LogiCORE™ IP Image Characterization v1.1 Bit Accurate C Model

User Guide

UG813 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: Image Characterization Statistics Input

About This Guide

This user guide provides information about the Xilinx® LogiCORE™ IP Image Characterization v1.1 bit accurate C model for 32-bit and 64-bit Linux platforms and 32-bit 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 Image Characterization v1.1 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 Image Characterization core.
- Chapter 4, C Model Example Code provides an example C file along with the executable for this example.
- Appendix A, Image Characterization Statistics Input provides an example of the image characterization statistics for one image frame.

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.

XILINX®

Chapter 1

Introduction

The Xilinx® LogiCORE™ IP Image Characterization v1.1 core has a bit accurate C model designed for system modeling.

Features

- Bit accurate with Image Characterization v1.1 core (v_ic_v1_1)
- Statically linked library (.lib, .o, .obj)
- Available for 32-bit Windows, 64-bit Windows, 32-bit Linux and 64-bit Linux platforms
- Supports all features of the Image Characterization 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

Overview

The LogiCORE IP Image Characterization v1.1 has a bit accurate C model for 32-bit and 64-bit Windows, as well as 32-bit and 64-bit Linux platforms. The model has an interface consisting of a set of C functions, which reside in a statically link library (shared library). Full details of the interface are provided in 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 Image Characterization Web page at: http://www.xilinx.com/products/ipcenter/EF-DI-IMG-CHAR.htm

Additional Core Resources

For detailed information and updates about the Image Characterization v1.1 core, see the documents listed on the core product page at:
http://www.xilinx.com/products/ipcenter/EF-DI-IMG-CHAR.htm

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the Image Characterization v1.1 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP Image Characterization 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 Image Characterization v1.1 core and the accompanying documentation.

Image Characterization v1.1 Bit Accurate C Model and IP Core

For comments or suggestions about the Image Characterization v1.1 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 Image Characterization v1.1 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_ic_v1_1_bitacc_model.zip` file, containing the bit accurate models for the Image Characterization IP Core. This creates the directory structure and files in Table 2-1.

Table 2-1: **Directory Structure and Files of the Image Characterization v1.1 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
ug813_v_ic.pdf	LogiCORE IP Image Characterization Bit Accurate C Model User Guide
v_ic_v1_1_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
example_420_512x512.yuv	Example YUV input file
ic_config_512.cfg	Example configuration file
/lin32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_ic_v1_1_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1.so.5.1	STL library, referenced by libIp_v_ic_v1_1_bitacc_cmodel.so
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_ic_v1_1_bitacc_cmodel.so	Model shared object library
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

Table 2-1: **Directory Structure and Files of the Image Characterization v1.1 Bit Accurate C Model**

libIp_v_ic_v1_1_bitacc_cmodel.lib	Precompiled library file for Win32 compilation
/win64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms
libIp_v_ic_v1_1_bitacc_cmodel.lib	Precompiled library file for Win64 compilation

Installation

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

- libIp_v_ic_v1_1_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Image Characterization v1.1 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_ic_v1_1_bitacc_cmodel.h`

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

```
struct  xilinx_ip_v_ic_v1_1_generics ic_generics;
struct  xilinx_ip_v_ic_v1_1_inputs   ic_inputs;
struct  xilinx_ip_v_ic_v1_1_outputs  ic_outputs
```

The declaration of the above structures are in the `v_ic_v1_1_bitacc_cmodel.h` file.

