LogiCORE IP Image Statistics v4.00.a

Product Guide

PG008 April 24, 2012

Table of Contents

LogiCORE IP Image Statistics v4.00.a

Introduction

The Xilinx LogiCORE™ IP Image Statistics core implements the computationally intensive metering functionality common in digital cameras, camcorders and imaging devices. This core generates a set of statistics for color histograms, mean and variance values, edge and frequency content for 16 user-defined zones on a per frame basis. The statistical information collected may be used in the control algorithms for Auto-Focus, Auto-White Balance, and Auto-Exposure for image processing applications.

Features

- Supports spatial resolutions from 32x32 up to 7680x7680
  - Supports 1080P60 in all supported device families
  - Supports 4kx2k @ 24 Hz in supported high performance devices
- AXI4-Stream data interfaces
- AXI4-Lite control interface
- 16 programmable zones
- 8, 10, or 12-bit input precision
- Outputs for all zones and color channels:
  - Minimum and maximum color values
  - Sum and sum of squares for each color value
  - Low and high frequency content
  - Horizontal, vertical and diagonal edge content
- Outputs for pre-selected zone(s):
  - Y channel histogram
  - R,G,B channel histograms
  - Two-dimensional Cr-Cb histogram

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family [(1)]	Zynq™-7000, Artix™-7, Virtex®-7, Kintex™-7, Virtex-6, Spartan®-6
Supported User Interfaces	AXI4-Lite, AXI4-Stream
Resources	See Table 2-1 through Table 2-4.
Provided with Core	
Design Files	NGC Netlist, Encrypted HDL
Example Design	Not Provided
Test Bench	Verilog [(2)]
Constraints File	Not Provided
Simulation Model	VHDL or Verilog Structural Model C Model [(2)]
Tested Design Tools	
Design Entry Tools	CORE Generator™, ISE® 14.1, Platform Studio (XPS)
Simulation [(3)]	Mentor Graphics ModelSim, ISim
Synthesis Tools	Xilinx Synthesis Technology (XST) 14.1
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C Model available on the Image Statistics product page.
3. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.

Overview

Most digital cameras implement a form of automatic white balance (AWB) and automatic exposure (AE) control. Cameras with an adjustable focus (AF) lens also implement automatic focus. All of these algorithms have the need for sophisticated image statistics gathered from the image or video to process and implement controls and algorithms for these functions. Generating image statistics is a data intensive process, and the Image Statistics core provides an efficient, programmable solution. The resulting image statistics provide the basic data for software based AE, AWB and AF algorithms. The Image Statistics core supports multi-zone metering on rectangular regions, and 16 software defined zones, as shown in Figure 1-1.

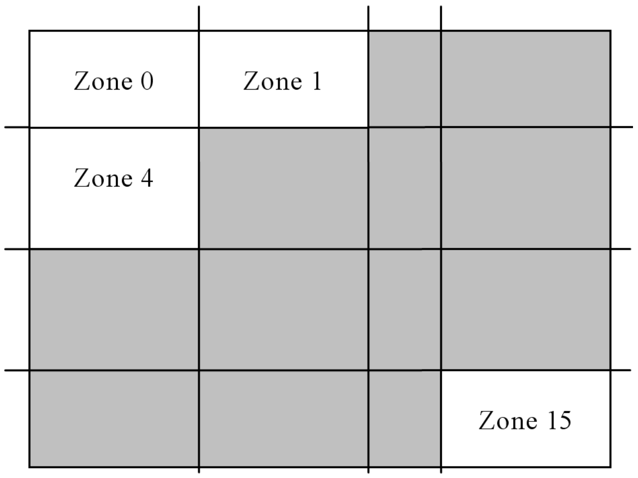


Figure 1-1: **16 Zone Metering**

Minimum, maximum, sum, and sum of squares values are calculated for all color channels for all 16 zones. Frequency and edge content is also calculated for all zones in luminance values provided by an RGB-to-YCrCb converter internal to the Image Statistics core. The RGB, Y, and two-dimensional Cr-Cb histograms are calculated over predefined sets of zones.

Feature Summary

The Image Statistics core provides data from video streams of a maximum resolution of 7680 columns by 7680 rows with 8, 10, or 12 bits per pixel and supports the bandwidth necessary for high-definition (1080p60) resolutions in all Xilinx FPGA device families. The core generates data from the video stream in the form of minimum and maximum color values, sum and sum of squares for each color value, low and high frequency content and horizontal, vertical and edge content from 16 programmable zones. The core can also generate histograms for RGB channels, Y channel and two dimensional Cr-Cb channels.

The core can be configured and instantiated from CORE Generator or EDK tools. Core functionality is controlled dynamically through the AXI4-Lite interface.

Applications

- Automatic Exposure (AE) control
- Automatic Sensor Gain (AG) control
- Auto Focus (AF) control of the lens assembly
- Digital contrast/brightness adjustment
- Global histogram equalization
- White Balance correction

Product Specification

This chapter details the performance, interfaces and registers of the Image Statistics core.

Standards Compliance

The Image Statistics core is compliant with the AXI4-Lite interconnect standard as defined in UG761, *AXI Reference Guide*.

Performance

The following sections detail the performance characteristics of the Image Statistics core.

Maximum Frequencies

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 2-4 for device-specific information.

Latency

The processing latency of the core is one frame before statistical information can be delivered. Because the video stream goes through the core unmodified, there is only one clock cycle latency between in the input video and the output video.

Throughput

When the core is processing data from a video source which can always provide valid data, e.g. a frame buffer, the throughput of the core is essentially the rate of the video.

In numeric terms, 1080P/60 represents an average data rate of 124.4 MPixels/second (1080 rows x 1920 columns x 60 frames / second), and a burst data rate of 148.5 MPixels/sec. In order to ensure that the core can process 124.4 MPixels/second, the core needs to operate at the pixel clock rate.

The core is limited to how quickly it can calculate the data from the video stream. Resource Utilization provides an estimate of how quickly the core can operate in certain conditions.

Resource Utilization

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

Information presented in Table 2-1 through Table 2-4 show guidelines to the resource utilization of the Image Statistics core for Zynq™-7000, Artix™-7, Virtex®-7, Kintex-7,Virtex-6, and Spartan®-6 FPGA families. The design was tested using Xilinx ISE v13.3 tools with area constraints (see table footnotes) and default tool options.

The information presented in Table 1-1 through Table 1-7 is a guide to the resource utilization and maximum clock frequency of this core for all input/output width combinations for Zynq-7000, Artix-7, Virtex-7, Kintex-7, Virtex-6, and Spartan-6 FPGA families. The Xtreme DSP Slice count is always 16, regardless of parameterization, and this core does not use any dedicated I/O or CLK resources. The design was tested using ISE® v14.1 tools with default tool options for characterization data. The design was tested with the AXI4-Lite interface and the debug features disabled. By default, the maximum number of pixels per scan line was set to 1920, active pixels per scan line was set to 1920.

Table 2-1: **Resource Utilization and Target Speed for Virtex-7 Devices**

Data Width	Max Rows Max Cols	Histogram			LUTs	FFs	RAM36	RAM18	DSP48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	1023	Yes	Yes	Yes	3109	3495	1	5	16	235
8	2200	Yes	Yes	Yes	3066	3495	1	5	16	221
10	1023	Yes	Yes	Yes	3326	3856	6	1	16	234
10	2200	Yes	Yes	Yes	3326	3856	6	1	16	234
12	1023	Yes	Yes	Yes	3568	4217	16	1	16	208
12	2200	Yes	Yes	Yes	3568	4217	16	1	16	208

Device and speed file: XC7V330T-1 (ADVANCED 1.03A 12-03-05)

Table 2-2: **Resource Utilization and Target Speed for Kintex-7 Devices**

Data Width	Max Rows Max Cols	Histogram			LUTs	FFs	RAM36	RAM18	DSP48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	1023	Yes	Yes	Yes	2975	3495	1	5	16	233
8	2200	Yes	Yes	Yes	2975	3495	1	5	16	233
10	1023	Yes	Yes	Yes	3280	3856	6	0	16	223
10	2200	Yes	Yes	Yes	3052	3856	6	1	16	223
12	1023	Yes	Yes	Yes	3557	4217	16	1	16	222
12	2200	Yes	Yes	Yes	3557	4217	16	1	16	222

Device and speed file: XC7K70T-1 (ADVANCED 1.04C 12-03-05)

Table 2-3: **Resource Utilization and Target Speed for Virtex-6 Devices**

Data Width	Max Rows Max Cols	Histogram			LUTs	FFs	RAM36	RAM18	DSP48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	1023	Yes	Yes	Yes	3045	3495	1	5	16	187
8	2200	Yes	Yes	Yes	2976	3495	1	5	16	180
10	1023	Yes	Yes	Yes	3244	3856	6	1	16	196
10	2200	Yes	Yes	Yes	3185	3856	6	1	16	180
12	1023	Yes	Yes	Yes	3157	4217	16	1	16	197
12	2200	Yes	Yes	Yes	3493	4217	16	1	16	169

Device and speed file: XC6VLX75T,-1 (PRODUCTION 1.17 12-03-05)

Table 2-4: **Resource Utilization and Target Speed for Spartan-6 Devices**

Data Width	Max Rows Max Cols	Histogram			LUTs	FFs	RAM16	RAM8	DSP48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	1023	Yes	Yes	Yes	3118	3517	2	5	16	142
8	2200	Yes	Yes	Yes	3118	3517	2	5	16	142
10	1023	Yes	Yes	Yes	3320	4002	7	6	16	147
10	2200	Yes	Yes	Yes	3320	4002	7	6	16	147
12	1023	Yes	Yes	Yes	3574	4367	33	0	16	137
12	2200	Yes	Yes	Yes	3574	4367	33	0	16	137

Device and speed file: XC6SLX25-2 (PRODUCTION 1.21 12-03-05)

Optional Histogram Calculation

Table 2-5 through Table 2-8 present resource utilization numbers for all possible combinations of optional histogram settings, for all supported device families, using 8-bit input data and the maximum numbers of rows and columns set to 2200.

Table 2-5: **Optional Histogram Calculation: Resource Utilization Virtex-7 Devices (XC7V330T-1)**

Data Width	MAX Rows MAX Cols	Histogram			LUTs	FF	RAM36	RAM18	DSP 48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	2200	Yes	Yes	Yes	2828	3777	1	7	16	202
8	2200	Yes	Yes	No	2787	3681	1	6	16	202
8	2200	Yes	No	Yes	2677	3591	1	6	14	202
8	2200	Yes	No	No	2582	3474	1	5	14	202
8	2200	No	Yes	Yes	2617	3444	1	4	16	202
8	2200	No	Yes	No	2498	3348	1	3	16	202
8	2200	No	No	Yes	2402	3258	1	3	14	202
8	2200	No	No	No	2372	3137	1	2	14	202

Table 2-6: **Optional Histogram Calculation: Resource Utilization Kintex-7 Devices (XC7K70T-1)**

Data Width	MAX Rows MAX Cols	Histogram			LUTs	FF	RAM36	RAM18	DSP 48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	2200	Yes	Yes	Yes	2791	3777	1	7	16	202
8	2200	Yes	Yes	No	2709	3681	1	6	16	202
8	2200	Yes	No	Yes	2562	3591	1	6	14	202
8	2200	Yes	No	No	2537	3474	1	5	14	202
8	2200	No	Yes	Yes	2594	3444	1	4	16	202
8	2200	No	Yes	No	2467	3348	1	3	16	202
8	2200	No	No	Yes	2384	3258	1	3	14	202
8	2200	No	No	No	2312	3137	1	2	14	202

Table 2-7: **Optional Histogram Calculation: Resource Utilization Virtex-6 Devices (XC6VLX75T-1)**

Data Width	MAX Rows MAX Cols	Histogram			LUTs	FF	RAM36	RAM18	DSP 48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	2200	Yes	Yes	Yes	2821	3777	1	7	16	202
8	2200	Yes	Yes	No	2741	3681	1	6	16	202
8	2200	Yes	No	Yes	2661	3591	1	6	14	202
8	2200	Yes	No	No	2479	3474	1	5	14	202
8	2200	No	Yes	Yes	2561	3444	1	4	16	202
8	2200	No	Yes	No	2542	3348	1	3	16	202
8	2200	No	No	Yes	2464	3258	1	3	14	202
8	2200	No	No	No	2309	3137	1	2	14	202

Table 2-8: **Optional Histogram Calculation: Resource Utilization Spartan-6 Devices (XC6SLX25-1)**

Data Width	MAX Rows MAX Cols	Histogram			LUTs	FF	RAM36	RAM18	DSP 48s	Clock F_{MAX} (MHz)
		Y	RGB	CC						
8	2200	Yes	Yes	Yes	4557	3811	4	5	16	158
8	2200	Yes	Yes	No	4433	3711	4	4	16	159
8	2200	Yes	No	Yes	4279	3621	4	4	14	158
8	2200	Yes	No	No	4096	3505	4	3	14	158
8	2200	No	Yes	Yes	4108	3472	4	2	16	130
8	2200	No	Yes	No	3882	3377	4	1	16	157
8	2200	No	No	Yes	2440	3277	4	1	14	160
8	2200	No	No	No	3542	3161	4	0	14	157

Port Descriptions

The Image Statistics 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 Image Statistics core.

