XAPP932 (1.0.1) July 28, 2010

# Chroma Resampler

Author: Clive Walker


## Summary

This application note describes the implementation of six circuits necessary to perform commonly used conversions between various chroma formats. It is accompanied by reference designs which include Generic RTL VHDL code. The code is written in such a way that it may be targeted at any Xilinx device family. System Generator tokens encapsulating the HDL code are included for System Generator users. System Generator test benches are also provided to visually inspect output results and for verification against known models in the MATLAB®/Simulink® software environment.


## Introduction

The native capture format RGB provides R, G, and B information in equal bandwidth. This information can be stored and transmitted in this form, although it is not the most efficient format. It is accepted that the human eye is not as receptive to chrominance (color) detail as luminance (brightness) detail. Using color-space conversion, it is possible to convert RGB into the YCbCr color space, where Y is Luminance information, and Cr and Cb are derived color-difference signals. Further, full bandwidth Y can be maintained, but Cr and Cb are subsampled. This provides a simple, but very effective first stage of video compression to ease storage and transmission costs. There are several subsampled chroma formats described in "Subsampled Graphical Formats and Supported Delivery Formats." This application note concentrates on the conversion between the most widely used of these chroma formats.

To find out more about the conversion between RGB and YCrCb, refer to the System Generator Color-Space Converter block description. Also see Xilinx application notes XAPP930, Color-Space Converter: RGB to YCrCb and XAPP931, Color-Space Converter: YCrCb to RGB.

### Notation

The notation shown in Figure 1 is used in Figure 2, Figure 4, and Figure 6:

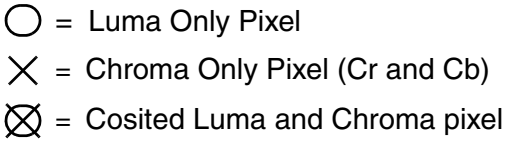


*Figure 1:* **Notation**

## Common Interface Signals

To describe how the blocks accept input data and deliver output data, it is necessary to define some of the interfacing signals.

Table 1 presents the signals found in the CRS blocks.

*Table 1:* **Interface Signals**

| Signal Name | Description |
|---|---|
| luma_in | Brought through the block for equalization of delay between luma and chroma channels. |
| cr_in/out | Cr color difference signal. Only used for 4:4:4 interfaces. The format is described in "4:4:4." |
| cb_in/out] | Cb color difference signal. Only used for 4:4:4 interfaces. The format is described in "4:4:4." |
| chroma_in/out | CrCb interleaved at full bandwidth. Used for 4:2:2 or 4:2:0 interfaces. By convention, Cr is always first. The format is described in "4:2:2" and "4:2:0 (MPEG2)." |
| luma_out | Delayed version of luma_din. Use is optional. |
| vs_in | Vertical sync input. Should be High during the vertical blanking period. The rising edge of vs_in is used internally as a field-based reset. Must be at least 1 H-period in duration to generate vs_out. |
| hs_in | Horizontal sync input. High during the active period of a horizontal line, for example, for 1920 cycles in 1080i. |
| din_valid | This signal should be exactly the same as hs_in on the lines to be filtered. However, to remove image edge-filtering artifacts, din_valid can be extended by the user if input data is mirrored or repeated at image edges. |
| hs_out | H sync signal. Delayed horizontally by a number of clock cycles equal to the latency of the filter. |
| vs_out | V sync signal. Delayed vertically and horizontally by a number of h-periods and clock cycles equal to the latency of the filter. |
| dout_valid | Delayed version of din_valid. Indicates valid data at luma_out and chroma_out/cr_out/cb_out. |
| saturate/sat_type | A High value on saturate indicates that the operation has overflowed or underflowed in completing the addition chain following filtering. The saturation type is indicated by the **sat_type** output.<br>When Saturate = 1:<br>sat_type = 0 indicates overflow (full_precision_result > $2^{(full\_precision\_result\_width)} - 1$)<br>sat_type = 1 indicates underflow (full_precision_result < 0)<br>This functionality allows the user to implement a subsequent clip/clamp structure, if desired. |
| chroma_dout_valid | 4:2:0-specific dout_valid flag. Use is optional. Only alternate line outputs are validated. |

## Common Features

- The blocks can be targeted at any device in any family.
- All blocks provide a luma delay path equal to the delay implied by the chroma filter path. Not connecting the luma output should result in minimization of the rest of the block to a small number of slices and no multiplier blocks.
- All blocks assume unsigned luma and chroma input data.
- All blocks that require vertical filtering provide a delay line for `h_sync` and `v_sync` in addition to the `dout_valid` signal output.
- All blocks require an input h_sync and v_sync.

