

LogiCORE Color Correction Matrix v2.0 Bit Accurate C Model

User Guide

UG830 (v1.0) June 20, 2011



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.

| Date | Version | Revision |
|----------|---------|-------------------------|
| 06/20/11 | 1.0 | Initial Xilinx release. |

Table of Contents

| | |
|--|----|
| Revision History | 2 |
| Chapter 1: Introduction | |
| Features | 5 |
| Overview | 5 |
| Technical Support | 6 |
| Feedback | 6 |
| Chapter 2: User Instructions | |
| Unpacking and Model Contents | 7 |
| Installation | 8 |
| Software Requirements | 8 |
| Chapter 3: Using the C Model | |
| CCM Input and Output Video Structure | 11 |
| Initializing the CCM Input Video Structure | 12 |
| Chapter 4: C Model Example Code | |
| Compiling CCM C Model with Example Wrapper | 15 |
| Appendix A: Additional Resources | |
| Xilinx Resources | 17 |
| Core Resources | 17 |

Introduction

This document introduces the bit accurate C model for the Xilinx[®] LogiCORE[™] IP Color Correction Matrix (CCM) v2.0 core, which has been developed primarily for system modeling.

Features

- Bit accurate with the CCM v2.0 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 CCM core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C and M code is provided to show how to use the function
 - Example application C code wrapper file supports 8-bit BMP only

Overview

The Xilinx LogiCORE IP CCM 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, 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 CCM web page at:

<http://www.xilinx.com/products/intellectual-property/EF-DI-CCM.htm>

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the CCM v2.0 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP Color Correction Matrix 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 CCM v2.0 core and the accompanying documentation.

CCM v2.0 Bit Accurate C Model and IP Core

For comments or suggestions about the CCM v2.0 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 CCM v2.0 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

User Instructions

Unpacking and Model Contents

Unzip the `v_ccm_v2_0_bitacc_model.zip` file, containing the bit accurate models for the Color Correction Matrix IP Core. This creates the directory structure and files in [Table 2-1](#).

Table 2-1: Directory Structure and Files of the CCM v2.0 Bit Accurate C Model

| File Name | Contents |
|------------------------------------|---|
| README.txt | Release notes |
| ug830_v_ccm.pdf | LogiCORE IP Color Correction Matrix Bit Accurate C Model User Guide |
| v_ccm_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 |
| 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_ccm_v2_0_bitacc_cmodel.so | Model shared object library |
| libstlport.so.5.1 | STL library, referenced by <code>libIp_v_ccm_v2_0_bitacc_cmodel.so</code> |
| /win32 | Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms. |
| libIp_v_ccm_v2_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_ccm_v2_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Software Requirements

The CCM 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 |

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_ccm_v2_0_bitacc_cmodel.h` file.

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

```

struct xilinx_ip_v_ccm_v2_0_generics ccm_generics;
struct xilinx_ip_v_ccm_v2_0_inputs ccm_inputs;
struct xilinx_ip_v_ccm_v2_0_outputs ccm_outputs;

```

The declaration of these structures is in the `v_ccm_v2_0_bitacc_cmodel.h` file.

[Table 3-1](#) lists the generic parameters taken by the CCM v2.0 IP core bit accurate model, as well as the default values. For an actual instance of the core, these parameters can only be set in generation time through the CORE Generator™ GUI.

Table 3-1: Model Generic Parameters and Default Values

| Generic variable | Type | Default Value | Range | Description |
|------------------|------|---------------|---------|------------------|
| IWIDTH | int | 8 | 8,10,12 | Input data width |
| OWIDTH | int | 8 | 8,10,12 | Output width |

Calling `xilinx_ip_v_ccm_v2_0_get_default_generics(&ccm_generics)` initializes the generics structure with the CCM GUI defaults, listed in [Table 3-1](#).

Coefficients, offsets, clipping and clamping values can also be set dynamically through the pCore and General Purpose Processor interfaces. Consequently, these values are passed as inputs to the core, along with the actual test image, or video sequence (Table 3-2).

Table 3-2: Core Generic Parameters and Default Values

| Input Variable | Type | Default Value | Range | Description |
|----------------|--------------|-----------------------|---------------------------------------|---|
| video_in | video_struct | null | N/A | Container to hold input image or video data. ² |
| coeffs | double[3][3] | identity ¹ | [-4 to 4] | 3x3 matrix of floating point numbers |
| offsets | double[3] | zeros ¹ | $-2^{OWIDTH} + 1$ to $2^{OWIDTH} - 1$ | Offsets applied to the output color channels |
| CLAMP | int | 0 | 0 to $2^{OWIDTH-1} - 1$ | Clamping value for outputs |
| CLIP | int | $2^{OWIDTH-1} - 1$ | 0 to $2^{OWIDTH-1} - 1$ | Clipping value for outputs |

¹ For a detailed description of inputs and other generic parameters, see the *LogiCORE IP Color Correction Matrix Data Sheet (DS720)*.

² For the description of the input structure, see [Initializing the CCM Input Video Structure](#).

The structure `ccm_inputs` defines the values of run time parameters and the actual input image. Calling `xilinx_ip_v_ccm_v2_0_get_default_inputs(&ccm_generics, &ccm_inputs)` initializes the input structure with the CCM GUI default values (see Table 3-2).

Note: The `video_in` variable is not initialized because the initialization depends on the actual test image to be simulated. [Chapter 4, C Model Example Code](#) describes the initialization of the `video_in` structure.

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

```
int xilinx_ip_v_ccm_v2_0_bitacc_simulate(
    struct xilinx_ip_v_ccm_v2_0_generics* generics,
    struct xilinx_ip_v_ccm_v2_0_inputs* inputs,
    struct xilinx_ip_v_ccm_v2_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_ccm_v2_0_destroy(
    struct xilinx_ip_v_ccm_v2_0_inputs *input,
    struct xilinx_ip_v_ccm_v2_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.

CCM Input and Output Video Structure

Input images or video streams can be provided to the CCM 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 3-3: 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. |
| 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 3-4 . |
| data | Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as <code>data[plane][frame][row][col]</code> . |

Table 3-4: Named 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 CCM 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 `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.

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 3-4](#). 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]
            }
        }
    }
}
```


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

During successful execution, two files with a `.bin` extension are created. The first file corresponds to the input BMP image, with the same path and name as the input file, and a `.bin` extension. The other file similarly corresponds to the output file. These files contain the inputs and outputs of the CCM algorithm in full precision, as the BMP format does not support color resolutions beyond 8-bits per component. 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 CCM 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_ccm_v2_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_ccm_v2_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit)

The precompiled library `v_ccm_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. 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_ccm_v2_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_ccm_v2_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.

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 CCM v2.0 core, see:

<http://www.xilinx.com/products/intellectual-property/EF-DI-CCM.htm>

- *CCM v2.0 Data Sheet (DS720)*
- *CCM v2.0 Release Notes*

