LogiCORE IP YCrCb to RGB Color-Space Converter 4.0

Product Guide

PG014 October 19, 2011

XILINX®

Table of Contents

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Debugging

Appendix C: C Model Reference

Appendix D: Additional Resources

LogiCORE IP YCrCb to RGB Color-Space Converter v4.0 Product Guide

Introduction

The YCrCb to RGB Color-Space Converter is a simplified 3x3 matrix multiplier converting three input color samples to three output samples in a single clock cycle. The optimized structure uses only four XtremeDSP™ slices by taking advantage of the dependencies between coefficients in the conversion matrix of most YCrCb or YUV to RGB standards.

Features

- Built in support for:
  - SD (*ITU 601*)
  - HD (*ITU 709*) PAL
  - HD (*ITU 709*) NTSC
  - YUV
- Support for user-defined conversion matrices
- Efficient use of DSP blocks
- 8-, 10-, and 12-bit input and output precision
- Delay match support for up to three sync signals

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex®-7, Kintex®-7, Virtex®-6, Spartan®-6
Supported User Interfaces	Constant Interface
Resources	See Table 1-1 through Table 1-4.
Provided with Core	
Documentation	Product Specification
Design Files	Netlists
Example Design	Not Provided
Test Bench	VHDL[2]
Constraints File	Not Provided
Simulation Models	VHDL and Verilog Structural, C Model [2]
Tested Design Tools	
Design Entry Tools	CORE Generator™ tool, Platform Studio (XPS)
Simulation[3]	Mentor Graphics ModelSim, Xilinx® ISim 13.3
Synthesis Tools	Xilinx Synthesis Technology (XST) 13.3
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. HDL test bench and C Model available on the product page on xilinx.com at: www.xilinx.com/products/ipcenter/YCrCb_to_RGB.htm
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.

Overview

A color space is a mathematical representation of a set of colors. The three most popular color models are:

- RGB or R'G'B', gamma corrected RGB, used in computer graphics
- YIQ, YUV and YCrCb used in video systems
- CMYK used in color printing

These color spaces are directly related to the intuitive notions of hue, saturation and brightness.

All color spaces can be derived from the RGB information supplied by devices such as cameras and scanners. Different color spaces have historically evolved for different applications. In each case, a color space was chosen for application-specific reasons.

A particular color space choice may be preferred because it requires less storage, bandwidth or computation in analog or digital domains.

The convergence of computers, the Internet, and a wide variety of video devices, all using different color representations, is forcing the digital designer today to convert between them. The objective is to have all inputs converted to a common color space before algorithms and processes are executed. Converters are useful for a number of markets, including image and video processing.

Applications

- Pre-processing block for image sensors
- Image compression
- Video surveillance
- Pre-processing block for video analytics

Licensing

The YCrCb to RGB core is provided at no cost with the ISE tools. You are not required to license the core before instantiating it in your design.

Performance

Maximum Frequencies

The following are typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the Field Programmable Gate Array (FPGA) device, using a different version of Xilinx tools, and other factors.

- Virtex®-7 FPGA: 250 MHz
- Kintex™-7 FPGA: 250 MHz
- Virtex-6 FPGA: 250 MHz
- Spartan®-6 FPGA: 150 MHz

Latency

The processing latency of the core is shown in the follow equation:

```
Latency = 5+ 1(if has clipping) + 1(if has clamping)
```

This example evaluates to seven clock cycles (Figure 2-2) for typical cases (unless in "custom" mode the clipping and/or clamping circuits are not used).

Throughput

The Color Space Converter core outputs one RGB sample per clock cycle.

Resource Utilization

The design was tested using ISE® 13.3 tools with default tool options for characterization data. As resource counts are functions of tool options (such as XST optimizations, map packing factor), the actual resource counts corresponding to the quoted operating frequencies are also listed.

For an accurate measure of the usage of device resources (for example, block RAMs, flip-flops, and LUTs) for a particular instance, click **View Resource Utilization** in CORE Generator software after generating the core.

The YcrCb to RGB core does not use any block RAMs or dedicated I/O or clock resources.

Table 1-1 through Table 1-4 provide the resource usage of the YCrCb to RGB core for different families with default parameterization for all permitted input/output width combinations.

Table 1-1: **Spartan-6 XC6SLX4 xc6slx4,-2 (PRODUCTION 1.20 2011-09-27) CPG196**

Input Width	Output Width	LUTs	FFs	DSP48E1	Clock Frequency (MHz)
8	8	62	161	4	210
8	10	77	185	4	210
8	12	64	173	4	210

Table 1-1: **Spartan-6 XC6SLX4 xc6slx4,-2 (PRODUCTION 1.20 2011-09-27) CPG196**

10	8	62	169	4	210
10	10	84	197	4	205
10	12	72	185	4	210
12	8	62	177	4	210
12	10	84	205	4	195
12	12	81	197	4	210

Table 1-2: **Virtex-6 XC6VLX75T xc6vlx75t,-1 (PRODUCTION 1.15 2011-09-27) FF484**

Input Width	Output Width	LUTs	FFs	DSP48E1	Clock Frequency (MHz)
8	8	74	161	4	353
8	10	88	185	4	353
8	12	68	173	4	353
10	8	73	169	4	369
10	10	92	197	4	369
10	12	80	185	4	369
12	8	69	177	4	369
12	10	94	205	4	369
12	12	88	197	4	369

Table 1-3: **Virtex-7 XC7V585T xc7v585t,-1 (ADVANCED 1.02 2011-09-27) FFG1157**

Input Width	Output Width	LUTs	FFs	DSP48E1	Clock Frequency (MHz)
8	8	70	161	4	365
8	10	84	185	4	365
8	12	76	173	4	365
10	8	71	169	4	384
10	10	90	197	4	375
10	12	76	185	4	384
12	8	69	177	4	389
12	10	89	205	4	384
12	12	92	197	4	389

-

Table 1-4: **Kintex-XC7K70T 7xc7k70t,-1 (ADVANCED 1.02 2011-09-27) FBG484**

Input Width	Output Width	LUTs	FFs	DSP48E1	Clock Frequency (MHz)
8	8	69	161	4	328
8	10	81	185	4	370
8	12	78	173	4	370
10	8	69	169	4	377
10	10	88	197	4	377
10	12	82	185	4	377
12	8	69	177	4	386
12	10	96	205	4	386
12	12	92	197	4	386

Chapter 2

Core Interfaces and Register Space

Core Symbol and Port Descriptions

The YCrCb to RGB core uses a set of signals that is common to all of the Xilinx Video cores called the Xilinx Streaming Video Interface (XSVI). This core has no ports other than the Xilinx Streaming Video Interface, clk, ce, and sclr signals. The core symbol with the clk, ce, sclr, and XSVI signals is shown in Figure 2-1 and described in Table 2-1.

Xilinx Streaming Video Interface

The Xilinx Streaming Video Interface (XSVI) is a set of signals common to all of the Xilinx video cores used to stream video data between IP cores. XSVI can also be defined as an Embedded Development Kit (EDK) bus type. This allows the EDK tool to automatically create input and output connections to EDK pCores that include this interface definition, and provide an easy way to cascade connections of Xilinx Video IP cores.

Note: The YCrCb to RGB core is not currently available with a pCore interface. Consequently, the core cannot be directly added to an EDK project and the tool cannot directly recognize the XSVI bus type. To use this core in an EDK project, you must import the core (see Importing Color Space Conversion Cores into EDK as pCore with XSVI Bus) and define the signals as an XSVI bus type. The tool allows easy connection of the signals to other video IP cores with XSVI bus type.

