XAPP637 (v1.0) September 12, 2002

Color Space Converter: R'G'B' to Y'CbCr

Author: Benoit Payette

Summary

This application note describes the implementation of R'G'B' Color Space to Y'CbCr Color Space conversion necessary in many video designs. The tick marks on red, green, blue, and Luma, assume the components are in the gamma corrected space. No gamma correction is applied to color difference signals Cr and Cb.

The reference design files show RTL (VHDL and Verilog) code to describe the conversion equations and synthesize to a target FPGA. The code is parameterizable for the input/output precision (8 bit or 10 bit) and the internal coefficient precision (8 to 13 bits have been defined). Simulation test vectors (25%, 50%, 75%, and 100% RGB Color Bars) are also provided in the reference design file to confirm compliance to ITU-R BT.601-2 component video standards (SDTV)[3].

As an implementation example, placed and routed design in a Spartan-IIE device (2S50E-6TQ144) takes about 20% of that device (150 slices) and clock performance of 99 MHz using simple constraints (8-bit input/output and 8-bit internal coefficients).

HDTV color space coefficients are different (covered in BT.709-3, June 1990). This application note does not cover this area.

Color Space Definition

Different color spaces have historically evolved for different applications. In each case, a color space was chosen for applicable reasons. A choice was made on a particular color space because the math elements needed to process were simpler or faster. A a certain choice was better because it required less storage and bandwidth on digital buses.

Whatever historical reasons caused color space choices in the past, 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 are converted to a common color space before algorithms and processes are executed. Converters are useful for a number of markets, including image processing and filtering. The converters' basic function is to convert from one color space to another. This application note describes one such conversion.

Three-Color Space Examples

RGB Color Space

RGB color space is a simple and robust color definition used in computer systems and the Internet to help ensure correct mapping of a color from one platform to another without significant loss of color information. RGB uses three numerical components to represent a color. This color space can be thought of as a three-dimensional coordinate system whose axes correspond to the three components, R or red, G or green, and B or Blue. RGB is the color space used by computer displays. RGB corresponds most closely to the behavior of the human eye.

RGB is an additive color system. The three primary colors red, green, and blue are added to form the desired color. Gamma-corrected values are noted R'G'B'. Each component has a

range of 0 to 255 (for a 8-bit representation), with all three 0s producing black and all three 255s producing white. Figure 1 shows the RGB color cube (on the left) with the eight corners.

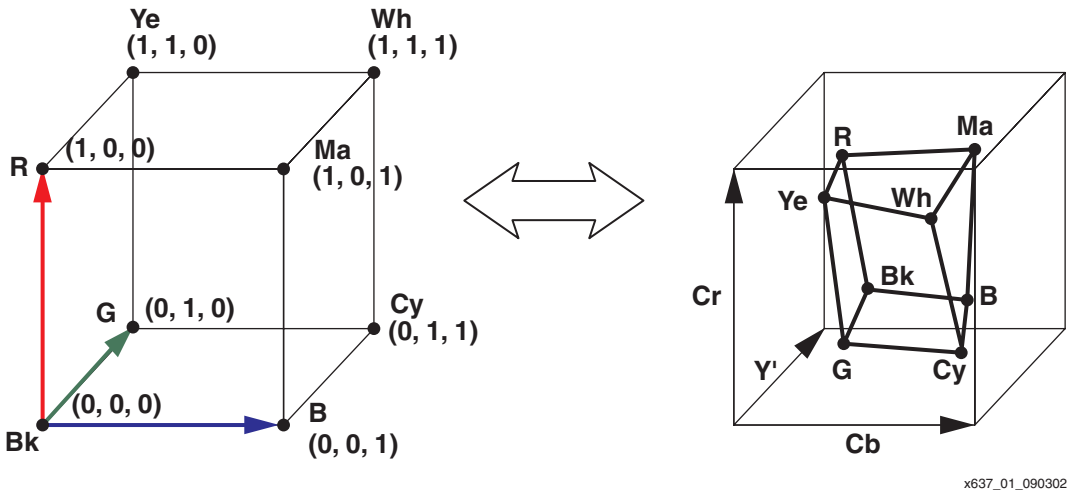

Figure 1: **RGB and Y'CbCr Color Cubes**

Y'CbCr Color Space

Y'CbCr Color Space was developed as part of the Recommendation ITU-R BT.601 for worldwide digital component video standard and is used in television transmissions. Y'CbCr is a scaled and offset version of the YUV color space where Y represents luminance (or brightness), U represents color, and V represents the saturation value. Here the RGB color space is separated into a luminance part (Y') and two chrominance parts (Cb and Cr).