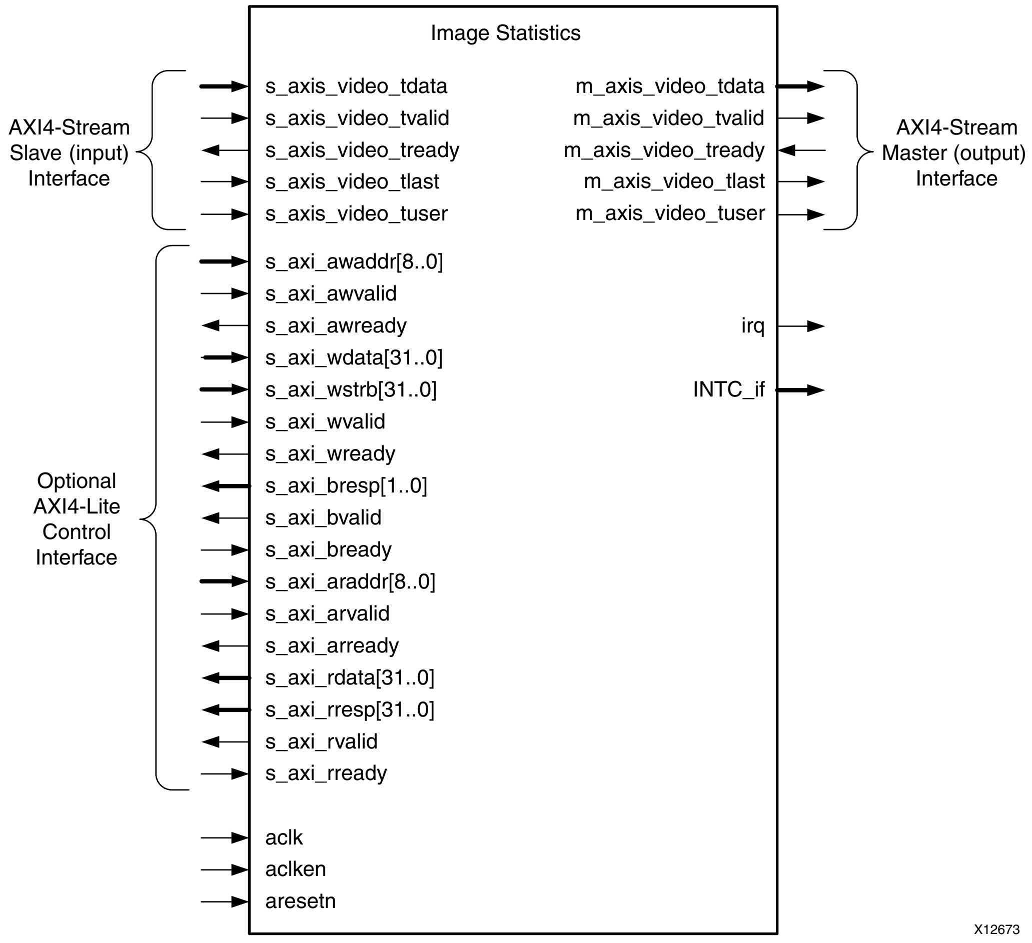


Figure 2-1: **Image Statistics Core Top-Level Signaling Interface**

Common Interface Signals

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

Table 2-9: **Common Interface Signals**

Signal Name	Direction	Width	Description
ACLK	In	1	Input Video Data
ACLKEN	In	1	Input Video Valid Signal
ARESETn	In	1	Input Ready
IRQ	Out	1	Optional Interrupt Request Pin. Available only when AXI4-Liter interface is selected on GUI.

The ACLK, ACLKEN and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and the AXI4-Lite control interface. Refer to The Interrupt Subsystem for a description of the INTC_IF and IRQ pins.

ACLK

All signals, including the AXI4-Stream and AXI4-Lite component interfaces, must be synchronous to the core clock signal ACLK. All interface input signals are sampled on the rising edge of ACLK. All output signal changes occur after the rising edge of ACLK.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to both the AXI4-Stream and AXI4-Lite 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.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to both the AXI4-Stream and AXI4-Lite 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.

Data Interface

The CFA core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the UG761, *Xilinx AXI Reference Guide*, Video IP: "AXI Feature Adoption."

AXI4-Stream Signal Names and Descriptions

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

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

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	24, 32, 40	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal
s_axis_video_tready	Out	1	Input Ready
s_axis_video_tuser	In	1	Input Video Start Of Frame

Table 2-10: **AXI4-Stream Data Interface Signal Descriptions *(Cont'd)***

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

Video Data

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

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

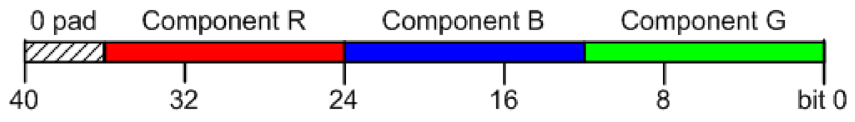


Figure 2-2: **RGB Data Encoding on s_axis_video_tdata and m_axis_video_tdata**

READY/VALID Handshake

A valid transfer occurs whenever READY, `VALID`, `ACLKEN`, and `ARESETn` are High at the rising edge of `ACLK`, as shown in Figure 2-3. During valid transfers, DATA only carries active video data. Blank periods and ancillary data packets are not transferred through the AXI4-Stream video protocol.

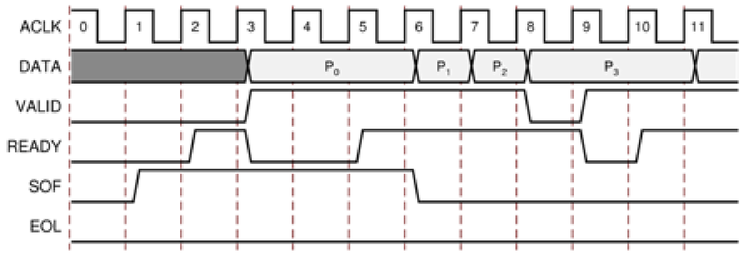


Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

Guidelines on Driving s_axis_video_tvalid

Once `s_axis_video_tvalid` is asserted, no interface signals (except the Image Statistics core driving `s_axis_video_tready`) can change value until the transaction completes (`s_axis_video_tready`, `s_axis_video_tvalid` and `ACLKEN` are High on the rising edge of `ACLK`). Once asserted, `s_axis_video_tvalid` can only be de-asserted after a transaction has completed. Transactions cannot be retracted or aborted. Following a transaction, `s_axis_video_tvalid` can either be de-asserted or remain asserted to initiate a new transfer.

Guidelines on Driving m_axis_video_tready

The `m_axis_video_tready` signal may be asserted before, during or after the cycle in which the Image Statistics core asserted `m_axis_video_tvalid`. The assertion of `m_axis_video_tready` may be dependent on the value of `m_axis_video_tvalid`. A slave that can immediately accept data qualified by `m_axis_video_tvalid`, should pre-assert its `m_axis_video_tready` signal until data is received. Alternatively, `m_axis_video_tready` can be registered and driven the cycle following `VALID` assertion. It is recommended that the AXI4-Stream slave should drive READY independently, or pre-assert READY to minimize latency.

Start of Frame Signals - m_axis_video_tuser0, s_axis_video_tuser0

The Start-Of-Frame (SOF) signal, physically transmitted over the AXI4-Stream `TUSER0` signal, marks the first pixel of a video frame. The SOF pulse is one valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-3. SOF serves as a frame synchronization signal, which allows downstream cores to re-initialize and detect the first pixel of a frame. The SOF signal may be asserted an arbitrary number of `ACLK` cycles before the first pixel value is presented on `DATA`, as long as a `VALID` is not asserted.

End of Line Signals - m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line signal, physically transmitted over the AXI4-Stream `TLAST` signal, marks the last pixel of a line. The EOL pulse is one valid transaction wide, and must coincide with the last pixel of a scan-line, as shown in Figure 2-4.

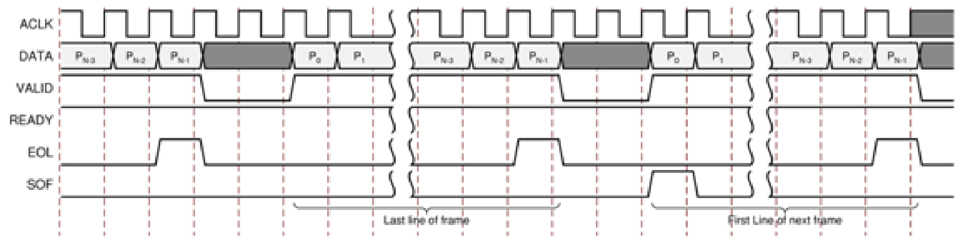


Figure 2-4: **Use of EOL and SOF Signals**

Control Interface

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

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

Table 2-11: **AXI4-Lite Interface Signals**

Signal Name	Direction	Width	Description
s_axi_lite_awvalid	In	1	AXI4-Lite Write Address Channel Write Address Valid.
s_axi_lite_awread	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
s_axi_lite_awaddr	In	32	AXI4-Lite Write Address Bus
s_axi_lite_wvalid	In	1	AXI4-Lite Write Data Channel Write Data Valid.
s_axi_lite_wready	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
s_axi_lite_wdata	In	32	AXI4-Lite Write Data Bus
s_axi_lite_bresp	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
s_axi_lite_bvalid	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
s_axi_lite_bready	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
s_axi_lite_arvalid	In	1	AXI4-Lite Read Address Channel Read Address Valid
s_axi_lite_arready	Out	1	Ready. Indicates DMA is ready to accept the read address.

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

Signal Name	Direction	Width	Description
s_axi_lite_araddr	In	32	AXI4-Lite Read Address Bus
s_axi_lite_rvalid	Out	1	AXI4-Lite Read Data Channel Read Data Valid
s_axi_lite_rready	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
s_axi_lite_rdata	Out	32	AXI4-Lite Read Data Bus
s_axi_lite_rresp	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core-specific registers. The Image Statistics core uses only one timing related register, `ACTIVE_SIZE` (0x0020), which allows specifying the input frame dimensions. By setting the core's control register, the method for gathering the statistical data is set. The generated data is read through the registers.

Table 2-12: **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 5: TEST_PATTERN* Bit 6: READOUT 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 4: DONE (Frame acquisition complete) Bit 16: SLAVE_ERROR
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to STATUS bits

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

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0010	VERSION	R	N/A	0x0400a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0014	SYSDEBUG0	R	N/A	0	31-0: Frame Throughput monitor*
0x0018	SYSDEBUG1	R	N/A	0	31-0: Line Throughput monitor*
0x001C	SYSDEBUG2	R	N/A	0	31-0: Pixel Throughput monitor*
0x0020	ACTIVE_SIZE	R/W	Yes	Specified via GUI	12-0: Number of Active Pixels per Scan line 28-16: Number of Active Lines per Frame
0x0100	HMAX0	R/W	Yes	¼ ACTIVE_COLS	Position of the first vertical zone delimiter
0x0104	HAMX1	R/W	Yes	½ ACTIVE_COLS	Position of the second vertical zone delimiter
0x0108	HAMX2	R/W	Yes	¾ ACTIVE_COLS	Position of the third vertical zone delimiter
0x010C	VAMX0	R/W	Yes	¼ ACTIVE_ROWS	Position of the first horizontal zone delimiter
0x0110	VAMX1	R/W	Yes	½ ACTIVE_ROWS	Position of the second horizontal zone delimiter
0x0114	VAMX2	R/W	Yes	¾ ACTIVE_ROWS	Position of the third horizontal zone delimiter
0x0118	HIST_ZOOM_FACTOR	R/W	Yes	0	Bit 0 and 1 control CC histogram zooming: 00: No zoom, full Cb and Cr range 01: Zoom by 2 10: Zoom by 4 11: Zoom by 8
0x011C	RGB_HIST_ZONE_EN	R/W	Yes	0xFFFF	Bits 0-15 correspond to zones 0-15, enabling RGB histogramming for the selected zones
0x0120	YCC_HIST_ZONE_EN	R/W	Yes	0xFFFF	Bits 0-15 correspond to zones 0-15, enabling Y and CC histogramming for the selected zones
0x0124	ZONE_ADDR	R/W	Yes	0	Bits 0-3 select a zone for readout

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

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0128	COLOR_ADDR	R/W	Yes	0	Bits 0-1 select a color channel for readout 00: Red 01: Green 1X: Blue
0x012C	HIST_ADDR	R/W	Yes	0	Bits 0-[DATA_WIDTH-1] address histograms.
0x0130	ADDR_VALID	R/W	Yes	0	Bit 0 qualifies ZONE_ADDR, COLOR_ADDR and HIST_ADDR
0x0134	MAX	R	No	0	Maximum value measured for the currently selected zone and color channel
0x0138	MIN	R	No	0	Minimum value measured for the currently selected zone and color channel
0x013C	SUM_LO	R	No	0	Lower 32 bits of the sum of values for the currently selected zone and color channel
0x0140	SUM_HI	R	No	0	Upper 32 bits of the sum of values for the currently selected zone and color channel
0x0144	POW_LO	R	No	0	Lower 32 bits of the summed of square values for the currently selected zone and color channel
0x0148	POW_HI	R	No	0	Upper 32 bits of the summed of square values for the currently selected zone and color channel
0x014C	HSOBEL_LO	R	No	0	Lower 32 bits of the sum of absolute values for the horizontal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x0150	HSOBEL_HI	R	No	0	Upper 32 bits of the sum of absolute values for the horizontal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x0154	VSOBEL_LO	R	No	0	Lower 32 bits of the sum of absolute values for the vertical Sobel Filter output, applied to luminance values of the currently selected zone and color channel

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

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0158	VSOBEL_HI	R	No	0	Upper 32 bits of the sum of absolute values for the vertical Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x015C	LSOBEL_LO	R	No	0	Lower 32 bits of the sum of absolute values for the diagonal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x0160	LSOBEL_HI	R	No	0	Upper 32 bits of the sum of absolute values for the diagonal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x0164	RSOBEL_LO	R	No	0	Lower 32 bits of the sum of absolute values for the anti-diagonal diagonal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x0168	RSOBEL_HI	R	No	0	Upper 32 bits of the sum of absolute values for the anti-diagonal Sobel Filter output, applied to luminance values of the currently selected zone and color channel
0x016C	HIFREQ_LO	R	No	0	Lower 32 bits of the sum of absolute values of the high frequency filter output, applied to luminance values of the currently selected zone
0x0170	HIFREQ_HI	R	No	0	Upper 32 bits of the sum of absolute values of the high frequency filter output, applied to luminance values of the currently selected zone
0x0174	LOFREQ_LO	R	No	0	Lower 32 bits of the sum of absolute values of the low frequency filter output, applied to luminance values of the currently selected zone
0x0178	LOFREQ_HI	R	No	0	Upper 32 bits of the sum of absolute values of the high frequency filter output, applied to luminance values of the currently selected zone
0x017C	RHIST	R	No	0	Red histogram values calculated over the zones selected by RGB_HIST_ZONE_EN

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

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0180	GHIST	R	No	0	Green histogram values calculated over the zones selected by RGB_HIST_ZONE_EN
0x0184	BHIST	R	No	0	Blue histogram values calculated over the zones selected by RGB_HIST_ZONE_EN
0x0188	YHIST	R	No	0	Luminance histogram values calculated over the zones selected by YCC_HIST_ZONE_EN
0x018C	CCHIST	R	No	0	Two-dimensional Cr-Cb chrominance histogram values calculated over the zones selected by YCC_HIST_ZONE_EN
0x0190	DATA_VALID	R	No	0	Bit 0 qualifies valid data in core registers corresponding to the address inputs

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.
- 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. For the double buffered registers in the Image Statistics core, 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 enabling consistency in data gathering between frames.
- Bit 2 of the CONTROL register, CLR_STATUS resets values of the STATUS register to 0, thereby clearing any interrupt requests (irq pin) as well.
- Bit 3 of the CONTROL register, CLR_ERROR resets values of the ERROR register to 0.
- Bit 5 of the CONTROL register, TEST_PATTERN, switches the core to test-pattern generator mode if debug features are enabled. See Appendix C, Debugging for more information. If debug features were not included at instantiation, this flag has no effect

on the operation of the core. Switching test-pattern generator mode on or off is not synchronized to frame processing, therefore can lead to image tearing.

