LogiCORE IP Object Segmentation v3.00a

Product Guide

Discontinued IP

PG018 December 18, 2012

XILINX®

XILINX®

Table of Contents

Discontinued IP

Discontinued IP

SECTION IV: APPENDICES

Discontinued IP

SECTION I: SUMMARY

Discontinued IP

Introduction

The Xilinx® LogiCORE™ Intellectual Property (IP) Object Segmentation core provides a hardware-accelerated method for identifying objects of interest within a video stream. The user provides a set of object criteria that describes the objects of interest and the core processes statistical data generated by the Image Characterization LogiCORE IP to "find" the objects of interest. The objects are output as Metadata for subsequent higher level analysis and processing. The core is programmed either directly through the register or by using the supplied software drivers when using the Embedded Development Kit (EDK) pCore configuration.

Features

- User-defined object criteria:
  - Up to eight Feature Combinations (upper and lower thresholds on mean, variance, edge, motion and color information)
  - Up to four Feature Selections (any Boolean combination of the eight feature combinations)
- Detects up to 31 objects per Feature Selection and up to 124 objects per frame
- Operates at all resolutions and frame rates supported by Image Characterization block
- Optional AXI4-Lite control interface
- AXI4-Stream data interfaces

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq ™-7000, Artix ™-7, Virtex®-7, Kintex®-7, Spartan®-6, Virtex®-6
Supported User Interfaces	AXI4-Stream, AXI4-Lite
Resources	See Table 2-1 to Table 2-8.
Provided with Core	
Design Files	ISE: Netlist Vivado: Encrypted RTL
Example Design	Not Provided
Test Bench	Verilog
Constraints File	Not Provided
Simulation Model	Verilog or VHDL Structural, C Model (2)
Supported S/W Driver	N/A
Tested Design Flows(3)	
Design Entry	ISE Design Suite v14.4 Vivado Design Suite v2012.4
Simulation	Mentor Graphics ModelSim, Vivado Simulator
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the release notes for this core.
2. C-Model available on the product page on Xilinx.com at http://www.xilinx.com/products/intellectual-property/EF-DI-VID-OBJ-SEG.htm
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Discontinued IP

Overview

Discontinued IP

The Object Segmentation core is part of a trio of IP Cores (along with Motion Adaptive Noise Reduction and Image Characterization) that enables video analytics systems. These cores provide a hardware-based solution for the computationally-intensive pixel level processing required in video analytics. They produce object Metadata for processing by a system processor or other processing block, eliminating the burden of pixel processing for these components. This approach enables video analytics solutions that can operate at high-definition resolutions and full-frame rates.

In the video analytics system, objects are defined as a rectangular region that matches a set of defined object characteristics (Figure 1-1). The Object Segmentation core plays a key role in the video analytics system. It is responsible for parsing a data structure that describes the characteristics of an image and then "finding" the objects in the image that meet a set of object characteristics.

At a high level, the video analytics system takes a frame of video and subdivides it into a 2-D grid of NxN subdivisions called image blocks. For each image block, a set of statistics is calculated. The Object Segmentation core then compares the statistics of each image block against a set of thresholds that define upper and lower bounds. An image block whose statistics match the set of thresholds is considered an object block. After all of the image blocks have been tested, the core aggregates the object blocks into full objects. Two blocks are aggregated if they are neighbors horizontally, vertically, or diagonally. After all the object blocks have been aggregated into full objects, each object is analyzed to define a box that completely bounds the object. The bounding box defines the object. The final step for the Object Segmentation core is to generate Metadata that consists of a list of all the objects that were found in the image. The Metadata is written to external memory where it can be read by a software application that performs higher-level analysis and processing.

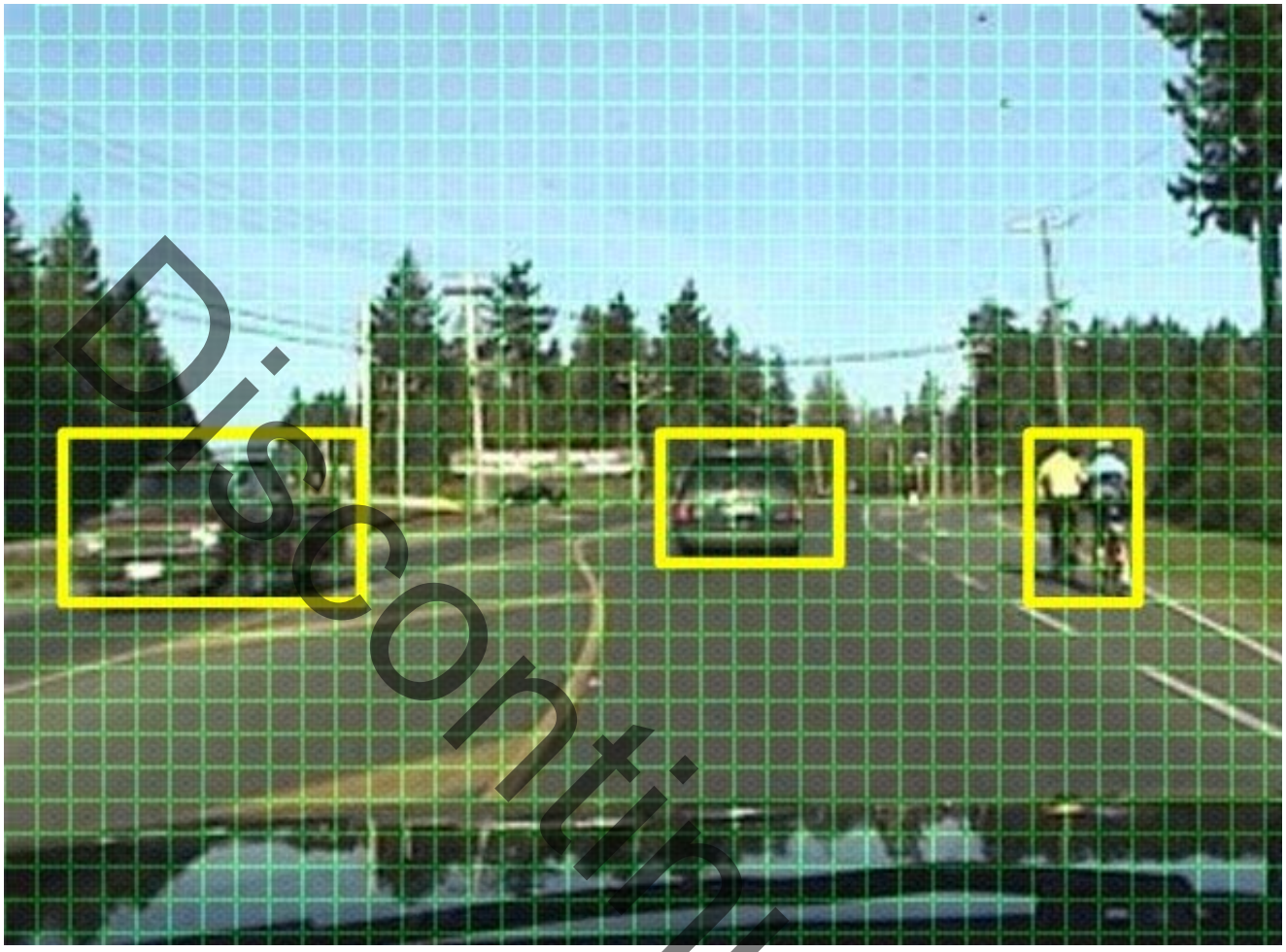

Figure 1-1: **Object Segmentation Image View**

Discontinued IP

Feature Summary

The Object Segmentation core supports up to eight Feature Combinations as described in the Feature Combination section. The user has the option of selecting 1– 8 Feature Combinations when generating the core. Selecting fewer Feature Combinations conserves resources. The Object Segmentation core supports up to four Feature Selects as described in the Feature Select section. The user has the option of selecting 1– 4 Feature Selects when generating the core. Selecting fewer Feature Selects conserves resources.

For each Feature Select that is instantiated, the core can detect up to 31 objects per frame. If four Feature Selects are instantiated, up to 124 objects can be detected for each frame.

The Object Segmentation core is capable of operating at all resolutions, frame rates and block sizes that are supported by the Xilinx® Image Characterization v3.00a IP core.

Applications

- Video Surveillance
- Industrial Control
- Machine Vision
- Automotive
- Other video applications requiring video analytics

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado Design Suite/ISE Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Object Segmentation product web page.

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

Discontinued IP

Product Specification

Discontinued IP

Standards

The Object Segmentation core is compliant with the AXI4-Stream protocol and AXI4-Lite interconnect standards as defined in (UG761) *AXI Reference Guide* [Ref 2].

Performance

The following sections detail the performance characteristics of the Object Segmentation v3.00a core.

Maximum Frequency

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

Latency

The Object Segmentation core outputs a Metadata structure after it has fully processed the input Image Characterization data structure. The Object Segmentation core requires approximately 40 clock cycles to process each set of block statistics in the Image Characterization data structure. Therefore, the latency depends on the number of blocks in the Image Characterization data structure.

Throughput

The Object Segmentation core process the input Image Characterization data structure in two passes. During the first pass, the core outputs one 32-bit word for each set of block statistics in the Image Characterization data structure. During the second pass, the core outputs the Metadata data structure. The size of the Metadata data structure that is output

depends on the number of Feature Selects that are instantiated in the core. For each Feature Select, the Object Segmentation core outputs 192 32-bit words. The core also outputs a data structure header that consists of 32 32-bit words.

Resource Utilization

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

Start the resource count with the resources from the "Base Core" which includes the resources for one Feature Combination and one Feature Select. If using more than one Feature Combination, multiply the resources in the Each additional Feature Combination row by the number of extra Feature Combinations and add the results to the resource count. If using more than one Feature Select, multiply the resources in the Each additional Feature Select row by the number of extra Feature Selects and add the results to the resource count.

Resource Utilization using Vivado Design Environment

The information presented in Table 2-1 through Table 2-3 is a guide to the resource utilization and maximum clock frequency of the Image Characterization core for Virtex-7, Kintex-7, Artix-7 and Zynq-7000 FPGA families using Vivado tools.

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

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2494	3214	2\3	6	250
Each additional Feature Combination	185	150	0\0	0	
Each additional Feature Select	712	680	0\1	0	

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

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2493	3214	2\3	6	250
Each additional Feature Combination	185	150	0\0	0	
Each additional Feature Select	712	680	0\1	0	

Discontinued IP

Table 2-3: **Artix-7 (and Zynq-7000 Devices with Kintex Based Fabric) Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2490	3214	2\3	6	250
Each additional Feature Combination	185	150	0\0	0	
Each additional Feature Select	712	680	0\1	0	

Resource Utilization using ISE Design Environment

The information presented in Table 2-4 through Table 2-8 is a guide to the resource utilization and maximum clock frequency of the Image Characterization core for Virtex-7, Kintex-7, Artix-7, Zynq-7000, Virtex-6, and Spartan-6 FPGA families using ISE tools.

Table 2-4: **Virtex-7 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2865	3198	2\3	4	250
Each additional Feature Combination	202	149	0\0	0	
Each additional Feature Select	575	558	0\1	0	

Table 2-5: **Kintex-7 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2852	3198	2\3	4	250
Each additional Feature Combination	199	149	0\0	0	
Each additional Feature Select	583	559	0\1	0	

Table 2-6: **Artix-7 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2818	3198	2\3	4	150
Each additional Feature Combination	245	149	0\0	0	
Each additional Feature Select	578	558	0\1	0	

Discontinued IP

Table 2-7: **Virtex-6 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	2930	3200	2\3	4	250
Each additional Feature Combination	202	149	0\0	0	
Each additional Feature Select	595	556	0\1	0	

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

Feature	LUTs	FFs	Block RAMs (36/18)	DSP	MHz
Base Core (Feature Combination = 1, Feature Select = 1)	3043	3204	2\3	4	156
Each additional Feature Combination	218	151	0\0	0	
Each additional Feature Select	637	557	0\1	0	

Discontinued IP

Core Interfaces and Register Space

Port Descriptions

The Object Segmentation core uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the core. Some signals are optional and not present for all configurations of the core. The `INTC_IF` interface is present only when the core is configured via the GUI with the INTC interface enabled.

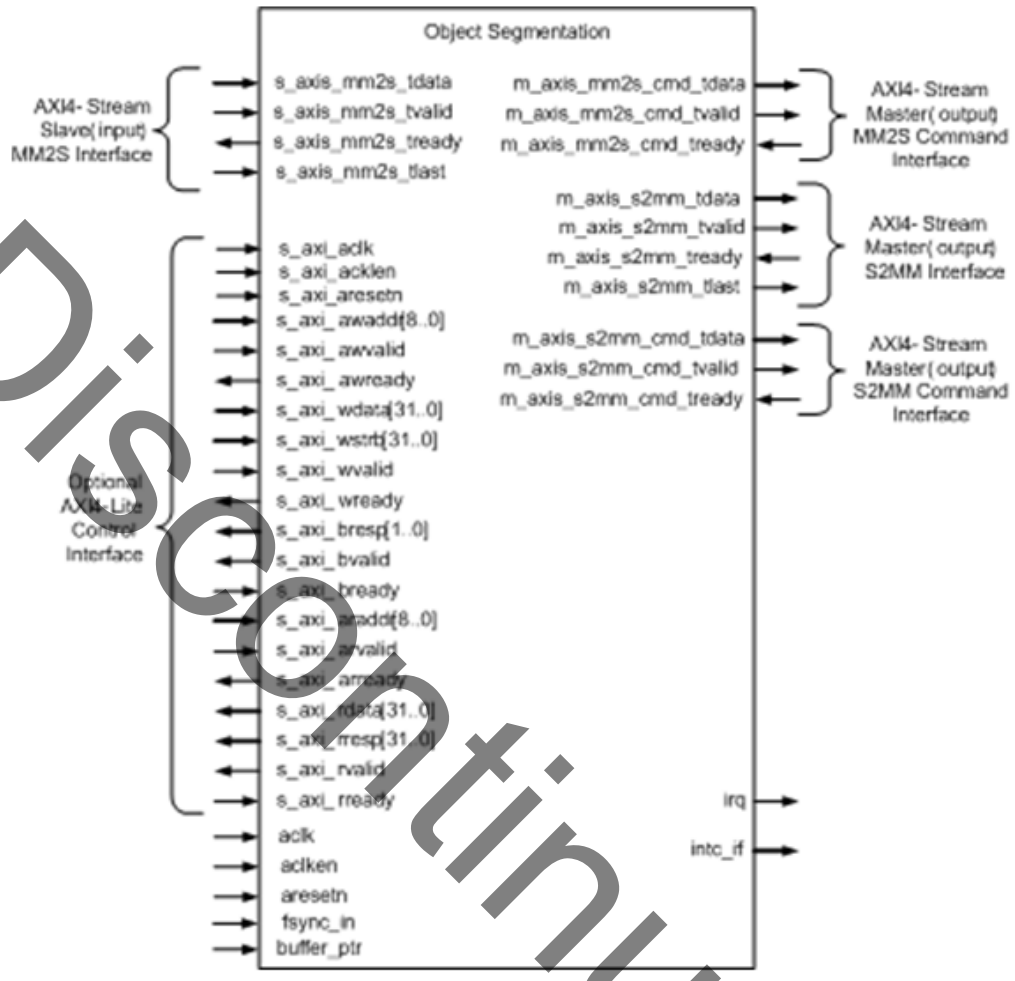

Figure 2-1: **Object Segmentation Top-Level Signaling Interface**

Discontinued IP

Common Interface Signals

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

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

Signal Name	Direction	Width	Description
ACLK	In	1	Video Core Clock
ACLKEN	In	1	Video Core Active High Clock Enable
ARESETn	In	1	Video Core Active Low Synchronous Reset
INTC_IF	Out	9	Optional External Interrupt Controller Interface. Available only when `INTC_IF` is selected on GUI.
IRQ	Out	1	Interrupt Request Pin.

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

Signal Name	Direction	Width	Description
fsync_in	In	1	Input that denotes the beginning of frame processing. Typically used to synchronize with the Image Characterization Core
buffer_ptr	Out	1	Input that specifies which output buffer the Image Characterization core is actively writing to. Typically used to synchronize with the Image Characterization Core.

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

ACLK

The AXI4-Stream interface must be synchronous to the core clock signal ACLK. All AXI4-Stream interface input signals are sampled on the rising edge of ACLK. All AXI4-Stream output signal changes occur after the rising edge of ACLK. The AXI4-Lite interface is unaffected by the ACLK signal.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to AXI4-Stream interfaces. Setting ACLKEN low (de-asserted) halts the operation of the core despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is de-asserted, core inputs are not sampled, except ARESETn, which supersedes ACLKEN. The AXI4-Lite interface is unaffected by the ACLKEN signal.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to only AXI4-Stream interfaces. ARESETn supersedes ACLKEN, and when set to 0, the core resets at the next rising edge of ACLK even if ACLKEN is de-asserted. The ARESETn signal must be synchronous to the ACLK and must be held low for a minimum of 32 clock cycles of the slowest clock. The AXI4-Lite interface is unaffected by the ARESETn signal.

Fsync_In

The fsync_in signal is a signal that denotes the beginning of the processing of a new frame of data. This signal is typically driven by the fsync_out signal of the Image Characterization core and is used for synchronization purposes.

Discontinued IP

Buffer_Ptr

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

Data Interface

The Object Segmentation core receives data using AXI4-Stream interface as defined in the *AXI Reference Guide* (UG761) [Ref 2].

