LogiCORE IP Object Segmentation v1.0

DS832 March 1, 2011 **Product Specification**

Introduction

The Xilinx® LogiCORE™ 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 set when using the General Purpose Processor configuration or by using the supplied software drivers when using the 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 (up to 720P60 and 1080P30)
- Selectable processor interface
    - EDK pCore
    - General Purpose Processor
- AXI4 memory mapped interface; AXI4-Lite processor interface (EDK pCore)
- For use with Xilinx CORE Generator™ tool 13.1 or later

LogiCORE IP Facts Table					
Core Specifics					
Supported Device Family[1]	Virtex-6, Spartan-6				
Supported User Interfaces	AXI4, AXI4-Lite, General Purpose Processor				
	Resources				**Frequency**
	LUTs	**FFs**	**DSP Slices**	**Block RAMs**	**Max. Freq.**
	See Table 18 and Table 19				
Provided with Core					
Documentation	Product Specification				
Design Files	Netlist or EDK pCore				
Example Design	Not Provided				
Test Bench	Provided on the product page				
Constraints File	Not Provided				
Simulation Model	VHDL or Verilog Structural model C model provided on the product page				
Tested Design Tools					
Design Entry Tools	ISE 13.1, XPS 13.1				
Simulation	QuestaSim 6.6d				
Synthesis Tools	XST 13.1				
Support					
Provided by Xilinx, Inc.					

1. For a complete listing of supported devices, see the release notes for this core.

Applications

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

Overview

Xilinx enables video analytics systems with three IP cores that provide the computationally-intensive pixel level processing implemented in hardware. This approach enables video analytics solutions to operate at high-definition resolutions and full-frame rates (720P60 and 1080P30). The IP cores (Motion Adaptive Noise Reduction, Image Characterization and Object Segmentation) produce object Metadata for subsequent processing by a system processor or other processing block, eliminating the burden of pixel processing for these components.

In the present implementation, objects are defined as a rectangular region that matches a set of defined object characteristics as seen in Figure 1. The Xilinx Object Segmentation core plays a key role in the Xilinx Video Analytics system. It is responsible for parsing a data structure that describes the characteristics of an image and then segmenting out 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 video analytics system 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 now considered an object block. After all of the image blocks have been tested, the video analytics system then begins the process of aggregating the object blocks into full objects. Two blocks are aggregated together if they are neighbors horizontally, vertically or diagonally. After all the object blocks have been aggregated into objects, each object is analyzed to define a box that will completely bound the object. The bounding box now defines the object. The final step is to generate Metadata which consists of a list of all the objects that were found in the image. This Metadata can then be read by a software application which performs higher level analysis and processing.

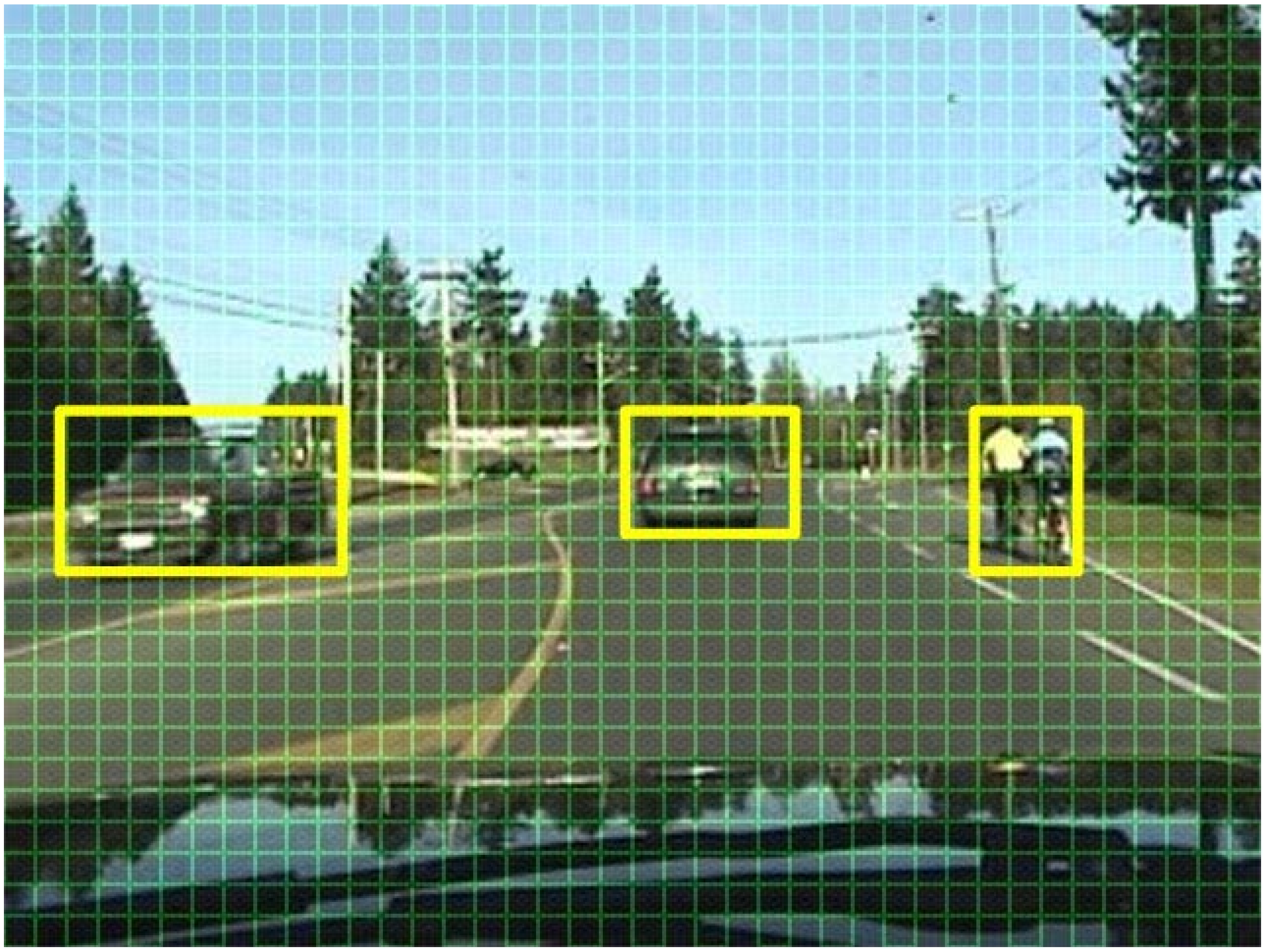


Figure 1: **Object Segmentation Image View**

Architecture

A high-level view of the Object Segmentation core can be seen in the block diagram shown in Figure 2. The core uses an AXI4 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.