- Bit 6 of the CONTROL register, READOUT, directs the Image Statistics core to bypass register readout or to exit register readout when set to 0. When set to 1, READOUT directs the core to enter register readout mode thereby producing the statistical data that had been calculated by the core.
- 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 progress will cause image tearing. For applications where the software reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 resets the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted.

  After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

STATUS (0x0004) Register

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

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

- Bit 0 of the STATUS register, PROC_STARTED, indicates that processing of a frame has commenced 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. In order 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 through the AXI4-Stream slave port. The number of pixels received between the last EOL signal surpassed the value programmed into the ACTIVE_SIZE register.
- Bit 2 of the ERROR register, SOF_EARLY, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the latest and the preceding Start-Of-Frame (SOF) signal was less than the value programmed into the ACTIVE_SIZE register.
- Bit 3 of the ERROR register, SOF_LATE, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the last SOF signal surpassed the value programmed into the ACTIVE_SIZE register.

IRQ_ENABLE (0x000C) Register

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

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 use this read-only value to verify that the software is matched to the correct version of the hardware.

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. See 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. See 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. See 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 lines per frame. Supported values are 32 to 7680. To avoid processing errors, restrict values written to ACTIVE_SIZE to the range supported by the core instance.

Setting Up Zone Boundaries

The zone boundaries for the 16 zones can be set up by programming the positions of three vertical and three horizontal delimiters as shown in Figure 2-5. Complemented by the constraints that the top-left corner of Zone 0 is flush with the left-top corner of the active image, and the bottom-right corner of Zone 15 is flush with the bottom-right corner of the active image, these values uniquely define the corners of all zones.

Data is collected during the active (and non-blank) period of the frame, and all zones traversed by the current scan-line are updated with the input data in parallel. Zone boundaries should be set up before acquiring the first frame of data by programming the HMAX and VMAX registers.

NOTE: **The minimum horizontal and vertical size of each zone must be at least 2**, along with the following geometric constraints:

- 0 < hmax0 < hmax1 < hmax2 < *MAX_COLS*
- 0 < vmax0 < vmax1 < vmax2 < *MAX_ROWS*

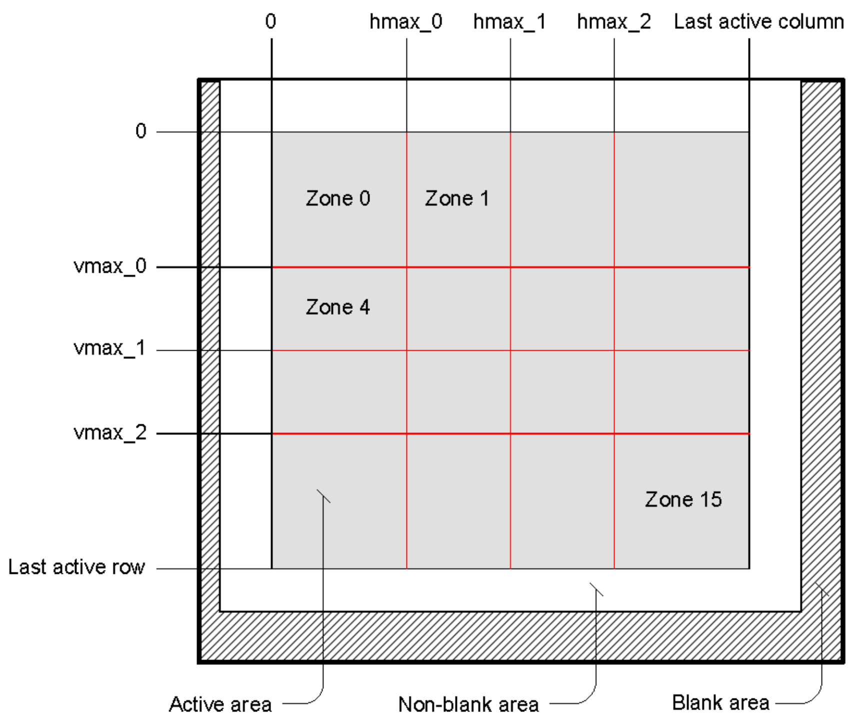


Figure 2-5: **Setting Up Zone Boundaries**

HMAX0 – HMAX2 (0x0100, 0x0104, 0x0108) Registers

Horizontal maximum registers involved in setting up the 16 zones used for gathering histogram data.

VMAX0 – VMAX2 (0x010C, 0x0110, 0x0114) Registers

Vertical maximum registers involved in setting up the 16 different zones used for gathering histogram data.

Histogram Data

For zones selected by a dynamically programmable register (`RGB_HIST_ZONE_EN`), the Image Statistics core bins R,G,B data and creates histograms as shown in Figure 2-6a. Similarly for zones selected by register `YCC_HIST_ZONE_EN`, Y and two-dimensional Cr-Cb histograms are calculated as shown in Figure 2-6c.

The two-dimensional Cr-Cb histogram (Figure 2-6c) contains information about the color content of a frame. Different hues have distinct locations in the Cr-Cb color-space (Figure 2-6b). The center location and variance of the color gamut can be derived from its two-dimensional Cr-Cb histogram. The bounding shape of the color gamut, along with the center location and variance of the two-dimensional Cr-Cb histogram, can be used to drive higher level algorithms [Ref 5] for white-balance correction.

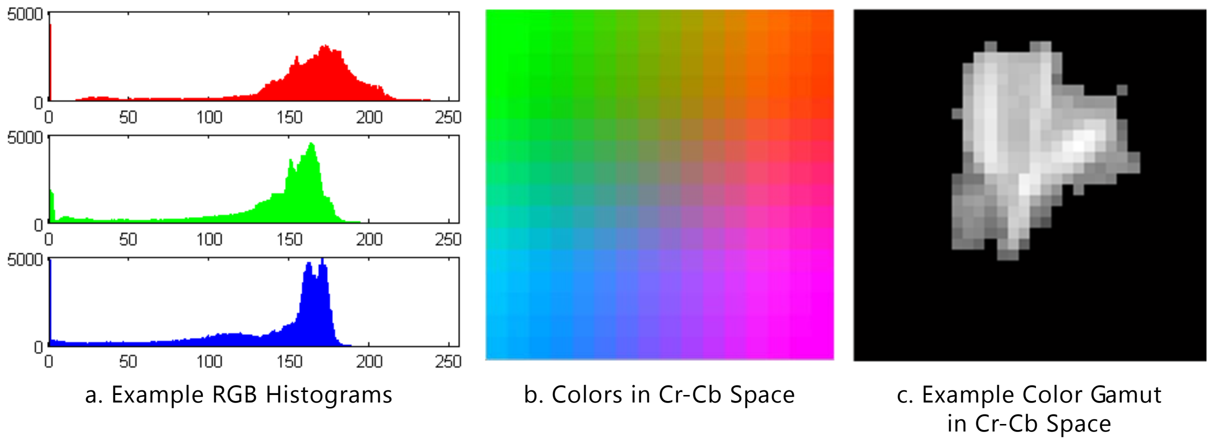


Figure 2-6: **Histogram Data**

For further details on histogram calculations, refer to Setting Up Histogram Calculations, page 28

The resolution of the R,G,B and Y histograms is the same as the resolution of the input data. The two-dimensional Cr-Cb histogram contains the same number of bins, but due to the two-dimensional configuration, the resolution is $2^{DATA\_WIDTH/2}$ along the Cr-Cb axes.

Setting Up Histogram Calculations

Histogram data is calculated and stored in block RAMs; hence calculating RGB and YCrCb histograms for all zones independently significantly increases the amount of FPGA resources required. Employing a mask to select which zones are involved in histogram calculation covers most typical applications:

- Calculating histogram values for one particular zone
- Calculating histogram values over an area of the image (such as the central zones, or zones in the corners)
- Calculating histogram values over the whole image

Figure 2-7 demonstrates how zones for RGB (red squares) and YCrCb (yellow circles) histogram calculations can be selected. For the example shown in Figure 2-7, zones 3, 4, 6, 7, 8 and 14 are selected for the Y and CrCb histograms. Correspondingly, bits 3,4,6,7,8 and 14 are set in `YCC_HIST_ZONE_EN`, resulting in a value of 0x000041D8.

Similarly, for the R,G, and B histograms, zones 1, 2, 3, 7, 8, 10, 11 and 14 are selected. Correspondingly bits 1, 2, 3, 7, 8, 10, 11 and 14 are set in `rgb_hist_zone_en`, resulting in a value of 0x00004D8E.

For the two-dimensional Cr-Cb histogram, there is another control, STATS_REG10_HIST_ZOOM_FACTOR, that helps tailor the Cr-Cb histogram calculation to the higher-level algorithm that consumes the 2D histogram results.

Consequently, Cr and Cb values have the same dynamic range as the input data. Cr and Cb are represented internally on DATA_WIDTH bits. A full precision Cr-Cb histogram would constitute a sparse 4k x 4k table that may be too large to implement within an FPGA. Therefore Cr and Cb are quantized to DATA_WIDTH/2 bits for histogramming. This quantization process inevitably involves loss of information. The use of the zoom factor (STATS_REG10_HIST_ZOOM_FACTOR) enables focusing on certain aspects of the histogram to minimize the effects of the information loss.

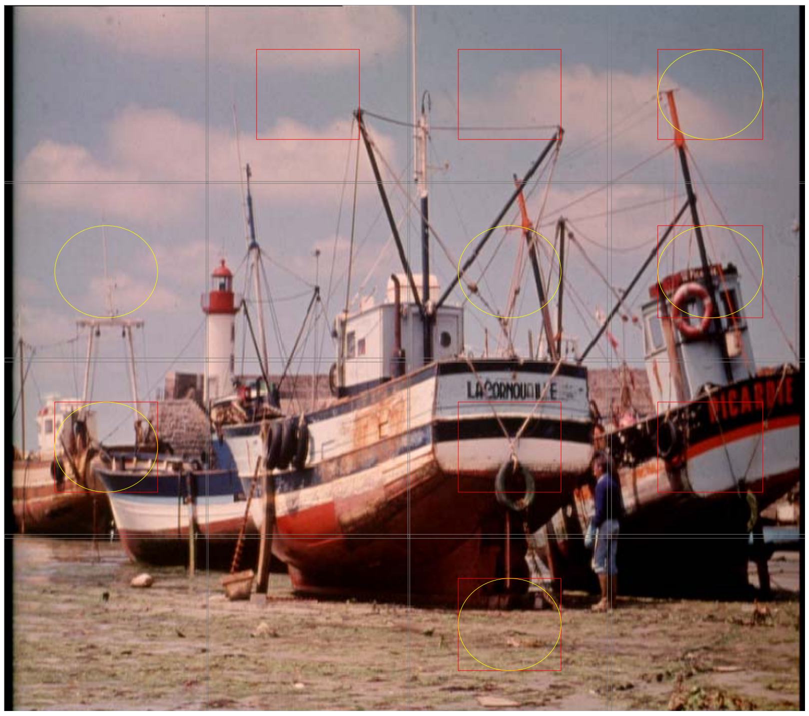


Figure 2-7: **Selecting Individual Zones for RGB and YCrCb Histograms**

Some higher level algorithms, such as gamut stretching [Ref 5], are concerned with the overall histogram, while other methods are concerned only with the central section, the area around the neutral point, to identify color casts. To support either type of algorithm, the histogram zoom factor allows the user to trade off resolution with range. The histogram zoom factor controls which bits of Cr and Cb values are selected for histogram binning. By setting the HIST_ZOOM_FACTOR to 0, the whole Cr-Cb histogram is represented at the output, as Cr and Cb values are simply quantized to DATA_WIDTH/2 bits. For example if DATA_WIDTH = 8, this quantization results in only the most significant four bits, bits 4, 5, 6 and 7, being used for histogram binning (see Table 2-13).

