LogiCORE™ IP Video Scaler v4.0 Bit Accurate C Model

User Guide

UG809 March 20, 2011

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/11	1.0	Initial Xilinx release.

Table of Contents

Chapter 5: Generating Coefficients

About This Guide

This user guide provides information about the Xilinx® LogiCORE IP Video Scaler v4.0 bit accurate C model 32-bit Windows and 64-bit Linux platforms.

Guide Contents

This manual contains the following chapters:

- Chapter 1, Introduction introduces the bit accurate C model for the Xilinx® LogiCORE™ IP Video Scaler v4.0 core, which has been developed primarily for system level modeling.
- Chapter 2, User Instructions provides information on the C model directory structure, files, installation, and software requirements.
- Chapter 3, Interface provides information on the C model interface, including defining the inputs, generics and output of the Video Scaler.
- Chapter 4, C Model Example Code provides an example C file along with the Win32 version of the executable for this example.
- Chapter 5, Generating Coefficients provides information on a coefficient generation routine that generates coefficients in the correct format.

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.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **`bus[7:0]`**, they are required.	**`ngdbuild`** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**`lowpwr =`**{**`on`**|**`off`**}
Vertical bar |	Separates items in a list of choices	**`lowpwr =`**{**`on`**|**`off`**}
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Horizontal ellipsis ...	Repetitive material that has been omitted	**`allow block`** *block_name loc1 loc2 ... locn;*

Convention	Meaning or Use	Example
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	**usr_teof_n** is active low.

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to www.xilinx.com for the latest speed files.

XILINX®

Chapter 1

Introduction

This document introduces the bit accurate C model for the Xilinx® LogiCORE™ IP Video Scaler v4.0 core, which has been developed primarily for system level modeling.

Features

- Bit accurate with v_scaler_v4_0 core
- Library module for the Video Scaler core function
- Available for 32-bit Windows and 64-bit Linux platforms
- Supports all features of the HW core that affect numerical results
- Designed for rapid integration into a larger system model
- Example application C code is provided to show how to use the function
  - Example application C code wrapper files support 8-bit YUV only
- Coefficient generator included in example application code
- Executable files for the example application provide instant access to the coefficient generator function

Overview

The bit accurate C model for the Xilinx® LogiCORE IP Video Scaler v4.0 can be used on 32-bit Windows and 64-bit Linux platforms. The model comprises a set of C functions, which reside in a statically linked library (shared library). Full details of the interface to these functions are provided in Chapter 3, Interface.

The main features of the C model package are:

- **Bit Accurate C Model** - produces the same output data as the Video Scaler v4.0 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.
- **Application Source Code** - uses the model library function. This can be used as example code showing how to use the library function. However, it also serves these purposes:
  - **A configuration file parser** is built into the application code. This simplifies the passing of many core parameters into the core (instead of a lengthy command line). A report file (`report.txt`) is generated detailing information and error conditions in the config file.
  - **A coefficient generator** is built into the application code allowing the user to create the .coe files used by the Video Scaler core. The output .coe file can be specified in CORE Generator™ or in EDK.

- **Input .yuv file** is (optionally) scaled by the application; 8-bit YUV422 format accepted.
- **Output .yuv file** is (optionally) generated by the application; 8-bit YUV422 format generated.

- **Executable Files** (Win32 and Lin64) - for the application summarized previously.

The latest version of the model is available for download on the LogiCORE IP Video Scaler Web page at: http://www.xilinx.com/products/ipcenter/EF-DI-VID-SCALER.htm

Additional Core Resources

For detailed information and updates about the Video Scaler v4.0 core, see these documents located on the core product page at: http://www.xilinx.com/products/ipcenter/EF-DI-VID-SCALER.htm

- *Video Scaler v4.0 User Guide (UG805)*
- *Video Scaler v4.0 Data Sheet (DS840)*
- *Video Scaler v4.0 Release Notes*

Technical Support

For technical support, go to www.xilinx.com/support. Questions are routed to a team with expertise using the Video Scaler v4.0 core.

Xilinx provides technical support for use of this product as described in this user guide (*LogiCORE IP Video Scaler 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 Video Scaler v4.0 core and the accompanying documentation.