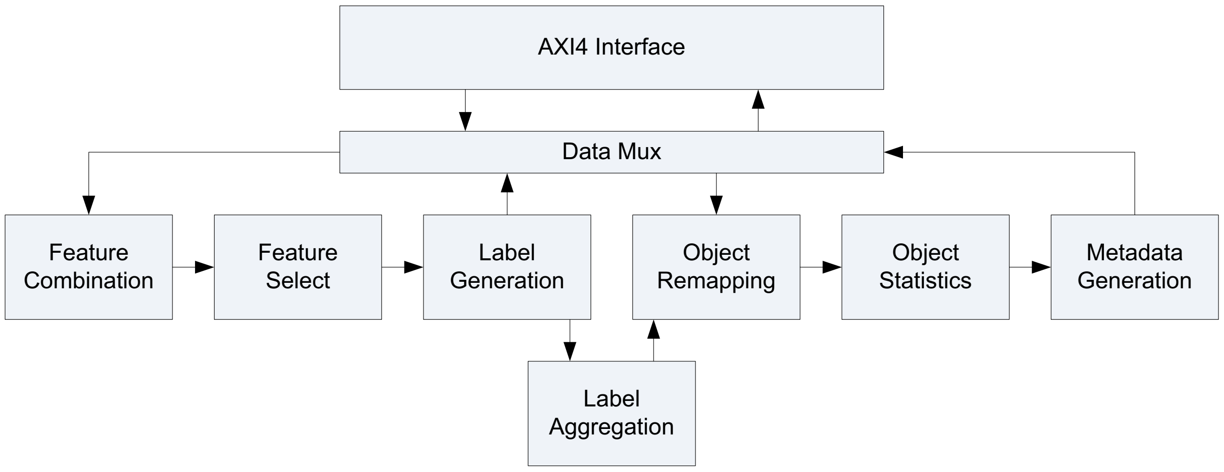


Figure 2: **Object Segmentation Block Diagram**

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. Notice that the Feature Combination Lower Global Thresholds and Upper Global Thresholds in Table 2 match the Image Characterization IP core Global Statistics ouput as shown in Table 6. Additionally, the Feature Combination Lower Block Thresholds and Upper Block Thresholds in Table 2 match the Image Characterization IP core output Block Statistics as shown in Table 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 in 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.

Feature Select

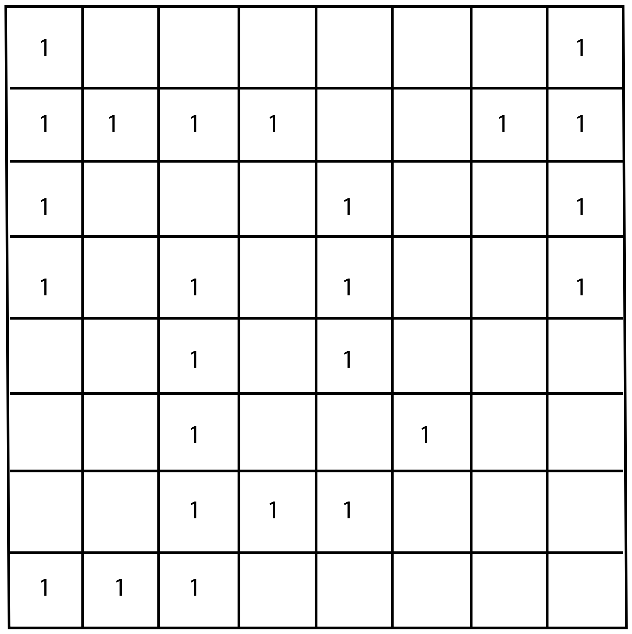
The Feature Select block takes the eight 1-bit values from the Feature Combination block and transforms them into a Feature Select. A Feature Select is defined as any logical equation using the Feature Combination results as terms in the equation.

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 the Feature Select Data Structure section 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.

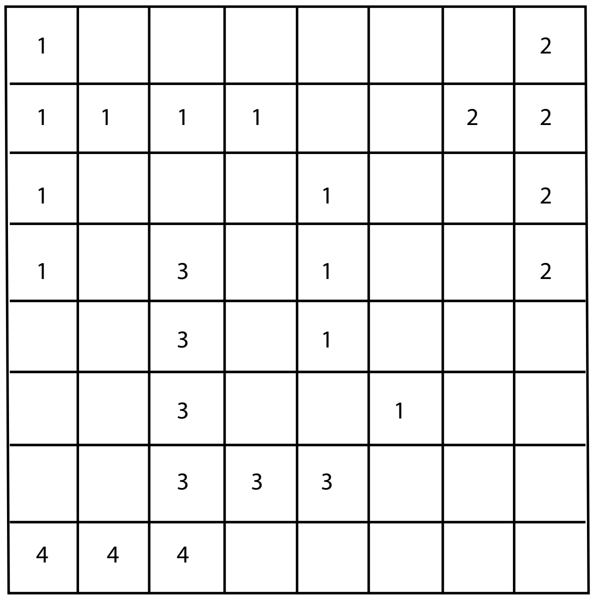
Label Generation

The Label Generation block is used to aggregate together 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 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 4 shows the results of the Label Generation processing. The Feature Select data is aggregated into four sets of labels. Notice that labels 1, 3 and 4 are neighbors.



1							1
1	1	1	1			1	1
1				1			1
1		1		1			1
		1		1			
		1			1		
		1	1	1			
1	1	1					

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



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 4: **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.

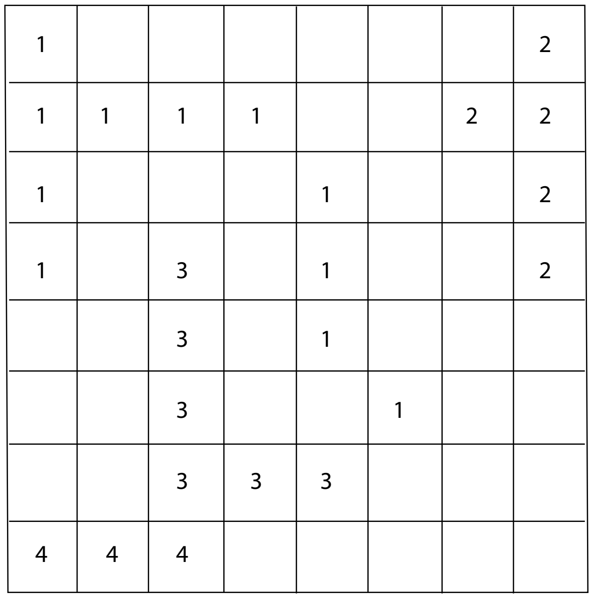
Label Aggregation

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.

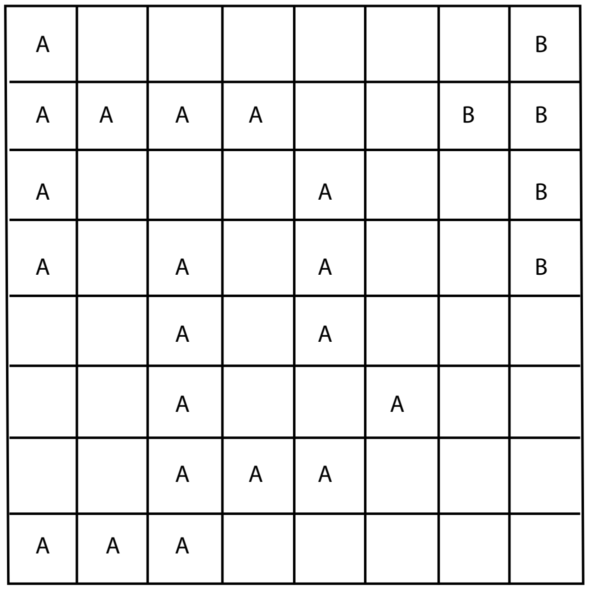
Object Remapping

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 5 and Figure 6 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 5: **Label Data for an 8x8 Frame**



A							B
A	A	A	A			B	B
A				A			B
A		A		A			B
		A		A			
		A			A		
		A	A	A			
A	A	A					

Figure 6: **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 6, 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.