Table 2-13: **Histogram Zoom Bit Selections for DATA_WIDTH = 8 Bit Input Data**

Histogram Zoom Factor	Bits Used for Binning							
0	▨ 7	▨ 6	▨ 5	▨ 4	3	2	1	0
1	▨ 7	▨ 6	▨ 5	▨ 4	▨ 3	2	1	0
2	▨ 7	▨ 6	▨ 5	▨ 4	▨ 3	▨ 2	1	0
3	▨ 7	▨ 6	▨ 5	▨ 4	▨ 3	▨ 2	▨ 1	0

When `HIST_ZOOM_FACTOR` is set to a value other than 0, the resulting two-dimensional histogram represents only the central portion of the Cr-Cb histogram; pixels with extreme Cr-Cb values may fall outside the range represented by `DATA_WIDTH`/2 bits.

To enable further reduction of core footprint, RGB, Y, and Cr-Cb histograms can be individually enabled/disabled during generation time via the CORE Generator graphical user interface. If a particular type of histogram is not needed by the higher level algorithms, the core footprint can be reduced by 1,2, or 4 block RAMs depending on the input data resolution (`DATA_WIDTH`) and the target family.

HIST_ZOOM_FACTOR (0x0118) Register

Histogram zoom factor enables you to zoom into the center of the CrCb color space histogram (see Figures Figure 2-6b and Figure 2-6c) for greater resolution where the three color components converge.

RHIST (0x017C) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

GHIST (0x0180) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

BHIST (0x0184) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

YHIST (0x0188) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

CCHIST (0x018C) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

Addressing

Due to the large number of statistical data collected by the core, presenting all data simultaneously on core outputs is not feasible. Registers ZONE_ADDR and COLOR_ADDR facilitate reading out the max, min, sum and power result for specific zones and color channels.

The register HIST_ADDR facilitates addressing of histogram values.

The Image Statistics core provides a simple handshaking interface for reading out data. After setting the address registers as needed, setting bit 0 of the ADDR_VALID register signals to the core that valid addresses are present. In turn, the core fetches data corresponding to the addresses and marks valid data on the core outputs by writing a '1' to bit 0 of the DATA_VALID register (Figure 2-8).

All maximum, minimum, sum, sum of squares, Sobel and frequency contents can be read out by accessing zones 0-15 and color channels (coded 0,1,2) sequentially. If the host processor interface and the Image Statistics core are in the same CLK domain, addressing can be simplified, such that multiple addresses are supplied during the active portion of ADDR_VALID. When a sequence of valid addresses is presented to the core, the sequence of corresponding valid data becomes available with a latency of five CLK cycles (Figure 2-9).

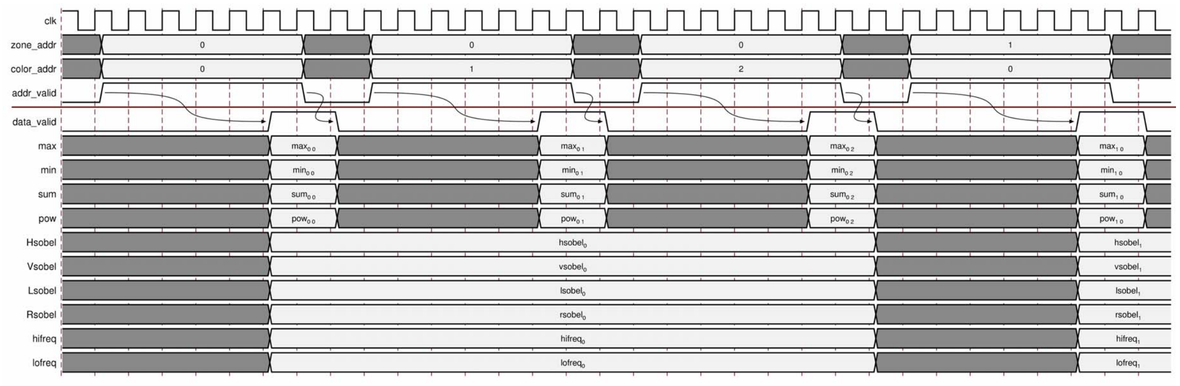


Figure 2-8: **Readout Addressing**

Figure 2-9 illustrates reading out histogram data. To shorten the readout period, histogram data can be read out in parallel with other statistical data.

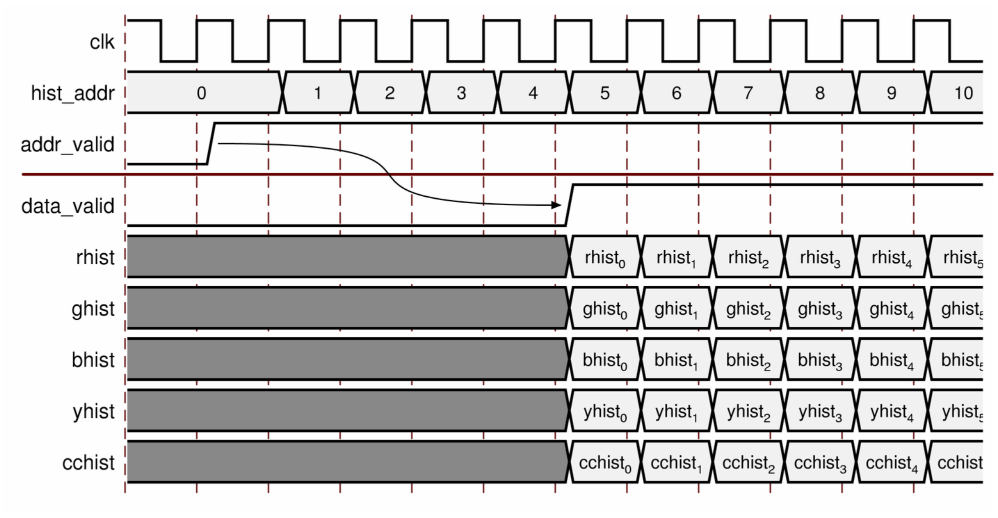

clk
hist_addr
0
1
2
3
4
5
6
7
8
9
10
addr_valid
data_valid
rhist
ghist
bhist
yhist
cchist
rhist0 rhist1 rhist2 rhist3 rhist4 rhist5
ghist0 ghist1 ghist2 ghist3 ghist4 ghist5
bhist0 bhist1 bhist2 bhist3 bhist4 bhist5
yhist0 yhist1 yhist2 yhist3 yhist4 yhist5
cchist0 cchist1 cchist2 cchist3 cchist4 cchist

Figure 2-9: **Reading Out Histogram Data**

RGB_HIST_ZONE_EN (0x011C) Register

This register enables zones 0 -15 (setup using the HMAX and VMAX registers) for RGB histogram data gathering.

YCC_HIST_ZONE_EN (0x0120) Register

This register enables zones 0 -15 (setup using the HMAX and VMAX registers) for Y and CC histogram data gathering.

ZONE_ADDR (0x0124) Register

This register selects which one of the sixteen zones will be read out.

COLOR_ADDR (0x0128) Register

This register selects which color component of the RGB histogram will be read out:

- 00 – Red
- 01 – Green
- 1x – Blue

HIST_ADDR (0x012C) Register

Selects the specific address [0 – DATA_WIDTH-1] for histogram data.

ADDR_VALID (0x0130) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

Minimum and Maximum Values

The minimum and maximum values can be useful for histogram stretching, Auto-Gain, Digital-Gain, Auto-Exposure, or simple White-Balance applications. These values are calculated for all zones and all R,G,B color channels simultaneously.

MAX (0x0134) Register

Maximum value measured for the currently selected zone and color channel.

MIN (0x0138) Register

Minimum value measured for the currently selected zone and color channel.

Sum of Color Values

The core provides the sum of color values for all zones (sum). The mean values for color channels can be calculated by dividing the sum value by the size of the zone (*N*):

$$\bar{x} = \frac{1}{N}\sum_{i=0}^{N-1} x_i = \frac{sum}{N}$$

Equation 2-1

SUM_LO/HI (0x013C, 0x0140) Register

Lower and upper values (64 bits total) containing the sum of the currently selected zone and color channel.

Sum of Squares of Color Values

The core provides the power output equal to the sum of squared values for all color channels and zones (*pow*), from which the signal power or the variance can be calculated:

$$\sigma^2 = \frac{1}{N}\left[\sum_{i=0}^{N-1} x_i^2\right] - \bar{x}^2 = \frac{pow}{N} - \bar{x}^2$$

Equation 2-2

POW_LO/HI (0x0144, 0x0148) Register

Lower and upper values (64 bits total) containing the sum of squares of the currently selected zone and color channel.

Edge Content

The Image Statistics core filters the luminance values calculated for all zones using the Sobel operators:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

Figure 2-10: **Horizontal, Vertical and Diagonal (Left and Right) Sobel Operators**

The Sobel operators are implemented without multipliers to reduce size and increase performance. The edge content outputs (*Hsobel*, *Vsobel*, *Lsobel*, *Rsobel*) provide the cumulative sums of absolute values of filtered luminance values:

$$Lsobel = \sum_{i=0}^{N-1} ABS\left(\frac{1}{32} FIR2D\left(x_i, \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}\right)\right)$$

Equation 2-3

Equation 2-3 describes the calculation of the upper-left to lower-right diagonal frequency content, *Lsobel*. Output values for *Vsobel*, *Lsobel*, and *Rsobel* are calculated similarly by using the corresponding coefficient matrixes from Figure 2-10.

HSOBEL_LO/HI (0x014C, 0x0150) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

VSOBEL_LO/HI (0x0154, 0x0158) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

LSOBEL_LO/HI (0x015C, 0x0160) Register

Bit 0 qualifies the `ZONE_ADDR`, `COLOR_ADDR`, and `HIST_ADDR`.

RSOBEL_LO/HI (0x0164, 0x0168) Register

Bit 0 qualifies the ZONE_ADDR, COLOR_ADDR, and HIST_ADDR.

Frequency Content

The frequency content for each zone is calculated using the luminance channel. To calculate low-frequency content, luminance values are first low-pass filtered with a 7 tap FIR filter, with fixed coefficients [-1 0 9 16 9 0 -1]/32.

The low frequency power output (*LoFreq*) of the core provides the cumulative sum of the squared values of the FIR filter output for each zone:

$$LoFreq = \sum_{i=0}^{N-1} \left[\frac{FIR(x_i, \{-1,0,9,16,9,0,-1\})}{32} \right]^2$$

Equation 2-4

In Equation 2-4, square brackets [] represent clipping at $max(x_i)=2^{DATA\_WIDTH-1}$ and clamping values at 0.

The high frequency power output (*HiFreq*) of the core provides the difference between the power of the original luminance values and the power of the low-pass filtered signal within each of these zones:

$$HiFreq = \lfloor pow - LoFreq \rfloor$$

Equation 2-5

In Equation 2-5, square brackets represent clamping values at 0.

HIFREQ_LO/HI (0x016C, 0x0170) Register

Bit 0 qualifies the ZONE_ADDR, COLOR_ADDR, and HIST_ADDR.

LOFREQ_LO/HI (0x0174, 0x0178) Register

Bit 0 qualifies the ZONE_ADDR, COLOR_ADDR, and HIST_ADDR.

DATA_VALID (0x0190) Register

Bit 0 qualifies the ZONE_ADDR, COLOR_ADDR, and HIST_ADDR.

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

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.

Customizing and Generating the Core

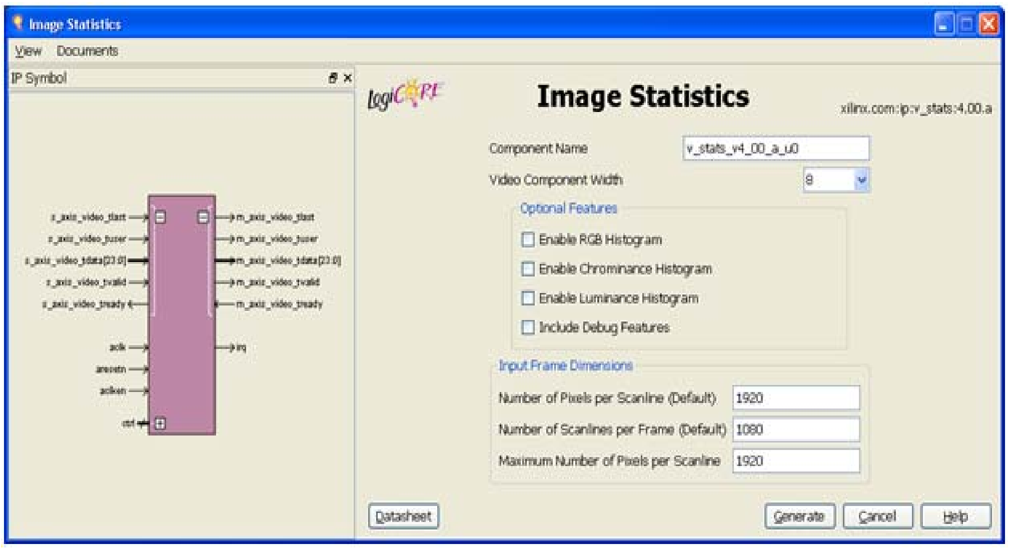
This chapter includes information on using Xilinx tools to customize and generate the core.

Graphical User Interface (GUI)

The Image Statistics core is easily configured to meet user-specific needs through the CORE Generator or EDK GUIs. This section provides a quick reference to parameters that can be configured at generation time.

CORE Generator GUI Controls

Figure 3-1 shows the Image Statistics CORE Generator GUI.



Component Name: v_stats_v4_00_a_u0

Video Component Width: 8

Optional Features

- [ ] Enable RGB Histogram
- [ ] Enable Chrominance Histogram
- [ ] Enable Luminance Histogram
- [ ] Include Debug Features

Input Frame Dimensions

Number of Pixels per Scanline (Default): 1920

Number of Scanlines per Frame (Default): 1080

Maximum Number of Pixels per Scanline: 1920

Figure 3-1: **Image Statistics CORE Generator GUI**

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

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters a to z, 0 to 9 and "_". The name v_stats_v4_00_a cannot be used as a component name.