Calling `xilinx_ip_v_ic_v1_1_get_default_generics` (and `ic_generics`) initializes the generics structure with the default values for each element of the structure. The generics defaults are:

```
start_frame_index = 1; // Frame index of the first frame
chroma_format = 0;  //Chroma Format 0=4:2:0, 1=4:2:2
frame_width = 1280; // Frame Width
frame_height = 720; // Frame Height
block_size = 8;     // Block Size (4,8,16,32,64)
num_h_blocks = 160; // Number of blocks horizontal
num_v_blocks = 90;  // Number of blocks vertical
gyw_scale = 410;    // Global Y Width scaling
gyh_scale = 728;    // Global Y Height scaling
by_scale = 1024;    // Block Y scaling
bc_scale = 4096;    // Block C scaling
hp_gain = 1;        // High Pass Gain
edge_gain_h = 1;    // Edge Gain Horizontal
edge_gain_v = 1;    // Edge Gain Vertical
edge_gain_r = 1;    // Edge Gain Right Diagonal
edge_gain_l = 1;    // Edge Gain Left Diagonal
hue_lower1 = 0;     // Color Select Thresholds #1
hue_upper1 = 255;
sat_lower1 = 0;
sat_upper1 = 255;
hue_lower2 = 0;     // Color Select Thresholds #2
hue_upper2 = 255;
sat_lower2 = 0;
sat_upper2 = 255;
hue_lower3 = 0;     // Color Select Thresholds #3
hue_upper3 = 255;
sat_lower3 = 0;
sat_upper3 = 255;
hue_lower4 = 0;     // Color Select Thresholds #4
hue_upper4 = 255;
sat_lower4 = 0;
```

```
sat_upper4 = 255;
hue_lower5 = 0;     // Color Select Thresholds #5
hue_upper5 = 255;
sat_lower5 = 0;
sat_upper5 = 255;
hue_lower6 = 0;     // Color Select Thresholds #6
hue_upper6 = 255;
sat_lower6 = 0;
sat_upper6 = 255;
hue_lower7 = 0;     // Color Select Thresholds #7
hue_upper7 = 255;
sat_lower7 = 0;
sat_upper7 = 255;
hue_lower8 = 0;     // Color Select Thresholds #8
hue_upper8 = 255;
sat_lower8 = 0;
sat_upper8 = 255;
use_y_mean_var = 1;    // Calculate Y Mean and Variance Values
use_u_mean_var = 1;    // Calculate U Mean and Variance Values
use_v_mean_var = 1;    // Calculate V Mean and Variance Values
use_mot_mean_var = 1;  // Calculate Motion Mean and Variance Values
use_freq_mean_var = 1; // Calculate Frequency Mean and Variance Values
use_edge_mean_var = 1; // Calculate Edge Mean and Variance Values
use_sat_mean_var = 1;  // Calculate Saturation Mean and Variance Values
num_color_selects = 8; // Number of Color Selects (0, 4 or 8)
use_y_histogram = 1;   // Calculate Y Histogram
use_u_histogram = 1;   // Calculate U Histogram
use_v_histogram = 1;   // Calculate V Histogram
use_hue_histogram = 1; // Calculate Hue Histogram
```

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

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

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

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

```
int xilinx_ip_v_ic_v1_1_bitacc_simulate(
      struct xilinx_ip_v_ic_v1_1_generics* generics,
      struct xilinx_ip_v_ic_v1_1_inputs*   inputs,
      struct xilinx_ip_v_ic_v1_1_outputs*  outputs).
```

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

```
void xilinx_ip_v_ic_v1_1_destroy(
       struct xilinx_ip_v_ic_v1_1_inputs *input,
       struct xilinx_ip_v_ic_v1_1_outputs *output)
```

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

Image Characterization Video Input Structure

Input images or video streams can be provided to the Image Characterization v1.1 reference model using the general purpose `video_struct` structure, defined in `video_utils.h`:

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

Table 3-1: **Member Variables of the Video Structure**

Member Variable	Designation
Frames	Number of video/image frames in the data structure
Rows	Number of rows per frame*
Cols	Number of columns per frame*
Bit_per_component	Number of bits per color channel/component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.
Mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table 3-2.
Data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

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

Table 3-2: **Named Constants for Video Modes With Corresponding Planes and Representations**

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

Table 3-2: **Named Constants for Video Modes With Corresponding Planes and Representations**

FORMAT_C444_M	5	444 YUV video with motion
FORMAT_RGBM	5	RGB video with motion

Initializing the Image Characterization Input Video Structure

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

The Image Characterization reference model expects to input a video structure with a mode of FORMAT_C420_M or FORMAT_C422_M with the video data mapped into the video structure as shown in Table 3-3.

Table 3-3: **Video Structure Mapping**

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

Binary Image/Video Files

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

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

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

Working With Video_struct Containers

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

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

Function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table 3-2. Functions `video_rows_per_plane` and `video_cols_per_plane` return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in

conjunction to process all pixels within a video stream stored in variable in_video, with the following construct:

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

YUV Binary Image/Video Files

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

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

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

Working With Yuv_video_struct Containers

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

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

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

Image Characterization Statistics Output Structure

The Image Characterization v1.1 reference model outputs a set of image characterization statistics for each frame that is processed. The output statistics are provided using the `image_char_stats_struct` structure, defined in the `image_char_stats_utils.h` file:

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

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

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

```
  uint16  color_sel[8]; // Color Select (x8)
};
```

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

Working With Image_char_stats_struct Containers

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

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

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

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

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

The output matches the format of the output of the Image Characterization core. See *DS727 - LogiCORE IP Image Characterization v1.1 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 Image Characterization C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information for use by the model.
- Writes the Image Characterization statistics to an output file.