AXI4-Stream Signal Names and Descriptions

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

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

Signal Name	Direction	Width	Description
s_axis_mm2s_tdata	In	24	Input Data
s_axis_mm2s_tvalid	In	1	Input Valid
s_axis_mm2s_tready	Out	1	Input Ready
s_axis_mm2s_tlast	In	1	Input End Of Line
m_axis_mm2s_cmd_tdata	Out	72	Input Command Data
m_axis_mm2s_cmd_tvalid	Out	1	Input Command Valid
m_axis_mm2s_cmd_tready	In	1	Input Command Ready
m_axis_s2mm_tdata	Out	32	Output Data
m_axis_s2mm_tvalid	Out	1	Output Valid
m_axis_s2mm_tready	In	1	Output Ready
m_axis_s2mm_tlast	Out	1	Output End of Line
m_axis_s2mm_cmd_tdata	Out	72	Output Command Data
m_axis_s2mm_cmd_tvalid	Out	1	Output Command Valid
m_axis_s2mm_cmd_tready	In	1	Output Command Ready

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, ACLKEN, and ARESETn are high at the rising edge of ACLK, as seen in Figure 2-2. During valid transfers, DATA only carries active video data.

Discontinued IP

Guidelines on Driving s_axis_video_tvalid

Once `s_axis_mm2s_tvalid` is asserted, no interface signals (except the Object Segmentation core driving `s_axis_mm2s_tready`) may change values until the transaction completes (`s_axis_mm2s_tready`, `s_axis_mm2s_tvalid`, `ACLKEN` high on the rising edge of `ACLK`). Once asserted, `s_axis_mm2s_tvalid` may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, `s_axis_mm2s_tvalid` can either be de-asserted or remain asserted to initiate a new transfer.

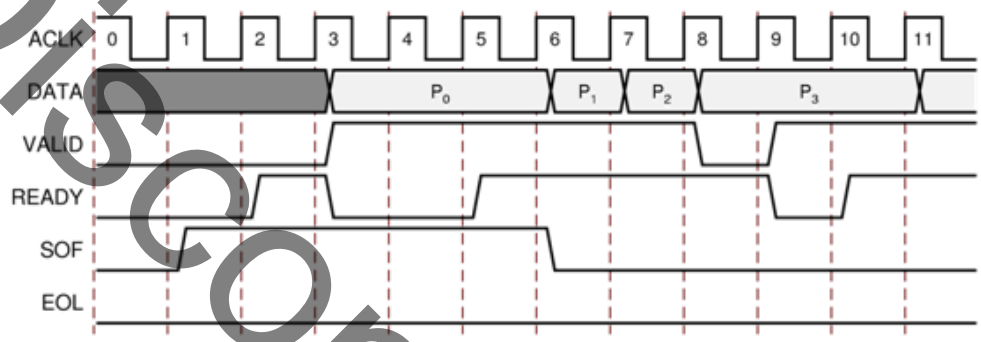

Figure 2-2: **Example of READY/VALID Handshake**

TLAST Signals - s_axis_mm2s_tlast, m_axis_s2mm_tlast

The `TLAST` pulse is 1 valid transaction wide, and must coincide with the last word of a transfer, as seen in Figure 2-3. All transactions on the `m_axis_mm2s_cmd` and `m_axis_s2mm_cmd` interfaces consist of one data word passed from the Master to the Slave. As a result, there is no `TLAST` signal for that interface.

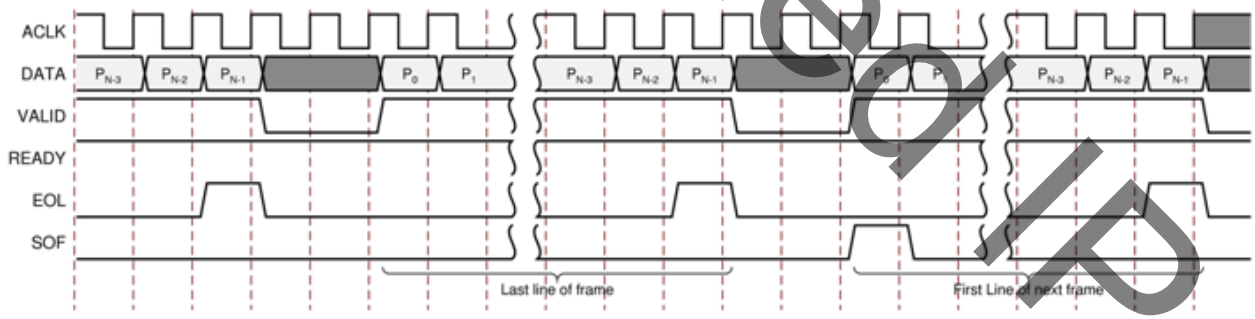

Figure 2-3: **Use of EOL and SOF Signals**

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 through AXI4-Lite interface to an AXI4-Lite master.

Discontinued IP

AXI4-Lite Interface

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

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

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

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

Discontinued IP

S_AXI_ACLK

The AXI4-Lite interface must be synchronous to the S_AXI_ACLK clock signal. The AXI4-Lite interface input signals are sampled on the rising edge of ACLK. The AXI4-Lite

output signal changes occur after the rising edge of ACLK. The AXI4-Stream interfaces signals are not affected by the S_AXI_ACLK.

S_AXI_ACLKEN

The S_AXI_ACLKEN pin is an active-high, synchronous clock-enable input for the AXI4-Lite interface. Setting S_AXI_ACLKEN low (de-asserted) halts the operation of the AXI4-Lite interface despite rising edges on the S_AXI_ACLK pin. AXI4-Lite interface states are maintained, and AXI4-Lite interface output signal levels are held until S_AXI_ACLKEN is asserted again. When S_AXI_ACLKEN is de-asserted, AXI4-Lite interface inputs are not sampled, except S_AXI_ARESETn, which supersedes S_AXI_ACLKEN. The AXI4-Stream interfaces signals are not affected by the S_AXI_ACLKEN.

S_AXI_ARESETn

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

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The core has 16 core-specific registers for setting the buffer addresses, scaling values and the color selects.

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 30: FRAME_SYNC_RESET (1: reset) Bit 31: SW_RESET (1: reset)
0x0004	STATUS	R/W	No	0	Bit 0: PROC_STARTED Bit 1: EOF Bit 16: SLAVE_ERROR
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to STATUS bits

Discontinued IP

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	0x0300a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR
0x0100	Image_Characterization_Start_Address_0	R/W	Yes	Specified via GUI	31-0: Starting Address for Image Characterization Buffer 0
0x0104	Image_Characterization_Start_Address_1	R/W	Yes	Specified via GUI	31-0: Starting Address for Image Characterization Buffer 1
0x0108	MetaData_Start_Address_0	R/W	Yes	Specified via GUI	31-0: Starting Address for MetaData Buffer 0
0x010C	MetaData_Start_Address_1	R/W	Yes	Specified via GUI	31-0: Starting Address for MetaData Buffer 1
0x0110	Label_Data_Start_Address	R/W	Yes	Specified via GUI	31-0: Starting Address for the Label Data Buffer
0x0114	Buffer_Control	R/W	Yes	0x0	Bit 0: MetaData Buffer Select, 0=Write to MetaData Buffer 0, 1=Wrire to MetaData Buffer 1 Bit 1: IC Buffer Pointer Mode, 0=Use Buffer_Ptr to select IC buffer to read, 1=Independently toggle between buffers on each frame.
0x0118	Feature_Select_Write_Bank_Addr	R/W	No	0x0	2-0: Address of the Feature Select Bank that is to be written to
0x011C	Feature_Select_Write_Bank_Data	R/W	No	0x0	3-0: Feature Select Bank Data
0x0120	Feature_Select_Active_Bank_Addr	R/W	Yes	Specified via GUI	2-0: Address of the Feature Select Bank that is be used by the core
0x0124	Feature_Combination_Write_Bank_Addr	R/W	No	0x0	Bit 3: Address of the Feature Combination Bank to be written to 2-0: Address of the Feature Combination Table to be written to
0x0128	Feature_Combination_Write_Bank_Data	R/W	No	0x0	31-0: Feature Combination Data
0x012C	Feature_Combination_Active_Bank_Addr	R/W	Yes	Specified via GUI	Bit 0: Address of the Feature Combination Bank that is to be used by the core
0x0130	Number of Horizontal Blocks	R/W	Yes	Specified via GUI	9-0: Number of Blocks Wide

Discontinued IP

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

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0134	Number of Vertical Blocks	R/W	Yes	Specified via GUI	9-0: Number of Blocks High
0x0138	Total Number of Blocks	R/W	Yes	Specified via GUI	19-0: Total Number of Blocks (Horizontal x Vertical)
0x013C	Block Size	R/W	Yes	Specified via GUI	6-0: Block Size (4, 8, 16, 32, or 64)

Discontinued

CONTROL (0x0000) Register

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

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

Bits 30 and 31 of the CONTROL register, FRAME_SYNC_RESET and SW_RESET facilitate software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Setting FRAME_SYNC_RESET to 1 will reset the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset via the ARESETn pin.

STATUS (0x0004) Register

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

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

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

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

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

ERROR (0x0008) Register

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

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

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

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

IRQ_ENABLE (0x000C) Register

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

Discontinued IP

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware. See Table 2-12 for details.

Image Characterization Start Address 0 (0x0100) Register

The `Image Characterization Start Address 0` register holds the start address for the first memory buffer in external memory where the Image Characterization data will be read. The value in this register is used when calculating addresses that are passed to the AXI DataMover over the AXI4 MM2S Command Interface.

Image Characterization Start Address 1 (0x0104) Register

The `Image Characterization Start Address 1` register holds the start address for the second memory buffer in external memory where the Image Characterization data will be read. The value in this register is used when calculating addresses that are passed to the AXI DataMover over the AXI4 MM2S Command Interface.

MetaData Start Address 0 (0x0108) Register

The `MetaData Start Address 0` register holds the start address for the first memory buffer in external memory where the MetaData will be written. The value in this register is used when calculating addresses that are passed to the AXI DataMover over the AXI4 S2MM Command Interface.

MetaData Start Address 1 (0x010C) Register

The `MetaData Start Address 1` register holds the start address for the second memory buffer in external memory where the MetaData will be written. The value in this register is used when calculating addresses that are passed to the AXI DataMover over the AXI4 S2MM Command Interface.

Label Data Start Address (0x0110) Register

The `Label Data Start Address` register holds the start address for the memory buffer in external memory where the Label Data will be stored. The value in this register is used when calculating addresses that are passed to the AXI DataMovers over the AXI4 Command Interfaces.

Buffer Control (0x0114) Register

Bit 0 of the `Buffer Control` register, `MetaData Buf Sel`, controls which of the MetaData buffers is actively written to by the Object Segmentation core. A value of '0'

Discontinued IP

means that the core will write to the MetaData Buffer 0. A value of '1' means that the core will write to MetaData Buffer 1. The core will repeatedly write to the specified buffer for each frame. When the processor is ready to read the current MetaData buffer, the MetaData Buf Sel bit should first be toggled so that the Object Segmentation core will switch to the other buffer and overwrite the current MetaData.

Bit 1 of the `Buffer Control` register, `IC Ptr Mode`, is used to specify the mode that the Object Segmentation core will use to determine which Image Characterization buffer to read during the current frame period. A value of '0' means that the core will use the buffer_ptr input signal to determine which buffer to read. When buffer_ptr = '1', the core will read from the Image Characterization Buffer 0. When buffer_ptr = '0', the core will read from the Image Characterization Buffer 1. A value of '1' in the IC Ptr Mode bit means that the Object Segmentation core will independently toggle between the two Image Characterization buffers.

Feature Select Write Bank Addr (0x0118) Register

The `Feature Select Write Bank Addr` register holds the address of the Feature Select Bank that will be written to by the Feature Select Write Bank Data register. There are 8 Feature Select Banks that can be written to. This register is not double buffered. Writing to the register immediately changes the value.

Feature Select Write Bank Data (0x011C) Register

The `Feature Select Write Bank Data` register is used to load data into the Feature Select Banks. A Feature Select bank consists of 256 entries. Loading data into a Feature Select Bank requires writing 256 consecutive values to the Feature Select Write Bank Data register. The first value is stored to entry 0; the 256th value is stored to entry 255 of the bank. This register is not double buffered. The user can dynamically load new data while the core is processing data.

IMPORTANT: *Care should be taken to avoid loading data to an active bank while it is being used to process data.*

Feature Select Active Bank Addr (0x0120) Register

The `Feature Select Active Bank Addr` register holds the address of the Feature Select Bank that will be used when the core processes data. There are 8 Feature Select Banks from which to select. Only one bank can be active during a frame period.

Feature Combination Write Bank Addr (0x0124) Register

Bit 3 of the `Feature Combination Write Bank Addr` register controls which bank is written to by the Feature Combination Write Bank Data register. There are two Feature Combination banks to chose from.

Discontinued IP

Bits 2-0 of the `Feature Combination Write Bank Addr` register control which table in the Feature Combination bank is written to by the Feature Combination Write Bank Data register. There are up to 8 Feature Combination Tables per Bank to select from.

This register is not double buffered. Writing to the register immediately changes the value.

Feature Combination Write Bank Data (0x0128) Register

The `Feature Combination Write Bank Data` register is used to load data into the Feature Combination Bank Tables. A Feature Combination Bank Table consists of 40 entries. Loading data into a Feature Combination Bank Table requires writing 40 consecutive values to the Feature Combination Write Bank Data register. The first value is stored to table entry 0; the 40th value is stored to table entry 39. This register is not double buffered. The user can dynamically load new data while the core is processing data. Care should be taken to avoid loading data to an active table while it is being used to process data.

Feature Combination Active Bank Addr (0x012C) Register

The `Feature Combination Active Bank Addr` register holds the address of the Feature Combination Bank that will be used when the core processes data. There are 2 Feature Select Banks from which to select. Only one bank can be active during a frame period.

Number of Horizontal Blocks (0x0130) Register

The `Number of Horizontal Blocks` register encodes the number of Horizontal blocks in each row.

Number of Vertical Blocks (0x0134) Register

The `Number of Vertical Blocks` register encodes the number of Vertical blocks in each column.

Total Number of Blocks (0x0138) Register

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

Block Size (0x013C) Register

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

Discontinued IP

The Interrupt Subsystem

STATUS register bits can trigger interrupts so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system. Irrespective of whether the AXI4-Lite control interface is present or not, the Object Segmentation core detects AXI4-Stream framing errors, as well as the beginning and the end of frame processing.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (IRQ) is present. Events associated with bits of the STATUS register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (IRQ_ENABLE) are set. Once set by the corresponding event, bits of the STATUS register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

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

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

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

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	Reserved
3	Reserved
4	Slave Error
5	EOL Early
6	EOL Late
7	Reserved
8	Reserved

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

Discontinued IP

Designing with the Core

Discontinued IP

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

Architecture

A high-level view of the Object Segmentation core is shown in Figure 3-1. The core uses an AXI4-Stream interface to transfer data between the core and buffers in external memory. The Object Segmentation core uses a two-phase architecture to find and segment objects in a video frame.

In the first phase, the core inputs the Image Characterization data structure and compares it against the Feature Combination threshold values. The results of that processing are combined to form the Feature Select results. The Feature Select results are processed to create labeled regions within the data structure. This label data is written to external memory and the list of labeled regions is processed to aggregate neighboring labels into an object. After this is done, the second phase of processing begins by reading the label data from external memory and remapping the labels into objects. After an object is defined, the statistics of the object are calculated and written out as Metadata.

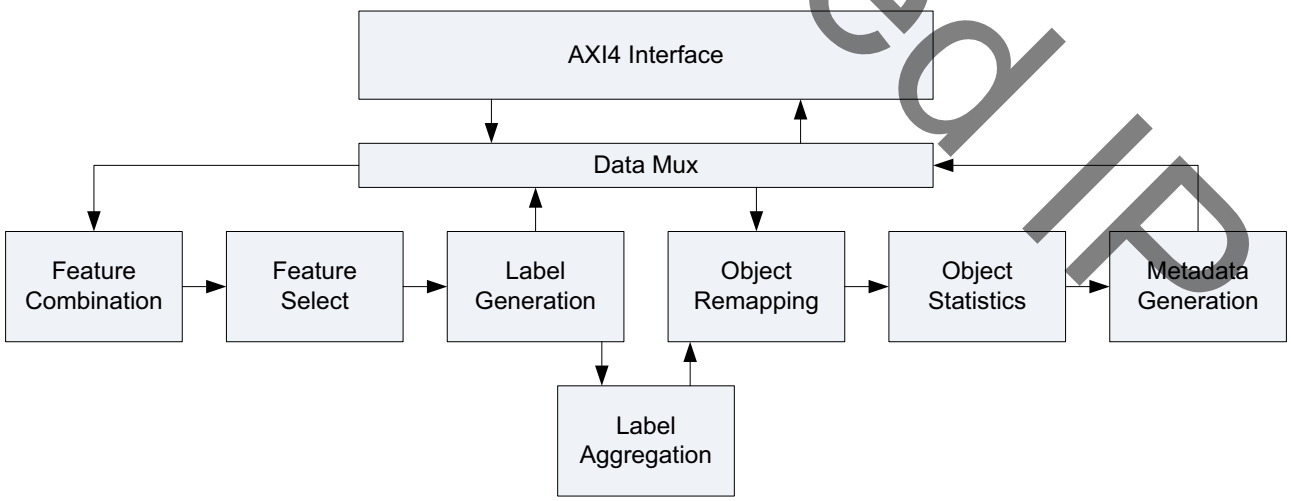

Figure 3-1: **Object Segmentation Block Diagram**

Feature Combination

A Feature Combination (FC) is defined in the Feature Combination Threshold Data Structure section. The Feature Combination Data Threshold Structure consists of a set of threshold values with a lower and an upper bound. The Feature Combination Lower Global Thresholds and Upper Global Thresholds in Table 3-2 match the Image Characterization IP core Global Statistics output as shown in Table 3-6. Additionally, the Feature Combination Lower Block Thresholds and Upper Block Thresholds in Table 3-2 match the Image Characterization IP core output Block Statistics shown in Table 3-7. A Feature Combination unit must be properly initialized with a Feature Combination data structure before it can begin processing.

