LogiCORE IP Color Filter Array Interpolation v3.0

Bit Accurate C-Model

User Guide

UG802 (v1.0) March 1, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/01/11	1.0	Initial Xilinx release.

Table of Contents

About This Guide

The Xilinx® LogiCORE™ IP Color Filter Array Interpolation (CFA) v3.0 core has a bit accurate C model designed for system modeling. A MATLAB® software pcode function for seamless MATLAB software integration is also available

Features

This document contains the following chapters:

- Chapter 1, Introduction
- Chapter 2, Installation and Directory Structure
- Chapter 3, Using the C-Model
- Chapter 4, Bit Accurate MATLAB Model

Resources

To find additional documentation, see the Xilinx web site at:

http://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 Web Case, see the Xilinx web site at:

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

Additional Core Resources

For more information and updates about the CFA v3.0 core, see the CFA v3.0 product page.

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the CFA v3.0 core. Xilinx provides technical support for use of this product as described in this 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 CFA v3.0 core and the accompanying documentation.

CFA v3.0 Bit Accurate C Model and IP Core

For comments or suggestions about the CFA v3.0 core and bit accurate C model, please submit a Web Case at 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 this guide, please submit a Web Case at 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 1

Introduction

The Xilinx LogiCORE™ Color Filter Array Interpolation (CFA) v3.0 IP core provides a Bit Accurate C Model designed for system modeling. A MATLAB® software pcode function for seamless MATLAB software integration is also available.

Features

The following lists the basic features of the CFA Bit Accurate C Model.

- Bit accurate with CFA v3.0 core
- Statically linked library (.lib, .o, .obj)
- Available for 32-bit Windows, and 64-bit Linux platforms
- MATLAB software Pcode function
- Supports all features of the CFA core that affect numerical results
- Designed for rapid integration into a larger system model
- Example C and M code showing how to use the function is provided

Overview

The Xilinx® LogiCORE IP CFA v3.0 bit accurate C model has an interface consisting of a set of C functions, which reside in a statically linked library (shared library). The details of the interface are given in "CFA v3.0 C Model Interface". An example piece of C code showing how to call the model is provided. The model is also available as a MATLAB software Pcode function for seamless MATLAB software integration.

The model is bit accurate, which 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 CFA web page: http://www.xilinx.com/products/ipcenter/EF-DI-CFA.htm.

XILINX®

Chapter 2

Installation and Directory Structure

This chapter contains information for installing the Color FIlter Array C-Model, and describes the file contents and directory structure.

Software Requirements

The CFA v3.0 C models were compiled and tested with the following software versions.

Table 2-1: **Supported Systems and Software Requirements**

Platform	C-Compiler
64- bit Linux	GCC 4.1.1
32-bit Windows	Microsoft Visual Studio 2005 (Visual C++ 8.0)

M files provided in the Matlab directory allow testing and demonstrating the pcode functionality on the platform or operating system where MATLAB is installed. Pcode functionality has been tested with MATLAB versions 2008a and 2009b.

Installation

The installation of the c-model requires updates to the PATH variable, as described below.

Linux

Ensure that the directory in which the `libIp_v_cfa_v3_0_bitacc_cmodel.so` and `libstlport.so.5.1` files are located is in your $LD_LIBRARY_PATH environment variable.

M files provided in the Matlab directory allow testing and demonstrating the pcode functionality on the platform or operating system where MATLAB is installed. Pcode functionality has been tested with MATLAB versions 2008a and 2009b.

C-Model File Contents

Unzipping the v_cfa_v3_0_bitacc_model.zip file creates the following directory structures and files which are described inTable 2-2.

Table 2-2: **C-Model Files**

File	Description
README.txt	The release notes for the Color Filter Array C-Model.
ug802_cfa_cmodel.pdf	This file
v_cfa_v3_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_cfa_v3_0_bitacc_cmodel.so	model shared object library (Linux platforms only)
libstlport.so.5.1	STL library, referenced by libIp_v_cfa_v3_0_bitacc_cmodel.so (Linux platforms only)
/win32	Precompiled bit accurate ANSI C reference model for simulation (32 bit Windows platforms only)
libIp_v_cfa_v3_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation (Windows platforms only)
/matlab	Platform independent, bit accurate reference model for MATLAB® simulation
cfa_reference_model.p	Bit Accurate MATLAB reference model for the Color Filter Array Interpolation IP core.
create_Bayer.m	MATLAB.m function to prepare stimuli for the CFA reference model. The function sub-samples the input image according to the Bayer pattern.

