LogiCORE IP YCrCb to RGB Color-Space Converter v3.0 Bit Accurate C Model

User Guide

UG833 (v1.0) June 22, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/22/11	1.0	Initial Xilinx release.

Table of Contents

XILINX®

Chapter 1

Introduction

This document introduces the bit accurate C model for the Xilinx® LogiCORE™ IP YCrCb to RGB Color-Space Converter v3.0 core, which has been developed primarily for system modeling.

Features

- Bit accurate with the YCrCb to RGB Color-Space Converter v3.0 core
- Statically linked library (.lib, .o, .obj – Windows)
- Dynamically linked library (.so – Linux)
- Available for 32-bit Windows and 64-bit Linux platforms
- Supports all features of the YCrCb to RGB core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C is provided to show how to use the function

Overview

The Xilinx LogiCORE IP YCrCb to RGB Color-Space Converter v3.0 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, Using the C Model. An example piece of C code is provided to show how to call the 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 YCrCb to RGB Color-Space Converter web page at:

http://www.xilinx.com/products/intellectual-property/YCrCb_to_RGB.htm

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the YCrCb to RGB Color-Space Converter v3.0 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP YCrCb to RGB Color-Space Converter v3.0 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 YCrCb to RGB Color-Space Converter v3.0 core and the accompanying documentation.

YCrCb to RGB Color-Space Converter v3.0 Bit Accurate C Model and IP Core

For comments or suggestions about the YCrCb to RGB Color-Space Converter v3.0 core and bit accurate C model, submit a WebCase from: http://www.xilinx.com/support/clearexpress/websupport.htm

Be sure to include this information:

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

Document

For comments or suggestions about the documentation for the YCrCb to RGB Color-Space Converter v3.0 core and bit accurate C model, submit a WebCase from: http://www.xilinx.com/support/clearexpress/websupport.htm

Be sure to include this 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_ycrcb2rgb_v3_0_bitacc_model.zip` file, containing the bit accurate models for the YCrCb to RGB Color-Space Converter IP core. This creates the directory structure and files in Table 2-1.

Table 2-1: **Directory Structure and Files of the YCrCb to RGB Color-Space Converter v3.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
ug833_v_ycrcb2rgb.pdf	LogiCORE IP YCrCb to RGB Color-Space Converter Bit Accurate C Model User Guide
v_ycrcb2rgb_v3_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image/video container type and support functions
yuv_utils.h	Header file declaring the YUV (.yuv) image file I/O 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
kodim19_128x192.bmp	128x192 sample test image of the lighthouse 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_ycrcb2rgb_v3_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so

Table 2-1: **Directory Structure and Files of the YCrCb to RGB Color-Space Converter v3.0 Bit Accurate C Model**

/win32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation

Installation

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

- `libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Software Requirements

The YCrCb to RGB Color-Space Converter v3.0 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

XILINX®

Chapter 3

Using the C Model

The bit accurate C model is accessed through a set of functions and data structures that are declared in the `v_ycrcb2rgb_v3_0_bitacc_cmodel.h` file. Before using the model, the structures holding the inputs, generics and output of the YCrCb to RGB Color-Space Converter instance must be defined:

```
struct  xilinx_ip_v_ycrcb2rgb_v3_0_generics generics;
struct  xilinx_ip_v_ycrcb2rgb_v3_0_inputs inputs;
struct  xilinx_ip_v_ycrcb2rgb_v3_0_outputs outputs;
```

The declaration of these structures is in the `v_ycrcb2rgb_v3_0_bitacc_cmodel.h` file. Table 3-1 lists the generic parameters taken by the YCrCb to RGB Color-Space Converter v3.0 IP core bit accurate model, as well as the default values.

Table 3-1: **Core Generic Parameters and Default Values**

Generic Variable	Type	Default Value	Range	Description
IWIDTH	int	8	8,10,12	Input data width
CWIDTH	int	18	8-18	Coefficient bits
OWIDTH	int	8	8,10,12	Output width
ACOEF	double	0.299	0.0 - 1.0	A Coefficient[1] 0.0 < ACOEFF + BCOEFF < 1.0
BCOEF	double	0.114	0.0 - 1.0	B Coefficient[1] 0.0 < ACOEFF + BCOEFF < 1.0
CCOEF	double	0.713	0.0 - 0.9	C Coefficient[1]
DCOEF	double	0.564	0.0 - 0.9	D Coefficient[1]
YOFFSET	int	16	$0 – 2^{OWIDTH}-1$	Offset for the Luminance Channel
COFFSET	int	128	$0 – 2^{OWIDTH}-1$	Offset for the Chrominance Channels
YMIN	int	16	$0 – 2^{OWIDTH-1}-1$	Clamping value for the Luminance Channel
CMIN	int	16	$0 – 2^{OWIDTH-1}-1$	Clamping value for the Chrominance Channels
YMAX	int	240	$2^{OWIDTH-1} – 2^{OWIDTH}-1$	Clipping value for the Luminance Channel
CMAX	int	240	$2^{OWIDTH-1} – 2^{OWIDTH}-1$	Clipping value for the Chrominance Channels

[1] For a detailed description of coefficients and other generic parameters, see the *LogiCORE IP YCrCb to RGB Color-Space Converter Data Sheet* (DS659).