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 from one 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 so the thresholds can be any value from 0 to 255. The global and block variance ("_var") statistics are 16-bit values so 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 1 shows the possible value ranges for the color_selects.

Table 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 2 shows the format of the Feature Combination Threshold data structure and the order that 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 the Feature Combination Bank Programming section for more details.

Table 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	
0x10	Reserved			
0x11	Reserved			
0x12	Reserved			
0x13	Reserved			

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

	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			

Feature Select Data Structure

The Object Segmentation core supports from one 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 lsb and Feature Combination 8 is the 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 RAM output data as illustrated in Table 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 the Feature Select Bank Programming section for more details.

Table 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 (See Table 5)
- Global Statistics and Histograms (See Table 6)
- Block Statistics (See Table 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
- Variance ("_var") - 16-bits
- Histogram - 32-bits (21-bits actual)
- Color_Select - 16-bits (12-bits actual)
- PAD - 32-bits (0x0000)

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

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

Table 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)			

Table 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 9. The Structure_Valid entry specifies if the entire frame has been written and is valid to be used. It will hold 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 will be written.

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

Table 9: **Object Segmentation Metadata Header *(Cont'd)***

Saturation_Var	Motion_Var
PAD (x22)	

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

Table 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			

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

Core Interfaces

AXI4 Memory Interface

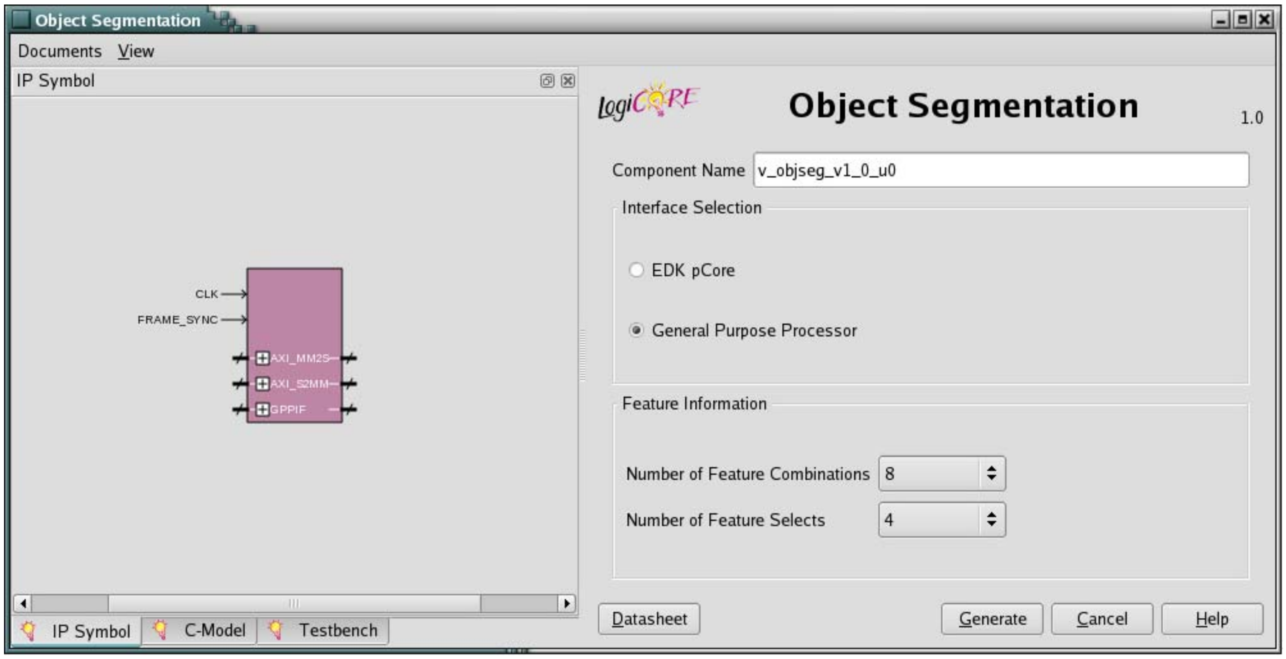
The Object Segmentation core uses a AXI4 interface to connect to the AXI4 Interconnect. The AXI4 Interconnect provides the access to external memory. The core provides registers that allow the user to specify the location in memory of the various data buffers that the Object Segmentation core accesses. See Table 12 for more details on these registers.

Processor Interface

There are many video systems developed that use an integrated processor system to dynamically control the parameters within the system. This is especially important when several independent image processing cores are integrated into a single FPGA. The Object Segmentation core can be configured with one of two interfaces: an EDK pCore Interface or a General Purpose Processor Interface.

CORE Generator Software Graphical User Interface (GUI)

The Xilinx Image Characterization core is easily configured to meet the developer's specific needs through the CORE Generator software 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 7.



Component Name: v_objseg_v1_0_u0

Interface Selection

EDK pCore [✓ General Purpose Processor]

Feature Information

Number of Feature Combinations: 8

Number of Feature Selects: 4

Figure 7: **Object Segmentation CORE Generator GUI**

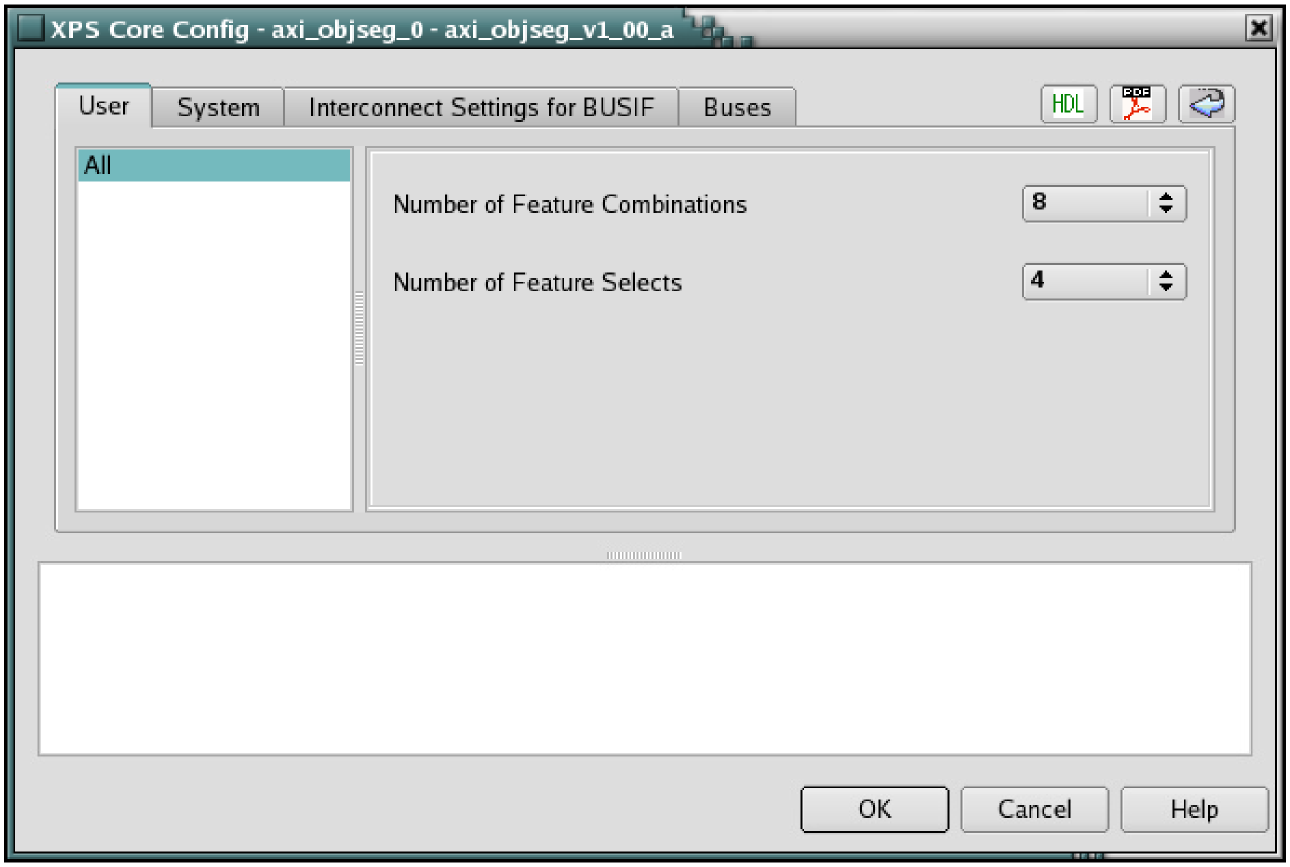
The screen displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, described as follows:

- **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_v1_0" is not allowed.
- **Interface Selection**: The Image Characterization core is generated with one of two processor interfaces.
  - **EDK pCore Interface**: CORE Generator software generates the core as a pCore that can be easily imported into an EDK project as a hardware peripheral. The core registers can then be programmed in real-time via an embedded microprocessor. See the EDK pCore Interface section for details. When the EDK pCore is selected, the rest of the options are disabled and set to the default value. All modifications to the Object Segmentation pCore are made with the EDK GUI.
  - **General Purpose Processor Interface**: CORE Generator software generates a set of ports that can be used to program the core. See the General Purpose Processor Interface section for details. When the General Purpose Processor interface is selected, the rest of the configuration options become active and can be used to generate a customized Object Segmentation core.
- **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.

EDK pCore Graphical User Interface (GUI)

When the Xilinx Object Segmentation core is generated from the CORE Generator software as an EDK pCore, 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 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 the CORE Generator Software Graphical User Interface (GUI) section for details about each option.


XPS Core Config - axi_objseg_0 - axi_objseg_v1_00_a
User
System
Interconnect Settings for BUSIF
Buses
HDL
All
OK
Cancel
Help

Number of Feature Combinations: 8

Number of Feature Selects: 4

Figure 8: **Object Segmentation pCore GUI**

Common I/O Signals

The EDK pCore interface and the General Purpose Processor interface share a number of the same I/O signals. The signals that both interfaces share are specified in Table 11.

Table 11: **Common I/O Signals**

Pin Name	Dir	Width	Description
Core Signals			
clk	I	1	Core clock
frame_sync	I	1	Frame Synchronization
AXI4 Interface Clocks			
m_axi_mm2s_aclk	I	1	MM2S Async clock
m_axi_s2mm_aclk	I	1	S2MM Async clock
AXI4 MM2S Read Address Channel			
m_axi_mm2s_araddr	O	32	MM2S Read Address
m_axi_mm2s_arlen	O	8	MM2S Read Length Qualifier
m_axi_mm2s_arsize	O	3	MM2S Read Size Qualifier
m_axi_mm2s_arburst	O	2	MM2S Read Burst Type Qualifier
m_axi_mm2s_arprot	O	3	MM2S Read Protection Qualifier
m_axi_mm2s_arcache	O	4	MM2S Read Cache Qualifier
m_axi_mm2s_arvalid	O	1	MM2S Read Address Valid Qualifier
m_axi_mm2s_arready	I	1	MM2S Read Address Ready Status
AXI4 MM2S Read Data Channel			
m_axi_mm2s_rdata	I	32	MM2S Read Data
m_axi_mm2s_rresp	I	2	MM2S Read Response
m_axi_mm2s_rlast	I	1	MM2S Read Last Indication
m_axi_mm2s_rvalid	I	1	MM2S Read Valid Handshake
m_axi_mm2s_rready	O	1	MM2S Read Ready Handshake
AXI4 S2MM Write Address Channel			
m_axi_s2mm_awaddr	O	32	S2MM Write Address
m_axi_s2mm_awlen	O	8	S2MM Write Length Qualifier
m_axi_s2mm_awsize	O	3	S2MM Write Size Qualifier
m_axi_s2mm_awburst	O	2	S2MM Write Burst Type Qualifier
m_axi_s2mm_awprot	O	3	S2MM Write Protection Qualifier
m_axi_s2mm_awcache	O	4	S2MM Write Cache Qualifier
m_axi_s2mm_awvalid	O	1	S2MM Write Address Valid Qualifier
m_axi_s2mm_awready	I	1	S2MM Write Address Ready Qualifier
AXI4 S2MM Write Data Channel			
m_axi_s2mm_wdata	O	32	S2MM Write Data
m_axi_s2mm_wstrb	O	4	S2MM Write Strobes
m_axi_s2mm_wlast	O	1	S2MM Write Last Indication
m_axi_s2mm_wvalid	O	1	S2MM Write Valid Handshake

Table 11: **Common I/O Signals *(Cont'd)***

m_axi_s2mm_wready	I	1	S2MM Write Ready Handshake
AXI4 S2MM Write Response Channel			
m_axi_s2mm_bresp	I	2	S2MM Write Response Data
m_axi_s2mm_bvalid	I	1	S2MM Write Response Valid Handshake
m_axi_s2mm_bready	O	1	S2MM Write Response Ready Handshake

EDK pCore Interface

The pCore interface creates a core that can be easily added to an EDK Project as a hardware peripheral. This section describes the Register Set, the pCore Driver Files, and the I/O signals associated with the Object Segmentation pCore.

When generated by CORE Generator software, the new pCore is located in the CORE Generator software project directory at <Component_Name>/pcores/axi_objseg_v1_00_a. The pCore should be copied to the user's <EDK_Project>/pcores directory or to a user pCores repository. The Object Segmentation pCore driver software is located in the CORE Generator project directory at <Component_Name>/drivers/os_v1_01_a. The driver software should be copied to the user's <EDK_Project>/drivers directory or to a user pCores repository.

Pcore Register Set

The pCore interface provides a memory-mapped interface for the programmable registers within the core, which are defined in Table 12.