Video Scaler v4.0 Bit Accurate C Model and IP Core

For comments or suggestions about the Video Scaler v4.0 core and bit accurate C model, submit a WebCase from 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 the documentation for the Video Scaler v4.0 core and bit accurate C model, submit a WebCase from 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 2

User Instructions

Unpacking and Model Contents

Unzip the `v_scaler_v4_0_bitacc_model.zip` file, containing the bit accurate models for the Video Scaler IP Core. This creates the directory structure and files in Table 2-1.

Table 2-1: **Directory Structure and Files of the Video Scaler v4.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
ug809_v_scaler.pdf	LogiCORE IP Video Scaler Bit Accurate C Model User Guide
v_scaler_v4_0_bitacc_cmodel.h	Model header file
v_ycrcb2rgb_v3_0_bitacc_cmodel.h	Color space converter model header file. This is used by the wrapper example.
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.
video_utils.h	Header file declaring the generalized image/video container type, I/O and support functions
xscaler_coefs.h	Coefficients – same values as delivered in CORE Generator as part of scaler driver (used in EDK with scaler pCore).
run_bitacc_cmodel.c	Example code calling the C model
video_in.yuv	10 frame video sequence for test purposes. Video format: 1280x720; YC422; 8 bits.
video_in.hdr	YUV file header
scaler.cfg	User programmable file containing configuration values for the core.
/lin64	Lin64 library directory
libIp_v_scaler_v4_0_bitacc_cmodel.so	Precompiled v_scaler_v4_0 dynamically linked shared object file for lin64 compilation.
libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so	Precompiled yCrCb to RGB converter dynamically linked shared object file for lin64 compilation
libstlport.so.5.1	STL library, referenced by libIp_ v_scaler_v4_0_bitacc_cmodel.so (Linux platforms only)
/win32	Win32 library directory

Table 2-1: **Directory Structure and Files of the Video Scaler v4.0 Bit Accurate C Model**

libIp_v_scaler_v4_0_bitacc_cmodel.lib	Precompiled statically linked scaler library file for win32 compilation
libIp_v_ycrcb2rgb_bitacc_model.lib	Precompiled statically linked color space converter library file for win32 compilation. This is used by the wrapper example.

Installation

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

- libIp_v_scaler_v4_0_bitacc_cmodel.so
- libstlport.so.5.1

Software Requirements

The Video Scaler v4.0 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)

Chapter 3

Interface

The scaler core function is a statically linked library. A higher level software project can make function calls to this function:

```
int xilinx_ip_v_scaler_v4_0_bitacc_simulate(
      struct xilinx_ip_v_scaler_v4_0_generics* generics,
      struct xilinx_ip_v_scaler_v4_0_inputs*   inputs,
      struct xilinx_ip_v_scaler_v4_0_outputs*  outputs).
```

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

```
struct  xilinx_ip_v_scaler_v4_0_generics scaler_generics;
struct  xilinx_ip_v_scaler_v4_0_inputs   scaler_inputs;
struct  xilinx_ip_v_scaler_v4_0_outputs  scaler_outputs
```

The declaration of these structures are in the `v_scaler_v4_0_bitacc_cmodel.h` file.

Before making the function call, complete these steps:

1. Populate the *generics* structure:
   - **num_h_taps** - Between 2 and 12.
   - **num_v_taps** - Between 2 and 12.
   - **max_phases** - 2-16, 32 or 64.
   - **Separate_YC_Coefs** - 0 = Shared coefficients between Y and C operations; 1= separate coefficient sets.
   - **Separate_HV_Coefs** - 0 = Shared coefficients between Y and C operations; 1= separate coefficient sets.
   - **UserCoefsEnabled** - 0 = Use internally generated coefficients; 1 = Specify user coefficients.
2. Populate the *inputs* structure to define the values of run time parameters:

   Note: This function scales *one frame at a time*.

   - **video_in** - Video structure that comprises these elements:
     - **bits_per_component** - 8, 10 or 12.
     - **cols** - Horizontal size of rectangle to be scaled: 32 to 4096.
     - **rows** - Vertical size of rectangle to be scaled: 32 to 4096.
     - **frames** - Set to 1; this function scales *one frame at a time.*
     - **mode** - Defines the chroma format (RGB, YUV422, and so on); see Table 3-2.
     - **data** - This is the frame of video data to be scaled, arranged in raster form.