A Feature Combination unit is implemented as a set of comparators. It compares the Image Characterization data input against the Feature Combination data structure that is loaded by the user. Each value in the Image Characterization data structure is compared against its corresponding threshold values in the Feature Combination data structure. A value passes the comparison if it meets the following criteria:

FC Lower Threshold < Image Characterization value < FC Upper Threshold

The first portion of the Image Characterization data that is read is the Global Statistics. As the Global Statistics are read in they are compared against the Feature Combination Global Thresholds. If any of the Global Statistics fail to match the threshold ranges, the entire frame is considered invalid and no objects will be found in the image. If all of the Global Statistics match the Global Statistics Thresholds, then processing advances to the Block data processing.

Next the Image Characterization Block Data is tested against the Feature Combination Block Thresholds. For each Image Characterization block a 1-bit result is calculated. If all of the statistics for an Image Characterization Block match the Feature Combination Block Thresholds, the block is given a value of '1'. If one or more of the statistics for an Image Characterization Block fails to match the Feature Combination Block Thresholds, the block is given a value of '0'.

Up to eight separate Feature Combination units are supported by the Object Segmentation core. Each Feature Combination unit is a separate entity and must be properly initialized with a unique Feature Combination data structure. Each Feature Combination unit generates a 1-bit result for each block in the Image Characterization data structure. These eight 1-bit values are passed to the Feature Select Generation block for further processing. If less than eight Feature Combination units are instantiated, the results are padded with '0's in the MSB to make 8-bits.

Discontinued IP

Feature Select

The Feature Select block takes the eight 1-bit values from the Feature Combination block and transforms them into a Feature Select. A Feature Select is defined as any logical equation using the Feature Combination results as terms in the equation.

A Feature Select is implemented as a 1-bit x 256-entry look-up table. The 8 bits of Feature Combination data are used as an address to this look-up table. The look-up table is used as a truth table. The initialization of the look-up table determines the logical equation that defines the Feature Select. See Feature Select Data Structure for more details.

The Feature Select block supports up to four Feature Selects. Each Feature Select generates a 1-bit result for each block in the original Image Characterization data structure. These four 1-bit results are passed on to the Label Generation block for further processing.

Label Generation

The Label Generation block is used to aggregate neighboring blocks with Feature Select values of '1'. As each block is examined, if it is a value of '0' then it is ignored because it did not test positive for Feature Select. If a block has a value of '1', it is assigned a label value. If the block has a neighbor to its left, left upper diagonal, upper, or right upper diagonal that already has a label value then the new block is assigned the same label value. If none of the blocks neighbors has a label value then the block is assigned a new label value. Figure 3-2 shows an example Feature Select result for an 8x8 block image. The blocks with a value of '1' passed the Feature Select processing. Empty blocks did not pass the Feature Select processing. Figure 3-3 shows the results of the Label Generation processing. The Feature Select data is aggregated into four sets of labels. Labels 1, 3 and 4 are neighbors.

Discontinued IP

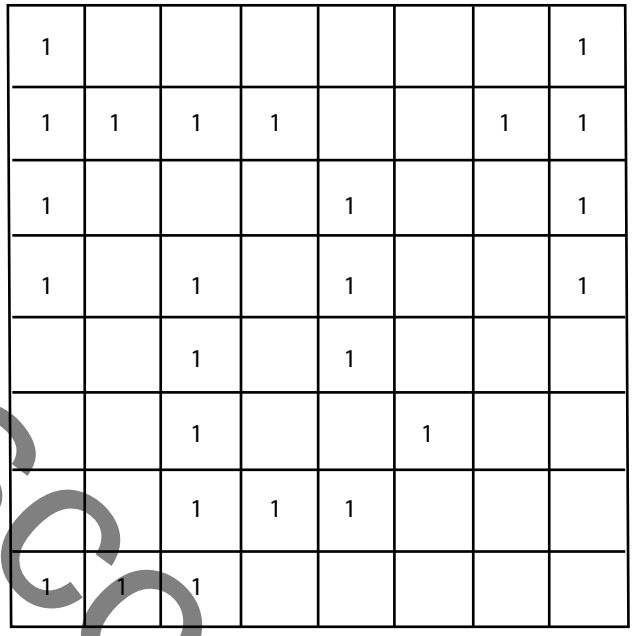

Figure 3-2: **Feature Select Results for an 8x8 Frame**

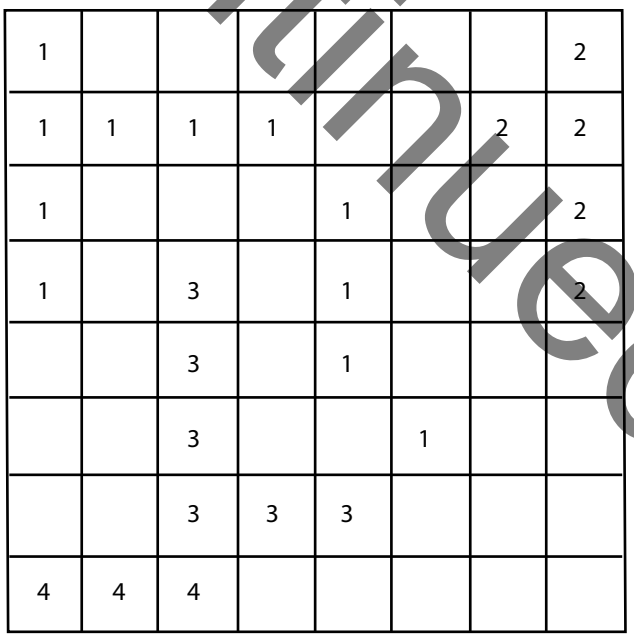

Figure 3-3: **Label Data for an 8x8 Frame**

There is a separate Label Generation circuit for each Feature Select. Label Generation circuits run in parallel with each other. The Label data is written to an external memory buffer to complete the first pass of processing.

Discontinued IP

Label Aggregation

The Label Aggregation block is active between the end of the first pass and the start of the second pass of processing. The Label Aggregation is responsible for finding labels that are neighbors and aggregating them into an object. The Label Generation block reports out a list of labels that are neighbors. The Label Aggregation block recursively searches the list for all labels that are connected and adds them to a list of new object values. This list of object values is used during the second pass processing to remap labels into objects. There is a separate Label Aggregation circuit for each Label Generation circuit. The Object Segmentation core supports up to four Label Aggregation circuits.

Continuing with the example used in the previous section, the Label Generation block reports that there are four labels (1, 2, 3 and 4) and that 3 connects to 1 and 4 connects to 3. The Label Aggregation block processes this list and remaps the labels 1, 3 and 4 to object A and label 2 to object B. This new object mapping is passed to the Object Remapping block.

Object Remapping

After the Label Aggregation block is finished consolidating labels into objects, the Object Remapping block uses the object mapping list to assign new object values to the label data. As the label data is read from memory, the Object Remapping block looks at the label assigned to a block and remaps the block to a new object value. Figure 3-4 and Figure 3-5 illustrate the remapping process for the example use in the previous sections.

1							2
1	1	1	1			2	2
1				1			2
1		3		1			2
		3		1			
		3			1		
		3	3	3			
4	4	4					

Figure 3-4: **Label Data for an 8x8 Frame**

Discontinued IP

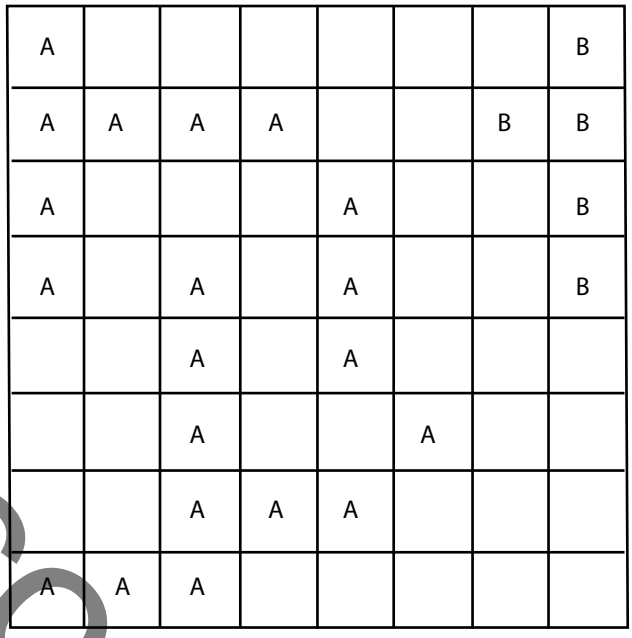

Figure 3-5: **Object Data for an 8x8 Frame**

There is a separate Object Remapping circuit for each instantiated Feature Select for up to four independent circuits.

Object Statistics

The new object data is passed along to the Object Statistics block. The Object Statistics block keeps track of the size of each object and calculates the coordinates of a rectangle that would completely bound the object. The Object Statistics block also counts the number of blocks that make up an object. For Figure 3-5, the Object Statistics block would count 20 blocks for object A and 5 blocks for object B.

There is a separate Object Statistics circuit for each instantiated Feature Select.

Metadata Generation

The calculated object statistics are passed from the Object Statistics block to the Metadata Generation block. This block is responsible for taking the object statistics for each of the Object Statistics circuits and formatting the data into the Object Segmentation Metadata that is written to external memory. Metadata is written for each instantiated Feature Select.

Discontinued IP

Data Structures

The Object Segmentation core uses several different data structures. The Feature Combination Threshold data structure and the Feature Select data structure define the data that is used to initialize the Object Segmentation core prior to processing data. The Image Characterization data structure defines the format of the input data that the core processes. The Object Segmentation Metadata data structure defines the format of the output data that the core produces for each frame of Image Characterization data.

Feature Combination Threshold Data Structure

The Object Segmentation core supports up to eight Feature Combination units. Each Feature Combination unit requires the input of a set of threshold values. The set of threshold values is defined by the Feature Combination Threshold data structure. The data structure consists of a lower threshold and an upper threshold for each of the global and block statistics in the Image Characterization Data Structure that is defined in the Image Characterization Data Structure section.

The global and block mean ("_mean") statistics are 8-bit values and the thresholds can be any value from 0 to 255. The global and block variance ("_var") statistics are 16-bit values and the thresholds can have any value from 0 to 65535. Each block color_select value represents the number of pixels in the block that matched the specified color range for that color_select in the Image Characterization core. The color_select's value range is determined by the block size used to produce the Image Characterization data, and by the video format that is processed. Table 3-1 shows the possible value ranges for color_selects.

Discontinued IP

Table 3-1: **Value Ranges for Image Characterization Color_Selects**

Block Size	Video Format 4:2:2	Video Format 4:2:0
64x64	0 to 2048	0 to 1024
32x32	0 to 512	0 to 256
16x16	0 to 128	0 to 64
8x8	0 to 32	0 to 16
4x4	0 to 8	0 to 4

Each Feature Combination unit must be properly initialized before it can be used to process input data. Table 3-2 shows the format of the Feature Combination Threshold data structure and the order in which each element should be loaded. Each Feature Combination unit is loaded independently. There are two memory banks associated with each Feature Combination unit, with the active bank specified by a register value. This configuration allows the user to load two different Feature Combination sets and then switch between the two in real-time by changing the active bank register selection. Changes to the active bank register are acted upon at the beginning of each frame. This configuration also allows the user to calculate a new Feature Combination set, load the new set into the inactive Feature Combination bank and then make the new set the active bank in real-time. See Feature Combination Bank Programming for more details.

Table 3-2: **Feature Combination Threshold Data Structure**

Line #	Byte 3	Byte 2	Byte 1	Byte 0
	Lower Global Threshold			
0x0	Low_Freq_Mean	V_mean	U_Mean	Y_Mean
0x1	Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
0x2	U_Var		Y_Var	
0x3	Low_Freq_Var		V_Var	
0x4	Edge_Var		High_Freq_Var	
0x5	Saturation_Var		Motion_Var	
	Lower Block Threshold			
0x6	Low_Freq_Mean	V_mean	U_Mean	Y_Mean
0x7	Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
0x8	U_Var		Y_Var	
0x9	Low_Freq_Var		V_Var	
0xA	Edge_Var		High_Freq_Var	
0xB	Saturation_Var		Motion_Var	
0xC	Color_Sel_2		Color_Sel_1	
0xD	Color_Sel_4		Color_Sel_3	
0xE	Color_Sel_6		Color_Sel_5	
0xF	Color_Sel_8		Color_Sel_7	

Discontinued IP

Table 3-2: **Feature Combination Threshold Data Structure *(Cont'd)***

0x10	Reserved			
0x11	Reserved			
0x12	Reserved			
0x13	Reserved			
	Upper Global Threshold			
0x14	Low_Freq_Mean	V_mean	U_Mean	Y_Mean
0x15	Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
0x16	U_Var		Y_Var	
0x17	Low_Freq_Var		V_Var	
0x18	Edge_Var		High_Freq_Var	
0x19	Saturation_Var		Motion_Var	
	Upper Block Threshold			
0x1A	Low_Freq_Mean	V_mean	U_Mean	Y_Mean
0x1B	Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
0x1C	U_Var		Y_Var	
0x1D	Low_Freq_Var		V_Var	
0x1E	Edge_Var		High_Freq_Var	
0x1F	Saturation_Var		Motion_Var	
0x20	Color_Sel_2		Color_Sel_1	
0x21	Color_Sel_4		Color_Sel_3	
0x22	Color_Sel_6		Color_Sel_5	
0x23	Color_Sel_8		Color_Sel_7	
0x24	Reserved			
0x25	Reserved			
0x26	Reserved			
0x27	Reserved			

Discontinued IP

Feature Select Data Structure

The Object Segmentation core supports up to four Feature Select units. A Feature Select unit takes the 1-bit results from each of the Feature Combination units and applies a logical expression to them. Each Feature Combination unit is represented as a separate entity in the logical expression. All logical expressions are supported.

A Feature Select unit is implemented as a 1-bit x 256 entry RAM. The eight Feature Combination units are used as address bits to the RAM. Feature Combination 1 is the Least Significant Bit (LSB) and Feature Combination 8 is the Most Significant Bit (MSB) of the

address. The RAM creates a look-up table that can be used as a truth-table and therefore allows the implementation of any logical expression of the eight Feature Combinations.

Because up to four Feature Selects are supported, each is implemented as one bit in a 4-bit x 256 entry RAM. Feature Select 1 is the LSB and Feature Select 4 is the msb of the Random Access Memory (RAM) output data as illustrated in Table 3-3. The Object Segmentation core supports eight banks of Feature Select data. The active bank is selected through an active bank register. This configuration allows the user to load multiple banks of Feature Select data and then switch between the banks in real-time. Changes to the active bank selection are processed at the beginning of a frame. Each Feature Select bank is loaded independently which allows the loading of new Feature Select data at any time. See Feature Select Bank Programming for more details.

Table 3-3: **Feature Select Data Structure**

Feature Combinations(8:1)	Bit 3	Bit 2	Bit 1	Bit 0
0x00 (00000000)	FS4_0	FS3_0	FS2_0	FS1_0
0x01 (00000001)	FS4_1	FS3_1	FS2_1	FS1_1
0x02 (00000010)	FS4_2	FS3_2	FS2_2	FS1_2
...	...	...	...	...
0xFD (11111101)	FS4_253	FS3_253	FS2_253	FS1_253
0xFE (11111110)	FS4_254	FS3_254	FS2_254	FS1_254
0xFF (11111111)	FS4_255	FS3_255	FS2_255	FS1_255

Image Characterization Data Structure

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

- Frame Header (Table 3-5)
- Global Statistics and Histograms (Table 3-6)
- Block Statistics (Table 3-7)

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

The values in the data structure use these bit widths:

- Mean ("_mean"): 8-bits

Discontinued IP

- Variance ("_var"): 16-bits
- Histogram: 32-bits (21-bits actual)
- Color_Select: 16-bits (12-bits actual)
- PAD: 32-bits (0x0000)

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

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

Table 3-5: **Image Characterization Data Structure Frame Header**

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

Table 3-6: **Image Characterization Data Structure Global Statistics**

Byte 3	Byte 2	Byte 1	Byte 0
Low_Freq_Mean	V_mean	U_Mean	Y_Mean
Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
U_Var		Y_Var	
Low_Freq_Var		V_Var	
Edge_Var		High_Freq_Var	
Saturation_Var		Motion_Var	
PAD (x26)			
Y_Histogram (x256)			
U_Histogram (x256)			
V_Histogram (x256)			
Hue_Histogram (x256)			

Discontinued IP

Table 3-7: **Image Characterization Data Structure Block Statistics**

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

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

Metadata Data Structure

The Metadata data structure contains all of the data calculated by the Object Segmentation core to describe the objects found in the current Image Characterization data structure. The Metadata data structure is a list of up to 31 objects described for up to four Feature Selects resulting to a total of 124 objects that can be found for each frame.

The Metadata data structure begins with a header that is specified in Table 3-9. The Structure_Valid entry specifies whether the entire frame has been written and is valid. It holds a value of 0x00000001 if the data structure is incomplete. It holds a value of 0xFFFFFFFF if the data structure is complete. The Frame Index is a unique identifier for each frame. The values for the number of objects correspond to the number of objects in the Metadata data structure for the specified value. The data structure header also includes the Global Image Characterization statistics that were copied directly from the Image Characterization data structure.

After the Metadata header, the objects for FS1 are provided followed by the objects for FS2, FS3 and FS4. The Object Segmentation core only writes out metadata for the Feature Selects that are instantiated in the core. If only two Feature Selects are instantiated, then only the metadata for FS1 and FS2 is written.

The metadata associated with an object is defined in Table 3-10, and consists of the following information: FS#, object number, X/Y start/stop values, X/Y centroid values, object

Discontinued

density and object identifier. The FS# corresponds to the Feature Select that the object belongs to. The object number is the number of the object in the list of objects for that FS. The X/Y start/stop values define the coordinates of the bounding box that surrounds the object. The X/Y centroid values are the coordinates of the center of the bounding box. The object density is the number of blocks inside the bounding box that belong to the object. The object identifier is a unique value specified for each object.

