, true
c_use_edge_mean_var	true	false, true
c_use_freq_mean_var	true	false, true
c_use_u_mean_var	true	false
c_use_v_mean_var	true	false
c_use_sat_mean_var	true	false, true
c_num_color_selects	8	0, 4, 8
c_use_y_histogram	true	false, true
c_use_u_histogram	true	false, true
c_use_v_histogram	true	false, true
c_use_hue_histogram	true	false, true
c_stats_start_addr0	0	0x00000000..0xFFFFFFFF
c_stats_start_addr1	0	0x00000000..0xFFFFFFFF

Table 8-1: **XCO Parameters**

XCO Parameter	Default	Valid Values
c_block_size	8	4, 8, 16, 32, 64
c_num_blocks_wide	160	2..(c_active_cols/ c_block_size)
c_num_blocks_high	90	2..(c_active_rows/ c_block_size)
c_global_y_height_scaling	728	1..65535
c_global_y_width_scaling	410	1..65535
c_block_y_scaling	1024	1..65535
c_block_c_scaling	4096	1..65535
c_high_freq_gain	1	1, 2, 4, 8
c_h_gain	1	0, 1, 2, 4, 8
c_v_gain	1	0, 1, 2, 4, 8
c_l_gain	1	0, 1, 2, 4, 8
c_r_gain	1	0, 1, 2, 4, 8
c_cs1_sat_max	255	0..255
c_cs1_sat_min	0	0..255
c_cs1_hue_max	255	0..255
c_cs1_hue_min	0	0..255
c_cs2_sat_max	255	0..255
c_cs2_sat_min	0	0..255
c_cs2_hue_max	255	0..255
c_cs2_hue_min	0	0..255
c_cs3_sat_max	255	0..255
c_cs3_sat_min	0	0..255
c_cs3_hue_max	255	0..255
c_cs3_hue_min	0	0..255
c_cs4_sat_max	255	0..255
c_cs4_sat_min	0	0..255
c_cs4_hue_max	255	0..255
c_cs4_hue_min	0	0..255
c_cs5_sat_max	255	0..255
c_cs5_sat_min	0	0..255
c_cs5_hue_max	255	0..255
c_cs5_hue_min	0	0..255
c_cs6_sat_max	255	0..255

Table 8-1: **XCO Parameters**

XCO Parameter	Default	Valid Values
c_cs6_sat_min	0	0..255
c_cs6_hue_max	255	0..255
c_cs6_hue_min	0	0..255
c_cs7_sat_max	255	0..255
c_cs7_sat_min	0	0..255
c_cs7_hue_max	255	0..255
c_cs7_hue_min	0	0..255
c_cs8_sat_max	255	0..255
c_cs8_sat_min	0	0..255
c_cs8_hue_max	255	0..255
c_cs8_hue_min	0	0..255

Output Generation

This section contains a list of the files generated from CORE Generator.

File Details

The CORE Generator output consists of a subset of the files listed in Table 8-2:.

Table 8-2: **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	
<component_name>.vho	The HDL template for instantiating the core.
<component_name>.v	
<component_name>.vhd	The structural simulation model for the core. It is used for functionally simulating the core.
<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.
<component_name>_flist.txt	A text file listing all of the output files produced when the customize d core was generated in the CORE Generator software.

Table 8-2: **CORE Generator Output Files**

Name	Description
<component_name>.asy	IP symbol file
<component_name>.gise	
<component_name>.xise	ISE subproject files for use when including the core in ISE designs.

Constraining the Core

This chapter contains information about constraining the core in the ISE® Design Suite environment.

Required Constraints

When using "Sync Video Clock" mode, the ACLK pin should be constrained at the desired pixel clock rate for your video stream. When using "Async Video Clock" mode, the Video_Clk pins should be constrained at the desired pixel clock rate for your video stream and the ACLK pin should be constrained to at the desired clock rate for your core logic.

The S_AXI_ACLK pin should be constrained at the frequency of the AXI4-Lite subsystem.

