LogiCORE™ IP Object Segmentation v1.0 Bit Accurate C Model

User Guide

UG815 April 25, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/25/11	1.0	Initial Xilinx release.

Table of Contents

Appendix A: Object Segmentation Metadata Output

About This Guide

This user guide provides information about the Xilinx® LogiCORE™ IP Object Segmentation v1.0 bit accurate C model for 32-bit Linux, 64-bit Linux, 32-bit Windows, and 64-bit Windows platforms.

Guide Contents

This manual contains the following chapters:

- Chapter 1, Introduction introduces the bit accurate C model for the Xilinx LogiCORE IP Object Segmentation v1.0 core, which has been developed primarily for system level modeling.
- Chapter 2, User Instructions provides information on the C model directory structure, files, installation, and software requirements.
- Chapter 3, Interface provides information on the C model interface, including defining the inputs, generics and output of the Object Segmentation core.
- Chapter 4, C Model Example Code provides an example C file along with the executable for this example.
- Appendix A, Object Segmentation Metadata Output includes an example of the object segmentation metadata output.

Additional Resources

To find additional documentation, see the Xilinx website at:

www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

www.xilinx.com/support/mysupport.htm.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

The Xilinx® LogiCORE™ IP Object Segmentation v1.0 core has a bit accurate C model designed for system modeling.

Features

- Bit accurate with Object Segmentation v1.0 core (v_objseg_v1_0)
- Statically linked library (.lib, .o, .obj - Windows)
- Dynamically linked library (.so – Linux)
- Available for 32-bit Windows, 64-bit Windows, 32-bit Linux, and 64-bit Linux platforms
- Supports all features of the Object Segmentation core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code is provided to show how to use the function
  - Example application C code wrapper files support 8-bit image characterization input only

Overview

The LogiCORE IP Object Segmentation v1.0 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 Chapter 3, 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

Additional Core Resources

For detailed information and updates about the Object Segmentation v1.0 core, see the documents listed on the core product page at:

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

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the Object Segmentation v1.0 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP Object Segmentation Bit Accurate C Model User Guide*).

Xilinx cannot guarantee functionality or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the Object Segmentation v1.0 core and the accompanying documentation.

Object Segmentation v1.0 Bit Accurate C Model and IP Core

For comments or suggestions about the Object Segmentation v1.0 core and bit accurate C model, submit a WebCase from:
http://www.xilinx.com/support/clearexpress/websupport.htm

Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about the documentation for the Object Segmentation v1.0 core and bit accurate C model, submit a WebCase from:
http://www.xilinx.com/support/clearexpress/websupport.htm

Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

XILINX®

Chapter 2

User Instructions

Unpacking and Model Contents

Unzip the `v_objseg_v1_0_bitacc_model.zip` file, containing the bit accurate models for the Object Segmentation IP Core. This creates the directory structure and files in Table 2-1.

Table 2-1: **Directory Structure and Files of the Object Segmentation v1.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
ug815_v_obj_seg.pdf	LogiCORE IP Object Segmentation Bit Accurate C Model User Guide
v_objseg_v1_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image/video container type and support functions
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions
yuv_utils.h	Header file declaring the YUV image/video container type and support functions
image_char_stats_utils.h	Header file declaring the Image Characterization Statistics container type and support functions
run_bitacc_cmodel.c	Example code calling the C model
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 ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_objseg_v1_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_objseg_v1_0_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_objseg_v1_0_bitacc_cmodel.so	Model shared object library

Table 2-1: **Directory Structure and Files of the Object Segmentation v1.0 Bit Accurate C Model**

libstlport.so.5.1	STL library, referenced by libIp_v_ic_v1_1_bitacc_cmodel.so
/win32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms
libIp_v_objseg_v1_0_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_v1_0_bitacc_cmodel.lib	Precompiled library file for Win64 compilation

Installation

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

- libIp_v_objseg_v1_0_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Object Segmentation v1.0 C models were compiled and tested with the software listed in Table 2-2.

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

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

Chapter 3

Interface

The bit accurate C model is accessed through a set of functions and data structures, declared in the header file `v_objseg_v1_0_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_v1_0_generics objseg_generics;
struct  xilinx_ip_v_objseg_v1_0_inputs   objseg_inputs;
struct  xilinx_ip_v_objseg_v1_0_outputs  objseg_outputs
```

The declaration of these structures are in the `v_objseg_v1_0_bitacc_cmodel.h` file.

Calling `xilinx_ip_v_objseg_v1_0_get_default_generics` (and `objseg_generics`) initializes the generics structure with the default values for each element of the structure. 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;
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.

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

```
int xilinx_ip_v_objseg_v1_0_bitacc_simulate(
      struct xilinx_ip_v_objseg_v1_0_generics* generics,
      struct xilinx_ip_v_objseg_v1_0_inputs*   inputs,
      struct xilinx_ip_v_objseg_v1_0_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_v1_0_destroy(
       struct xilinx_ip_v_objseg_v1_0_inputs *input,
       struct xilinx_ip_v_objseg_v1_0_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 ariance
  uint16  m_var;        // Motion variance
  uint16  e_var;        // Edge variance
  uint16  lp_var;       // Low Frequency variance
  uint16  hp_var;       // High Frequency variance
  uint16  sat_var;      // Saturation variance
};

struct block_stats_struct
{
  uint8   y_mean;        // Y mean
  uint8   u_mean;        // U mean
  uint8   v_mean;        // V mean
  uint8   m_mean;        // Motion mean
  uint8   e_mean;        // Edge mean
  uint8   lp_mean;       // Low Frequency mean
  uint8   hp_mean;       // High Frequency mean
  uint8   sat_mean;      // Saturation mean
  uint16  y_var;         // Y variance
  uint16  u_var;         // U variance
  uint16  v_var;         // V 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`, 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 *DS727 - LogiCORE IP Image Characterization v1.1 Data Sheet* 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
  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`.

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. See *DS832 - LogiCORE IP Object Segmentation Data Sheet* for more information.

XILINX®

Chapter 4

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
- 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

# 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.

```
// Intialize 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];
}
```

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);
```

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 v1.0 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_v1_0_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_v1_0_bitacc_cmodel -Wl,-rpath,.
```

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

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

Windows (32-bit and 64-bit)

Precompiled library `v_scaler_v4_0_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_v1_0_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_v1_0_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_v1_0_bitacc_cmodel.so
   - 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
```

XILINX®

Appendix A

Object Segmentation Metadata Output

Image Characterization Statistics Input

The image characterization statistics input is described in the Appendix of the *LogiCORE IP Image Characterization Bit Accurate C Model User Guide* (UG813). For additional information on the image characterization statistics input structure, see the *LogiCORE IP Image Characterization Data Sheet* (DS727).

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 the *LogiCORE IP Object Segmentation Data Sheet* (DS832).

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