Video Color Processing

Draft version: 1.9

Sakinder Ali

Contents

INTRODUCTION

Video color processing based upon implementation of raspberry pi camera link running on a Kria KV260 board. This design takes raw Bayer format data and convert into video 16-bit YCbCr [4:2:2] (YUV) color space. This design reads camera input data, then transform into rgb color space of 24-bit data bus with 8 bits each for the red pixel, green pixel, and blue pixel. Each rgb pixel either filtered or converted into color space. Sharp, blur, emboss and sobel filtered are used in this design. RGB into Ycbcr and color correction space are implemented in this design.

Image frame resolution is set to 1920x1080 at 23 frames per second and maximum full resolution of 2592x1944 is also supported but limited to 15 frames per second.

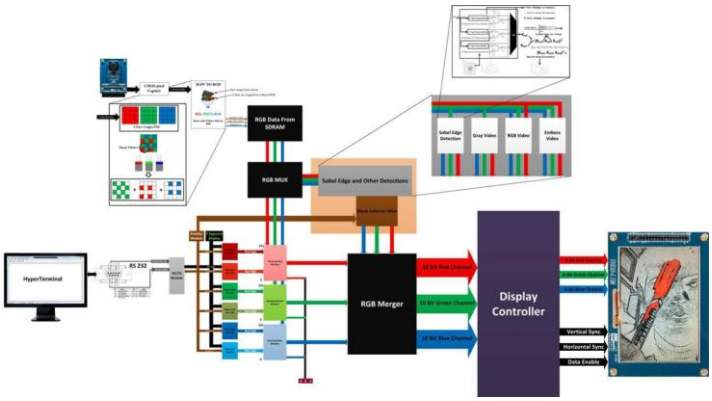


Figure 1

The idea of this project is to design and develop a stand-alone, FPGA-based smart camera that can perform the processing tasks such as Sobel, Prewitt, dilation edge detection, color object detection, emboss effects and output the alarm signal in the case of detection to short-range wireless systems using the zigbee. With the rapid development of remote sensing camera has become a popular sensor device for monitoring and surveillance systems. It is the first step of video analysis and understanding with remote sensing video especially the detection of object which is necessary step to extract the information from stream of binary raw data from the sensors. The purpose of the detection is to discover the information about the identified object in the presence of the surrounding objects. The sensor device used is a THDB-D5M, which is five mega pixel digital CMOS color image sensor, and it contains a power voltage circuit, a low noise amplifier and clock driver circuit. Its flexibility and ease of use makes it a good choice for the system. The raw data of color video stream and edge detection is very large, so speed of video processing is a difficult problem. Therefore, in order to improve the performance of real time video processing FPGA is an effective device to realize real-time parallel processing of vast amounts of video data. Moreover, adopting FPGA as a development platform than PC and other IC is to perform data intensive video processing tasks with real requirements. Therefore, architecture is implemented on a Zedboard (Xilinx Zynq xc7z020clg484-1) chip which communicate to an external camera D5M. The synthesized architecture has been tested in real-time environment to process full HD video. The processed filter data are sent to the on-board ARM (Cortex-A9) processor through VDMA which re-transmits to a remote desktop computer over Ethernet.

ARCHITECTURE

The Video processing frame core provides a modular expandable interface for video frame processing.

General architecture consists of camera interface module “camera_raw_to_rgb” which convert bayer format data into rgb format” rgb_set”. Video stream module which filters rgb data into various filters and axis external module which stream the filtered data to axi4-stream.

The idea behind to use the D5M camera for the FPGA was to have the easy connection of 40 pins between these two devices. This 40 pins connection provide input/output communication using the pixel clock, pixel data (12 bits), power (3.3V), Serial clock, serial data(I2C), frame valid and line valid. A typical camera system is shown in figure 1. At the heart of every digital camera is an image sensor either CCD image senor or a CMOS sensor. Both types of sensors capture the light through the lens and convert into the electrical signal which is further processed by ADC. Nowadays, majority of sensors consist of high-performance ADC converters which are employed to produce the digital output. Most CMOS image sensors have ADC resolutions of 10 to 12 bits. In this project, D5M camera has 12 bits ADC resolution. The D5M pixel array consists of a 2752-coloumn by 2004-row matrix. The pixel array is addressed by column and row. The array consists of a 2592-column by 1944-row active region and a border region as shown below. The output images are divided into frames, which are further divided into lines. The output signals frame valid and lines valid are used to indicate the boundaries between the lines and frames [3]. Pixel clock is used as a clock to latch the data. For each pixel clock cycle, one 12-bit pixel data outputs on the data out pins [3]. When both frame valid and line valid are asserted, the pixel is valid. Pixel clock cycles that occur when frame valid is negated are called vertical blanking. Pixel clock cycles that occur when only line valid is negated are called horizontal blanking. The camera has an array of 255 register, a lot of them are configurable, and it is possible to set up the operation of the camera by writing to these registers. This communication is performed using a generic two wire serial interface commonly referred to as I^2C[3]. There are two different communication modes, control / configuration mode which included read/write values to the register in the D5M card, and pixel data read out which consists of reading the pixel data from the camera card [3].

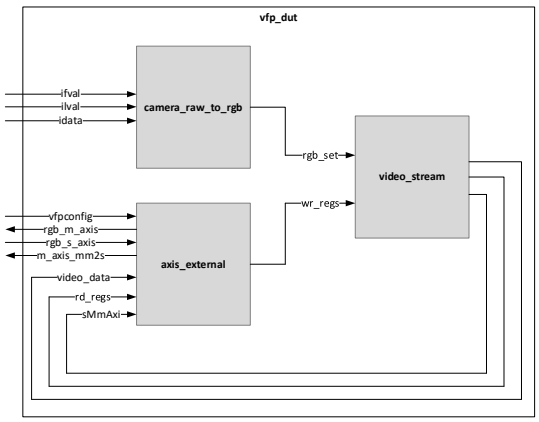


Figure 2 : Video Frame Processing Skeleton Architecture

FEATURES

Input format: Raw Bayer format
Output format: 16-bit YCbCr [4:2:2] (YUV) color space
24-bit User AXI Stream input.

Filters:

1. HSL
2. HSV
3. Sharp
4. Blur
5. Emboss
6. Sobel
7. YCbCr
8. Color Adjust Correction

CLOCKS

There are two clocks used in this design, pixel clock is nominally the pixel clock rate, with a set pll programed design frequency of 96MHz or below, and system clock which is a 150 Mhz.

CMOS PIXEL CAPTURE

A video frame processing contains a collection of control registers and maintain a local small buffer to store video frame location. Processors interact with the vfp core with vdma, vfp writes the video core's registers to configure its operation. A vfp core takes input raw video pixel stream, performs computation, or adds new contents to the stream through filters, and then outputs the processed pixel stream. Video frame processing can be either a pixel transformation or a pixel generation.

Pixel capture module is the first module inside the FPGA which communicate with camera. It receives the data of 12 bits per pixel at each clock cycle from the cmos camera when the frame valid and line valid are asserted high. Pixel clock (50MHz) is used to latch the data on the rising edge of the clock. In the case of when clear is set, then it captures on the falling edge of the pixel clock. Vertical blank occurs when frame valid is high. However, horizontal blank only occurs when both frames line valid and frame valid are high. To pause, restart, snapshot and change the exposure level of the video, certain registers need to be assigned the hex values. These hex values are given in the datasheet of this camera. Setting these values will enable the features and functionality of the camera. These features were enabled by using the I2C bus which has two wires SDA and SCL. SDA is the data line. SCL is the clock line which is used to synchronize all data transfers over the I2C bus. The SCL & SDA lines are connected to the FPGA and the camera on the I2C bus. Once the registers were assigned the values and valid signal are asserted, the camera output the 12 bits data at the maximum data rate of 96Mp/s. Input clock for the camera is 96MHz which gives the maximum data rate, but this data rate should be latched at the 50MHz clock. Active pixels: 2,592H x 1,944V Pixel size: 2.2µm x 2.2µm Color filter array RGB Bayer pattern Full resolution Programmable up to 15 fps Frame rate VGA (640 x 480) Programmable up to 70 fps ADC resolution: 12-bit Pixel dynamic range: 70.1dB SNRMAX: 38.1dB Supply Power Voltage: 3.3V I/O Voltage: 1.7V～3.1V

VFP

Prior to the implementation of the various filters, it necessary to overview top module and io connections to internal modules. Generic defined in vfp give user option to select various filter, config axi4-lite address/data bus width and set revision number. The TDATA_WIDTH generic specifies the number of bits in a word and the ADDR_WIDTH specifies the number of address bits, which implies that there is 2^ADDR_WITH words. BMP_WIDTH AND BMP_HEIGHT specifies image frame width and height.

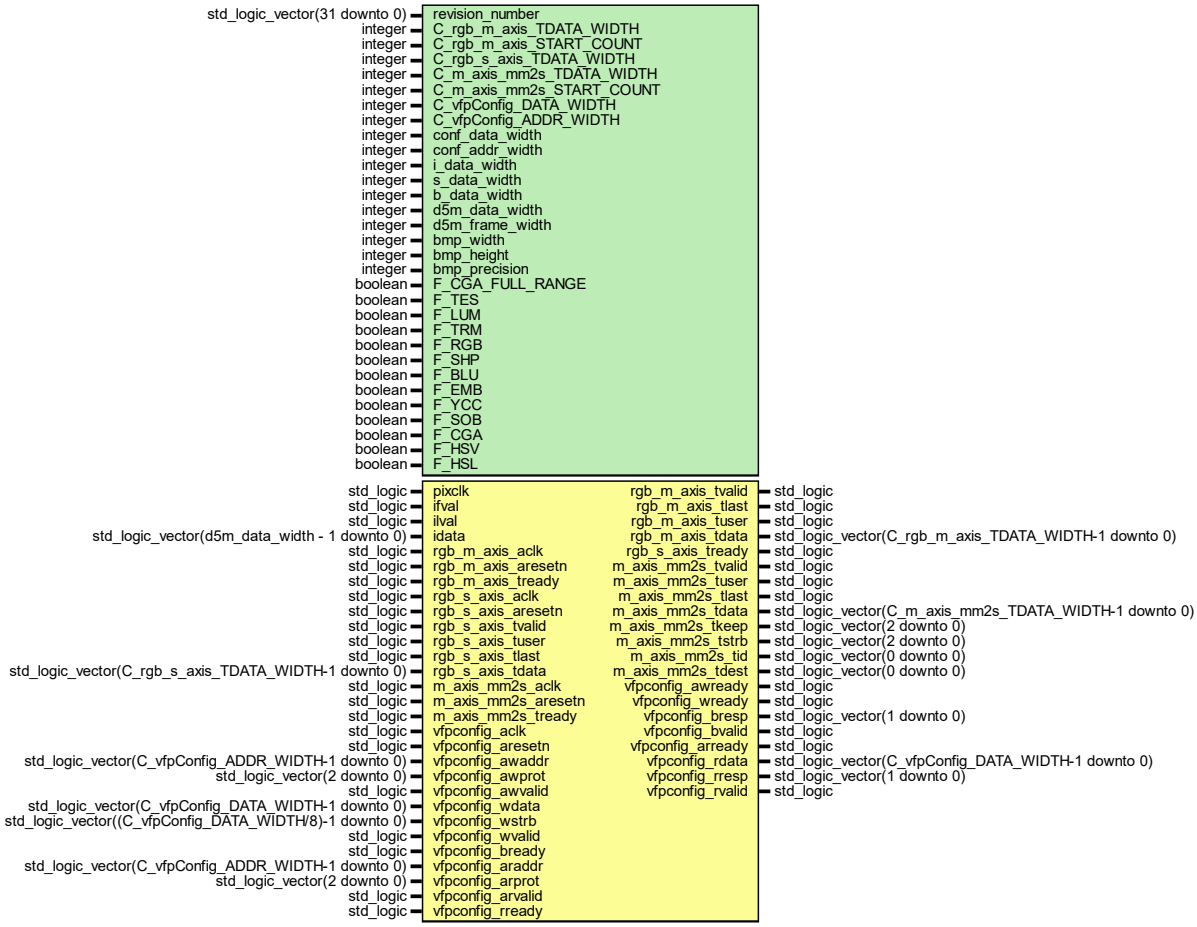


Figure 3

To configure video stream register, axi4-lite protocol is implemented which present a memory map register interface to the processor.

HSL COLOR SPACE

This module converts rgb color space to hsl color space. First logic calculates maximum and minimum value of rgb values. Hue is calculated first determining the hue fraction from greatest rgb channel value. If current max channel is red than Hue numerator will be set to be green subtract blue only if green is greater than blue else blue is subtracted from green and Hue degree would be zero. If current max channel is green than Hue numerator will be set to be blue subtract red only if blue is greater than red else red is subtracted from blue and Hue degree would be 129. Similarly, if current channel is blue than Hue numerator will be set to be red subtract green only if red is greater than green else green subtracted from red and Hue degree would be 212. Hue denominator would be rgb delta. Once Hue fraction values are calculated than fraction values would be added to hue degree which would give final hue value as done logic. Saturate value is calculated from difference between rgb max and min over rgb max whereas Lightness value rgb max value.

HSV, HSL and HSI: Hue is a color range degree from 0 degree to 360 degree which describe a pure color. Saturate is shade of gray to full color. Amount of saturation of color is known as chroma. Higher value of chroma is clear and bight. The strongest magnitude is value and its range correspond to brightness and balanced magnitude corresponds to the intensity.

From RGB triplet saturation equation shown below where max is calculated of between red, green, and blue channel. It represents the strength of the color and the radius of the cone.

$$saturate = \frac{(\max(rgb) - \min(rgb))}{\max(rgb)}$$

- 0 degree – Red
- 60 degree – Yellow
- 120 degree – Green
- 240 degree – Blue
- 300 degree – Magenta

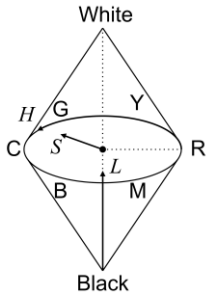


Figure 4

The

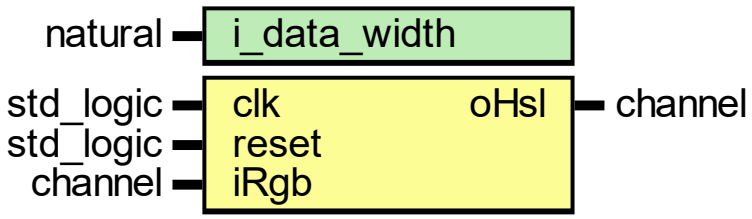
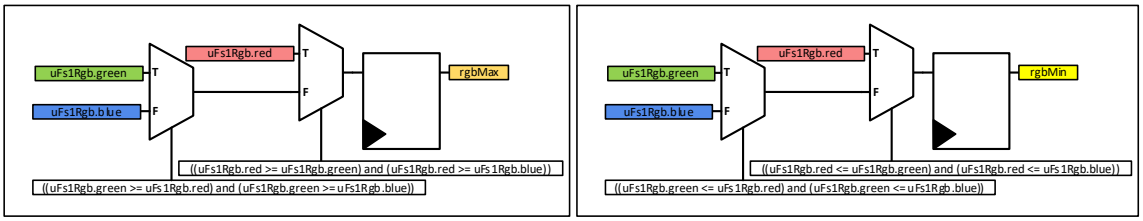


Figure 5

Rgb to hsl module convert input rgb to hsl color space. The module has clock and reset ports. Port iRGB consist of red, green, and blue rgb channels with valid signal, which is used to convert to rgb pixels to hsl pixels.

Ports	Description
clk	Reference clock for input and output data stream.
reset	Specifies module asynchronous active low reset.
iRgb.red	8-bit input data. Red value.
iRgb.green	8-bit input data. Green value.
iRgb.blue	8-bit input data. Blue value.
iRgb.valid	Input data valid. Specifies whether the next data point has arrived for processing.
oHsl.red	8-bit output data. Hue value.
oHsl.green	8-bit output data. Saturate value.
oHsl.blue	8-bit output data. Luminosity value.
oHsl.valid	Output data valid. Control signal to indicate the validity of each pixel.

The Functional block diagram of the implemented rgb to hsl color space conversion is shown in Figure 9-12.



The max and min rgb value is calculated according to logic implementation shown in figure above.

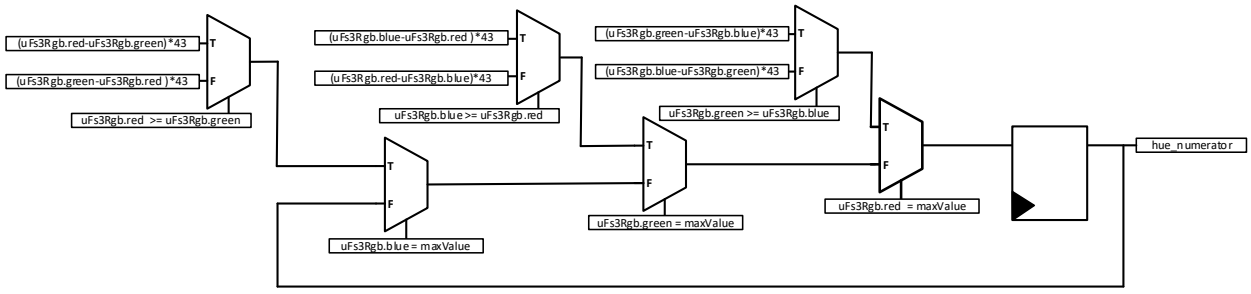


Figure 6: Hue Numerator Logic

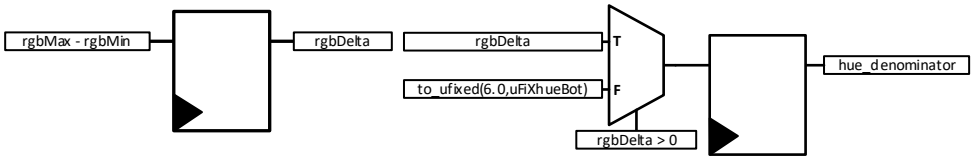


Figure 7: Hue Denominator Logic

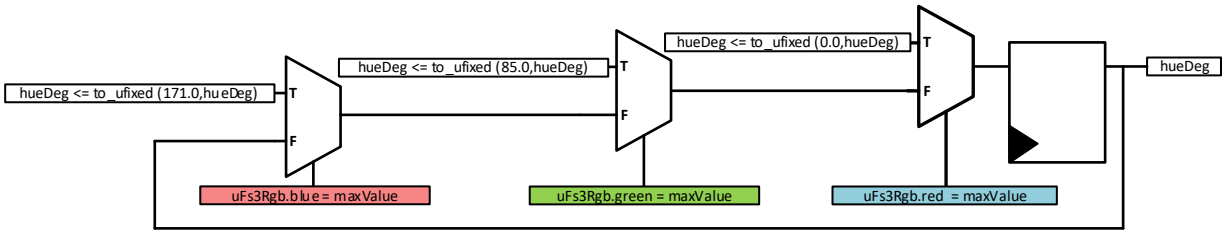


Figure 8: Hue Degree Logic

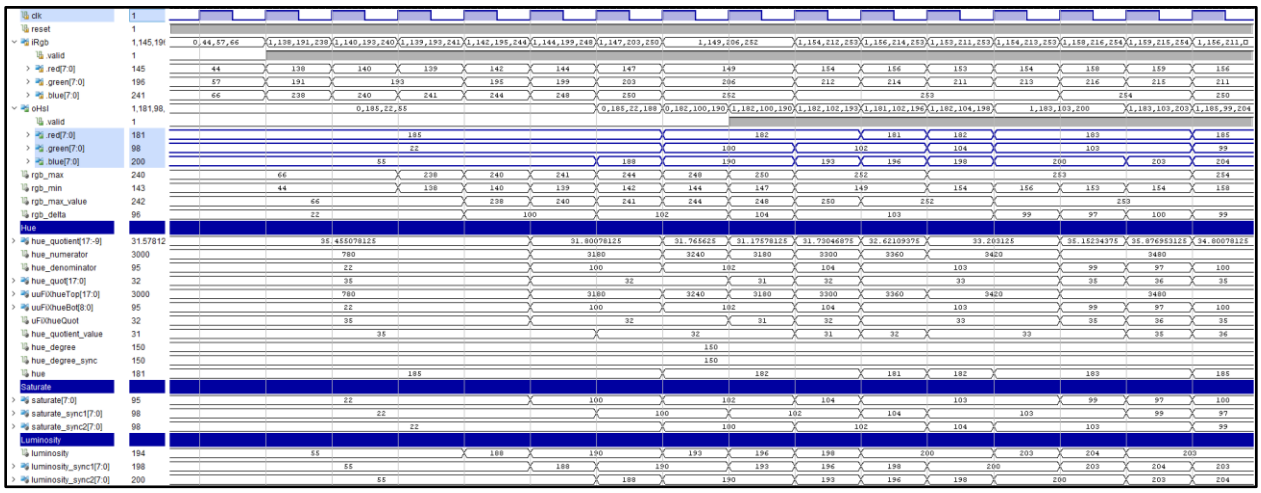


Figure 9 : HSL Filter Wave Diagram

The simulation results of rgb channel conversion to hsl color space is presented in wave diagram figure 8.

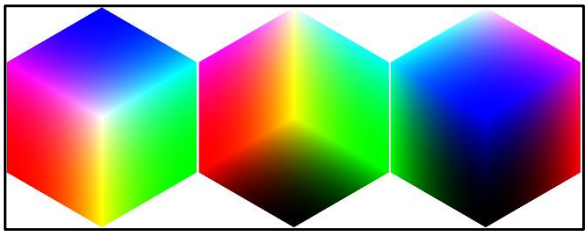


Figure 10

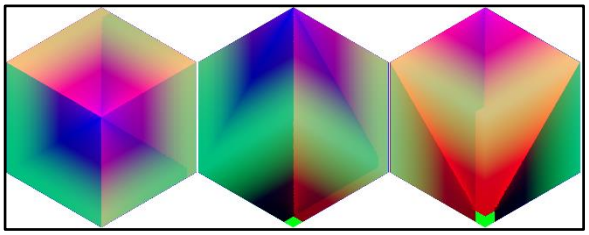


Figure 11

Below figures shows rgb color space conversion implementation into six regions of the hexagon images. The representation of the RGB color space shown in 1st and 3rd figure.

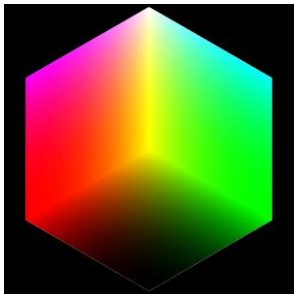


Figure 12: RGB image

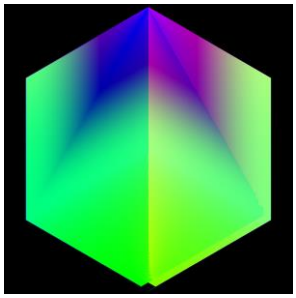


Figure 13: HSL converted image

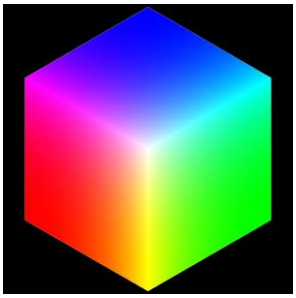


Figure 14: RGB image

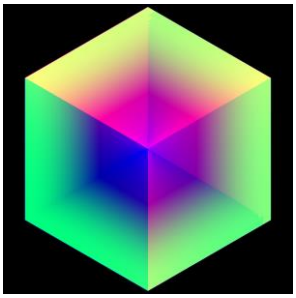


Figure 15: HSL converted image

1st and 3rd figure are rgb image and whereas 2nd and 4th are hsl simulated results.