Table 3-8: **Object Segmentation Metadata Data Structure**

Byte 3	Byte 2	Byte 1	Byte 0
Metadata Header (32 words)			
FS1 Object Data (32 instances)			
FS2 Object Data (32 instances)			
FS3 Object Data (32 instances)			
FS4 Object Data (32 instances)			

Table 3-9: **Object Segmentation Metadata Header**

Byte 3	Byte 2	Byte 1	Byte 0
Struct_Valid			
Frame_Index			
Total_Num_Objects			
FS4_num_obj	FS3_num_obj	FS2_num_obj	FS1_num_obj
Low_Freq_Mean	V_mean	U_Mean	Y_Mean
Saturation_Mean	Motion_mean	Edge_Mean	High_Freq_Mean
U_Var		Y_Var	
Low_Freq_Var		V_Var	
Edge_Var		High_Freq_Var	
Saturation_Var		Motion_Var	
PAD (x22)			

Note: The Mean and Variance values are the global values from the corresponding Image Characterization data structure.

Table 3-10: **Object Segmentation Metadata Object Data**

Byte 3	Byte 2	Byte 1	Byte 0
FS#		Object_Number	
X_stop		X_start	
Y_stop		Y_start	
Y_centroid		X_centroid	
Object_density			
Object_identifier			

Discontinued IP

Note: Repeat 6 object words for total of 32 objects. Object_number = 0 signals end of objects for this Feature Select. The remaining objects to 32 will = 0.

General Design Guidelines

Object Segmentation Control and Timing

The Object Segmentation core provides a great deal of operational flexibility through a simple register set. The Feature Combinations and Feature Selects define the operation of the core. These features are fully configurable and can be updated with configurations in real-time. As a result, the Object Segmentation core must be properly initialized before it can be used to process Image Characterization data. Each of the instantiated Feature Combinations and Feature Selects must be initialized with valid data structures as described in the Feature Combination Threshold Data Structure and Feature Select Data Structure sections.

Bank Programming

Feature Select Bank Programming

The Object Segmentation core supports eight banks of Feature Select data regardless of the number of Feature Selects that are instantiated. All of the Feature Select banks can be loaded at any time. It is the user's responsibility to verify that a bank is not loaded while it is in active use.

To load the Feature Selects:

1. Set the Feature Select Write Bank Address to the address (0 – 7) of the bank to be loaded.
2. Write 256 Feature Select data values to the Feature Select Data register. See the Feature Select Data Structure section for more details.
3. Repeat steps 1 and 2 for any additional Feature Select banks to be loaded.
4. Specify the active Feature Select bank by writing the bank's address to the Feature Select Active Bank Address register.

Feature Combination Bank Programming

The Object Segmentation core supports up to eight Feature Combinations. Each Feature combination is implemented with two banks of Feature Combination data. This makes for a total of up to 16 Feature Combination banks that can be independently loaded. All of the Feature Combination banks can be loaded at any time. It is the user's responsibility to verify that banks are not loaded while they are active.

Discontinued IP

To load the Feature Combinations:

1. Set the Feature Combination Write Bank Address to the address (0 –15) of the Feature Combination bank to be loaded. See Table 2-4 or Table 2-3 for more detail on the Feature Combination Write Bank Address register.
2. Write 40 Feature Combination data values to the Feature Combination Data register. See the Feature Combination Threshold Data Structure section for more detail.
3. Repeat steps 1 and 2 for any additional Feature Combination banks to be loaded.
4. Specify the active Feature Combination bank by writing the banks address to the Feature Combination Active Bank Address register.

Register Updates

The Object Segmentation core is controlled by a register set that must be initialized before the core begins processing. See Table 2-4 for more detail about the Object Segmentation register set. The registers should be initialized as follows:

1. Set the register update enable bit of the control register (bit 1) to '0' to disable register updates.
2. Load the Image Characterization Start Address 0 and 1 registers to the start addresses of the Image Characterization data buffers. See the Buffer Management section for more details.
3. Load the Metadata Start Address 0 and 1 registers to the start addresses of the Metadata buffers.
4. Load the Label Mask Start Address 0 register to the start address of the Label data buffer.
5. Load the Number of Horizontal Blocks register to the number of blocks in the horizontal direction. Typically this value is the horizontal resolution of the processed frame divided by the block size. Truncate any decimal portion.
6. Load the Number of Vertical Blocks register with the number of blocks in the vertical direction. Typically this value is the vertical resolution of the processed frame divided by the block size. Truncate any decimal portion.
7. Load the Number of Total Blocks with the number blocks that are going to be processed per frame. This value should be the Number of Horizontal blocks x the Number of Vertical blocks.
8. Load the Block Size register with the block size of the blocks that are being processed. A block is defined as a NxN 2-D grid of pixels where N is the block size. The Image Characterization core supports block sizes of 4, 8, 16, 32 and 64.
9. Set the register update enable bit of the control register (bit 1) to '1' and the core enable bit of the control register (bit 0) to '1' to fully enable the core. All of the preceding

Discontinued IP

register values are written into the core on the next falling edge of the register values are written into the core on the next falling edge of the `fsync_in` signal.

Any of the core's registers can be updated while the core is running. It is recommended that the register update enable bit of the control register (bit 1) be set to '0' before any register are updated.

After all register changes have been written, the register update enable bit should be set to '1'. The new register values are written into the core on the next falling edge of the `fsync_in` signal.

Discontinued IP

Buffer Management

The Object Segmentation core makes use of five external memory buffers to handle the transfer of data. The buffer locations are defined by registers that specify their starting address location. Two of the buffers are used to input Image Characterization data. Two buffers are used for outputting the Object Segmentation Metadata. The last buffer is used to hold intermediate results from the first pass of processing until it is read in during the second pass of processing.

The Image Characterization Start Addr 0/1 registers hold the start addresses of the buffers that the Object Segmentation core uses to input Image Characterization data. Two buffers are used so that the Image Characterization core can write to one buffer while the Object Segmentation core can read from the other buffer. This arrangement ensures that the Object Segmentation core is always processing a valid Image Characterization data structure.

The IC Buffer Pointer Mode bit (1) of the Buffer Control register (See Table 2-3) determines how the Image Characterization Buffers are used. When the Buffer Selection bit = "0", the "buffer_ptr" signal is sampled on the falling edge of "fsync_in" signal. If `buffer_ptr` = 0, Image Characterization Buffer 1 is used. If `buffer_ptr` = 1, Image Characterization Buffer 0 is used. When the Buffer Selection bit = "1", Image Characterization Buffer 0 is used for the first frame. For the next frame, Buffer 1 is use. The core continues to switch between buffers on each successive frame.

The Metadata Start Addr 0/1 registers hold the start addresses of the buffers that the Object Segmentation core uses when writing the Object Segmentation Metadata. The Object Segmentation core uses the "Buffer Select" register (Buffer_Control[0]) to specify which buffer is being actively written. The Object Segmentation core only writes to the active buffer. When the processor is ready to read the latest frame of Metadata, it first modifies the Metadata Address Selection register to swap the inactive buffer to be the active buffer and the active buffer to be the inactive buffer. After the next frame cycle begins, the newly inactive buffer contains the Metadata from the previous frame and is safe for the processor to access without danger of being overwritten. This mechanism is used because a processor might need to use multiple frame cycles to process the Metadata.

The Label Mask Start Addr 0 register holds the start address of the buffer that the Object Segmentation core uses to hold the intermediate Label data. This Label data is the output of the first pass of processing and the input of the second pass of processing. Only the Object Segmentation core uses this buffer, so there are no synchronization issues with which to be concerned.

Example Case

The key to understanding how to use the Object Segmentation core lies in learning how to properly configure the Feature Combinations and the Feature Selects, as described in the previous sections.

For this example, we will use the Object Segmentation core to detect objects that match either of the following feature descriptions:

1. Road signs that are Green with edges.
2. Road signs that are Yellow with edges.

For the purposes of this example we do not care to differentiate between the two feature descriptions, we just want to find objects of either type. To accomplish this, two Feature Combination units are needed; one to detect the "green signs" and one to detect the "yellow signs". Only one Feature Select is needed for this example because we are looking for either "green signs" or "yellow signs".

To properly setup the Object Segmentation core for this example, use the following configuration options:

- Color Select 1 is configured to detect the color "Green"
- Color Select 2 is configured to detect the color "Yellow"
- The video format is 4:2:0
- The frame resolution is 1280x720
- The Block Size is 8x8

Then to set these register values:

- Number of Horizontal Blocks = 1280/8 = 160
- Number of Vertical Blocks = 720/8 = 90
- Number of Total Blocks = 160x90 = 14400
- Block Size = 8

The next step is to configure the two Feature Combinations (FC1 and FC2). FC1 is looking for blocks that match the "green sign" feature description. To do this, it sets lower and upper block thresholds for "Color_Sel_1" (green) and for "Edge_Mean" (edges). FC2 is looking for blocks that match the "yellow sign" feature description. To do this it sets lower

Discontinued IP

and upper block thresholds for "Color_Sel_2" (yellow) and for "Edge_Mean" (edges). For FC1 and FC2, block values that are not part of the feature description should be set to their widest threshold settings so that they do not limit the data comparisons.

For this example, the global statistics are not of interest so the corresponding global thresholds in FC1 and FC2 should also be set to their widest threshold settings. Two separate Feature Combination Threshold data structures are illustrated in Table 3-11, one for FC1 and another for FC2. The values in red highlight the threshold values that correspond to the feature descriptions for FC1 and FC2.

The FC1 Feature Combination Threshold data structure has these values:

1. Line 0x7 - The Edge_Mean lower block threshold was set to a value of 0x21
2. Line 0xC - the Color_Sel_1 lower block threshold was set to a value 0x0004
3. Line 0x1B - The Edge_Mean upper block threshold was set to a value of 0x53
4. Line 0x20 - The Color_Sel_1 upper block threshold was set to a value of 0x0010

Discontinued IP

The FC2 Feature Combination Threshold data structure has these values:

1. Line 0x7 - The Edge_Mean lower block threshold was set to a value of 0x32
2. Line 0xC - the Color_Sel_1 lower block threshold was set to a value 0x0005
3. Line 0x1B - The Edge_Mean upper block threshold was set to a value of 0x61
4. Line 0x20 - The Color_Sel_1 upper block threshold was set to a value of 0x0012

Table 3-11: **Feature Combination Threshold Data Structures for FC1 and FC2**

Line #	FC1	FC2
Lower Global Statistics		
0x0 – 0x5	0x00000000	0x00000000
Lower Block Statistics		
0x6	0x00000000	0x00000000
0x7	0x00002100	0x00003200
0x8-0xB	0x00000000	0x00000000
0xC	0x00000004	0x00050000
0xD – 0x13	0x00000000	0x00000000
Upper Global Statistics		
0x14 – 0x19	0xFFFFFFFF	0xFFFFFFFF
Upper Block Statistics		
0x1A	0xFFFFFFFF	0xFFFFFFFF
0x1B	0x00005300	0x00006100
0x1C – 0x1F	0xFFFFFFFF	0xFFFFFFFF
0x20	0xFFFF0010	0x0012FFFF
0x21 – 0x23	0xFFFFFFFF	0xFFFFFFFF
0x24 -0x27	0x00000000	0x00000000

The last step is to configure the Feature Select (FS1). FS1 should combine FC1 and FC2 such that any block that matches FC1 or FC2 is considered a member of FS1. FS1 can be described by the following Boolean logic equation:

FS1=FC2 or FC1

This equation is implemented as a 1-bit x 256-entry look-up table. Table 3-12 has an 8-bit address range. FC1 is mapped to the lsb of the address range, FC2 is mapped to the "lsb + 1" and FC8 is mapped to the "lsb + 7". In this example, Only FC1 and FC2 are instantiated so FC3 - FC8 are each set to a value of "0". The Feature Select data structure that implements the logic equation for FS1 in this example is shown in Table 3-12. The Feature Select data structure is 4-bits wide and 256 entry deep regardless of the number of Feature Selects that are instantiated. The entire data structure must be created and loaded to properly configure the Feature Selects.

Discontinued IP

.

Table 3-12: **Feature Select Data Structure for FS1 = FC2 or FC1**

Look-up Table Address FC8=msb, FC1=lsb	FS4	FS3	FS2	FS1
0x00 (00000000)	0	0	0	0
0x01 (00000001)	0	0	0	1
0x02 (00000010)	0	0	0	1
0x03 (00000011)	0	0	0	1
0x04 (00000100)	0	0	0	0
0x05 (00000101)	0	0	0	0
0x06 – 0xFF	0	0	0	0

Because FS4 - FS2 are not instantiated, the column under FS1 is the only portion of Table 3-12 that is of concern. FC1 and FC2 are the only bits of the address range that can change because FC8 - FC3 are not instantiated and therefore each is set to a value of "0". As a result, the effective address range is 0x0 (00000000) - 0x3 (00000011). Because FS1 is equal to the "FC1 or FC2", FS1 has a value of '1' any time FC1 is a '1' as well as anytime FC2 is a '1'.

Use Model

Figure 3-6 illustrates using the Object Segmentation core in a larger system. In this system, the Image Characterization core writes its calculated image statistics to an external memory buffer. This external memory buffer is then read by the Object Segmentation core. The core then analyzes the data with the user-defined object characteristics to find the specified objects. A list of objects found is written back to an external Metadata buffer for use in higher level analysis and processing. Such a system can be easily built using the building blocks provided by Xilinx (for example, AXI_VDMA, Timing Controller, On Screen Display (OSD).

Discontinued IP

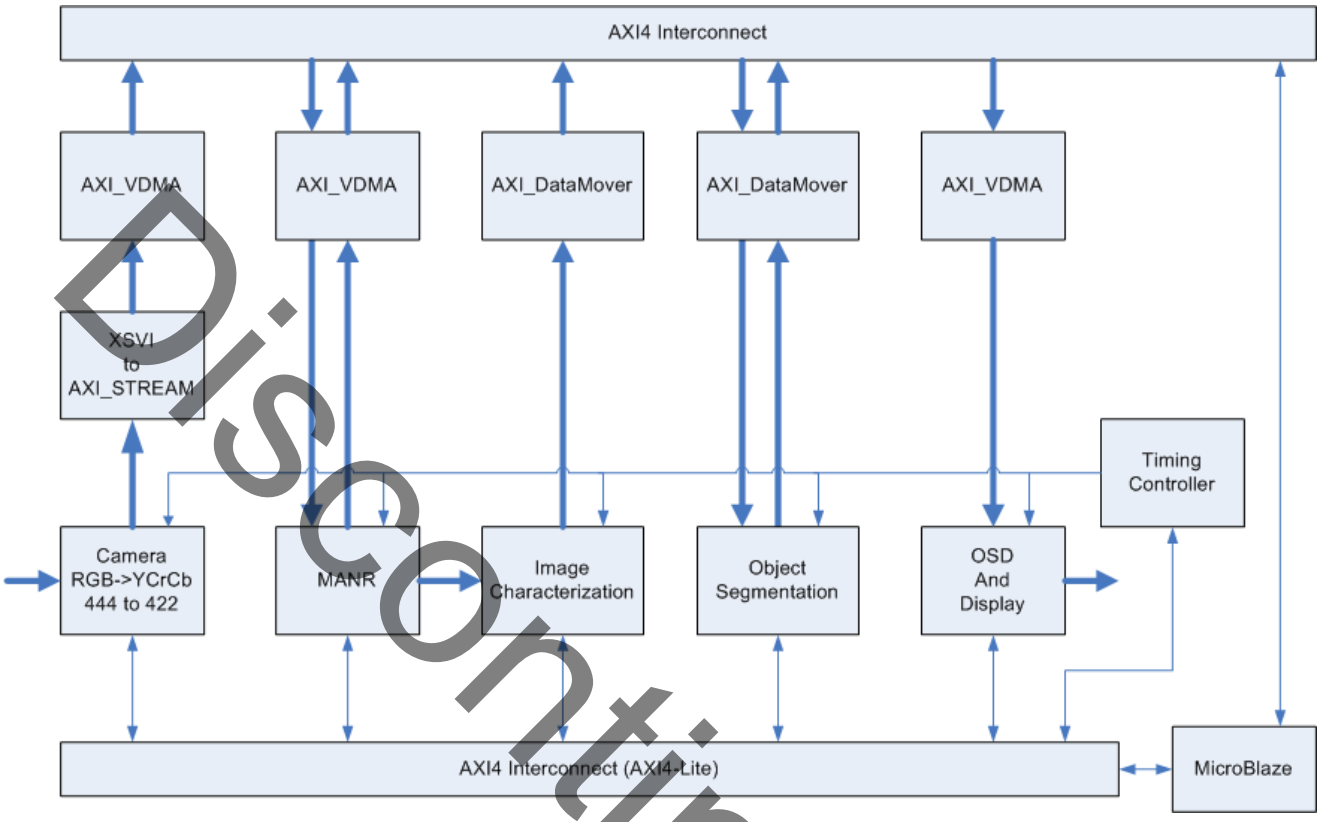

Figure 3-6: **Object Segmentation Example Use Model**

Discontinued IP

Clocking

ACLK

The ACLK clock is used to drive the master and slave AXI4-Stream video interfaces as well as the internal logic of the core.

S_AXI_ACLK

The AXI4-Lite interface uses the A_AXI_ACLK pin as its clock source. The ACLK pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The Object Segmentation core contains clock-domain crossing logic between the ACLK (AXI4-Stream and Video Processing) and S_AXI_ACLK (AXI4-Lite) clock domains. The core automatically ensures that the AXI4-Lite transactions complete even if the video processing is stalled with ARESETn, ACLKEN or with the video clock not running.

Resets

The Object Segmentation core has one reset ("sclr") that is used for the entire core. The reset is active high.

ARESETn

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

The external reset pulse needs to be held for 32 ACLK cycles to reset the core. The ARESETn signal only resets the AXI4-Stream interfaces. The AXI4-Lite interface is unaffected by the ARESETn signal to allow the video processing core to be reset without halting the AXI4-Lite interface.