XILINX®

Chapter 3

Using the C-Model

The bit-accurate C model is accessed through a set of functions and data structures, declared in the header file v_cfa_v3_0_bitacc_cmodel.h. Before using the model, the structures holding the inputs, generics and output of the CFA instance have to be defined, as illustrated below.

```
struct  xilinx_ip_v_cfa_v3_0_generics cfa_generics;
struct  xilinx_ip_v_cfa_v3_0_inputs   cfa_inputs;
struct  xilinx_ip_v_cfa_v3_0_outputs  cfa_outputs;
```

Declaration of the above structs are located in the `v_cfa_v3_0_bitacc_cmodel.h` file.

The only generic parameter the CFA v3.0 IP Core bit accurate C model takes is `DATA_WIDTH`, corresponding to the Core Generator *Data Width* parameter. Allowed values are 8, 10 and 12. Calling

```
xilinx_ip_v_cfa_v3_0_get_default_generics (&cfa_generics)
```

initializes the generics structure with the CFA GUI default DATA_WIDTH value (8).

The structure `cfa_inputs` defines the values of run-time parameters BAYER_PHASE and the actual input image. For the description of `BAYER_PHASE`, please see Figure 4-2, page 18. For the description of the input structure, see CFA Input and Output Video Structure.

Calling `xilinx_ip_v_cfa_v3_0_get_default_inputs(&cfa_generics, &cfa_inputs)` initializes the `BAYER_PHASE` member of the input structure with the CFA GUI default value (3).

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.

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

```
int xilinx_ip_v_cfa_v3_0_bitacc_simulate(
  struct xilinx_ip_v_cfa_v3_0_generics* generics,
  struct xilinx_ip_v_cfa_v3_0_outputs*  outputs).
```

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

After the outputs were evaluated and saved, dynamically allocated in memory for input and output video structures have to be released by calling function:

```
void xilinx_ip_v_cfa_v3_0_destroy(
  struct xilinx_ip_v_cfa_v3_0_inputs *input,
  struct xilinx_ip_v_cfa_v3_0_outputs *output).
```

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

CFA Input and Output Video Structure

Input images or video streams can be provided to the Color Filter Array 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 3-1: **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[a]
cols	Number of columns per frame[a]
bits_per_component	Number of bits per color channel/component[b]
mode	Contains information about the designation of data planes[c]
data	Set of five pointers to three-dimensional arrays containing data for image planes.[d]

a. Pertaining to the image plane with most rows and columns, such as t6he luminance channel for y,u,v data. Frame dimensions are assumed constant through all frames of the video stream; however, different planes (such as y, u, and v) may have different dimensions.

b. All image planes are assumed to have the same color/component representation. Maximum number of bits per component is 16.

c. Named constants to be assigned to mode are listed in Table 3-2.

d. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col]

Table 3-2: **Named Video Modes Constants with Planes and Representations**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome- Luminance only
FORMAT_RGB	3	RGB image/video data
FORMAT_C444	3	444YUV, 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 CFA Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or sequence of images. 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 three 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: Note: All image / video manipulation utility functions expect both input and output structures initialized, (for example, pointing to a structure which has been allocated in memory), either as static or dynamic variables. Moreover, the input structure has to 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, un-compressed 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 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 example below demonstrates all pixels within a video stream stored in variable `in_video`:

```
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]
      }
    }
  }
}
```

Destroying 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, 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 BMP file
```

During successful execution, two other files with the extension 'bin', are created. The first file corresponds to the input bmp image, and has the same path and name as the input file, with extension '.bin'. The other file similarly corresponds to the output file. These files contain the inputs and outputs of the CFA algorithm in full precision, as the BMP format does not support color resolutions beyond 8 bits per component. The structure of.bin files are detailed in the section Binary Image/Video Files.

Compiling with the CFA v3.0 C Model

Linux (64-bit)

To compile the example code, first ensure that the directory in which the files `libIp_v_cfa_v3_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, 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_cfa_v3_0_bitacc_cmodel -Wl,-rpath,.
```

Windows (32-bit)

Precompiled library v_cfa_v3_0_bitacc_cmodel.dll, 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:

- llibIpv_cfa_v3_0_bitacc_cmodel.dll to the "Resource Files" folder of the project
- run_bitacc_cmodel.c to the "Source Files" folder of the project
- v_cfa_v3_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) 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 Manage**r under the Build menu.