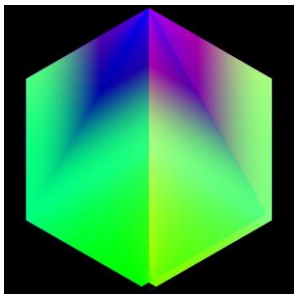


Figure 16: HSL Image

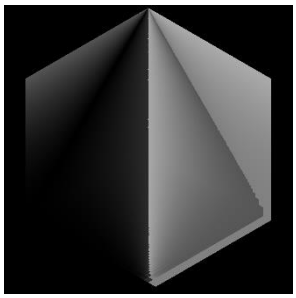


Figure 17: Hue channel

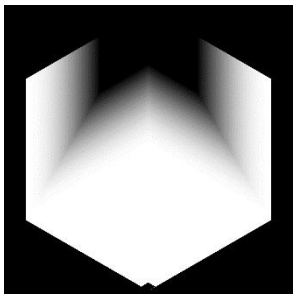


Figure 18: Saturate channel

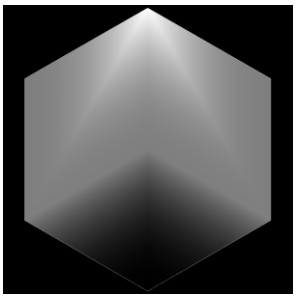


Figure 19: Luminosity channel

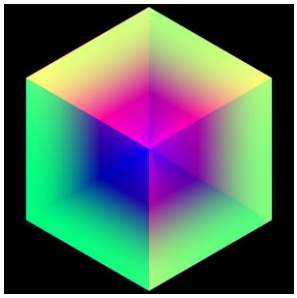


Figure 20: HSL Image

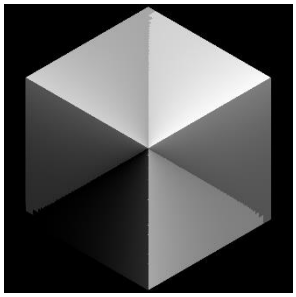


Figure 21: Hue channel

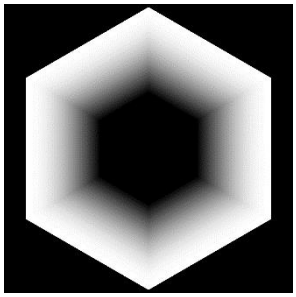


Figure 22: Saturate channel

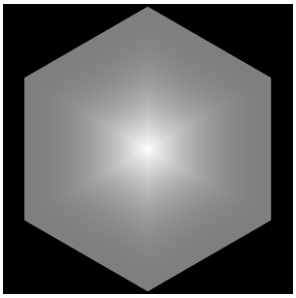
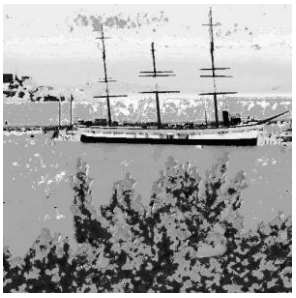
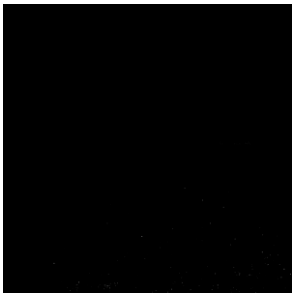


Figure 23: Luminosity channel

1st image is hsl image, 2nd represent hue channel, 3rd saturate channel and 4th luminosity channel.





1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

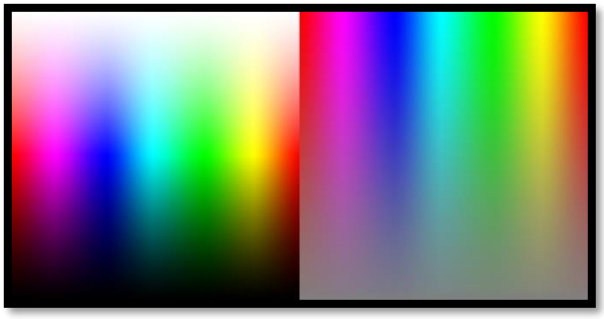


Figure 24

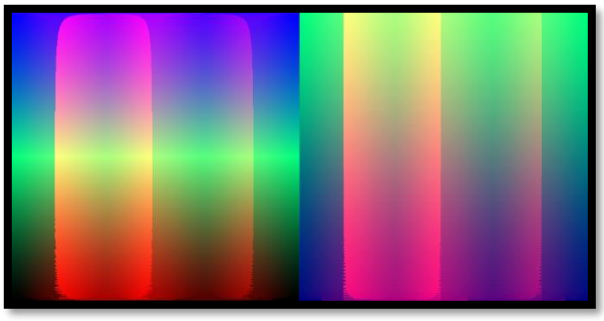


Figure 25

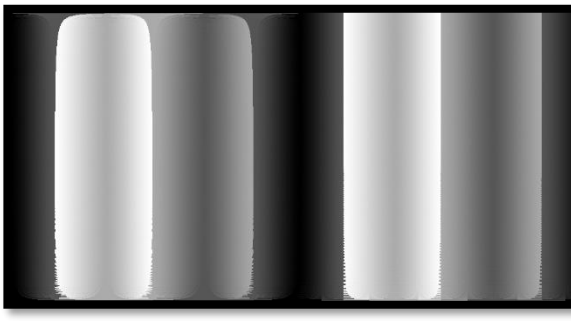


Figure 26

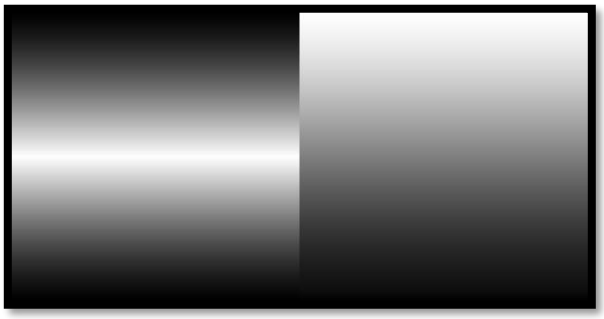


Figure 27

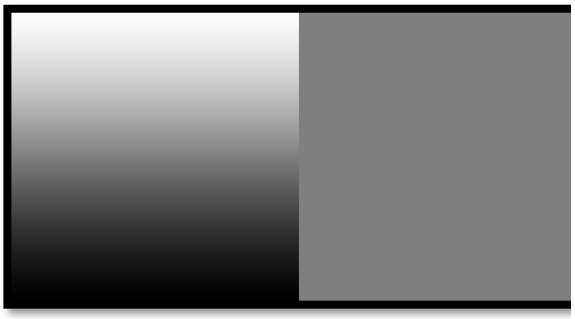


Figure 28

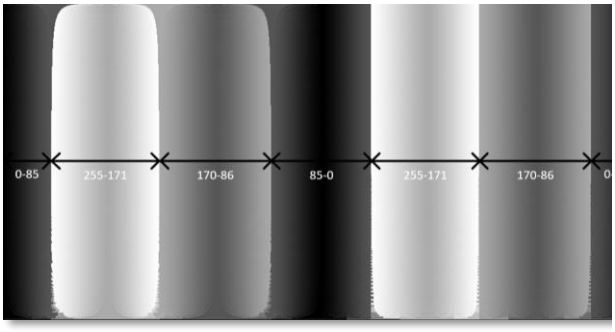


Figure 29

1st image is hsl image, 2nd represent hue channel, 3rd saturate channel and 4th luminosity channel. HSL is cylindrical geometry with hue, angular dimension colors transition from red to orange, yellow, green, cyan, blue, magenta.

HSV COLOR SPACE

This module converts rgb color space to hsl color space. First logic calculates maximum and minimum value of rgb values. Hue is calculated first determining the hue fraction from greatest rgb channel value. If current max channel is red than Hue numerator will be set to be green subtract blue only if green is greater than blue else blue is subtracted from green and Hue degree would be zero. If current max channel is green than Hue numerator will be set to be blue subtract red only if blue is greater than red else red is subtracted from blue and Hue degree would be 129. Similarly, if current channel is blue than Hue numerator will be set to be red subtract green only if red is greater than green else green subtracted from red and Hue degree would be 212. Hue denominator would be rgb delta. Once Hue fraction values are calculated than fraction values would be added to hue degree which would give final hue value as done logic. Saturate value is calculated from difference between rgb max and min over rgb max whereas Intensity value rgb max value.

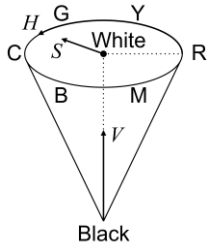


Figure 30

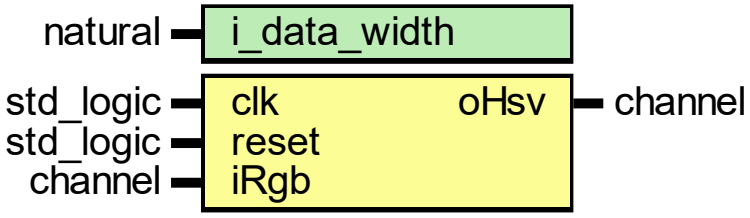


Figure 31

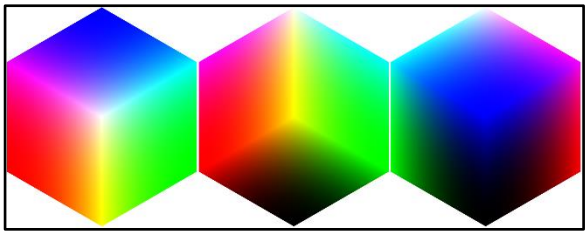


Figure 32

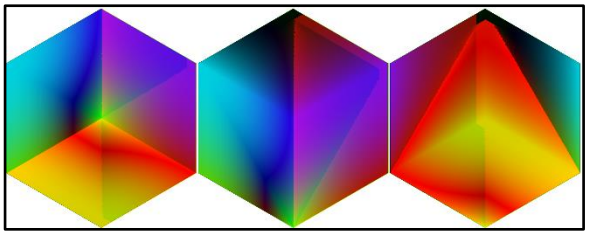


Figure 33

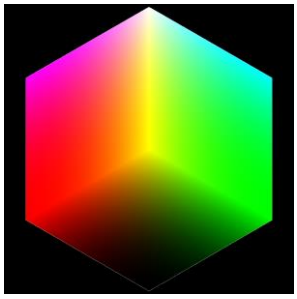
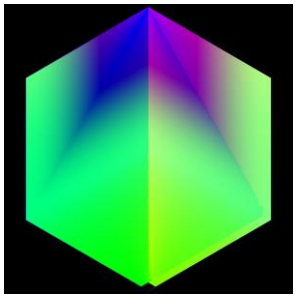
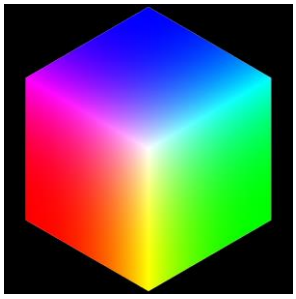
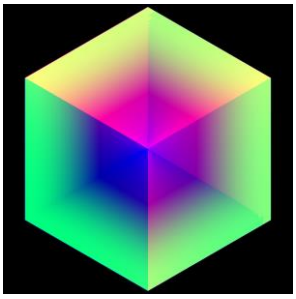





Figure 34 *Figure 35* *Figure 36* *Figure 37*

1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results.

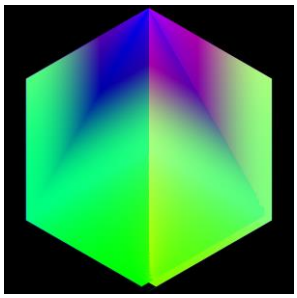
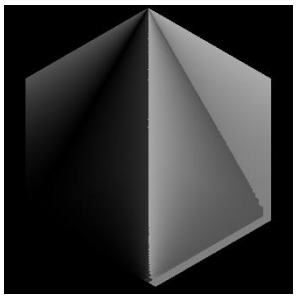
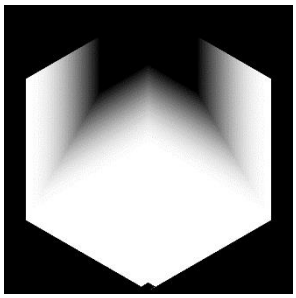
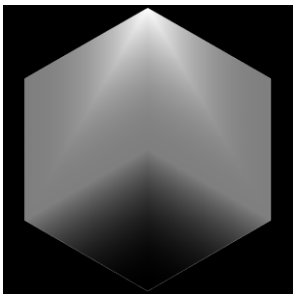





Figure 38 *Figure 39* *Figure 40* *Figure 41*

1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results.

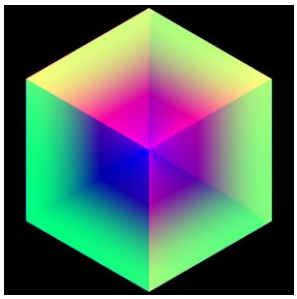
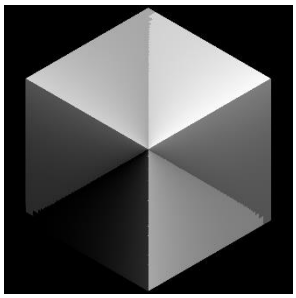
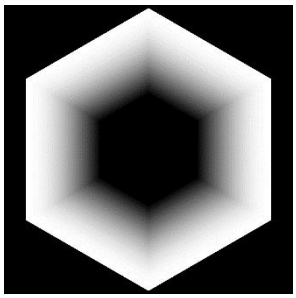
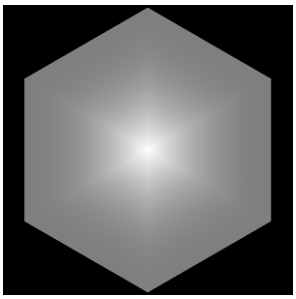





Figure 42 *Figure 43* *Figure 44* *Figure 45*

1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results.

Figure 46

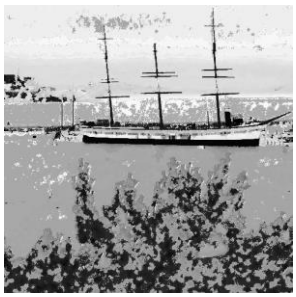


Figure 47

Figure 48

Figure 49

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSL COLOR RANGE SPACE

Figure 50

Figure 51

Figure 52

Figure 53

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

COLORHSL

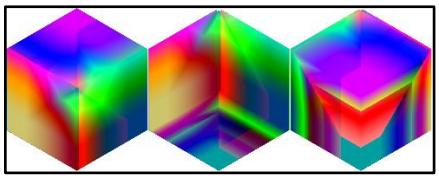


Figure 54

Figure 55

Figure 56

Figure 57

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSL_1RANGE

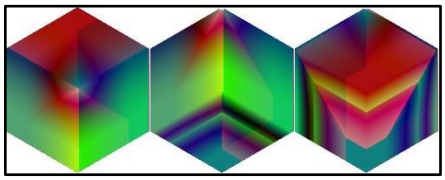


Figure 58

Figure 59

Figure 60

Figure 61

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSL_2RANGE

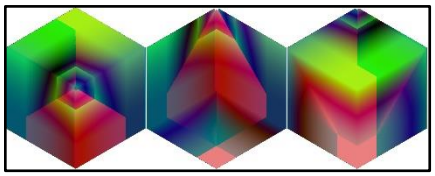


Figure 62

Figure 63

Figure 64

Figure 65

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSL_3RANGE

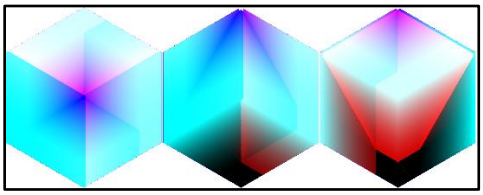


Figure 66

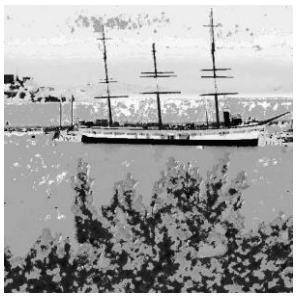


Figure 67

Figure 68

Figure 69

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSVL_1RANGE

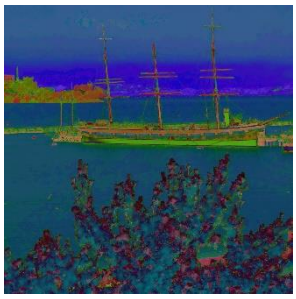


Figure 70

Figure 71

Figure 72

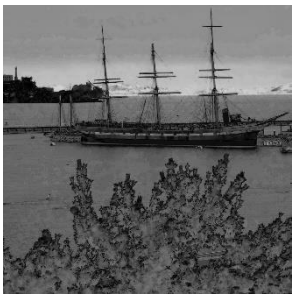
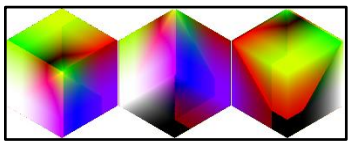


Figure 73

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSVL_2RANGE



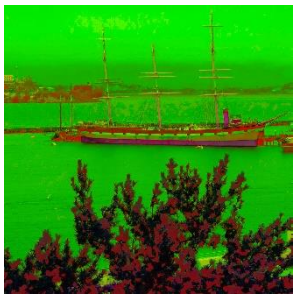


Figure 74

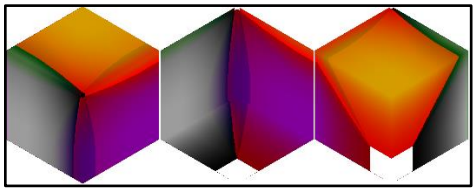
Figure 75

Figure 76

Figure 77

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSVL_3RANGE



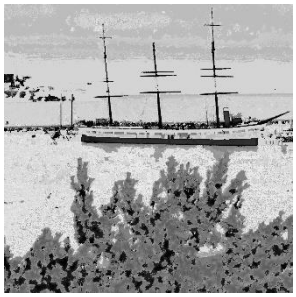


Figure 78 *Figure 79* *Figure 80* *Figure 81*

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

HSVL_4RANGE

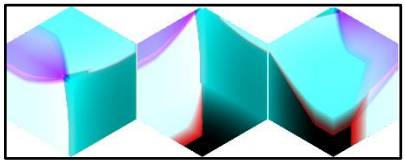
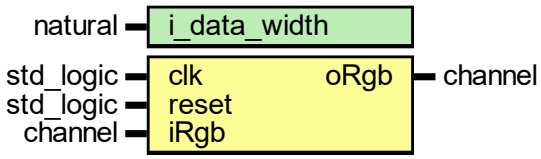


Figure 82 *Figure 83* *Figure 84* *Figure 85*

1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

CMYK COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam cmyk color space in output oRgb channel.



CMYK (Cyan, Magenta, Yellow, Key/Black) is the color space for printed materials.

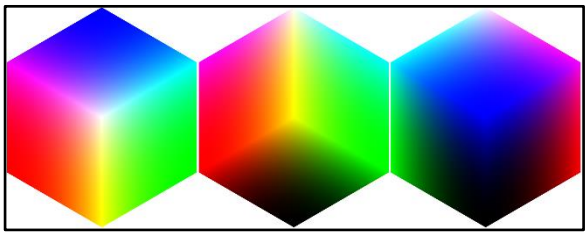


Figure 86

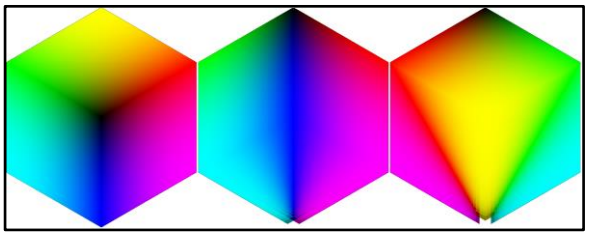


Figure 87

Figure 88

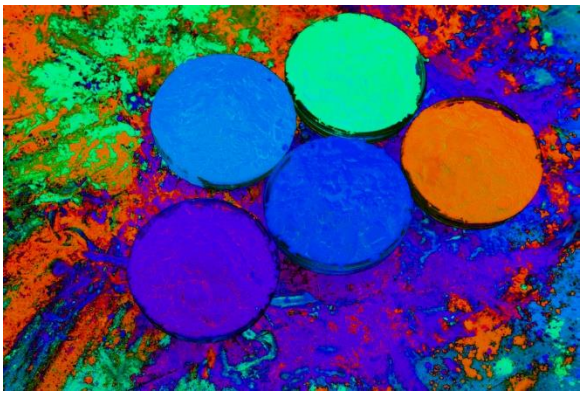


Figure 89

Figure 90

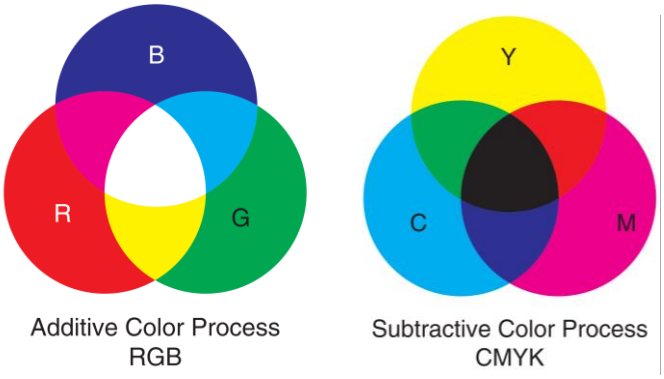
Figure 91

From these values, the value of colors can be calculated from the following equations:

$$K = 1 - MAX(R, G, B)$$

$$C = (1 - R - K)(1 - K)$$
$$M = (1 - G - K)(1 - K)$$
$$Y = (1 - B - K)(1 - K)$$

CMYK color space is used for printing color images. The color printer makes use of subtractive(secondary) color scheme. The minimum requirement for color printer is equipped with three primary colors: cyan(C), magenta (M), and yellow (Y). The secondary colors Yellow, Cyan and Magenta are generated by adding primary colors Red, Green and Blue. Above figures shows, a 500 x 500 color image in RGB and CYM formats. The pinkish color in RGB transformed into greenish, as the red and blue component intensities are high compared to green. The black color is generated by combining all the colors whereas white is set with no colors.

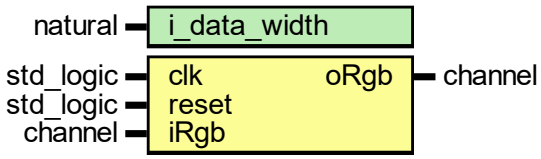


Additive Color Process
RGB

Subtractive Color Process
CMYK

YD_BD_R COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam YD_RD_B color space in output oRgb channel.