## Subsampled Graphical Formats and Supported Delivery Formats

### 4:4:4

This format originates from RGB, However, for the purposes of these conversion blocks, it is used to describe YCbCr at the same sample-rate as the original RGB. Indeed, RGB is also in 4:4:4 format. It is used for image capture and display purposes. Cr and Cb channels are sampled at the same rate as the full bandwidth luminance. Hence, all pixel locations have luma and chroma data cosited. See Figure 2.

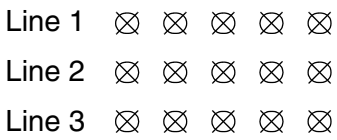


*Figure 2:* **4:4:4 Graphical Representation**

Delivery of this format is simple. A separate bus is required for each of Y, Cr, and Cb. When this is the input format, a `din_valid` signal should accompany the data. The `dout_valid` signal will be the same as the `din_valid` signal, but might be delayed by a number of clock cycles, and a number of h-periods, depending upon the nature of the conversion and the number of filter taps in both the horizontal and vertical filters See Figure 3.

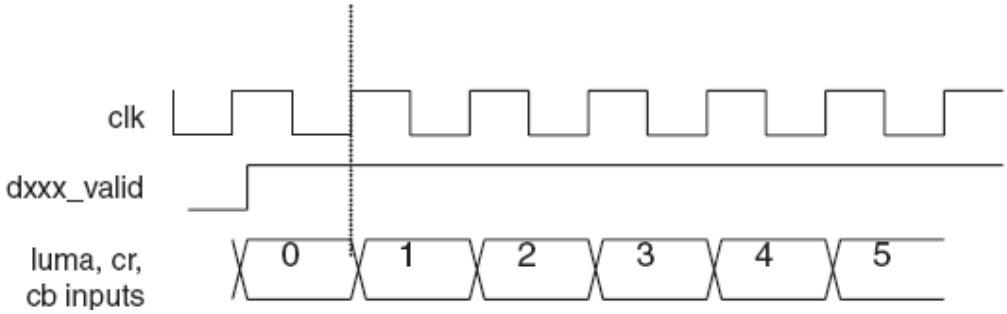


*Figure 3:* **4:4:4 Delivery Format**

### 4:2:2

This format contains horizontally subsampled chroma. For every two luma samples, there is an associated pair of Cr and Cb samples. The subsampled chroma locations are cosited with alternate luma samples. See Figure 4.

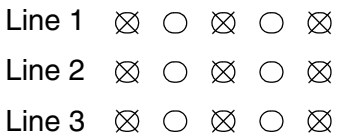


*Figure 4:* **4:2:2 Graphical Representation**

Delivery of this format involves interleaving Cr and Cb on a single bus, and running this bus at full sample rate, ***putting Cr first***. See Figure 5.

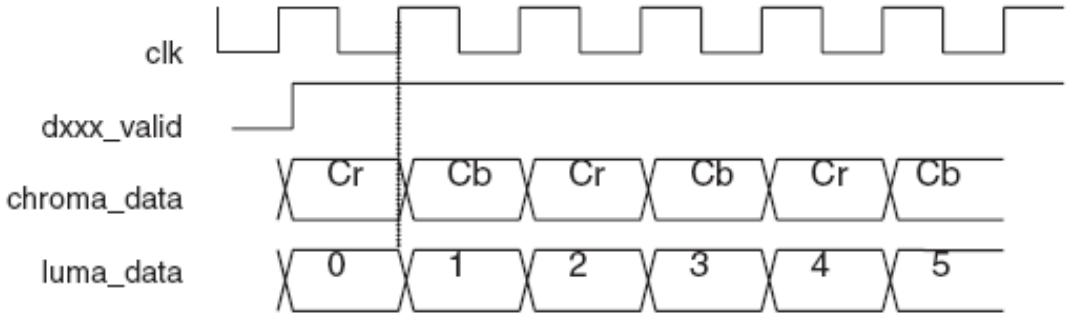


*Figure 5:* **4:2:2 Delivery Format**

### 4:2:0 (MPEG2)

The version of 4:2:0 that is used chiefly for MPEG2 encoding contains horizontally and vertically subsampled chroma. An additional feature with this format is that the chroma pixels are not cosited with the luma pixels. In fact, in creating them, vertical filtering is used, and their effective *location* puts them directly between alternate pairs of lines. Their value is interpolated. Their horizontal location is cosited with alternate luma samples, as indicated in Figure 6.