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

S_AXI_ARESETn

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

Protocol Description

For the pCore version of the Object Segmentation core, the register interface is compliant with the AXI4-Lite interface. The S2MM and MM2S interfaces are compliant with the AXI4 Memory Mapped interface.

System Considerations

The Object Segmentation core must be configured to operate properly.

Discontinued IP

Clock Domain Interaction

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

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

Discontinued IP

C Model Reference

Discontinued IP

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

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

The latest version of the model is available for download on the LogiCORE IP Object Segmentation Web page at:
http://www.xilinx.com/products/ipcenter/EF-DI-VID-OBJ-SEG.htm

Instructions

Unpacking and Model Contents

Unzip the `v_objseg_v3_00_a_bitacc_model.zip` file, containing the bit accurate models for the Object Segmentation IP Core. This creates the directory structure and files in Table 4-1.

Table 4-1: **Directory Structure and Files of the Object Segmentation Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
pg018_v_obj_seg.pdf	LogiCORE IP Object Segmentation Product Guide
v_objseg_v3_00_a_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image/video container type and support functions
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions
yuv_utils.h	Header file declaring the YUV image/video container type and support functions

Table 4-1: **Directory Structure and Files of the Object Segmentation Bit Accurate C Model**

image_char_stats_utils.h	Header file declaring the Image Characterization Statistics container type and support functions
run_bitacc_cmodel.c	Example code calling the C model
ic_stats_512.txt	Example Image Characterization statistics files
objseg_config_512.cfg	Example configuration file
fc1.cfg	Example Feature Combination configuration file
fs1.cfg	Example Feature Select configuration file
/lin	Precompiled bit accurate American National Standards Institute (ANSI) C reference model for simulation on 32-bit Linux platforms
libIp_v_objseg_v3_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_objseg_v3_00_a_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_objseg_v3_00_a_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_ic_v3_00_a_bitacc_cmodel.so
/win32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms
libIp_v_objseg_v3_00_a_bitacc_cmodel.lib	Precompiled library file for Win32 compilation
/win64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms
libIp_v_objseg_v3_00_a_bitacc_cmodel.lib	Precompiled library file for Win64 compilation

Discontinued IP

Installation

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

- libIp_v_objseg_v3_00_a_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Object Segmentation C models were compiled and tested with the software listed in Table 4-2.

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

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

Interface

The bit accurate C model is accessed through a set of functions and data structures, declared in the header file `v_objseg_v3_00_a_bitacc_cmodel.h`

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

```
struct  xilinx_ip_v_objseg_v3_00_a_generics objseg_generics;
struct  xilinx_ip_v_objseg_v3_00_a_inputs   objseg_inputs;
struct  xilinx_ip_v_objseg_v3_00_a_outputs  objseg_outputs
```

The declaration of these structures are in the `v_objseg_v3_00_a_bitacc_cmodel.h` file.

Calling `xilinx_ip_v_objseg_v3_00_a_get_default_generics` (and `objseg_generics`) initializes the generics structure with the default values for each element of the structure.

Discontinued IP

The generics defaults are:

```
frames = 1                      // Number of frames
num_feature_combinations = 8; // Number of Feature Combination units
num_feature_selects = 4;       // Number of Feature Selection units
num_h_blocks = 160;            // Number of Horizontal blocks in IC Stats
num_v_blocks = 90;             // Number of Vertical blocks in IC Stats
num_total_blocks = 14400;      // Number of Total blocks in IC Stats
block_size = 8;                // Block Size (4, 8, 16, 32 or 64)

For fc[1] - fc[8]
   // Global Stats Lower Thresholds
   fc[i].lower.global_y_mean = 0;
   fc[i].lower.global_u_mean = 0;
   fc[i].lower.global_v_mean = 0;
   fc[i].lower.global_lf_mean = 0;
   fc[i].lower.global_hf_mean = 0;
   fc[i].lower.global_edge_mean = 0;
   fc[i].lower.global_mot_mean = 0;
   fc[i].lower.global_sat_mean = 0;
   fc[i].lower.global_y_var = 0;
   fc[i].lower.global_u_var = 0;
   fc[i].lower.global_v_var = 0;
   fc[i].lower.global_lf_var = 0;
   fc[i].lower.global_hf_var = 0;
   fc[i].lower.global_edge_var = 0;
   fc[i].lower.global_mot_var = 0;
   fc[i].lower.global_sat_var = 0;
   // Global Stats Upper Thresholds
   fc[i].upper.global_y_mean = 255;
   fc[i].upper.global_u_mean = 255;
   fc[i].upper.global_v_mean = 255;
   fc[i].upper.global_lf_mean = 255;
   fc[i].upper.global_hf_mean = 255;
   fc[i].upper.global_edge_mean = 255;
   fc[i].upper.global_mot_mean = 255;
   fc[i].upper.global_sat_mean = 255;
   fc[i].upper.global_y_var = 65535;
   fc[i].upper.global_u_var = 65535;
   fc[i].upper.global_v_var = 65535;
   fc[i].upper.global_lf_var = 65535;
   fc[i].upper.global_hf_var = 65535;
   fc[i].upper.global_edge_var = 65535;
   fc[i].upper.global_mot_var = 65535;
   fc[i].upper.global_sat_var = 65535;
   fc[i].upper.global_sat_var = 65535;
   // Block Stats Lower Thresholds
   fc[i].lower.block_y_mean = 0;
   fc[i].lower.block_u_mean = 0;
   fc[i].lower.block_v_mean = 0;
   fc[i].lower.block_lf_mean = 0;
   fc[i].lower.block_hf_mean = 0;
   fc[i].lower.block_edge_mean = 0;
   fc[i].lower.block_mot_mean = 0;
   fc[i].lower.block_sat_mean = 0;
   fc[i].lower.block_y_var = 0;
   fc[i].lower.block_u_var = 0;
   fc[i].lower.block_v_var = 0;
```

Discontinued IP

```
    fc[i].lower.block_lf_var = 0;
    fc[i].lower.block_hf_var = 0;
    fc[i].lower.block_edge_var = 0;
    fc[i].lower.block_mot_var = 0;
    fc[i].lower.block_sat_var = 0;
    fc[i].lower.block_col_sel1 = 0;
    fc[i].lower.block_col_sel2 = 0;
    fc[i].lower.block_col_sel3 = 0;
    fc[i].lower.block_col_sel4 = 0;
    fc[i].lower.block_col_sel5 = 0;
    fc[i].lower.block_col_sel6 = 0;
    fc[i].lower.block_col_sel7 = 0;
    fc[i].lower.block_col_sel8 = 0;
    // Block Stats Upper Thresholds
    fc[i].upper.block_y_mean = 255;
    fc[i].upper.block_u_mean = 255;
    fc[i].upper.block_v_mean = 255;
    fc[i].upper.block_lf_mean = 255;
    fc[i].upper.block_hf_mean = 255;
    fc[i].upper.block_edge_mean = 255;
    fc[i].upper.block_mot_mean = 255;
    fc[i].upper.block_sat_mean = 255;
    fc[i].upper.block_y_var = 65535;
    fc[i].upper.block_u_var = 65535;
    fc[i].upper.block_v_var = 65535;
    fc[i].upper.block_lf_var = 65535;
    fc[i].upper.block_hf_var = 65535;
    fc[i].upper.block_edge_var = 65535;
    fc[i].upper.block_mot_var = 65535;
    fc[i].upper.block_sat_var = 65535;
    fc[i].upper.block_sat_var = 65535;
    fc[i].upper.block_col_sel1 = 4095;
    fc[i].upper.block_col_sel2 = 4095;
    fc[i].upper.block_col_sel3 = 4095;
    fc[i].upper.block_col_sel4 = 4095;
    fc[i].upper.block_col_sel5 = 4095;
    fc[i].upper.block_col_sel6 = 4095;
    fc[i].upper.block_col_sel7 = 4095;
    fc[i].upper.block_col_sel8 = 4095;

fs[1] = fc[1];
fs[2] = fc[2];
fs[3] = fc[3];
fs[4] = fc[4];
```

The structure `objseg_inputs` defines the values of the input image characterization statistics. For a description of the input structure, see Image Characterization Statistics Input Structure.

The structure `objseg_outputs` defines the values of the output object segmentation metadata. For a description of the output structure, see Object Segmentation Metadata Output Structure.

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

Discontinued IP

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

```
int xilinx_ip_v_objseg_v3_00_a_bitacc_simulate(
      struct xilinx_ip_v_objseg_v3_00_a_generics* generics,
      struct xilinx_ip_v_objseg_v3_00_a_inputs*   inputs,
      struct xilinx_ip_v_objseg_v3_00_a_outputs*  outputs).
```

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

```
void xilinx_ip_v_objseg_v3_00_a_destroy(
      struct xilinx_ip_v_objseg_v3_00_a_inputs *input,
      struct xilinx_ip_v_objseg_v3_00_a_outputs *output)
```

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

Image Characterization Statistics Input Structure

The Object Segmentation reference model inputs a set of image characterization statistics for each frame that is processed. The input statistics are provided by `image_char_stats_struct`, which is defined in `image_char_stats_utils.h`.

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

struct global_stats_struct
{
  uint8   y_mean;      // Y mean
  uint8   u_mean;      // U mean
  uint8   v_mean;      // V mean
  uint8   m_mean;      // Motion mean
  uint8   e_mean;      // Edge mean
  uint8   lp_mean;     // Low Frequency mean
  uint8   hp_mean;     // High Frequency mean
  uint8   sat_mean;    // Saturation mean
  uint16  y_var;       // Y variance
  uint16  u_var;       // U variance
  uint16  v_var;       // V variance
  uint16  m_var;       // Motion variance
  uint16  e_var;       // Edge variance
  uint16  lp_var;      // Low Frequency variance
```

Discontinued IP

```
  uint16  hp_var;      // High Frequency variance
  uint16  sat_var;     // Saturation variance
};

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

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

Working With Image_char_stats_struct Containers

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

```
int alloc_ic_stats_buff(struct image_char_stats_struct* ic_stats);
void free_ic_stats_buff(struct image_char_stats_struct* ic_stats);
int write_ic_stats(FILE *output_fid, struct image_char_stats_struct *stats);
int read_ic_stats(FILE *input_fid, struct image_char_stats_struct* stats);
```

The `alloc_ic_stats_buff` function can be used to dynamically create an `image_char_stats_struct`. The frame, `num_blocks_wide` and `num_blocks_high`,

Discontinued IP

elements of the structure must be specified before calling this routine. The `free_ic_stats_buff` function can be used to destroy `image_char_stats_struct`.

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

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

The data structure matches the format of the output of the Image Characterization core. See *PG015 - LogiCORE IP Image Characterization v3.00a Product Guide* for more information.

Object Segmentation Metadata Output Structure

The Object Segmentation reference model outputs a set of object metadata for each frame that is processed. The object metadata are provided by `obj_seg_metadata_struct`, which is defined in `obj_seg_metadata_utils.h`.

```
struct obj_seg_metadata_struct
{
  int   frames;               // Number of frames
  int*  frame_index;          // Frame Index for each frame
  int*  total_num_objects;    // Total Number of Objects in the Frame
  int**  fs_num_objects;      // Number of Objects for FS1-4 in the Frame
  int*  y_mean;               // Global Y Mean for the Frame
  int*  u_mean;               // Global U Mean for the Frame
  int*  v_mean;               // Global V Mean for the Frame
  int*  lf_mean;              // Global Low Frequency Mean for the Frame
  int*  hf_mean;              // Global High Frequency Mean for the Frame
  int*  edge_mean;            // Global Edge Content Mean for the Frame
  int*  mot_mean;             // Global Motion Mean for the Frame
  int*  sat_mean;             // Global Saturation Mean for the Frame
  int*  y_var;                // Global Y Variance for the Frame
  int*  u_var;                // Global U Variance for the Frame
  int*  v_var;                // Global V Variance for the Frame
  int*  lf_var;               // Global Low Frequency Variance for the Frame
  int*  hf_var;               // Global High Frequency Variance for the Frame
  int*  edge_var;             // Global Edge Content Variance for the Frame
  int*  mot_var;              // Global Motion Variance for the Frame
  int*  sat_var;              // Global Saturation Variance for the Frame

  struct object_metadata_struct*** fs; // FS1-4 Object Data (32 objects each FS)
};

struct object_metadata_struct
{
  uint16   object_number;      // Object Number
  uint16   FS_number;          // Feature Select Number
```

Discontinued IP

```
  uint16   xstart;             // X Start coordinate of the bounding box
  uint16   xstop;              // X Stop coordinate of the bounding box
  uint16   ystart;             // Y Start coordinate of the bounding box
  uint16   ystop;              // Y Stop coordinate of the bounding box
  uint16   xcentroid;          // X Centroid of the object
  uint16   ycentroid;          // Y Centroid of the bounding box
  int      object_density;     // Number of object blocks inside the bounding box
  int      object_identifier;  // Unique object identifier
};
```

`Obj_seg_meta_data_struct` can hold the results of multiple processed frames. The number of frames in the structure is specified in the frames element of the structure. The `frame_index` is an array with one value per frame; it holds the index values of each frame. The `total_num_objects` element is an array with one value per frame. Each value holds the total number of objects found in all of the feature selects (fs1-4) for the corresponding frame. The element `fs_num_objects` is a 2-D array in which the first dimension is based on the number of frames and the second dimension is based on the number of Feature Selects. They hold the number of objects found for each Feature Select for each frame. The "*_mean" and "*_var" elements are arrays of global statistics; each array contains one value per frame. The fs element is a 3-D array of object data. The first dimension is based on the number of frames, the second dimension is based on the number of Feature Selects (fs1-fs4), and the third dimension is based on the maximum number of objects (32) for a feature select. The leaf element is a pointer to `object_metadata_struct`.

`Object_metadata_struct` holds the metadata associated with an object that was found by the Object Segmentation core, and consists of:

- The object number (1 - 32)
- The number of the Feature Select associated with it
- The coordinates of the objects bounding box
- The number of blocks inside the box that belongs to the object
- A unique object identifier

Working With Obj_seg_metadata_struct Containers

The `obj_seg_metadata_utils.h` file defines functions to simplify the use of Object Segmentation Metadata structures.

```
int alloc_obj_seg_metadata_buff(struct obj_seg_metadata_struct* obj_seg_meta);
void free_obj_seg_metadata_buff(struct obj_seg_metadata_struct* obj_seg_meta);

int write_obj_seg_metadata(FILE *output_fid, struct obj_seg_metadata_struct meta);
int read_obj_seg_metadata(FILE *output_fid, struct obj_seg_metadata_struct* meta);
```

The `alloc_obj_seg_metadata_buff` function can be used to dynamically create `obj_seg_metadata_struct`. The frame element of the structure must be specified before calling this routine. The `free_ic_stats_buff` function can be used to destroy `obj_seg_metadata_struct`.

Discontinued IP

The `write_obj_seg_metadata` function writes `obj_seg_metadata_struct` to a text file. The `read_obj_seg_metadata` function reads `obj_seg_metadata_struct` from a text file. Each frame of metadata in the text file is stored in this order:

1. Structure header
2. FS1 object 1 - FS1 0bject 32
3. FS2 object 1 - FS2 0bject 32
4. FS3 object 1 - FS3 0bject 32
5. FS4 object 1 - FS4 0bject 32

The data structure matches the format of the output of the Object Segmentation core. C Model Example Code

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

- Contains an example of how to write an application that makes a function call to the Object Segmentation C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Reads the Image Characterization statistics from an input file.
- Writes the Object Segmentation metadata to an output file.

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

```
Usage: run_bitacc_cmodel in_file config_file out_file

 in_file        :  Path/name of the input file.
 config_file    :  Path/name of the configuration file.
 out_file       :  Path/name of the output IC Stats file.
```

Config Files

The Object Segmentation model must be initialized using configuration files.

Object Segmentation Config File

During successful execution, the specified config file is parsed by the `run_bitacc_cmodel` example. This is the top-level config file that is specified in the command line arguments. In this file, you must specify:

- Number of frames to process
- Number of feature combinations and feature selects
- Number of horizontal and vertical blocks

Discontinued IP

- Feature combination config file for each feature combination
- Feature select config file the feature selects

The following example config file provides more information on the formatting of this file.

```
num_frames 2                  # Number of Frames
num_feature_combinations 8    # Number of Feature Combinations 1-8
num_feature_selects 4         # Number of Feature Selects 1-4
num_h_blocks 64               # Number of Horizontal Blocks in IC input
num_v_blocks 64               # Number of Vertical Blocks in IC input
fc1 fc1.cfg                   # Feature Combination config file for FC1
fc2 fc1.cfg                   # Feature Combination config file for FC2
fc3 fc1.cfg                   # Feature Combination config file for FC3
fc4 fc1.cfg                   # Feature Combination config file for FC4
fc5 fc1.cfg                   # Feature Combination config file for FC5
fc6 fc1.cfg                   # Feature Combination config file for FC6
fc7 fc1.cfg                   # Feature Combination config file for FC7
fc8 fc1.cfg                   # Feature Combination config file for FC8
fs fs1.cfg                    # Feature Select config file for FS1-FS4
```

Feature Combination Config File

The feature combination config file contains the data necessary to properly configure one feature combination in the object segmentation model. A separate feature combination config file should be loaded for each feature combination that is used. The config file specifies a set of lower and upper thresholds that are used when testing the global and block statistics in the image characterization input.