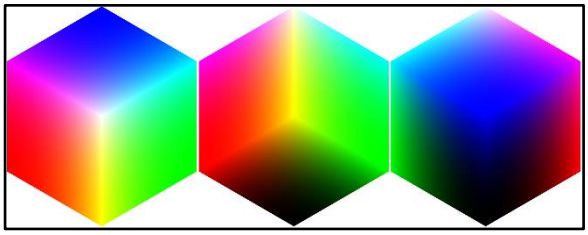


Figure 92 RGB Image

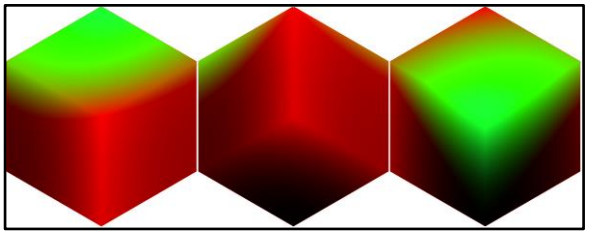


Figure 93 YDBDR Image

Figure 94 RGB Image

Figure 95 YDBDR Image

In this color space, Figure 95 image represent by three components, namely, Y, D_B, and D_R.

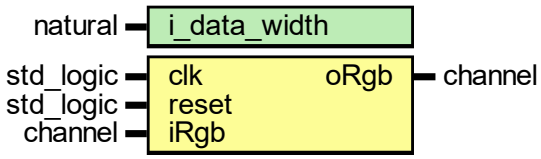
The R, G, B tristimulus values converted to Y, D_B, D_R tristimulus values as follows:

$$\begin{bmatrix} Y \\ D_R \\ D_B \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.450 & -0.883 & 1.333 \\ -1.333 & -1.160 & 0.217 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Above figures illustrates the result of convert color image space in RGB domain to image with YDRDB components.

CIEXYZ COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam CIE-XYZ color space in output oRgb channel.



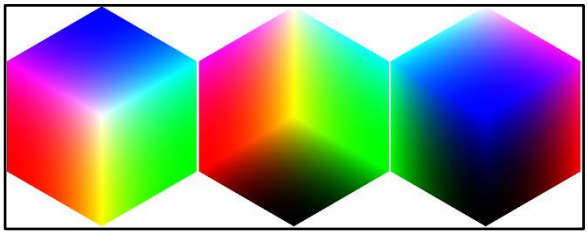


Figure 96

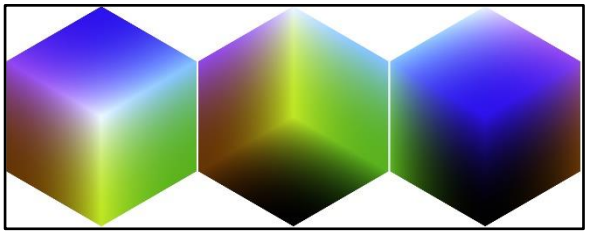


Figure 97

Figure 98

Figure 99

Above figures illustrates the result of convert color image space in RGB domain to image with XYZ components.

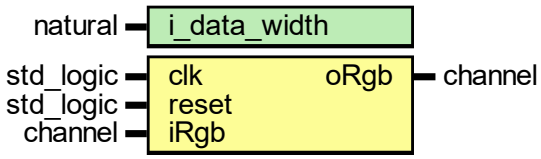
The R, G, B tristimulus values converted to X, Y, Z tristimulus values. The relationship between RGB and the XYZ values expressed by a 3x3 matrix as shown in equation below.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412 & 0.357 & 0.180 \\ 0.212 & 0.715 & 0.072 \\ 0.019 & 0.119 & 0.950 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The coefficients values in 3x3 matrix are multiplied to input RGB components and the sum of the result values as per XYZ channel.

CIEYUV COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam CIE-YUV color space in output oRgb channel.



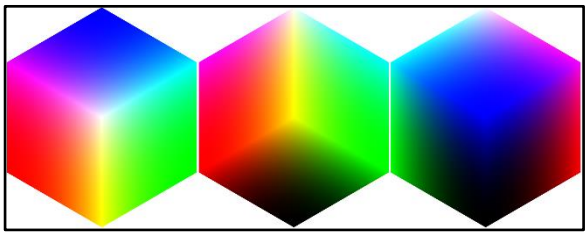


Figure 100

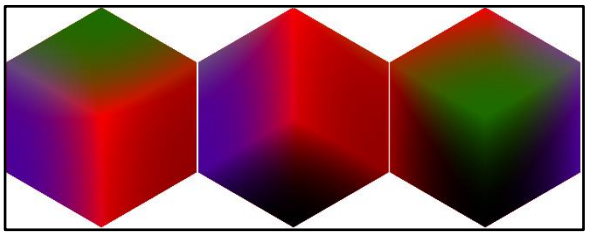


Figure 101

Figure 102

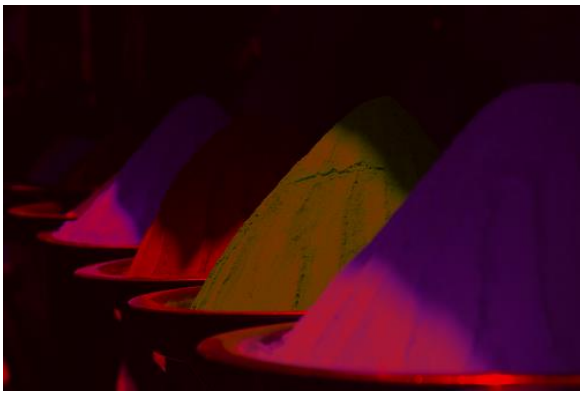


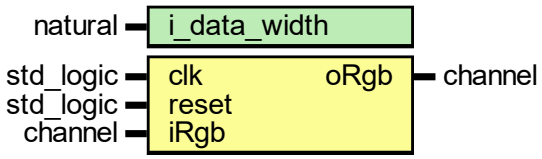
Figure 103

The R, G, B tristimulus values converted to Y, U, V tristimulus values as follows:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

YIQ COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam YIQ color space in output oRgb channel.



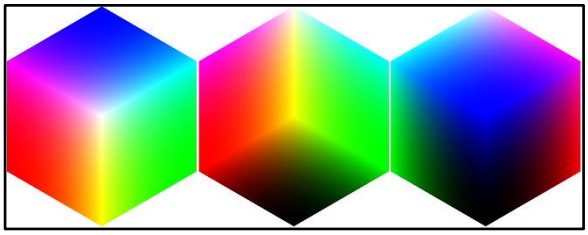


Figure 104

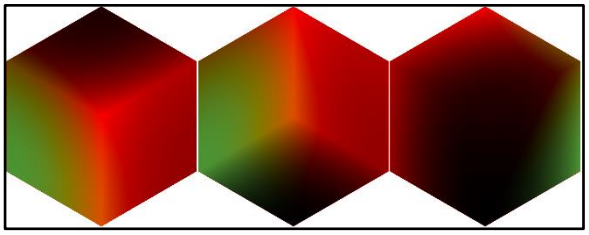


Figure 105

Figure 106

Figure 107

In this color space, Figure 107 image represent by three components, namely, Y, I, and Q. The Y-channel (this corresponds roughly with intensity) is the luminance component, and I(In-Phase)/Q(Quadrature) represent chrominance (color information: chroma, jointly describe the hue and saturation). This color space separate intensity information from color details.

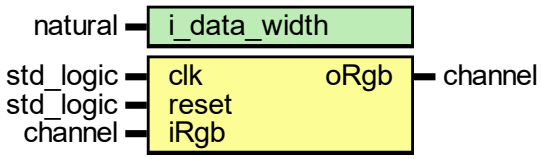
The R, G, B tristimulus values converted to Y, I, Q tristimulus values as follows:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595 & -0.274 & -0.321 \\ 0.211 & -0.522 & -0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figure 106 shows RGB converted to YIQ color space Figure 106 using the transformation of above equation. Note the elements of 1st row in the conversion matrix sum to 1 and 2nd and 3rd rows sum to zero. IQ components are transmitted with slower amplitude variations with limited image details whereas Y component contain more gray level image details of information.

YPBPR COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam YP_BP_R color space in output oRgb channel.



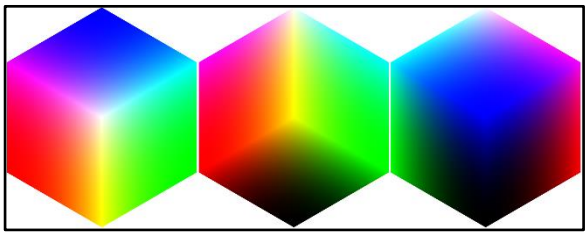


Figure 108

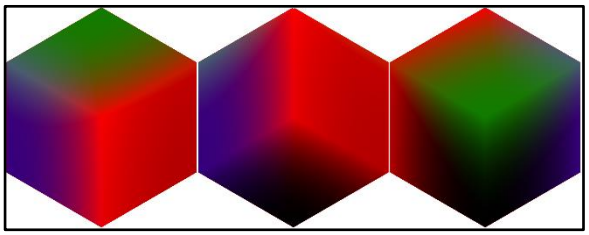


Figure 109

Figure 110

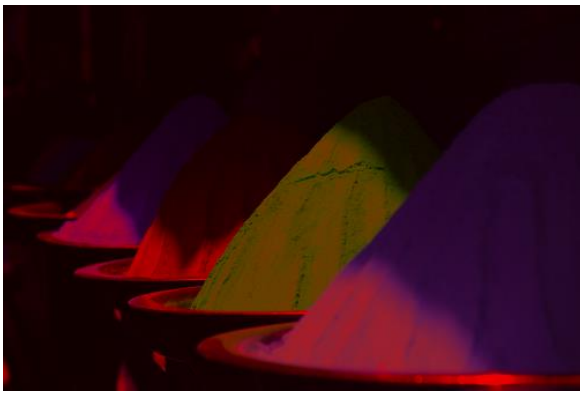


Figure 111

In this color space, Figure 111 image represent by three components, namely, Y, P_B, and P_R. The Y-channel corresponds to luminance component, P_B channel corresponds to difference between blue and luma and whereas P_R channel corresponds to difference between red and luma.

The R, G, B tristimulus values converted to Y, P_R, P_B tristimulus values as follows:

$$\begin{bmatrix} Y \\ P_R \\ P_B \end{bmatrix} = \begin{bmatrix} 0.213 & 0.715 & 0.072 \\ -0.115 & -0.385 & 0.500 \\ 0.500 & -0.454 & -0.046 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

I1I2I3 OHTA COLOR INTENSITIES

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam I1I2I3 color space in output oRgb channel.

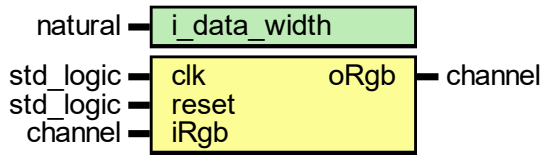


Figure 112

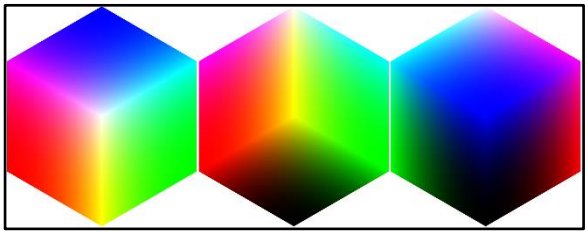


Figure 113

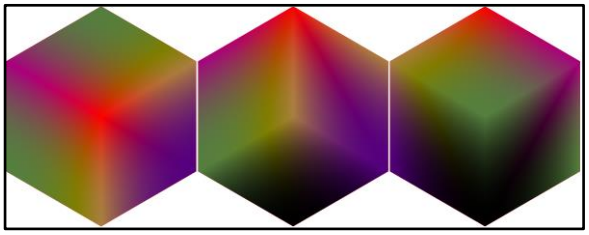


Figure 114

Figure 115

Figure 116

$$RED_{INTENSITY} = I_1 = \frac{\text{RED} + \text{GREEN} + \text{BLUE}}{3}$$

$$GREEN_{INTENSITY} = I_2 = \frac{\text{RED} - \text{BLUE}}{2}$$

$$BLUE_{INTENSITY} = I_3 = \frac{2\text{xGREEN} - \text{RED} - \text{BLUE}}{4}$$

LMS COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam LMS color space in output oRgb channel.

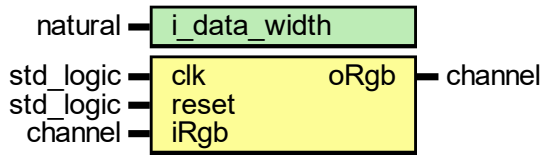


Figure 117

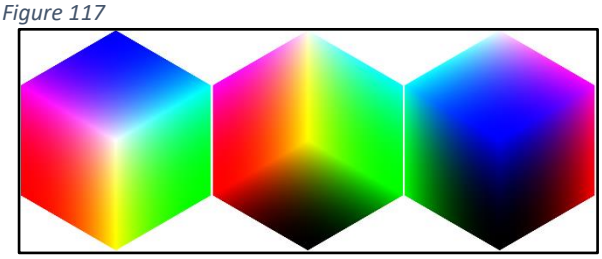


Figure 118

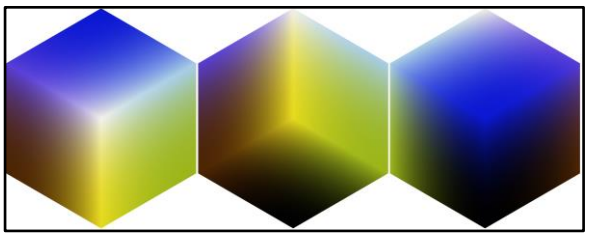


Figure 119

Figure 120

Figure 121

The R, G, B tristimulus values converted to L, M, S tristimulus values as follows:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = \begin{bmatrix} 0.400 & 0.707 & -0.080 \\ -0.228 & 1.150 & 0.061 \\ 0.000 & 0.000 & 0.918 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

IC_TC_P COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam IC_TC_P color space in output oRgb channel.

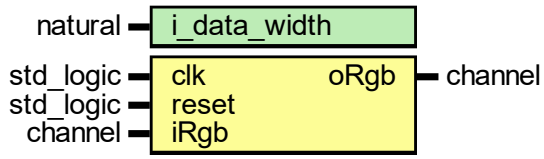


Figure 122

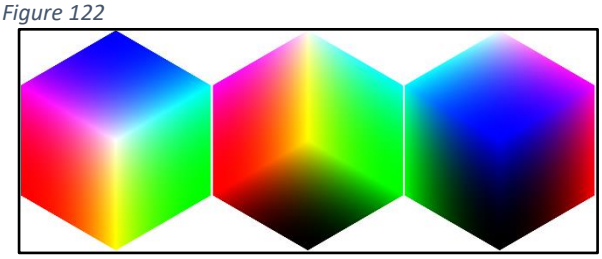


Figure 123

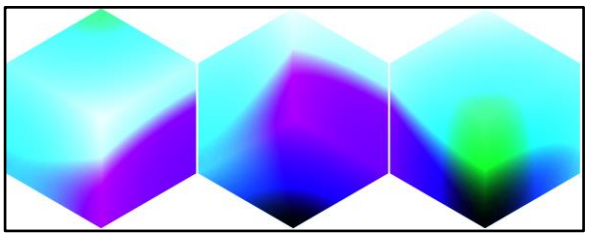


Figure 124

Figure 125

Figure 126

The R, G, B tristimulus values converted to I, C_T, C_P tristimulus values as follows:

$$\begin{bmatrix} I \\ C_T \\ C_P \end{bmatrix} = \begin{bmatrix} 0.400 & 0.400 & 0.200 \\ 4.455 & -4.851 & 3.960 \\ 8.056 & 3.572 & -1.162 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

HED COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam HED color space in output oRgb channel.

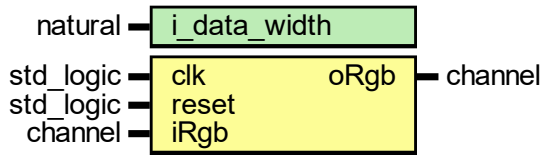


Figure 127

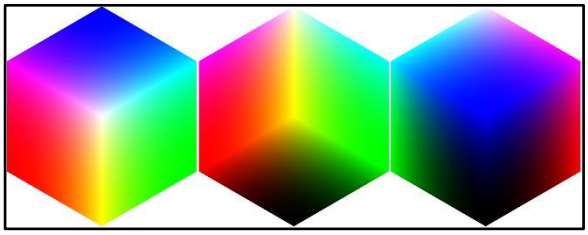


Figure 128

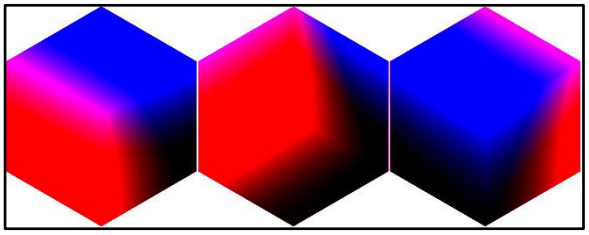


Figure 129

Figure 130

Figure 131

The R, G, B tristimulus values converted to H, E, D tristimulus values as follows:

$$\begin{bmatrix} H \\ E \\ D \end{bmatrix} = \begin{bmatrix} 1.800 & -0.070 & -0.600 \\ -1.020 & -1.130 & -0.480 \\ -0.550 & -0.130 & 1.570 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

YC1C2 COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam YC_1C_2 color space in output oRgb channel.

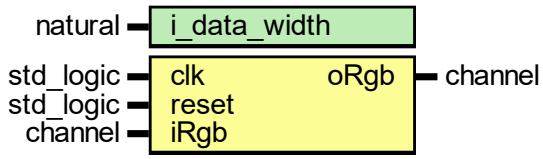


Figure 132

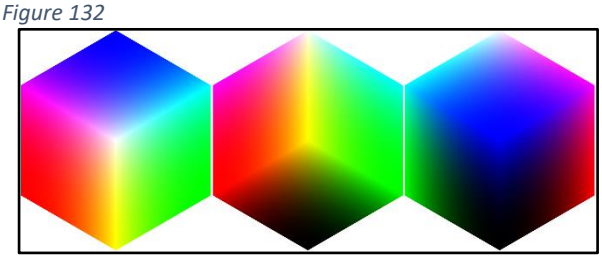


Figure 133

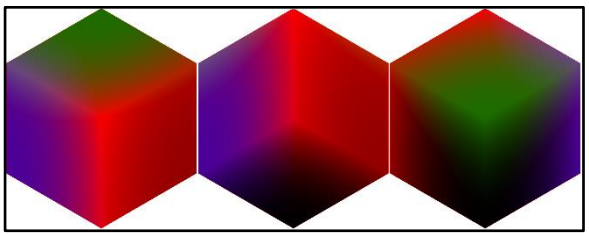


Figure 134

Figure 135

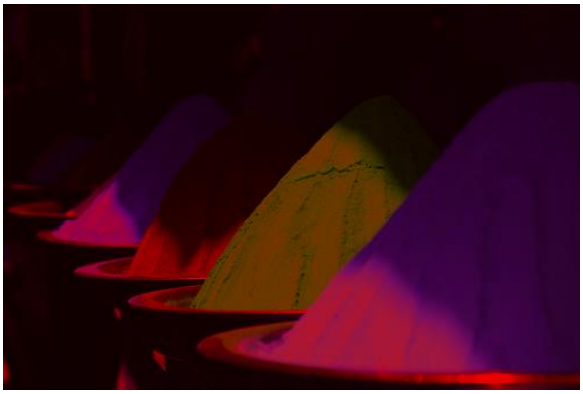


Figure 136

The R, G, B tristimulus values converted to Y, C_1, C_2 tristimulus values as follows:

$$\begin{bmatrix} Y \\ P_R \\ P_B \end{bmatrix} = \begin{bmatrix} 0.213 & 0.715 & 0.072 \\ -0.115 & -0.385 & 0.500 \\ 0.500 & -0.454 & -0.046 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

SHARP FILTER

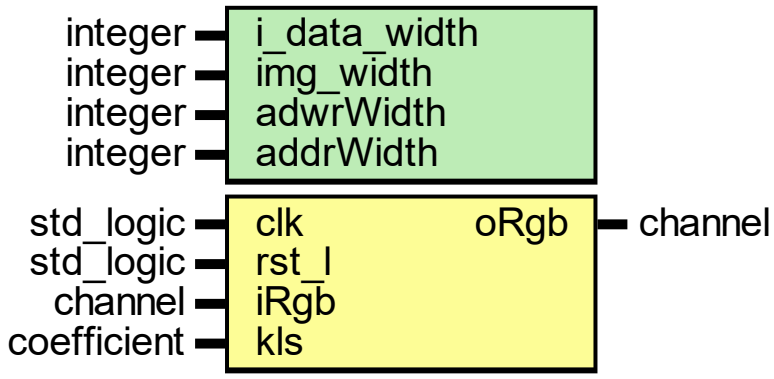
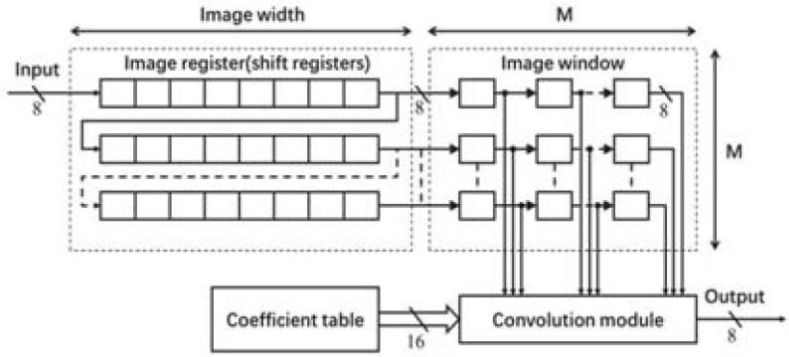


Figure 137



The sharp filter has been implemented and tested for an image size of 1024 × 1024 as shown figure below.