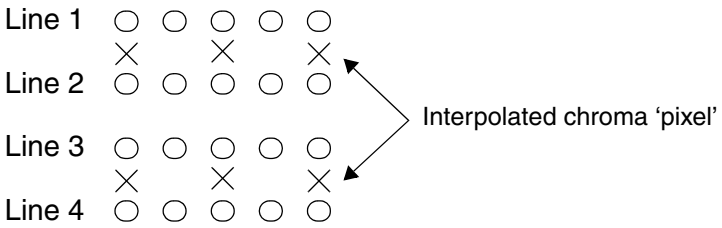


*Figure 6:* **4:2:0 Graphical Representation**

Delivery of this format is similar to 4:2:2 as shown in Figure 4, except that only the alternate lines are valid (see Figure 6). This is true for inputs and outputs. `dout_valid` is given as an output validation signal. Where luma lines are given in the order 0, 1, 2, 3, 4, 5…. (0 being the top line), valid output chroma lines will be given on lines 0, 2, 4, 6, …. Cr and Cb samples are interleaved as per 4:2:2. This is always the case, but a chroma validation signal is also given as an output (see Figure 7) although its use is optional.

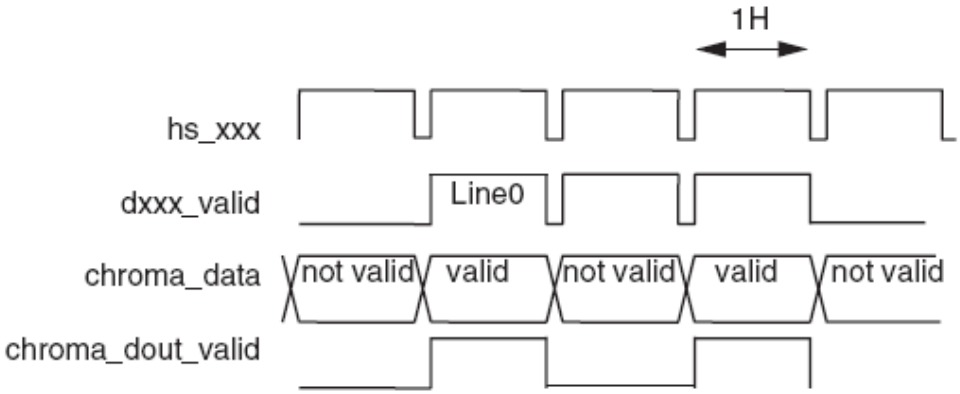


*Figure 7:* **4:2:0 Delivery Format**

Similarly, at a 4:2:0 input, the data present on the chroma input when `din_valid` is high for the first time (since `vs_in` is deactivated), is taken as line 0 of chroma input. However, the second vertical sample of a 4:2:0 input is not taken until the third luma line is valid on the input.

## Converter Blocks

Currently, there are six conversions available. Conversion in these blocks is intended to be achieved using a FIR filter approach. Some require filtering in only the H dimension, some only in the V dimension, and some in both H and V dimensions. These are detailed in Table 2 along with default filter information.

*Table 2:* **Filter Summary**

| Converter | Filter Orientation | Default FIR Size | Notes |
|---|---|---|---|
| convert444to422 | H down conversion | 3 H Taps | |
| convert444to420 | H & V down conversion | 2 V Taps x 3 H Taps | |
| convert422to444 | H up conversion | 2 H Taps | 2-phases – one phase just replicates input |
| convert422to420 | V down conversion | 4 V Taps | |
| convert420to444 | H and V up conversion | 2 H Taps x 2 V Taps | 2-phase structure |
| convert420to422 | V up conversion | 2 H Taps x 2 V Taps | 2-phases – phase 1 is inverse of phase 2 |

## Parameters

The parameters vary between blocks. All blocks have been coded in RTL VHDL and are generically parameterized. This allows the user to provide parameters of his choice according to his own criteria. Some parameters are calculated automatically by MATLAB software when using System Generator as the delivery mechanism. Others need to be specified directly.

### Replicate

This option is available in all up converters:

- convert422to444
- convert420to444
- convert420to422

It applies in both vertical and horizontal domains as appropriate. Using the replicate option results in up conversion with no filter. Replication of the previous input sample occurs instead. Consequently, synthesis runs of this block minimize the filter hardware from the system.

### Pixel_drop

This option applies to the down converter:

- convert444to422 (H filtering only)

Using the `pixel_drop` option results in down conversion with no filter. Some samples are passed directly to the output, but others are dropped entirely as appropriate. This occurs on a pixel-by- pixel basis only. Consequently, synthesis runs of this block minimize the filter hardware from the system.

### Line_drop

This option applies to the down converter:

- convert422to420 (V filtering only)

Using the `line_drop` option results in down conversion with no filter. Some samples are passed directly to the output, but others are dropped entirely as appropriate. This occurs on a line-by-line basis only. Consequently, synthesis runs of this block minimize the filter hardware from the system.

### Sample_drop

This option applies to the down converter:

- convert444to420 (H and V filtering)

Using the `sample_drop` option results in down conversion with no filter. Some samples are passed directly to the output, but others are dropped entirely as appropriate. This occurs on a line-by-line basis and on a pixel-by-pixel basis. Consequently, synthesis runs of this block minimize the filter hardware from the system.

### Coefficients

Coefficients should be provided to all blocks as **positive or negative integer** parameters. Defaults have been set that are appropriate as deemed by Xilinx.

### Num_h_taps/Num_v_taps

For all blocks, it is necessary to configure the size of the filter. For System Generator, these parameters are calculated automatically. MATLAB software counts the number of coefficients entered in the coefficients parameter.

### Data_width

This parameter is the input and output data width for chroma and luma streams.

***Note:*** The output data bit width does not exhibit any growth that occurs during filtering. The outputs have been rounded by adding half an output LSB in the full precision domain prior to truncation.

For System Generator, the input data width is automatically drawn from the bit width entered in the Xilinx Input Gateway.

### Coefficient Width

Xilinx recommends that the coefficients should **sum to exactly $2^{(coef\_width - 1)}$** to achieve unity gain. If they sum to **less than $2^{(coef\_width - 1)}$**, then some loss of dynamic range is observed. Also, no single coefficient must exceed $2^{(coef\_width - 1)} - 1$.

### Samples per Active Input Line

This option is available in all converters that require vertical filtering:

- convert420to444
- convert420to422
- convert422to420
- convert444to420

It is required to configure the size of the line-buffers at implementation time.

### Padding/Border Value

This is the value that is fed into the FIR structure before and after valid data is indicated by `din_valid`. **These designs do not repeat or mirror input samples at image edges.** Hence, the padding value is taken as the value that is pumped into the filters before and after valid data at image edges. Xilinx recommends that the user repeat, as needed, the input samples externally to this core, extending `din_valid` accordingly. Without doing this, the fixed padding value introduces artifacts at the image edges. This is true vertically and horizontally.

## Block Specifics

### Convert422to444 Block

This block (Figure 8) is a 2:1 horizontal up conversion operation.

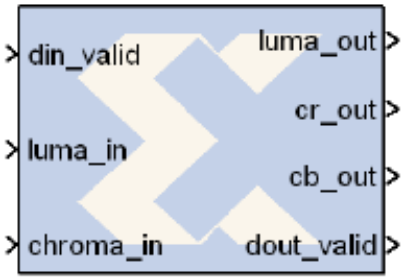


*Figure 8:* **Convert422to444 Block**

While this would most correctly be achieved using a 2-phase polyphase structure, it is noted that one of the two output pixels is cosited with one of the input samples, and the most ideal output is achieved simply by replicating this input sample. Advantage has been taken of this to reduce hardware usage, which equates to the coefficient for that phase being equal to 1 for one tap and zero for the others. For the second phase, the default coefficients are equivalent to **[0.5 0.5]**. These coefficients are represented in 2-bit form such that the integer representations are **[1 1]**.

Coefficients for the second phase should be provided by the user. They are supplied as a generic to this block, thus:

num_taps : integer: = 4;

coefs : INTEGER_ARRAY: = (a, b, c, d, ...)

**a** is the coefficient applied to the **right-most input** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 9. If more than `num_taps` are supplied, others will be ignored.

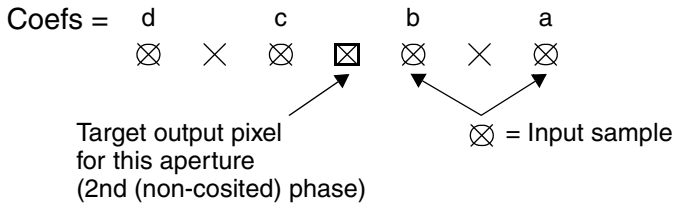


*Figure 9:* **4:2:2 to 4:4:4 Coefficient Configuration**

The latency through this default filter is eight clock cycles. For non-default filters, the latency can be calculated according to the formula:

Latency = (2*num_taps) + 4

When using the **replicate** option, the latency is equal to eight clock cycles.

### Convert444to422 Block

This block (Figure 10) is a 2:1 horizontal down conversion operation. By default, this is achieved using a three-tap structure with coefficients equivalent to **[0.25 0.5 0.25]** represented in 3-bit form by the integer set **[1 2 1]**.