The historical reasons for this choice, over R'G'B', were to reduce storage and bandwidth. Since the eye is more sensitive to change in brightness than change in color, the reduction in bandwidth requirement seemed a valid trade for little or no visual difference.

Engineers found 60% to 70% of luminance or brightness is in the "green color." In the chrominance part Cb and Cr, the brightness information can be removed from the blue and red colors.

To generate the same color in the RGB format, all three color components should be of equal bandwidth. This requires more storage space and bandwidth. Also, processing an image in the RGB space is more complex since any change in the color of any pixel requires all the three RGB values to be read, calculations performed, and then stored. If the color information is stored in the intensity and color format, some of the processing steps can be made faster.

As a result, Cb and Cr provide the hue and saturation information of the color and Y' provides the brightness information of the color. Y' is defined to have a range of 16 to 235 and Cb and Cr have a range of 16 to 240 with 128 equal to zero. [1] [2] [4] Because the eye is less sensitive to Cb and Cr, engineers did not need to transmit Cr and Cb at the same rate as Y'. Less storage and bandwidth was needed, resulting in design costs being reduced.

Figure 1 shows the Y'CbCr color cube (on the right) with the eight corners converted from the RGB color cube.

Converting from R'G'B' to Y'CbCr

A color in the R'G'B' color space is converted to the Y'CbCr color space using the following equations: [1] [2]

$$Y' = 16 + (0.257R + 0.504G + 0.098B)$$
$$Cb = 128 + (-0.148R - 0.291G + 0.439B)$$
$$Cr = 128 + (0.439R - 0.368 - 0.071B)$$

Conversion Equations

where R'G'B' are gamma-corrected RGB values, all input and output signals are 8-bit values.

Figure 2 shows a direct mapping of these three equations. There is no obvious duplication to reduce the implementation. However, it is possible to reorganize the color difference equations to reduce the number of constant multipliers. The actual reference design does not develop these.

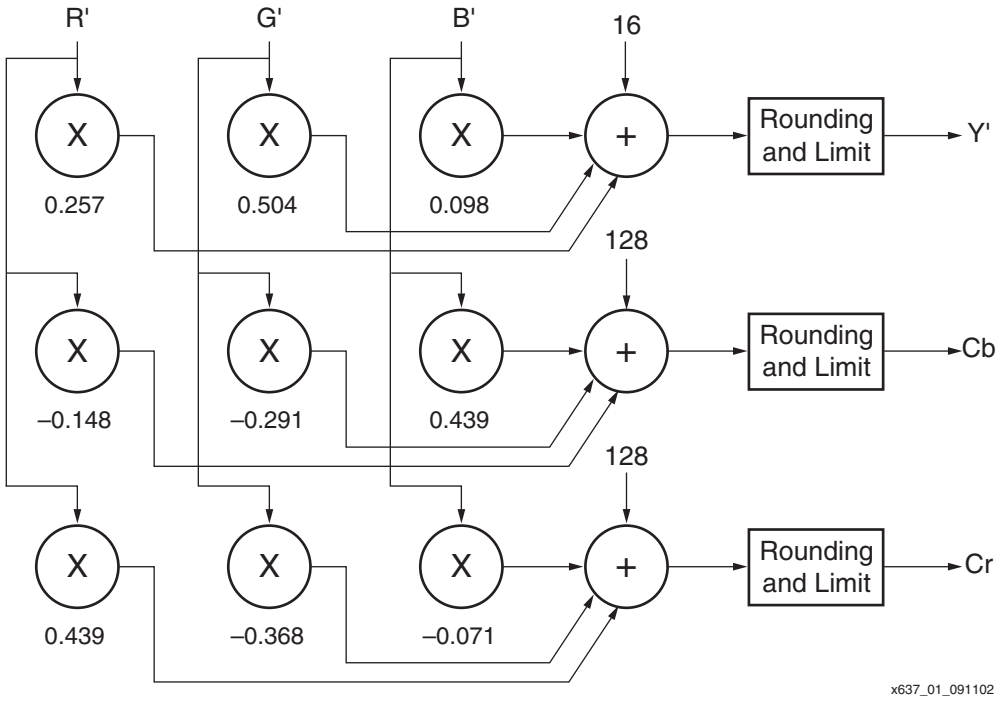

Figure 2: **General Block Diagram Showing Math Elements**

The recommendation ITU-R BT.601-2 defines the rate for Y, Cb, and Cr to be transmitted in a 4:2:2 sampling scheme (i.e., for every four samples of Y, only two samples of Cb and Cr are transmitted.) If 8 bits per component are used, a 16-bit system is required. Effectively, this is an 8/24 = 1/3 saving in bandwidth. The sampling of Y'CbCr is shown below.

Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	…
Cb0	Cr0	Cb2	Cr2	Cb4	Cr4	Cb6	Cr6	…

The 4:2:2 is the standard for digital studio equipment (usually 27 MHz clock for SDTV, 75 MHz for HDTV). However, this reference design will not implement the 4:2:2 sampling scheme. Instead, it implements the 4:4:4 sampling scheme as defined in the Appendix A of ITU-R BT.601-2 that matches the input data rates.

Code Implementation of the Color Space Conversion (csc.vhd or csc.v)

The **Conversion Equations** and Figure 2 show a general but inefficient flow. Rounding usually looks at the decimal value and if it is greater than or equal to 0.5, then the result is increased by one. This implies a condition to verify and another operation. A more efficient way to round a number is to add 0.5 to the result and truncate the decimal value. This operation is included in the constant number.

The results of the constant multiplier are not added in a large tree of adders to get the final output. Instead, the following equations are used to ease future pipelining and to avoid negative internal numbers. All input/output signals are assumed to be 8-bit values. For 10-bit values, the development is similar (**Appendix A**).

$$Y' = \text{round and limit}\{16 + (0.257R + 0.504G + 0.098B)\}$$
$$Cb = \text{round and limit}\{128 + (-0.148R - 0.291G + 0.439B)\}$$
$$Cr = \text{round and limit}\{128 + (0.439R - 0.368G - 0.071B)\}$$

$$Y' = \text{truncate}\{16.5 + (0.257R + 0.504G + 0.098B)\}$$
$$Cb = \text{truncate}\{128.5 + (-0.148R - 0.291G + 0.439B)\}$$
$$Cr = \text{truncate}\{128.5 + (0.439R - 0.368G - 0.071B)\}$$

$$Y' = \text{truncate}\{[(16.5 + 0.504G) + (0.257R + 0.098B)]\}$$
$$Cb = \text{truncate}\{[(128.5 + 0.439B) - (0.148R + 0.291G)]\}$$
$$Cr = \text{truncate}\{[(128.5 + 0.439R) - (0.368G + 0.071B)]\}$$

First, the constant multipliers are computed, then two adders are used in parallel to compute the inner results, then a last adder or subtractor is used to compute the final result. The way the internal elements are grouped will never produce a negative number. Only three clock cycles (the latency) are required to compute the result.

Internal Precision

The reference design has the capability to set the internal precision for the coefficients of Y', Cb, and Cr. The minimum precision is 8 bits and the code offers a coefficient precision of up to 13 bits. When the test vectors are used to test the code, the output results would probably differ by ±1 compared to a calculator's answer for 8-bit signals. This only depends on the internal precision of the coefficient. To satisfy the 25%, 50%, 75%, and 100% RGB color bars, use a 13-bit Y', an 11-bit Cb, and a 10-bit Cr (see the following details). As a comparison, the 8-bit precision is presented in the next section **Reference Design Implementation**.

$$Y' = \text{truncate}\{[(16.5 + 0.504G) + (0.257R + 0.098B)]\}$$

If scaled up by 8192 (= 2^{13} or 13-bit range)

$$\Rightarrow Y' = (1/8192) \times [(135168 + 4129G) + (2105R + 803B)]$$

$$Cb = \text{truncate}\{[(128.5 + 0.439B) - (0.148R + 0.291G)]\}$$

If scaled by 2048 (= 2^{11} or 11-bit range)…

$$\Rightarrow Cb = (1/2048) \times [(263168 + 899B) - (303R + 596G)]$$

$$Cr = \text{truncate}\{[(128.5 + 0.439R) - (0.368G + 0.071B)]\}$$

If scaled by 1024 (= 2^{10} or 10-bit range)…

$$\Rightarrow Cr = (1/1024) \times [(131584 + 450R) - (377G + 73B)]$$

Reference Design Implementation

Using RTL Code (csc_top.vhd or csc_top.v)

In this implementation, the R'G'B' to Y'CbCr conversion equations were synthesized. All the signals are registered at the input and at the output to simulate a real system (thus a total of five clock cycles). The synthesized netlist file was placed and routed using ISE4.2i SP3. A simple timing constraint of 12.5 ns was given to the synthesis and place and route tools. The implementation results are listed in Table 1.

An 8-bit I/O requires 51 ports (eight for each color input and output, one for each clock, clock enable and reset). A 10-bit I/O requires 63 ports (ten for each color input and output, one for each clock, clock enable and reset).

The VHDL and Verilog reference designs for this application note are available on the Xilinx FTP site at **ftp://ftp.xilinx.com/pub/applications/xapp/xapp637.zip**. The design files include RTL code, an auto-verifiable testbench and an excel spreadsheet about coefficient precision.