BLUR FILTER

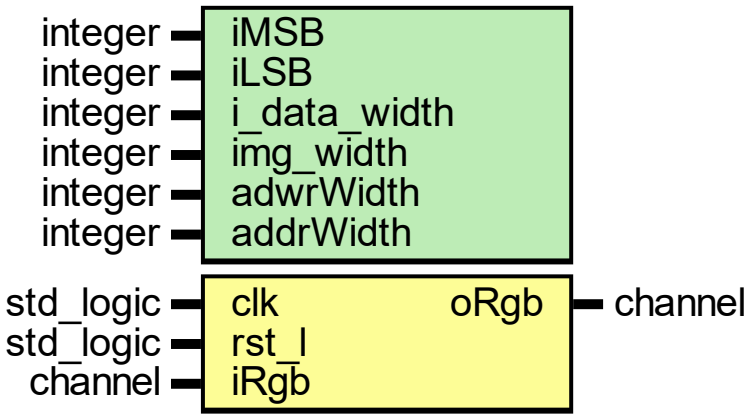
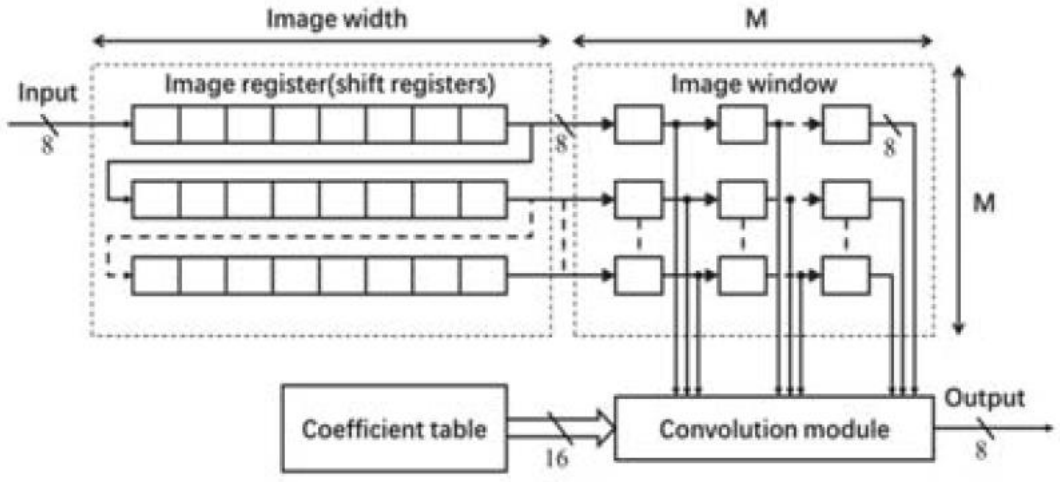
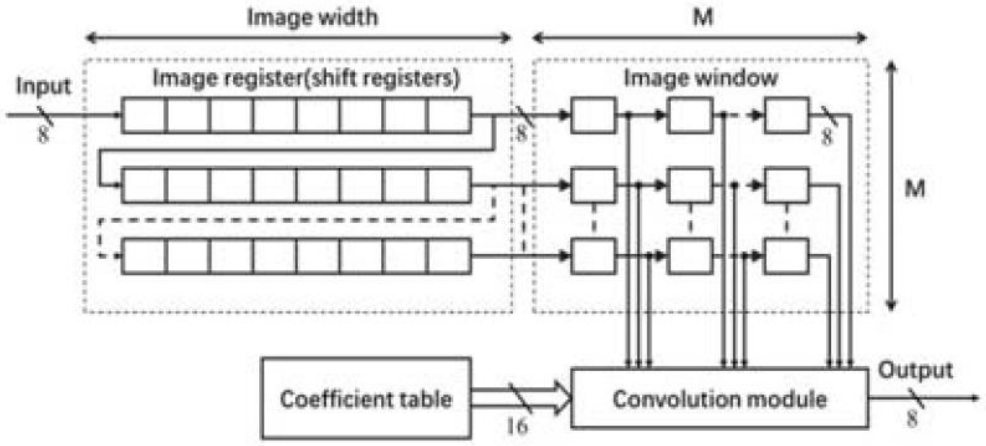


Figure 138



Gray Level Colors			
0.400	0.350	0.200	0.950
0.200	0.700	0.100	1.00
0.05	0.100	0.900	1.05
0.605	1.15	1.20	**∑**

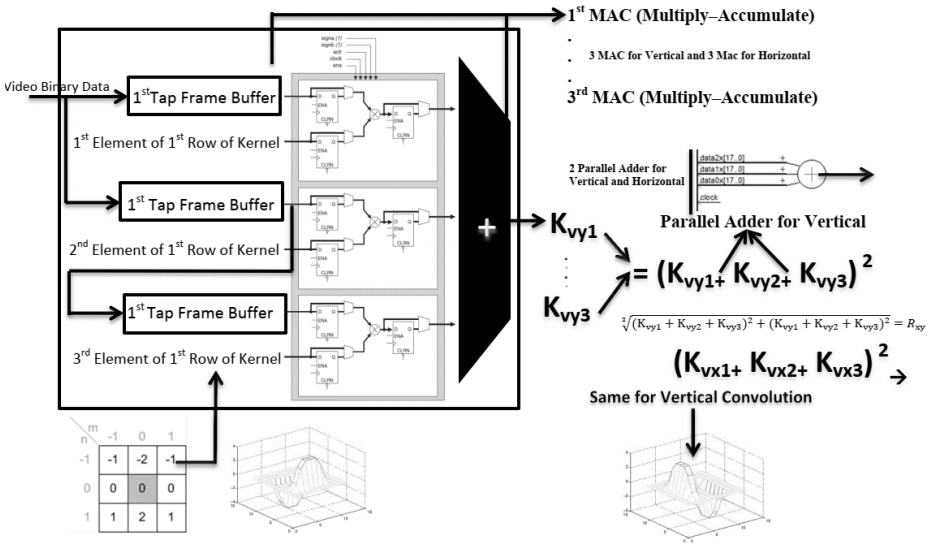
EMBOSS FILTER



An emboss filter will take video frame and convert into an embossed image.

SOBEL FILTER

For Sobel edge detection, monochrome RGB pixel are used of 8 bits which are stored in row-by-row memory of successive pixels. Two Sobel filters are used for edge detection. There are two distinct filters, one which computes vertical edge left and one for horizontal edge right. The filters are convolved with the entire input 3-line buffer which holds the pixel information. The vertical edge component is calculated with vertical filter K_y into input the first tap1 of 8 bits and then tap2 and tap3 sum to 3 K_y taps (3 elements of 1st rows of filter). The Same calculation is applied to horizontal filter 3 columns Kx gives the output of sum 3 K_x taps(3 elements of 1st column of filter). The resulted sum 3 K_y taps gives us the final sum of Vertical K_{vy}1,2,3 and 3 sum for horizontal Kvx1,2,3. Final output of vertical and horizontal is given by R_{xy} which is the square root of sum K_{vy} and K_{vx}. √ . The Figure below illustrates the concept of Sobel Filter for Video Edge detection.



Three lines video buffer have used for a filter with 3x3 kernel for both vertical and horizontal edge detection. Then the video buffer enters the filter in a parallel form. Each 3 t-taps module takes the data taps from the line buffer with a rate faster than system clock rate by 3 times and multiply them by the 3 filter coefficients. The output from each multiplications process is accumulated in the accumulator. Then finally sum all three 3-Tap filter from the accumulator fed into parallel adder which form one processed pixel. Once the R_{xy} is calculated the value is compared with threshold which is the user input from the switches (12 to 1). If the pixel output from R_{xy} is greater than the threshold, the pixel is detected as the edge otherwise no output to the RGB channel.

A Sobel filter will take video frame and convert into an Sobel image.

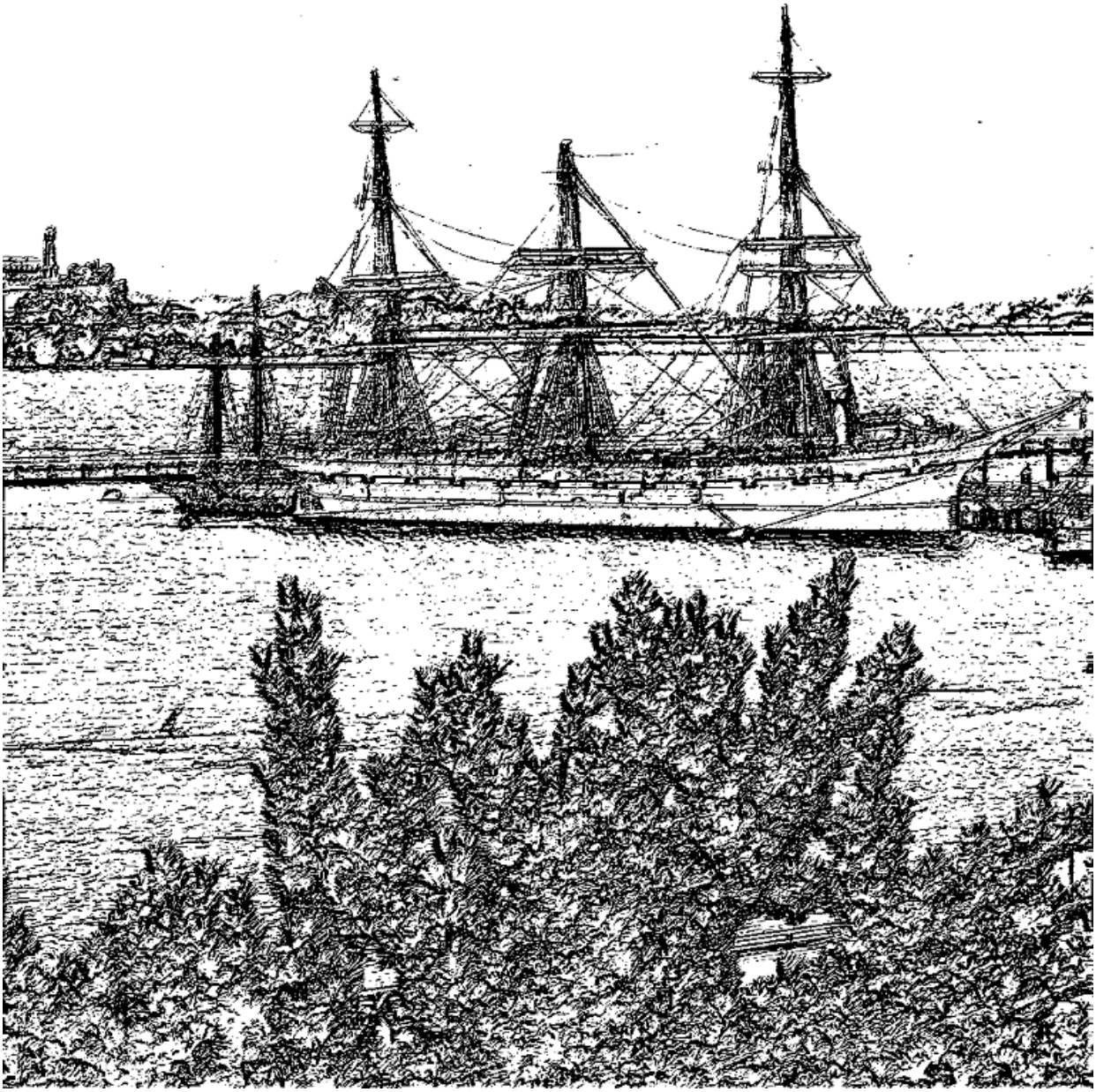


Figure 139

YCBCR COLOR SPACE

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs iRgb.red, iRgb.red, iRgb.red, iRgb.valid, iRgb.eol, iRgb.eof and iRgb.sof. The result of this module steam CIE-XYZ color space in output oRgb channel.

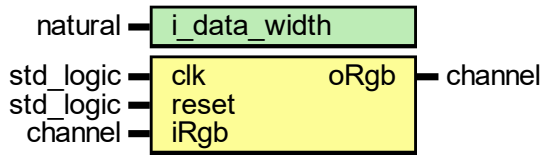


Figure 140

Cb refers to the blue-difference chroma component, and Cr refers to the red-difference chroma component.

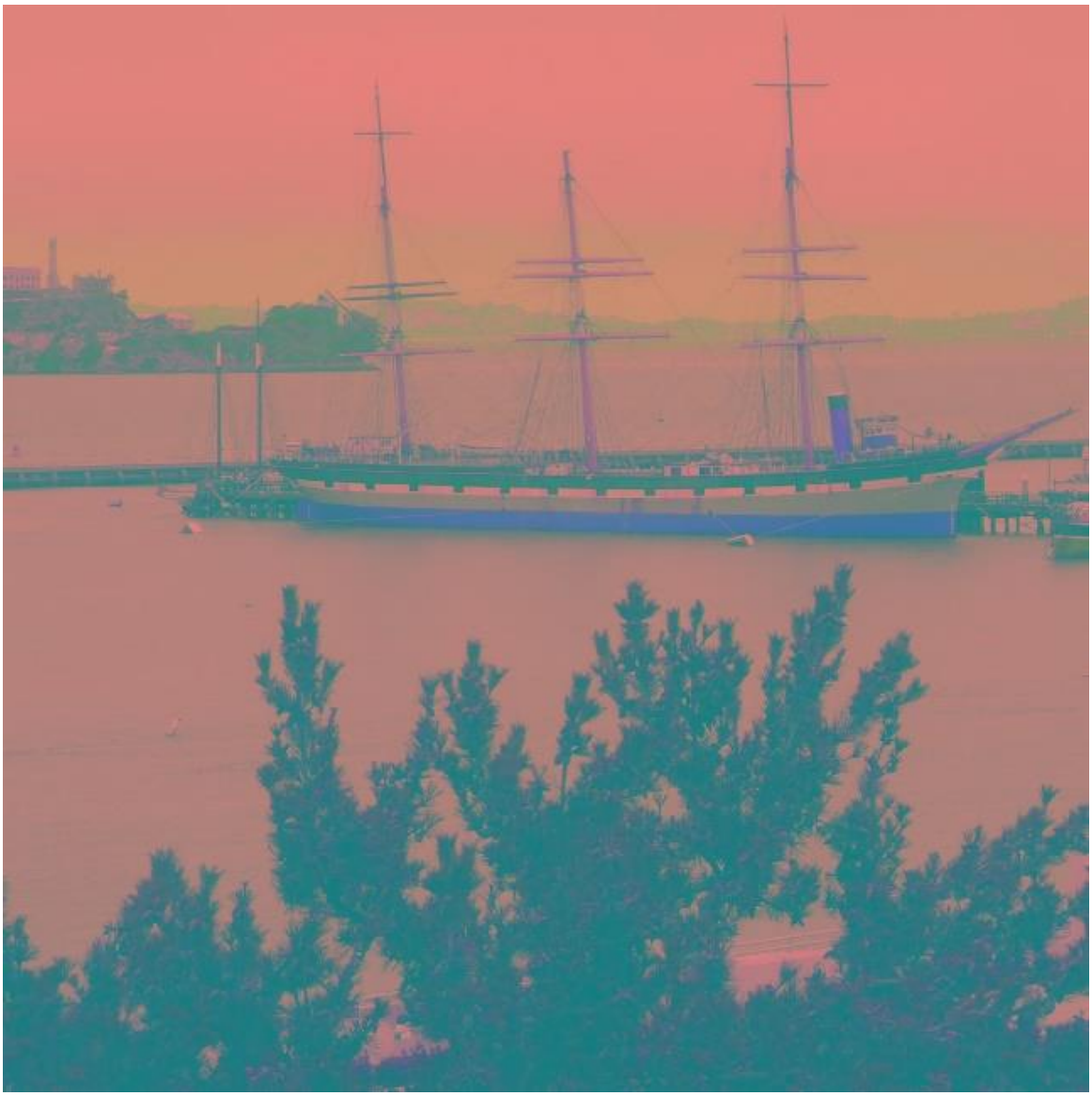


Figure 141

COLOR ADJUST MATRIX

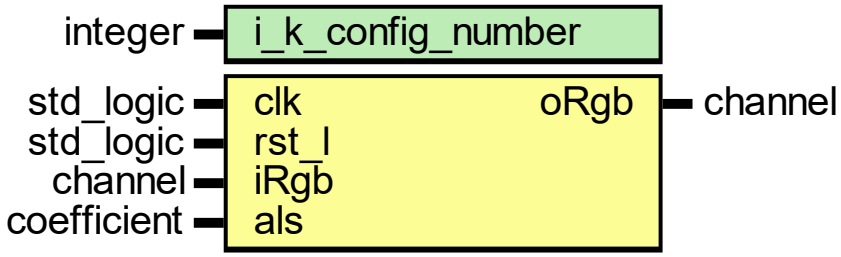
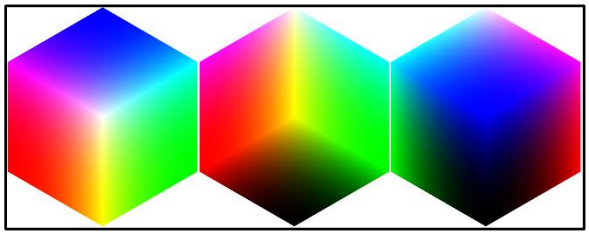


Figure 142

The objective of the color adjust matrix is to transform each pixel red, green and blue channel into improve suitable agreement in display and light sources sensitivity. The color adjust matrix can eb expressed as a 3x3 matrix as in equation below. The 'CAM' is used to denote the color adjust matrix rgb values.

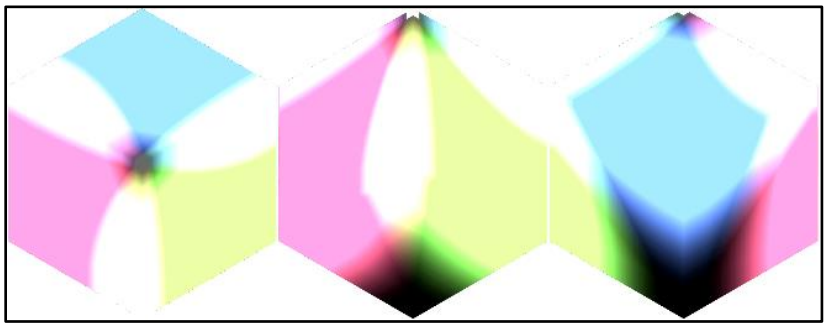
$$\begin{bmatrix} R_{CAM} \\ G_{CAM} \\ B_{CAM} \end{bmatrix} = \begin{bmatrix} K1 & K2 & K3 \\ K4 & K5 & K6 \\ K7 & K8 & K9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



The color transformations defined in this section are for each color in pixel changed into unique color as multicolor colors range changed into single color.

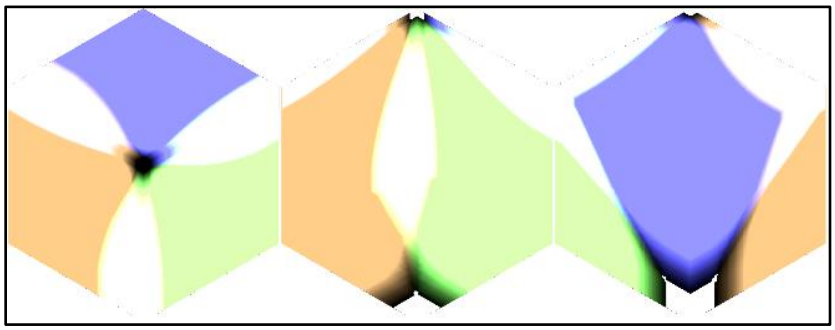
COLOR MANIPULATION INDEX # 01

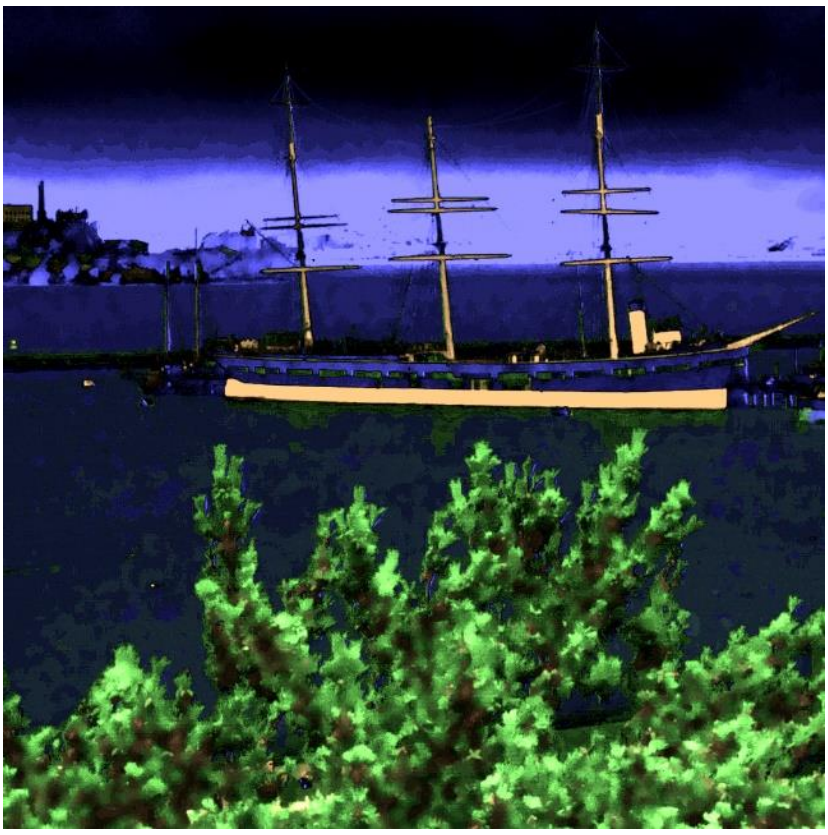
Color Adjust Matrix				
CAM-1	5.000	-3. 313	-1. 687	0.00
	-1. 687	5.000	-3. 313	0.00
	-3. 313	-1. 687	5.000	0.00
	0.00	0.00	0.00	
CAM-2	1.200	-0.435	-0.165	0.600
	-0.165	1.200	-0.435	0.600
	-0.435	-0.165	1.200	0.600
	0.600	0.600	0.600	
CAM-3	0.900	0.300	0.350	1.650
	0.350	0.900	0.300	1.650
	0.300	0.350	0.900	1.650
	1.650	1.650	1.650	



Color Manipulation Index # 02

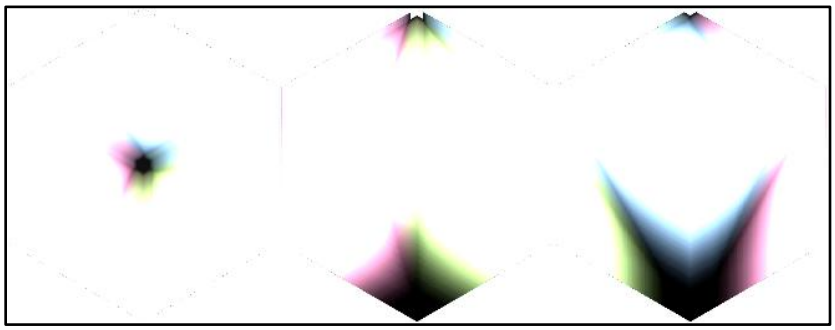
Color Adjust Matrix				
CAM-1	1.340	-0.850	-0.400	0.09
	-0.400	1.340	-0.850	0.09
	-0.850	-0.400	1.340	0.09
	0.09	0.09	0.09	
CAM-2	2.000	-0.200	-0.100	1.70
	-0.100	2.000	-0.200	1.70
	-0.200	-0.100	2.000	1.70
	1.70	1.70	1.70	
CAM-3	1.5	0.900	0.650	3.05
	0.650	1.5	0.900	3.05
	0.900	0.650	1.5	3.05
	3.05	3.05	3.05	