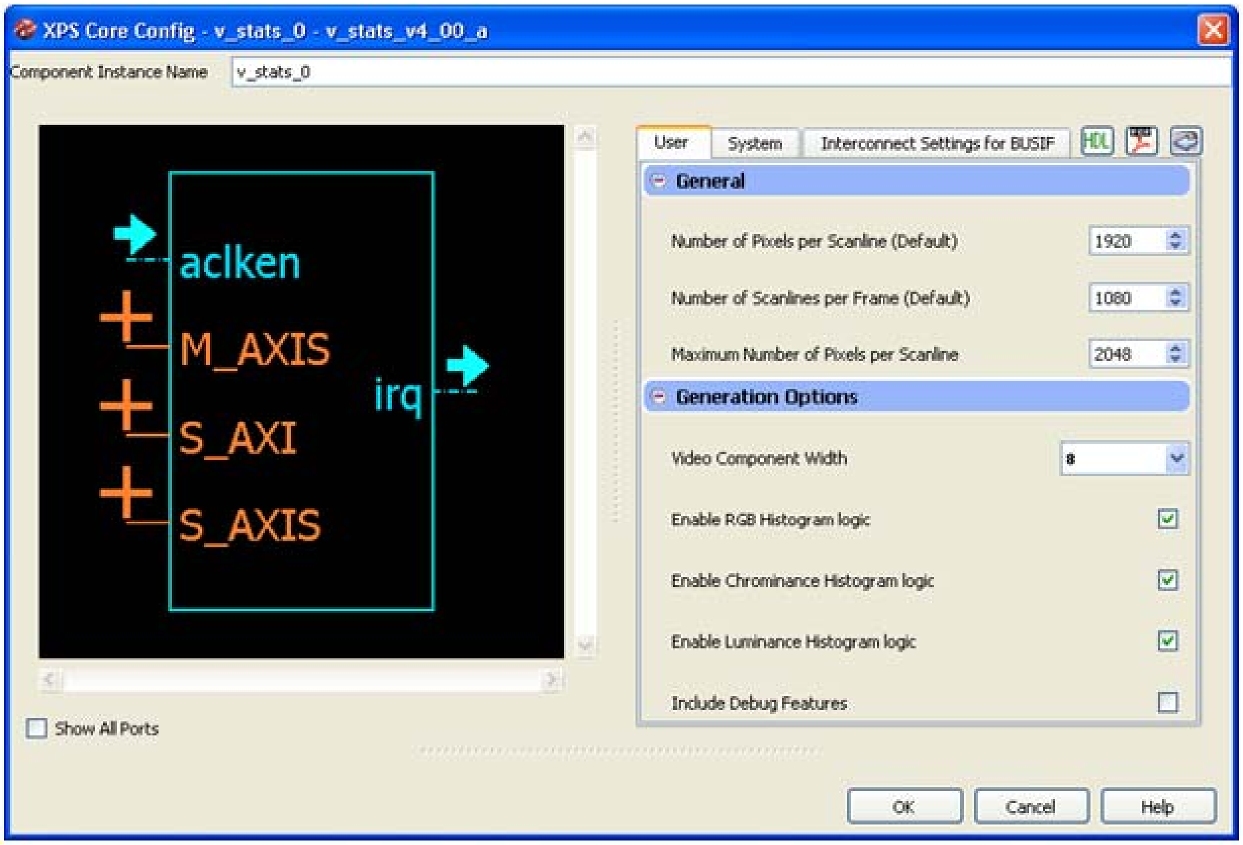
- **Video Component Width:** Specifies the bit width of input data. Permitted values are 8, 10, or 12.
- **Optional Features:** Storing and calculating histograms utilize block RAM resources in the FPGA. By specifying which histograms calculations are needed, this option can be used to reduce FPGA resources required for the generated core instance.
    - **Enable RGB Histograms:** The check box enables/disables instantiation of the R,G and B histogram calculating modules for zones pre-selected for RGB histogramming.
    - **Enable Chrominance Histogram:** The check box enables/disables instantiation of the two-dimensional chrominance (Cr-Cb) histogram calculating module for zones pre-selected for Y and CrCb histogramming
    - **Enable Luminance Histogram:** The check box enables/disables instantiation of the luminance histogram calculating module for zones pre-selected for Y and CrCb histogramming.

  Note: Storing and calculating histograms utilize block RAM resources in the FPGA. By specifying which histograms calculations are needed, this option can reduce FPGA resources required for the generated core instance.
- **Include Debugging Features**: When selected, the core will be generated with debugging features, which simplify system design, testing and debugging. For more information, refer to Debugging Features in Appendix C.

  Note: Debugging features are only available when the AXI4-Lite Register Interface is selected.
- **Input Frame Dimensions**:
    - **Number of Active Pixels per Scan line**: 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.
    - **Number of Active Lines per Frame**: 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.
    - **Maximum Number of Active Pixels Per Scan line**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of line buffers. The actual value selected for Number of Active Pixels per Scan line, or the corresponding lower half-word of the ACTIVE_SIZE register must always be less than the value provided by Maximum Number of Active Pixels Per Scan line. Using a tight upper-bound results in optimal block RAM usage.

EDK GUI Controls

Definitions of the EDK GUI controls are identical to the corresponding CORE Generator GUI functions, as shown in Figure 3-2.



Component Instance Name: v_stats_0

Number of Pixels per Scanline (Default): 1920

Number of Scanlines per Frame (Default): 1080

Maximum Number of Pixels per Scanline: 2048

Video Component Width: 8

Enable RGB Histogram logic ✓

Enable Chrominance Histogram logic ✓

Enable Luminance Histogram logic ✓

Include Debug Features ☐

Figure 3-2: **EDK GUI Screen**

Parameter Values in the XCO File

Table 3-1 defines valid entries for the XCO parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 3-1: **Parameter Values**

XCO Parameter	Default Values
active_cols	1920
active_rows	1080
component_name	v_stats_v4_00_a_u0

Table 3-1: **Parameter Values *(Cont'd)***

XCO Parameter	Default Values
data_width	8
has_cc_hist	false
has_debug	false
has_rgb_hist	false
has_y_hist	false
max_cols	1920

Output Generation

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

File Details

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

Table 3-2: **CORE Generator Output Files**

File 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.

Designing with the Core

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

General Design Guidelines

The Image Statistics core generates statistical information about the video data stream that passes through it. The statistical information is useful for applications such as Auto White Balance, Auto Exposure and Auto Gain. The core processes pixels provided through an AXI4-Stream slave interface, outputs pixels through an AXI4-Stream master interface, and can be controlled via an AXI4-Lite interface. The Image Statistics block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate nor does it modify the video stream in any way. It is recommended that the Image Statistics core is used in conjunction with the Video In to AXI4-Stream and Video Timing Controller cores. The Video Timing Controller core measures the timing parameters, such as number of active scan lines, number of active pixels per scan line of the image sensor. The Video In to AXI4-Stream core converts the incoming video data stream to AXI4-Stream.

Typically, the Image Statistics core is part of an Image Sensor Pipeline (ISP) System, as shown in Figure 4-1.

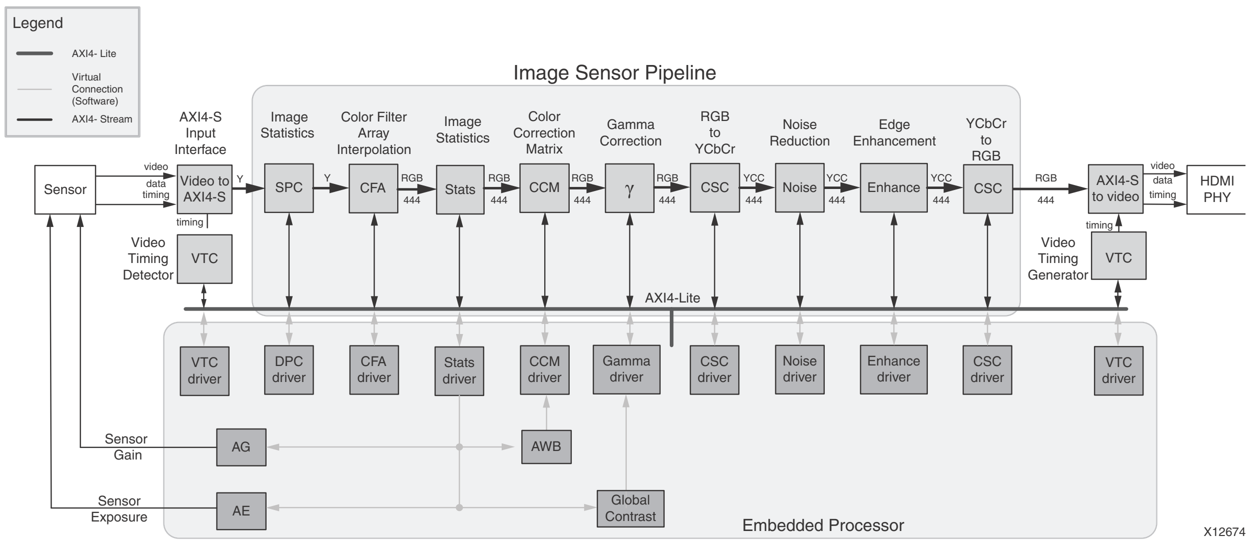


Figure 4-1: **Image Sensor Pipeline System**

Clock, Enable, and Reset Considerations

This section contains details about clocking, enable, and reset options for this core.

ACLK

The master and slave AXI4-Stream video interfaces use the ACLK clock signal as a shared clock reference, as shown in Figure 4-2.

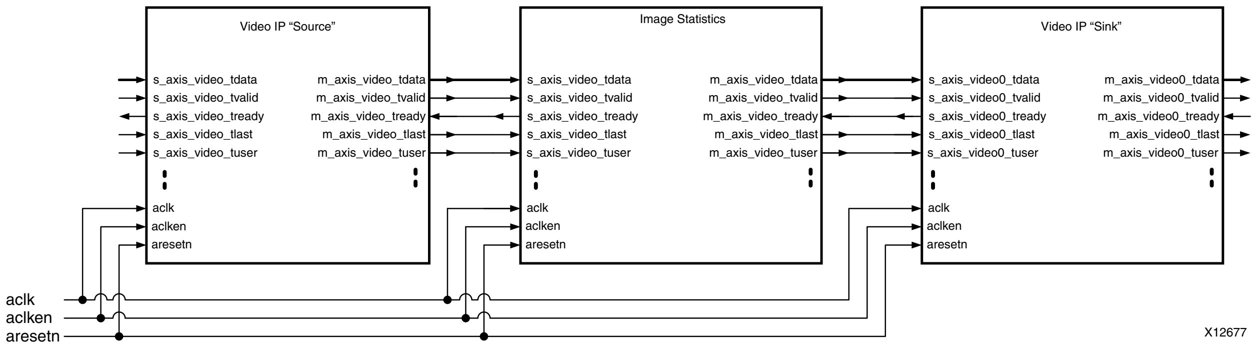


Figure 4-2: **Example of ACLK Routing in an ISP Processing Pipeline**

The ACLK pin is also shared between the AXI4-Lite and AXI4-Stream interfaces. The Image Statistics core does not contain optional clock-domain crossing logic. If in the user system AXI4-Lite Control interface clock (CLK_LITE) is different from the AXI4-Stream clock (CLK_STREAM), clock-domain crossing logic needs to be inserted as follows:

- Is $F_{CLK\_STREAM} > F_{CLK\_LITE}$, clock-domain crossing logic needs to be inserted in front of the AXI4-Lite Control interface and the CFA core can be clocked at the AXI4-Stream clock via ACLK, as shown in Figure 4-3.

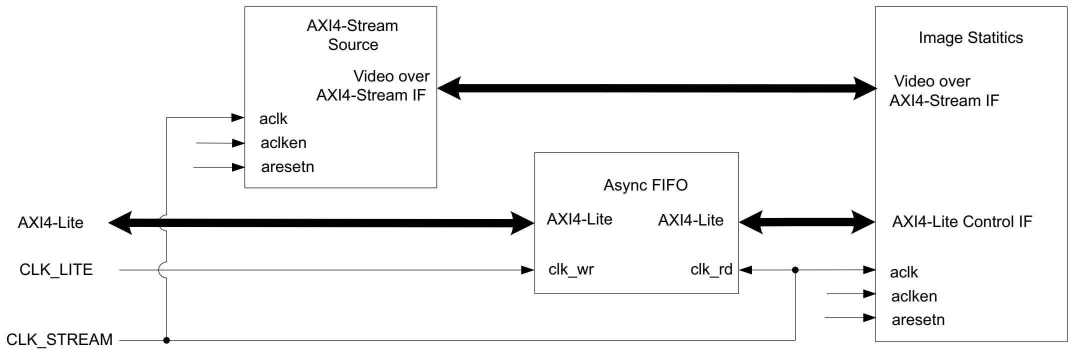


Figure 4-3: **Clock-Domain Crossing when $F_{CLK\_STREAM} > F_{CLK\_LITE}$**

- If $F_{CLK\_STREAM} < F_{CLK\_LITE}$, clock-domain crossing logic needs to be inserted before the AXI4-Stream interface, and the CFA core needs to be clocked at the AXI4-Lite clock via the ACLK pin, as shown in Figure 4-4.