- **aperture_start_pixel**, **aperture_end_pixel**, **aperture_start_line**, **aperture_end_line**: Aperture definition; see *LogiCORE Video Scaler v4.0 Data Sheet* for more information.
- **num_h_phases**, **num_v_phases**: Number of horizontal and vertical phases - 2-16, 32 or 64; the maximum is defined by `generic max_phases.`
- **SingleFrameCoefs** - Defines coefficients to be used in the *current frame*. User must specify a single set of coefficients using a `coefs_struct` (defined in the file `v_scaler_v4_0_bitacc_cmodel.h`.)
- **filename** - Not used by model core; used in wrapper. **Default**: set string to "".
- **video_in_txt_fid** - Not used by model core; used in wrapper. **Default**: set file pointer type to NULL.

3. Populate the *outputs* structure.
   - **video_out** - Video structure that comprises these elements:
     - **bits_per_component** - 8, 10 or 12.
     - **cols** - Horizontal size of output rectangle: 32 to 4096.
     - **rows** - Vertical size of output rectangle: 32 to 4096.
     - **frames** - Set to 1; this function scales *one frame at a time.*
     - **mode** - Defines the chroma format (RGB, YUV422 etc); see Table 3-2. *Must be set the same as the input mode*.

     Note: This structure also includes a *data* element, which is written by the scaler model itself.
   - **GenerateStimulusFiles** - Not used. **Default**: Set integer to 0.
   - **filename** - Not used in model core; used in wrapper. **Default**: set string to "".
   - **coefs_txtfid** - Not used in model core; used in wrapper. **Default**: set file pointer type to NULL.
   - **video_out_txt_fid** - Not used in model core; used in wrapper. **Default**: set file pointer type to NULL.

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

Results are provided in the outputs structure, which contains the output video data in the form of type `video_struct`. After the outputs have been evaluated or saved, dynamically allocated memory for input and output video structures must be released. See Delete the Video Structure for more information. Successful execution of all provided functions return a value of 0. Otherwise, a non-zero error code indicates that problems were encountered during function calls.

Input and Output Video Structure

Input images or video streams can be provided to the Video Scaler v4.0 reference model using the general purpose `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*
Cols	Number of columns per frame*
Bit_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 3-2.
Data	Contains information about the designation of data planes. Named constants to be assigned to mode are listed in Table 3-2.
	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]. In the scaler C model case, only one frame is scaled at any one time. Hence the '[frame]' index is always set to 0.

*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: **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

Table 3-2: **Named Constants for Video Modes With Corresponding Planes and Representations**

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

Working With Video_struct Containers

The header file `video_utils.h` 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 function `video_planes_per_mode` returns the number of component planes defined by the mode variable, as described in Table 3-2. The 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`:

```
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]
      }
    }
  }
}
```

Delete the Video Structure

Large arrays such as the `video_in` element in the video structure must be deleted to free up memory.

The following example function is defined as part of the `video_utils` package.

```
void free_video_buff(struct video_struct* video )
{
  int plane, frame, row;

  if (video->data[0] != NULL) {
   for (plane = 0; plane <video_planes_per_mode(video->mode); plane++)
{
      for (frame = 0; frame < video->frames; frame++) {
        for (row = 0; row<video_rows_per_plane(video,plane); row++) {
          free(video->data[plane][frame][row]);
        }
        free(video->data[plane][frame]);
      }
      free(video->data[plane]);
    }
```

```
    }
  }
```

This function can be called as follows:

```
free_video_buff(&scaler_yuv422_8b_inputs.video_in);
```

XILINX®

Chapter 4

C Model Example Code

An example C file, `run_bitacc_cmodel.c`, is provided along with the Win32 version of the executable for this example (`run_bitacc_cmodel.exe`). This C file has these characteristics:

- Contains an example of how to write an application that makes a function call to the Video Scaler C model core function.
- Contains an example of how to populate the video structures at the input and output, including allocation of memory to these structures.
- Uses a YUV file reading function to extract video information for use by the model.
- Uses a YUV file writing function to provide an optional output YUV file, which allows the user to visualize the result of the scaling operation.
- Generates coefficients in .coe file format, which can be used directly in the Video Scaler CORE Generator GUI to preload the scaler with real coefficients.
- Includes a config file parser, which allows the user to pass parameters into the model for multiple test cases.

