



XAPP283 (v1.3.1) March 24, 2005

## Color Space Converter: Y'CrCb to R'G'B'

Author: Latha Pillai

### Summary

This application note describes three ways to implement the Y'CrCb Color Space to R'G'B' 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 first implementation shows how to simply write behavioral Verilog to describe the conversion equations, and then synthesize to a silicon target. This technique infers MULT\_ANDs for the constant coefficient multiplier.

The second implementation uses the Xilinx feature of embedded RAM functioning as a Look-up Table (LUT), or ROM, to store all possible intermediate results for the terms in the three equations. Since three of the seven total terms are identical, only five ROMs are needed. The depth of the ROM, 1K, is driven by the color component bit width of 10 bits or studio quality video. To target Spartan-II devices, either add more ROMs or use commercial 8-bit video instead of 10-bit studio quality.

The third implementation makes use of the embedded multiplier in the Virtex™-II series of devices to perform the color space conversion. Again, only five multipliers are used. The Verilog model using the embedded multiplier is synthesized, placed, and routed. The design has a clock performance of 185 MHz after place and route, using simple constraints.

### Color Space Definition

The human eye has three types of photoreceptor cells called cones. Stimulating the cells causes the human brain to “perceive” color. Colors can be specified, created, and visualized using different color formats or “color spaces.”

Different color spaces have historically evolved for different applications. In each case, a color space was chosen for reasons that may no longer be applicable. Maybe a choice was made on a particular color space because the math elements needed to process were simpler or faster. Maybe 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 a common color space that all inputs are converted to before algorithms and processes are executed. The converters are useful for a number of markets, such as image processing and filtering. Their basic function is to convert from one color space to another. This application note describes one such conversion.

© 2005 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

## 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 that a color is correctly mapped 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 that computer displays use. 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. Each component has a range of 0 to 255, with all three 0s producing black and all three 255s producing white.

### 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).

As mentioned earlier, 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 that 60 to 70 percent of luminance or brightness is found 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.

The result is that 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. 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.

## Converting from Y'CrCb to R'G'B'

A color in the Y'CrCb color space is converted to the RGB color space using the following equations:

$$R' = 1.164(Y' - 16) + 1.596(Cr - 128)$$

$$G' = 1.164(Y' - 16) - (0.813)(Cr - 128) - 0.392(Cb - 128)$$

$$B' = 1.164(Y' - 16) + 2.017(Cb - 128)$$

Where R'G'B' are gamma-corrected RGB values and Y', Cr, and Cb are 8-bit inputs.

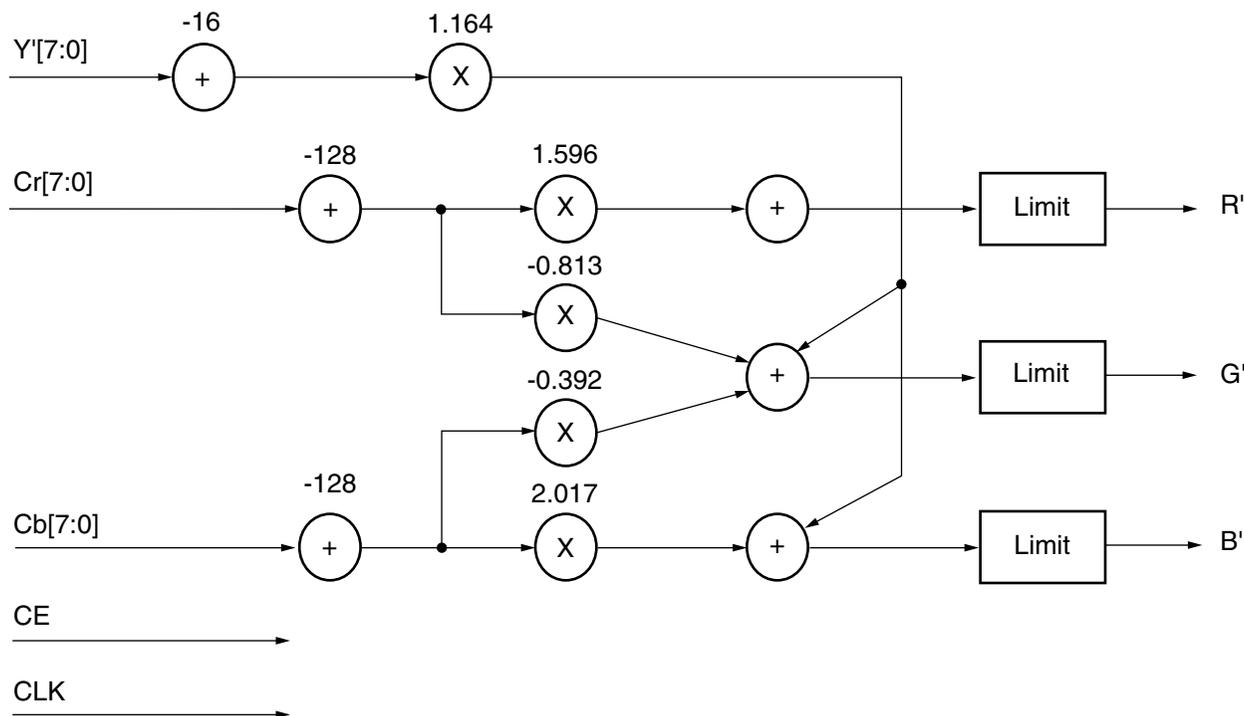
For 10-bit inputs, the equations are:

$$R' = 1.164(Y' - 64) + 1.596(Cr - 512)$$

$$G' = 1.164(Y' - 64) - (0.813)(Cr - 512) - 0.392(Cb - 512)$$

$$B' = 1.164(Y' - 64) + 2.017(Cb - 512)$$

Figure 1 shows a direct mapping of the above three equations. Notice that three of the seven terms are duplicates. This term is computed once and fed to the output adders for the Y', Cr, and Cb results.



x283\_01\_101701

Figure 1: Block Diagram Showing Math Elements

## Virtex-II Implementation Examples

The high density, on-chip memory in the Virtex-II designs increase overall system bandwidth by providing fast and resource-efficient FIFO buffers, shift registers, and CAMs. With embedded multipliers and improved arithmetic functions, Virtex-II solutions deliver over 600 billion MACs/s of Xtreme DSP performance.

There are up to 192 18 x 18 signed multipliers in a single device, supporting up to 36-bit signed multiplications. Cascading these multipliers supports even larger numbers. The multipliers can be combinatorial or pipelined, running between 140 MHz and 250 MHz depending on bit width. These features make Virtex-II devices the ideal choice for implementing the color space converter.

### Verilog Examples

As mentioned at the start of this application note, there are three different implementation examples. The following are the results of synthesizing and implementing each example.

Three different implementation examples are detailed in this application note. A fourth example is a CoreGen distributed arithmetic approach. The CoreGen approach is not implemented, but estimated results are given. The following sections show the results of synthesizing and implementing each example.

### Implementation Using Behavioral Verilog (gen\_model.\*)

In this implementation, the basic Y'CrCb2R'G'B' conversion equations are synthesized using Synplicity. All the signals are registered at the input and at the output. The synthesized EDIF

file is then placed and routed using Design Manager. A timing constraint of 10 ns was given to the place and route tool. The implementation results are listed in [Table 1](#).

**Note:** See Verilog file, `gen_model.v`.

### Design Summary

Table 1: Behavioral Implementation Design Summary

| Device                             | LUTs | FFs | Ports | Performance                                       |
|------------------------------------|------|-----|-------|---|
| XC2V500-5<br>(slowest speed grade) | 258  | 52  | 68    | 14 ns / 71 MHz<br>(inputs and outputs registered) |
| XC2V500-5<br>(slowest speed grade) | 260  | 85  | 68    | 9.4 ns / 106 MHz<br>(one intermediate pipe stage) |

### Implementation Using Block RAM as Look-Up ROM (`ram_model.*`)

$Y'$ ,  $C_b$ , and  $C_r$  are 10-bits wide and so have a range of 0 to 1023. This would give the following values for each of the terms in the  $R'$ ,  $G'$ , and  $B'$  equations:

$$1.164(Y - 64) = 1.164[(0 - 64) \text{ to } (1023 - 64)] = 1.164(-64 \text{ to } 959)$$

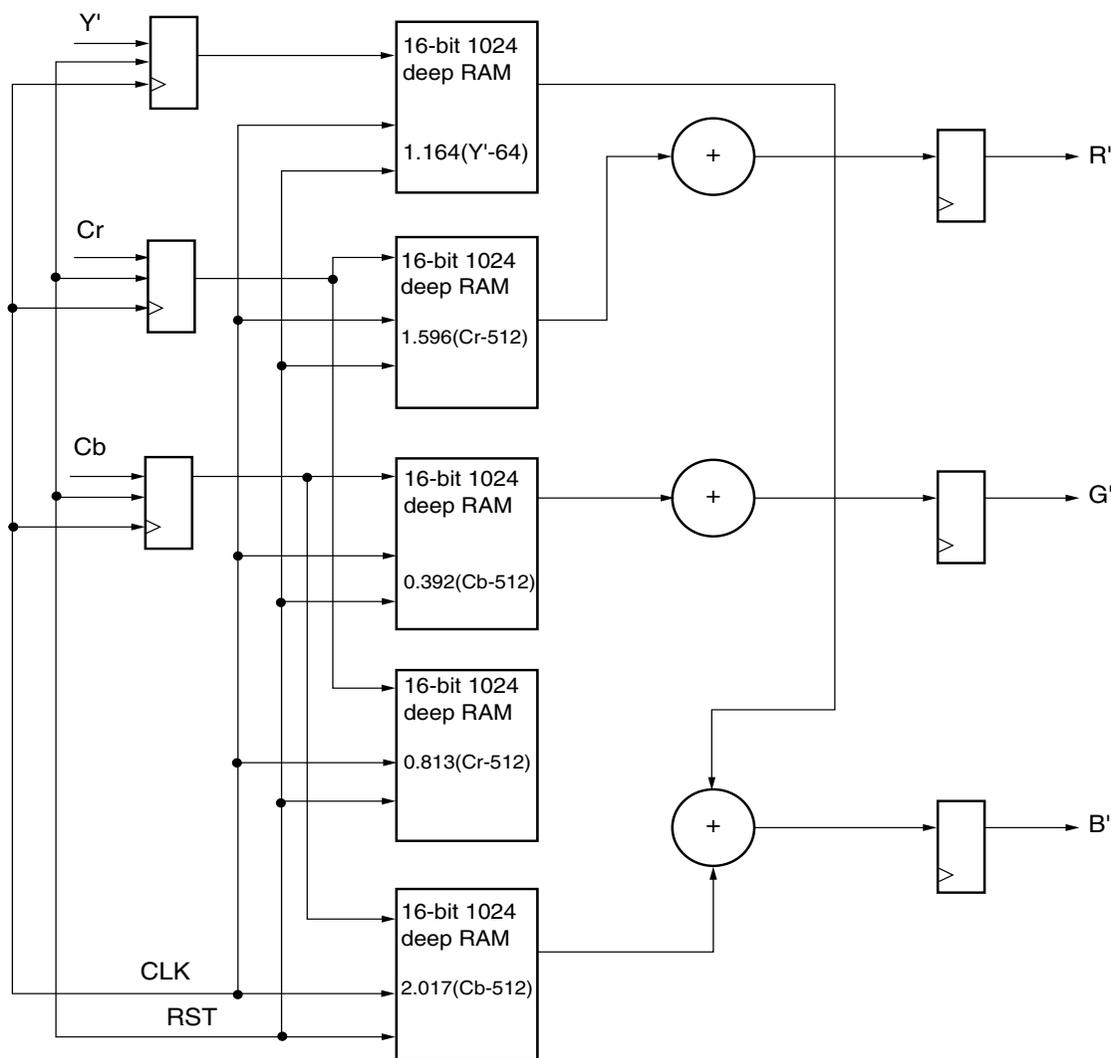
$$1.596(C_r - 512) = 1.596[(0 - 512) \text{ to } (1023 - 512)] = 1.596(-512 \text{ to } 511)$$

$$0.813(C_r - 512) = 0.813[(0 - 512) \text{ to } (1023 - 512)] = 0.813(-512 \text{ to } 511)$$

$$0.392(C_b - 512) = 0.392[(0 - 512) \text{ to } (1023 - 512)] = 0.392(-512 \text{ to } 511)$$

$$2.017(C_b - 512) = 2.017[(0 - 512) \text{ to } (1023 - 512)] = 2.017(-512 \text{ to } 511)$$

Each of these terms is calculated for all the possible input values. The results can then be stored in a 16-bit wide, 1024-deep RAM. Five RAMs are used for the five terms. The address lines to the RAMs are the respective input signals that are used in each of the terms. The output of the RAM is the data stored in the location addressed by the input signals,  $Y'$ ,  $C_r$ , and  $C_b$ . The output of the RAMs are added using an adder. The block diagram and the implementation results for this method are shown in [Figure 2](#).



x283\_02\_062503

Figure 2: Implementation Using RAM

### Implementation Results Using Embedded Multiplier in Virtex-II Device

The model with the instantiated block RAM was synthesized using Synplicity and the resulting EDIF file was placed and routed using Design Manager. A timing constraint of 5 ns was given to the place and route tool. The implementation results (push button) for the color space converter using the instantiated block RAM are in [Table 2](#).

**Note:** See Verilog file, `ram_model.v`.

Table 2: Block RAM Implementation Design Summary

| Device                             | LUTs | FFs | RAM | Ports | Performance                                       |
|------------------------------------|------|-----|-----|-------|---|
| XC2V500-5<br>(slowest speed grade) | 60   | 10  | 5   | 68    | 9 ns / 103 MHz<br>(inputs and outputs registered) |

### Implementation Using Embedded Multiplier (mult\_model.\*)

The block diagram for the implementation using embedded multiplier is as shown in [Figure 3](#). A two's complement circuit is provided to take care of the negative results for  $(Y'-16)$ ,

(Cr -128),and (Cb -128) values. The two's complement circuit can be omitted if the inputs are assumed to be in two's complement format.

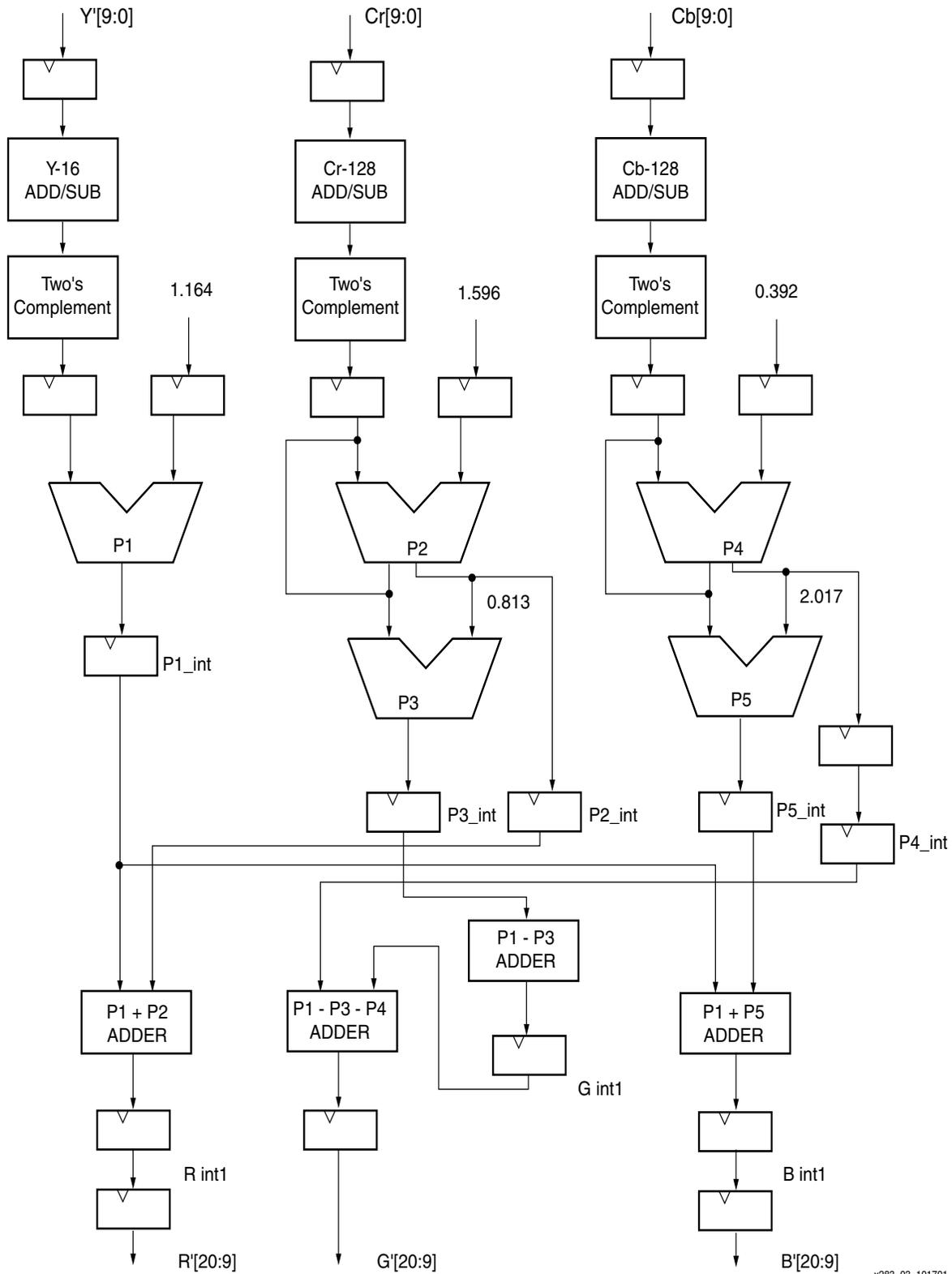


Figure 3: Implementation Using Instantiated Multiplier

x283\_03\_101701

### Implementation Results Using Embedded Multiplier in Virtex-II Device

The model with the instantiated multiplier was synthesized using Synplicity and the resulting EDIF file was placed and routed using Design Manager. A timing constraint of 5 ns was given to the place and route tool. The implementation result (push button) for the color space converter using the instantiated multiplier is shown in [Table 3](#).

**Note:** See Verilog file, `mult_model.v`.

### Design Summary

Table 3: Embedded Multiplier Implementation Design Summary

| Device                             | LUTs | FFs | Mult<br>18 x 18 | Ports | Performance      |
|------------------------------------|------|-----|-----------------|-------|------------------|
| XC2V500-5<br>(slowest speed grade) | 131  | 177 | 5               | 68    | 8.9 ns / 111 MHz |

### Reference Design

The VHDL and Verilog reference designs for this application note are available on the Xilinx web site in a .zip file:

<ftp://ftp.xilinx.com/pub/applications/xapp/xapp283.zip>

## Conclusion

The results of the synthesis and implementations demonstrate how the three examples trade off one math resource for another. The behavioral Verilog describing the conversion equations uses a resource available in Virtex, Virtex-E, and Virtex-II devices, known as "MULT\_AND" to form the basis of the multiplies in the equations. No block RAM or embedded multipliers are consumed. In the second example, the math resource used is block RAM/ROM, again available in all Virtex families. Finally, the Virtex-II family now provides the most flexible math resource for DSP in the form of an embedded, high-speed, two's complement multiplier.

## Revision History

The following table shows the revision history for this document.

| Date     | Version | Revision  |
|----------|---------|---|
| 07/11/01 | 1.0     | Initial Xilinx release  |
| 04/22/02 | 1.1     | Updated <a href="#">Figure 1</a> and <a href="#">Figure 2</a> . Changed implementation summaries with newer data. Updated to include Virtex-II Pro devices. Modified the 10-bit equation on page 2. |
| 06/26/02 | 1.2     | Updated revision date.  |
| 07/03/03 | 1.3     | Changed " <a href="#">Implementation Using Block RAM as Look-Up ROM (ram_model.*)</a> " section and <a href="#">Figure 2</a> .  |
| 03/24/05 | 1.3.1   | Title change only.  |