Chapter 4

Bit Accurate MATLAB Model

This chapter contains information about running the Bit-Accurate MATLAB model, and assumes that you have MATLAB installed and running.

Top-level MATLAB Demonstrator

While in MATLAB, cd into the directory where the contents of the zip file are located. The m-script file `ref_model_test.m` can be invoked directly from the MATLAB command window or through the editor by opening the file and selecting **Run** from the Debug menu.

The script opens the Lighthouse test-image, and invokes the CFA reference model to produce the bit accurate simulation results corresponding to the input image:

Figure 4-1: **Stimuli and Result of MATLAB Bit Accurate Reference Model**

Parameterization

The bit accurate MATLAB reference model function is declared as:

output_image: cfa_reference_model(input_image, DATA_WIDTH, BAYER_PHASE),

where:

input_image: The input image is expected either as monochrome image (MxN, two-dimensional array of integers), or as an RGB image (MxNx3, three-dimensional array of integers). In case input_image is provided as an RGB image, the reference_model automatically sub-samples it according to the Bayer-pattern and the provided BAYER_PHASE.

DATA_WIDTH: Corresponds to Core Generator "Data Width".
Allowed values are 8,10 and 12.

BAYER_PHASE: Corresponds to the value of Core Generator Parameter "BAYER_PHASE". Also, the value of Bayer Phase may be set through the pCore interface when the IP Core is generated as a pCore instance or directly through the core pinout when the IP Core is generated with the General Purpose Processor Interface. Allowed values for BAYER_PHASE are = {0,1,2,3}, which encode the Color Filter Array position relative to the top-left corner of the active area of the image sensor, as illustrated on Figure 4-2.

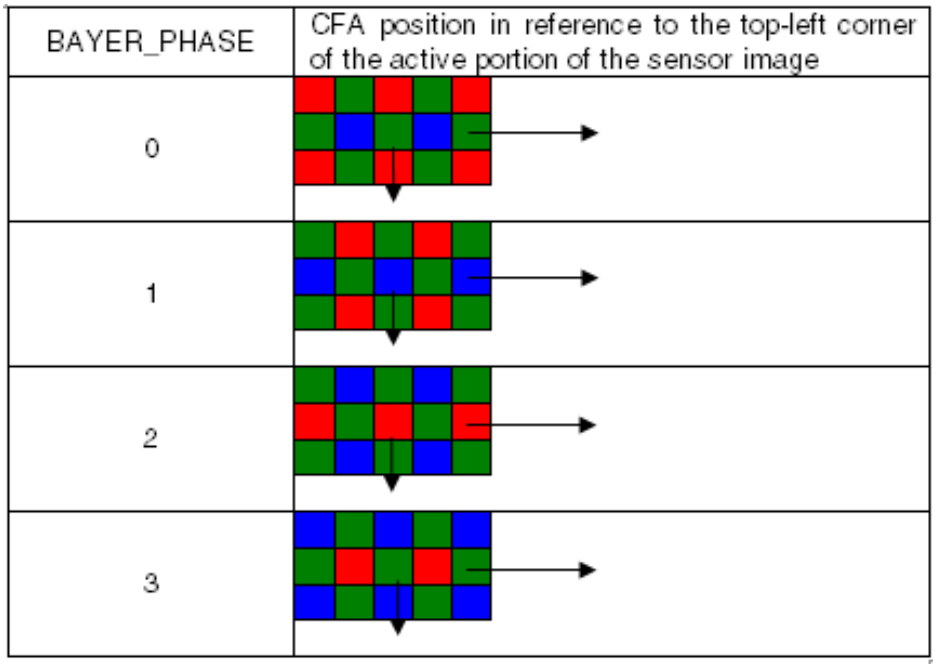

Figure 4-2: **BAYER_PHASE Codes for Color Filter Array Spatial Phase Relations**