Table 12: **Object Segmentation pCore Memory Mapped Register Set**

Address (hex) BASEADDR +	**Register Name**	**Access Type**	**Description**	
0x0000	Control	R/W	Control Register	
			31:3	Reserved
			2	Meta Data Address Selection 0 = Meta Data Start Addr 0, 1 = Meta Data Start Addr 1
			1	Register Update Enable. This bit communicates to the IP Core to take new values at the next frame_sync rising edge. Usage: This bit is cleared when the IP Core next frame_sync happens.
			0	Enable the Object Segmentation core on the next frame_sync
0x0004	Status	R	Status Register	
			31:1	Reserved
			0	Meta Data Address. Specifies which buffer is actively being written to: 0 = Meta Data Start Addr 0, 1 = Meta Data Start Addr 1

Table 12: **Object Segmentation pCore Memory Mapped Register Set *(Cont'd)***

0x0008	Status Error	R	Status Register for Errors	
			31:3	Reserved
			2	MM2S Error. This active high signal is asserted whenever a Error condition is encountered within the MM2S.
			1	S2MM Error. This active high signal is asserted whenever a Error condition is encountered within the S2MM.
			0	Frame Error The core did not finish before the beginning of the next frame. Usage: This bit is cleared when any value is written to the register.
0x000C	Status Done	R	General read register for status done	
			31:1	Reserved
			0	Frame Done Done bit can be polled by software for end of object segmentation operation. Usage: This bit is cleared when any value is written to the register
0x0010	Image Characterization Start Address 0	R/W	Starting address for input Image Characterization Buffer 0	
			31:0	
0x0014	Image Characterization Start Address 1	R/W	Starting address for input Image Characterization Buffer 1	
			31:0	
0x0018	Metadata Start Address 0	R/W	Starting address for output Metadata Buffer 0	
			31:0	
0x001C	Metadata Start Address 1	R/W	Offset address for output Metadata Buffer 1	
			31:0	
0x0020	Label Mask Start Address 0	R/W	Starting address for label mask output	
			31:0	
0x0024	Reserved			
0x0028	Reserved			
0x002C	Reserved			
0x0030	Feature Select Write Bank Address	R/W	Bank address of feature select being written	
			31:3	Reserved
			2:0	Bank address to which the Feature Select Data will be written
0x0034	Feature Select Data	R/W	Feature Select input data truth table	
			31:4	Reserved
			3:0	Data for Feature Select truth table, 256 values per bank

Table 12: **Object Segmentation pCore Memory Mapped Register Set *(Cont'd)***

0x0038	Feature Select Active Bank Address	R/W	Active Feature Select Bank	
			31:3	Reserved
			2:0	Feature Select Bank for next frame
0x003C	Feature Combination Write Bank Addr	R/W	bank address of feature combination data being written	
			31:4	Reserved
			3	Corresponds to Feature Combination Active Bank Address
			2:0	Write Feature Combination Bank Address internal to core for feature combination 0 - 7.
0x0040	Feature Combination Data	R/W	Feature Combination input data for thresholds	
			31:0	Data for Feature Combination Thresholds, 40 values per bank.
0x0044	Feature Combination Active Bank Addr	R/W	Active feature combination bank for next frame	
			31:1	Reserved
			0	Active Feature Combination Bank to be use for the next frame.
0x0048	Number of Horizontal Blocks	R/W	Number of horizontal blocks in the input data set	
			31:10	Reserved
			9:0	Number of horizontal blocks in the system, that is, horizontal resolution divided by block size
0x004C	Number of Vertical Blocks	R/W	Number of vertical blocks in the input data set	
			31:10	Reserved
			9:0	Number of vertical blocks in the system, that is, vertical resolution divided by block size
0x0050	Number of Total Blocks	R/W	Number of total blocks in the input data set	
			31:20	Reserved
			19:0	Number of total blocks in the system, that is, number of horizontal blocks * number of vertical blocks
0x0054	Bock Size	R/W	Block size of VA system	
			31:8	Reserved
			7:0	Block Size
0x00F0	Version Register	R	Version Register	
			31:28	Version Major
			27:20	Version Minor
			19:16	Version Revision
			15:0	Reserved
0x0100	Software Reset	R/W	Software Reset	
			31:1	Reserved
			0	1 = reset core

Table 12: **Object Segmentation pCore Memory Mapped Register Set *(Cont'd)***

0x021C	GIER	R/W	Global Interrupt Enable	
			31	Mask to enable global interrupts
			30:0	Reserved
0x0220	ISR	R/W	Interrupt Status Register Read to determine the source of the interrupt Write to clear the interrupt	
			31:4	Reserved
			3	Edge sensitive interrupt for MM2S Error
			2	Edge sensitive interrupt for S2MM Error
			1	Edge sensitive interrupt for Frame Done
			0	Edge sensitive interrupt for Frame Error
0x0228	IER	R/W	Interrupt Enable Register 0 = mask out an interrupt 1 = enable an interrupt	
			31:4	Reserved
			3	Mask or Enable for MM2S Error
			2	Mask or Enable for S2MM Error
			1	Mask or Enable for Frame Done
			0	Mask or Enable for Frame Error

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 the previous section.

Table 13 lists the files included with the Object Segmentation pCore driver.

Table 13: **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 I/O Signals

The I/O signals for the Object Segmentation pCore are shown in Figure 9. The signals can be broken into two groups: I/O signals and AXI4-Lite signals. The I/O signals are specified in Table 11. The AXI4-Lite signals are specified in Table 14.

Core Signals	
clk	
frame_sync	
AXI4 MM2S Interface	
m_axi_mm2s_aclk	m_axi_mm2s_araddr
m_axi_mm2s_arready	m_axi_mm2s_arlen
m_axi_mm2s_rdata	m_axi_mm2s_arsize
m_axi_mm2s_rresp	m_axi_mm2s_arburst
m_axi_mm2s_rlast	m_axi_mm2s_arprot
m_axi_mm2s_rvalid	m_axi_mm2s_arcache
m_axi_mm2s_rrready	m_axi_mm2s_arvalid
AXI4 S2MM Interface	
m_axi_s2mm_aclk	m_axi_s2mm_awaddr
m_axi_s2mm_awready	m_axi_s2mm_awlen
m_axi_s2mm_wready	m_axi_s2mm_awsize
m_axi_s2mm_bresp	m_axi_s2mm_awburst
m_axi_s2mm_bvalid	m_axi_s2mm_awprot
	m_axi_s2mm_awcache
	m_axi_s2mm_awvalid
	m_axi_s2mm_wdata
	m_axi_s2mm_wstrb
	m_axi_s2mm_wlast
	m_axi_s2mm_wvalid
	m_axi_s2mm_bready
AXI4-Lite Interface	
S_AXI_ACLK	IP2INTC_Irpt
S_AXI_ARESETN	S_AXI_AWREADY
S_AXI_AWADDR	S_AXI_BRESP
S_AXI_AWVALID	S_AXI_BVALID
S_AXI_WDATA	S_AXI_ARREADY
S_AXI_WSTRB	S_AXI_RDATA
S_AXI_WVALID	S_AXI_RRESP
S_AXI_BREADY	S_AXI_RVALID
S_AXI_ARADDR	
S_AXI_ARVALID	
S_AXI_RREADY	

Figure 9: **pCore I/O Diagram**

Table 14: **AXI4-Lite pCore I/O Signals**

Pin Name	Dir	Width	Description
AXI4-Lite Global System Signals [(1)]			
S_AXI_ACLK	I	1	AXI4-Lite Clock
S_AXI_ARESETN	I	1	AXI4-Lite Reset, active low
IP2INTC_Irpt	O	1	Interrupt request output
AXI4-Lite Write Address Channel Signals [(1)]			
S_AXI_AWADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Write Address Bus. The write address bus gives the address of the write transaction.
S_AXI_AWVALID	I	1	AXI4-Lite Write Address Channel Write Address Valid. This signal indicates that valid write address is available. • 1 = Write address is valid. • 0 = Write address is not valid.
S_AXI_AWREADY	O	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates core is ready to accept the write address. • 1 = Ready to accept address. • 0 = Not ready to accept address.
AXI4-Lite Write Data Channel Signals [(1)]			
S_AXI_WDATA	I	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Write Data Bus.
S_AXI_WSTRB	I	[C_S_AXI_DATA_WIDTH/8-1:0]	AXI4-Lite Write Strobes. This signal indicates which byte lanes to update in memory.
S_AXI_WVALID	I	1	AXI4-Lite Write Data Channel Write Data Valid. This signal indicates that valid write data and strobes are available. • 1 = Write data/strobes are valid. • 0 = Write data/strobes are not valid.
S_AXI_WREADY	O	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates core is ready to accept the write data. • 1 = Ready to accept data. • 0 = Not ready to accept data.
AXI4-Lite Write Response Channel Signals [(1)]			
S_AXI_BRESP [(2)])	O	[1:0]	AXI4-Lite Write Response Channel. Indicates results of the write transfer. • 00b = OKAY - Normal access has been successful. • 01b = EXOKAY - Not supported. • 10b = SLVERR - Error. • 11b = DECERR - Not supported.

Table 14: **AXI4-Lite pCore I/O Signals *(Cont'd)***

S_AXI_BVALID	O	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid. • 1 = Response is valid. • 0 = Response is not valid.
S_AXI_BREADY	I	1	AXI4-Lite Write Response Channel Ready. Indicates Master is ready to receive response. • 1 = Ready to receive response. • 0 = Not ready to receive response.
AXI4-Lite Read Address Channel Signals (1)			
S_AXI_ARADDR	I	[(C_S_AXI_ADDR_WIDTH-1):0]	AXI4-Lite Read Address Bus. The read address bus gives the address of a read transaction
S_AXI_ARVALID	I	1	AXI4-Lite Read Address Channel Read Address Valid. • 1 = Read address is valid. • 0 = Read address is not valid.
S_AXI_ARREADY	O	1	AXI4-Lite Read Address Channel Read Address Ready. Indicates core is ready to accept the read address. • 1 = Ready to accept address. • 0 = Not ready to accept address.
AXI4-Lite Read Data Channel Signals (1)			
S_AXI_RDATA	O	[(C_S_AXI_DATA_WIDTH-1):0]	AXI4-Lite Read Data Bus.
S_AXI_RRESP (2)	O	[1:0]	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer. • 00b = OKAY - Normal access has been successful. • 01b = EXOKAY - Not supported. • 10b = SLVERR - Error. • 11b = DECERR - Not supported.
S_AXI_RVALID	O	1	AXI4-Lite Read Data Channel Read Data Valid. This signal indicates that the required read data is available and the read transfer can complete. 1 = Read data is valid. 0 = Read data is not valid.
S_AXI_RREADY	I	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates master is ready to accept the read data. • 1 = Ready to accept data. • 0 = Not ready to accept data.

1. The function and timing of these signals are defined in the *AMBA® AXI Protocol Version: 2.0 Specification*.
2. For signals S_AXI_RRESP[1:0] and S_AXI_BRESP[1:0], the core does not generate the Decode Error ('11') response. Other responses like '00' (OKAY) and '10' (SLVERR) are generated by the core based upon certain conditions.

General Purpose Processor Interface

The other interface option is the General Purpose Processor (GPP) interface. The GPP Interface is shown in Figure 10 and consists of the Common I/O signals listed in Table 11 and the Dynamic Configuration Interface signals detailed in Table 15. The signals in Table 15 correspond to the registers in Table 12.

The directly exposed Dynamic Configuration Interface signals allow the user to wrap these signals with a user-defined bus interface targeting any arbitrary processor. It is recommended to disable the `control[1]` (Register Update enable) signal of the control bus before updating the other Dynamic Configuration Interface signals. After the Dynamic Configuration Interface signals are ready to be updated in the core, the control[1] signal should be enabled. Values are written into the core on the falling edge of the `frame_sync` input.

Core Signals	
clk	
ce	
sclr	
frame_sync	
AXI4 MM2S Interface	
m_axi_mm2s_aclk	m_axi_mm2s_araddr
m_axi_mm2s_arready	m_axi_mm2s_arlen
m_axi_mm2s_rdata	m_axi_mm2s_arsize
m_axi_mm2s_rresp	m_axi_mm2s_arburst
m_axi_mm2s_rlast	m_axi_mm2s_arprot
m_axi_mm2s_rvalid	m_axi_mm2s_arcache
m_axi_mm2s_rready	m_axi_mm2s_arvalid
AXI4 S2MM Interface	
m_axi_s2mm_aclk	m_axi_s2mm_awaddr
m_axi_s2mm_awready	m_axi_s2mm_awlen
m_axi_s2mm_wready	m_axi_s2mm_awsize
m_axi_s2mm_bresp	m_axi_s2mm_awburst
m_axi_s2mm_bvalid	m_axi_s2mm_awprot
	m_axi_s2mm_awcache
	m_axi_s2mm_awvalid
	m_axi_s2mm_wdata
	m_axi_s2mm_wstrb
	m_axi_s2mm_wlast
	m_axi_s2mm_wvalid
	m_axi_s2mm_bready
Dynamic Configuration Interface	
control	reg_update_done
image_char_start_addr0	status_done
image_char_start_addr1	status_error
meta_data_start_addr0	mm2s_err
meta_data_start_addr1	s2mm_err
label_mask_start_addr	status
feature_select_write_bank_addr	version
feature_select_write_bank_addr_we	
feature_select_data	
feature_select_we	
feature_select_active_bank_addr	
feature_combination_write_bank_addr_we	
feature_combination_data	
feature_combination_we	
feature_combination_active_bank_addr	
num_h_blocks	
num_v_blocks	
num_total_blocks	
block_size	

Figure 10: **General Purpose Processor I/O Diagram**

Table 15: **Dynamic Configuration Interface Signals**

Pin Name	Dir	Width	Description	
		Control		
ce	I	1	Clock Enable	
sclr	I	1	Synchronous Clear	
reg_update_done	O	1	Register Update Done	
status_done	O	1	Frame Done	
status_error	O	1	Frame Error	
mm2s_err	O	1	MM2S Channel Error	
s2mm_err	O	1	S2MM Channel Error	
		Registers		
control	I	3	Control Register	
			2	Metadata Address Selection 0 = Meta_data_start_addr0 1 = Meta_data_start_addr1
			1	Register Update Enable
			0	Core Enable
image_char_start_addr0	I	32	Image Characterization Start Address 0	
image_char_start_addr1	I	32	Image Characterization Start Address 1	
meta_data_start_addr0	I	32	Metadata Start Address 0	
meta_data_start_addr1	I	32	Metadata Start Address 1	
label_mask_start_addr0	I	32	Label Mask Start Address	
feature_select_write_bank_addr	I	3	Feature Select Write Bank Address (0 -7)	
feature_select_write_bank_addr_we	I	1	Feature Select Write Bank Address Write Enable	
feature_select_data	I	4	Feature Select Data	
feature_select_we	I	1	Feature Select Data Write Enable	
feature_select_active_bank_addr	I	3	Feature Select Active Bank Address (0 – 7)	
feature_combination_write_bank_addr	I	4	Feature Combination Write Bank Address	
			3	Corresponds to Feature Combination Active Bank Address
			2:0	Write Feature Combination Bank Address internal to core for feature combination 0 - 7.
feature_combination_write_bank_addr_we	I	1		
feature_combination_data	I	32		
feature_combination_we	I	1		
feature_combination_active_bank_addr	I	1	Feature Combination Active Bank Address	
num_h_blocks	I	10	Number of Horizontal Blocks	
num_v_blocks	I	10	Number of Vertical Blocks	
num_total_blocks	I	20	Total Number of Blocks (H x V)	
block_size	I	8	Block Size	