Table 1: **RTL Implementation Design Summary (from VHDL code)**

Device	In/Out Precision (bits	Coefficient Precision (Y'/Cb/Cr)	Number of Slices (% of Device)	MIN Period in ns (MAX Frequency)
Virtex Device XCV50-4TQ144	8	8/8/8	150/768 (20%)	12.051 (83 MHz)
		13/11/10	236/768 (31%)	12.405 (81 MHz)
	10	8/8/8	173/768 (23%)	12.361 (81 MHz)
		13/11/10	283/768 (37%)	15.784 (63 MHz)
Spartan-II Device XC2S30-5TQ144	8	8/8/8	150/432 (35%)	10.903 (92 MHz)
		13/11/10	236/432 (55%)	12.378 (81 MHz)
	10	8/8/8	173/432 (40%)	10.698 (93 MHz)
		13/11/10	283/432 (66%)	13.415 (75 MHz)
Virtex-E Device XCV50E-6CS144	8	8/8/8	150/768 (20%)	9.844 (102 MHz)
		13/11/10	236/768 (31%)	11.959 (84 MHz)
	10	8/8/8	173/768 (23%)	10.421 (96 MHz)
		13/11/10	283/768 (37%)	12.911 (77 MHz)
Spartan-IIE Device XC2S50E-6TQ144	8	8/8/8	150/768 (20%)	10.148 (99 MHz)
		13/11/10	236/768 (30%)	11.863 (84 MHz)
	10	8/8/8	173/768 (22%)	10.605 (94 MHz)
		13/11/10	283/768 (37%)	13.316 (75 MHz)
Virtex-II Device XC2V80-4CS144	8	8/8/8	150/768 (19%)	10.734 (93 MHz)
		13/11/10	236/768 (30%)	10.151 (99 MHz)
	10	8/8/8	173/768 (22%)	9.647 (104 MHz)
		13/11/10	283/768 (36%)	11.986 (83 MHz)
Virtex-II Pro Device XC2VP2-6FG256	8	8/8/8	150/768 (19%)	7.489 (134 MHz)
		13/11/10	236/768 (30%)	7.323 (137 MHz)
	10	8/8/8	173/768 (22%)	7.170 (139 MHz)
		13/11/10	283/768 (36%)	8.177 (122 MHz)

Conclusion

The result of the synthesis and implementation demonstrates the converter functionality. The RTL VHDL or Verilog code describing the conversion equations uses available resources in the Virtex, Spartan-II, Virtex-E, Spartan-IIE, and Virtex-II devices. No block RAMs or embedded multipliers are consumed. Faster implementations are possible using more pipelines or by using dedicated cores (constant multipliers). A different approach with color difference equations can also be explored.

References

The basic color space conversion equations are defined in ITU-R BT.601 standard. Other noted references are listed.

1. Keith Jack, Video Demystified: A Handbook for the Digital Engineer, Third Edition (Eagle Rock, VA: LLH Technology Publishing, 2001) ISBN 1-878707-95-7
2. Charles Poynton, Frequently Asked Questions about Color (www.inforamp.net/~poynton, 12/30/1999) PDF document on the web
3. ITU-R Recommendation BT.601-2, Encoding Parameters of Digital Television for Studios (1982-1986-1990), [formerly CCIR Rec. 601-2] (Geneva: ITU, 1990)
4. Charles Poynton, A Guided Tour of Color Space (www.inforamp.net/~poynton, 8/19/1997) PDF document on the web

Appendix A

This appendix has the calculations for color space conversion of 10-bit values. The development is similar to the 8-bit values (page 3).

$$Y' = \text{round and limit}\{64 + (0.257R + 0.504G + 0.098B)\}$$
$$Cb = \text{round and limit}\{512 + (-0.148R - 0.291G + 0.439B)\}$$
$$Cr = \text{round and limit}\{512 + (0.439R - 0.368G - 0.071B)\}$$

$$Y' = \text{truncate}\{64.5 + (0.257R + 0.504G + 0.098B)\}$$
$$Cb = \text{truncate}\{512.5 + (-0.148R - 0.291G + 0.439B)\}$$
$$Cr = \text{truncate}\{512.5 + (0.439R - 0.368G - 0.071B)\}$$

$$Y' = \text{truncate}\{[(64.5 + 0.504G) + (0.257R + 0.098B)]\}$$
$$Cb = \text{truncate}\{[(512.5 + 0.439B) - (0.148R + 0.291G)]\}$$
$$Cr = \text{truncate}\{[(512.5 + 0.439R) - (0.368G + 0.071B)]\}$$

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/12/02	1.0	Initial Xilinx release.