Notice of Disclaimer
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.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/22/11</td>
<td>1.0</td>
<td>Initial Xilinx release.</td>
</tr>
</tbody>
</table>
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:

Chapter 1: Introduction

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
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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>README.txt</td>
<td>Release notes</td>
</tr>
<tr>
<td>ug833_v_ycrcb2rgb.pdf</td>
<td>LogiCORE IP YCrCb to RGB Color-Space Converter Bit Accurate C Model User Guide</td>
</tr>
<tr>
<td>v_ycrcb2rgb_v3_0_bitacc_cmodel.h</td>
<td>Model header file</td>
</tr>
<tr>
<td>rgb_utils.h</td>
<td>Header file declaring the RGB image/video container type and support functions</td>
</tr>
<tr>
<td>yuv_utils.h</td>
<td>Header file declaring the YUV (.yuv) image file I/O functions</td>
</tr>
<tr>
<td>bmp_utils.h</td>
<td>Header file declaring the bitmap (.bmp) image file I/O functions</td>
</tr>
<tr>
<td>video_utils.h</td>
<td>Header file declaring the generalized image/video container type, I/O and support functions</td>
</tr>
<tr>
<td>run_bitacc_cmodel.c</td>
<td>Example code calling the C model</td>
</tr>
<tr>
<td>kodim19_128x192.bmp</td>
<td>128x192 sample test image of the lighthouse image from the True Color Kodak test images</td>
</tr>
<tr>
<td>/lin64</td>
<td>Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms</td>
</tr>
<tr>
<td>libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so</td>
<td>Model shared object library</td>
</tr>
<tr>
<td>libstlport.so.5.1</td>
<td>STL library, referenced by libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so</td>
</tr>
</tbody>
</table>
Chapter 2: User Instructions

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

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libIp_v_yrcrb2rgb_v3_0_bitacc_cmodel.lib</td>
<td>Precompiled library file for win32 compilation</td>
</tr>
</tbody>
</table>

Installation

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

- libIp_v_yrcrb2rgb_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

<table>
<thead>
<tr>
<th>Platform</th>
<th>C Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit Linux</td>
<td>GCC 4.1.1</td>
</tr>
<tr>
<td>32-bit Windows</td>
<td>Microsoft Visual Studio 2005</td>
</tr>
</tbody>
</table>
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:

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

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:

```c
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:

```c
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`:

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

<table>
<thead>
<tr>
<th>Member Variable</th>
<th>Designation</th>
</tr>
</thead>
<tbody>
<tr>
<td>frames</td>
<td>Number of video/image frames in the data structure.</td>
</tr>
<tr>
<td>rows</td>
<td>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.</td>
</tr>
<tr>
<td>cols</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
Exchanging data between \texttt{rgb8\_video\_struct} and general \texttt{video\_struct} type frames/videos is facilitated by these functions:

\begin{verbatim}
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 );
\end{verbatim}

\textbf{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.

\section*{Binary Image/Video Files}

The \texttt{video\_utils.h} header file declares functions that help load and save generalized video files in raw, uncompressed format.

\begin{verbatim}
int read\_video(FILE* infile, struct video\_struct* in\_video);  
int write\_video(FILE* outfile, struct video\_struct* out\_video);  
\end{verbatim}

These functions serialize the \texttt{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.

\section*{YUV Image Files}

The \texttt{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 \texttt{yuv8\_video\_struct}, which is defined in \texttt{yuv\_utils.h}:

\begin{verbatim}
int write\_yuv8(FILE* outfile, struct yuv8\_video\_struct* yuv8\_video);  
int read\_yuv8(FILE* infile, struct yuv8\_video\_struct* yuv8\_video);  
\end{verbatim}

Exchanging data between \texttt{yuv8\_video\_struct} and general \texttt{video\_struct} type frames/videos is facilitated by these functions:

\begin{verbatim}
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 );
\end{verbatim}
Working with Video_struct Containers

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

```c
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:

```c
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 < video_rows_per_plane(in_video,plane); row++) {
            for (int col = 0; col < video_cols_per_plane(in_video,plane); col++) {
                // User defined pixel operations on
                // in_video->data[plane][frame][row][col]
            }
        }
    }
}
```
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.
Appendix A

Additional Resources

Xilinx Resources

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


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


Core Resources

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


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