Calling `xilinx_ip_v_ycrcb2rgb_v3_0_get_default_generics(&generics)` initializes the generics structure with the default value.

The `inputs` structure defines the actual input image. For the description of the input video structure, see Input and Output Video Structures.

Calling `xilinx_ip_v_ycrcb2rgb_v3_0_get_default_inputs(&generics, &inputs)` initializes the input video structure before it can be assigned an image or video sequence using the memory allocation or file I/O functions provided in the BMP, RGB or video utility functions.

Note: The `video_in` variable is not initialized to point to a valid image / video container, as the container size depends on the actual test image to be simulated. The initialization of the `video_in` structure is described in Initializing the Input Video Structure.

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

```
int xilinx_ip_v_ycrcb2rgb_v3_0_bitacc_simulate(
struct xilinx_ip_v_ycrcb2rgb_v3_0_generics* generics,
struct xilinx_ip_v_ycrcb2rgb_v3_0_inputs*   inputs,
struct xilinx_ip_v_ycrcb2rgb_v3_0_outputs*  outputs).
```

Results are included in the outputs structure, which contains only one member, type `video_struct`. After the outputs are evaluated and saved, dynamically allocated memory for input and output video structures must be released by calling this function:

```
void xilinx_ip_v_ycrcb2rgb_v3_0_destroy(
struct xilinx_ip_v_ycrcb2rgb_v3_0_inputs *input,
struct xilinx_ip_v_ycrcb2rgb_v3_0_outputs *output).
```

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

Input and Output Video Structures

Input images or video streams can be provided to the YCrCb to RGB Color-Space Converter v3.0 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 all frames of the video stream. However different planes, such as y, u and v can 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 all frames of the video stream. However different planes, such as y, u and v can have different dimensions.

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

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.
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 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].

Table 3-3: **Named Video Modes with Corresponding Planes and Representations[1]**

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

[1] See the *LogiCORE IP YCrCb to RGB Color-Space Converter v3.0 Data Sheet (DS659)* for modes supported by the core.

Initializing the 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`, `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 that 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 three planes (R,G,B). Consequently, the following functions 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.

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

Exchanging data between `rgb8_video_struct` and general `video_struct` type frames/videos is facilitated by these 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 issue 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.

Binary Image/Video Files

The `video_utils.h` header file declares functions that help load and save generalized video files in raw, uncompressed format.

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

These functions 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 can differ within each frame as defined by the actual video mode selected.

YUV Image Files

The `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 these 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 );
```

Working with Video_struct Containers

The video_utils.h header file 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);
```

The video_planes_per_mode function returns the number of component planes defined by the mode variable, as described in Table 3-3. The video_rows_per_plane and video_cols_per_plane functions 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 the in_video variable:

```
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]
      }
    }
  }
}
```

XILINX®

Chapter 4

C Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided to demonstrate the steps required to run the model. After following the compilation instructions, 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, this help message is issued:

```
Usage:...
in_file : path/name of the input file (BIN file)
out_file : path/name of the output file (24-bit RGB BMP file)
```

During successful execution, two files are created. One file has a .bin extension and contains the output image in binary format, retaining OWIDTH bits. The other file has a .bmp extension and contains the output RGB image in bitmap format. The structure of .bin files are described in Binary Image/Video Files.

To ease modifying and debugging the provided top-level demonstrator using the built-in debugging environment of Visual Studio, the top-level command line parameters can be specified through the Project Property Pages using these steps:

1. In the Solution Explorer pane, right-click the project name and select Properties in the context menu.
2. Select Debugging on the left pane of the Property Pages dialog box.
3. Enter the paths and file names of the input and output images in the Command Arguments field.

Compiling YCrCb to RGB Color-Space Converter C Model with Example Wrapper

Linux (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 using a command such as:

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

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

   `libstlport.so.5.1`

   `libIp_v_ycrcb2rgb_v3_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_ycrcb2rgb_v3_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit)

The precompiled library `v_ycrcb2rgb_v3_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Win32 Console Application project.
2. As existing items, add:
   a. `libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.lib` to the Resource Files folder of the project
   b. `run_bitacc_cmodel.c` to the Source Files folder of the project
   c. `v_ycrcb2rgb_v3_0_bitacc_cmodel.h` to the Header Files folder of the project
3. After the project is created and populated, it must be compiled and linked (built) to create a win32 executable. To perform the build step, select "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.

XILINX®

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Core Resources

For detailed information and updates about the Xilinx LogiCORE IP YCrCb to RGB Color-Space Converter v3.0 core, see:

http://www.xilinx.com/products/intellectual-property/YCrCb_to_RGB.htm

- *YCrCb to RGB Color-Space Converter v3.0 Data Sheet (DS659)*
- *YCrCb to RGB Color-Space Converter v3.0 Release Notes*