After following the compilation instructions in this chapter, you should run the example executable. The executable takes the path/name of the config file as a parameter. If invoked with insufficient parameters, this help message is generated:

```
Usage: run_bitacc_cmodel <cfg_file>
           cfg_file  : path/name of the input  config file
           Example   : run_bitacc_cmodel ./scaler.cfg
```

Config File Format

During successful execution, the specified config file is parsed by the run_bitacc_cmodel example. In this file, you must specify:

- Repository directory path.
  - **All model input and output files are read from here and written to here.**
- Optional input YUV file name, with resolution.
- Optional output YUV file name.
- Coefficient output file name.
- Generic settings.
  - Num_h_taps, num_v_taps, max_phases, max_coef_sets, Separate_YC_Coefs, Separate_HV_Coefs.
  - These settings are held constant for all test cases specified in this file.

    - See *DS840 - LogiCORE IP Video Scaler Data Sheet* and *UG805 - LogiCORE IP Video Scaler User Guide* for more information on these parameters.
- Test cases.
    - The user can specify up to 16 test cases.
    - Each test case occupies one line, starting with the word "test".
    - Each line specifies:
        - Number of frames.
        - Input aperture size.
        - Output size.
        - Number of phases (H and V).
    - Source data override option.
        - Use data read from the source file, or override with useful ramp data or random data.
- Unity coefficient control on/off.
    - All coefficients, in all phases, are set to 0 except the central tap. This turns the scaler into a sample skip/repeat system.
    - User coefficients override unity coefficients.
- User coefficient controls.
    - User can specify his own correctly formatted .coe file for one set of coefficients to replace the default coefficients that would have been used otherwise by that test.
    - User specified coefficients replace any coefficients that would have been written into the coefficients output file.
    - User coefficients override unity coefficients.
- For each test case, one set of coefficients are written into the Coefficient Output file specified previously.

The example config file in the zip file (and shown here) provides more information on the formatting of this file.

```
################################################
# scaler.cfg: Scaler model example config file
################################################

################################################
# TESTCASE REPOSITORY.
################################################
# Scaler model output information is written to the Repo directory (defined below).
# Please create this directory manually.
# If user coefficients are defined, place the .coe file in this location.
# eg:
# Windows:  RepoDir C:\test
# Linux:  RepoDir /home/test
RepoDir C:\Test

################################################
# INPUT YUV FILE.
################################################
# Note1: Currently, only 8-bit 4:2:2 YUV formats are accepted by this program.
# Setting ParseYUVFile to 0 disables the ability of the model to read
```

```
# a YUV file.
# The input YUV file is expected to be found in the RepoDir specified above.
# Syntax:
#     InputFile <file.yuv> <Input file HSize> <Input file VSize>

ParseYUVFile 1
InputFile video_in_128x128.yuv 128 128

##################################################
# OUTPUT YUV FILE.
##################################################
# Note2: Currently, only 8-bit 4:2:2 YUV formats are generated by this program.
# Set EnableYUVOutputGeneration to 1 if you want to generate a YUV file
# IMPORTANT: GENERATED YUV OUTPUT FILES MUST BE OF A CONSISTANT RESOLUTION.
# OTHERWISE XILINX RECOMMENDS SETTING EnableYUVOutputGeneration to 0
# The output YUV file will be written to the RepoDir specified above.
# Syntax:
#     OutputFile <file.yuv>

EnableYUVOutputGeneration 0
OutputFile video_out.yuv

##################################################
# OUTPUT .COE FILE.
##################################################
# Provide name of an output .coe file.
# The Model will concatenate one set of coefficients to this file for each
# test-case listed under "TEST CASES" below.
# The output .coe file will be written to the RepoDir specified above.
# Syntax:
#     COEFileOut <filename>.coe

COEFileOut ModelOutput.coe

##################################################
# GENERIC PARAMETERS
##################################################
# a. num_h_taps : 2 - 12
# b. num_v_taps : 2 - 12
# c. max_phases : 2 - 16, 32 or 64
# d. max_coef_sets : 1 - 16
# e. Separate_YC_Coefs : 0=No; 1=Yes
# f. Separate_HV_Coefs : 0=No; 1=Yes

#         a   b   c   d   e   f
generics  9   7   4   16  0   1

##################################################
# TEST CASES
##################################################
# Each test case occupies one line, which must begin with the word 'test'.
# a. frames                                - Number of frames to be read/processed
# b. aperture_start_pixel                  - Cropping left edge
# c. aperture_end_pixel                    - Cropping right edge
# d. aperture_start_line                   - Cropping top edge
# e. aperture_end_line                     - Cropping bottom edge
# f. output_h_size                         - Desired horizontal size of output rectangle
# g. output_v_size                         - Desired horizontal size of output rectangle
```

```
# h. num_h_phases                        - Number of phases desired in the current horizontal
filter coefficients
# i. num_v_phases                         - Number of phases desired in the current vertical
filter coefficients
# j. Data source: video, ramp or random    - 0=video from file; 1=ramp; 2=random
# k. Enable Unity coefficients          - 1=unity coefs (but overridden by user coefficients
if enabled below); 0=generated/default
# l. Enable user-defined coefs             - 1=enabled; 0=disabled
# m. User coefficient file.                - Provide the filename of user coefficient file

# Each line represents one test case.
# Please restrict the number of test-cases in this section to no more than 16.

# test  a     b     c     d    e      f      g    h    i     j    k    l    m
test    1     0     127   0   127    128    128   4    4     2    0    0    user_coefs.coe
test    1     0     127   0   127    120    120   4    4     1    0    0    user_coefs.coe
test    1     0     127   0   127    112    112   4    4     0    0    0    user_coefs.coe
test    1     0     127   0   127    256    256   4    4     0    0    0    user_coefs.coe

#
##################################################
# Short example, for generating coefficients only.
##################################################
# For only generating coefficients, copy the following into a config file (without
comments!),
# and modify as desired:

# RepoDir C:\Test
# ParseYUVFile 0
# EnableYUVOutputGeneration 0
# COEFileOut ModelOutput.coe
# generics  4   4   4   4   1   0
# test  1     0     127   0   127    128    128   4    4     0    0    0    user_coefs.coe
# test  1     0     127   0   127    120    120   4    4     0    0    0    user_coefs.coe
#
##################################################
# End of example.
##################################################
```

Initializing the Video Scaler Input Video Structure

In the example code wrapper, data is assigned to a video structure by reading from a .yuv video file. This file is described in C Model Example I/O Files. The `yuv_util.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O. The `run_bitacc_cmodel` example code uses these functions to read from the YUV file.

YUV Image Files

The header `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 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 );
```

Note: All image/video manipulation utility functions expect both input and output structures to be initialized. For example, pointing to a structure to which memory has been allocated, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data[] or y[],u[],v[]) 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 generate 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.

C Model Example I/O Files

Input Files

- **<config_file>** (For example, `scaler.cfg`).
  - Scaler configuration; described in previous section.
- **<input_filename>.yuv** (Optional; for example, `video_in.yuv`, `video_in_128x128.yuv`).
  - Standard 8-bit YUV file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player, such as YUVPlayer.
  - No header.
- **User defined coefficient file** (Optional; for example, `user_coefs.coe`).
  - File name specified for each test case.
    - This file is not necessary if you do not enable user defined coefficients for the test case. However, it is necessary to specify a dummy file name for each case.
  - The same or different files can be specified for each test case.
  - The file must contain only one set of coefficients, formatted correctly.
  - Correct format is defined in the *UG805 - LogiCORE IP Video Scaler User Guide*.
  - Xilinx recommends generating one .coe file (using this model) and using this file as an example, before generating your own.
  - Coefficients specified in this file are written to the coefficient output file.

Output Files

- **<output_filename>.yuv** (Optional; for example, `video_out.yuv`).
  - Standard 8-bit 4:2:2 yuv file format. Entire Y plane followed by entire Cb plane, followed by the entire Cr plane.
  - Can be viewed in a YUV player, such as YUVPlayer.
- **Coefficient output file** (For example, `ModelOutput.coe`).
  - File name specified.
  - Formatted correctly for direct use in CORE Generator.

- **Report.txt**
  - Information.
  - Warnings and errors.
- **Test_info.txt**
  - Individual test information.

Compiling the Video Scaler v4.0 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, as shown in this example:

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

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

   `libstlport.so.5.1`

   `libIp_v_scaler_v4_0_bitacc_cmodel.so`

   `libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so`

3. In the root directory, compile with the GNU C Compiler using this command:

```
gcc -x c++ run_bitacc_cmodel.c -o run_bitacc_cmodel -L. -
lIp_v_ycrcb2rgb_v3_0_bitacc_cmodel -lIp_v_scaler_v4_0_bitacc_cmodel -Wl,-rpath,.
```

4. This command creates the executable run_bitacc_cmodel, which can be run with this command:

```
./run_bitacc_cmodel <config_file>
```

Windows (32-bit)

Precompiled `library v_scaler_v4_0_bitacc_cmodel.lib`, and top-level demonstration code `run_bitacc_cmodel.c` should be compiled with an ANSI C compliant compiler under Windows. The following is an example using Microsoft Visual Studio. In Visual Studio create a new, empty Win32 Console Application project. To existing items, add:

- **libIp_v_scaler_v4_0_bitacc_cmodel.lib** to the "Resource Files" folder of the project
- **libIp_v_ycrcb2rgb_bitacc_model.lib** to the "Resource Files" folder of the project
- **run_bitacc_cmodel.c** to the "Source Files" folder of the project
- **v_scaler_v4_0_bitacc_cmodel.h** to "Header Files" folder of the project
- **v_ycrcb2rgb_v3_0_bitacc_cmodel.h** to the "Header Files" folder of the project
- **yuv_utils.h** to the "Header Files" folder of the project
- **rgb_utils.h** to the "Header Files" folder of the project
- **video_utils.h** to the "Header Files" folder of the project
- **xscaler_coefs.h** to the "Header Files" folder of the project

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 is created either in the Debug or Release subdirectories under the project location based on whether "Debug" or "Release" was selected in the "Configuration Manager" in the Build menu.

Running the Delivered Executables

Included in the zip file are precompiled executable files for use with Win32 and Linux64 platforms.

Lin64 Executable

1. Set your $LD_LIBRARY_PATH environment variable to include the root directory where you unzipped the model zip file, as shown in this example:

```
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_scaler_v4_0_bitacc_cmodel.so`

   `libIp_v_ycrcb2rgb_v3_0_bitacc_cmodel.so`

   `run_bitacc_cmodel`

3. Execute the model:

   `./run_bitacc_cmodel <config_file>`

Win32 Executable

1. Copy run_bitacc_cmodel.exe from the /Win32 directory to the root directory:
2. Execute the model:

   `run_bitacc_cmodel <config_file>`

Chapter 5

Generating Coefficients

Manual generation of the .coe file used by the Video Scaler CORE Generator GUI is complex and error prone. The C model includes a coefficient generation routine that generates coefficients in the correct format.

Several parameters influence the content of the output file:

- num_h_taps, num_v_taps
- max_phases
- num_h_phases, num_v_phases
- separate_YC_Coefs
- separate_HV_Coefs
- Input aperture
- Output size
- max_coef_sets
- Number of test cases defined in the config file

The model assesses these parameters as follows:

- The number of coefficient sets written to the file are always equal to **max_coef_sets**.
- One non-zero coefficient set **per test case** defined in the config file is written to the output file, even if that test runs for multiple frames.
  - If the number of defined test cases is fewer than max_coef_sets then the remaining sets are written as zeros.
- Each coefficient set comprises a number of banks equal to 4, 2 or 1. The number of banks is defined with this routine:

```
num_banks = 4;
if ((Separate_YC_Coefs == 0) or (Chroma_format = RGB_444))
   num_banks = num_banks/2;
end;
if (Separate_HV_Coefs == 0)
   num_banks = num_banks/2;
end;
```

- The size of all banks is equal to:

```
bank_size=Max(num_h_taps, num_v_taps) * max_phases;
```

- If num_h_taps is not equal to num_v_taps then zero padding is inserted accordingly.
- If num_x_phases < max_phases then zero padding is inserted accordingly.

The numerical content of the file is determined by the input aperture and output size settings. These values are processed to assess whether upscaling or downscaling coefficients are required. When downscaling, the level of anti-aliasing required in the filter can also affect the coefficients. This depends on the downscaling scale factor.

The following is a simple config file that shows a few simple test cases. In summary, there are four H taps, four V taps, four H phases and four V phases. Coefficient sharing is switched on in both HV and YC domains to shorten the example. No YUV file is parsed, and no YUV file is generated.

The first test case features equal input and output sizes (128Hx128V). The first phase of the first test illustrates this by using a unity set of coefficients (0, 0, 16384, 0). The second and third test cases exhibit different levels of downscaling. The final test case is an upscaling case.