The following example config file provides more information on the formatting of this file.

```
# Block Stats
y_mean          100   255          # Block Y Mean
y_var             0 65535          # Block Y Variance
u_mean            0   255          # Block U Mean
u_var             0 65535          # Block U Variance
v_mean            0   255          # Block V Mean
v_var             0 65535          # Block V Variance
LP_y_mean         0   255          # Block Low Frequency Mean
LP_y_var          0 65535          # Block Low Frequency Variance
HP_y_mean         0   255          # Block High Frequency Mean
HP_y_var          0 65535          # Block High Frequency Variance
edge_y_mean       0   255          # Block Edge Content Mean
edge_y_var        0 65535          # Block Edge Content Variance
motion_y_mean     0   255          # Block Motion Mean
motion_y_var      0 65535          # Block Motion Variance
img_sat_mean      0   255          # Block Saturation Mean
img_sat_var       0 65535          # Block Saturation Variance
color_select1     0   255          # Block Color Select 1
color_select2     0   255          # Block Color Select 2
color_select3     0   255          # Block Color Select 3
color_select4     0   255          # Block Color Select 4
color_select5     0   255          # Block Color Select 5
color_select6     0   255          # Block Color Select 6
color_select7     0   255          # Block Color Select 7
color_select8     0   255          # Block Color Select 8
```

Discontinued IP

```
# Global Stats
global_y_mean          3   250      # Global Y Mean
global_y_var           0 65535      # Global Y Variance
global_u_mean          5   225      # Global U Mean
global_u_var           0 65535      # Global U Variance
global_v_mean          0   255      # Global V Mean
global_v_var           0 65535      # Global V Variance
global_LP_y_mean       0   255      # Global Low Frequency Mean
global_LP_y_var        0 65535      # Global Low Frequency Variance
global_HP_y_mean       0   255      # Global High Frequency Mean
global_HP_y_var        0 65535      # Global High Frequency Variance
global_edge_y_mean     0   255      # Global Edge Content Mean
global_edge_y_var      0 65535      # Global Edge Content Variance
global_motion_y_mean   0   255      # Global Motion Mean
global_motion_y_var    0 65535      # Global Motion Variance
global_img_sat_mean    0   255      # Global Saturation Mean
global_img_sat_var     0 65535      # Global Saturation Variance
```

Feature Select Config File

The feature select config file initializes all of the feature selects that are used. Each feature select is a Boolean equation that uses the feature combinations as terms in the equation. It is implemented as a 1-bit x 256-entry look-up table that uses the feature combinations as addresses into the table. FC1 is mapped to the LSB, FC2 is mapped to the LSB+1, and FC8 is mapped to the MSB.

The feature selects are initialized by loading 256 values that each consist of 4-bits. FS1 corresponds to bit 0, FS2 corresponds to bit 1, FS3 corresponds to bit 2 and FS4 corresponds to bit 3.

The feature select config file consists of 256 entries that are used to initialize the 4-bit x 256 entry feature select look-up table. The first value in the file corresponds to address 0x0 (00000000) in the table. Each subsequent value corresponds to the next address location in the table, incrementing all the way up to the top address of 0xFF (11111111). The following C code illustrates a simple way to calculate the data for a feature select config file.

```
// Initialize the 4 Feature Selection Banks
for(i=0;i<256;i++) {
   fc1 = i & 0x01;
   fc2 = (i >> 1 ) & 0x01;
   fc3 = (i >> 2 ) & 0x01;
   fc4 = (i >> 3 ) & 0x01;
   fc5 = (i >> 4 ) & 0x01;
   fc6 = (i >> 5 ) & 0x01;
   fc7 = (i >> 6 ) & 0x01;
   fc8 = (i >> 7 ) & 0x01;

   fs1[i] = fc1;
   fs2[i] = (fc2 && fc5) || fc7;
   fs3[i] = fc3 || fc4 || fc6;
   fs4[i] = fc1 && fc7 && fc8;

   fs[i] = fs4[i]<<3 | fs3[i]<<2 | fs2[i]<<1 | fs1[i];
```

Discontinued IP

```
}
```

Initializing the Image Characterization Input Data Structure

In the example code wrapper, data is assigned to `image_char_stats_struct` by reading from a file containing image characterization data. The `image_char_stats_utils.h` file provided with the bit accurate C model contains functions to facilitate this file I/O. The `run_bitacc_cmodel` example code uses this function to read from the delivered image characterization data file.

Image Characterization Data Files

The `image_char_stats_utils.h` file declares functions that help access image characterization files. The following functions operate on arguments of type `image_char_stats_struct`, which is defined in `image_char_stats_utils.h`.

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

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

Use the `alloc_ic_stats_buff` and `free_ic_stats_buff` commands to dynamically manage the memory associated with an image characterization data buffer. Use the `write_ic_stats` and `read_ic_stats` functions for file I/O operations.

Initializing the Object Segmentation Metadata Output Data Structure

In the example code wrapper, the object segmentation model writes the results to `obj_seg_metadata_struct`. These results are then written to a file. The `obj_seg_metadata_utils.h` file provided with the bit accurate C model contains functions to facilitate this file I/O.

Object Segmentation Metadata Files

The `obj_seg_metadata_utils.h` file declares functions that help access object segmentation metadata files. The following functions operate on arguments of type `obj_seg_metadata_struct`, which is defined in `obj_seg_metadata_utils.h`.

```
int alloc_obj_seg_metadata_buff(struct obj_seg_metadata_struct* obj_seg_meta);
void free_obj_seg_metadata_buff(struct obj_seg_metadata_struct* obj_seg_meta);

int write_obj_seg_metadata(FILE *output_fid,
                           struct obj_seg_metadata_struct meta);
int read_obj_seg_metadata(FILE *output_fid,
                          struct obj_seg_metadata_struct* meta);
```

Discontinued IP

Use the `alloc_obj_seg_metadata_buff` and `free_obj_seg_metadata_buff` commands to dynamically manage the memory associated with an object segmentation metadata buffer. Use the `write_obj_seg_metadata` and `read_obj_seg_metadata` functions for file I/O operations.

C-Model Example I/O Files

Input Files

- **<in_filename>** (for example, `ic_stats_in.txt`)
  - Image Characterization statistics
- **<config_file>** (for example, `objseg_config_512.cfg`)
  - Object Segmentation configuration

Output Files

- **<out_filename>** (for example, `objseg_out.txt`)
  - Object Segmentation metadata

Compiling the Object Segmentation v3.00a C Model With Example Wrapper

Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

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

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

2. Copy these files from the /lin (for 32-bit) or from the /lin64 (for 64-bit) directory to the root directory:
   - libstlport.so.5.1
   - libIp_v_objseg_v3_00_a_bitacc_cmodel.so
3. In the root directory, compile using the GNU C Compiler with this command:

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L.
-lIp_v_objseg_v3_00_a_bitacc_cmodel -Wl,-rpath,.
```

4. This results in the creation of the executable `run_bitacc_cmodel`, which can be run using:

Discontinued IP

```
./run_bitacc_cmodel ic_stats_512.txt objseg_config_512.cfg
objseg_out.txt
```

Windows (32-bit and 64-bit)

Precompiled library `v_objseg_v3_00_a_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` must be compiled with an ANSI C compliant compiler under Windows. Here, an example is provided using Microsoft Visual Studio.

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

- **libIp_v_objseg_v3_00_a_bitacc_cmodel.lib** to the "Resource Files" folder of the project
- **run_bitacc_cmodel.c** to the "Source Files" folder of the project
- **v_objseg_v3_00_a_bitacc_cmodel.h** to "Header Files" folder of the project
- **yuv_utils.h** to the "Header Files" folder of the project
- **rgb_utils.h** to the "Header Files" folder of the project
- **video_utils.h** to the "Header Files" folder of the project
- **image_char_stats_utils.h** to the "Header Files" folder of the project
- **obj_seg_metadata_utils.**h to the "Header Files" folder of the project

After the project is created and populated, it must be compiled and linked (built) to create a Win32 or Win64 executable. To perform the build step, choose **Build Solution** from the Build menu. An executable matching the project name is created in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" is selected in the "Configuration Manager" in the Build menu.

Running the Delivered Executables

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

Linux (32-bit and 64-bit)

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

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

2. Copy these files from the /lin or /lin64 directory to the root directory:
   - libstlport.so.5.1
   - libIp_v_objseg_v3_00_a_bitacc_cmodel.so

Discontinued IP

- run_bitacc_cmodel

3. Execute the model:

```
./run_bitacc_cmodel ic_stats_512.txt objseg_config_512.cfg
objseg_out.txt
```

Windows (32-bit and 64-bit)

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

```
./run_bitacc_cmodel ic_stats_512.txt objseg_config_512.cfg
objseg_out.txt
```

Discontinued IP

Object Segmentation Metadata Output

Image Characterization Statistics Input

The image characterization statistics input is described in the Image Characterization Statistics Input Structure section. For additional information on the image characterization statistics input structure, see the *LogiCORE IP Image Characterization Product Guide* (PG015).

Metadata Output

This section includes an example of the object segmentation metadata output. The comments explain how to read this output. For more information on the object segmentation metadata structure, see Object Segmentation Metadata Output Structure.

The first 32 lines of the data are the metadata frame header. The header contains a frame index, the number of objects in the image frame, and the image characterization global statistics for the frame. Following the frame header are 32 object descriptions for Feature Select 1 (FS1). Each object description contains:

- The coordinate information for a box that bounds the object (X start/stop, Y start/stop)
- The centroid of the box (X, Y)
- The object density (the number of blocks inside the box that belong to the object)
- An object identifier (Cyclic Redundancy Check (CRC) of the other five lines of object information)

If less than 32 objects are found for FS1, the remaining object descriptions are still present, but the contents are set to 0. The FS1 object descriptions are followed by the FS2 object descriptions, which are followed by the FS3 objects descriptions, which are followed by the FS4 object descriptions. If fewer then four Feature Selects are instantiated, then only that number of Feature Select object descriptions are output. For example, if two Feature Selects are instantiated, then only the object descriptions for FS1 and FS2 are output.

Multiple object segmentation metadata structures can be in one output file. The next metadata structure begins directly after the end of the previous metadata structure.

```
# Metadata Frame Header
-1              # Frame Struct Valid (0xFFFFFFFF)
1               # Frame Index
12              # Total Number of Objects (FS4 thru FS1)
16777226        # FS 4 = 1, FS 3 = 0, FS 2 = 0, FS 1 = 10
                # Global Statistics
-1786936427     # Low Frequency Mean = 0x95, V Mean = 0x7D,
                # U Mean = 0x83, Y Mean = 0x95
117446148       # Saturation Mean = 0x07, Motion Mean = 0x00,
                # Edge Mean = 0x16, High Frequency Mean = 0x04
3539068         # U Variance = 0x0036, Y Variance = 0x007C
7798842         # Low Frequency Variance = 0x0077, V Variance = 0x003A
49348637        # Edge Variance = 0x02F1, High Frequency Variance = 0x001D
262144          # Saturation Variance = 0x0004, Motion Variance = 0x0000
0               # Frame Header Padding (22 lines)
…
0
# Metadata Feature Select #1
65537           # FS #1, Object 1
17826056        # XStop = 0x110, XStart = 0x108
4718656         # YStop = 0x048, YStart = 0x040
4456716         # YCentroid = 0x044, XCentroid = 0x10C
1               # Object Density (number of blocks in the object)
18677828        # Object Identifier
65538           # FS #1, Object 2
32506344
10485912
10224108
1
30212255
65539           # FS 1, Object 3
30933416
13107376
12321216
```

Discontinued IP

```
13
28115158
65540          # FS 1, Object 4
33554912
15728816
13631984
16
35717300
65541          # FS 1, Object 5
33554936
12058800
11796988
1
34406576
65542          # FS 1, Object 6
14155776
22020280
17039468
210
9240576
65543          # FS 1, Object 7
19398872
14680248
13369600
19
17105268
65544          # FS 1, Object 8
19923240
12583096
12321068
1
21823669
65545          # FS 1, Object 9
25690424
21495992
16777568
85
29425852
65546          # FS 1, Object 10
26214792
14155984
13894028
1
```

Discontinued IP

```
27066591
65536          # FS 1, Object 11
0
0
0
0
0
#FS 1 Objects 12 - 32 are repeats of Object 11
# Metadata Feature Select #2
131073         # FS 2, Object 1
33554432
33554432
16777472
4096
16912641
131072         # FS 2, Object 2
0
0
0
0
0
#FS 2 Objects 3 - 32 are repeats of Object 2
# Metadata Feature Select #3
196608         # FS 3, Object 1
0
0
0
0
0
#FS 3 Objects 2 - 32 are repeats of Object 1
# Metadata Feature Select #4
262145         # FS 4, Object 1
33554432
33554432
16777472
4096
17043713
262144         # FS 4, Object 2
0
0
0
0
0
```

Discontinued IP