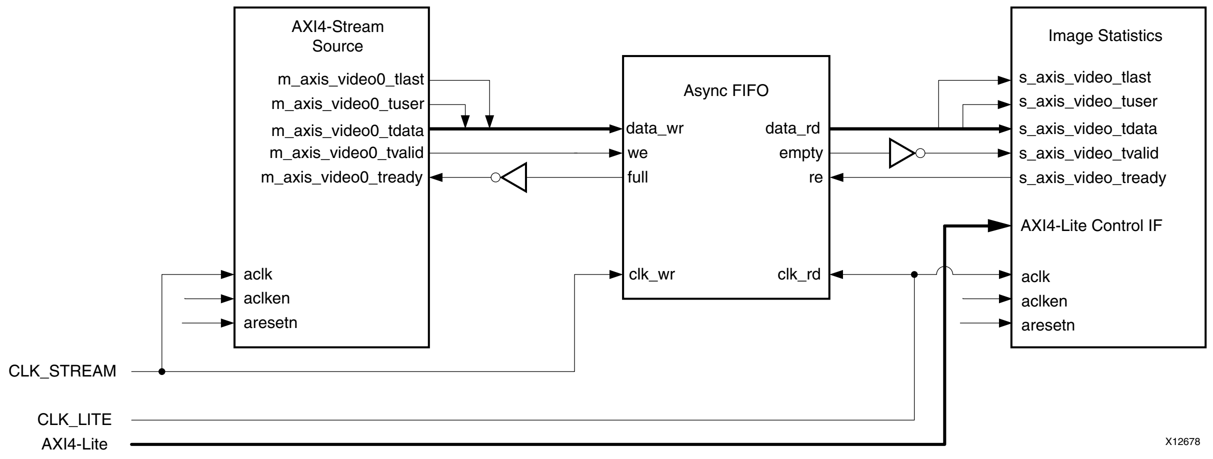


Figure 4-4: **Image Statistics Core Top-Level Signaling Interface**

Regardless of if F_{CLK_STREAM} is less than or greater than F_{CLK_LITE}, Xilinx System Integrator tools, such as EDK, can automatically infer clock-domain crossing logic using the AXI interconnect core, when the tool detects that the master / slave side of AXI4 interfaces operate on different CLK rates. For manual instantiation of clock-domain crossing logic, HDL users can take advantage of the FIFO Generator IP core, as shown in Figure 4-4.

ACLKEN

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

The ACLKEN pin facilitates:

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

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

If two cores are connected through AXI4-Stream interfaces and only the master or the slave interface has an ACLKEN port (not permanently tied High), the two interfaces should be connected through the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).

ARESETn

The Image Statistics core has two reset sources:

- ARESETn pin (hardware reset)
- Software reset option provided through the AXI4-Lite control interface (when present)

ARESETn is not synchronized internally to AXI4-Stream frame processing. Therefore de-asserting ARESETn while a frame is being processed will lead to image tearing. The external reset pulse must be held for 32 ACLK cycles to reset the core.

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

System Considerations

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

Programming Sequence

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

Error Propagation and Recovery

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

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

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

Constraining the Core

This chapter contains information on applicable constraints for the Image Statistics core.

Required Constraints

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

Device, Package, and Speed Grade Selections

There are no device, package or speed grade requirements for this core. The core has not been characterized for use in low power devices."

Clock Frequencies

The clock for this should be run at the required pixel clock frequency.

Clock Management

There is only one clock for this core.

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 this time. For a comprehensive listing of Video and Imaging application notes, white papers, related IP cores including the most recent reference designs available, see the Video and Imaging Resources page at:

www.xilinx.com/esp/video/refdes_listing.htm

Demonstration Test Bench

A demonstration test bench is provided which demonstrates a typical use scenario. The user is encouraged to make simple modifications to the test conditions and observe the changes in the waveform.

Test Bench Structure

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

- DUT

  The Image Statistics 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 Mentor Graphics 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 `run_mti.bat`.

Simulation using iSim:

Double-click `run_isim.bat.`

Directory and File Contents

The directory structure underneath the top-level folder is:

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

  The available core C model can be used to generate stimuli and expected results for any user BMP image. For more information, see Appendix E, C Model Reference.

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

- 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

Appendix A

Verification, Compliance, and Interoperability

This chapter contains details about verification and testing of the core.

Simulation

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

- Register accesses
- Processing of multiple frames of data
- AXI4-Stream bidirectional data-throttling tests
- Testing detection and recovery from various AXI4-Stream framing error scenarios
- Testing different ACLKEN and ARESETn assertion scenarios
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Image Statistics core has been tested in a variety of hardware platforms at Xilinx to represent a variety of parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Lite interconnect and various other peripherals. The software for the test system included pre-generated input and output data along with live video stream. The MicroBlaze processor was responsible for:
    - Initializing the appropriate input and output buffers
    - Initializing the Image Statistics core.
    - Launching the test.

- Comparing the output of the core against the expected results.
- Reporting the pass/fail status of the test and any errors that were found.

Migrating

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

Migrating from v3.0 to v4.00.a

From version v3.0 to v4.00.a of the Image Statistics core the following significant changes took place:

- XSVI interfaces were replaced by AXI4-Stream interfaces.
- The pCore and General Purpose Processor became obsolete and were removed.
- Native support for EDK was added. The Image Statistics core now appears in the EDK IP Catalog.
- Debugging features were added.
- The AXI4-Lite control interface register map is now standard for all Xilinx video IP cores.

Because of the complex nature of these changes, replacing a v3.0 version of the core in a customer design is not trivial. An existing EDK pCore instance can be converted from XSVI to AXI4-Stream, using the Video In to AXI4-Stream core or components from XAPP521, *Bridging Xilinx Streaming Video Interface with the AXI4-Stream Protocol*.

A v3.0 pCore instance in EDK can be replaced from v4.00.a directly from the EDK IP Catalog. However, the application software needs to be updated for the changed functionality and addresses of the IRQ_ENABLE, STATUS, ERROR, and the core's specific registers.

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

- Timing detection and generation using the Video Timing Controller core.
- Replacing XSVI interfaces with conversion modules described in XAPP521.

Debugging

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

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

1. Bring up the AXI4-Lite interface
2. Bring up the AXI4-Stream interfaces
3. (Optional) Balancing throughput

Bringing up the AXI4-Lite Interface

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

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

Symptom	Solution
Readback value for the VERSION_REGISTER is different from expected default values	Does the core receive ACLK? Is the core enabled? Set ACLKEN=1 Is the core in reset? Set ARESETn=1. The address maps between software and hardware could get out of sync. Regenerate addresses in EDK, make sure the MHS file in EDK and the xparameters.h in the SDK project are up to date.
Readback values from values are stuck, and cannot be overwritten.	Is the target address writable?
The interface is unreliable. Subsequent reads from the same address return different value, readback values differ from values written to the same address.	Clock domain crossing issues between the host processor and the peripheral. HDL users need to make sure the AXI4-Lite Master is in the same clock domain as the AXI4-Lite Slave. If not, proper clock-domain crossing logic, such as asynchronous FIFOs need to be inserted.

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

Bringing up the AXI4-Stream Interfaces

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

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

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using ChipScope analyzer, 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 analyzer, 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 Image Statistics 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 Image Statistics 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 STATS 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 Image Statistics core (Asynchronous FIFO)? Did the design meet timing? Is the frequency of the clock source driving the Image Statistics ACLK pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the STATS 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 Image Statistics core (Asynchronous FIFO)? Did the design meet timing? Is the frequency of the clock source driving the Image Statistics ACLK pin lower than the reported Fmax reached?

Debugging Features

The Image Statistics core is equipped with optional debugging features to accelerate system bring-up, optimize memory and data-path architecture and reduce time to market. The optional debug features can be turned on/off via the **Include Debug Features** checkbox on the GUI. Turning off debug features reduces the core slice footprint.

Core Bypass Option

The bypass option facilitates establishing a straight through connection between input (AXI4-Stream slave) and output (AXI4-Stream master) interfaces bypassing any processing functionality.

Flag BYPASS (bit 4 of the CONTROL register) can turn bypass on (1) or off, when the core instance Debugging Features were enabled at generation. Within the IP this switch controls multiplexers in the AXI4-Stream path.

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

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

Built in Test-Pattern Generator

The optional built-in test-pattern generator facilitates to temporarily feed the output AXI4-Stream master interface with a predefined pattern. Flag TEST_PATTERN (bit 5 of the CONTROL register) can turn test-pattern generation on (1) or off, when the core instance Debugging Features were enabled at generation. Within the IP this switch controls multiplexers in the AXI4-Stream path, switching between the regular core processing output and the test-pattern generator. When enabled, a set of counters generate 256 scan-lines of color-bars, each color bar 64 pixels wide, repetitively cycling through Black, Red, Green, Yellow, Blue, Magenta, Cyan, and White colors till the end of each scan-line. After the Color-Bars segment, the rest of the frame is filled with a monochrome horizontal and vertical ramp.

To enable successive bring-up and parameterization of subsequent cores:

1. Start the system with all processing cores set to test-pattern mode.
2. Then turn the test-pattern generation off from the system output towards the system input.

Throughput Monitors

Throughput monitors enable the user to monitor processing performance within the core. This information can be used to help debug frame-buffer bandwidth limitation issues, and if possible, allow video application software to balance memory pathways.

Often video systems with multi-port access to a shared external memory, have different processing islands. For example, a pre-processing sub-system working in the input video clock domain may clean up, transform, and write a video stream, or multiple video streams, to memory. The processing sub-system may read the frames out, process, scale, encode, then write frames back to the frame buffer, in a separate processing clock domain. Finally, the output sub-system may format the data and read out frames locked to an external clock.

Typically, access to external memory using a multi-port memory controller involves arbitration between competing streams. However, to maximize the throughput of the system, different memory ports may need different specific priorities. To fine tune the arbitration and dynamically balance frame rates, it is beneficial to have access to throughput information measured in different video data paths.

The SYSDEBUG0 (0x0014), or Frame Throughput Monitor, register indicates the number of frames processed since power-up or the last time the core was reset. The SYSDEBUG1 (0x0018), or Line Throughput Monitor, register indicates the number of lines processed since power-up or the last time the core was reset. The SYSDEBUG2 (0x001C), or Pixel

Throughput Monitor, register indicates the number of pixels processed since power-up or the last time the core was reset.

Priorities of memory access points can be modified by the application software dynamically to equalize frame, or partial frame rates.

Interfacing to Third-Party IP

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

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

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

Application Software Development

This appendix contains details about programming the core.

Processing States

The core distinguishes acquisition and readout periods to avoid modification of single-buffered measurement data while it is being read out. After readout, block RAMs and registers have to be re-initialized before the next acquisition cycle may commence.

After reset or power-up, the core cycles through the "Initialization," "Wait for Start of Frame," and "Data Acquisition" states.

Figure D-1 shows the top-level state diagram of the Image Statistics core.

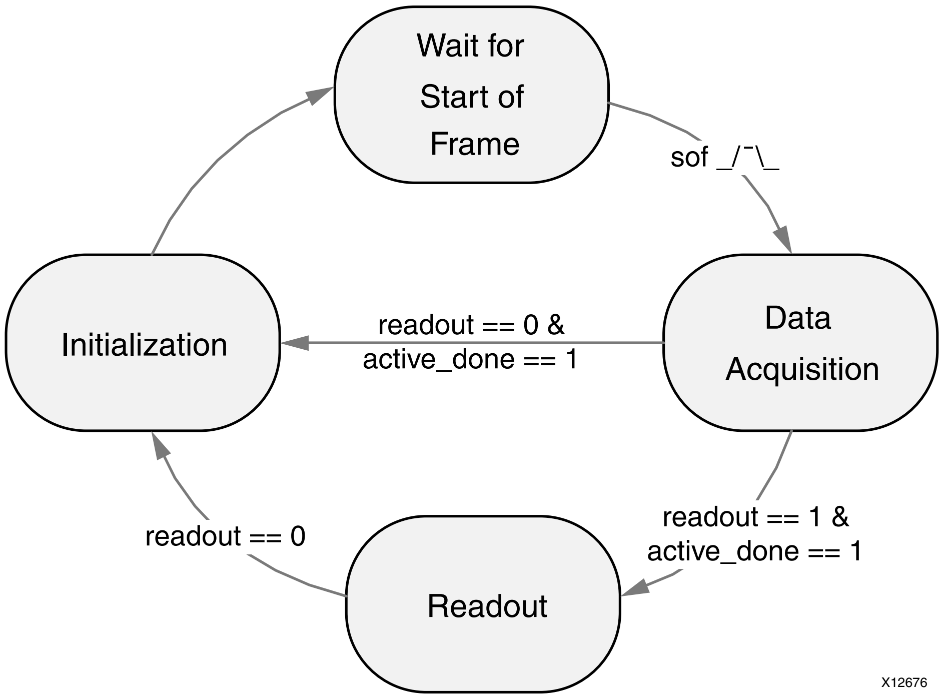


Figure D-1: **Image Statistics IP Core State Diagram**

Initialization

The one- and two-dimensional histograms are stored in block RAMs, which need to be cleared before the IP core can start data acquisition. Clearing of block RAMs may take 256, 1024 or 4096 CLK cycles corresponding to 8, 10, or 12-bit wide input data. Block RAM initialization may take several scan-lines, depending on the input resolution.

Once the core is finished with block RAM initialization, it progresses to the “Wait for Start of Frame” state where it remains until a rising edge on `sof` (Start of Frame) is detected, at which time it enters the “Data Acquisition” state.

Data Acquisition

In the “Data Acquisition” state, the core updates all internal measurement values with the pixel data presented on `video_data_in` when data is qualified with `active_video_in` = 1.

If the `READOUT` bit in the `CONTROL` register is set, the core proceeds to the “Readout” state (bit 6). Otherwise, the core proceeds to the Initialization state after the last active scan-line, which may occur several scan-lines before the rising edge on End of Frame (`eof`).

Readout

In the readout state, the core does not collect any more statistical information, and the multiplexing/addressing mechanism on the output is activated. Once the user provides addresses that are qualified valid by asserting the `addr_valid` pin, the core fetches and displays information on its output ports pertaining to the input addresses. Valid output data is identified by the `DATA_VALID` register (address offset 0x0190).

Reading Out Statistical Results

Figure D-3 shows the reg_update register being asserted to a '1' and then the HMAXn and VMAXn registers transitioning from their default values to the new values.