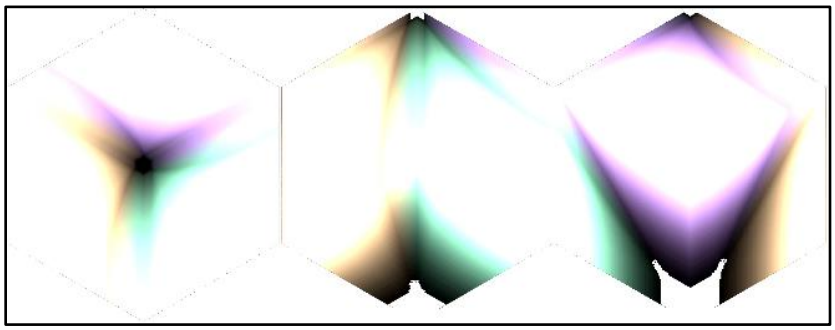
Color Manipulation Index # 03

Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



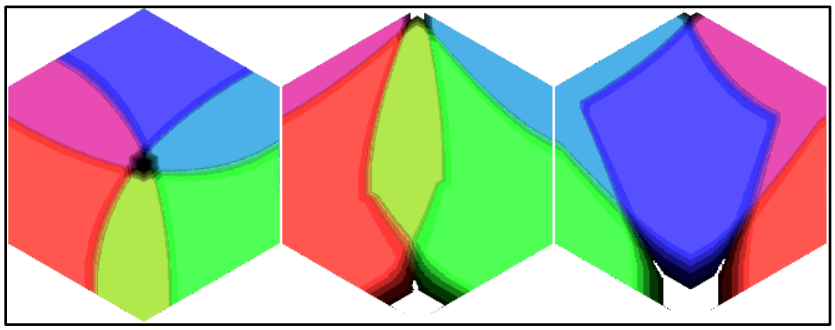
Color Manipulation Index # 04

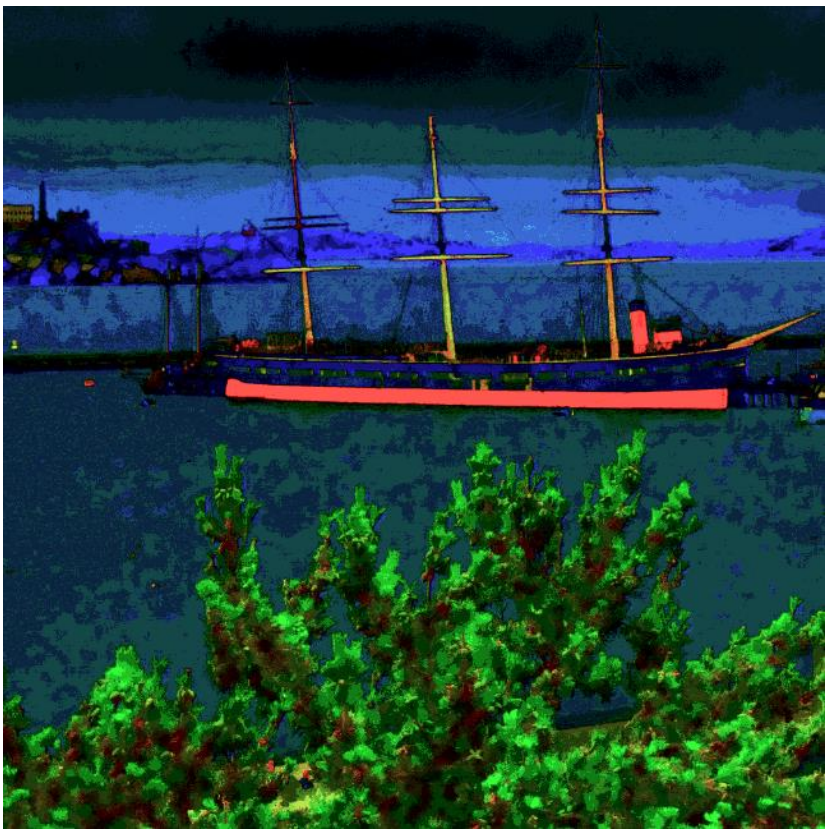
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



Color Manipulation Index # 05

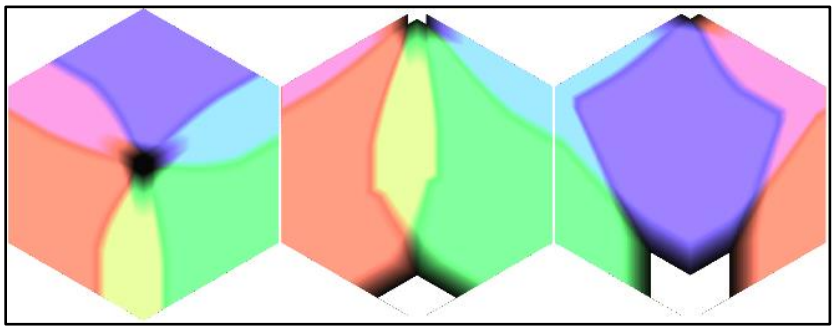
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	

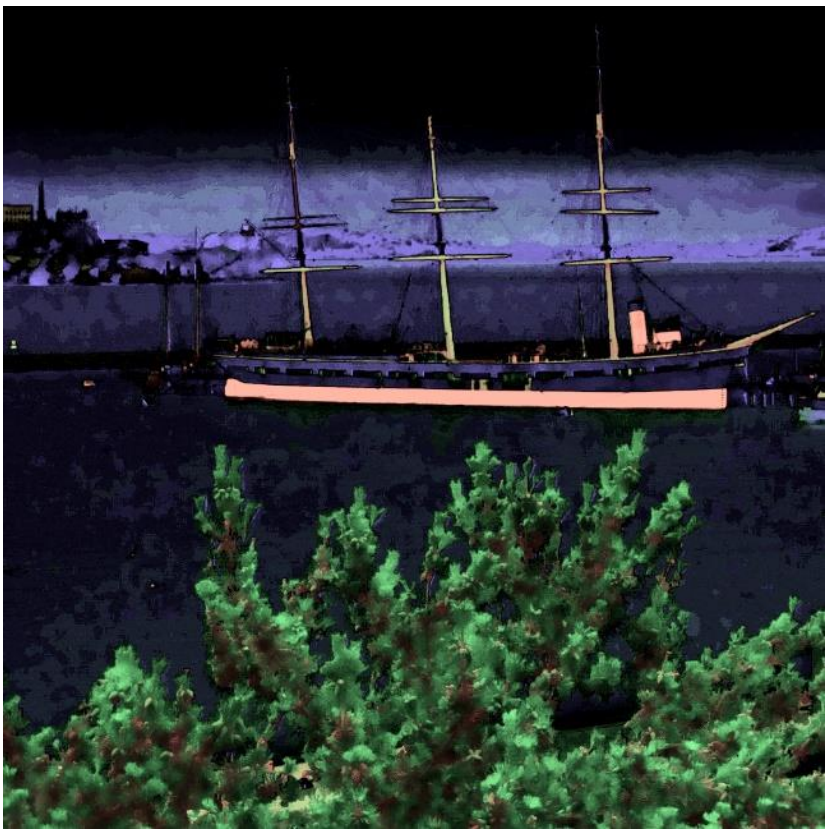




Color Manipulation Index # 06

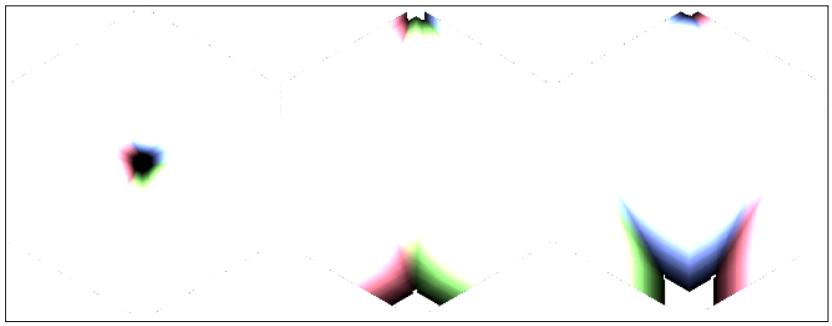
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	





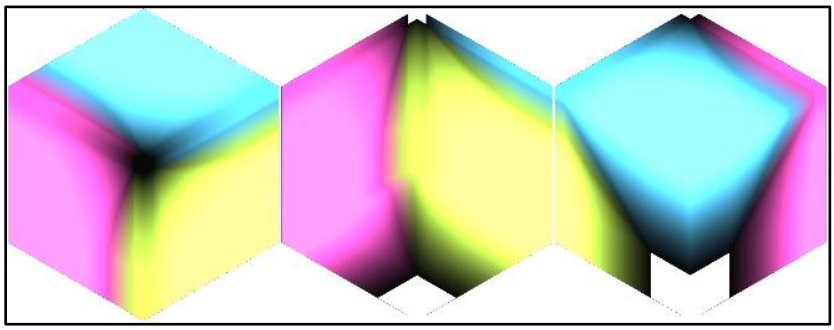
Color Manipulation Index # 07

Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



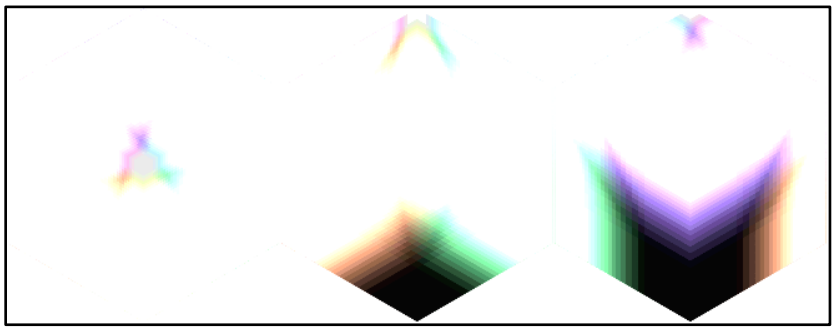
Color Manipulation Index # 08

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



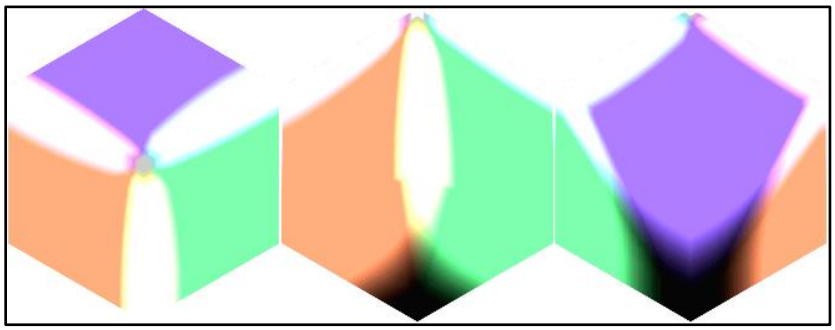
Color Manipulation Index # 09

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



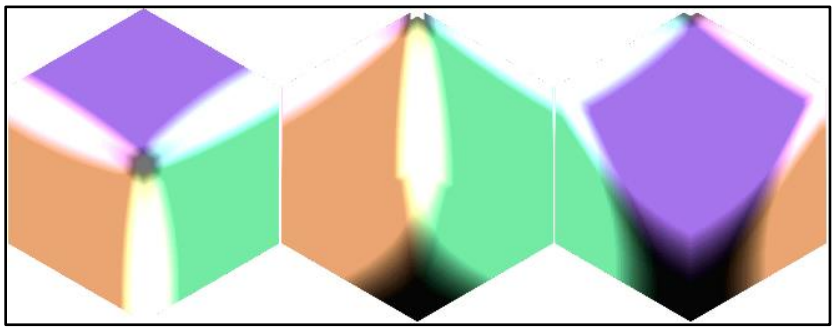
Color Manipulation Index # 10

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



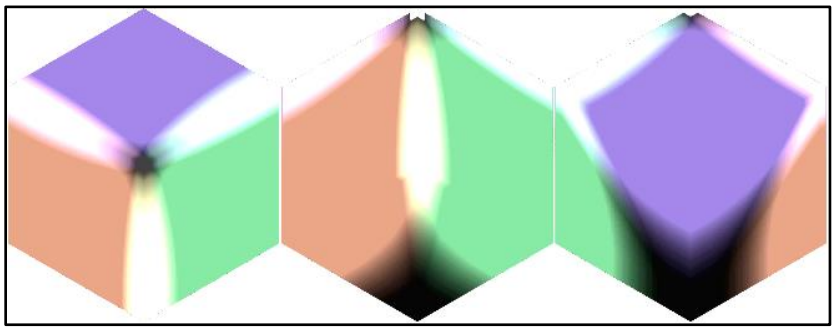
Color Manipulation Index # 11

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



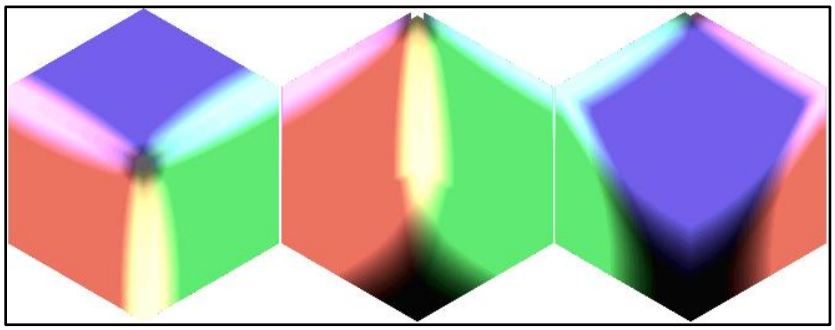
Color Manipulation Index # 12

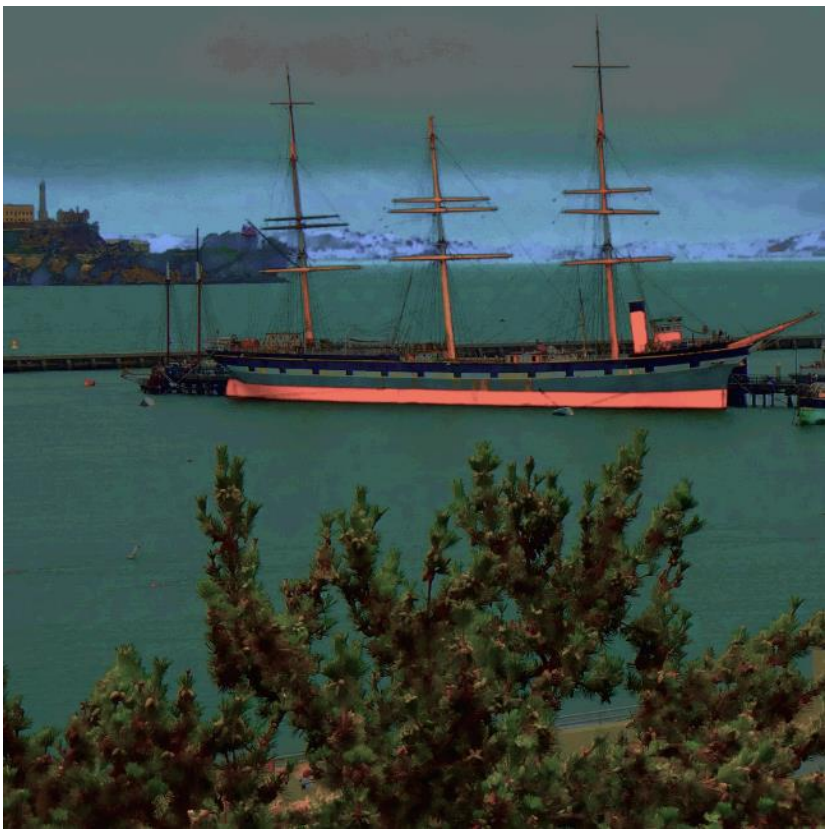
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



Color Manipulation Index # 13

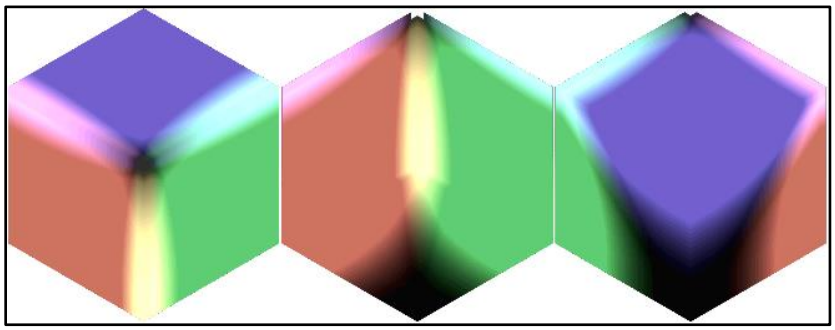
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	

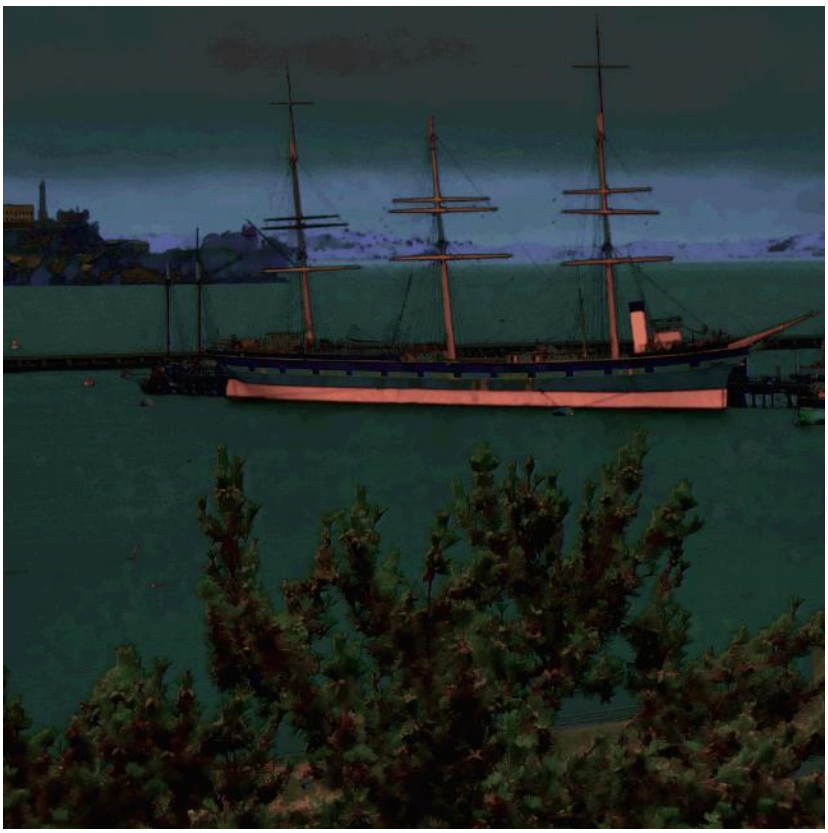




Color Manipulation Index # 14

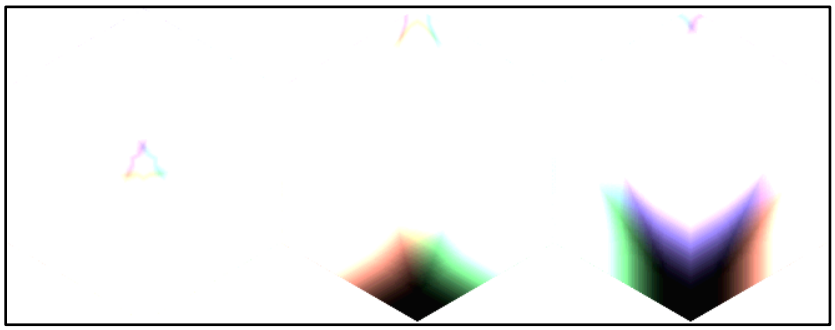
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	





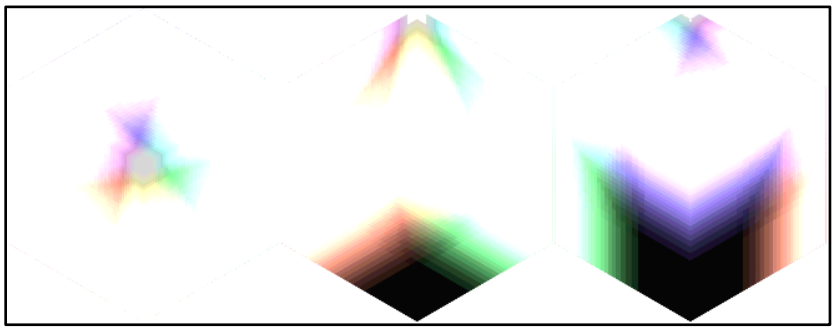
Color Manipulation Index # 15

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



Color Manipulation Index # 16

Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



COLOR K-MEANS CLUSTERING

This section of the document describes computing the k-mean rgb clustering of rgb video stream.

The implemented image segment uses K-Mean clustering algorithm for FPGA Devices, and it has been designed with a standard Xilinx AXI4 streaming interface, so that it can be inserted as module ip within any image processing pipeline.

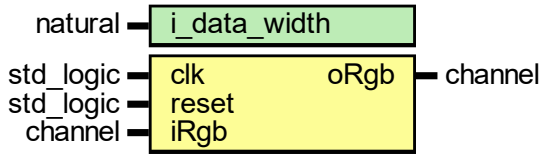


Figure 143

This module takes a stream of camera data in pixel pipeline format. This stream must be presented to the inputs data (iRgb.red, iRgb.green, iRgb.blue) and control signals (iRgb.valid, iRgb.eol, iRgb.eof, iRgb.sof). The result of this module steam K-Mean rgb cluster color space in output oRgb channel.

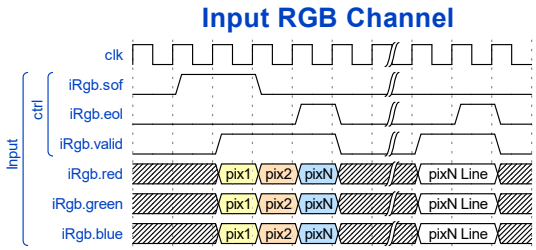


Figure 144

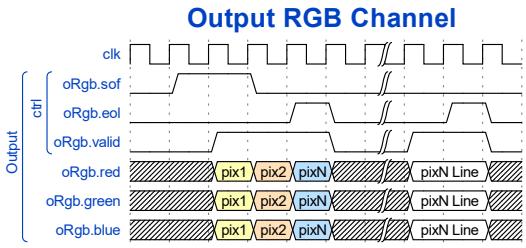
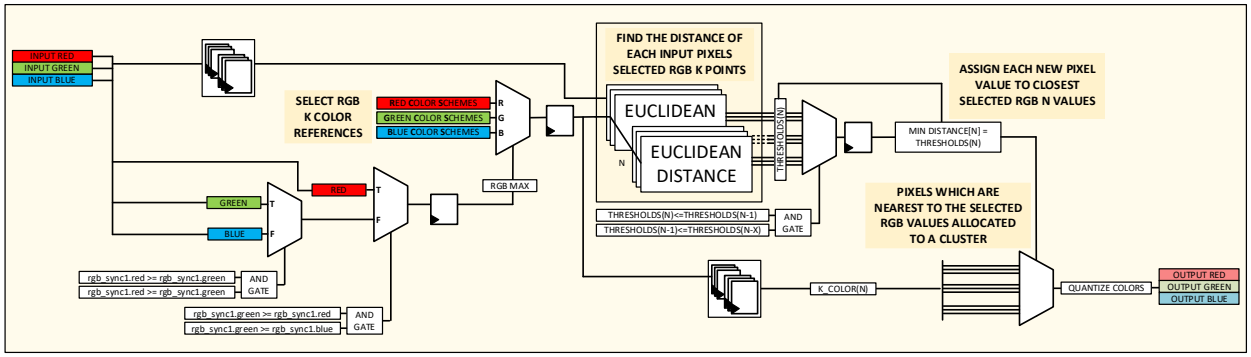


Figure 145

This module is synthesized and implemented using Vivado 2022.1 for KRIA KV260 board and verified using ModelSim 2020 edition simulator.

The Functional block diagram of the implemented rgb data path to n clustered conversion is shown in Figure below.



In this module, K mean-based color quantization algorithm is applied to rgb input stream pixels.

- Select rgb k color references for k points to decide number of clusters.
- Find the distance of each input pixels with selected rgb k points.
- Assign each new pixel value to closest selected rgb n values using Euclidean distance.
- Pixels which are nearest to the selected rgb values allocated to a cluster.

Euclidean distance is calculated between original image pixel Red_1, $Green_1$ and $Blue_1$ and reference pixel color schemes.

$$D = \sqrt{(RED_2 - RED_1)^2 + (GREEN_2 - GREEN_1)^2 + (BLUE_2 - BLUE_1)^2}$$

Rgb Image of video frame consist of 0 to 255 values per rgb channel which gives 256*256*256 colors, and the goal is to use color k mean cluster to set number of clusters. Minimum distance is the final candidate for being closest to the source rgb color. If an RGB image color depth of 24 bits which is 16 million of colors, after K-mean clustering with value n, then image is converted to a version of n colors.

