LogiCORE IP Image Statistics v2.0 Bit Accurate C Model

User Guide

UG837 April 30, 2011

XILINX®

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

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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/30/11	1.0	Initial Xilinx release.

Table of Contents

Revision History 2

Preface: About This Guide

Additional Resources 5
List of Acronyms 5

Chapter 1: Introduction

Features 7
Overview 7
Additional Core Resources 7
Technical Support 8
Feedback 8
Image Statistics v2.0 Bit Accurate C Model and IP Core 8
Document 8
Software Requirements 8

Chapter 2: User Instructions

Unpacking and Model Contents 9
Installation 10
Software Requirements 10

Chapter 3: Image Statistics v2.0 Bit Accurate C Model

The Image Statistics Input and Output Video Structure 13
Initializing the Image Statistics Input Video Structure 15
Bitmap Image Files 15
Binary Image/Video Files 16
Working with video_struct Containers 16
Destroy the Video Structure 16
C Model Example Code 17
Compiling with the Image Statistics v2.0 C Model 17
Linux (64-bit) 17
Windows (32-bit) 17

About This Guide

This user guide provides information about the Xilinx® LogiCORE™ IP Images Statistics v2.0 bit accurate C model 32-bit Windows and 64-bit Linux platforms.

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.

List of Acronyms

Acronym	Spelled Out
ANSI	American National Standards Institute
GCC	GNU Compiler Collection
GUI	Graphical User Interface
I/O	Input/Output
IP	Intellectual Property
RGB	Red Green Blue
STL	Standard Template Library

XILINX®

Chapter 1

Introduction

The Xilinx® LogiCORE™ IP Image Statistics v2.0 core has a bit accurate C model designed for system modeling.

Features

- Bit accurate with Image Statistics v2.0 core
- Statically linked library (.lib, .o, .obj)
- Available for 32-bit Windows, and 64-bit Linux platforms
- Supports all features of the Image Statistics core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C code showing how to use the function is provided

Overview

The Xilinx LogiCORE IP Image Statistics v2.0 has a bit accurate C model for 32-bit Windows 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 given in Chapter 3, Image Statistics v2.0 Bit Accurate C Model. An example piece of C code showing how to call the model is provided to demonstrate the use of the C model.

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

The latest version of the model is available for download on the Xilinx LogiCORE IP Image Statistics web page.

Additional Core Resources

For detailed information and updates about the Image Statistics v2.0 core, see the following documents, located on the Image Statistics product page.

Technical Support

For technical support, go to www.xilinx.com/support.For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the Image Statistics v2.0 core. Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP Image Statistics v2.0 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 Statistics v2.0 core and the accompanying documentation.

Image Statistics v2.0 Bit Accurate C Model and IP Core

For comments or suggestions about the Image Statistics v2.0 core and bit accurate C model, submit a WebCase from 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 Image Statistics v2.0 core and bit accurate C model, submit a WebCase from 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

Software Requirements

The Image Statistics v2.0 C models were compiled and tested with the following software:

Table 1-1: **Compilation Tools for the Bit Accurate C models**

Platform	C Compiler
64 bit Linux	GCC 4.1.1
32 bit Windows	Microsoft Visual Studio 2005

XILINX®

Chapter 2

User Instructions

Unpacking and Model Contents

Unzip the v_stats_v2_0_bitacc_model.zip file, containing the bit accurate models for the Image Statistics IP Core. This produces the directory structure and files shown in Table 2-1.

Table 2-1: **Directory Structure and Files of the Image Statistics v2.0 Bit Accurate Model**

File Name	Contents
README.txt	release notes
ug837_v_stats_ref_model.pdf	this file
v_stats_v2_0_bitacc_cmodel.h	model header file
rgb_utils.h	header file declaring the RGB image / video container type and support functions
bmp_utils.h	header file declaring the bitmap (.bmp) image file I/O functions.
video_utils.h	header file declaring the generalized image / video container type, I/O and support functions
run_bitacc_cmodel.c	example code calling the C model
kodim11.bmp	768x512 sample test image from the True-color Kodak test images
/lin64	Precompiled bit accurate ANSI C reference model for simulation on 64 bit Linux platforms.
libIp_v_stats_v2_0_bitacc_cmodel.so	Image Statistics v2.0 model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by the Image Statistics and RGB to YCrCb object libraries (Linux platforms only)
/win32	Precompiled bit accurate ANSI C reference model for simulation on 32 bit Windows platforms.
libIp_v_stats_v2_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation (Windows platforms only.)

Installation

On Linux, ensure that the directory in which the files libIp_v_stats_v2_0_bitacc_cmodel.so and libstlport.so.5.1 are located is in your $LD_LIBRARY_PATH environment variable.

Software Requirements

The Image Statistics v2.0 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
64-bit Linux	GCC 4.1.1
32-bit Windows	Microsoft Visual Studio 2005 (Visual C++ 8.0)

XILINX®

Chapter 3

Image Statistics v2.0 Bit Accurate C Model

The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file v_stats_v2_0_bitacc_cmodel.h.

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

```
struct  xilinx_ip_v_stats_v2_0_generics stats_generics;
struct  xilinx_ip_v_stats_v2_0_inputs   stats_inputs;
struct  xilinx_ip_v_stats_v2_0_outputs  stats_outputs;
```

Declaration of the preceding structs can be found in v_stats_v2_0_bitacc_cmodel.h.

The only generic parameter the Image Statistics v2.0 IP Core bit accurate model takes is the DATA_WIDTH, corresponding to the CORE Generator™ software "Data Width" parameter. Allowed values are 8,10 and 12.

Calling `xilinx_ip_v_stats_v2_0_get_default_generics(&stats_generics)` initializes the generics structure with the Image Statistics GUI default DATA_WIDTH value (8).

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

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

Type	Name	Function
video_struct	video_in	Holds the input video stream (may contain multiple frames)
int	hmax0	Column index of the first vertical zone delineator (1)
int	hmax1	Column index of the second vertical zone delineator (1)
int	hmax2	Column index of the third vertical zone delineator (1)
int	vmax0	Row index of the first horizontal zone delineator (1)
int	vmax1	Row index of the second horizontal zone delineator (1)
int	vmax2	Row index of the third horizontal zone delineator (1)

Table 3-1: **Member Variables of the Input Structure *(Cont'd)***

Type	Name	Function
int	hist_zoom_factor	Controls CrCb histogram zoom around the gray (center point), which can be useful for white-balance algorithms. 0: No zoom, full Cb and Cr range represented (lowest resolution) 1: Zoom by 2 2: Zoom by 4 3: Zoom by 8 (highest resolution around gray point)
int	rgb_hist_zone_en	16 bit value, each bit controlling whether or not the corresponding zone is included in RGB histogram calculation.
int	ycc_hist_zone_en	16 bit value, each bit controlling whether or not the corresponding zone is included in YCC histogram calculation.
int	frame	The input video struct video_in may contain multiple frames. This value identifies which frame in the input video struct will be analyzed.

1. See Figure 3-1 for the definition of zone delineators.

`xilinx_ip_v_stats_v2_0_get_default_inputs(&stats_generics, &stats_inputs)` initializes members of the input structure with the Image Statistics GUI default values.

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

Note: Before calling `xilinx_ip_v_stats_v2_0_get_default_inputs()` it is advised to initialize the `video_in` structure by loading an image or set of video frames. The horizontal and vertical delineators are set by default such that the input images are split into zones with identical dimensions.