Table 15: **Dynamic Configuration Interface Signals *(Cont'd)***

status	O	1	Status Register	
			Meta Data Address. Specifies which buffer is active. 0 = Meta_data_start_addr0 1 = Meta_data_start_addr1	
version	O	32	Version Register	
			31:28	Version Major
			27:20	Version Minor
			19:16	Version Revision
			15:0	Reserved

Object Segmentation Control and Timing

The Object Segmentation core provides a great deal of flexibility in its operation through a simple register set. The core's Feature Combinations and Feature Selects define the operation of the core. These features are fully configurable by the user 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 valid data structures as described in the Feature Combination Threshold Data Structure and Feature Select Data Structure sections.

The process of programming the Feature Combinations and Feature Selects is slightly different depending upon whether the core is generated with an EDK pCore interface or a General Purpose Processor interface. Both methods are discussed in the sections that follow. The process of the programming the rest of the register set is essentially the same for both interfaces.

pCore 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.

Loading the Feature Selects should follow these steps:

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.

Loading the Feature Combinations should follow these steps:

1. Set the Feature Combination Write Bank Address to the address (0 - 15) of the Feature Combination bank to be loaded. See Table 12 or Table 15 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.

GPP 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.

Loading the Feature Selects is illustrated in Figure 11 and should follow these steps:

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. The `feature_select_we` signal must be toggled for the data to be written. See the Feature Select Data Structure section for more details about the Feature Select data set.
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.

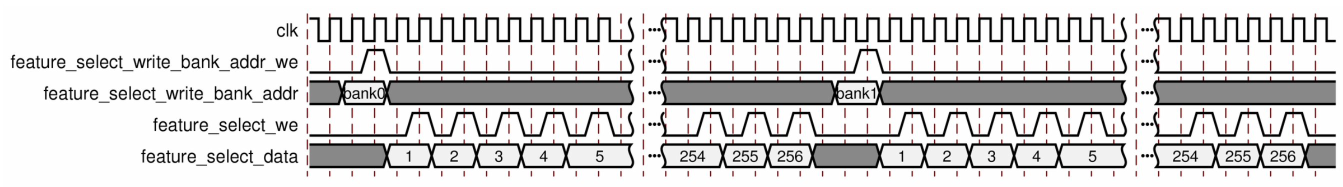

clk
feature_select_write_bank_addr_we
feature_select_write_bank_addr
feature_select_we
feature_select_data
bank0
bank1
1
2
3
4
5
254
255
256

Figure 11: **Feature Select Bank Programming**

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.

Loading the Feature Combinations is illustrated in Figure 12 and should follow these steps:

1. Set the Feature Combination Write Bank Address to the address (0 - 15) of the Feature Combination bank to be loaded. See Table 12 or Table 15 for more detail on the Feature Combination Write Bank Address register.
2. Write 40 Feature Combination data values to the feature_combination_data. The `feature_combination_we` signal must be toggled for the data to be written. See the Feature Combination Threshold Data Structure section for more details on the Feature Combination Threshold data set.
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.

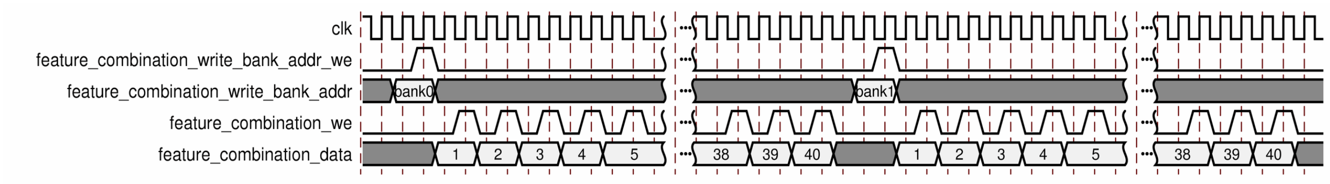

clk
feature_combination_write_bank_addr_we
feature_combination_write_bank_addr
feature_combination_we
feature_combination_data
bank0
bank1
1
2
3
4
5
38
39
40

Figure 12: **Feature Combination Bank Programming**

Register Updates

The Object Segmentation core is controlled by a register set that must be initialized before the core begins processing. See Table 12 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 will 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 register values are written into the core on the next falling edge of the `frame_sync` 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 `frame_sync` signal.

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. The core uses these buffers in a Ping-Pong fashion. At start-up, the buffer specified by register 0 is used for the first frame of processing. For the next frame buffer 1 is used. The core continues to switch between buffers on each successive frame. 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 makes sure that the Object Segmentation core is always processing a valid Image Characterization data structure.

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 "Metadata Address Selection" register (control[2]) 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. The Object Segmentation core is the only user of this buffer so there are synchronization issues with which to be concerned.

Interrupts

The Object Segmentation core can flag four interrupts. The Status Error and Status Done interrupts report the operation of the core's processing. The MM2S Error and S2MM Error interrupts report errors that occurred while transferring data across the AXI4 interface.

Status Error

On the falling edge of `frame_sync` (which signifies the start of the next frame), the Object Segmentation core checks to make sure that all of the Metadata from the previous frame has been written to memory. If any of the data has not been written to memory, then the core flags a status error. When using the General Purpose Processor interface, the error is indicated by the logic '1' state of the `status_error` signal. The `status_error` signal is reset to '0' on the next rising edge of `frame_sync`. When using the pCore interface, the `status_error` signal is used to drive bit 0 of the interrupt controller. It also sets bit 0 of the Status Error Register. The value in the Status Error Register can be reset by writing any value to the register.

Status Done

When the Object Segmentation core finishes writing all of the Metadata to memory, it flags that it has completed processing the current frame. When using the General Purpose Processor interface, the `status_done` signal is set to '1'. The `status_done` signal is reset to '0' on the falling edge of `frame_sync`, which denotes the start of the next frame. When using the pCore interface, the `status_done` signal is used to drive bit 1 of the interrupt controller. It also sets bit 0 of the Status Done Register. The value in the Status Done Register can be reset by writing any value to the register.

MM2S Error

The MM2S Error is asserted whenever an error condition is encountered within the MM2S portion of the AXI4 interface. When using the General Purpose Processor interface, the `mm2s_err` signal is set to '1' when an error is reported. When using the pCore interface, the `mm2s_err` signal is used to drive bit 3 of the interrupt controller. It also sets bit 2 of the Status Error Register.