The YCrCb to RGB IP Core uses the following subset of the XSVI signals:

- video_data
- vblank
- hblank
- active_video

Other XSVI signals on the XSVI bus, such as video_clk, vsync, hsync, field_id, and active_chr do not affect the function of this core.

Note: These signals are neither propagated, nor driven on the XSVI output of this core.

Importing Color Space Conversion Cores into EDK as pCore with XSVI Bus

1. Parameterize and generate the core.
2. Create a wrapper file, using the provided instantiation template, either the .veo or .vho file.
3. Open EDK and follow the Create and Import Peripheral Wizard. This tool is documented in the UG111: Embedded System Tools Reference Manual.
4. Modify the .mpd file created by the Create and Import Peripheral Wizard. This file is in the Data directory created by the Create and Import Peripheral Wizard.

You must define the XSVI bus type and appropriately tag the signals as shown in the following example. IWIDTH and OWIDTH are the values you selected when you generated the IP in Core Generator. (i.e. 8,10, or 12)

Input Side:

```
BUS_INTERFACE BUS    = XSVI_YCRCB2RGB_IN, BUS_STD = XSVI, BUS_TYPE
= TARGET

PORT active_video_in = active_video, DIR = IN,  BUS =
XSVI_YCRCB2RGB_IN

PORT hblank_in       = hblank,       DIR = IN,  BUS =
XSVI_YCRCB2RGB_IN

PORT vblank_in       = vblank,       DIR = IN,  BUS =
XSVI_YCRCB2RGB_IN

PORT video_data_in   = video_data,   VEC = [0:((IWIDTH*3)-1)], DIR
= IN, BUS = XSVI_YCRCB2RGB_IN
```

Output Side:

```
BUS_INTERFACE BUS = XSVI_YCRCB2RGB_OUT, BUS_TYPE = INITIATOR,
BUS_STD = XSVI

PORT active_video_out = active_video, DIR = OUT, BUS =
XSVI_YCRCB2RGB_OUT

PORT hblank_out       = hblank,       DIR = OUT, BUS =
XSVI_YCRCB2RGB_OUT

PORT vblank_out       = vblank,       DIR = OUT, BUS =
XSVI_YCRCB2RGB_OUT

PORT video_data_out   = video_data, VEC = [0:((OWIDTH*3)-1)], DIR =
OUT, BUS = XSVI_YCRCB2RGB_OUT
```

For more information on the MPD format, see UG642: Platform Specification Format Reference Manual

The YCrCb to RGB IP core is fully synchronous to the core clock, clk. Consequently, the input XSVI bus is expected to be synchronous to the input clock, clk. Similarly, to avoid clock resampling issues, the output XSVI bus for this IP is synchronous to the core clock, clk. The video_clk signals of the input and output XSVI buses are not used.

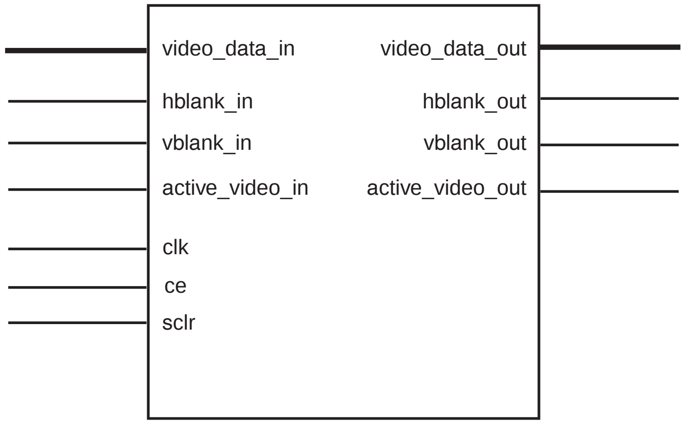

Figure 2-1: **Core Symbol**

Table 2-1: **Port Descriptions for the YCrCb to RGB Core**

Port Name	Port Width	Direction	Description
video_data_in	3*IWIDTH	IN	Data input bus
hblank_in	1	IN	Horizontal blanking input
vblank_in	1	IN	Vertical blanking input
active_video_in	1	IN	Active video signal input
video_data_out	3*OWIDTH	OUT	Data output bus
hblank_out	1	OUT	Horizontal blanking output
vblank_out	1	OUT	Vertical blanking output
active_video_out	1	OUT	Active video signal output
clk	1	IN	Rising-edge clock
ce	1	IN	Clock enable (active high)
sclr	1	IN	Synchronous clear – reset (active high)

- **video_data_in**: This bus contains the three individual inputs in the following order. Luminance and chrominance values are expected in IWIDTH bits wide unsigned integer representation.

Bits	3IWIDTH-1:2IWIDTH	2IWIDTH-1:IWIDTH	IWIDTH-1:0
Video Data Signals	Cb	Cr	Y

- **hblank_in:** The `hblank_in` signal conveys information about the blank / non-blank regions of video scan lines. This signal is not actively used in the core, but passed through the core with a delay matching the latency of the corrected data.
- **vblank_in**: The `vblank_in` signal conveys information about the blank / non-blank regions of video frames. This signal is passed through the core with a delay matching the latency of the corrected data.

- **active_video_in**: The `active_video_in` signal is high when valid data is presented at the input. This signal is not actively used in the core, but passed through the core with a delay matching the latency of the corrected data.
- **clk - clock**: Master clock in the design, synchronous with, or identical to the video clock.
- **ce - clock enable**: Pulling `CE` low suspends all operations within the core. Outputs are held, no input signals are sampled, except for reset (`SCLR` takes precedence over `CE`).
- **sclr - synchronous clear:** Pulling `SCLR` high results in resetting all output ports to zero. Internal registers within the XtremeDSP slice and D-flip-flops are cleared. However, the core uses `SRL16`/`SRL32` based delay lines for `hblank`, `vblank` and `active_video` generation, which are not cleared by `SCLR`. This may result in non-zero outputs after `SCLR` is deasserted, until the contents of `SRL16`/`SRL32`s are flushed. Unwanted results can be avoided if `SCLR` is held active until `SRL16`/`SRL32`s are flushed. `SCLR` should be held active for the duration of the processing latency of the core. The latency is defined in the Control Signals and Timing section.
- **video_data_out**: This bus contains the three individual RGB color outputs in the following order from MSB to LSB [red: blue: green]. Color values are expected in OWIDTH bits wide unsigned integer representation.

Bits	3OWIDTH-1:2OWIDTH	2OWIDTH-1:OWIDTH	OWIDTH-1:0
Video Data Signals	Red	Blue	Green

- **hblank_out** and **vblank_ou**t: The corresponding input signals are delayed so blanking outputs are in phase with the video data output, maintaining the integrity of the video stream. The blanking outputs are connected to the corresponding inputs via delay-lines matching the propagation delay of the video processing pipe. Unwanted blanking inputs should be tied high, and corresponding outputs left unconnected, which will result in the trimming of any unused logic within the core.
- **active_video_out**: The `active_video_out` signal is high when valid data is present at the output. The `active_video_out` signal is connected to `active_video_in` via delay-lines matching the propagation delay of the video processing pipe. The `active_video` signal does not affect the processing behavior of the core. Asserting or deasserting it will not stall processing or the video stream, nor will it force video outputs to zero.