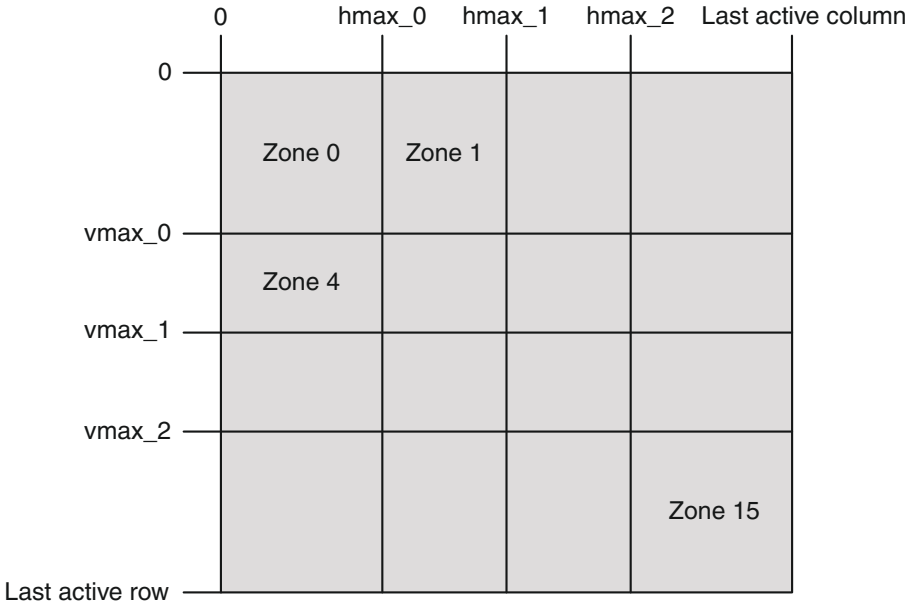

Figure 3-1: **Definition of Zone Delineators**

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

```
int xilinx_ip_v_stats_v2_0_bitacc_simulate(
     struct xilinx_ip_v_stats_v2_0_generics* generics,
     struct xilinx_ip_v_stats_v2_0_inputs*   inputs,
     struct xilinx_ip_v_stats_v2_0_outputs*  outputs).
```

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

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

```
void xilinx_ip_v_stats_v2_0_destroy(
     struct xilinx_ip_v_stats_v2_0_inputs *input,
     struct xilinx_ip_v_stats_v2_0_outputs *output).
```

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

The Image Statistics Input and Output Video Structure

Input images or video streams can be provided to the Image Statistics v2.0 reference model using the video_struct structure, defined in video_utils.h:

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

Table 3-2: **Member Variables of the Video Structure**

Member Variable	Designation
Frames	Number of video/image frames in the data structure
Rows	Number of rows per frame Pertaining to the image plane with the most rows and columns, such as the luminance channel for yuv data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y,u and v may have different dimensions.
Cols	Number of columns per frame Pertaining to the image plane with the most rows and columns, such as the luminance channel for yuv data. Frame dimensions are assumed constant through the all frames of the video stream, however different planes, such as y,u and v may have different dimensions.
bits_per_ component	Number of bits per color channel / component. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.

Table 3-2: **Member Variables of the Video Structure *(Cont'd)***

Member Variable	Designation
Mode	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table 3-3.
data	Set of 5 pointers to 3 dimensional arrays containing data for image planes. data is in 16 bit unsigned integer format accessed as data[plane][frame][row][col]

Table 3-3: **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
FORMAT_C444_M	5	444 YUV video with Motion
FORMAT_RGBM	5	RGB video with Motion

Initializing the Image Statistics Input Video Structure

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

Bitmap Image Files

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

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

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

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

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

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

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

Binary Image/Video Files

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

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

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

Working with video_struct Containers

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

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

Function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table 6. Functions `video_rows_per_plane` and `video_cols_per_plane` return the number of rows and columns in a given plane of the selected video structure. The following example demonstrates using these functions in conjunction to process all pixels within a video stream stored in variable `in_video`, with the following construct:

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

Destroy the Video Structure

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

C Model Example Code

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

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

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

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

Compiling with the Image Statistics v2.0 C Model

Linux (64-bit)

To compile the example code, first ensure that the directory in which the files libIp_v_stats_v2_0_bitacc_cmodel.so and libstlport.so.5.1 are located is present in your $LD_LIBRARY_PATH environment variable. These shared libraries are referenced during the compilation and linking process. Then cd into the directory where the header files, the library files and run_bitacc_cmodel.c were unpacked. The libraries and header files are referenced during the compilation and linking process.

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

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -
lIp_v_stats_v2_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit)

Precompiled library v_stats_v2_0_bitacc_cmodel.lib, and top level demonstration code run_bitacc_cmodel.c should be compiled with an ANSI C compliant compiler under Windows. Here an example is presented using Microsoft Visual Studio.

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

- libIpv_stats_v2_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_stats_v2_0_bitacc_cmodel.h header files to "Header Files" folder of the project (optional),

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