```
#FS 4 Objects 3 - 32 are repeats of Object 2
```

Discontinued IP

SECTION II: VIVADO DESIGN SUITE

Discontinued IP

Customizing and Generating the Core

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

Discontinued IP

Graphical User Interface (GUI)

The Xilinx® Image Characterization core is easily configured to meet the developer's specific needs through the Vivado IP Catalog's Graphical User Interface (GUI). This section provides a quick reference to the parameters that can be configured at generation time. The GUI is shown in Figure 5-1 through Figure 5-3.

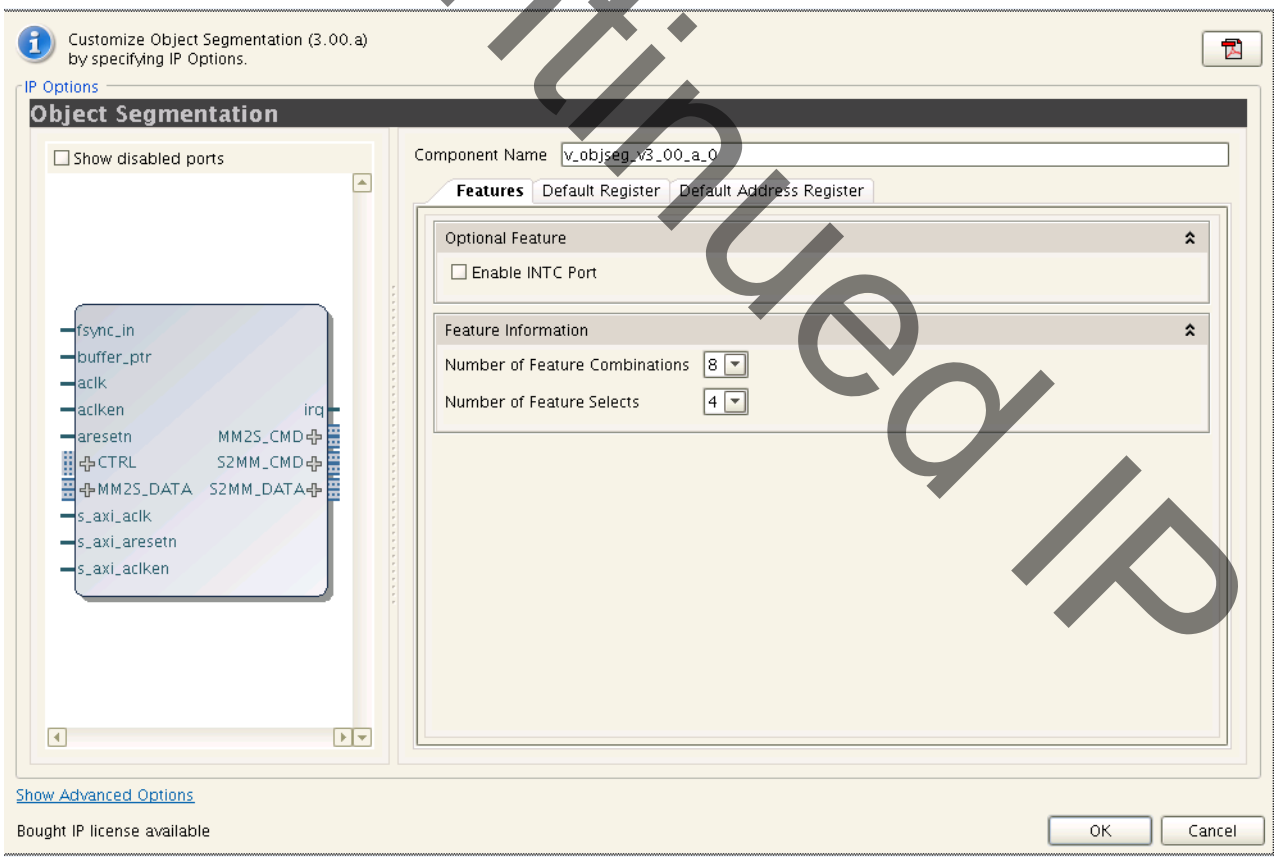

Figure 5-1: **Object Segmentation Vivado GUI**

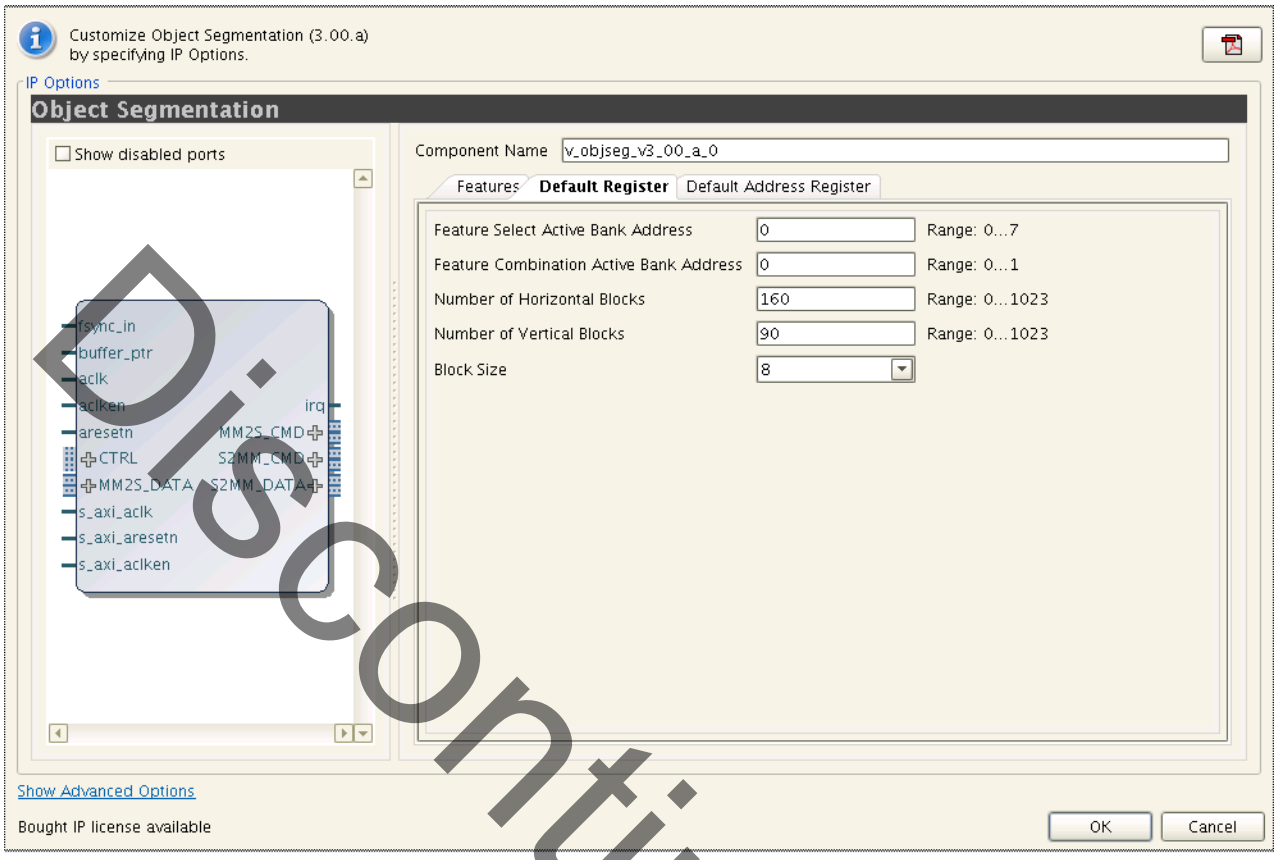

Figure 5-2: **Object Segmentation Vivado GUI - Default Registers (Page 1)**

Discontinued IP

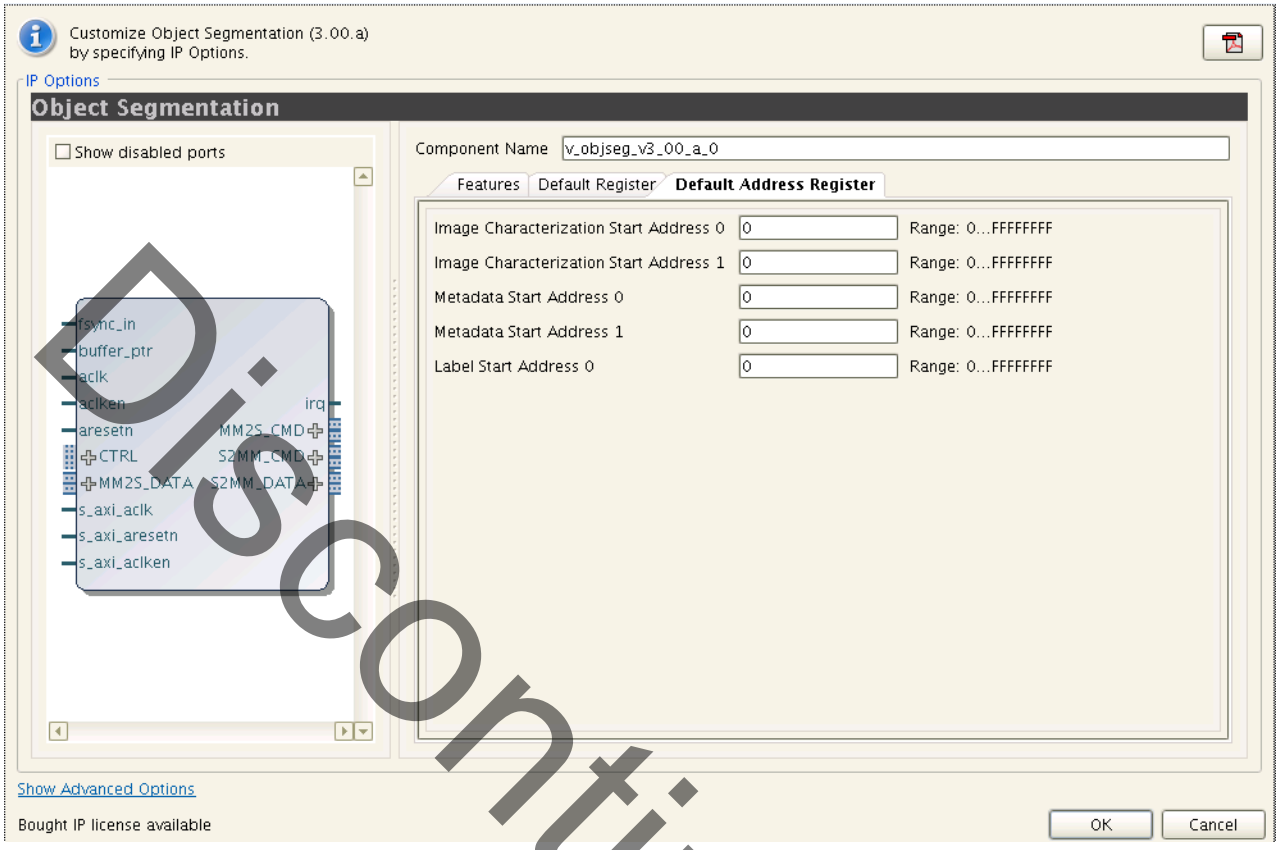

Figure 5-3: **Object Segmentation Vivado GUI - Default Registers (Page 2)**

Discontinued IP

Figure 5-1 displays the GUI parameters that affect the instantiation of the Object Segmentation core. The selections on this page will affect the amount of resources that re use when the core is generated.

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9, and "_". Note: The name "**v_objseg_v3_00_a**" is not allowed.
- **Optional Features**
  - **Enable INTC Port**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions.
- **Feature Information**
  - **Number of Feature Combinations**: Sets the number of Feature Combination units that are instantiated in the core. The range of valid choices is 1 - 8. The higher the selected value the more resources that are used.
  - **Number of Feature Selects**: Sets the number of Feature Select units that are instantiated in the core. The range of valid choices is 1 - 4. The higher the selected value the more resources that are used.

Figure 5-2 and Figure 5-3 display the GUI parameters that affect the default register values of the Object Segmentation core. These values set the default values of the corresponding registers in the AXI4-Lite interface.

- **Feature Select Active Bank Address**: Sets the default bank address of the Feature Select bank that will be used when processing data. There are 8 banks with addresses 0 through 7.
- **Feature Combination Active Bank Address**: Sets the default bank address of the Feature Combination bank that will be used when processing data. There are 2 banks with addresses 0 and 1.
- **Number of Horizontal Blocks**: Sets the default number of blocks in the horizontal direction.
- **Number of Vertical Blocks**: Sets the default number of blocks in the vertical direction.
- **Block Size**: Specifies the size of the block that was used when the Image Characterization data was calculated. Valid selections are 4, 8, 16, 32, or 64.
- **Image Characterization Start Address 0**: Specifies the Start Address of the first external memory buffer that will hold the Image Characterization data structure.
- **Image Characterization Start Address 1**: Specifies the Start Address of the second external memory buffer that will hold the Image Characterization data structure.
- **Metadata Start Address 0**: Specifies the Start Address of the first external memory buffer that will hold the MetaData data structure.
- **Metadata Start Address 1**: Specifies the Start Address of the second external memory buffer that will hold the MetaData data structure.
- **Label Start Address**: Specifies the start address of the external memory buffer that will hold the Label data.

Output Generation

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

File Details

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

Discontinued IP

Table 5-1: **Output Files**

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

Discontinued IP

Constraining the Core

Discontinued IP

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

Required Constraints

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

XDC

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

Device, Package, and Speed Grade Selections

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

Clock Frequencies

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

Clock Management

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

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

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

Discontinued IP

Detailed Example Design

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

Demonstration Test Bench

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

RECOMMENDED: *Make simple modifications to the test conditions and observe the changes in the waveform.*

To obtain the test bench, go the Object Segmentation core's product page.

http://www.xilinx.com/products/intellectual-property/EF-DI-VID-OBJ-SEG.htm

Test Bench Structure

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

- DUT

  The Object Segmentation v3.00a core instance under test.

- axi4lite_mst

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

- axi4s_video_mst

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

- axi4s_video_slv

Discontinued IP

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

- ce_gen

  Programmable Clock Enable (ACLKEN) generator

Running the Simulation

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

Directory and File Contents

The directory structure underneath the top-level folder is:

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

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

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

Discontinued IP

- 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

Discontinued IP

SECTION III: ISE DESIGN SUITE

Discontinued IP

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

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

Discontinued IP

Graphical User Interface (GUI)

CORE Generator Software GUI

The Xilinx® Image Characterization core is easily configured to meet the developer's specific needs through the CORE Generator™ software GUI. This section provides a quick reference to the parameters that can be configured at generation time. The GUI is shown in Figure 8-1 through Figure 8-2.

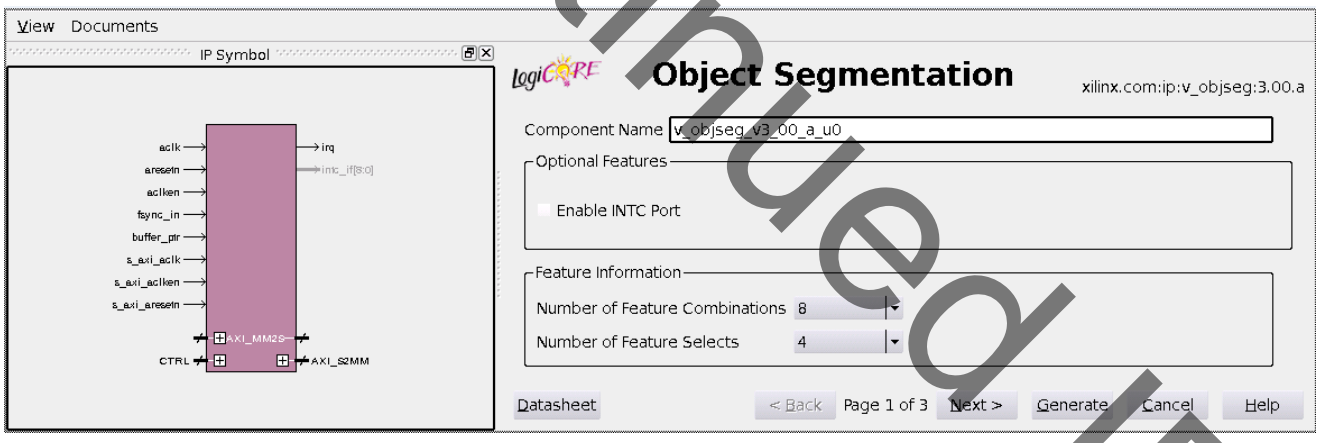

Figure 8-1: **Object Segmentation CORE Generator GUI**

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

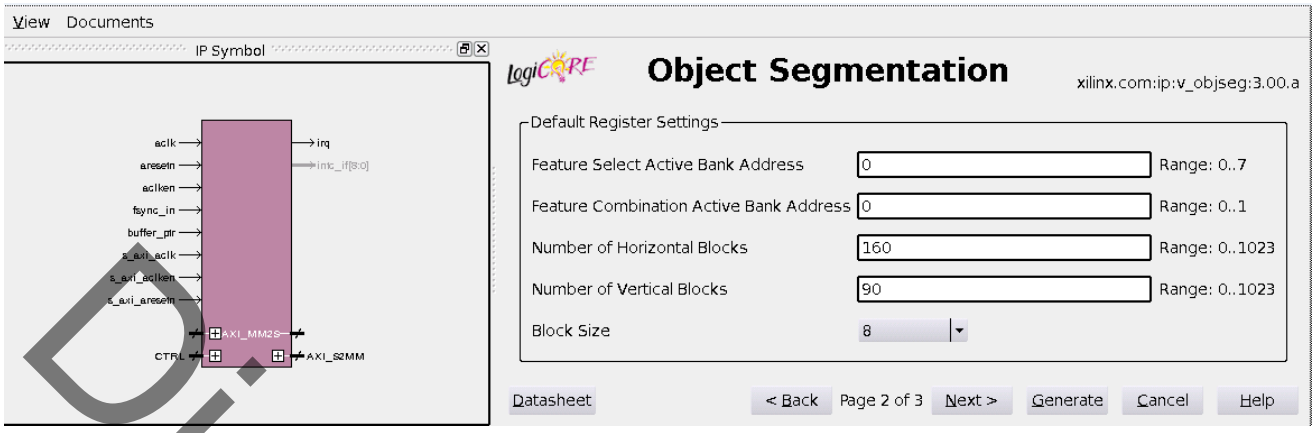

Figure 8-2: **Object Segmentation CORE Generator GUI - Default Registers (Page 1)**

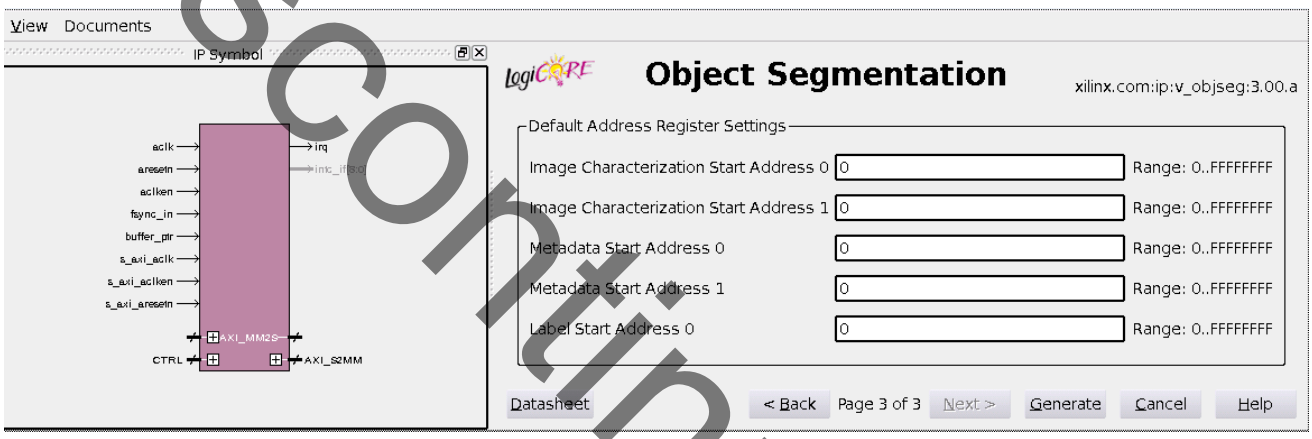

Figure 8-3: **Object Segmentation CORE Generator GUI - Default Registers (Page 2)**

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

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9, and "_". Note: The name "**v_objseg_v3_00_a**" is not allowed.
- **Optional Features**
  - **Enable INTC Port**: When selected, the core will generate the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions.
- **Feature Information**
  - **Number of Feature Combinations**: Sets the number of Feature Combination units that are instantiated in the core. The range of valid choices is 1 - 8. The higher the selected value the more resources that are used.

Discontinued IP

- **Number of Feature Selects**: Sets the number of Feature Select units that are instantiated in the core. The range of valid choices is 1 - 4. The higher the selected value the more resources that are used.

Figure 8-2 through Figure 8-3. displays the GUI parameters that affect the default register values of the Object Segmentation core. These values set the default values of the corresponding registers in the AXI4-Lite interface.

- **Feature Select Active Bank Address**: Sets the default bank address of the Feature Select bank that will be used when processing data. There are 8 banks with addresses 0 through 7.
- **Feature Combination Active Bank Address**: Sets the default bank address of the Feature Combination bank that will be used when processing data. There are 2 banks with addresses 0 and 1.
- **Number of Horizontal Blocks**: Sets the default number of blocks in the horizontal direction.
- **Number of Vertical Blocks**: Sets the default number of blocks in the vertical direction.
- **Block Size**: Specifies the size of the block that was used when the Image Characterization data was calculated. Valid selections are 4, 8, 16, 32, or 64.
- **Image Characterization Start Address 0**: Specifies the Start Address of the first external memory buffer that will hold the Image Characterization data structure.
- **Image Characterization Start Address 1**: Specifies the Start Address of the second external memory buffer that will hold the Image Characterization data structure.
- **Metadata Start Address 0**: Specifies the Start Address of the first external memory buffer that will hold the MetaData data structure.
- **Metadata Start Address 1**: Specifies the Start Address of the second external memory buffer that will hold the MetaData data structure.
- **Label Start Address**: Specifies the start address of the external memory buffer that will hold the Label data.

EDK pCore Graphical User Interface (GUI)

When the Xilinx Object Segmentation core is generated from the EDK software, it is generated with each option set to the default value. All customizations of an Object Segmentation pCore are done with the EDK pCore graphical user interface (GUI). Figure 8-4 illustrates the EDK pCore GUI for the Object Segmentation pCore. All of the options in the EDK pCore GUI for the Object Segmentation core correspond to the same options in the CORE Generator software GUI. See CORE Generator Software GUI for details about each option.

Discontinued IP

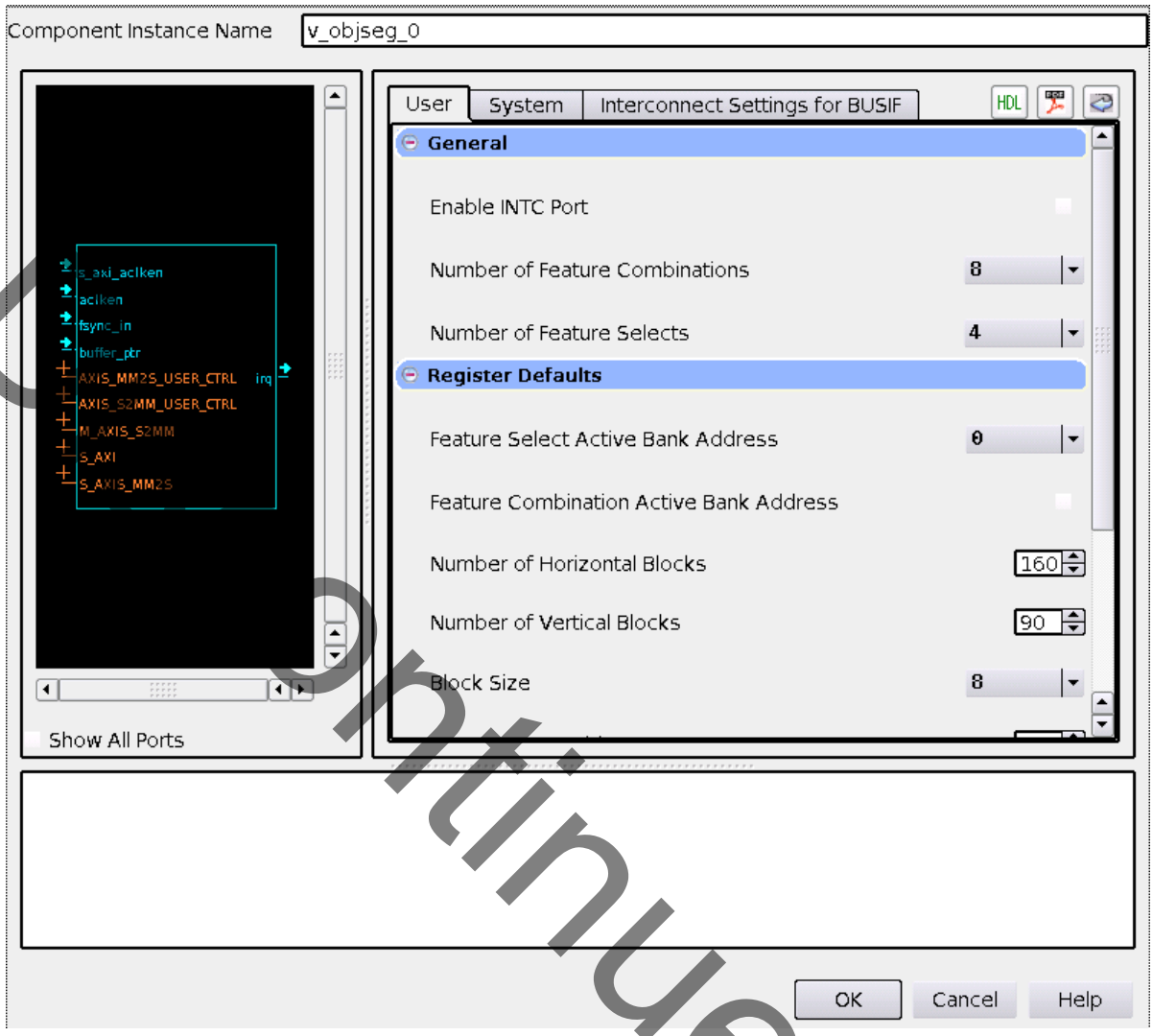

Figure 8-4: **Object Segmentation pCore GUI**

Discontinued IP

Parameter Values in the XCO File

Table 8-1 defines valid entries for the Xilinx CORE Generator (XCO) software 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 8-1: **XCO Parameters**

XCO Parameter	Default	Valid Values
component_name	v_objseg_v3_00_a_u0	ASCII text using characters: a..z, 0..9 and "_" starting with a letter. Note: "v_objseg_v2_0" is not allowed.
c_has_intc_if	false	false, true
c_num_feature_combinations	8	1,2,3,4,5,6,7,8
c_num_feature_selects	4	1,2,3,4
c_feature_select_active_bank_addr	0	0,1,2,3,4,5,6,7
c_feature_combination_active_bank_addr	0	0,1
c_num_h_blocks	160	2..1023
c_num_v_blocks	90	2..1023
c_block_size	8	4, 8, 16, 32, 64
c_image_stats_start_addr0	0	0..0xFFFFFFFF
c_image_stats_start_addr1	0	0..0xFFFFFFFF
c_meta_data_start_addr0	0	0..0xFFFFFFFF
c_meta_data_start_addr1	0	0..0xFFFFFFFF
c_label_mask_start_addr0	0	0..0xFFFFFFFF

Discontinued IP

Output Generation

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

File Details

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

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo <component_name>.vho	The HDL template for instantiating the core.
<component_name>.v <component_name>.vhd	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.
<component_name>_flist.txt	A text file listing all of the output files produced when the customized core was generated in the CORE Generator software.

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

Discontinued IP

Constraining the Core

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

Discontinued IP

Required Constraints

The ACLK pin should be constrained to at the desired clock rate for your core logic.

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

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

UCF

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

Device, Package, and Speed Grade Selections

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

Clock Frequencies

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

Clock Management

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

Clock Placement

There are no specific Clock placement requirements for this core.

Banking

There are no specific Banking rules for this core.

Transceiver Placement

There are no Transceiver Placement requirements for this core.

I/O Standard and Placement

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

Discontinued IP

Detailed Example Design

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

Demonstration Test Bench

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

RECOMMENDED: *Make simple modifications to the test conditions and observe the changes in the waveform.*

To obtain the test bench, go the Object Segmentation core's product page.

http://www.xilinx.com/products/intellectual-property/EF-DI-VID-OBJ-SEG.htm

Test Bench Structure

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

- DUT

  The Object Segmentation v3.00a core instance under test.

- axi4lite_mst

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

- axi4s_video_mst

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

- axi4s_video_slv

Discontinued IP

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

- ce_gen

  Programmable Clock Enable (ACLKEN) generator

Running the Simulation

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

Directory and File Contents

The directory structure underneath the top-level folder is:

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

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

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

Discontinued IP

- 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

Discontinued IP

SECTION IV: APPENDICES

Discontinued IP

Appendix A

Verification, Compliance, and Interoperability

Discontinued IP

Simulation

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

- Register accesses
- Processing of multiple frames of data
- Testing of various frame sizes and block sizes
- Varying instantiations of the core (Feature Selects = 1 – 4 and Feature Combinations = 1 – 8)
- Varying Feature Select Bank usage
- Varying Feature Combination Bank usage

Hardware Testing

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

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4 Interface and various other peripherals. The software for the test system included pre-generated input and output for the Object Segmentation core. Various tests could be supported by varying the configuration of the Object Segmentation core or by loading a different software executable. The MicroBlaze processor was responsible for:
  - Initializing the appropriate input and output buffers in external memory.
  - Initializing the Object Segmentation core.
  - Launching the test.

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

Interoperability

The Object Segmentation core's AXI4-Stream interfaces are designed to be connected to the AXI-Data Mover core which in tern is connected to the AXI-Interconnect.

Discontinued IP

Migrating

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

Discontinued IP

Parameter Changes in the XCO File

All of the XCO parameter names have changed and new parameters have been added for the v3.00.a release of the core. See Parameter Values in the XCO File in Chapter 8 for a full listing of the current XCO parameters.

Port Changes

The Object Segmentation v3.00.a core replaces the AXI4-Memory Mapped interfaces with AXI4-Stream interfaces. The new interfaces require the instantiation of an AXI-Data Mover core to connect to these AXI4-Stream interfaces and then connect to the AXI-Interconnect.

Functionality Changes

The basic functionality of the Object Segmentation core has not changed. The most notable change for the v3.00.a version of the core was the change from AXI4-Memory Mapped interfaces to AXI4-Stream interfaces. See Port Changes for more details.

Special Considerations when Migrating to AXI

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

Debugging

Discontinued IP

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

The following topics are included in this appendix:

Finding Help on Xilinx.com

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

Documentation

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

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

Release Notes

Known issues for all cores, including the Object Segmentation are described in the IP Release Notes Guide (XTP025).

Known Issues

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

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

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

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

Contacting Technical Support

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

To contact Xilinx Technical Support:

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

When opening a WebCase, include:

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

Discontinued IP

Debug Tools

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

Example Design

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

ChipScope Pro Tool

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

Vivado Lab Tools

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

Reference Boards

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

- 7 series evaluation boards
  - KC705
  - KC724
  - ZC702

C-Model Reference

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

Discontinued IP

License Checkers

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

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

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

Hardware Debug

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

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

- General Checks
- Evaluation Core License

General Checks

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

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.

Discontinued IP

Evaluation Core Timeout

When generated with an Full System Hardware Evaluation license, the core includes a timeout circuit that disables the core after a specific period of time. The timeout circuit can only be reset by reloading the FPGA bitstream. The timeout period for this core is set to approximately 8 hours for a 75 MHz clock. Using a faster or slower clock changes the timeout period proportionally. For example, using a 150 MHz clock results in a timeout period of approximately four hours.

Interface Debug

AXI4-Lite Interfaces

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

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

Symptom	Solution
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Are the S_AXI_ACLK and ACLK pins connected? In EDK, verify that the S_AXI_ACLK and ACLK pin connections in the system.mhs file. The VERSION_REGISTER readout issue may be indicative of the core not receiving the AXI4-Lite interface.
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core enabled? Is s_axi_aclken connected to vcc? In EDK, verify that signal ACLKEN is connected in the **system.mhs** to either net_vcc or to a designated clock enable signal.
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core in reset? S_AXI_ARESETn and ARESETn should be connected to vcc for the core not to be in reset. In EDK, verify that the S_AXI_ARESETn and ARESETn signals are connected in the **system.mhs** to either net_vcc or to a designated reset signal.
Readback value for the VERSION_REGISTER is different from expected default values	The core and/or the driver in a legacy EDK/SDK project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared.

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

AXI4-Stream Interfaces

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

Discontinued IP

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

Symptom	Solution
Bit 0 of the ERROR register reads back set.	Bit 0 of the ERROR register, EOL_EARLY, indicates the number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.
Bit 1 of the ERROR register reads back set.	Bit 1 of the ERROR register, EOL_LATE, indicates the number of pixels received between the last End-Of-Line (EOL) signal surpassed the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between EOL pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the Object Segmentation 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 Object Segmentation 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 EOL signal (m_axis_video_tlast) signal misplaced.	Check the ERROR register.
Data samples lost between Upstream core and the Object Segmentation core. Inconsistent EOL 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 Object Segmentation core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the Object Segmentation ACLK pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the Object Segmentation core. Inconsistent EOL 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 Object Segmentation core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the Object Segmentation ACLK pin lower than the reported Fmax reached?

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

Discontinued IP

Application Software Development

Discontinued IP

pCore Driver Files

The Object Segmentation pCore includes a software driver written in the C programming language that the user can use to control the core. A high-level API provides application developers easy access to the features of the Xilinx® Object Segmentation core. A low-level API is also provided for developers to access the core directly through the system registers described in Register Space in Chapter 2.

Table D-1 lists the files included with the Object Segmentation pCore driver.

Table D-1: **Object Segmentation pCore Drivers**

File name	Description
xos.c	Provides the API access to all features of the Object Segmentation device driver.
xos.h	Provides the API access to all features of the Object Segmentation device driver.
xos_g.c	Contains a template for a configuration table of Object Segmentation core.
xos_hw.h	Contains identifiers and register-level driver functions (or macros) that can be used to access the Object Segmentation core.
xos_intr.c	Contains interrupt-related functions of the Object Segmentation device driver.
xos_sinit.c	Contains static initialization methods for the Object Segmentation device driver.

pCore API Functions

This section describes the functions included in the pcore Driver files generated for the Object Segmentation pCore. The software API is provide to allow easy access to the registers of the pCore as defined in Table 2-2 in the Register Space section. To utilize the API functions provided, the following header files must be included in the user's C code:

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