RepoDir C:\Test

```
ParseYUVFile 0
EnableYUVOutputGeneration 0
COEFileOut ModelOutput.coe
generics  4   4   4   4   0   0
test    1     0     127   0   127    128    128   4    4     0    0    0
user_coefs.coe
test    1     0     127   0   127    120    120   4    4     0    0    0
user_coefs.coe
test    1     0     127   0   127    112    112   4    4     0    0    0
user_coefs.coe
test    1     0     127   0   127    256    256   4    4     0    0    0
user_coefs.coe
```

The resulting .coe file is given here (with added comments for clarity):

```
memory_initialization_radix=10;
memory_initialization_vector=
0,     1st Test; Phase 0; Tap 0;
0,     1st Test; Phase 0; Tap 1;
16384, 1st Test; Phase 0; Tap 2;
0,     1st Test; Phase 0; Tap 3;
-187,  1st Test; Phase 1; Tap 0;
4208,  1st Test; Phase 1; Tap 1;
12625, 1st Test; Phase 1; Tap 2;
-262,  1st Test; Phase 1; Tap 3;
-294,  1st Test; Phase 2; Tap 0;
8486,  1st Test; Phase 2; Tap 1;
8486,  1st Test; Phase 2; Tap 2;
-294,  1st Test; Phase 2; Tap 3;
-262,  1st Test; Phase 3; Tap 0;
12625, 1st Test; Phase 3; Tap 1;
4208,  1st Test; Phase 3; Tap 2;
-187,  1st Test; Phase 3; Tap 3;
-104,  2nd Test; Phase 0; Tap 0;
1018,  2nd Test; Phase 0; Tap 1;
15364, 2nd Test; Phase 0; Tap 2;
106,   2nd Test; Phase 0; Tap 3;
-240,  2nd Test; Phase 1; Tap 0;
4793,  2nd Test; Phase 1; Tap 1;
12022, 2nd Test; Phase 1; Tap 2;
-191,  2nd Test; Phase 1; Tap 3;
```

```
-282,  2nd Test; Phase 2; Tap 0;
8474,  2nd Test; Phase 2; Tap 1;
8474,  2nd Test; Phase 2; Tap 2;
-282,  2nd Test; Phase 2; Tap 3;
-191,  2nd Test; Phase 3; Tap 0;
12022, 2nd Test; Phase 3; Tap 1;
4793,  2nd Test; Phase 3; Tap 2;
-240,  2nd Test; Phase 3; Tap 3;
-252,  3rd Test; Phase 0; Tap 0;
2919,  3rd Test; Phase 0; Tap 1;
13413, 3rd Test; Phase 0; Tap 2;
303,   3rd Test; Phase 0; Tap 3;
-264,  3rd Test; Phase 1; Tap 0;
5751,  3rd Test; Phase 1; Tap 1;
10916, 3rd Test; Phase 1; Tap 2;
-19,   3rd Test; Phase 1; Tap 3;
-192,  3rd Test; Phase 2; Tap 0;
8384,  3rd Test; Phase 2; Tap 1;
8384,  3rd Test; Phase 2; Tap 2;
-192,  3rd Test; Phase 2; Tap 3;
-19,   3rd Test; Phase 3; Tap 0;
10916, 3rd Test; Phase 3; Tap 1;
5751,  3rd Test; Phase 3; Tap 2;
-264,  3rd Test; Phase 3; Tap 3;
-104,  4th Test; Phase 0; Tap 0;
1018,  4th Test; Phase 0; Tap 1;
15364, 4th Test; Phase 0; Tap 2;
106,   4th Test; Phase 0; Tap 3;
-240,  4th Test; Phase 1; Tap 0;
4793,  4th Test; Phase 1; Tap 1;
12022, 4th Test; Phase 1; Tap 2;
-191,  4th Test; Phase 1; Tap 3;
-282,  4th Test; Phase 2; Tap 0;
8474,  4th Test; Phase 2; Tap 1;
8474,  4th Test; Phase 2; Tap 2;
-282,  4th Test; Phase 2; Tap 3;
-191,  4th Test; Phase 3; Tap 0;
12022, 4th Test; Phase 3; Tap 1;
4793,  4th Test; Phase 3; Tap 2;
-240;  4th Test; Phase 3; Tap 3;
```