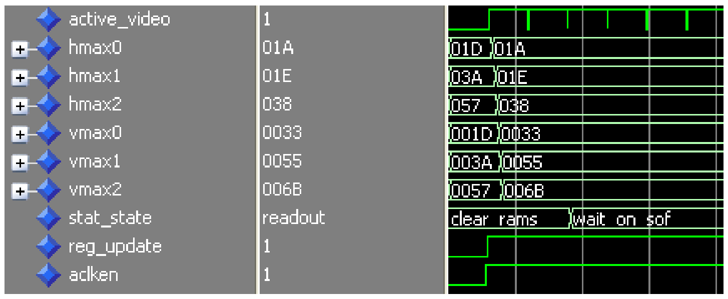
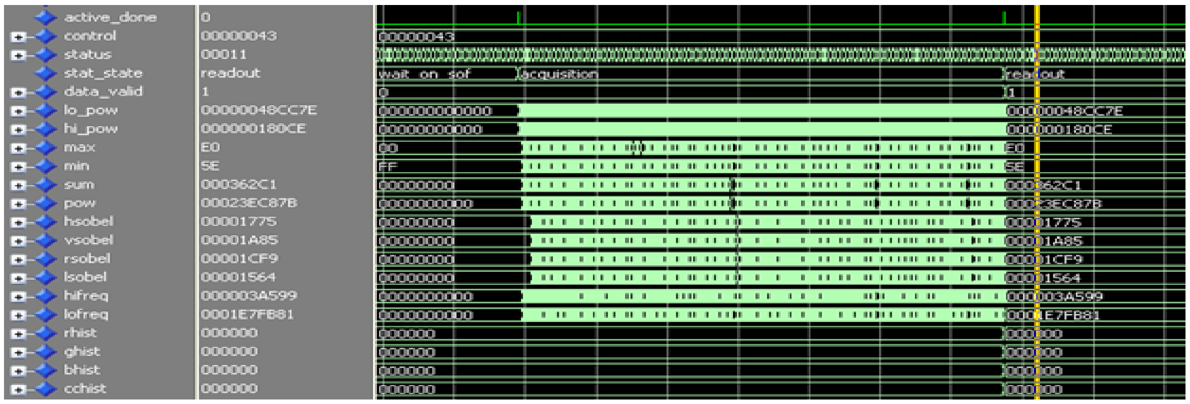


Figure D-2: **Register Update Example**

Figure D-3 shows an example of data readout timing and use of the control (specifically bits `REG_UPADTE` and `READOUT`) and `STATUS` (shown in hexadecimal format) registers. In this example, an empty frame followed by a test frame (the frames are bounded by the `active_done` signal) are processed by the core. The core cycles through the "Initialization" (clearing block RAMs) and "Wait on Start of Frame" stages, from which it transitions to the "Data Acquisition" state on the second falling edge of `SOF`.



active_done	0
control	00000043
status	00011
stat_state	readout
data_valid	1
lo_pow	00000048CC7E
hi_pow	0000000180CE
max	E0
min	5E
sum	000362C1
pow	00023EC87B
hsobel	00001775
vsobel	00001A85
rsobel	00001CF9
lsobel	00001564
hifreq	000003A599
lofreq	0001E7FB81
rhist	000000
ghist	000000
bhist	000000
cchist	000000

Figure D-3: **Frame and Readout Timing**

As discussed in the Synchronization, page 62 section, the Image Statistics core signals the end of data acquisition by asserting the `DONE` flag of the `STATUS` register, which can also trigger an interrupt if the `IRQ_ENABLE` register is set up to enable interrupts.

During the frame, bit 2 (`READOUT`) was asserted, which at the end of the active portion of the frame instructs the core to enter the "Readout" state.

Bit `READOUT` of the `CONTROL` register has to be set before the end of the frame; otherwise the core does not enter the readout mode but clears measurement data and arms itself for capturing the next frame.

After the host processor is finished reading out relevant statistical data, it programs bit 6 (`READOUT`) of the `CONTROL` register to 0, which instructs the core to re-initialize by entering the "clear-RAMs" state. The example in Figure D-3 also demonstrates the use of the `REG_UPDATE` flag. The user at any point could have modified values for input registers, such as `HMAX0`, `HMAX1`, `HMAX2`, `VMAX0`, `VMAX1`, `VMAX2`, `RGB_HIST_ZONE_EN`, `YCC_HIST_ZONE_EN`, or `HIST_ZOOM_FACTOR`. After setting all input registers to their desired value, `REG_UPDATE` was asserted, which resulted to all internal registers to latch in the user input at the rising edge of End of Frame. Registers `HMAXn`, `VMAXn`, `RGB_HIST_ZONE_EN`, `YCC_HIST_ZONE_EN` and `HIST_ZOOM_FACTOR` values displayed in Figure D-3 demonstrate how the values change simultaneously on the rising edge of End of Frame if `REG_UPDATE` is set to 1.

Synchronization

A semaphore-based mechanism is used to synchronize external frame timing with host processor register writes, data readouts and data acquisition.

The semaphores involved in synchronizing host processor activity with the incoming video stream are:

- `DONE` flag (bit 4 of the STATUS register)
- `REG_UPDATE` (bit 1 of the CONTROL register)
- `READOUT` (bit 6 of the CONTROL register)
- `CLR_STATUS` (bit 2of the CONTROL register)

The software flow diagram for normal system operation is shown in Figure D-4.

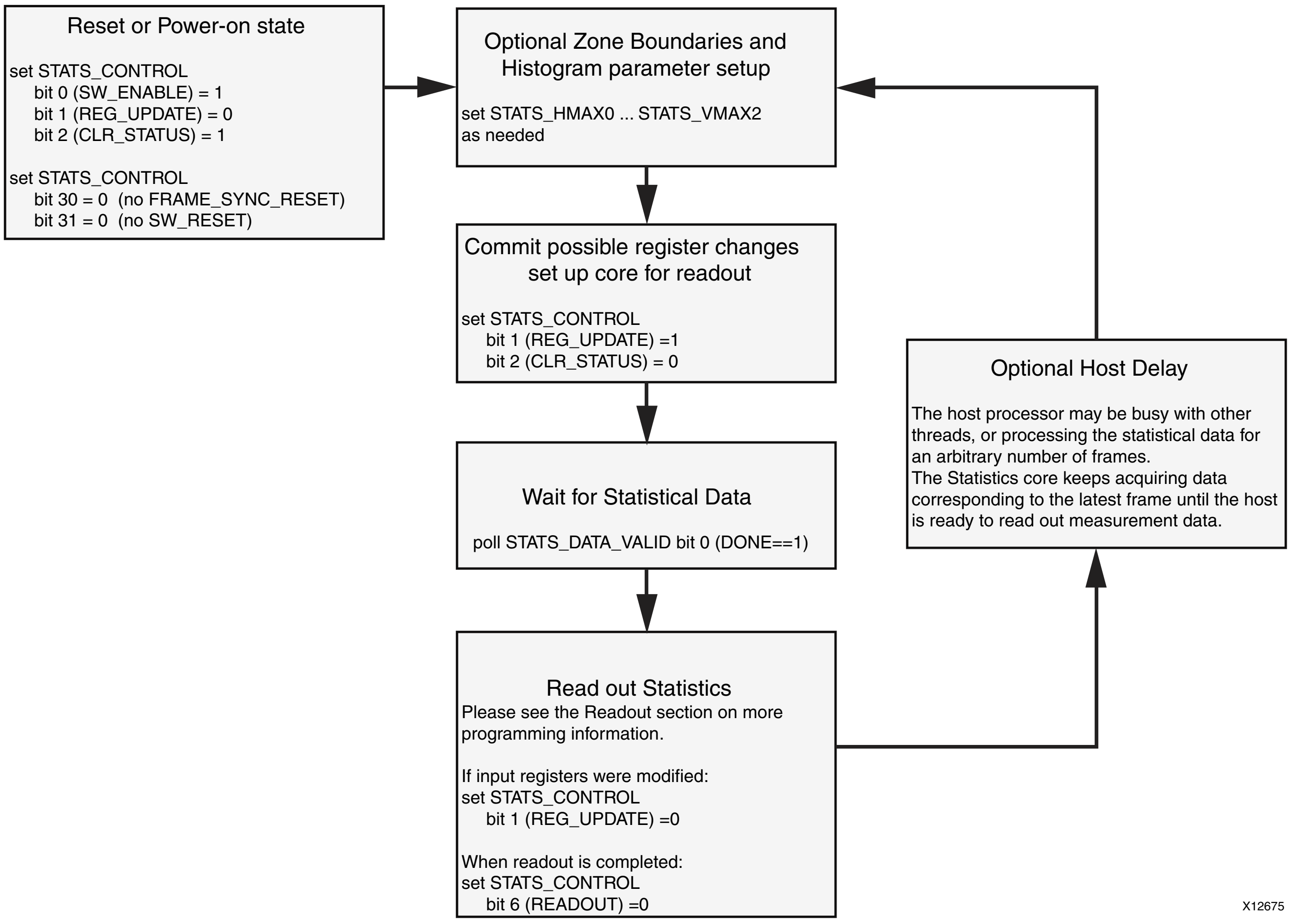

Reset or Power-on state
set STATS_CONTROL
bit 0 (SW_ENABLE) = 1
bit 1 (REG_UPDATE) = 0
bit 2 (CLR_STATUS) = 1
set STATS_CONTROL
bit 30 = 0 (no FRAME_SYNC_RESET)
bit 31 = 0 (no SW_RESET)
Optional Zone Boundaries and Histogram parameter setup
set STATS_HMAX0 ... STATS_VMAX2 as needed
Commit possible register changes set up core for readout
set STATS_CONTROL
bit 1 (REG_UPDATE) =1
bit 2 (CLR_STATUS) = 0
Wait for Statistical Data
poll STATS_DATA_VALID bit 0 (DONE==1)
Read out Statistics
Please see the Readout section on more programming information.
If input registers were modified:
set STATS_CONTROL
bit 1 (REG_UPDATE) =0
When readout is completed:
set STATS_CONTROL
bit 6 (READOUT) =0
Optional Host Delay
The host processor may be busy with other threads, or processing the statistical data for an arbitrary number of frames.
The Statistics core keeps acquiring data corresponding to the latest frame until the host is ready to read out measurement data.
X12675

Figure D-4: **Software Flow Diagram for Normal System Operation**

When data acquisition is finished, the core asserts status flag `DONE`.

After the data acquisition state, depending on the state of the READOUT flag, the core either enters the readout state (READOUT = 1), or discards measurement data by re-initializing block RAMs and registers and preparing for acquiring data from the next frame (READOUT = 0).

This mechanism relieves the host processor from having to service the core when statistical data is not needed and allows the core to continuously process frames, so when the host processor is ready to poll statistical information, information from the last frame is available.

If the core enters the readout state, it remains there until the host processor signals being done with reading out measurement data, which may take a few lines, or several frames depending on the speed and the workload of the host processor. Therefore termination of the readout state is decoupled from the input video stream.

Once the host processor deasserts READOUT, the core immediately clears (re-initializes) block RAMs and registers and proceeds to acquire data from the next frame.

After deasserting READOUT, the host processor may assert it again immediately, enabling the core to enter the "Readout" state after acquisition of the current frame is complete.

NOTES:

1. READOUT has to be asserted before the statistics core is done acquiring the next frame, or the core discards the data and self-triggers to acquire the next frame.
2. For the core to process every subsequent frame, the vertical blanking period has to be at least as long as the number of scan-lines it takes to initialize the block RAMs. For example, in the pessimistic case of using SD sensor (720 pixels per line) with 12 bit data (4k deep block RAMs), the minimum vertical blanking period has to be 4096/720 = 5.68, or at least six lines.

Programmer's Guide

The software API is provided to allow easy access to the Image Statistics AXI4-Lite registers defined in Table 2-12. To use the API functions, the following two header files must be included in the user C code:

```
#include "stats.h"
#include "xparameters.h"
```

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

For examples on API function calls and integration into a user application, the driver's subdirectory contains a file, `example.c`, in the `stats_v4_00_a0_a/example` subfolder. This file is a sample C program that demonstrates how to use the Image Statistics API.

Table D-1: **Image Statistics Driver Function Definitions**

Function name and parameterization	Description
STATS_Enable(uint32 BaseAddress)	Enables a STATS instance.
STATS_Disable(uint32 BaseAddress)	Disables a STATS instance.
STATS_Reset(uint32 BaseAddress)	Immediately resets a STATS instance. The core stays in reset until the RESET flag is cleared.
STATS_ClearReset(uint32 BaseAddress)	Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
STATS_FSyncReset(uint32 BaseAddress)	Resets a STATS instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.
STATS_ReadReg(uint32 BaseAddress, uint32 RegOffset)	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-12).
STATS_WriteReg(uint32 BaseAddress, uint32 RegOffset, uint32 Data)	Write the register selected by RegOffset (defined in Table 2-12). Data is the 32-bit value to write to the register.
STATS_RegUpdateEnable(uint32 BaseAddress)	Enables copying double buffered registers at the beginning of the next frame. Please see section "Double Buffering" below.
STATS_RegUpdateDisable(uint32 BaseAddress)	Disables copying double buffered registers at the beginning of the next frame. Please see section "Double Buffering" below.
STATS_RegUpdateDisable(uint32 BaseAddress)	Clears the status register of the STATS instance, by first asserting then deasserting the CLEAR_STATUS flag of STATS_CONTROL. This function only works when the STATS core is enabled.

Software Reset

Software reset reinitializes registers of the AXI4-Lite control interface to their initial value, resets FIFOs, forces `m_axis_video_tvalid` and `s_axis_video_tready` to 0. STATS_Reset() and STATS_FSync_Reset () reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling STATS_Reset(), or setting bit 30 of the CONTROL register to 1 will cause statistical data to be lost for that frame.

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

Note: Calling STATS_FSync_Reset() does not guarantee prompt, or real-time resetting of the core.

If the AXI4-Stream communication is halted mid frame, the core will not reset until the upstream core finishes sending the current frame or starts a new frame.