```
#include "xos.h"
```

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

More detailed documentation of the API functions can be found by opening the file index.html in the pCore directory os_v3_00_a/doc/html/api.

Functions in xos.c

- int XOS_CfgInitialize (XOS *InstancePtr, XOS_Config *CfgPtr, u32 EffectiveAddr)

  This function initializes an OS device.

- void XOS_SetImageStatAddr (XOS *InstancePtr, u32 Addr1, u32 Addr2)

  This function sets up input image statistics frame buffer addresses for an OS device.

- void XOS_GetImageStatAddr (XOS *InstancePtr, u32 *Addr1Ptr, u32 *Addr2Ptr)

  This function fetches the input image statistics frame buffer addresses for an OS device.

- void XOS_SetMetaDataAddr (XOS *InstancePtr, u32 Addr1, u32 Addr2)

  This function sets up output meta data frame buffer addresses for an OS device.

- void XOS_GetMetaDataAddr (XOS *InstancePtr, u32 *Addr1Ptr, u32 *Addr2Ptr)

  This function fetches output meta data frame buffer addresses for an OS device.

- void XOS_SetLabelMaskAddr (XOS *InstancePtr, u32 Addr1)

  This function sets up the output label mask data frame buffer addresses for an OS device.

- void XOS_GetLabelMaskAddr (XOS *InstancePtr, u32 *Addr1Ptr)

  This function fetches the output label mask data frame buffer addresses for an OS device.

- void XOS_FlipMetaDataAddr (XOS *InstancePtr)

  This function flips the meta data output buffer for an OS device.

- u32 * XOS_GetReadyMetaDataAddr (XOS *InstancePtr)

  This function returns the active meta data output buffer address for an OS device.

- void XOS_SetBlock (XOS *InstancePtr, XOS_DimensionCfg *DimensionCfgPtr)

Discontinued IP

This function sets up dimension related configuration information used by an OS device.

- void XOS_SetFeatureSelectWriteBankAddr (XOS *InstancePtr, u8 BankIndex)

  This function sets the feature select write bank address to be used by an OS device.

- void XOS_GetFeatureSelectWriteBankAddr (XOS *InstancePtr, u8 *BankIndex)

  This function fetches the feature select write bank address being used by an OS device.

- void XOS_SetFeatureSelectBank (XOS *InstancePtr, u8 *BankData)

  This function loads a feature select bank to be used by an OS device.

- void XOS_SetFeatureSelectActiveBankAddr (XOS *InstancePtr, u8 BankIndex)

  This function sets the feature select active bank address to be used by an OS device.

- void XOS_GetFeatureSelectActiveBankAddr (XOS *InstancePtr, u8 *BankIndex)

  This function fetches the feature select active bank address being used by an OS device.

- void XOS_LoadFeatureSelectBank (XOS *InstancePtr, u8 BankIndex, u8 *BankData)

  This function loads a feature select bank to be used by an OS device.

- void XOS_SetFeatureCombinationWriteBankAddr (XOS *InstancePtr, u8 BankIndex)

  This function sets the feature combination write bank address to be used by an OS device.

- void XOS_GetFeatureCombinationWriteBankAddr (XOS *InstancePtr, u8 *BankIndex)

  This function fetches the feature combination write bank address being used by an OS device.

- void XOS_SetFeatureCombinationBank (XOS *InstancePtr, u32 *BankData)

  This function loads a feature combination bank to be used by an OS device.

- void XOS_SetFeatureCombinationActiveBankAddr (XOS *InstancePtr, u8 BankIndex)

  This function sets the feature combination active bank address to be used by an OS device.

- void XOS_GetFeatureCombinationActiveBankAddr (XOS *InstancePtr, u8 *BankIndex)

  This function fetches the feature combination active bank address being used by an OS device.

Discontinued IP

- void XOS_LoadFeatureCombinationBank (XOS *InstancePtr, u8 BankIndex, u32 *BankData)

  This function loads a feature combination bank to be used by an OS device.

- void XOS_GetVersion (XOS *InstancePtr, u16 *Major, u16 *Minor, u16 *Revision)

  This function returns the version of an OS device.

Functions in xos_sinit.c

- XOS_Config * XOS_LookupConfig (u16 DeviceId)

  XOS_LookupConfig returns a reference to an XOS_Config structure based on the unique device id, DeviceId.

Functions in xos_intr.c

- void XOS_IntrHandler (void *InstancePtr)

  This function is the interrupt handler for the Object Segmentation driver.

- int XOS_SetCallBack (XOS *InstancePtr, u32 HandlerType, void *CallBackFunc, void *CallBackRef)

  This routine installs an asynchronous callback function for the given HandlerType:.

Discontinued IP

Appendix E

Additional Resources

Discontinued IP

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.

Note: The glossary also contains acronyms.

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.

List of Acronyms

Table E-1: **List of Acronyms**

Acronym	Description
AMBA	Advanced Microcontroller Bus Architecture
API	Application Program Interface
AXI	Advanced eXtensible Interface
DSP	Digital Signal Processing
EDK	Embedded Development Kit
FF	Flip-Flop
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
GUI	Graphical User Interface

Table E-1: **List of Acronyms**

Acronym	Description
HDL	Hardware Description Language
I/O	Input/Output
IP	Intellectual Property
ISE	Integrated Software Environment
LSB	Least Significant Bit
LUT	Lookup Table
MHz	Mega Hertz
MM2S	Memory Map to Stream
MSB	Most Significant Bit
OSD	On Screen Display
R	Read
R/W	Read/Write
RAM	Random Access Memory
S2MM	Stream to Memory Map
VHDL	VHSIC Hardware Description Language (VHSIC an acronym for Very High-Speed Integrated Circuits)
XPS	Xilinx Platform Studio (part of the EDK software)
XST	Xilinx Synthesis Technology

Discontinued IP

References

These documents provide supplemental material useful with this user guide:

1. *AMBA® AXI4-Stream Protocol Specification*
2. UG761, *AXI Reference Guide*
3. DS768, *AXI Interconnect IP Data Sheet*
4. PG015, *LogiCORE IP Image Characterization Product Guide*
5. UG934, *AXI4-Stream Video IP and System Design Guide*
6. Vivado Design Suite Migration Methodology Guide (UG911)
7. Vivado™ Design Suite user documentation

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

Technical Support

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

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

- New Features
- Resolved Issues
- Known Issues

Discontinued IP

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/11	1.0	Initial Xilinx release.
12/18/12	2.0	Updated core version. Added Vivado section. Updated Debugging Appendix.

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2011 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA is a registered trademark of ARM in the EU and other countries. All other trademarks are the property of their respective owners.

Discontinued IP