In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

UCF

```
INST "*U_VIDEO_CTRL*/*SYNC2PROCCLK_I*/data_sync_reg[0]*" TNM =
"async_clock_conv_FFDEST";
TIMESPEC "TS_async_clock_conv" = FROM FFS TO "async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
INST "*U_VIDEO_CTRLk*/*SYNC2VIDCLK_I*/data_sync_reg[0]*" TNM =
"vid_async_clock_conv_FFDEST";
TIMESPEC "TS_vid_async_clock_conv" = FROM FFS TO "vid_async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. For a complete listing of supported devices, see the release notes for this core. For a complete listing of supported devices, see the release notes for this core.

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for this core. See Chapter 2, Maximum Frequency. 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 this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for this core.

Detailed Example Design

No example design is available at the time for the LogiCORE IP Image Characterization v3.00a core.

Demonstration Test Bench

A demonstration test bench is provided which enables core users to observe core behavior in a typical use scenario.

RECOMMENDED: *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 Image Characterization v3.00a core instance under test.

- axi4lite_mst

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

- axi4s_video_mst

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

- axi4s_video_slv

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

- ce_gen

  Programmable Clock Enable (ACLKEN) generator

Running the Simulation

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

Directory and File Contents

The directory structure underneath the top-level folder is:

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

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

SECTION IV: APPENDICES

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the Image Characterization core. Testing included the following:

- Register accesses
- Processing of multiple frames of data
- Testing of various frame sizes and block sizes
- Testing of various Video formats
- Varying instantiations of the Maximum Frame Size
- Varying instantiations of the Minimum Block Size
- Varying instantiations of the Video Input Interface
- Varying instantiations of the Statistics Selections

Hardware Testing

The Image Characterization core has been tested on 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-Interface and various other peripherals. The software for the test system included pre-generated input and output for the Image Characterization core. Various tests could be supported by simply varying the configuration of the Image Characterization core or by loading a different software executable. The MicroBlaze was responsible for:
  - Initializing the appropriate input and output buffers in external memory.
  - Initializing the Image Characterization core.

- Launching the test.
- Comparing the output of the Image Characterization core against the expected results.
- Reporting the Pass/Fail status of the test and any errors that were found.

Interoperability

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

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

Parameter Changes in the XCO File

All of the existing XCO parameters were renamed in v3_00_a version of the core. In addition, new XCO parameters were added as well. See Parameter Values in the XCO File in Chapter 8 for a full listing of the current XCO parameters.

Port Changes

The Image Characterization v2.0 changed all of its interfaces to use AXI4 based interfaces. For more information, see *AXI Reference Guide (UG762)*.

PLB to AXI4-Lite

The Image Characterization v2.0 changed from the PLB 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.

VFBC to AXI4

The Image Characterization v2.0 changed from the Video Frame Buffer Controller (VFBC) interface to the AXI4 interface. As a result, all of the VFBC-related connections have been replaced with an AXI4

AXI4 to AXI4-Stream

The Image Characterization v3.00.a changed from the AXI4 Memory Mapped interface that connected directly to the AXI Interconnect to a set of AXI4-Stream interfaces that connect to an AXI-DataMover core which then connects to the AXI Interconnect. As a result, the Image Characterization core now has 2 AXI4-Stream interfaces: `Data_output` and `Command_output`. The `Data_output` interface connects to the AXI-DataMover core's

S2MM Data interface and carries the output data of the Image Characterization core. The `Command_output` interface connects to the AXI_DataMover core's S2MM Command interface and carries the memory address and data block size information that the AXI-DataMover then uses to transfer the data across the AXI Interconnect into external memory.

Streaming Video input to AXI4-Stream

Image Characterization v2.0 changed from a streaming video input interface to the AXI4-Stream interface. As a result, all of the VFBC-related connections have been replaced with an AXI4 interface. Image Characterization v3.00.a changed the AXI4-Stream interface by adding an additional signal "tuser0". The following table details the changes to port naming and additional or deprecated ports from v1.1 to v2.0 to v3.00.a.

Version 1.1	Version 2.0	Version 3.00.a	Notes
core_clk	clk	aclk	Rename only
vblank_in	fsync_in	s_axis_tuser0	Rename only
	fsync_out	fsync_out	Unchanged
	buffer_ptr	buffer_ptr	Unchanged
ycm_in	s_axis_tdata	s_axis_tdata	Unchanged
active_video_in	s_axis_tvalid	s_axis_tvalid	Unchanged
	s_axis_tlast	s_axis_tlast	Unchanged
	s_axis_tready	s_axis_tready	Unchanged
chroma_in			Deprecated (See Chroma Formatting section)

Functionality Changes

The basic functionality of the Image Characterization core has not changed with the migration to AXI4 interfaces. The latency of the core will increase due to the use of FIFO on the s_axis_tdata channel. Latency may also be increased by differences in the operation of the AXI4 interconnect as compared to the MPMC/VFBC memory controller.

