

## AURA LABS CONSULTING COMMUNITY

---

### RGB to YUV (Y/Cr/Cb) color space converter (design example)

Many image processing applications require separation of luminance (brightness) and chrominance (color) components of image data. For instance, such basic image manipulations as brightness, contrast, saturation, hue, and gamma might be easily performed in YUV(Y/Cr/Cb) domain, but it is more difficult in RGB color space.

Complex video processing algorithms is also involving various manipulations and analysis of luminance and chrominance components separately. The reasons are for example:

- in **video compression** the luma and chroma components separation is necessary since their different MTF (Modulation Transfer Function). As a result, chroma components require less data to be kept after compression than luma components; i.e. chroma components might be compressed strongly with no salient distortions addition.
- in **video improvement** systems it's needed since most sharpness and noise reduction algorithms, that is based on temporal (frame-to-frame) and spatial (into frame) adaptive filtration technique, are strongly different in respect of luma and chroma image components. Applying such video improvement algorithms to RGB video data brings about low effectiveness and visible color hue distortion.

Unfortunately, in many cases a developer is getting RGB video components to his video processor mainly from digitized analog RGB, DVI or HDMI interfaces, CCD or CMOS cameras. Thus, he often requires starting development from RGB to YUV color space conversion stage.

The presented RGB to YUV color space converter was successfully used in many different projects, such as: real-time HDTV video processing module for PDP/LCD display controller ASIC, endoscope video improvement processor, MPEG2 compression pre-processing system, etc.

## **Main features:**

- *Input / Output precision: 8 bit/channel;*
- *multiplication factors precision:8 bit*
- *Video format: 1080i/30, 1080i/25, 720p/60, 720p/50, 576p, 480p, 576i, 480i*
- *Video data stream clock: 27MHz...74.25MHz (depending on video format)*
- *Destination platform: Xilinx FPGA Virtex II, Spartan 3, Altera Cyclone, etc.*

## **Description**

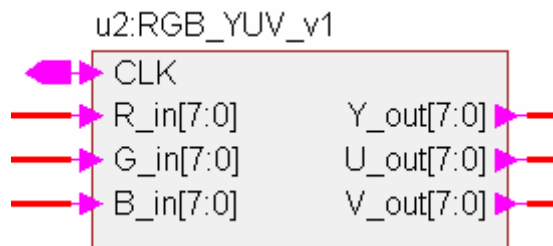


Fig.1, Top module

(see module source code in APPENDIX A)

### **Ports description (Fig.1):**

- CLK** – up to 74.25 MHz system clock input;
- R\_in[7:0]** – red pixel data input;
- G\_in[7:0]** – green pixel data input;
- B\_in[7:0]** – blue pixel data input;
- Y\_out[7:0]** – luma pixel data output;
- U\_out[7:0]** – U (formally Cr) pixel data output;
- V\_out[7:0]** – V (formally Cb) pixel data output;

**APPENDIX A**

```

module RGB_YUV_v1 (CLK, R_in, G_in, B_in, Y_out, U_out, V_out);

input [7:0] B_in;
input      CLK;
input [7:0] G_in;
input [7:0] R_in;

output [7:0] U_out;
output [7:0] V_out;
output [7:0] Y_out;

reg [7:0] Y_out ;
reg [8:0] y, u, v, u_out, v_out;
reg [7:0] R, G, B;
reg [6:0] k1R, k1B;
reg [7:0] k1G, k2B, k2R, k2G, k3R, k3G, k3B;
reg [7:0] aaa;
reg [8:0] bbb, ccc, ddd, eee;

assign U_out = u_out[7:0];
assign V_out = v_out[7:0];

always @(posedge CLK)
begin
    R = R_in;
    B = B_in;
    G = G_in;
end

always @(posedge CLK)
begin
    k1R[6:0] = (R * 7'd77) / 256;
    k1B[6:0] = (B * 7'd29) / 256;
    k1G[7:0] = (G * 8'd150) / 256;
    k2B[7:0] = (B * 8'd131) / 256;
    k2R[7:0] = (R * 8'd44) / 256;
    k2G[7:0] = (G * 8'd87) / 256;
    k3R[7:0] = (R * 8'd131) / 256;
    k3G[7:0] = (G * 8'd110) / 256;
    k3B[7:0] = (B * 8'd21) / 256;
end

always @(posedge CLK)
begin
    aaa[7:0] = k1R + k1B;
    bbb[8:0] = 255 + k2B;
    ccc[8:0] = k2R + k2G;
    ddd[8:0] = 255 + k3R;
    eee[8:0] = k3G + k3B;
end

always @(posedge CLK)
begin
    y[8:0] = aaa + k1G;
    u[8:0] = bbb - ccc;
end

```

```
        v[8:0] = ddd - eee;
    end

always @(posedge CLK)
    begin
        if (y > 9'd255) Y_out = 8'd255;
        else Y_out = y[7:0];

        if (u > 9'd383) u_out = 9'd255;
        else if (u > 9'd128) u_out = u - 9'd128;
        else u_out = 9'd0;

        if (v > 9'd383) v_out = 9'd255;
        else if (v > 9'd128) v_out = v - 9'd128;
        else v_out = 9'd0;
    end

endmodule //RGB_YUV_v1(rtl)
```