The codebook created for K-Means is called the color palette or reference color scheme.

K-mean cluster module convert 16 million of rgb colors into n color version. The module has clock and reset ports. Port iRGB and oRGB consist of red, green, and blue rgb channels with valid signal.

Ports	Description
clk	Reference clock for input and output data stream.
reset	Specifies module asynchronous active low reset.
iRgb.red	8-bit input data. Red value.
iRgb.green	8-bit input data. Green value.
iRgb.blue	8-bit input data. Blue value.
iRgb.valid	Input data valid. Specifies whether the next data point has arrived for processing.
oRgb.red	8-bit output data. N clustered red value.
oRgb.green	8-bit output data. N clustered green value.
oRgb.blue	8-bit output data. N clustered blue value.
oRgb.valid	Output data valid. Control signal to indicate the validity of each pixel.

K EQUALTO 6 REFERENCE COLOR SCHEMES

Generated image below is the result of k-mean clustering using 6 colors references of palette schemes.

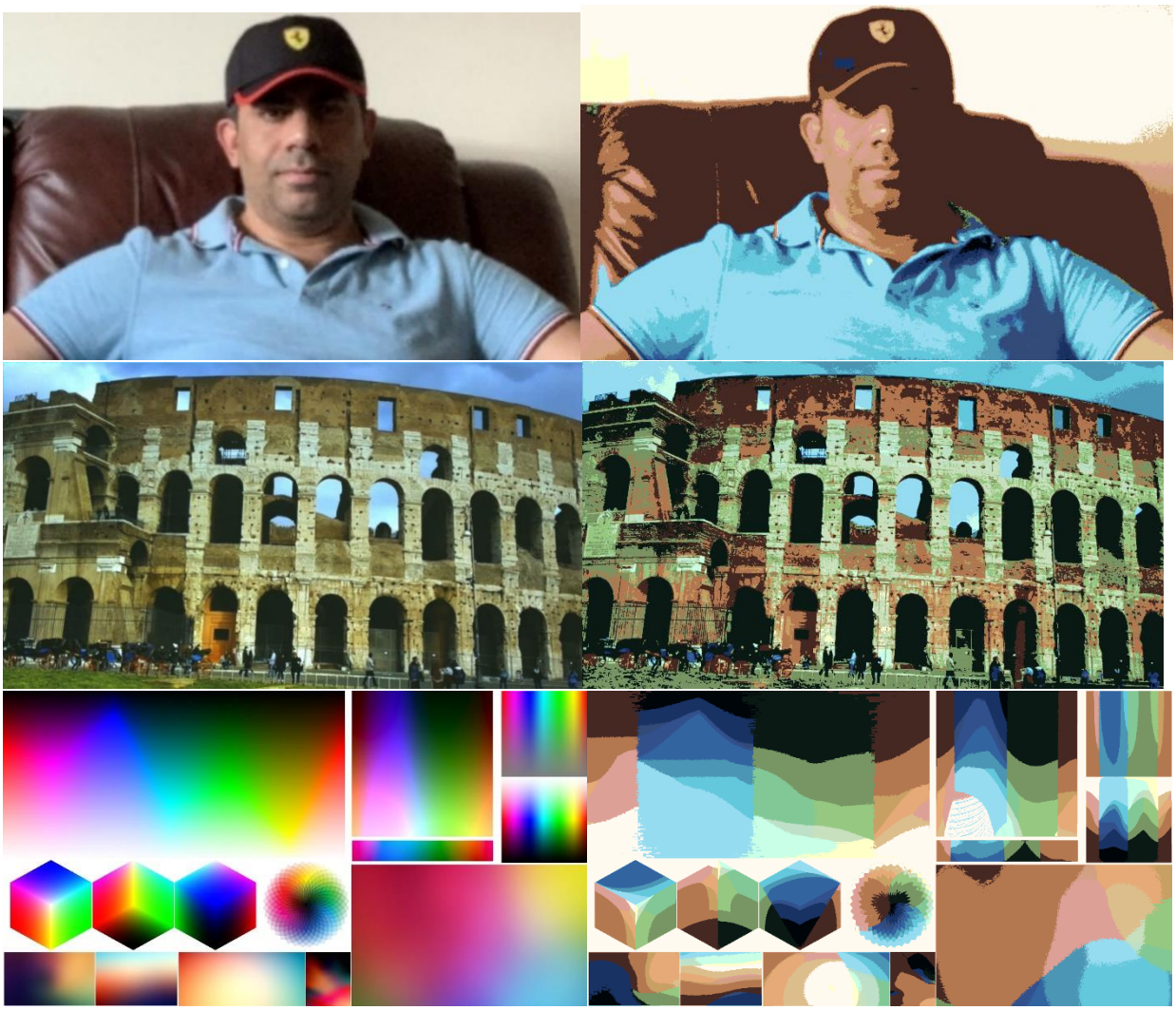
#	Red	Green	Blue
1	230	170	120
2	70	40	35
3	150	200	130
4	20	25	10
5	75	150	180
6	15	30	60

K EQUALTO 9 REFERENCE COLOR SCHEMES

In this reference, k parameter set to k = 9, where k is the number of clusters. K-mean color quantization quantizes input image to number of colors into 9 clusters from 9 refences of color schemes.

K EQUALTO 24 REFERENCE COLOR SCHEMES

In this reference, k parameter set to k = 24, where k is the number of clusters. K-mean color quantization quantizes input image to number of colors into 24 clusters from 24 refences of color schemes.



K EQUALTO 51 REFERENCE COLOR SCHEMES

In this reference, k parameter set to k = 51, where k is the number of clusters. K-mean color quantization quantizes input image to number of colors into 51 clusters from 51 refences of color schemes.

Image below is generated using 51 reference color schemes.

#	RED	GREEN	BLUE	#	RED	GREEN	BLUE	#	RED	GREEN	BLUE	#	RED	GREEN	BLUE	#	RED	GREEN	BLUE
1	255	250	240	11	150	80	60	21	255	255	200	31	50	75	30	41	100	200	20
2	230	200	150	12	140	80	60	22	200	255	230	32	30	75	50	42	90	180	0
3	220	190	140	13	120	80	55	23	200	224	190	33	40	50	20	43	75	150	80
4	240	180	120	14	100	80	50	24	190	224	200	34	20	50	40	44	50	100	60
5	230	170	120	15	80	60	40	25	150	200	130	35	10	25	20	45	50	80	40
6	220	160	150	16	70	40	35	26	130	200	150	36	20	25	10	46	50	75	20
7	210	160	130	17	60	40	30	27	120	150	100	37	15	10	0	47	25	50	0

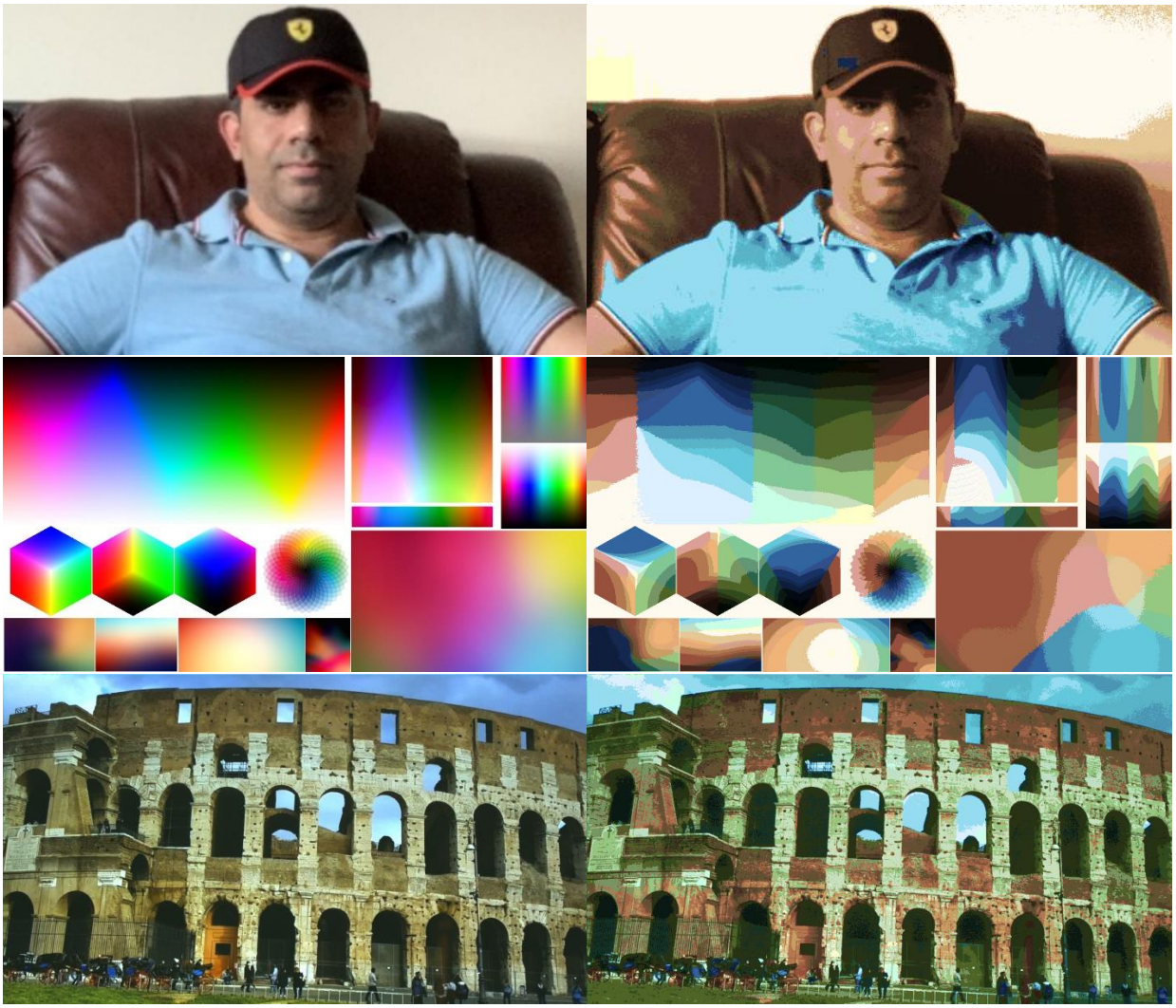
8	200	160	100
9	180	120	80
10	160	100	70

18	50	30	20
19	40	20	15
20	25	15	10

28	100	150	120
29	80	100	40
30	40	100	80

38	0	10	5
39	220	240	55
40	150	220	40

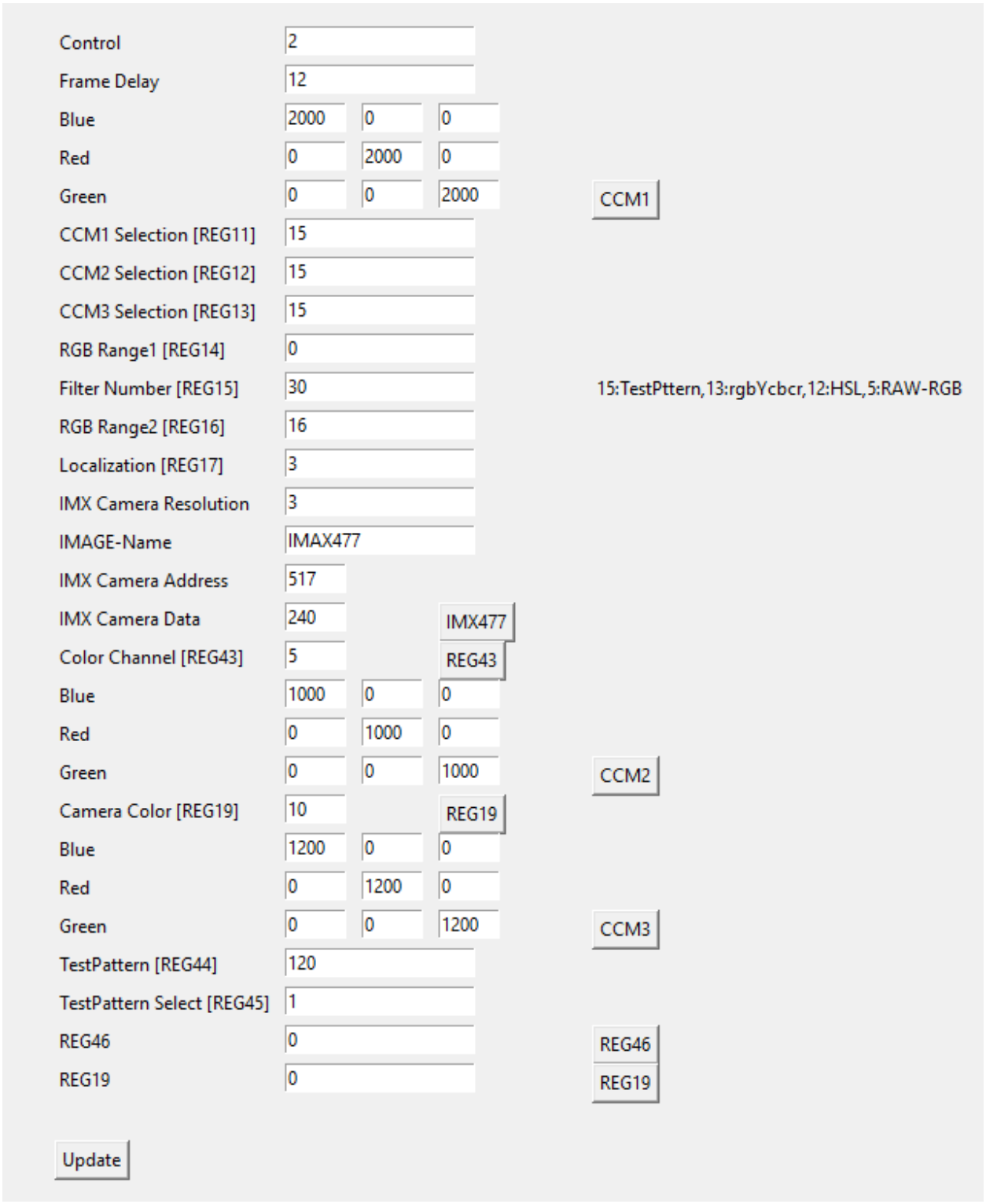
48	20	40	80
49	15	30	60
50	5	10	20
51	0	5	10



IMX477 CAMERA: K EQUALTO 90 REFERENCE COLOR SCHEMES

Figure 29. (a) Original image; (b) results obtained using the K-Mean Clustering.

RGB 170-85 CLUSTER VALUES					RGB 85-0 CLUSTER VALUES					RGB 170-255 CLUSTER VALUES				
001	170	160	150	IX_01	001	85	0	0	IX_01	061	240	200	0	IX_01
002	170	170	80	IX_02	002	85	85	0	IX_02	062	240	180	0	IX_02
003	170	170	0	IX_03	003	85	45	0	IX_03	003	230	180	0	IX_03
004	170	130	0	IX_04	004	85	30	0	IX_04	004	230	170	0	IX_04
005	170	70	0	IX_05	005	85	10	0	IX_05	005	230	160	0	IX_05
006	170	0	0	IX_06	006	85	0	0	IX_06	006	230	180	0	IX_06
007	150	150	50	IX_07	007	85	0	0	IX_07	007	200	160	0	IX_07
008	150	150	0	IX_08	008	85	0	0	IX_08	008	200	150	0	IX_08
009	150	100	0	IX_09	009	85	85	0	IX_09	009	200	140	0	IX_09
010	150	50	0	IX_10	010	85	85	0	IX_10	010	200	130	0	IX_10
011	150	0	0	IX_11	011	85	85	0	IX_11	011	200	100	0	IX_11
012	130	130	40	IX_12	012	85	85	0	IX_12	012	160	160	0	IX_12
013	130	130	0	IX_13	013	85	85	0	IX_13	013	160	777	0	IX_13
014	130	80	0	IX_14	014	70	70	0	IX_14	014	160	110	50	IX_14
015	130	40	0	IX_15	015	70	50	15	IX_15	015	160	100	40	IX_15
016	130	0	0	IX_16	016	70	30	15	IX_16	016	160	90	40	IX_16
017	120	120	0	IX_17	017	70	10	0	IX_17	017	160	90	10	IX_17
018	120	90	0	IX_18	018	60	60	0	IX_18	018	160	90	20	IX_18
019	120	60	0	IX_19	019	60	30	20	IX_19	019	140	140	0	IX_19
020	120	30	0	IX_20	020	60	20	10	IX_20	020	140	100	40	IX_20
021	120	0	0	IX_21	021	60	10	0	IX_21	021	140	80	40	IX_21
022	100	100	0	IX_22	022	40	40	0	IX_22	022	140	60	0	IX_22
023	100	75	0	IX_23	023	40	20	10	IX_23	023	120	120	0	IX_23
024	100	50	0	IX_24	024	40	10	5	IX_24	024	120	50	0	IX_24
025	100	25	0	IX_25	025	40	10	0	IX_25	025	120	80	0	IX_25
026	100	0	0	IX_26	026	30	30	0	IX_26	026	120	70	0	IX_26
027	90	90	0	IX_27	027	30	20	15	IX_27	027	100	100	0	IX_27
028	90	60	0	IX_28	028	30	15	0	IX_28	028	100	75	0	IX_28
029	90	45	0	IX_29	029	30	10	0	IX_29	029	100	50	0	IX_29
030	90	0	0	IX_30	030	20	10	0	IX_30	030	100	0	0	IX_30


Control
2
Frame Delay
12
Blue
2000
0
0
Red
0
2000
0
Green
0
0
2000
CCM1
CCM1 Selection [REG11]
15
CCM2 Selection [REG12]
15
CCM3 Selection [REG13]
15
RGB Range1 [REG14]
0
Filter Number [REG15]
30
15:TestPttern,13:rgbYcbcr,12:HSL,5:RAW-RGB
RGB Range2 [REG16]
16
Localization [REG17]
3
IMX Camera Resolution
3
IMAGE-Name
IMAX477
IMX Camera Address
517
IMX Camera Data
240
IMX477
Color Channel [REG43]
5
REG43
Blue
1000
0
0
Red
0
1000
0
Green
0
0
1000
CCM2
Camera Color [REG19]
10
REG19
Blue
1200
0
0
Red
0
1200
0
Green
0
0
1200
CCM3
TestPattern [REG44]
120
TestPattern Select [REG45]
1
REG46
0
REG46
REG19
0
REG19
Update

LOCAL DYNAMIC THRESHOLD SEGMENTATION

This module takes the pixel as the center and check its neighboring pixels and compared with one by one. If the difference level is within the threshold limits than average, it and the new pixel value will be compared with each neighborhood pixels. The produced new region is uniform and difference in region would be small.

The aim is to average pixel by thresholding. Typically, in the design for this module, threshold of value 10 is selected with the pixel values in a 0-255 range. Pixel with values less than the threshold will map to average, whilst those with values greater than or equal to the threshold will map to original pixel values. Average pixel values are close to current and next values are averaged within the threshold; this mapping gives local clustering of image pixel.

Not implemented Yet: For given selected region of area below the define threshold such that there would exist a maximum value before averaging. Such those max value does not need to be part of average values. This would better result for averaging the pixels within the threshold. If all values are same than there is no max, in other words all value are constant in that region. Such implementation would be valid for at-least 3 pixels or greater.

The module rgb pixel data is available on when valid data is asserted high by the module.

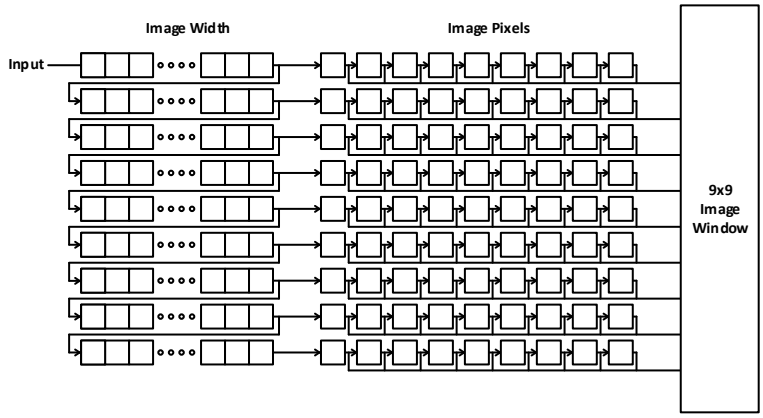


IMAGE CONTRAST AND BRIGHTNESS

CAMERA RAW DATA

This module read raw bayer filter mosaic format 12-bit data from d5m camera. External Pixel clock is from camera used to sample 1 pixel which equal to 1 pll generated master external clock. Host camera data is than stored into buffer line by line which is ready to be fetch at system clock rate. Input line and frame valid signals are used to start reading stored buffer data. Valid read is enabled when both frame and line valid signal are asserted high. Buffer size set to be the size of frame width. The read controller side reads whole frame width from the buffer. Values written to buffer run at pixel clock rate whereas buffer read side run at faster rate than pixel clock.

Buffersize is auto size supported which is controlled by input line valid from host camera and maximum is set to default value of 3071. A base configuration consisting of a single 24-bits pipeline, capable of processing 1080p HD video at 30 frames per second.

Input data from camera is color filtered which is arranged in a bayer patten. Input data is read with line-by-line transfer rate of pixel clock which later synchronize into system clock.

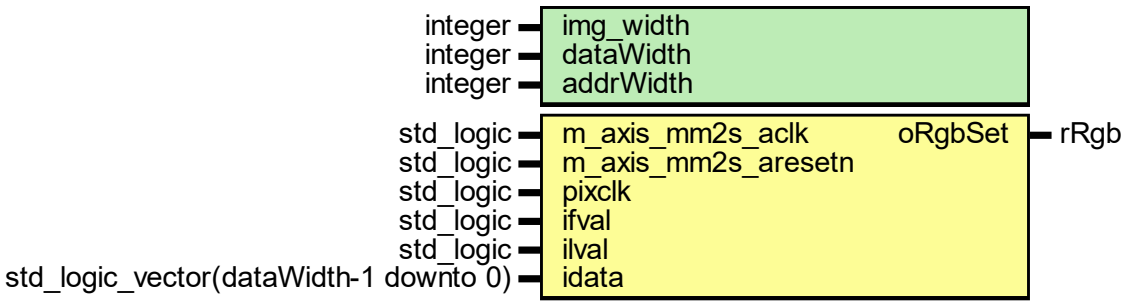


Figure 146 : camera raw data module

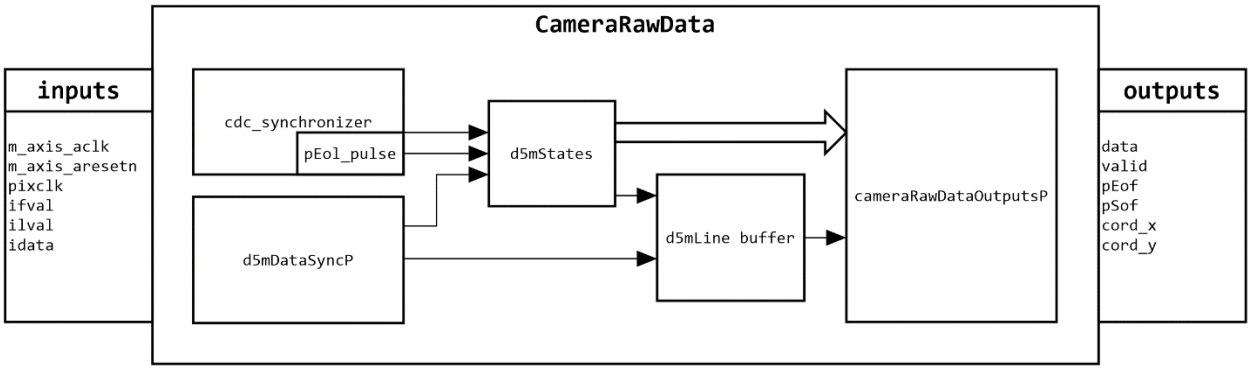


Figure 147 : General view of camera raw data flow

Host camera interface uses 12 bits parallel input data with line and frame valid control signals.