Double Buffering

Image Statistics core and ACTIVE_SIZE registers are double-buffered. Values from the AXI4-Lite interface are latched into processor registers immediately after writing, and processor register values are copied into the active register set at the Start Of Frame (SOF) signal. Double-buffering decouples AXI4-Lite register updates from the AXI4-Stream processing, allowing software a large window of opportunity to update processing parameter values.

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the RegUpdateEnable() and RegUpdateDisable() functions allows synchronous commitment of register changes.

The Image Statistics core will start using the updated ACTIVE_SIZE and core register values only if the REGUPDATE flag of the CONTROL register is set (1), after the next Start-Of-Frame signal (`s_axis_video_tuser`) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 2-12 has a constant that is defined in `stats.h` which is set to the offset for that register listed in Table D-2. It is recommended that the application software uses the predefined register names instead of register values when accessing core registers. This ensures that future updates to the Image Statistics drivers that change register locations will not affect the application dependent on the driver.

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

Constant Name Definition	Value	Target Register
STATS_CONTROL	0x0000	CONTROL
STATS_STATUS	0x0004	STATUS
STATS_ERROR	0x0008	ERROR
STATS_IRQ_ENABLE	0x000C	IRQ_ENABLE
STATS_VERSION	0x0010	VERSION
STATS_SYSDEBUG0	0x0014	SYSDEBUG0
STATS_SYSDEBUG1	0x0018	SYSDEBUG1
STATS_SYSDEBUG2	0x001C	SYSDEBUG2
STATS_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
STATS_HAMX0	0x0100	HMAX0
STATS_HAMX1	0x0104	HMAX1
STATS_HAMX2	0x0108	HMAX2
STATS_VAMX0	0x010C	VMAX0
STATS_VAMX1	0x0110	VMAX1
STATS_VAMX2	0x0114	VMAX2
STATS_ HIST_ZOOM_FACTOR	0x0118	HIST_ZOOM_FACTOR
STATS_ RGB_HIST_ZONE_EN	0x011C	RGB_HIST_ZONE_EN
STATS_ YCC_HIST_ZONE_EN	0x0120	YCC_HIST_ZONE_EN
STATS_ ZONE_ADDR	0x0124	ZONE_ADDR
STATS_COLOR_ADDR	0x0128	COLOR_ADDR
STATS_ HIST_ADDR	0x012C	HIST_ADDR
STATS_ ADDR_VALID	0x0130	ADDR_VALID
STATS_MAX	0x0134	MAX
STATS_MIN	0x0138	MIN
STATS_SUM_LO	0x013C	SUM_LO
STATS_ SUM_HI	0x0140	SUM_HI
STATS_POW_LO	0x0144	POW_LO
STATS_POW_HI	0x0148	POW_HI
STATS_ HSOBEL_LO	0x014C	HSOBEL_LO
STATS_ HSOBEL_HI	0x0150	HSOBEL_HI
STATS_VHSOBEL_LO	0x0154	VSOBEL_LO
STATS_ VSOBEL_HI	0x0158	VSOBEL_HI
STATS_ LSOBEL_LO	0x015C	LSOBEL_LO
STATS_ LSOBEL_HI	0x0160	LSOBEL_HI
STATS_ RSOBEL_LO	0x0164	RSOBEL_LO

Table D-2: **Predefined Constants Defined in stats.h *(Cont'd)***

Constant Name Definition	Value	Target Register
STATS_ RSOBEL_HI	0x0168	RSOBEL_HI
STATS_ HIFREQ_LO	0x016C	HIFREQ_LO
STATS_ HIFREQ_HI	0x0170	HIFREQ_HI
STATS_ LOFREQ_LO	0x0174	LOFREQ_LO
STATS_ LOFREQ_HI	0x0178	LOFREQ_HI
STATS_RHIST	0x017C	RHIST
STATS_GHIST	0x0180	GHIST
STATS_BHIST	0x0184	BHIST
STATS_YHIST	0x0188	YHIST
STATS_CCHIST	0x018C	CCHIST
STATS_DATA_VALID	0x0190	DATA_VALID

C Model Reference

This chapter contains information for installing the Image Statistics Engine C-Model, and describes the file contents and directory structure.

Software Requirements

The Image Statistics v4.00.a C models were compiled and tested with the following software:

Table E-1: **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 2005 Visual Studio 2008 (Visual C++ 8.0)

Installation

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

Linux

Ensure that the directory in which the `libIp_v_stats_v4_00_a_bitacc_cmodel.so` and `libstlport.so.5.1` files are located is in your $LD_LIBRARY_PATH environment variable.

C Model File Contents

Unzipping the `v_stats_v4_00_a_bitacc_model.zip` file creates the directory structures and files shown in Table E-2.

Table E-2: **C Model Directory Structure and Contents**

Name	Description
/lin	Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Linux Platforms
libIp_v_stats_v4_00_a_bitacc_cmodel.lib	Image Statistics model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by the Image Statistics and RGB to YCrCb object libraries (Linux platforms only)
run_bitacc_cmodel	Pre-compiled bit accurate executable for simulation on 32-bit Linux Platforms
/lin64	Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Linux Platforms
libIp_v_stats_v4_00_a_bitacc_cmodel.lib	Image Statistics model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by the Image Statistics and RGB to YCrCb object libraries (Linux platforms only)
run_bitacc_cmodel	Pre-compiled bit accurate executable for simulation on 64-bit Linux Platforms
/nt	Pre-compiled bit accurate ANSI C reference model for simulation on 32-bit Windows Platforms
libIp_v_stats_v4_00_a_bitacc_cmodel.lib	Pre-compiled library file for win32 compilation (Windows platforms only)
run_bitacc_cmodel.exe	Pre-compiled bit accurate executable for simulation on 32-bit Windows Platforms
/nt64	Pre-compiled bit accurate ANSI C reference model for simulation on 64-bit Windows Platforms
libIp_v_stats_v3_0_bitacc_cmodel.lib	Pre-compiled library file for win64 compilation (Windows platforms only)
run_bitacc_cmodel.exe	Pre-compiled bit accurate executable for simulation on 64-bit Windows Platforms
README.txt	Release notes
pg008_v_stats.pdf	LogiCORE IP Image Statistics Product Guide
v_stats_v4_00_a_bitacc_cmodel.h	Model header file
run_bittacc_cmodel.c	Example code calling the C model
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I/O functions
video_utils.h	Header file declaring the generalized image / video container type, I/O, and support functions
Kodim11.bmp	768x512 sample test image from the True-color Kodak test images

Using the C Model

The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file `v_stats_v4_00_a_bitacc_cmodel.h`.

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

```
struct  xilinx_ip_v_stats_v4_00_a_generics stats_generics;
struct  xilinx_ip_v_stats_v4_00_a_inputs   stats_inputs;
struct  xilinx_ip_v_stats_v4_00_a_outputs  stats_outputs;
```

Declaration of the preceding structs can be found in `v_stats_v4_00_a_bitacc_cmodel.h`.

The only generic parameter the Image Statistics v3.0 IP Core bit accurate model takes is the DATA_WIDTH, corresponding to the CORE Generator™ software "Data Width" parameter. Allowed values are 8,10 and 12.

Calling `xilinx_ip_v_stats_v4_00_a_get_default_generics(&stats_generics)` initializes the generics structure with the Image Statistics GUI default DATA_WIDTH value (8).

The structure `stats_inputs` defines the values of run-time parameters and the actual input image. The structure holds the following members:

Table E-3: **Member Variables of the Input Structure**

Type	Name	Function
video_struct	video_in	Holds the input video stream (may contain multiple frames)
int	hmax0	Column index of the first vertical zone delineator (1)
int	hmax1	Column index of the second vertical zone delineator (1)
int	hmax2	Column index of the third vertical zone delineator (1)
int	vmax0	Row index of the first horizontal zone delineator (1)
int	vmax1	Row index of the second horizontal zone delineator (1)
int	vmax2	Row index of the third horizontal zone delineator (1)
int	hist_zoom_factor	Controls CrCb histogram zoom around the gray (center point), which can be useful for white-balance algorithms. 0: No zoom, full Cb and Cr range represented (lowest resolution) 1: Zoom by 2 2: Zoom by 4 3: Zoom by 8 (highest resolution around gray point)
int	rgb_hist_ zone_en	16 bit value, each bit controlling whether or not the corresponding zone is included in RGB histogram calculation.

Table E-3: **Member Variables of the Input Structure *(Cont'd)***

Type	Name	Function
int	ycc_hist_ zone_en	16 bit value, each bit controlling whether or not the corresponding zone is included in YCC histogram calculation.
int	frame	The input video struct video_in may contain multiple frames. This value identifies which frame in the input video struct will be analyzed.

1. See Figure 4-1 for the definition of zone delineators.

`xilinx_ip_v_stats_v4_00_a_get_default_inputs(&stats_generics, &stats_inputs)` initializes members of the input structure with the Image Statistics GUI default values.

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

Note: Before calling `xilinx_ip_v_stats_v4_00_a_get_default_inputs()` it is advised to initialize the `video_in` structure by loading an image or set of video frames. The horizontal and vertical delineators are set by default such that the input images are split into zones with identical dimensions.

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

```
int xilinx_ip_v_stats_v4_00_a_bitacc_simulate(
     struct xilinx_ip_v_stats_v4_00_a_generics* generics,
     struct xilinx_ip_v_stats_v4_00_a_inputs*   inputs,
     struct xilinx_ip_v_stats_v4_00_a_outputs*  outputs).
```

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

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

```
void xilinx_ip_v_stats_v4_00_a_destroy(
     struct xilinx_ip_v_stats_v4_00_a_inputs *input,
     struct xilinx_ip_v_stats_v4_00_a_outputs *output).
```

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

Image Statistics Input and Output Video Structure

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

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

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

Member Variable	Designation
Frames	Number of video/image frames in the data structure
Rows	Number of rows per frame Pertaining to the image plane with the most rows and columns, such as the luminance channel for yuv data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y,u and v may have different dimensions.
Cols	Number of columns per frame 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 the all frames of the video stream, however different planes, such as y,u and v may have different dimensions.
bits_per_ component	Number of bits per color channel / component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
Mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table E-5.
data	Set of 5 pointers to 3 dimensional arrays containing data for image planes. data is in 16 bit unsigned integer format accessed as data[plane][frame][row][col]

Table E-5: **Named Constants for Video Modes with Corresponding Planes and Representations**

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

1. The Image Statistics core supports the FORMAT_RGB mode

Initializing the Image Statistics Input Video Structure

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

Bitmap Image Files

The header bmp_utils.h declares functions which help access files in Windows Bitmap format (http://en.wikipedia.org/wiki/BMP_file_format). However, this format limits color depth to a maximum of 8 bits per pixel, and operates on images with 3 planes (R,G,B). Therefore, functions

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

operate on arguments type `rgb8_video_struct`, which is defined in rgb_utils.h. Also, both functions support only true-color, non-indexed formats with 24 bits per pixel.

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

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

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

Note: All image / video manipulation utility functions expect both input and output structures initialized, for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data[ ] or r[ ],g[ ],b[ ]) structures already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and throw an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated, the utility functions will create the appropriate container to hold results.

Binary Image/Video Files

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

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

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

Working with video_struct Containers

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

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

Function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table 6. 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]
      }
    }
  }
}
```

Destroy the Video Structure

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

C Model Example Code

An example C file, run_bitacc_cmodel.c, is provided. This demonstrates the steps required to run the model. After following the compilation instructions, you will want to run the example executable.

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

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

During successful execution a text file, containing the statistical information by zones and color channels, is created. These output values are bit-true to the corresponding IP core output values, addressed by `ZONE_ADDR`, and `COLOR_ADDR`. Histogram values are also contained in the resulting text file in a tabulated format.

Compiling with the Image Statistics C Model

Linux

To compile the example code, first ensure that the directory in which the files `libIp_v_stats_v4_00_a_bitacc_cmodel.so` and `libstlport.so.5.1` are located is present in your $LD_LIBRARY_PATH environment variable. These shared libraries are referenced during the compilation and linking process. Then cd into the directory where the header files, the library files and `run_bitacc_cmodel.c` were unpacked. The libraries and header files are referenced during the compilation and linking process.

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

```
gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o run_bitacc_cmodel -L.
-lIp_v_stats_v4_00_a_bitacc_cmodel -Wl,-rpath,.

gcc –m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o run_bitacc_cmodel -L.
-lIp_v_stats_v4_00_a_bitacc_cmodel -Wl,-rpath,.
```

Windows

Precompiled library `v_stats_v4_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. Here an example is presented using Microsoft Visual Studio.

In Visual Studio create a new, empty Win32 Console Application project. As existing items, add:

- `libIpv_stats_v4_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_stats_v4_00_a_bitacc_cmodel.h` header files to “Header Files” folder of the project (optional)

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

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

1. AMBA AXI Protocol Version: 2.0 Specification
2. UG761, *Xilinx AXI Design Reference Guide*
3. Vicent Caselles, Jose-Luis Lisani, Jean-Michel Morel, Guillermo Sapiro: *Shape Preserving Local Histogram Modification*
4. Joung-Youn Kim, Lee-Sup Kim, and Seung-Ho Hwang: *An Advanced Contrast Enhancement Using Partially Overlapped Sub-Block Histogram Equalization*
5. Simone Bianco, Francesca Gasparini and Raimondo Schettini: *Combining Strategies for White Balance*
6. G. Finlayson, M. Drew, and B. Funt, “Diagonal Transform Suffice for Color Constancy” in *Proc. IEEE International Conference on Computer Vision*, Berlin, pp. 164-171, 1993
7. Keith Jack: Video Demystified, 4th Edition, *ISBN 0-7506-7822-4*, pp 15-19

Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

Ordering Information

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

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

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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/11	1.0	Initial Xilinx release.
04/24/12	2.0	Updated core to v4.00a and ISE Design Suite to v14.1.

Notice of Disclaimer