Control Signals and Timing

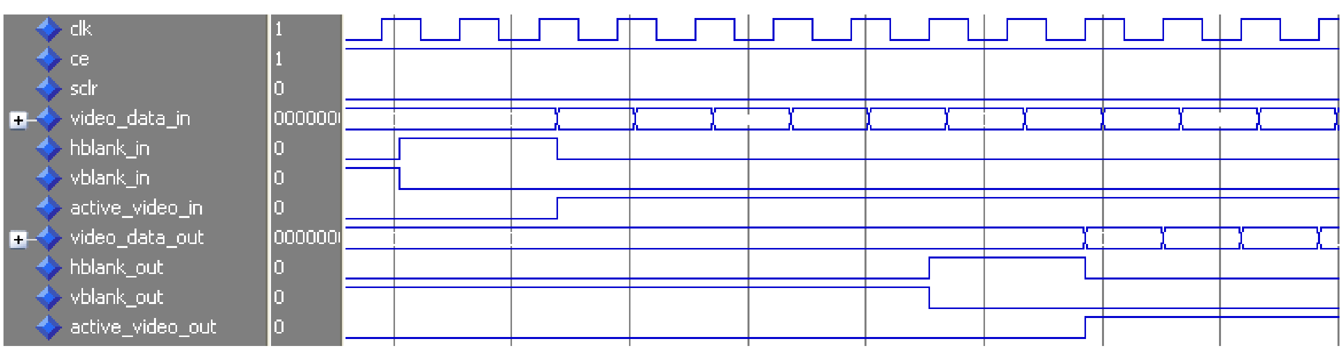

Figure 2-2: **Timing Example**

The propagation delay of the YcrCb to RGB core is dependent on parameterization but independent of actual signal (`video_data`, `hblank`, `vblank`, `active_video`) values. Deasserting `CE` suspends processing, which may be useful to temporarily cease processing of a video stream in order to match the delay of other processing components.

See Core Symbol and Port Descriptions, page 9, for an explanation on how other ports may affect the timing behavior of the core.

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

Graphical User Interface (GUI)

The main screen of the Graphical User Interface (GUI) of the CORE Generator, shown in Figure 3-1, allows quick implementation of standard YCrCb to RGB or YUV to RGB converters without having to manually enter values from Table 4-2, Table 4-3 and Table 4-4. The Color-Space Converter core also supports proprietary (non-standard) converter implementations, by selecting “custom” from the Standard Selection drop-down menu, as long as the custom conversion matrix can be transformed to the form of Equation 4-3. The front pages of the RGB to YCrCb and the YCrCb to RGB GUIs are very similar, allowing easy generation of complementing converter pairs.

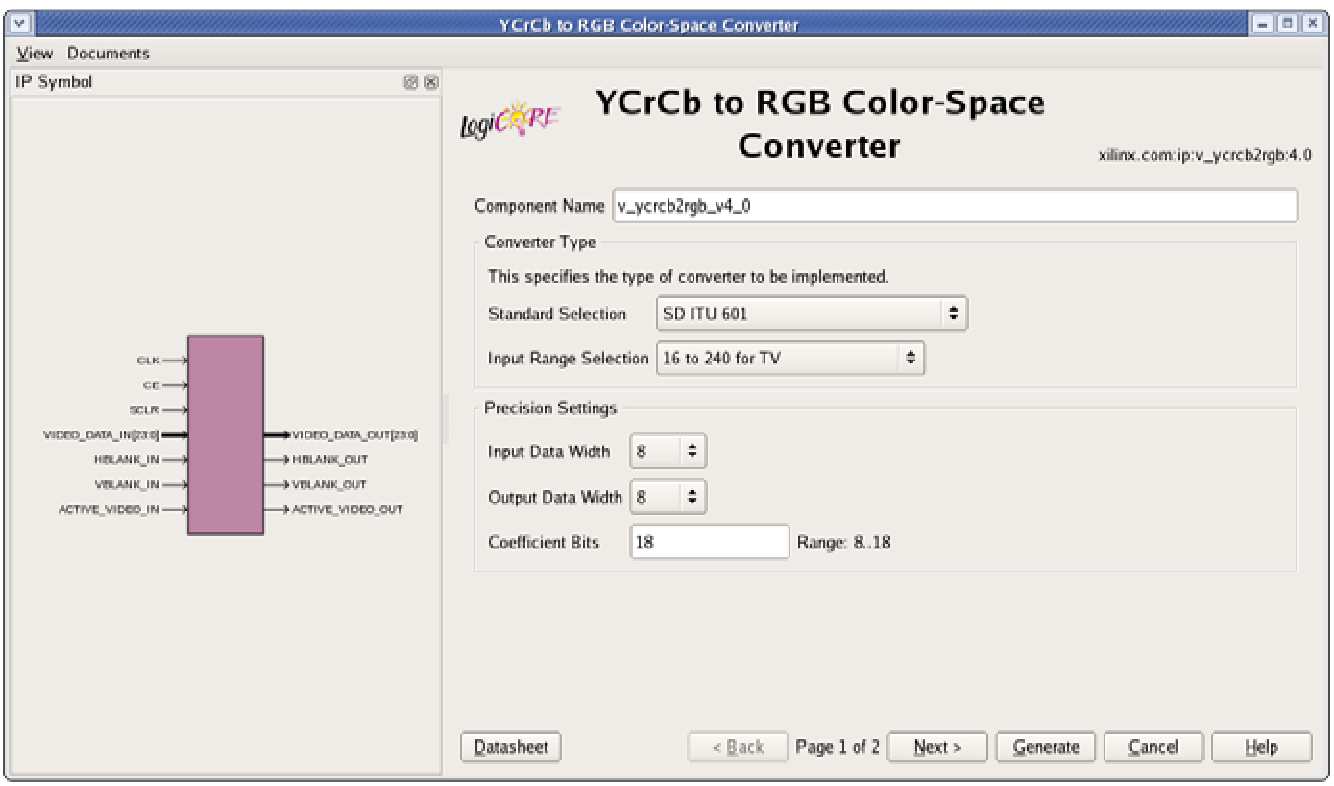

Figure 3-1: **Color Space Converter Main Screen**

- **Component Name**: The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_".
- Converter Type
  - **Standard Selection**: Select the standard to be implemented. The offered standards are:
    - YCrCb *ITU 601* (SD)
    - YCrCb *ITU 709* (HD) 1125 / 60 (PAL)
    - YCrCb *ITU 709* (HD) 1250 / 50 (NTSC)
    - YUV
    - custom

  Selecting "custom" enables the controls on page 2 of the GUI, so conversion settings can be customized. Otherwise, page 2 only displays the parameters to implement the selected standard.

  - **Output Range Selection**: This selection governs the range of outputs R, G and B by affecting the conversion coefficients as well as the clipping and clamping values. The core supports typical output ranges:
    - 16 to 235, typical for studio equipment
    - 16 to 240, typical for broadcast or television
    - 0 to 255, typical for computer graphics

    The previously-mentioned ranges are characteristic for 8-bit outputs. If 10- or 12-bit outputs are used, the ranges are extended proportionally. For example, 16 to 240 mode for 10-bit outputs will result in output values ranging from 64 to 960.
- Precision Settings
  - **Input Width** (IWIDTH): This field specifies the width of inputs Y, Cr and Cb.
  - **Output Width** (OWIDTH): This field specifies the width of outputs R, G and B.
  - **Coefficient Bits**: Sets the number of bits used to represent CA, CB, CC and CD. As displayed on Figure 4-2, the width of coefficients affects the width of multiplier results, which may affect the size of fabric-based adders further down the processing pipe. Reducing the coefficient size may save some slices by trading off precision with logic resources.

The Conversion Matrix, Offset Compensation, Clipping and Clamping screen (Figure 3-2), displays and enables editing of conversion coefficients, similar to Equation 4-3. Contents are editable only when "custom" is selected as the standard on page 1 (Figure 3-1).