THREE TAPS DATA

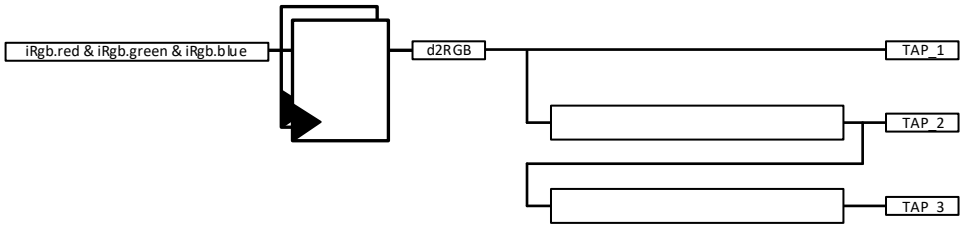


Figure 148

FOUR TAPS DATA

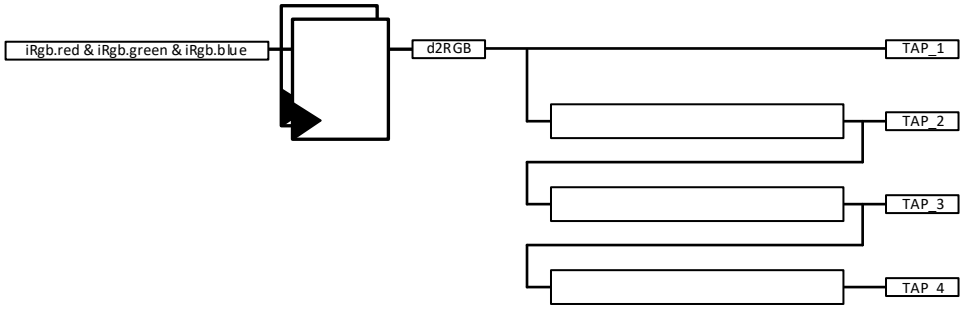


Figure 149

PIXEL COORDINATES

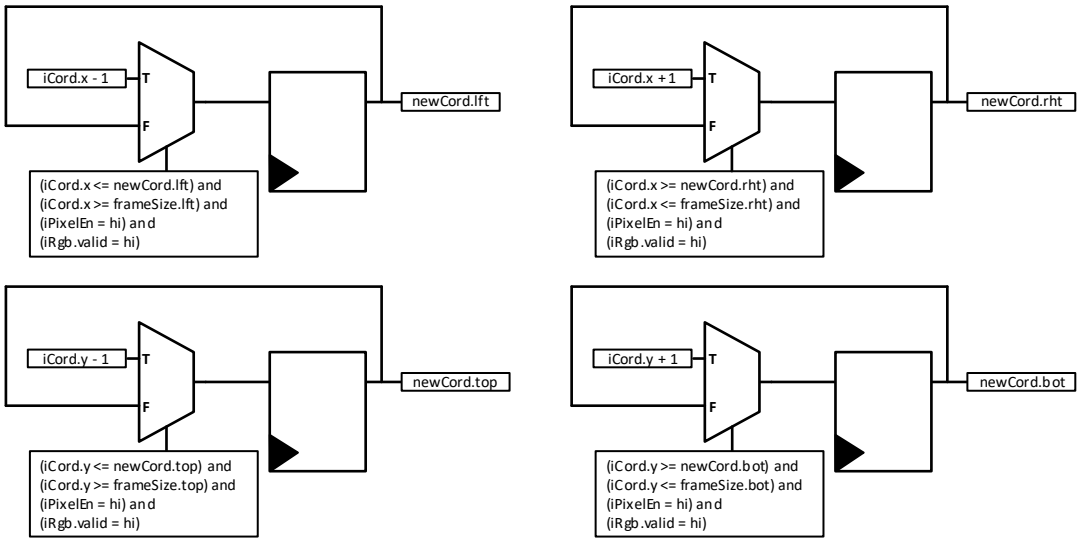


Figure 150

K1	K2	K3	K4
K5	K6	K7	K8
K9	K10	K11	K12
K13	K14	K15	K16

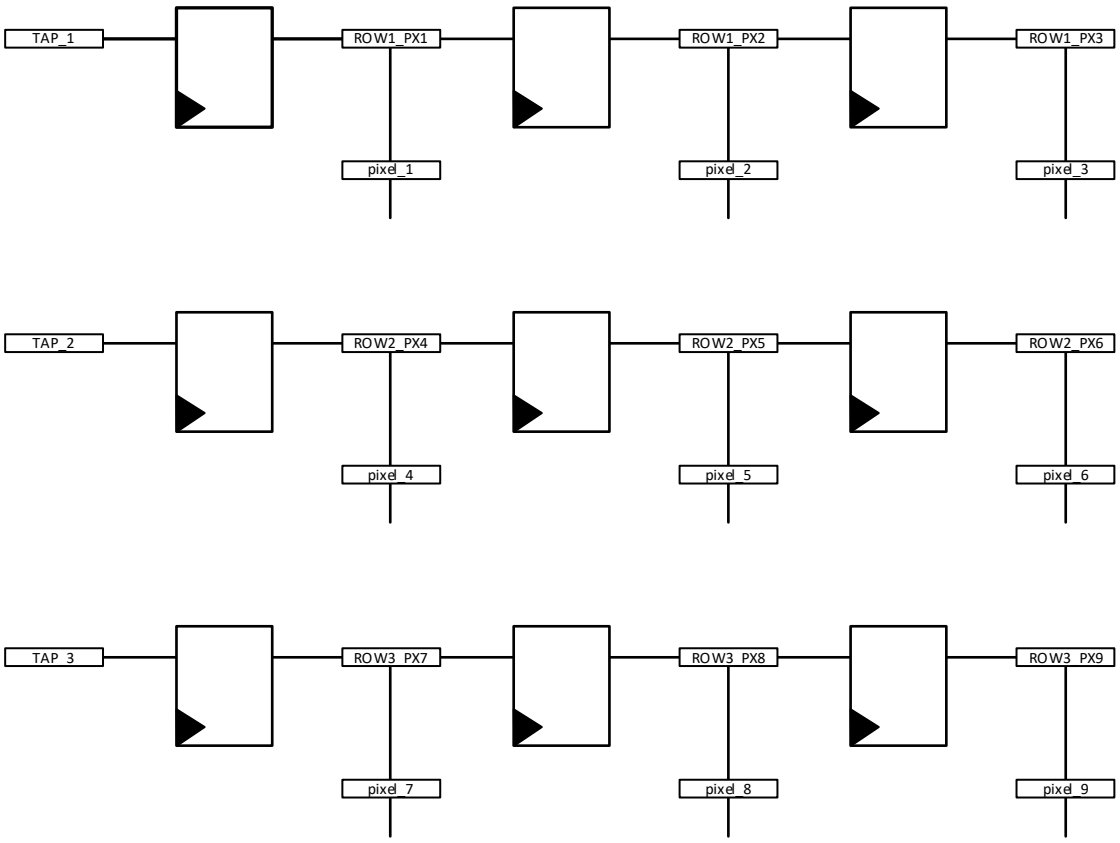
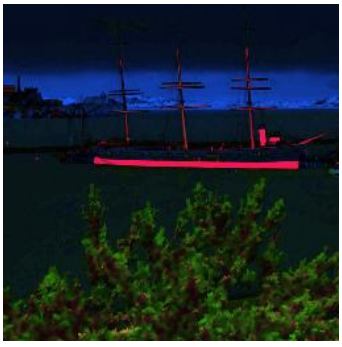


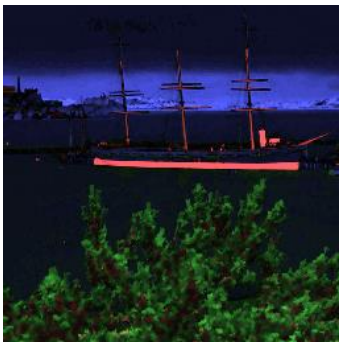
Figure 151

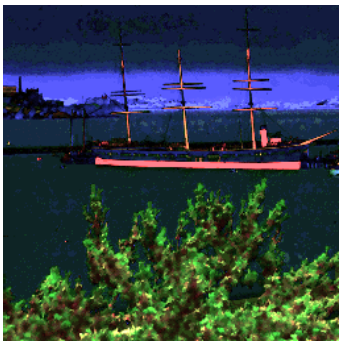
K1	K2	K3
K4	K5	K6
K7	K8	K9

Figure 152

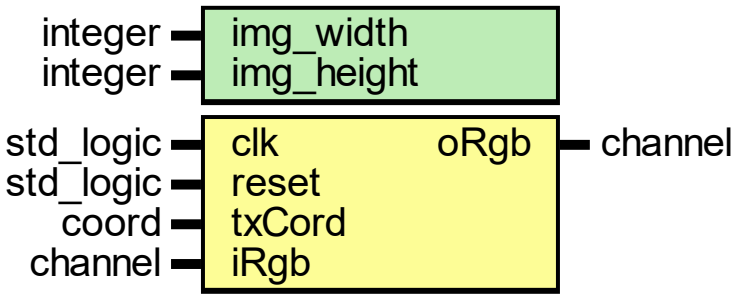
CAMERA RAW DATA







HISTOGRAM



This module assigns memory location as rgb red channel input integer between 0 to 255 which equally 8 bits to express 256 levels address. Every input value would accumulate to its location to show how many hits per level.

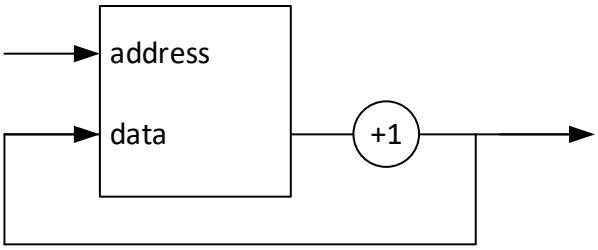
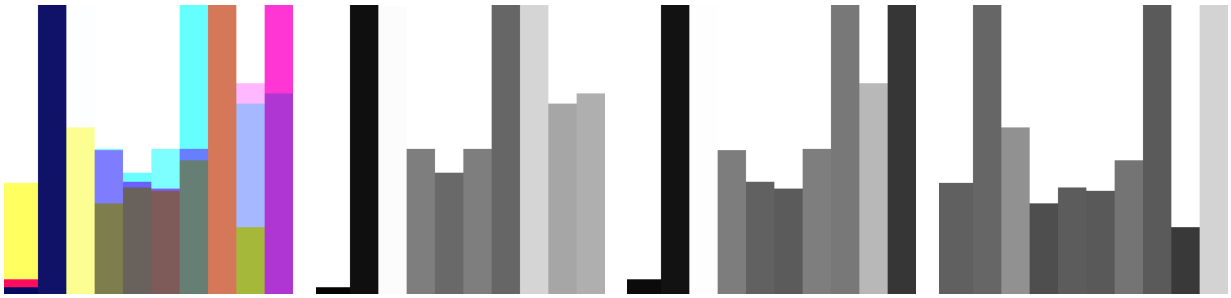


Figure 153

Histogram Equation



1ST figure shows rgb image. 2nd figure shows red channel. 3rd figure shows green channel and 4th figure shows blue channel.

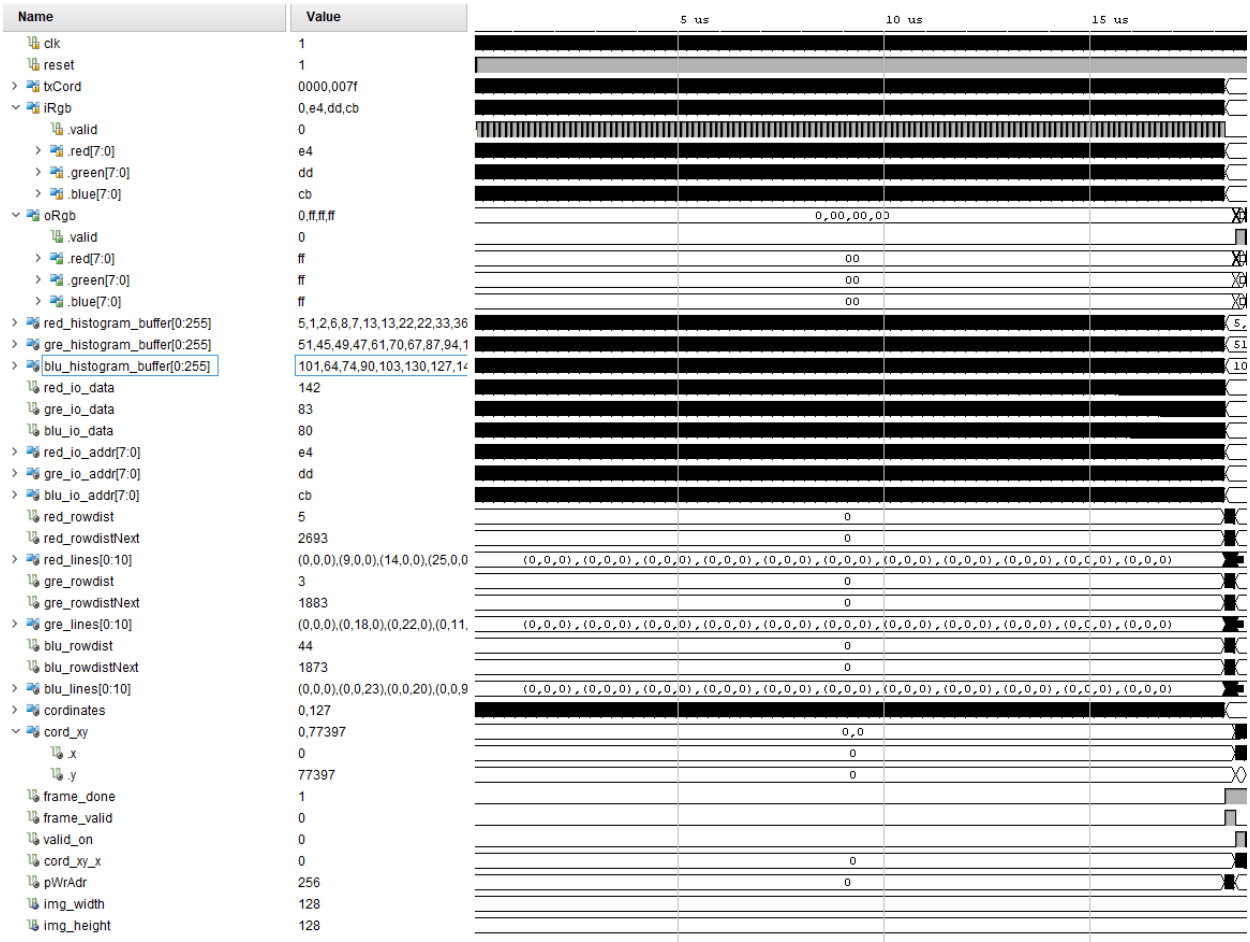


Figure 154

TESTBENCH

For filter modules verification vhdl testbench is being used to generate image. Approach used to generate stimulus and checking response by using single process testbench. Testbench include a process which read image file to apply rgb stimulus to design filter section and then wait until end of frame to generate valid output image bmp file.

TESTBENCH COMPONENTS/OBJECTS

D5M verification testbench is created through configuration, the factory and the phase build process which allows stimulus and randomization generation. The testbench describes image filters of design, where source of input stimuli, or test patterns are applied and monitor of output results inform of output generated images. Testbench requires the user to manually check filtered output image and check pass/fail test pattern testing.

UVM testbench build from classes. Test has environment, the environment has d5m agent.

Flow of testbench is to connect DUT and testbench, configure d5m agent, generate stimulus.

In the filter's generic package, a common verification scenario for vfp tests is to initialize set of parameters by setting to define enable variable in frame enable package. This type of initialization can be done at terminal during compilation unit scope. In table shows available define variable with available size for image testing.

`define	size	description
rgb	**0**=64x64,	
sharp	**1**=128x125,	
blur	**2**=255x255,	
embross	**3**=1920x1080	
hsl		
hsv		
cgain		
sobel		

Set of parameters for image filter type and their size are assigned for current test values as shown in table 2 via defined macro in frame enable package according to table 1.

parameter	value	description
F_CGA	1/0	Enable '1' Cgain color adjust for test.
F_SHP	1/0	Enable '1' Sharp Filter for test.
F_BLU	1/0	Enable '1' Blure Filter for test.
F_HSL	1/0	Enable '1' HSL color space for test.
F_HSV	1/0	Enable '1' HSV color space for test.
F_RGB	1/0	Enable '1' RGB color space for test.
F_SOB	1/0	Enable '1' Sobel filter for test.

F_EMB	1/0	Enable ‘1’ Embross Filter for test.

For axi4-lite module registers in rtl, following parameters offset for base address are set for axi4 lite transections as shown table below.

Register Name	Offet	Description
initAddr	8'h00	
oRgbOsharp	8'h00	
oEdgeType	8'h04	
filter_id	8'h08	
aBusSelect	8'h0C	
threshold	8'h10	
videoChannel	8'h14	
dChannel	8'h18	
cChannel	8'h1C	
kls_k1	8'h20	
kls_k2	8'h24	
kls_k3	8'h28	
kls_k4	8'h2C	
kls_k5	8'h30	
kls_k6	8'h34	
kls_k7	8'h38	
kls_k8	8'h3C	
kls_k9	8'h40	
kls_config	8'h44	
als_k1	8'h54	
als_k2	8'h58	
als_k3	8'h5C	
als_k4	8'h60	
als_k5	8'h64	
als_k6	8'h68	
als_k7	8'h6C	
als_k8	8'h70	
als_k9	8'h74	
als_config	8'h78	
pReg_pointInterest	8'h7C	
pReg_deltaConfig	8'h80	
pReg_cpuAckGoAgain	8'h84	
pReg_cpuWgridLock	8'h88	
pReg_cpuAckoffFrame	8'h8C	
pReg_fifoReadAddress	8'h90	
pReg_clearFifoData	8'h94	
rgbCoord_rl	8'hC8	
rgbCoord_rh	8'hCC	
rgbCoord_gl	8'hD0	

rgbCoord_gh	8'hD4	
rgbCoord_bl	8'hD8	
rgbCoord_bh	8'hDC	
oLumTh	8'hE0	
oHsvPerCh	8'hE4	
oYccPerCh	8'hE8	

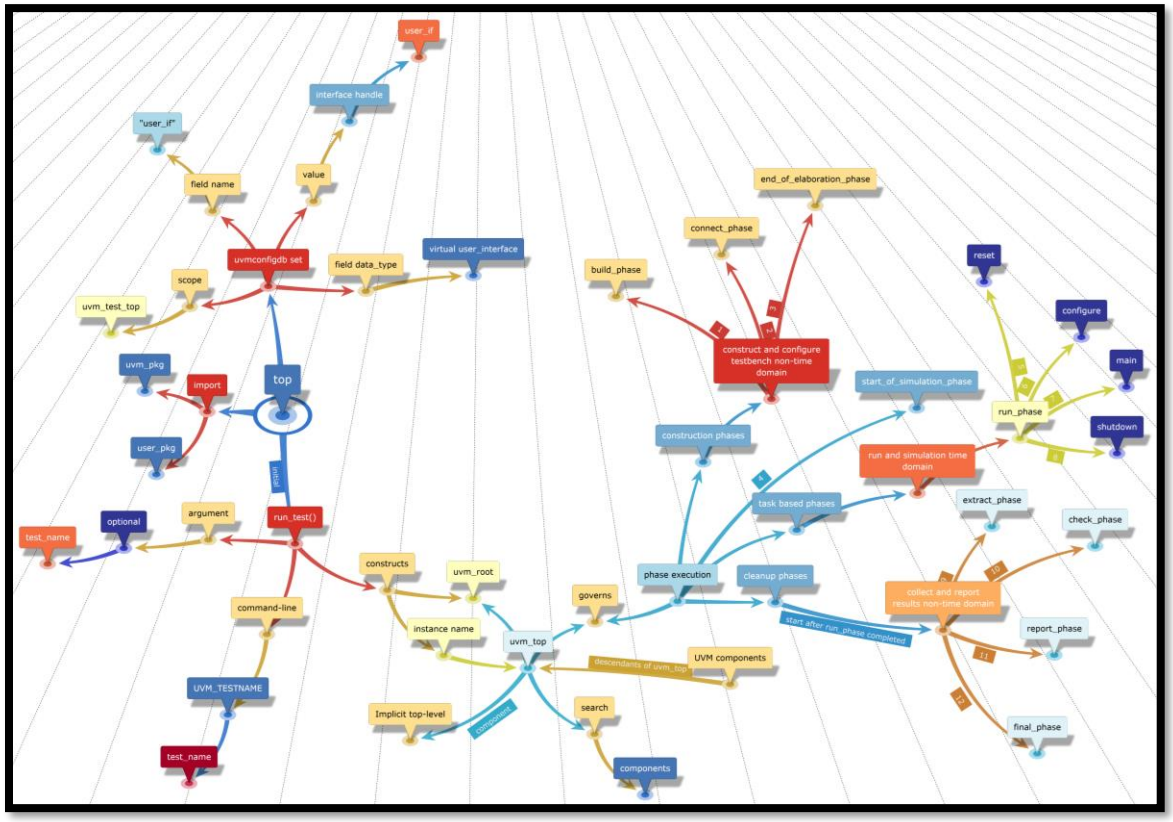


Figure 155

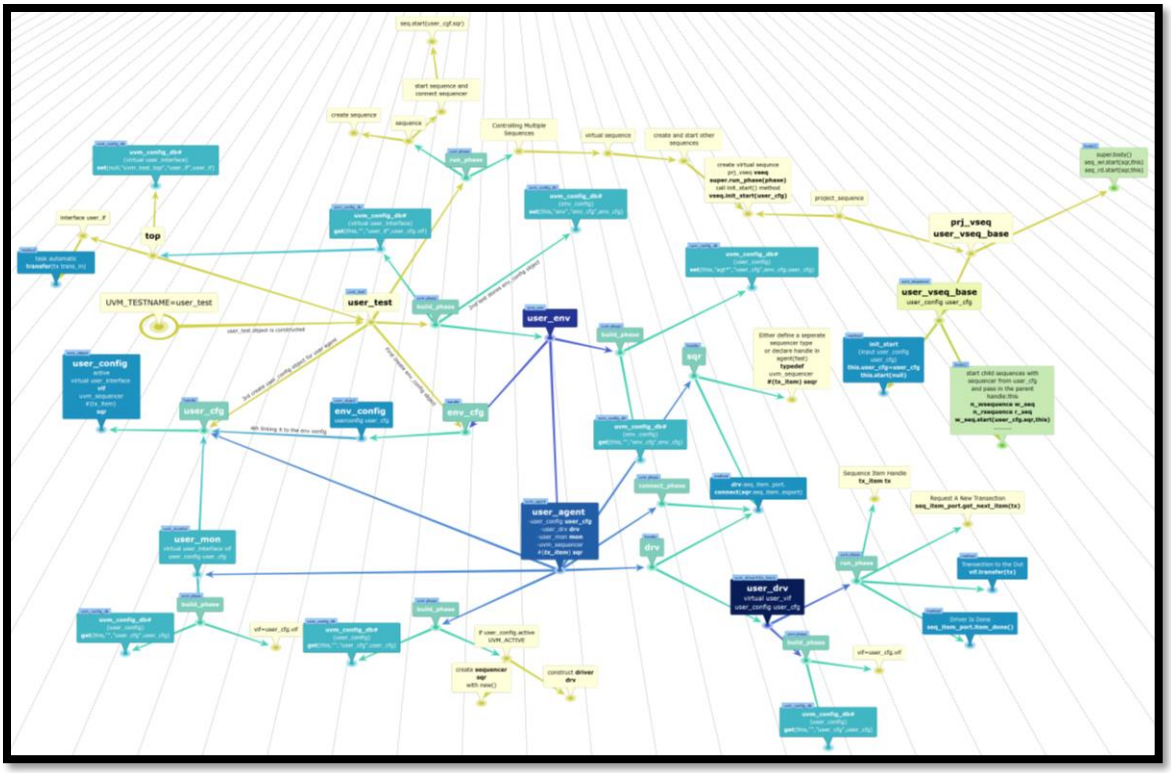

top
UVM_TESTNAME=user_test
user_test
user_env
user_config
user_cfg
env_config
env_cfg
user_agent
user_mon
user_drv
sqr
drv
prj_vseq
user_vseq_base
run_phase
build_phase
connect_phase

Figure 156

DUT CONNECTIONS

Input protocol interface

Output protocol interface

SEQUENCE ITEM

SEQUENCE

SEQUENCER

DRIVER

This class which extended from uvm driver pull data items generated by a sequencer and drive it to the DUT. In run phase, methods are used for reading and writing operation to dut through dut interface handle.