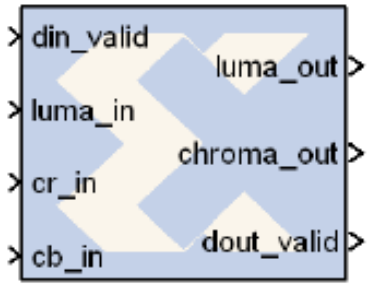


*Figure 10:* **Convert444to422 Block**

The coefficients should be provided by the user. They are supplied as a generic to this block, thus:

```
num_taps    : integer := 5;
coefs       : INTEGER_ARRAY := (a, b, c, d, e, ...)
```

**a** is the coefficient applied to the **right-most input** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 11. If more than `num_taps` taps are supplied, others will be ignored.

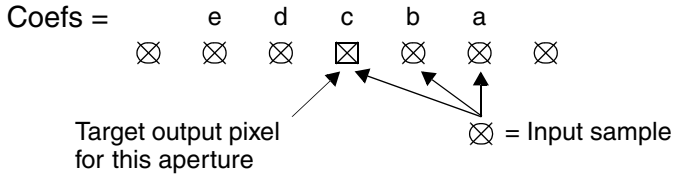


*Figure 11:* **4:4:4 to 4:2:2 Coefficient Configuration**

The latency through this default filter is eight clock cycles. For non-default filters, the latency can be calculated according to the formula:

Latency = (num_taps + 5)

When using the `pixel_drop` option, the latency is equal to four clock cycles.

### Convert420to422 Block

This block (Figure 12) is a 2:1 vertical up conversion operation. By default, this is achieved using a 2-tap 2-phase structure with coefficients equivalent to **[0.25 0.75]** represented in 3-bit form by the integer set **[1 3]**. The second phase is achieved by reversing these coefficients.

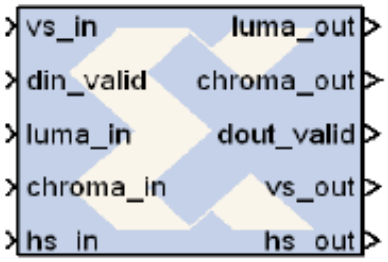


*Figure 12:* **Convert420to422 Block**

For this 2-phase operation, the user need only supply the coefficients for one phase. They are supplied as a generic to this block, thus:

```
num_v_taps    : integer := 4;
coefs         : INTEGER_ARRAY := (a, b, c, d, ...)
```

For the first phase (Phase 0), **a** is the coefficient applied to the **lowest input** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 13. If more than `num_v_taps` taps are supplied, others will be ignored.

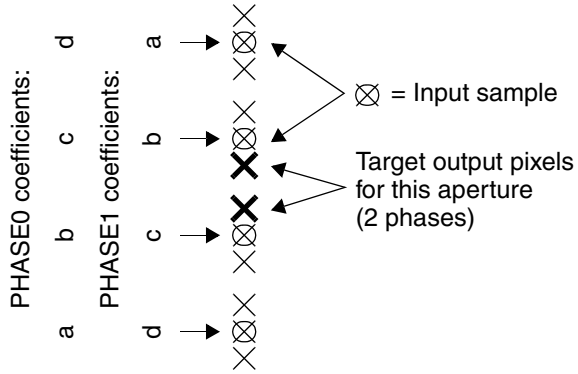


*Figure 13:* **4:4:4 to 4:2:2 Coefficient Configuration**

The latency through this default filter is 1 line + 9 clock cycles. For non-default filters, the latency can be calculated according to the formulae:

Vertical_Latency = num_v_taps – 1

Horizontal_Latency = 9 cycles (constant)

### Convert422to420 Block

This block (Figure 14) is a 2:1 vertical down conversion operation. By default, this is achieved using a 4-tap structure with coefficients equivalent to **[0.125 0.375 0.375 0.125]** represented in 4-bit form by the integer set **[1 3 3 1]**.

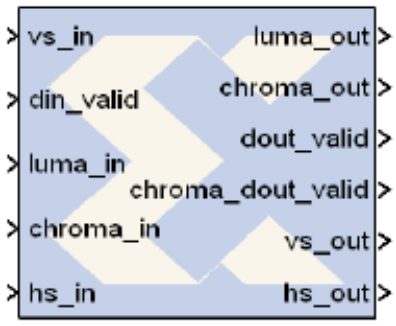


*Figure 14:* **Convert422to420 Block**

The coefficients are supplied as a generic to this block, thus:

```
num_v_taps      : integer := 4;
coefs           : INTEGER_ARRAY := (a, b, c, d, ...)
```

**a** is the coefficient applied to the **lowest input** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 15. If more than `num_v_taps` taps are supplied, others will be ignored.

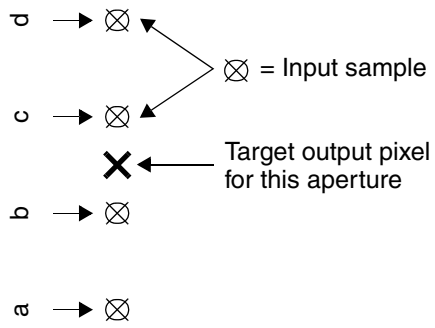


*Figure 15:* **4:2:2 to 4:2:0 Coefficient Configuration**

The latency through this default filter is 2 lines + 8 clock cycles. For non-default filters, the latency can be calculated according to the formulae:

Vertical_Latency = num_v_taps/2

Horizontal_Latency = num_v_taps/2 + 6

### Convert420to444 Block

This block (Figure 16) is a 2D operation. This is achieved using a 4-tap structure with two horizontal and two vertical taps in the default configuration, using two H-phases and two V-phases.

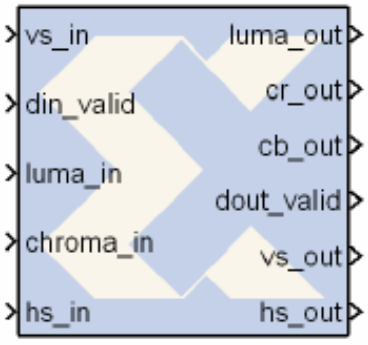


*Figure 16:* **Convert420to444 Block**

1. First H-phase: Two output pixels are horizontally aligned with two input samples, so the horizontal contribution from the non-aligned samples is zero. The contributions from the other two samples are the same as in the 420to422 conversion (**[0.25 0.75]** reversed for opposite vertical phase).
2. Second H-phase: Two output pixels are located at horizontally equivalent distances from two successive input pixels, but maintain the same vertical offset. Hence, contributions are required from all four input samples. In the first vertical phase, the notation could be

   [0.375 0.375]

   [0.125 0.125]

But this would flip vertically for the second vertical phase.

The coefficients are supplied in two H-phases as generics to this block, thus:

```
num_v_taps      : integer := 4;
num_h_taps      : integer := 4;
coefs_hphase0 : INTEGER_ARRAY := (a, b, c, d, ...);
coefs_hphase1 : INTEGER_ARRAY := (e, f, g, h, ...)
```

Each generic needs to be `num_h_taps` * `num_v_taps` in length. The first `num_h_taps` coefficients for each H-phase generic are used for the lower-most line of input samples in V-phase 0, and for the upper-most line in V-phase 1.

For V-phase 0, **a** is the coefficient applied for H-phase 0 to the **lowest** and **right-most** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 17. This order is flipped vertically for V-phase 1.

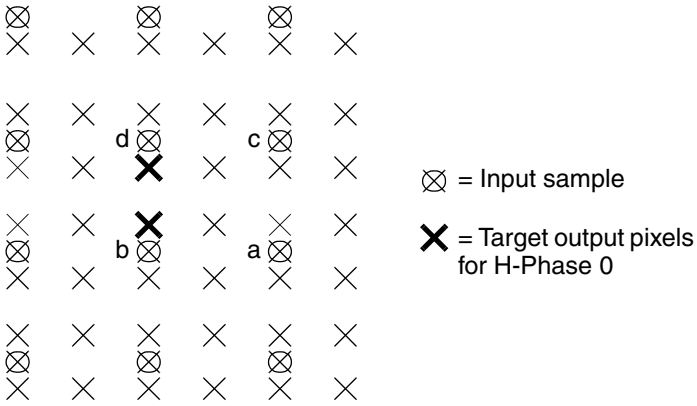


*Figure 17:* **4:2:0 to 4:4:4 Coefficient Configuration (H-phase 0)**

For V-phase 0, **e** is the coefficient applied for H-phase 1 to the **lowest** and **right-most** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 18. This order is flipped vertically for V-phase 1.

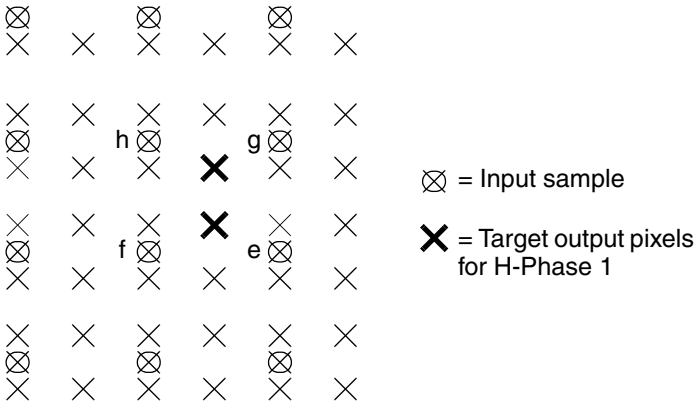


*Figure 18:* **44:4:4 to 4:2:0 Coefficient Configuration (H-phase 1)**

The latency through this default filter is 1 line + 14 clock cycles. For non-default filters, the latency may be calculated according to the formulae:

Vertical_Latency = num_v_taps/2

Horizontal_Latency = 2*num_h_taps + 10

### Convert444to420 Block

This block (Figure 19) is a 2D operation, down converting both vertically and horizontally by a factor of 2. By default, this is achieved using a 6-tap structure with three horizontal and two vertical taps. This filter is ideally symmetrical vertically and horizontally, and this is reflected in the default coefficients which are equivalent to **[0.0625 0.375 0.0625]**, represented in 5-bit form by the integer set **[1 6 1]**, repeated for two vertical locations. The user can, however, still provide his own filter coefficients in both dimensions.

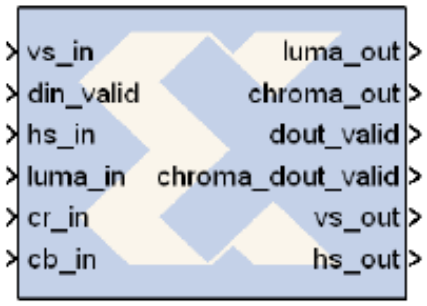


*Figure 19:* **Convert444to420 Block**

The coefficients are supplied as a generic to this block, thus:

```
num_v_taps      : integer := 3;
num_h_taps      : integer := 2;
coefs           : INTEGER_ARRAY := (a, b, c, d, e, f, ...);
```

**a** is the coefficient applied to the **lowest input** sample in the filter aperture (that is, the newest in raster-scan order) as shown in Figure 20.

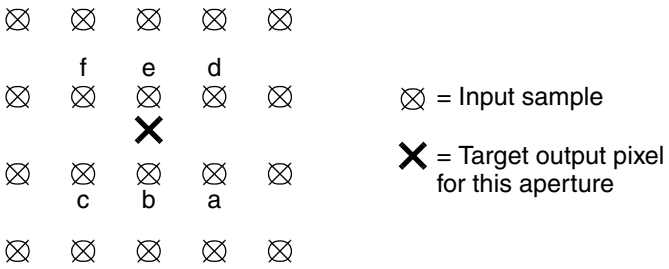


*Figure 20:* **4:4:4 to 4:2:0 Coefficient Configuration**

The latency through this default filter is 1 line + 11 clock cycles. For non-default filters, the latency may be calculated according to the formulae:

```
Vertical_Latency = num_v_taps/2
Horizontal_Latency = num_h_taps + num_v_taps + 4
```

## System Generator Test Bench

To help prototyping, testing, and verification of the RGB to YCrCb subsystem, a System Generator test bench is included with the reference design. Test bench files are under the `chroma_resampler/testbench` directory.

To run tests on these files, it is necessary to have the following software installed:

- MATLAB R14 SP3
- ModelSim SE 6.1a
- System Generator 8.1

There are six test benches—one for each of the six conversions. Each test bench has an associated MATLAB software model (.mdl) file named convertxxxtoyyy_model.mdl. It is necessary to follow this procedure when running the tests and experimenting with them:

1. Change directory (CD) in the MATLAB software environment to the /chroma_resampler/testbench directory.
2. Run pre_proc.m (type pre_proc <ENTER>) at a MATLAB software prompt. This does the following:
   a. Runs reference.mdl which takes an image from a source file (.png file).
   b. Converts the source data from RGB format into YCbCr at 4:4:4.
   c. Uses Simulink software models to convert from 4:4:4 to 4:2:2 and 4:2:0 (MPEG2).
   d. Uses Simulink software models to convert from 4:2:0 to 4:2:2 (MPEG2) and 4:4:4.
   e. Uses Simulink software models to convert from 4:2:2 (MPEG2) to 4:2:0 and 4:4:4.
   f. Creates 1D vector data from the 2D stimuli data for all conversions.
   g. Sets up the default filter configuration for all conversions.

   In this way, stimulus and golden vector data are generated for test and verification purposes for all conversions.
3. Run the appropriate convertxxxtoyyy_model.mdl by pressing the play (▶) button in Simulink software. ModelSim should be spawned.

   This creates some 1D vector files as outputs from the models, for example, luma_out_444to422.mat.
4. Run the appropriate post-processing file. (Type post_proc_xxxtoyyy). This causes the following:
   a. The previous .mat files are read. The 1D data inside them is converted into 2D form.
   b. The 2D data is compared with the golden vectors previously created. The difference is displayed as a delta value for Cr and Cb only.

      Note that this value is usually non-zero because Simulink software models that are used as reference conversions do not use the same kind of rounding as that used in the hardware in this reference design. Also, for some conversions, coefficients can differ. For the 4:4:4 to 4:2:2 conversion, the 4:4:4 to 4:2:0 conversion, and the 4:2:2 to 4:2:0 conversion, coefficients in the reference models can be changed by altering the Simulink software parameters for the appropriate conversion in the reference.mdl file, and re-running pre_proc.

      When running a conversion that creates 4:4:4, view the results by opening and running the show444.mdl file.

      When running a conversion that creates 4:2:2, view the results by opening and running the reconstruct_422to444.mdl file.

      When running a conversion that creates 4:2:0, view the results by opening and running the reconstruct_420to444.mdl file.

      It should also be noted that regression testing can also be performed by running the appropriate rtest_xxxtoyyy.m file from the MATLAB software prompt. This runs the previously described routine, setting different coefficients and conditions. It also creates a copy of the required output files for each test. Inspect the rtest_xxxtoyyy.m file for information about which files are created in each test.

## Reference Design Files

The following reference design source files are available for download from the Xilinx Web site at: www.xilinx.com/bvdocs/appnotes/xapp932.zip.

The top-level design files are `convertxxxtoyyy.vhd`. Compilation is dependent on some common design files in the `/XLIB directory: imagexlib_utils.vhd and imagexlib_arch.vhd` which must be compiled locally into the `work` directory.

For all six designs, a simple FIR filter approach is used. Coefficients are user-programmable, that is, any combination can be specified as a parameter within the bounds of the definition of each conversion. All filters are always non-symmetric to maintain user flexibility. The code is parameterizable for any input bit width—the output bit width is always equal to the input bit width.

Internally, full precision is used until the final stage where in the FIR structure. For all designs, the user can select an alternative option, which either replicates samples or drops them, rather than using any FIR filter at all if the user's application permits. This generic option reduces hardware resource usage at the cost of output video quality.

The fully scalable designs are sensitive to the parameters used, that is, resource usage scales accordingly.

## Resource Utilization

The figures in Table 3 were observed when using ISE® 8.1 and Synplify Pro Version 8.1. They should only be used as guideline figures and are subject to change with any architecture/ design changes.

***Note:*** This table was generated using all the default parameter settings.

*Table 3:* **Performance and Resource Utilization Profile (approximate) for Chroma Resampling Blocks**

| Target Device | XC4VFX12-10 | | | | 3S200-4 | | | |
|---|---|---|---|---|---|---|---|---|
| **Resources** | **Slices** | **Block RAMs** | **DSP48** | **Fmax (MHz)** | **Slices** | **Block RAMs** | **Mults** | **Fmax (MHz)** |
| **Synplify Pro Synthesis Tool** | | | | | | | | |
| Convert422to444(1) | 49 | 0 | 0 | 400 | 49 | 0 | 0 | 200 |
| Convert444to422(1) | 58 | 0 | 0 | 400 | 58 | 0 | 0 | 200 |
| Convert420to422(2) | 224 | 3 | 0 | 200 | 151 | 3 | 0 | 150 |
| Convert422to420(2) | 288 | 5 | 0 | 200 | 180 | 5 | 0 | 125 |
| Convert420to444(2) | 308 | 3 | 8 | 200 | 300 | 3 | 4 | 125 |
| Convert444to420(2) | 277 | 3 | 0 | 200 | 220 | 3 | 0 | 125 |
| **XST Synthesis Tool** | | | | | | | | |
| Convert422to444(1)) | 57 | 0 | 0 | 300 | 57 | 0 | 0 | 200 |
| Convert444to422(1) | 60 | 0 | 0 | 200 | 60 | 0 | 0 | 150 |
| Convert420to422(2) | 183 | 3 | 0 | 200 | 166 | 3 | 0 | 150 |
| Convert422to420(2) | 218 | 5 | 0 | 200 | 201 | 5 | 0 | 125 |
| Convert420to444(2) | 233 | 3 | 8 | 150 | 304 | 3 | 8 | 125 |
| Convert444to420(2) | 252 | 3 | 0 | 250 | 232 | 3 | 0 | 125 |

***Notes:***

1. Data Width = 8.
2. Data Width = 8; Line Length = 1920.

## Revision History

The following table shows the revision history for this document.

| Date | Version | Description of Revisions |
|---|---|---|
| 05/09/06 | 1.0 | Initial Xilinx release. |
| 07/28/10 | 1.0.1 | Correction to Figure 1; minor text edits. |

## Notice of Disclaimer