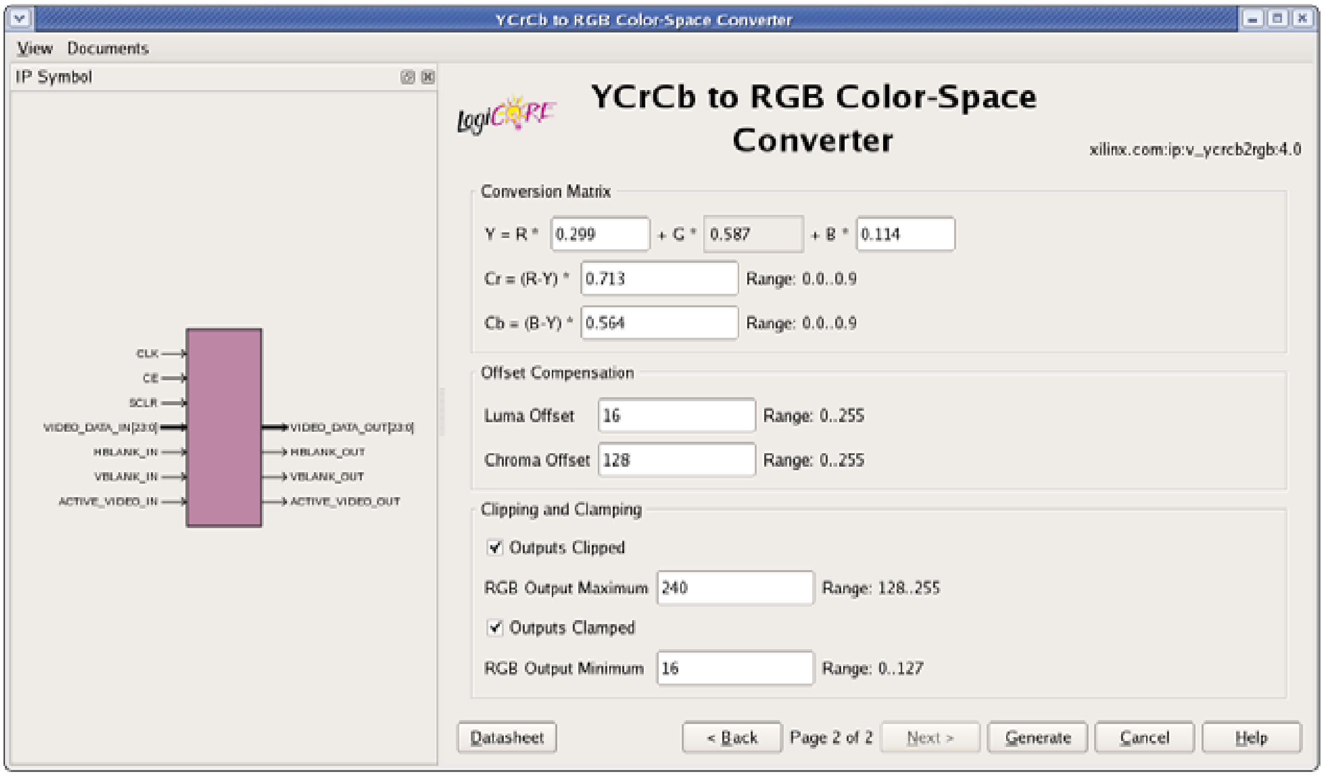

Figure 3-2: **Conversion Matrix, Offset Compensation, Clipping and Clamping Screen**

- **Conversion Matrix**: Enter floating-point conversion constants, ranging from 0 to 1, into the four fields representing CA, CB, CC and CD.
- **Offset Compensation**: Enter the offset compensation constants (OY and OC in Equation 4-17, Equation 4-18 and Equation 4-19). These constants are scaled to the output representation. If OY and OC are in the 0.0 - 1.0 range, and the output is represented as 10-bit unsigned integers, then luminance and chrominance offsets should be entered as integers in the 0 - 1023 range.
- **Outputs Clipped/Outputs Clamped**: These check boxes control whether clipping/clamping logic will be instantiated in the generated netlist. The clipping/clamping logic ensures no arithmetic wrap-arounds happen at overflows, at the expense of extra slice-based logic resources.
- **Minimum and Maximum Values**: Similar to offset values, the edit boxes take unsigned integer values in the range permitted by the current output representation.

Parameter Values in the XCO File

Table 3-1 defines valid entries for the XCO parameters. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator software GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 3-1: **XCO Parameters**

XCO Parameter	Default Values
component_name	v_ycrcb2rgb_v4_0
iwidth	8
owidth	8
mwidth	18
cwidth	17
ca	0.299
cb	0.114
cc	0.713
c_d	0.564
rgbmin	16
rgbmax	240
coffset	128
yoffset	16
has_clamp	true
has_clip	true
input_range	16_to_240_for TV
standard_sel	SD_ITU_601

Output Generation

CORE Generator will output the core as a netlist that can be inserted into a processor interface wrapper or instantiated directly in an HDL design. The output is placed in the <project director>.

File Details

The CORE Generator output consists of some or all the following files.

Name	Description
<component_name>_readme.txt	Readme file for the core.
<component_name>.ngc	The netlist for the core.
<component_name>.veo	The HDL template for instantiating the core.
<component_name>.vho	

Name	Description
<component_name>.v	The structural simulation model for the core. It is used for functionally simulating the core.
<component_name>.vhd	
<component_name>.xco	Log file from CORE Generator software describing which options were used to generate the core. An XCO file can also be used as an input to the CORE Generator software.

Chapter 4

Designing with the Core

RGB Color Space

The red, green and blue (RGB) color space is widely used throughout computer graphics. Red, green and blue are three primary additive colors: individual components are added together to form a desired color, and are represented by a three dimensional, Cartesian coordinate system, as shown in Figure 4-1.

Table 4-1 presents the RGB values for 100% saturated color bars, a common video test signal.

Table 4-1: **100% RGB Color Bars**

	Normal Range	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
R	0 to 255	255	255	0	0	255	255	0	0
G	0 to 255	255	255	255	255	0	0	0	0
B	0 to 255	255	0	255	0	255	0	255	0

The RGB color space is the most prevalent choice for computer graphics because color displays use red, green and blue to create the desired color. Also, a system that is designed using the RGB color space can take advantage of a large number of existing software algorithms.

However, RGB is not very efficient when dealing with real-world images. All three components need equal bandwidth to generate arbitrary colors within the RGB color cube. Also, processing an image in the RGB color space is usually not the most efficient method. For example, to modify the intensity or color of a given pixel, all three RGB values must be read, modified and written back to the frame buffer. If the system had access to the image stored in the intensity and color format, the process would be faster.

R'G'B' Color Space

While the RGB color space is ideal to represent computer graphics, 8-bit linear-light coding performs poorly for images to be viewed [Ref 2]. It is necessary to have 12 or 14 bits per component to achieve excellent quality. The best perceptual use is made of a limited number of bits by using nonlinear coding that mimics the nonlinear response of human vision. In video, JPEG, MPEG, computing, digital photography, and many other domains, a nonlinear transfer function is applied to the RGB signals to give nonlinearly coded gamma-corrected components, denoted with symbols R'G'B'. Excellent image quality can be obtained with 10-bit nonlinear coding with a transfer function similar to that of *Rec. 709* [Ref 4] or RGB.

YUV Color Space

The YUV color space is used by the analog PAL, NTSC and SECAM color video / TV standards. In the past, black and white systems used only the luminance (Y) information. Chrominance information (U and V) was added in such a way that a black and white receiver can still display a normal black and white picture.

YCrCb (or YCbCr) Color Space

The YCrCb or YCbCr color space was developed as part of the *ITU-R BT.601* [Ref 3] during the development of a world-wide digital component video standard. YCbCr is a scaled, offset version of the YUV color space. Y has a nominal range of 16-235; Cb and Cr have a nominal range of 16-240. There are several YCbCr sampling formats, such as 4:4:4, 4:2:2 and 4:2:0.

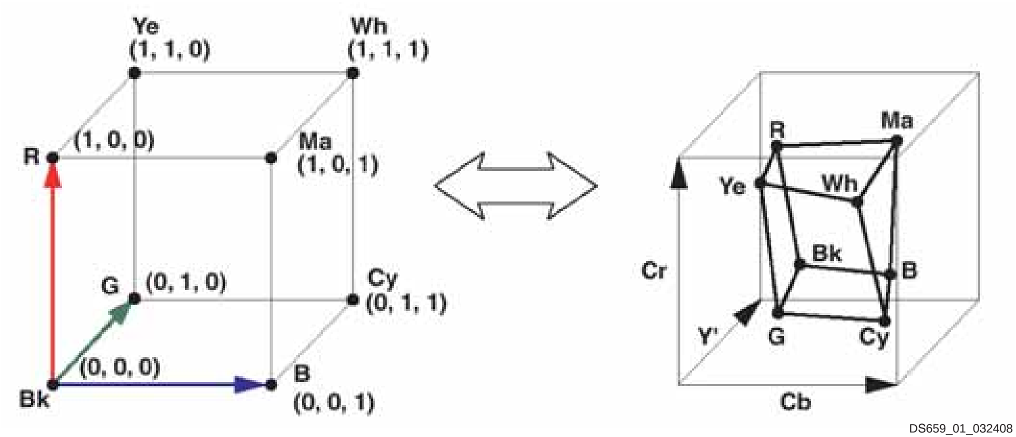

Figure 4-1: **RGB and YCrCb Color Representations**

Conversion Equations

Derivation of Conversion Equations

To generate the luminance (Y, or gray value) component, biometric experiments were employed to measure how the human eye perceives the intensities of the red, green and blue colors. Based on these experiments, optimal values for coefficients CA and CB were determined, such that:

$$Y = CA*R + (1 - CA - CB)*G + CB*B \qquad \text{Equation 4-1}$$

Actual values for CA and CB differ slightly in different standards.

Conversion from the RGB color space to luminance and chrominance (differential color components) could be described with the following equation:

$$\begin{bmatrix} Y \\ R-Y \\ B-Y \end{bmatrix} = \begin{bmatrix} CA & 1-CA-CB & CB \\ 1-CA & CA+CB-1 & -CB \\ -CA & CA+CB-1 & 1-CB \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \text{Equation 4-2}$$

Coefficients CA and CB are chosen between 0 and 1, which guarantees that the range of Y is constrained between the maximum and the minimum RGB values permitted, RGB_{max} and RGB_{min} respectively.

In most practical implementations, the range of the luminance and chrominance components should be equal. There are two ways to accomplish this: the chrominance components (B-Y and R-Y) can be normalized (compressed and offset compensated), or values above and below the luminance range can be clipped / clamped.

Both clipping and dynamic range compression results in loss of information; however, the introduced artifacts are different. To leverage differences in the input (RGB) range, different standards choose different tradeoffs between clipping and normalization.

The YCrCb to RGB Color-Space Converter core supports only the conversions that fit the following general form:

$$\begin{bmatrix} Y \\ C_R \\ C_B \end{bmatrix} = \begin{bmatrix} CA & 1-CA-CB & CB \\ CC(1-CA) & CC(CA+CB-1) & CC(-CB) \\ CD(-CA) & CD(CA+CB-1) & CD(1-CB) \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} O_Y \\ O_C \\ O_C \end{bmatrix} \qquad \text{Equation 4-3}$$

CC and CD allow dynamic range compression for B-Y and R-Y, and constants O_Y and O_C facilitate offset compensation for the resulting CB and CR. To avoid arithmetic under- and overflows while converting from the RGB to the YCrCb domain, with RGB values in the [0.1] range, a choice for CC and CD is:

$$CC = \frac{1}{2(1-CA)} \qquad CD = \frac{1}{2(1-CB)} \qquad \text{Equation 4-4}$$

The YCrCb to RGB core facilitates both range de-compression and optional clipping and clamping. Range, offset, clipping and clamping levels are parameterizable.

By inverting the transformation matrix in Equation 4-3, the transformation from the YCrCb color space to the RGB color space can be defined as:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1/CC & 0 \\ 1 & \dfrac{-CA}{CC(1-CA-CB)} & \dfrac{-CB}{CD(1-CA-CB)} \\ 1 & 0 & 1/CD \end{bmatrix} \begin{bmatrix} Y-O_Y \\ C_R-O_C \\ C_B-O_C \end{bmatrix} \qquad \text{Equation 4-5}$$

Hardware Implementation

The YCrCb to RGB color-space transformation (Equation 4-5) can be expressed as:

$$R = Y - O_Y + ACOEF(C_R - O_C) \quad \text{Equation 4-6}$$

$$G = Y - O_Y + BCOEF(C_R - O_C) + CCOEF(C_B - O_C) \quad \text{Equation 4-7}$$

$$B = Y - O_Y + DCOEF(C_B - O_C) \quad \text{Equation 4-8}$$

This cannot efficiently utilize the MADD capabilities of XtremeDSP slices. As offsets and coefficients are constants, the preceding equations can be rewritten as:

$$R = ACOEF \bullet C_R + ROFFSET + Y \quad \text{Equation 4-9}$$

$$G = BCOEF \bullet C_R + CCOEF \bullet C_B + GOFFSET + Y \quad \text{Equation 4-10}$$

$$B = DCOEF \bullet C_B + BOFFSET + Y \quad \text{Equation 4-11}$$

This can be directly mapped to the architecture shown in Figure 4-2. The blue and gray boxes represent logic blocks, which are always implemented using XtremeDSP slices.

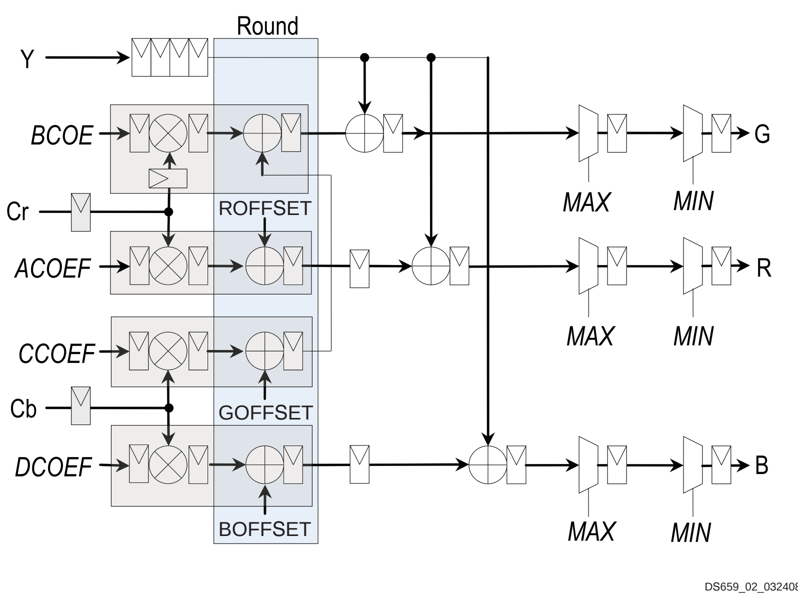

Figure 4-2: **YcrCb to RGB Schematic**

Assigning Values to Design Parameters

The following section specifies parameter values for some widely used standards. Most parameter values, except for COEF and OFFSET parameters, can be assigned from Table 4-2, Table 4-3 and Table 4-4 directly. These parameters have to be calculated, scaled and rounded before assigning integer values to corresponding VHDL parameters, using the following equations:

$$ACOEF = \frac{1}{CC}$$ *Equation 4-12*

$$BCOEF = \frac{-CA}{CC(1-CA-CB)}$$ *Equation 4-13*

$$CCOEF = \frac{-CB}{CD(1-CA-CB)}$$ *Equation 4-14*

$$DCOEF = \frac{1}{CD}$$ *Equation 4-15*

Coefficients are passed to the core in CWIDTH bits wide two's complement format. Y, Cr and Cb are passed as IWIDTH bits wide unsigned integers. After multiplication, in devices without DSP blocks, results are truncated to MWIDTH bits. For these families, MWIDTH is user definable, and for families with DSP blocks, MWIDTH is preset to IWIDTH + CWIDTH.

$$\text{ROUNDING\_CONST} = 2^{MWIDTH-OWDITH-2}$$ *Equation 4-16*

Equation 4-17

$$ROFFSET = \text{ROUNDING\_CONST} - (ACOEF \bullet COFFSET + YOFSET) \bullet \text{SCALE\_M}$$

Equation 4-18

$$GOFFSET = \text{ROUNDING\_CONST} - ((BOEF + CCOEF) \bullet COFFSET + YOFFSET) \bullet \text{SCALE\_M}$$

Equation 4-19

$$BOFFSET = \text{ROUNDING\_CONST} - (DCOEF \bullet COFFSET + YOFFSET) \bullet \text{SCALE\_M}$$

$$\text{SCALE\_M} = 2^{MWDITH-IWIDTH-CWIDTH}$$ *Equation 4-20*

ITU 601 (SD) and 709 - 1125/60 (NTSC) Standard Conversion Coefficients

Table 4-2: **Parameterization Values for the SD (ITU 601) and NTSC HD (ITU 709) Standards**

Coefficient/ Parameter	Range		
	16-240	16-235	0-255
CA	0.299		0.2568
CB	0.114		0.0979
CC	0.713	0.7295	0.5910
CD	0.564	0.5772	
YOFFSET	$2^{OWIDTH-4}$		
COFFSET	$2^{OWIDTH-1}$		
YMAX	$240*2^{OWIDTH-8}$		$235*2^{OWIDTH-8}$

Table 4-2: **Parameterization Values for the SD (ITU 601) and NTSC HD (ITU 709) Standards *(Cont'd)***

Coefficient/ Parameter	Range		
	16-240	16-235	0-255
CMAX	$240*2^{OWIDTH-8}$		$235*2^{OWIDTH-8}$
YMIN	$16*2^{OWIDTH-8}$	0	$2^{OWIDTH}-1$
CMIN	$16*2^{OWIDTH-8}$	0	$2^{OWIDTH}-1$

Standard ITU 709 (HD) 1250/50 (PAL)

Table 4-3: **Parameterization Values for the PAL HD (ITU 709) Standard**

Coefficient/ Parameter	Input range		
	16-240	16-235	0-255
CA	0.2126		0.1819
CB	0.0722		0.0618
CC	0.6350	0.6495	0.6495
CD	0.5389	0.5512	
YOFFSET	$2^{OWIDTH-4}$		
COFFSET	$2^{OWIDTH-1}$		
YMAX	$240*2^{OWIDTH-8}$	$235*2^{OWIDTH-8}$	$2^{OWIDTH}-1$
CMAX	$240*2^{OWIDTH-8}$	$235*2^{OWIDTH-8}$	$2^{OWIDTH}-1$
YMIN	$16*2^{OWIDTH-8}$		0
CMIN	$16*2^{OWIDTH-8}$		0

YUV Standard

Table 4-4: **Parameterization Values for the YUV Standard**

Coefficient/ Parameter	Value		
	16-240	16-235	0-255
CA	0.299		
CB	0.114		
CC	0.877283		
CD	0.492111		
YOFFSET	$2^{OWIDTH-4}$		
COFFSET	$2^{OWIDTH-1}$		
YMAX	$240*2^{OWIDTH-8}$	$235*2^{OWIDTH-8}$	$2^{OWIDTH}-1$
CMAX	$240*2^{OWIDTH-8}$	$235*2^{OWIDTH-8}$	$2^{OWIDTH}-1$
YMIN	$16*2^{OWIDTH-8}$		0
CMIN	$16*2^{OWIDTH-8}$		0

Clipping and Clamping

Output Quantization Noise

Coefficients CC and CD in Equation 4-3 allow standard designers to trade off output quantization and clipping noise. Actual noise inserted depends on the probability statistics of the Cb and Cr variables, but in general if CC and CD are larger than the maximum values calculated in Equation 4-4, output values may clip, introducing clipping noise. However, the lower CC and CD values are chosen, the worse Cb and Cr values will use the available dynamic range, thus introducing more quantization noise. Therefore, the designer's task is to equalize output quantization and clipping noise insertion by carefully choosing CC and CD values based on knowing the statistics of Cb and Cr values. For instance, when probabilities of extreme chrominance values are small, it is beneficial to increase CC and CD values, as the extra noise inserted by occasional clipping is less than the gain in average signal power (and thus SQNR).

Output Clipping Noise

If coefficients CC and CD in Equation 4-3 are larger than the maximum values calculated in Equation 4-4, Cr and Cb output values may get larger (overflow) than the maximum or smaller (underflow) than minimum value the output representation can carry. If overflow occurs and the design does not have clipping logic, binary values wrap around and insert substantial noise to the output. If clamping/clipping logic is used, output values saturate and less noise is introduced, as shown in Figure 4-3. Use of clipping and clamping increases slice count of the design by approximately 6*OWIDTH slices.

If a targeted standard limits output of values to a predefined range other than those of binary representation, such as *ITU-R BT.601-5* [Ref 3], use of clipping and clamping logic facilitates constraining output values to the predefined range by setting RGB_{max} and RGB_{min} values according to the standard specifications.

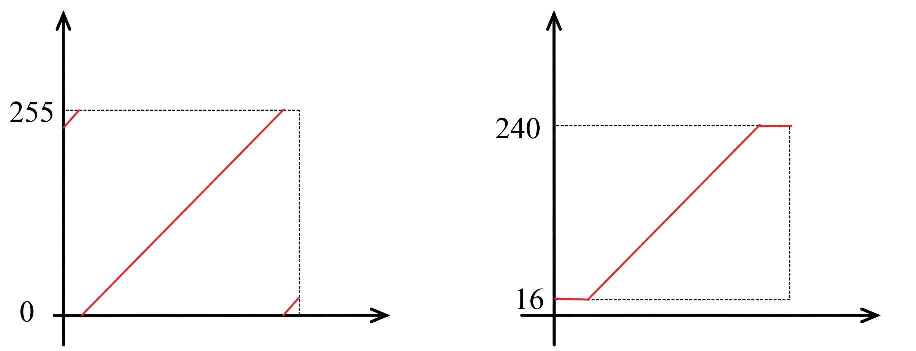

Figure 4-3: **Wrap-Around and Saturation**

Clocking

The Color Space Converter core has one clock ("clk") that is used to clock the entire core.

Resets

The Color Space Converter core has one reset ("sclr") that is used for the entire core. The reset is active high.

Chapter 5

Constraining the Core

Required Constraints

The clk pin should be constrained at the maximum pixel clock rate desired for the video stream.

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for this core. This core has not been characterized for use in low power devices.

Clock Frequencies

The clk pin should be run at the required pixel clock frequency for the YCrCb to RGB core. See Maximum Frequency in Performance in Chapter 1.