Special Considerations when Migrating to AXI

Migration to the AXI4 and AXI4-Lite interfaces should be of minimal concern. You not typically interact with those interfaces directly.

Migration to the AXI4-Stream input interface is likely the biggest issue to consider. The AXI4-Stream interface adds two new signals that the input source must handle. The first is the s_axis_tready signal that is output by the Image Characterization v2.0 core. This signal is used to apply back-pressure on a pixel basis and halt the flow of data into the core. The data source must be compatible with this mode of operation. The second new signal is the s_axis_tlast signal that is an input to the Image Characterization core. This signal is used to denote the last pixel of each line of video. For the v2.0 release of the core, the s_axis_tlast signal is only used to trigger tlast_early and tlast_late error conditions and interrupts. These error conditions are calculated by comparing the arrival of the s_axis_tlast against the expected arrival. See the Interrupts section for more detail. Typically, this input port is driven by an output on the Motion Adaptive Noise Reduction v3.0 core which provides the proper connectivity.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the Image Characterization core.

The following topics are included in this appendix:

Finding Help on Xilinx.com

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

Documentation

This product guide is the main document associated with the Image Characterization core. For the Video over AXI4-Stream specification, refer to User Guide 934, AXI4-Stream Video IP and System Design Guide [Ref 3]. These guides, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Release Notes

Known issues for all cores, including the Image Characterization core are described in the IP Release Notes Guide (XTP025).

Known Issues

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

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

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

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

Contacting Technical Support

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

To contact Xilinx Technical Support:

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

When opening a WebCase, include:

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

Debug Tools

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

Example Design

No example design is delivered with the Image Characterization core, however, you can generate a functional test bench for an instantiation of the video core. You can find more information in *Chapter 6, Example Design for the Vivado™ Design Suite*.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Vivado Lab Tools

Vivado Lab Tools inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed.

Reference Boards

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

- 7 series evaluation boards
  - KC705
  - KC724
  - ZC702

C-Model Reference

Please see *C Model Reference in Chapter 4* in this guide for tips and instructions for using the provided C-Model files to debug your design.

License Checkers

If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds, the IP may continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Hardware Debug

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

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- General Checks

General Checks

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

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing. The evaluation version of the core will time out after running for 8 hours at 75 MHz.

Interface Debug

AXI4-Lite Interfaces

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

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

Symptom	Solution
Readback from the Version Register 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.
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.

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.
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.

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

Symptom	Solution
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 this core.

Appendix D

Application Software Development

Device Drivers

pCore Driver Files

The Image Characterization pCore includes a software driver written in the C programming language that you can use to control the core. A high-level API is provided to hide the details of the Xilinx Image Characterization core, and application developers are encouraged to use it to access the device features. A low-level API is also provided in case developers prefer to access the devices directly through the system registers.

Table D-1 lists the files included with the Image Characterization pCore driver.

Table D-1: **Software Driver Files Provided with the Image Characterization pCore**

File name	Description
xic.c	Provides the API access to all features of the Image Characterization device driver.
xic.h	Provides the API access to all features of the Image Characterization device driver.
xic_g.c	Contains a template for a configuration table of Image Characterization devices.
xic_hw.h	Contains identifiers and register-level driver functions (or macros) that can be used to access the Image Characterization device.
xic_intr.c	Contains interrupt-related functions of the Image Characterization device driver.
xic_sint.c	Contains static initialization methods for the Image Characterization device driver.
example.c	Examples that demonstrate how to control the Image Characterization device.

pCore API Functions

This section describes the functions included in the pcore Driver files generated for the O Image Characterization pCore. The software API is provide to allow easy access to the registers of the pCore as defined in Table 6 in the Register Space section. To utilize the API functions provided, the following header files must be included in the user's C code:

```
#include "xparameters.h"

#include "xic.h"
```

The hardware settings of your system, including the base address of your Image Characterization core are defined in the xparameters.h file. The xic.h file provides the API access to all of the features of the Image Characterization device driver.

More detailed documentation of the API functions can be found by opening the file index.html in the pCore directory ic_v3_00_a/doc/html/api.

Functions in xic.c

- int XIC_CfgInitialize (XIC *InstancePtr, XIC_Config *CfgPtr, u32 EffectiveAddr)
  This function initializes an IC device.
- void XIC_SetImageStatAddr (XIC *InstancePtr, u32 Addr1, u32 Addr2,
  This function sets up statistics data output frame addresses of the statistics data for an IC device.
- void XIC_GetImageStatAddr (XIC *InstancePtr, u32 *Addr1Ptr, u32 *Addr2Ptr
  This function fetches statistics data output frame addresses of the statistics data used by an IC device.
- void XIC_SetFrameSize (XIC *InstancePtr, XIC_DimensionCfg *DimensionCfgPtr)
  This function sets up dimension related configuration information used by an IC device.
- void XIC_GetFrameSize (XIC *InstancePtr, XIC_DimensionCfg *DimensionCfgPtr)
  This function fetches dimension configuration currently used by an IC device.
- void XIC_SetEdgeGain (XIC *InstancePtr, u8 HoriGain, u8 VertGain, u8 LeftDiagGain, u8 RightDiagGain)
  This function sets up edge gain parameters to be used by an IC device.
- void XIC_GetEdgeGain (XIC *InstancePtr, u8 *HoriGainPtr, u8 *VertGainPtr, u8 *LeftDiagGainPtr, u8 *RightDiagGainPtr)
  This function fetches edge gain parameters currently used by an IC device.
- void XIC_SetColorSelect (XIC *InstancePtr, XIC_ColorSelect *ColorSelectCfgPtr)
  This function sets up color selection configuration to be used by an IC device.
- void XIC_GetColorSelect (XIC *InstancePtr, XIC_ColorSelect *ColorSelectCfgPtr)
  This function fetches color selection configuration currently used by an IC device.
- void XIC_GetVersion (XIC *InstancePtr, u16 *Major, u16 *Minor, u16 *Revision)
  This function returns the version of an IC device.

Functions in xic_sint.c

- XIC_Config * XIC_LookupConfig (u16 DeviceId)
  XIC_LookupConfig returns a reference to an XIC_Config structure based on the unique device id, DeviceId.

Functions in xic_intr.c

- void XIC_IntrHandler (void *InstancePtr)
  This function is the interrupt handler for the Image Characterization driver.
- int XIC_SetCallBack (XIC *InstancePtr, u32 HandlerType, void *CallBackFunc, void *CallBackRef)
  This routine installs an asynchronous callback function for the given HandlerType:.

Functions in example.c

- void SetupColorSelect(XIC_ColorSelect *ColorSelectCfgPtr)
  This function shows an example of setting all of the Color Select registers using default Min/Max values.
- int IcExample(u16 ICDeviceID)
  This function is an example of initializing the Image Characterization core.

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/support/documentation/sw_manuals/glossary.pdf.

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

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

References

These documents provide supplemental material useful with this product guide:

1. UG761, AXI Reference Guide
2. DS768, AXI Interconnect IP Data Sheet
3. UG934, AXI4-Stream Video IP and System Design Guide
4. Vivado Design Suite Migration Methodology Guide (UG911)
5. Vivado™ Design Suite user documentation

To search for Xilinx documentation, go to http://www.xilinx.com/support.

Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.
12/18/2012	2.0	Updated core version. Added Vivado section and Debugging appendix.

Notice of Disclaimer