S2MM Error

The S2MM Error is asserted whenever an error condition is encountered within the S2MM portion of the AXI4 interface. When using the General Purpose Processor interface, the `s2mm_err` signal is set to '1' when an error is reported. When using the pCore interface, the `s2mm_err` signal is used to drive bit 2 of the interrupt controller. It also sets bit 1 of the Status Error Register.

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. Each is described the preceding sections.

Consider the case of using 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, some knowledge about the configuration of the Image Characterization core is necessary as follows:

- 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

Using this information we know 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 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 the purposes of 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 16, 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.

For the FC1 Feature Combination Threshold data structure we set 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

For the FC2 Feature Combination Threshold data structure we set 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 16: **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. This table 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 17. 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.

Table 17: **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 17 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 13 shows an example of 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 (AXI_VDMA, Timing Controller, OSD, etc.)

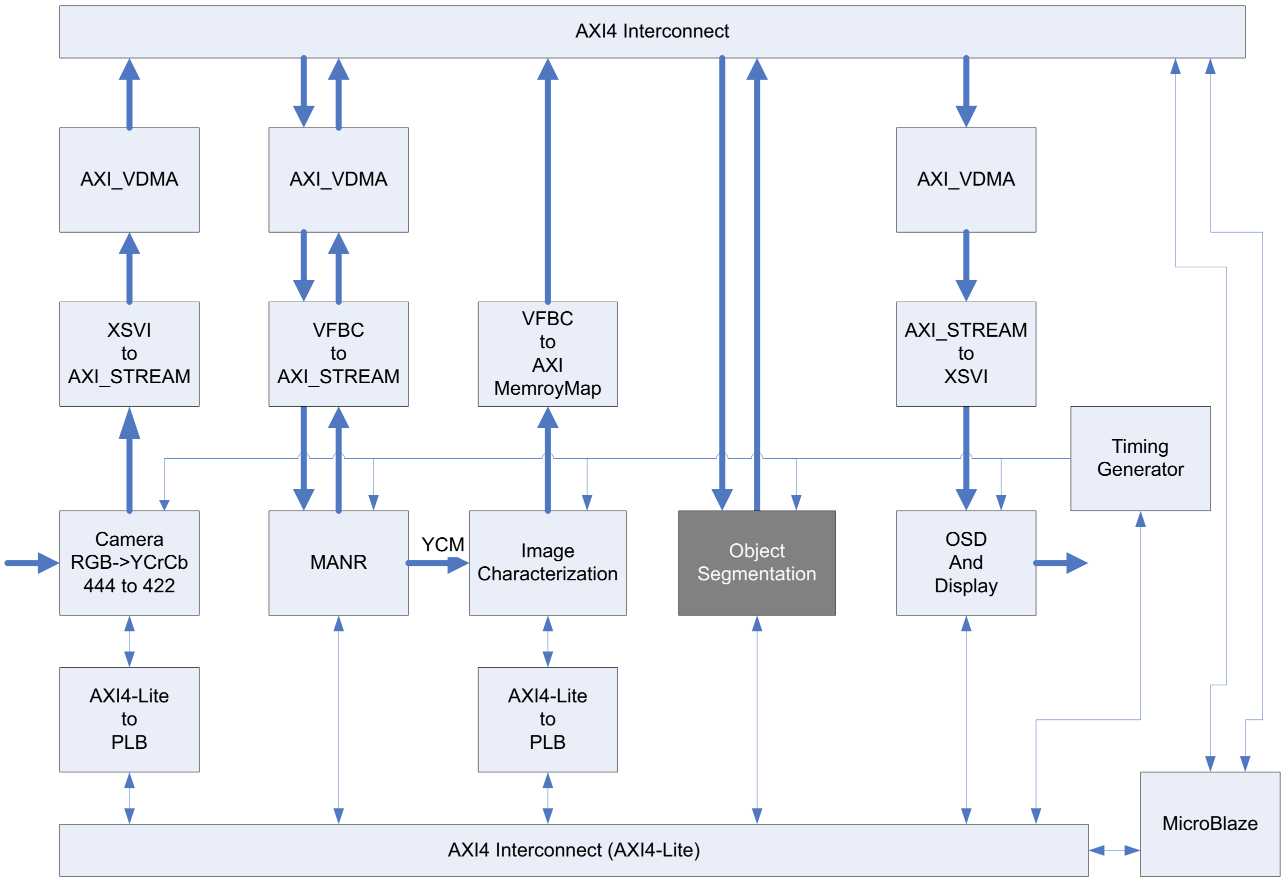


Figure 13: **Object Segmentation Example Use Model**

Core Resource Utilization

Resources required for the Object Segmentation core have been estimated for the Spartan®-6 (Table 18) and Virtex®-6 (Table 19). These values were generated using the Xilinx CORE Generator tools v13.1. They are derived from post-synthesis reports, and might change during MAP and PAR.

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" 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" by the number of extra Feature Selects and add the results to the resource count. If using the pCore Interface, add the corresponding resources to the results count.

Table 18: **Spartan-6 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (16/8)	DSP48A1s
Base Core (Feature Combination = 1, Feature Select = 1)	3284	2971	6/1	4
Each additional Feature Combination	323	85	0/0	0
Each additional Feature Select	795	505	1/0	0
pCore Interface	1900	1650	0/0	0

Table 19: **Virtex-6 Resource Estimates**

Feature	LUTs	FFs	Block RAMs (36/18)	DSP48A1s
Base Core (Feature Combination = 1, Feature Select = 1)	3322	2968	3/2	4
Each additional Feature Combination	326	85	0/0	0
Each additional Feature Select	790	498	0/1	0
pCore Interface	1900	1650	0/0	0

Performance

The following are typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors.

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

References

DS768, *AXI Interconnect IP Data Sheet*

(http://www.xilinx.com/support/documentation/ip_documentation/ds768_axi_interconnect.pdf)

DS727, *LogiCORE IP Image Characterization Data Sheet*

(http://www.xilinx.com/support/documentation/ip_documentation/v_ic_ds727.pdf)

AMBA AXI Protocol Version: 2.0 Specification

Support

Xilinx provides technical 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.

License Options

The Xilinx Image Characterization core provides three licensing options. After installing the required Xilinx ISE® software and IP Service Packs, choose a license option.

Simulation Only

The Simulation Only Evaluation license key is provided with the Xilinx CORE Generator tool. This key lets you assess the core functionality with your own design and demonstrates the various interfaces on the core in simulation. (Functional simulation is supported by a dynamically-generated HDL structural model.)

Full System Hardware Evaluation License

To obtain a Full System Hardware Evaluation license:

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

Obtaining a Full License

To obtain a Full license key, you must purchase a license for the core. After doing so, click the "Access Core" link on the Xilinx.com IP core product page for further instructions.

Installing Your License File

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

Ordering Information

The Object Segmentation v1.0 core is provided under the SignOnce IP Site License and can be generated using the Xilinx CORE Generator system v13.1 or higher. The CORE Generator system is shipped with the Xilinx ISE Design Suite development software. Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

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
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

Revision History

This table shows the revision history for this document.

Date	Version	Description of Revisions
03/01/11	1.0	Initial Xilinx release.

Notice of Disclaimer