Clock Management

There is only one clock for this core.

Clock Placement

There are no specific clock placement requirements for this core.

Banking

There are no specific banking rules for this core.

Transceiver Placement

There are no transceivers used in this core.

I/O Standard and Placement

There are no specific I/O standard or placement requirements.

Detailed Example Design

Demonstration Test Bench

Overview

This chapter describes how to use the files that come with the demo testbench package for YCrCb to RGB Color-Space Converter v4.0.

This demo testbench is provided as a simple introductory package that enables core users to observe the core generated by Coregen operating in a waveform simulator. The user is encouraged to observe core-specific aspects in the waveform, make simple modifications to the test conditions, and observe the changes in the waveform.

Software Tools and System Requirements

- Xilinx ISE 13.3 or higher (Includes XST, ISIM, and Coregen).
- ModelSim v6.6d
- ISE Simulator 13.3

Design File Hierarchy

The directory structure underneath this top-level folder is described below:

- Expected
  - Contains the pre-generated expected / golden data used by the testbench to compare actual output data.
- Stimuli
  - Contains the pre-generated input data used by the testbench to stimulate the core (including register programming values).
- Results
  - Actual output data is written to a file in this folder.
- src
  - Contains the .vhd & .xco files of the core.

  The .vhd file is a netlist generated using Coregen.

  You can regenerate a new netlist using the .xco file in Coregen.
- tb_src
  - Contains the top-level testbench design.

  This directory also contains other packages used by the testbench.
- isim_wave.wcfg - Waveform configuration for ISIM
- mti_wave.do - Waveform configuration for ModelSim
- run_isim.bat - Runscript for iSim in Windows OS
- run_isim.sh - Runscript for iSim in Linux OS
- run_mti.bat - Runscript for ModelSim in Windows OS
- run_mti.sh - Runscript for ModelSim in Linux OS

Operating Instructions

- Simulation using ModelSim for Linux:
  From the console, Type "source run_mti.sh".
- Simulation using ModelSim for Windows:
  Double click on "run_mti.bat" file.
- Simulation using iSim for Linux:
  From the console, Type "source run_isim.sh".
- Simulation using iSim for Windows:
  Double click on "run_isim.bat" file.

Support

To obtain technical support for this reference design, go to www.xilinx.com / support to locate answers to known issues in the Xilinx Answers Database or to create a WebCase.

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench used to test the Color Space Converter core. Testing includes the following:

- Testing various coefficients
- Testing the clipping and clamping
- Testing of various data widths

Debugging

Evaluation Core Timeout

The Color Space Converter hardware evaluation core times out after approximately 8 hours of operation. The output is driven to zero. This results in a black screen for RGB systems and a dark-green screen for YUV color systems.

See Solution Centers in Appendix D for information helpful to the debugging progress.

Appendix C

C Model Reference

This document introduces the bit accurate C model for the Xilinx® LogiCORE™ IP YCrCb to RGB Color-Space Converter v4.0 core, which has been developed primarily for system modeling.

Features

- Bit accurate with the YCrCb to RGB Color-Space Converter v4.0 core
- Statically linked library (.lib, .o, .obj – Windows)
- Dynamically linked library (.so – Linux)
- Available for 32- and 64-bit Windows and 32- 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 v4.0 has a bit accurate C model for 32- and 64-bit Windows and 32- 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). An example piece of C code is provided to show how to call the model.

The model is bit accurate, as it produces exactly the same output data as the core on a frame-by-frame basis. However, the model is not cycle accurate, as it does not model the core's latency or its interface signals.

The latest version of the model is available for download on the Xilinx™ LogiCORE IP YCrCb to RGB Color-Space Converter web page at:

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

Unpacking and Model Contents

Unzip the v_ycrcb2rgb_v4_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 C-1.

Table C-1: **Directory Structure and Files of the YCrCb to RGB Color-Space Converter v4.0 Bit Accurate C Model**

File Name	Contents
README.txt	Release notes
pg014_v_ycrcb2rgb.pdf	LogiCORE IP YCrCb to RGB Color-Space Converter Product Guide
v_ycrcb2rgb_v4_0_bitacc_cmodel.h	Model header file
rgb_utils.h	Header file declaring the RGB image / video container type and support functions
yuv_utils.h	Header file declaring the YUV (.yuv) image file I / O functions
bmp_utils.h	Header file declaring the bitmap (.bmp) image file I / O functions
video_utils.h	Header file declaring the generalized image / video container type, I / O and support functions
run_bitacc_cmodel.c	Example code calling the C model
kodim19_128x192.bmp	128x192 sample test image of the lighthouse image from the True Color Kodak test images
/ lin64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Linux platforms
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so
/ nt32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Windows platforms.
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.lib	Precompiled library file for win32 compilation
/ lin32	Precompiled bit accurate ANSI C reference model for simulation on 32-bit Linux platforms
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so	Model shared object library
libstlport.so.5.1	STL library, referenced by libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so

Table C-1: **Directory Structure and Files of the YCrCb to RGB Color-Space Converter v4.0 Bit Accurate C Model**

/ nt64	Precompiled bit accurate ANSI C reference model for simulation on 64-bit Windows platforms.
libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.lib	Precompiled library file for nt64compilation

Installation

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

- `libIp_v_ycrcb2rgb_v4_0_bitacc_cmodel.so`
- `libstlport.so.5.1`

Software Requirements

The YCrCb to RGB Color-Space Converter v4.0 C models were compiled and tested with the software listed in Table C-2.

Table C-2: **Compilation Tools for the Bit Accurate C Models**

Platform	C Compiler
32- and 64-bit Linux	GCC 4.1.1
32- and 64-bit Windows	Microsoft Visual Studio 2005

The bit accurate C model is accessed through a set of functions and data structures that are declared in the `v_ycrcb2rgb_v4_0_bitacc_cmodel.h` file. Before using the model, the structures holding the inputs, generics and output of the YCrCb to RGB Color-Space Converter instance must be defined:

```
struct  xilinx_ip_v_ycrcb2rgb_v4_0_generics generics;
struct  xilinx_ip_v_ycrcb2rgb_v4_0_inputs inputs;
struct  xilinx_ip_v_ycrcb2rgb_v4_0_outputs outputs;
```

The declaration of these structures is in the `v_ycrcb2rgb_v4_0_bitacc_cmodel.h` file. Table C-3 lists the generic parameters taken by the YCrCb to RGB Color-Space Converter v4.0 IP core bit accurate model, as well as the default values.

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

Generic Variable	Type	Default Value	Range	Description
IWIDTH	int	8	8,10,12	Input data width
CWIDTH	int	18	8-18	Coefficient bits
OWIDTH	int	8	8,10,12	Output width
ACOEF	double	0.299	0.0 - 1.0	A Coefficient[1] 0.0 < ACOEFF + BCOEFF < 1.0
BCOEF	double	0.114	0.0 - 1.0	B Coefficient[1] 0.0 < ACOEFF + BCOEFF < 1.0
CCOEF	double	0.713	0.0 - 0.9	C Coefficient[1]

Table C-3: **Core Generic Parameters and Default Values *(Cont'd)***

DCOEF	double	0.564	0.0 - 0.9	D Coefficient[1]
YOFFSET	int	16	$0 - 2^{OWIDTH}-1$	Offset for the Luminance Channel
COFFSET	int	128	$0 - 2^{OWIDTH}-1$	Offset for the Chrominance Channels
YMIN	int	16	$0 - 2^{OWIDTH-1}-1$	Clamping value for the Luminance Channel
CMIN	int	16	$0 - 2^{OWIDTH-1}-1$	Clamping value for the Chrominance Channels
YMAX	int	240	$2^{OWIDTH-1} - 2^{OWIDTH}-1$	Clipping value for the Luminance Channel
CMAX	int	240	$2^{OWIDTH-1} - 2^{OWIDTH}-1$	Clipping value for the Chrominance Channels

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

Calling `xilinx_ip_v_ycrcb2rgb_v4_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_v4_0_get_default_inputs(&generics, &inputs)` initializes the input video structure before it can be assigned an image or video sequence using the memory allocation or file I/O functions provided in the BMP, RGB or video utility functions.

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

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

```
int xilinx_ip_v_ycrcb2rgb_v4_0_bitacc_simulate(
struct xilinx_ip_v_ycrcb2rgb_v4_0_generics* generics,
struct xilinx_ip_v_ycrcb2rgb_v4_0_inputs*   inputs,
struct xilinx_ip_v_ycrcb2rgb_v4_0_outputs*  outputs).
```

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

```
void xilinx_ip_v_ycrcb2rgb_v4_0_destroy(
struct xilinx_ip_v_ycrcb2rgb_v4_0_inputs *input,
struct xilinx_ip_v_ycrcb2rgb_v4_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 v4.0 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 C-4: **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 C-5.
data	Set of five pointers to three dimensional arrays containing data for image planes. Data is in 16-bit unsigned integer format accessed as data[plane][frame][row][col].

Table C-5: **Named Video Modes with Corresponding Planes and Representations[1]**

Mode	Planes	Video Representation
FORMAT_MONO	1	Monochrome – Luminance only
FORMAT_RGB	3	RGB image / video data
FORMAT_C444	3	444 YUV, or YCrCb image / video data
FORMAT_C422	3	422 format YUV video, (u, v chrominance channels horizontally sub-sampled)
FORMAT_C420	3	420 format YUV video, (u, v sub-sampled both horizontally and vertically)
FORMAT_MONO_M	3	Monochrome (Luminance) video with Motion
FORMAT_RGBA	4	RGB image / video data with alpha (transparency) channel
FORMAT_C420_M	5	420 YUV video with Motion
FORMAT_C422_M	5	422 YUV video with Motion
FORMAT_C444_M	5	444 YUV video with Motion
FORMAT_RGBM	5	RGB video with Motion

[1] The Color Space Conversion core C model supports FORMAT_C444 for input data and FORMAT_RGB for output data.

Initializing the Input Video Structure

The easiest way to assign stimuli values to the input video structure is to initialize it with an image or video. The `yuv_utils.h`, `bmp_util.h` and `video_util.h` header files packaged with the bit accurate C models contain functions to facilitate file I/O.

Bitmap Image Files

The header `bmp_utils.h` declares functions that help access files in Windows Bitmap format (http://en.wikipedia.org/wiki/BMP_file_format). However, this format limits color depth to a maximum of 8-bits per pixel, and operates on images with three planes (R,G,B). Consequently, the following functions operate on arguments type `rgb8_video_struct`, which is defined in `rgb_utils.h`. Also, both functions support only true-color, non-indexed formats with 24-bits per pixel.

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

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

```
int copy_rgb8_to_video(struct rgb8_video_struct* rgb8_in,
                       struct video_struct* video_out );
int copy_video_to_rgb8(struct video_struct* video_in,
                       struct rgb8_video_struct* rgb8_out );
```

Note: All image/video manipulation utility functions expect both input and output structures initialized; for example, pointing to a structure that has been allocated in memory, either as static or dynamic variables. Moreover, the input structure must have the dynamically allocated container (data or r, g, b) structures already allocated and initialized with the input frame(s). If the output container structure is pre-allocated at the time of the function call, the utility functions verify and issue an error if the output container size does not match the size of the expected output. If the output container structure is not pre-allocated, the utility functions create the appropriate container to hold results.

Binary Image/Video Files

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

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

These functions serialize the `video_struct` structure. The corresponding file contains a small, plain text header defining, "Mode", "Frames", "Rows", "Columns", and "Bits per Pixel". The plain text header is followed by binary data, 16-bits per component in scan line continuous format. Subsequent frames contain as many component planes as defined by the video mode value selected. Also, the size (rows, columns) of component planes can differ within each frame as defined by the actual video mode selected.

YUV Image Files

The `yuv_utils.h` file declares functions that help access files in standard YUV format. It operates on images with three planes (Y, U and V). The following functions operate on arguments of type `yuv8_video_struct`, which is defined in `yuv_utils.h`:

```
int write_yuv8(FILE *outfile, struct yuv8_video_struct *yuv8_video);

int read_yuv8(FILE *infile, struct yuv8_video_struct *yuv8_video);
```

Exchanging data between `yuv8_video_struct` and general `video_struct` type frames / videos is facilitated by these functions:

```
int copy_yuv8_to_video(struct yuv8_video_struct* yuv8_in,
                       struct video_struct* video_out );
int copy_video_to_yuv8(struct video_struct* video_in,
                       struct yuv8_video_struct* yuv8_out );
```

Working with Video_struct Containers

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

```
int video_planes_per_mode(int mode);
int video_rows_per_plane(struct video_struct* video, int plane);
int video_cols_per_plane(struct video_struct* video, int plane);
```

The `video_planes_per_mode` function returns the number of component planes defined by the mode variable, as described in Table C-5. 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:...
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 (32- and 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_v4_0_bitacc_cmodel.so`

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

   ```
   gcc -m32 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_ycrcb2rgb_v4_0_bitacc_cmodel -Wl,-rpath,.

   gcc –m64 -x c++ ../run_bitacc_cmodel.c ../parsers.c -o
   run_bitacc_cmodel -L. -lIp_v_ycrcb2rgb_v4_0_bitacc_cmodel -Wl,-rpath,.
   ```

Windows (32- and 64-bit)

The precompiled library `v_ycrcb2rgb_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. An example procedure is provided here using Microsoft Visual Studio.

1. In Visual Studio, create a new, empty Console Application project.
2. As existing items, add:
   a. `libIp_v_ycrcb2rgb_v4_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_v4_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 an 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 D

Additional Resources

Xilinx Resources

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

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.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

1. Jack, Keith. 2004. *Video Demystified*, 4th Edition. Burlington, MA: Newnes: pp 15-19.
2. Poynton, Charles. 2003. *Digital Video and HDTV*. San Francisco: Morgan Kaufmann: pp 302 - 321.
3. *ITU Recommendation BT.601-5*, International Telecommunication Union, 1995.
4. *ITU Recommendation BT.709-5*, International Telecommunication Union, 2002.
5. Proakis, John G., and Dimitris G. Manolakis. *Digital Signal Processing*, 3rd edition. Upper Saddle River, NJ: Prentice Hall: pp 755-756.
6. Sullivan, Gary. 2003. Approximate theoretical analysis of RGB to YCbCr to RGB conversion error. Presented for Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6), July 22-24, in Trondheim, Norway
7. AXI Reference Guide.

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Ordering Information

The YCrCb to RGB Color-Space Converter v4.0 core is provided under the Xilinx End User License Agreement and can be generated using the Xilinx® CORE Generator™ system. The CORE Generator system is shipped with Xilinx ISE® Design Suite software.

A simulation evaluation license for the core is shipped with the CORE Generator system. To access the full functionality of the core, including FPGA bitstream generation, a full license must be obtained from Xilinx. For more information, visit the product page for this core.

Contact your local Xilinx sales representative for pricing and availability of additional Xilinx LogiCORE IP modules and software. Information about additional Xilinx LogiCORE IP modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/19/2011	1.0	Initial Xilinx release.

Notice of Disclaimer