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

```
Usage: run_bitacc_cmodel in_file config_file out_file

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

Config File Format

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

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

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

Initializing the Image Characterization Input Video Structure

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

YUV Image Files

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

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

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

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

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

C Model Example I/O Files

Input Files

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

Output Files

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

Compiling Image Characterization C Model With Example Wrapper

Linux (32-bit and 64-bit)

To compile the example code, perform these steps:

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

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

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

   `libstlport.so.5.1`

   `libIp_v_ic_v1_1_bitacc_cmodel.so`

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

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -lIp_v_ic_v1_1_bitacc_cmodel
-Wl,-rpath,.
```

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

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

Windows (32-bit and 64-bit)

Precompiled `library v_ic_v1_1_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. The following is an example using Microsoft Visual Studio. In Visual Studio create a new, empty Win32 Console Application project. To existing items, add:

- **libIp_v_ic_v1_1_bitacc_cmodel.lib** to the "Resource Files" folder of the project
- **run_bitacc_cmodel.c** to the "Source Files" folder of the project
- **v_ic_v1_1_bitacc_cmodel.h** to "Header Files" folder of the project
- **yuv_utils.h** to the "Header Files" folder of the project
- **rgb_utils.h** to the "Header Files" folder of the project
- **video_utils.h** to the "Header Files" folder of the project
- **image_char_stats_utils.h** to the "Header Files" folder of the project

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

Running the Delivered Executables

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

Linux (32-bit and 64-bit)

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

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

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

   `libstlport.so.5.1`

   `libIp_v_ic_v1_1_bitacc_cmodel.so`

   `run_bitacc_cmodel`

3. Execute the model:

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

Windows (32-bit and 64-bit)

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

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

XILINX®

Appendix A

Image Characterization Statistics Input

This appendix provides information on the image characterization statistics input. For additional information, see the *LogiCORE IP Image Characterization Data Sheet* (DS727).

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

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

```
### Frame #1 Header ###
ffffffff     # Frame Valid
00000001     # Frame Index
00000000     # Frame Header Padding (30 lines)
…
00000000
### Frame #1 Global Statistics ###
957d8395     # Low Frequency Mean = 0x95, V Mean = 0x7D,
             # U Mean = 0x83, Y Mean = 0x95
07001604     # Saturation Mean = 0x07, Motion Mean = 0x00,
             # Edge Mean = 0x16, High Frequency Mean = 0x04
0036007c     # U Variance = 0x0036, Y Variance = 0x007c
0077003a     # Low Frequency Variance = 0x0077, V Variance = 0x003a
02f1001d     # Edge Variance = 0x02f1, High Frequency Variance = 0x001d
00040000     # Saturation Variance = 0x0004, Motion Variance = 0x0000
00000000     # Global Statistics Padding (26 lines)
…
00000000
### Frame #1 Y Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 U Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 V Histogram ###
00000000     # 256 lines of Histogram data (Bin 1 – Bin 256)
```

```
…
00000000
### Frame #1 Hue Histogram ###
00000e71     # 256 lines of Histogram data (Bin 1 – Bin 256)
…
00000000
### Frame #1 Block #0 ([0][0]) ###
bc6f92bc     # Low Frequency Mean = 0x95, V Mean = 0x7D,
             # U Mean = 0x83, Y Mean = 0x
18000000     # Saturation Mean = 0x07, Motion Mean = 0x00,
             # Edge Mean = 0x16, High Frequency Mean = 0x
009100ac     # U Variance = 0x0036, Y Variance = 0x
00ac0046     # Low Frequency Variance = 0x0077, V Variance = 0x
00000000     # Edge Variance = 0x02f1, High Frequency Variance = 0x
000d0000     # Saturation Variance = 0x0004, Motion Variance = 0x
00000000     # Color Select 2 = 0x0000, Color Select 1 = 0x0000
00100010     # Color Select 4 = 0x0010, Color Select 3 = 0x0010
00000000     # Color Select 6 = 0x0000, Color Select 5 = 0x0000
00000000     # Color Select 8 = 0x0000, Color Select 7 = 0x0000
00000000     # Reserved
00000000     # Reserved
00000000     # Reserved
00000000     # Reserved
### Frame #1 Block #1 ([0][1]) ###
bd7091bd
17000000
004800a2
00a70046
00000001
00030000
00000000
00100010
00000000
00000000
00000000
00000000
00000000
00000000
…
# Frames #2 - #4094 follow the same format at Frame #0
…
### Frame #1 Block #4095 ([63][63]) ###
80817f80
```

```
02001307
0028002d
00400069
015a0020
00010000
0003000d
00100000
00000000
00000010
00000000
00000000
00000000
00000000
```