Axi4lite communication flow across 5 parallel channels. Read transection phases split into 2 phases. Request on read address channel and response on read data channel. Write transection split into 3 phases: address, data and response with the status. Multiple read and write transaction are concurrent. Write address transfer can come before, during, or after the write data. Signal for read address channels start with AR* and read data signals start with R*. Write address signal start with AW*, write data signal name start with W* and write response start with B*. Axi4lite has flow control on each channel to control the rate the information is exchange such as address, data and response. The source generates valid to indicate information is available. Destination generates ready signal to accept the information. The source assert valid first and wait for destination to be ready and then transfer the info. Or destination could go first, asserting ready to receive and wait for the source. When source is ready, it asserts valid and transfer the info. The transfer only occurs when both valid and ready are high.

Basic overview:

1. Declare the virtual interface.
2. Get the interface handle using get config_db.
3. Add the get config_db in the build_phase.
4. Add driving logic. get the seq_item and drive to dut signals.

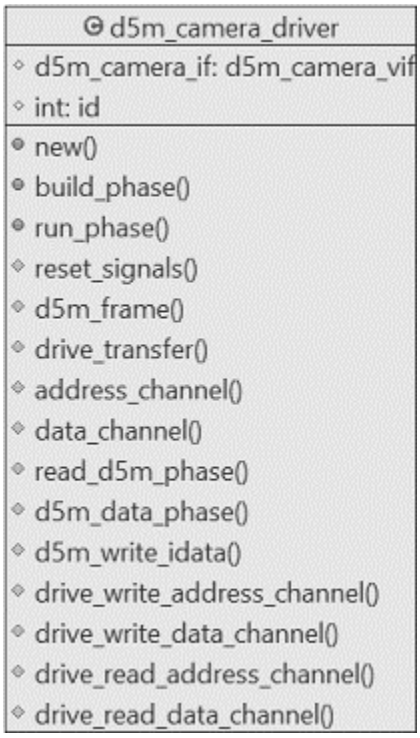


Figure 157

DECLARATION:

This uvm driver class is derived from uvm component.

```
class d5m_camera_driver extends uvm_driver #(d5m_trans);
```

DATA MEMEBERS

```
protected virtual d5m_camera_if d5m_camera_vif;
protected int     id;
```

NEW CONSTRUCT

For Each component, the constructor must execute and complete in order to bring the component into existence. Therefore, new () must run before build() or any other subsequent phase can execute.

```
function new (string name, uvm_component parent);
    super.new(name, parent);
endfunction: new
```

BUILD_PHASE

Build method run top-down and the rest of the phases run bottom-up. In this phase, config the dut interface handle through get method. The uvm_config_db parameterized class provides a convenience interface on top of uvm_resource_db is used to for reading resource database. The uvm_config_db is derived from uvm_resource_db class. Get value of field_name "`d5m_camera_vif`", using component cntxt "this" as starting search point.

```
function void build_phase (uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db#(virtual d5m_camera_if)::get
            (this, "", "d5m_camera_vif", d5m_camera_vif))
    `uvm_fatal("NOVIF", {"virtual interface must be set for:
        ",get_full_name(), ".d5m_camera_vif"});
endfunction: build_phase
```

RUN_PHASE

In this method, fork join constructs are used to separate threads that drive each of the channels.

```
    virtual task run_phase (uvm_phase phase);
        fork
            reset_signals();
            d5m_frame();
        join
    endtask: run_phase
```

RESET SIGNALS

Reset the dut axi4 lite and d5m_cam_mod input signals when system reset is asserted from low to high.

```
virtual protected task reset_signals();
    forever begin
    @(posedge d5m_camera_vif.ARESETN);
        d5m_camera_vif.axi4.AWADDR        <=  8'h0;
        d5m_camera_vif.axi4.AWPROT        <=  3'h0;
        d5m_camera_vif.axi4.AWVALID       <=  1'b0;
```

```
        d5m_camera_vif.axi4.WDATA         <= 32'h0;
        d5m_camera_vif.axi4.WSTRB         <=  4'h0;
        d5m_camera_vif.axi4.WVALID        <=  1'b0;
        d5m_camera_vif.axi4.BREADY        <=  1'b0;
        d5m_camera_vif.axi4.ARADDR        <=  8'h0;
        d5m_camera_vif.axi4.ARPROT        <=  3'h0;
        d5m_camera_vif.axi4.ARVALID       <=  1'b0;
        d5m_camera_vif.axi4.RREADY        <=  1'b0;
        d5m_camera_vif.d5p.iImageTypeTest <=  1'b0;
        d5m_camera_vif.d5p.iReadyToRead   <=  1'b0;
        d5m_camera_vif.d5p.fvalid         <=  1'b0;
        d5m_camera_vif.d5p.lvalid         <=  1'b0;
    end
endtask: reset_signals
```

D5M FRAME

In this method, drive the signals from defined seq in uvm_sequence.

```
virtual protected task d5m_frame();
    forever begin
        @(posedge d5m_camera_vif.clkmm);
            seq_item_port.get_next_item(req);
            drive_transfer(req);
            seq_item_port.item_done();
    end
endtask: d5m_frame
```

DRIVE TRANSFER

This method which is master to dut axi4lite interface write/read data at given address using bus handshaking protocol. First valid address is transmitted and then wait for valid response in given time of 61 clock cycles in axi4_address method. Timeout accord on 62 clock cycle, if no response is asserted high on bvalid signal from dut and timeout is flagged using uvm_error macro. If valid response is asserted then axi4_data method write/read data depending on case statement. Once axi4 bus config the video process module in dut then write/read operation can be initiated by calling the d5m_pixel method.

```
virtual protected task drive_transfer (d5m_trans d5m_tx);
axi4_address(d5m_tx);
axi4_data(d5m_tx);
d5m_pixel(d5m_tx);
endtask: drive_transfer
```

AXI4 ADDRESS CHANNEL

In this method, write/read axi4 address channel.

```
virtual protected task axi4_address (d5m_trans d5m_tx);
    case (d5m_tx.d5m_txn)
        AXI4_WRITE : axi4_write_address(d5m_tx);
        AXI4_READ  : axi4_wread_address(d5m_tx)
    endcase
endtask: axi4_address
```

AXI4 DATA CHANNEL

In this method, write/read axi4 data channel through axi4_write data and axi4_read_data methods on selected case.

```
virtual protected task axi4_data (d5m_trans d5m_tx);
    bit[31:0] rw_data;
    bit err;
    rw_data = d5m_tx.axi4_lite.data;
    case (d5m_tx.d5m_txn)
        AXI4_WRITE : axi4_write_data(d5m_tx);
        AXI4_READ  : axi4_read_data(rw_data, err);
    endcase
endtask: axi4_data
```

D5M PIXEL

In this method, d5m read/write frame rgb pixel per transaction.

```
virtual protected task d5m_pixel (d5m_trans d5m_tx);
    case (d5m_tx.d5m_txn)
        D5M_WRITE  : d5m_write_pixel_data(d5m_tx);
        IMAGE_READ : d5m_read_pixel_data(d5m_tx);
    endcase
endtask: d5m_pixel
```

D5M WRITE PIXEL DATA

In this method, write data to d5m camera mod from d5m_trans sequence.

```
virtual protected task d5m_write_pixel_data (d5m_trans d5m_tx);
    d5m_camera_vif.d5p.iReadyToRead    <= 1'b0;
    d5m_camera_vif.d5p.iImageTypeTest <= d5m_tx.d5p.iImageTypeTest;
    d5m_camera_vif.d5p.rgb                       <= d5m_tx.d5p.rgb;
    d5m_camera_vif.d5p.fvalid                   <= d5m_tx.d5p.fvalid;
    d5m_camera_vif.d5p.lvalid                   <= d5m_tx.d5p.lvalid;
endtask: d5m_write_pixel_data
```

D5M READ PIXEL DATA

In this method, config test type during read operation and wait for end of frame pulse.

```
virtual protected task d5m_read_pixel_data(d5m_trans d5m_tx);
        @(posedge d5m_camera_vif.clkmm);
        d5m_camera_vif.d5p.iImageTypeTest  <= 1'b0;
        d5m_camera_vif.d5p.iReadyToRead    <= 1'b1;
    forever begin
        @(posedge d5m_camera_vif.clkmm);
            if (d5m_camera_vif.d5m.eof) break;
    end
endtask: d5m_read_pixel_data
```

AXI4 WRITE ADDRESS

In this method, write address and assert write valid high and then wait for response from dut BVALID signal within 62 clock cycles.

```
virtual protected task axi4_write_address (d5m_trans d5m_tx);
    int axi_lite_ctr;
    d5m_camera_vif.axi4.AWADDR  <= {8'h0, d5m_tx.axi4_lite.addr};
    d5m_camera_vif.axi4.AWPROT  <= 3'h0;
    d5m_camera_vif.axi4.AWVALID <= 1'b1;
    // wait for write response
    for(axi_lite_ctr = 0; axi_lite_ctr <= 62; axi_lite_ctr ++) begin
```

```
        @(posedge d5m_camera_vif.clkmm);
            if (d5m_camera_vif.axi4.BVALID) break;
    end
    if (axi_lite_ctr == 62) begin
        `uvm_error("axi_lite_master_driver","AWVALID timeout");
    end
endtask: axi4_write_address
```

AXI4 WRITE DATA

```
virtual protected task axi4_write_data (d5m_trans d5m_tx);
    int axi_lite_ctr;
    d5m_camera_vif.axi4.WDATA  <= d5m_tx.axi4_lite.data;
    d5m_camera_vif.axi4.WSTRB  <= 4'hf;
    d5m_camera_vif.axi4.WVALID <= 1'b1;
    @(posedge d5m_camera_vif.clkmm);
        for(axi_lite_ctr = 0; axi_lite_ctr <= 62; axi_lite_ctr ++) begin
        @(posedge d5m_camera_vif.clkmm);
        if (d5m_camera_vif.axi4.WREADY)
            d5m_camera_vif.axi4.AWADDR  <= 8'h0;
            d5m_camera_vif.axi4.AWPROT  <= 3'h0;
            d5m_camera_vif.axi4.AWVALID <= 1'b0;
            break;
        end
        if (axi_lite_ctr == 62) begin
            `uvm_error("axi_lite_master_driver","AWVALID timeout");
        end
    @(posedge d5m_camera_vif.clkmm);
        d5m_camera_vif.axi4.WDATA  <= 32'h0;
        d5m_camera_vif.axi4.WSTRB  <= 4'h0;
        d5m_camera_vif.axi4.WVALID <= 1'b0;
    // wait for write response
    for(axi_lite_ctr = 0; axi_lite_ctr <= 62; axi_lite_ctr ++) begin
        @(posedge d5m_camera_vif.clkmm);
        if (d5m_camera_vif.axi4.BVALID) break;
    end
    if (axi_lite_ctr == 62) begin
        `uvm_error("axi_lite_master_driver","BVALID timeout");
    end
    else begin
        if (d5m_camera_vif.axi4.BVALID == 1'b1 && d5m_camera_vif.axi4.BRESP != 2'h0)
            `uvm_error("axi_lite_master_driver","Received ERROR Write Response");
        d5m_camera_vif.axi4.BREADY <= d5m_camera_vif.axi4.BVALID;
        @(posedge d5m_camera_vif.clkmm);
    end
endtask: axi4_write_data
```

AXI4 WREAD ADDRESS

```
virtual protected task axi4_wread_address (d5m_trans d5m_tx);
    int axi_lite_ctr;
    d5m_camera_vif.axi4.ARADDR  <= {8'h0, d5m_tx.axi4_lite.addr};
    d5m_camera_vif.axi4.ARPROT  <= 3'h0;
    d5m_camera_vif.axi4.ARVALID <= 1'b1;
    for(axi_lite_ctr = 0; axi_lite_ctr <= 62; axi_lite_ctr ++) begin
        @(posedge d5m_camera_vif.clkmm);
        if (d5m_camera_vif.axi4.ARREADY) break;
```

```
    end
    if (axi_lite_ctr == 62) begin
        `uvm_error("axi_lite_master_driver","ARVALID timeout");
    end
    @(posedge d5m_camera_vif.clkmm);
        d5m_camera_vif.axi4.ARADDR  <= 8'h0;
        d5m_camera_vif.axi4.ARPROT  <= 3'h0;
        d5m_camera_vif.axi4.ARVALID <= 1'b0;
endtask: axi4_wread_address
```

AXI4 READ DATA

In this method, axi4lite read data.

```
virtual protected task axi4_read_data (output bit [31:0] data, output bit error);
    int axi_lite_ctr;
    for(axi_lite_ctr = 0; axi_lite_ctr <= 62; axi_lite_ctr ++) begin
        @(posedge d5m_camera_vif.clkmm);
        if (d5m_camera_vif.axi4.RVALID) break;
    end
    data = d5m_camera_vif.axi4.RDATA;
    if (axi_lite_ctr == 62) begin
        `uvm_error("axi_lite_master_driver","RVALID timeout");
    end
    else begin
    if (d5m_camera_vif.axi4.RVALID == 1'b1 && d5m_camera_vif.axi4.RRESP != 2'h0)
        `uvm_error("axi_lite_master_driver","Received ERROR Read Response");
        d5m_camera_vif.axi4.RREADY <= d5m_camera_vif.axi4.RVALID;
        @(posedge d5m_camera_vif.clkmm);
    end
endtask: axi4_read_data
```

MONITOR

The monitor is a system is critical component in the verification environment. It obtains and collect events and data related activity in the DUT. The information collected by monitor from dut, or stimulus is for checkers, scoreboard and coverage.

This class which extended from uvm_monitor.

Basic overview:

- Declare the virtual interface.
- Get the interface handle using get config_db.

DECLARATION:

This uvm_monitor class is derived from uvm component.

class d5m_mon_dut extends uvm_monitor;

DATA MEMEBERS

```
protected virtual d5m_camera_if d5m_camera_vif;
protected int     id;
uvm_analysis_port #(d5m_trans) mon_d5m_dut;
```

NEW CONSTRUCT

For Each component, the constructor must execute and complete in order to bring the component into existence. Therefore, new () must run before build() or any other subsequent phase can execute.

```
function new (string name, uvm_component parent);
   super.new(name, parent);
endfunction: new
```

BUILD_PHASE

In this phase, config the dut interface handle through get method. The uvm_config_db parameterized class provides a convenience interface on top of uvm_resource_db is used to for reading resource database. The uvm_config_db is derived from uvm_resource_db class. Get value of field_name "d5m_camera_vif", using component cntxt "this" as starting search point.

```
function void build_phase (uvm_phase phase);
   super.build_phase(phase);
   if(!uvm_config_db#(virtual d5m_camera_if)::get(this, "", "d5m_camera_vif", d5m_camera_vif))
   `uvm_fatal("NOVIF",{"virtual interface must be set for: ",get_full_name(), ".d5m_camera_vif"});
   mon_d5m_dut = new("mon_d5m_dut", this);
endfunction: build_phase
```

RUN_PHASE

In this method, call collection transection method.

COLLECT_TRANSACTIONS

```
virtual protected task collect_transactions();
   d5m_trans rx_fdut;
   rx_fdut         = d5m_trans::type_id::create("rx_fdut");
   forever begin
   @(posedge d5m_camera_vif.clkmm)
      rx_fdut.d5m.valid  = d5m_camera_vif.d5m.valid;
      rx_fdut.d5m.red    = d5m_camera_vif.d5m.red;
      rx_fdut.d5m.green  = d5m_camera_vif.d5m.green;
      rx_fdut.d5m.blue   = d5m_camera_vif.d5m.blue;
      rx_fdut.d5m.rgb    = d5m_camera_vif.d5m.rgb;
      rx_fdut.d5m.lvalid = d5m_camera_vif.d5m.lvalid;
      rx_fdut.d5m.fvalid = d5m_camera_vif.d5m.fvalid;
      rx_fdut.d5m.x      = d5m_camera_vif.d5m.x;
      rx_fdut.d5m.y      = d5m_camera_vif.d5m.y;
      rx_fdut.d5m.eof    = d5m_camera_vif.d5m.eof;
      mon_d5m_dut.write(rx_fdut);

   end
endtask: collect_transactions
```

d5m_monitor extended from uvm monitor. Rx monitor watches transection from dut. It receives transection from dut through virtual interface. During build phase, the monitor gets the virtual interface from the configuration database. Tlm anaysis port is declared, first is mon_d5m_dut for sending input

items in d5m_monitor_dut monitor and then d5m_mon_prd port for predicted values in d5m_monitor_predict monitor. Both ports are specialized are transection type. TLM connections are components and must be constructed in build phase. The TLM classes are never extended, just call new directly.

AGENT

D5M agent class is derived from uvm_agent which bundles together a sequencer, driver, monitor and coverage as a reusable verification component. In connect phase, agent connect analysis ports to the monitor's port. Connection to the driver and sequencer is conditional connect to driver's port and sequencer export of the agent which is configured active.

SCOREBOARD

The term scoreboard whose function is to answer does it work. The main purpose of scoreboard is to collect data about the operation of dut and compares it with expected values. D5M camera scoreboard class is derived from uvm component. It receives expected results transections and from a predictor and actual dut output transections from a monitor. It observes transactions to input of dut and computes the expected effects of those transaction and stores a representation in suitable format for later checking when corresponding transaction asserted from dut output.

ENVIRONMENT/ENV

D5M camera environment creates and configures agent to stimulate the dut and create scoreboard for the agent. Environment first register the class in the factory. Declare handles to components. Connect the agent to scoreboard.

TEST

Test class initiate and execute sequence on the specified sequencer in the run phase. First, simulator command line specifies the name of the test +UVM_TESTNAME=test to run. Then UVM factory creates a component of the test and starts its phase methods through run_test task which is called from the static part of test bench which is in initial block of the top-level test bench module.

When run_test method is called, it first creates the object of top test and then call all phases. It constructs the root component of the uvm environment in the top test which than trigger and initiate the uvm phasing component. Basically, calling run_test task causes the selected test to be constructed which first build uvm environment from top to downward and responsible for getting a reference to the uvm_root class instance from UVM core services. UVM infrastructure build phase start from selected test and continue to flow for next phasing until all uvm phases are completed which include connect and run sub phases. UVM calls $finish once all the phases are completed and return the control to top test bench module initial block. If no test and environment is created than fatal message will issued.

However, test can be run in a uvm environment by specifying the test name as argument to run_test or call test name in command line argument. Once build, connect and end_of_elaboration phases are completed start_of_simulation phase gets initiated before time consuming run phase. This phase display banners and test bench topology and configuration information.

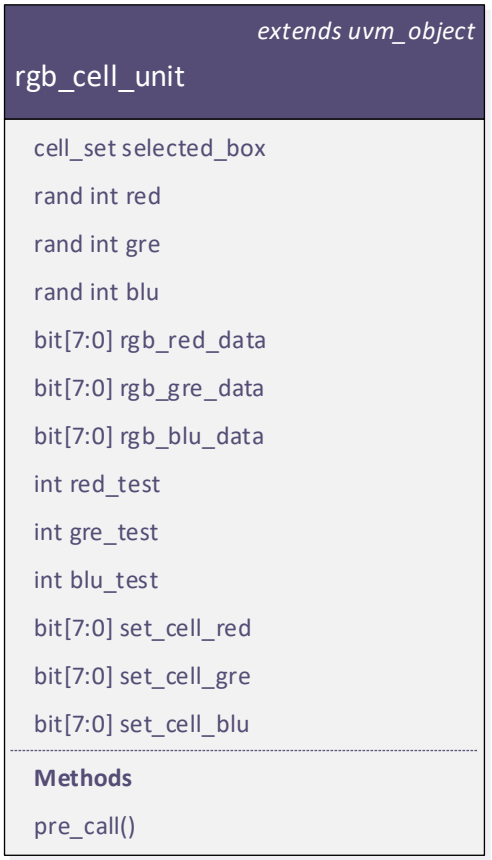
After start_of_simulation phase run phase gets initiated which is used for the stimulus generation and checking activities of the test bench and execution of all uvm_component in parallel. Each uvm_component run phase task run in parallel where main phase gets initiated, and stimulus of specified test case is generated and applied to the dut.

Most commonly in the user test, the uvm_phase object is used to raise and drop objection to void moving on the next phase, raise_objection() and drop_objection () are the methods to that. Both methods are used in the user test in the run phase and in between raise and drop objection sequence get started. When drop objection condition is met, action taken is to move to next phase of non-time-consuming cleanup phase and finally test ends.

Raise and drop objection mechanism allow hierarchical status communication among components and once all raised objections are dropped for run phase than phase end.

There are various types of image filters and color space implemented in rtl which need to be verified. Therefore, for each filter and color space tests has been created to verify rtl code. Each test is capable to configure various image dimension size and vfp configuration registers.

D5M TRANSECTION



This class which consists of d5m data items which are implemented as struct objects.

DECLARATION

This class is inherited from "uvm_sequence_item".

```
class d5m_trans extends uvm_sequence_item;
```

DATA MEMEBERS

```
rand    rgb_channel         vfp;
rand    rgb_channel         d5m;
rand    cof_channel         cof;
rand    axi4_lite_channel   axi4_lite;
rand    pattern_channel     d5p;
vfp_axi4                    axi4;
rand d5m_txn_e              d5m_txn;
```

In data member section d5m data items are listed. Data items vfp and d5m are rgb_channel type which consist of clock, valid, line valid, frame valid end of frame, start of frame, rgb channels x/y coordinates data members.

TESTBENCH_TOP

References: