Video Frame Processing

Draft version: 1.2

Contents

INTRODUCTION

The purpose of this design to take 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.

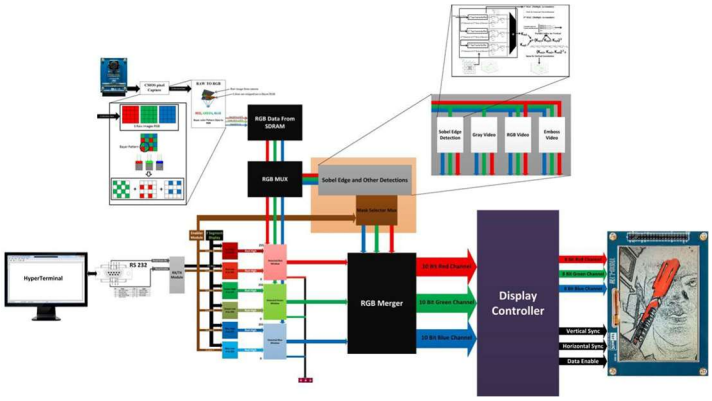


Figure 1

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 of detection is designed with a FPGA chip called Cyclone II 2C70, and it can process 800 x 480 x 10 RGB color scale video successfully. This device can locate the detected color object and the edge of the gray frame quickly and efficiently.

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 I2C[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].

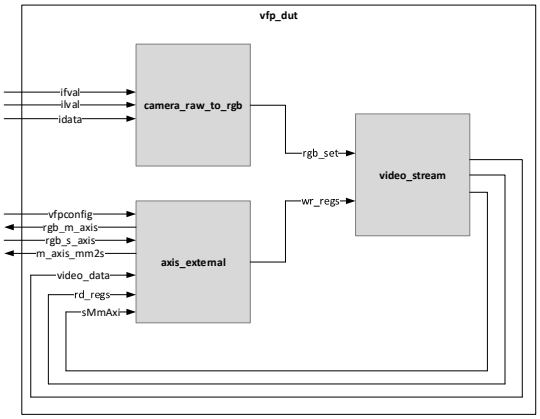


Figure 2 : Video Frame Processing Skeleton Architecture

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

This 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. In order 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 are 2^ADDR_WITH words. BMP_WIDTH AND BMP_HEIGHT specifies image frame width and height.

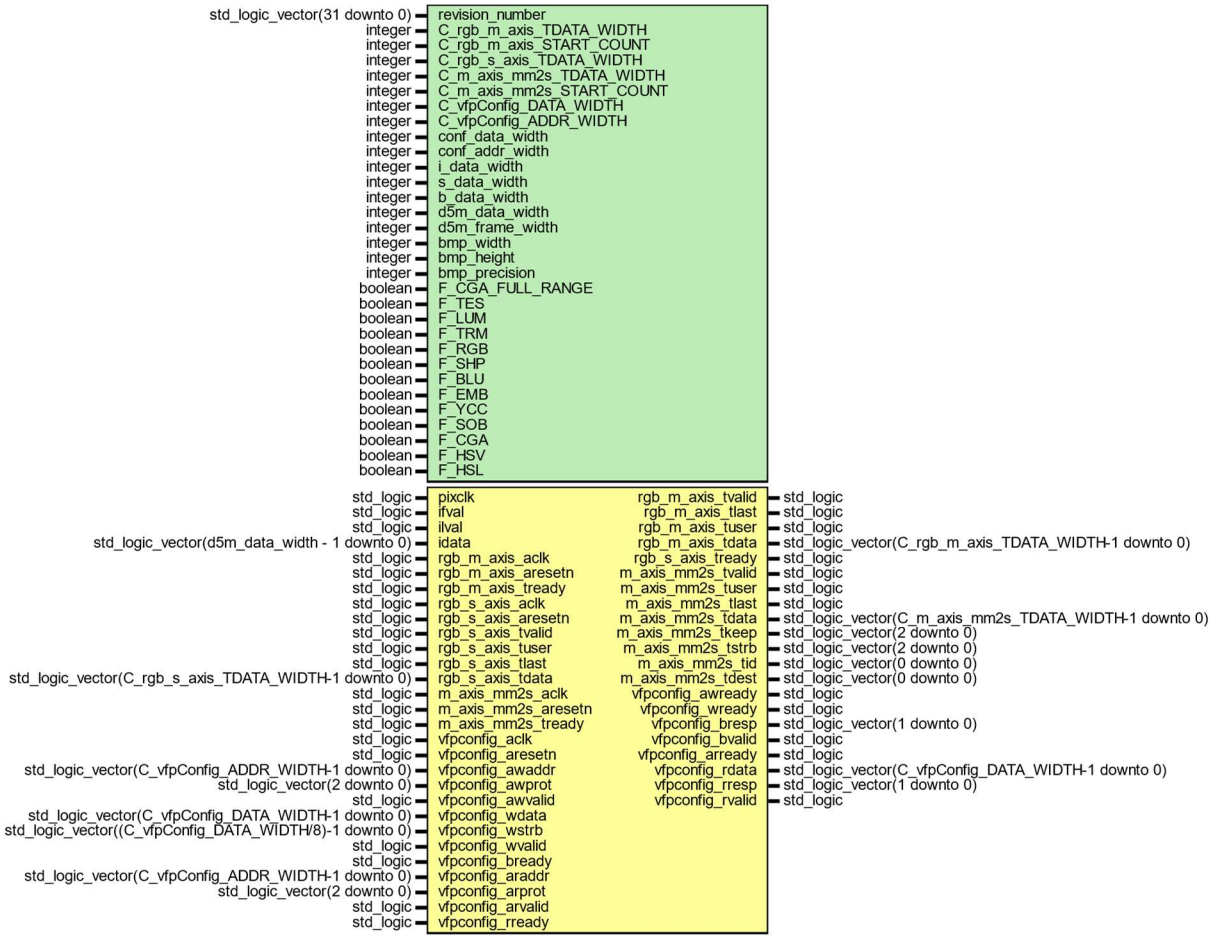


Figure 3

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. 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.

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

Figure 4

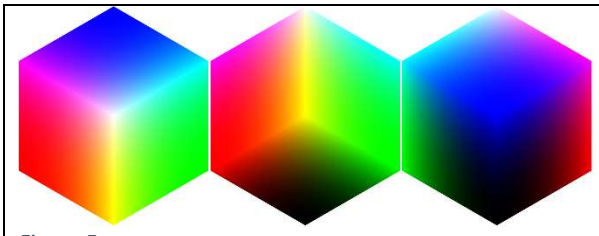


Figure 5

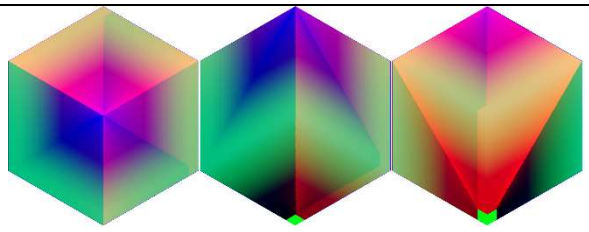


Figure 6

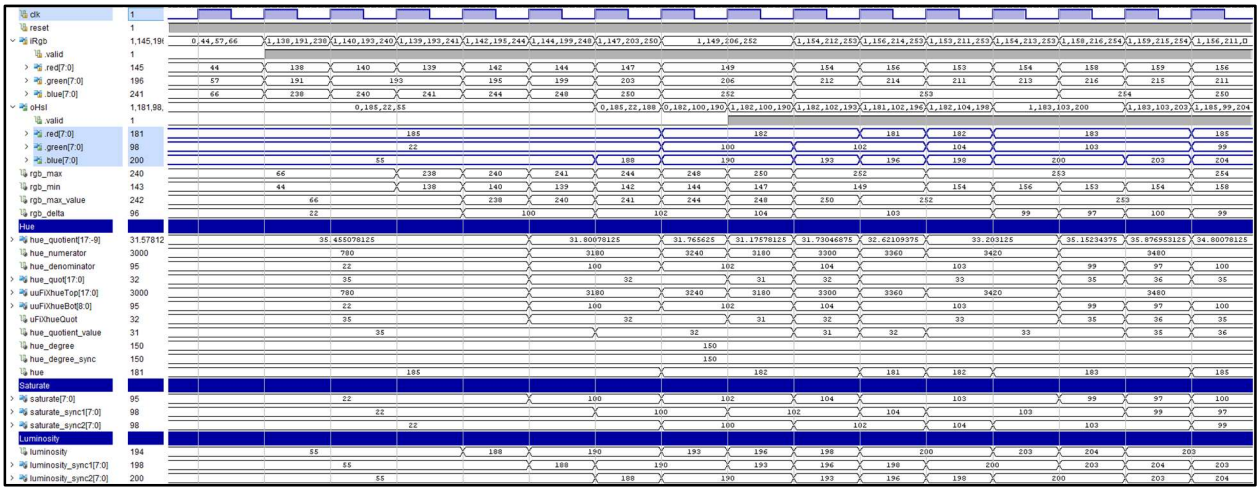


Figure 7 : HSL Filter Wave Diagram

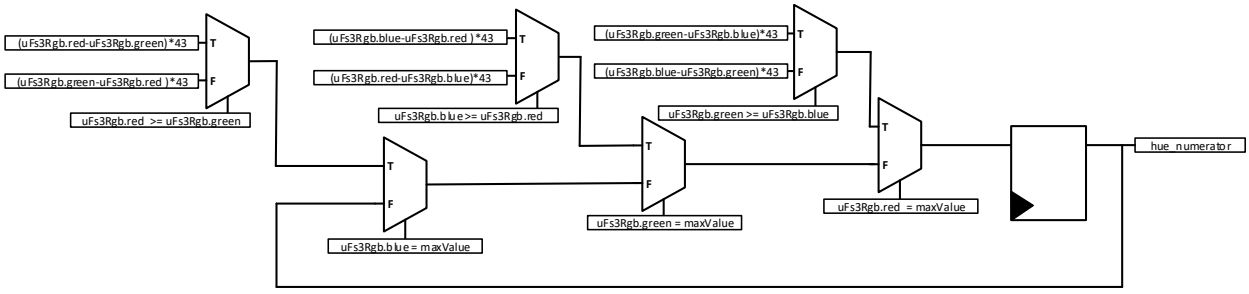


Figure 8: Hue Numerator Logic

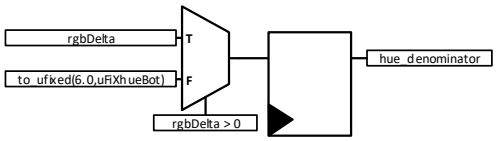


Figure 9: Hue Denominator Logic

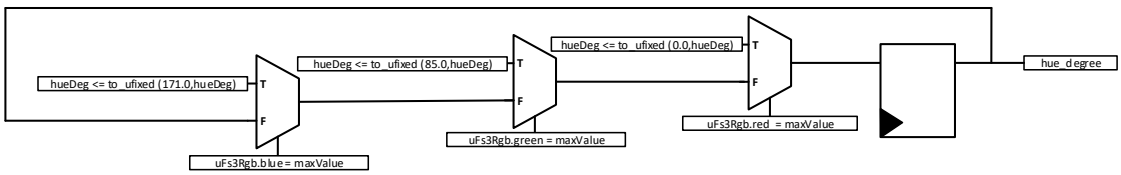


Figure 10: Hue Degree Logic

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.

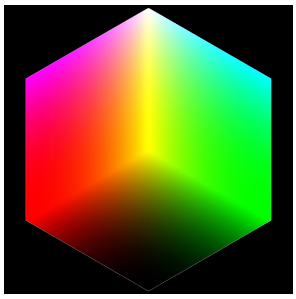


Figure 11: RGB image

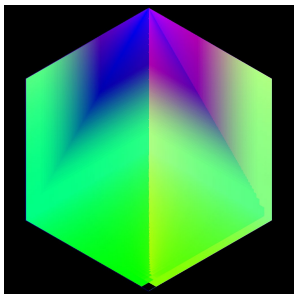


Figure 12: HSL converted image

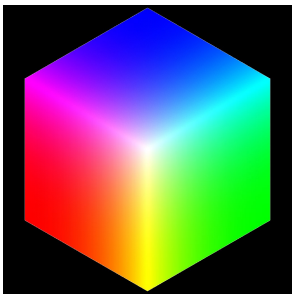


Figure 13: RGB image

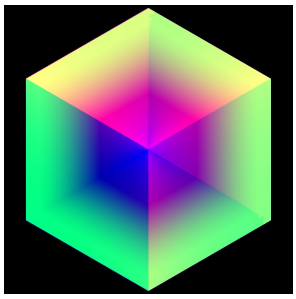


Figure 14: HSL converted image

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

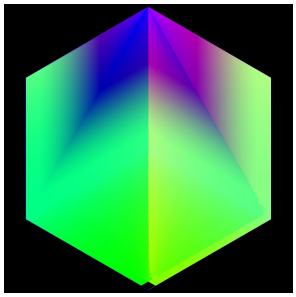


Figure 15: HSL Image

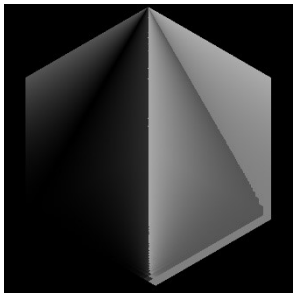


Figure 16: Hue channel

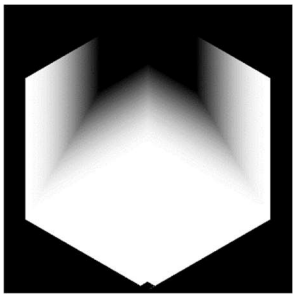


Figure 17: Saturate channel

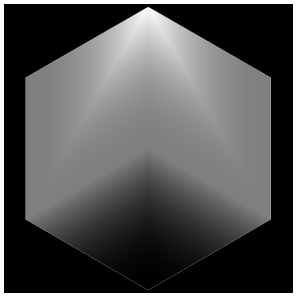


Figure 18: Luminosity channel

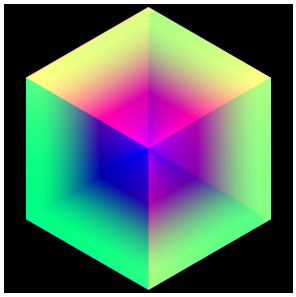


Figure 19: HSL Image

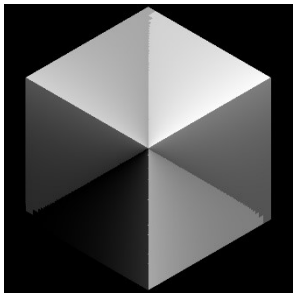


Figure 20: Hue channel

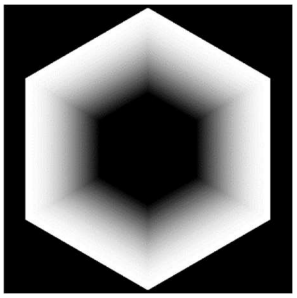


Figure 21: Saturate channel

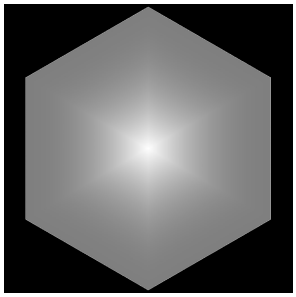


Figure 22: Luminosity channel

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.

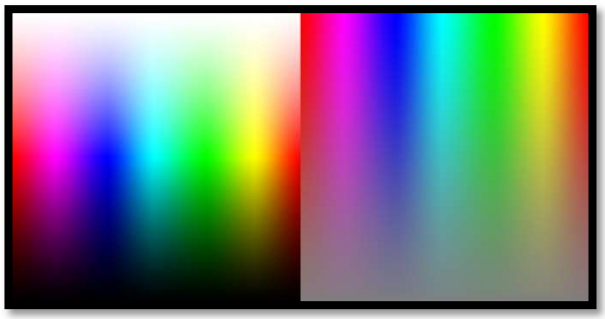


Figure 23

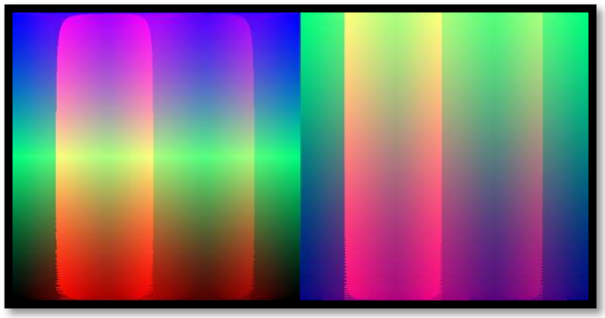


Figure 24

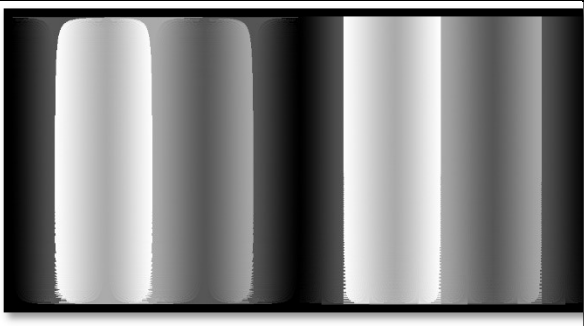


Figure 25

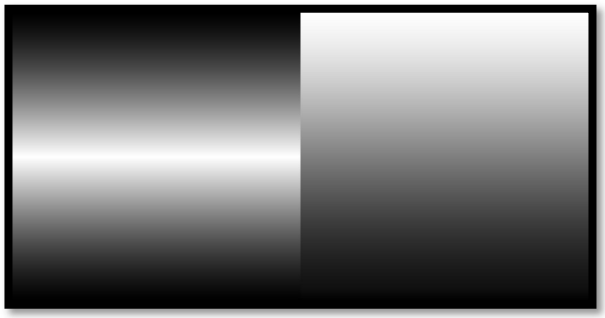


Figure 26

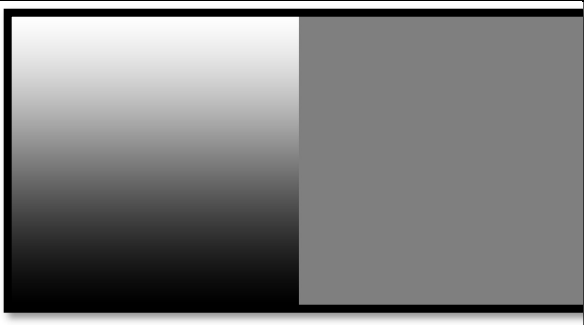


Figure 27

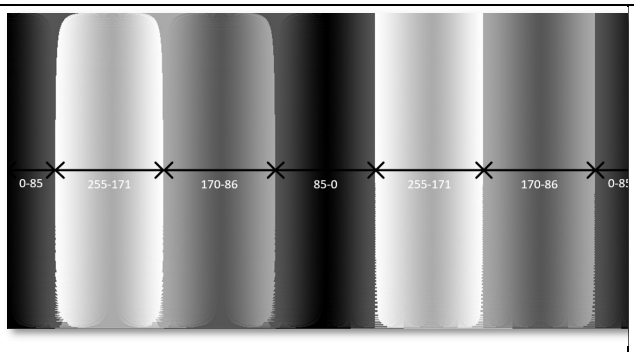


Figure 28

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.

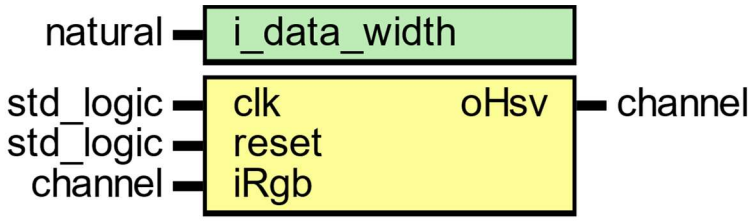


Figure 29

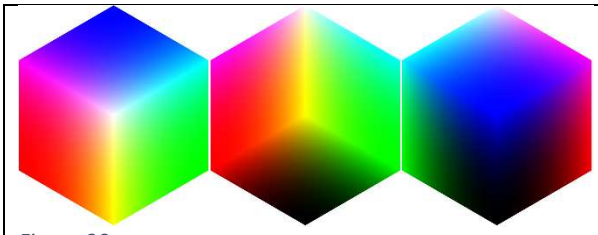


Figure 30

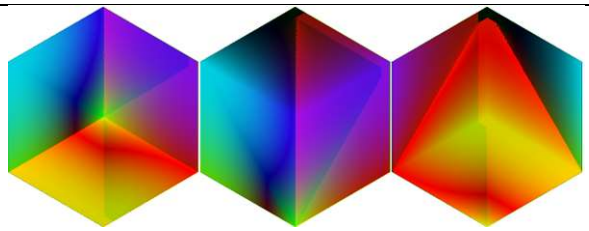


Figure 31

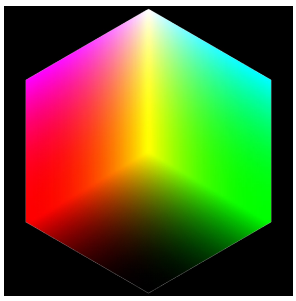


Figure 32

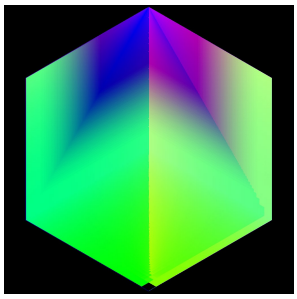


Figure 33

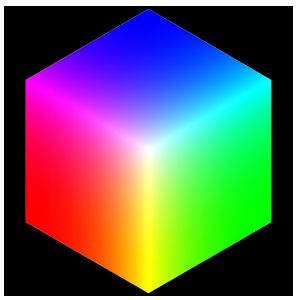


Figure 34

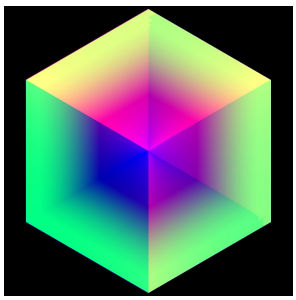


Figure 35

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

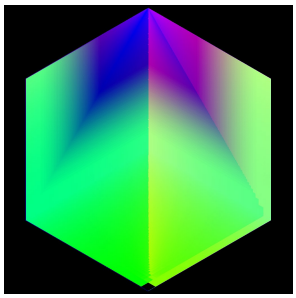


Figure 36

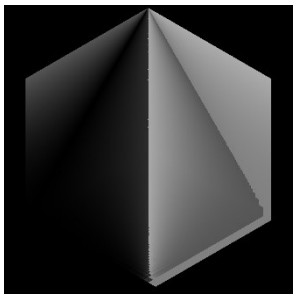


Figure 37

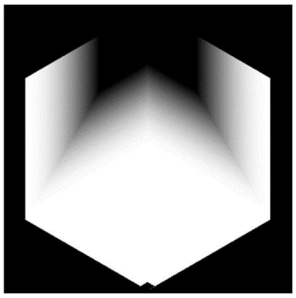


Figure 38

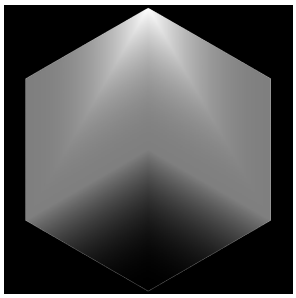


Figure 39

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

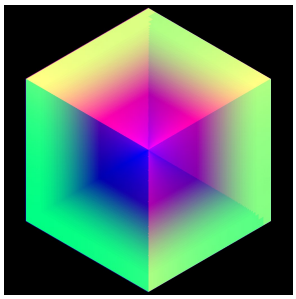


Figure 40

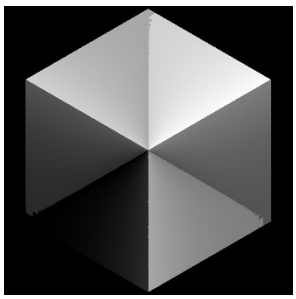


Figure 41

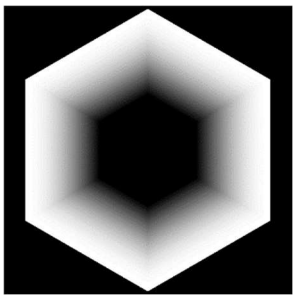


Figure 42

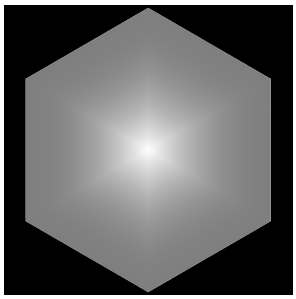


Figure 43

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

Figure 44

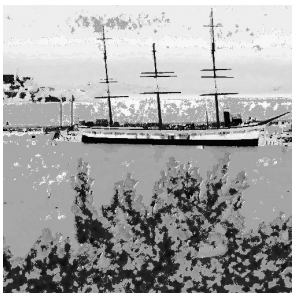


Figure 45

Figure 46

Figure 47

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 48

Figure 49

Figure 50

Figure 51

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

Figure 52

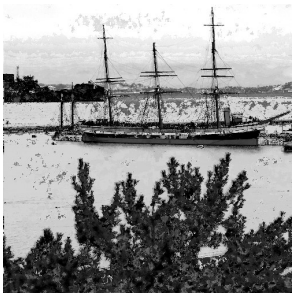


Figure 53

Figure 54

Figure 55

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

Figure 56

Figure 57

Figure 58

Figure 59

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

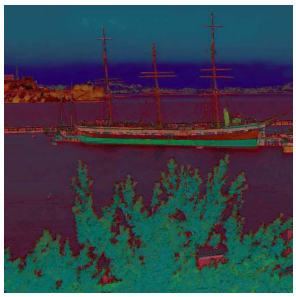


Figure 60

Figure 61

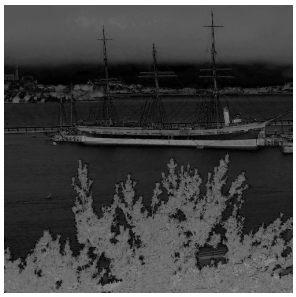


Figure 62

Figure 63

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

Figure 64

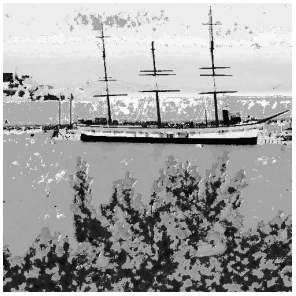


Figure 65

Figure 66

Figure 67

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

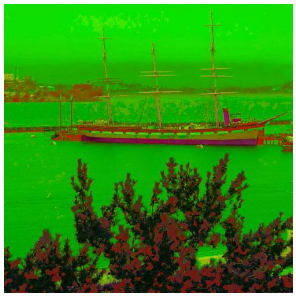


Figure 68

Figure 69

Figure 70

Figure 71

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

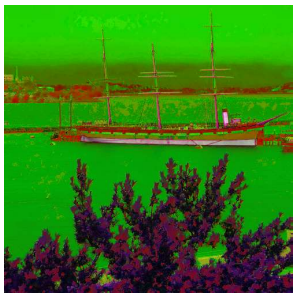


Figure 72

Figure 73

Figure 74

Figure 75

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

Figure 76

Figure 77

Figure 78

Figure 79

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

CMYK COLOR SPACE

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

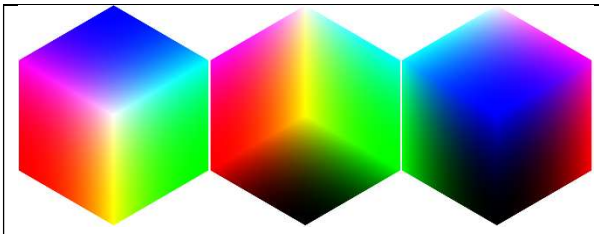


Figure 80

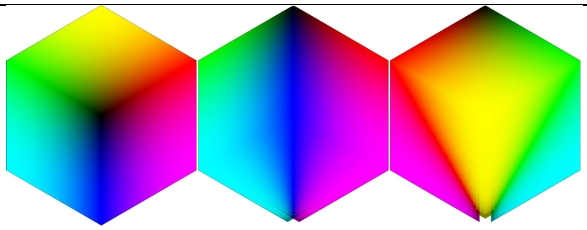


Figure 81

Figure 82

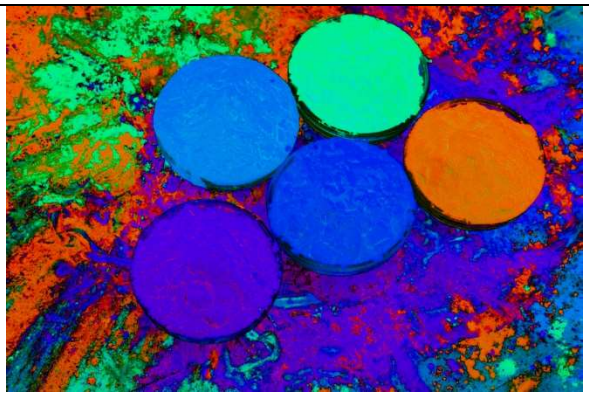


Figure 83

Figure 84

Figure 85

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

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

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

YD_RD_B COLOR SPACE

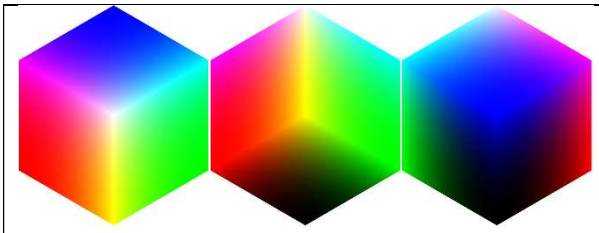


Figure 86

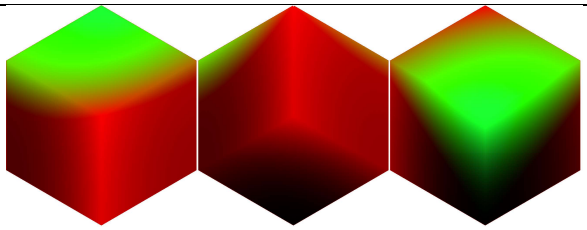


Figure 87

Figure 88

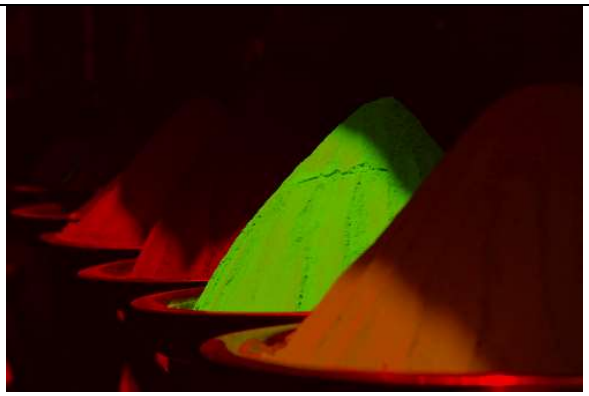


Figure 89

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

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

CIEXYZ COLOR SPACE

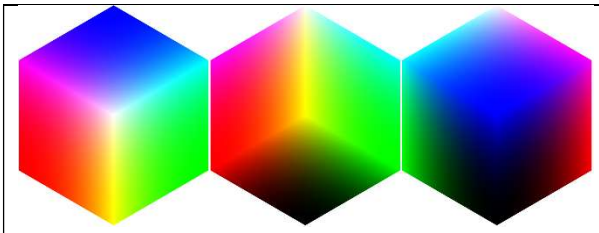


Figure 90

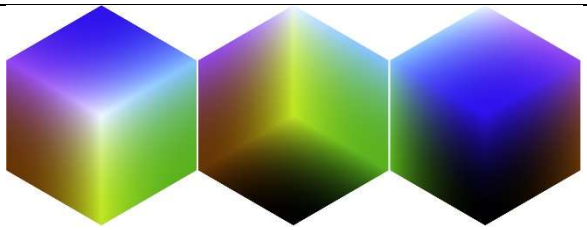


Figure 91

Figure 92

Figure 93

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.

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

CIEYUV COLOR SPACE

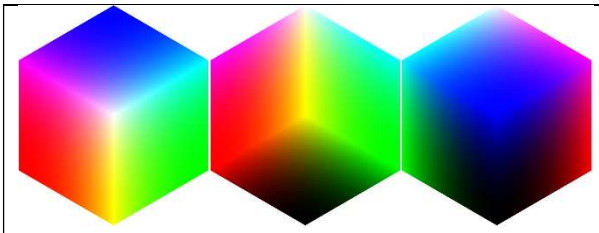


Figure 94

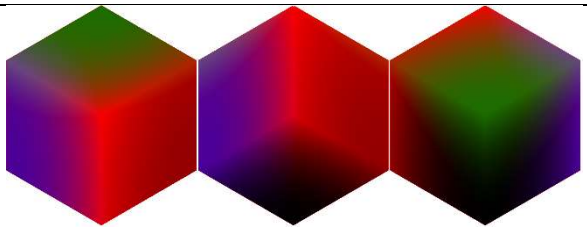


Figure 95

Figure 96

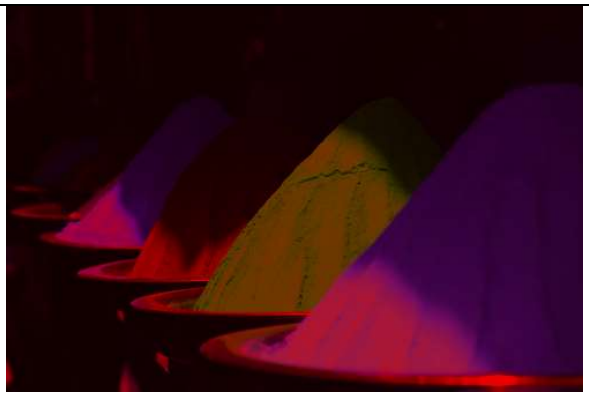


Figure 97

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

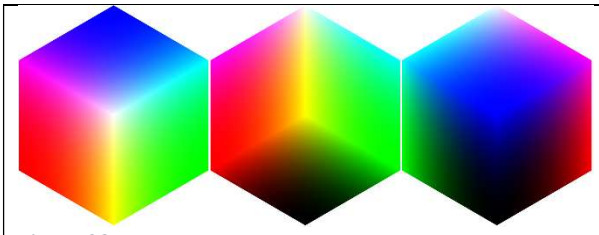


Figure 98

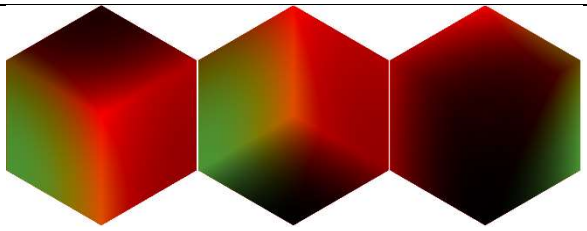


Figure 99

Figure 100

Figure 101

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

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

YPBPR COLOR SPACE

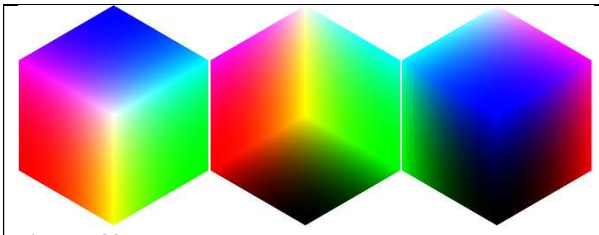


Figure 102

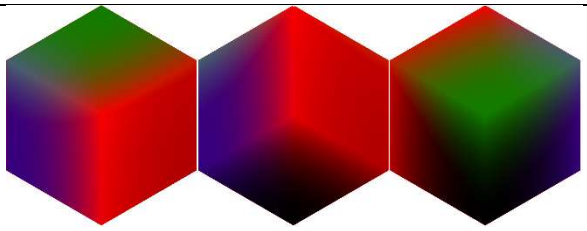


Figure 103

Figure 104

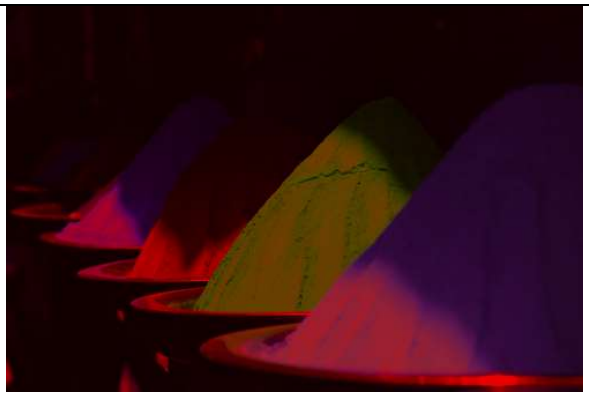


Figure 105

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

Figure 106

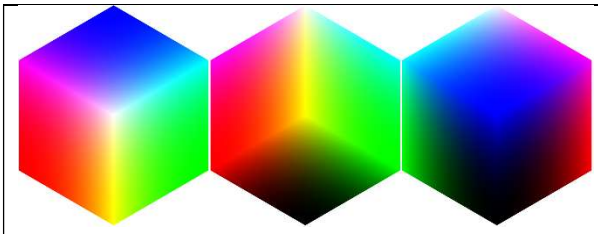


Figure 107

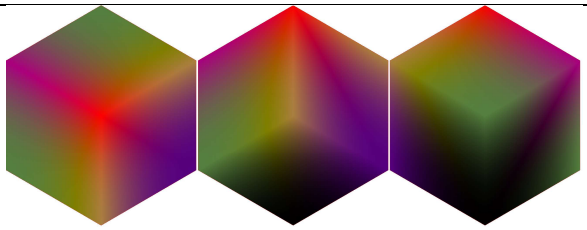


Figure 108

Figure 109

Figure 110

LMS COLOR SPACE

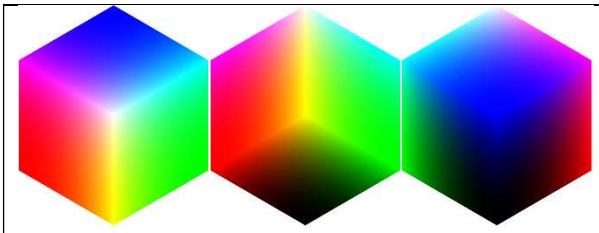


Figure 111

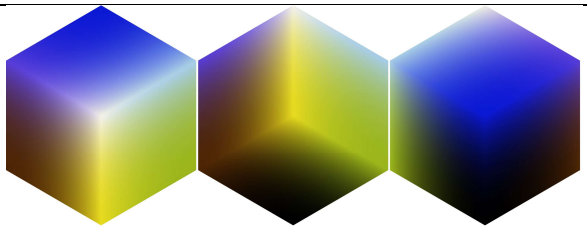


Figure 112

Figure 113

Figure 114

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

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

$ICTCP$ COLOR SPACE

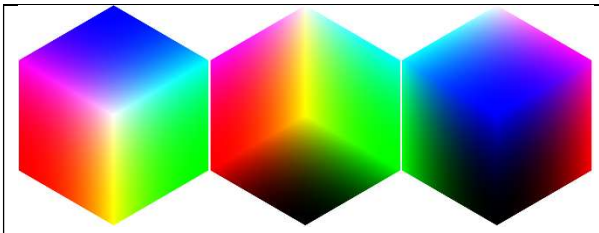


Figure 115

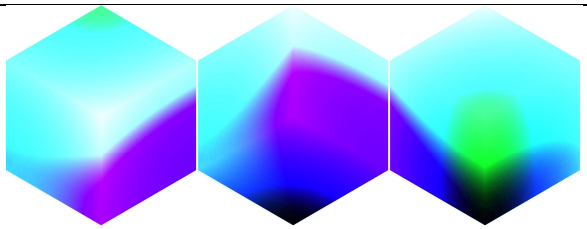


Figure 116

Figure 117

Figure 118

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

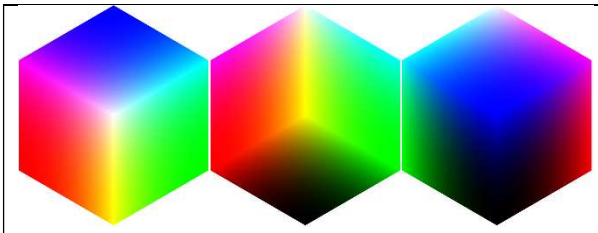


Figure 119

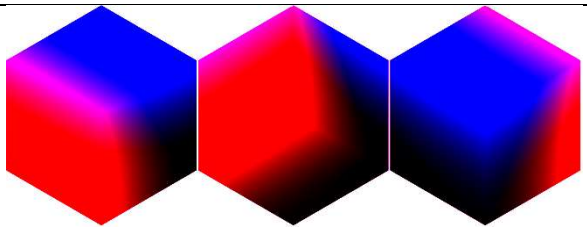


Figure 120

Figure 121

Figure 122

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

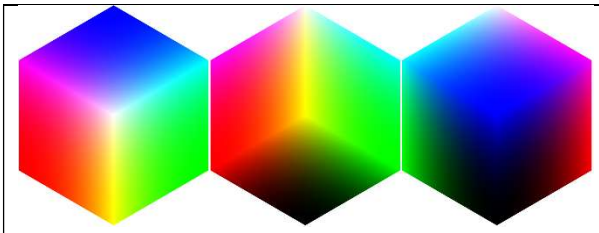


Figure 123

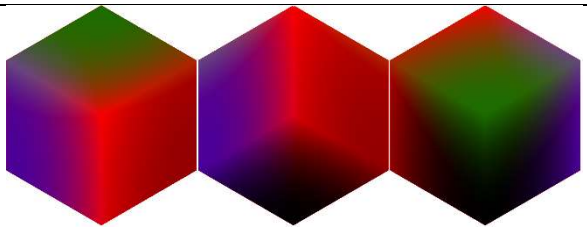


Figure 124

Figure 125

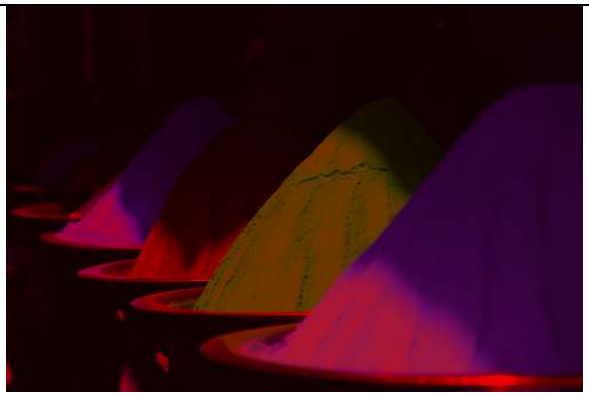


Figure 126

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

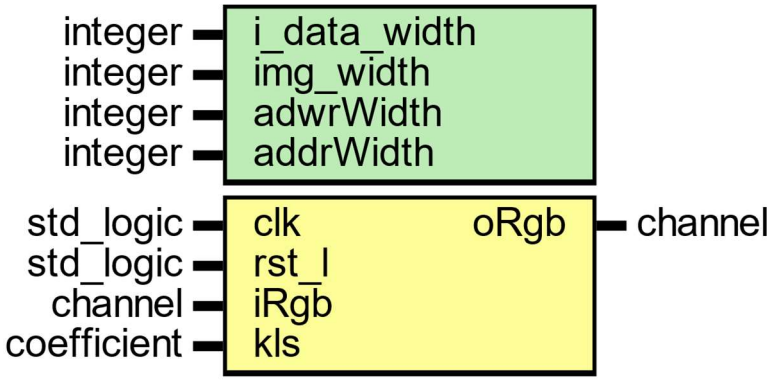


Figure 127

BLUR FILTER

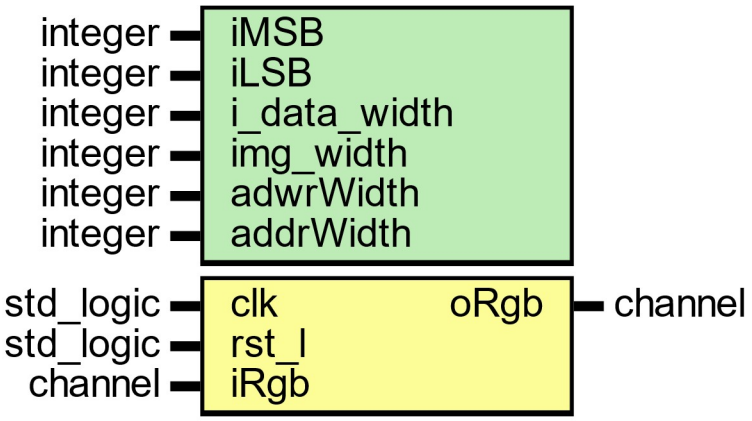


Figure 128

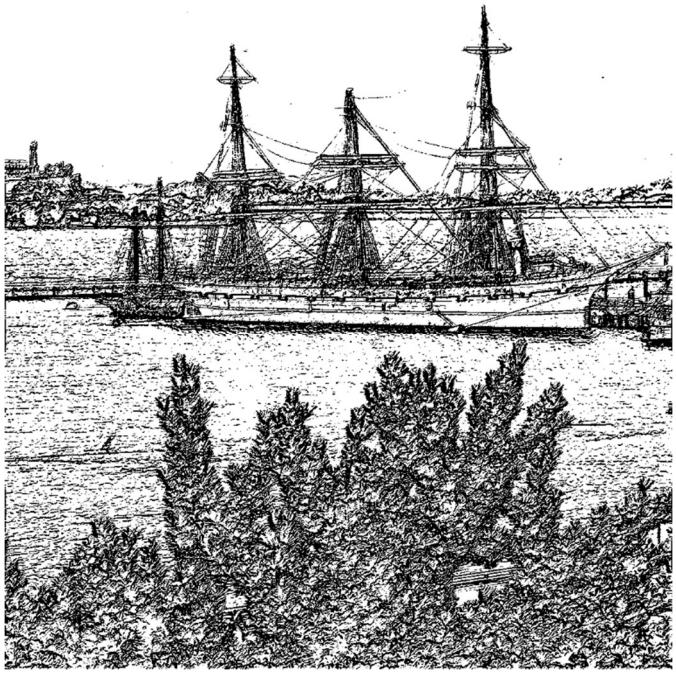
Gray Level Colors			
0.400	0.350	0.200	0.950
0.200	0.700	0.100	1.00
0.05	0.100	0.900	1.05
0.605	1.15	1.20	**∑**

EMBOSS FILTER

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

SOBEL FILTER

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



YCBCR COLOR SPACE

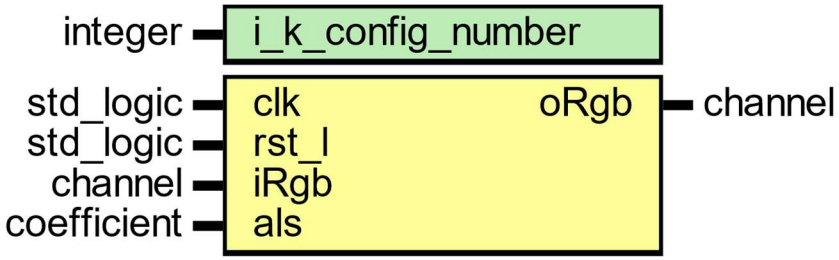


Figure 129

COLOR ADJUST MATRIX

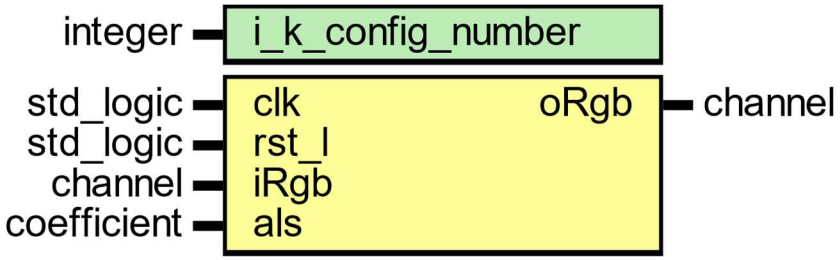
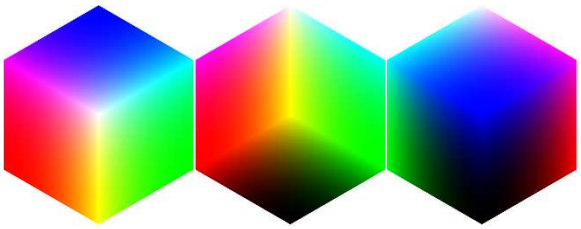


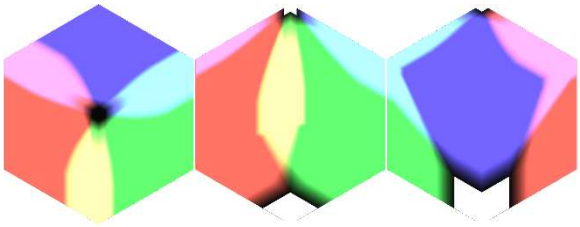
Figure 130

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.

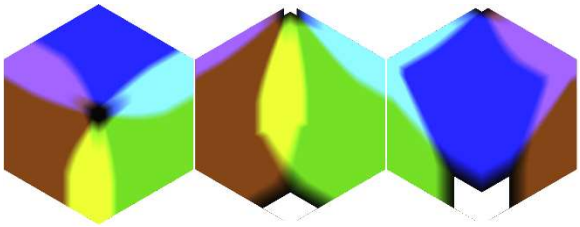
$$\begin{bmatrix} R_{CAM} \\ G_{CAM} \\ B_{CAM} \end{bmatrix} = \begin{bmatrix} K1 & K2 & K3 \\ K4 & K5 & K6 \\ K7 & K8 & K9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



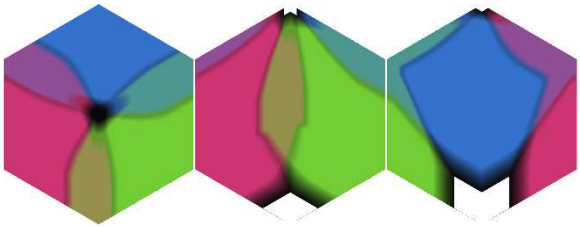
Color Adjust Matrix				
CAM-1	5.000	-3. 313	-1. 687	0.00
	-1. 687	5.000	-3. 313	0.00
	-3. 313	-1. 687	5.000	0.00
	0.00	0.00	0.00	
CAM-2	1.200	-0.435	-0.165	0.600
	-0.165	1.200	-0.435	0.600
	-0.435	-0.165	1.200	0.600
	0.600	0.600	0.600	
CAM-3	0.900	0.300	0.350	1.650
	0.350	0.900	0.300	1.650
	0.300	0.350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



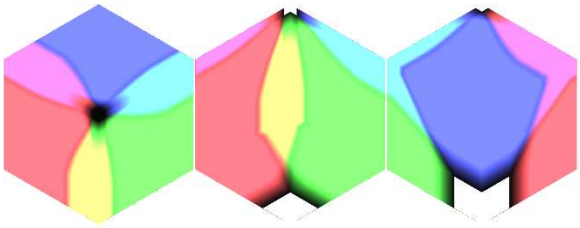
Color Adjust Matrix				
CAM-1	1.340	-0.850	-0.400	0.09
	-0.400	1.340	-0.850	0.09
	-0.850	-0.400	1.340	0.09
	0.09	0.09	0.09	
CAM-2	2.000	-0.200	-0.100	1.70
	-0.100	2.000	-0.200	1.70
	-0.200	-0.100	2.000	1.70
	1.70	1.70	1.70	
CAM-3	1.5	0.900	0.650	3.05
	0.650	1.5	0.900	3.05
	0.900	0.650	1.5	3.05
	3.05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

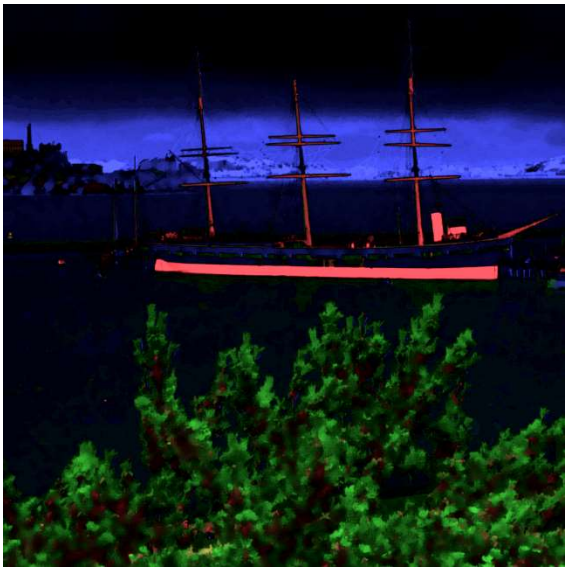


Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	

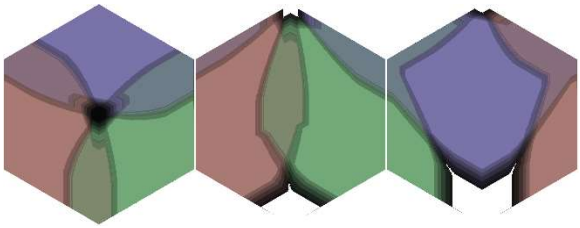


Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	

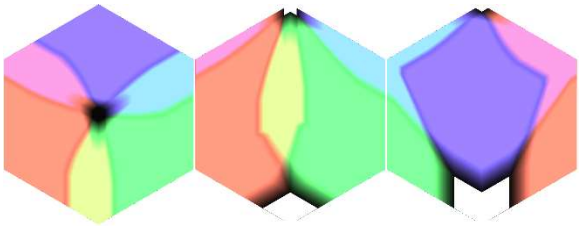




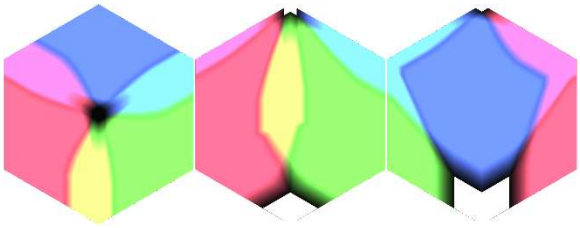
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



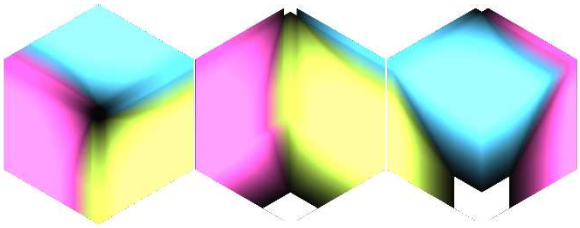
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



Color 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 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	



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 merge 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.

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.

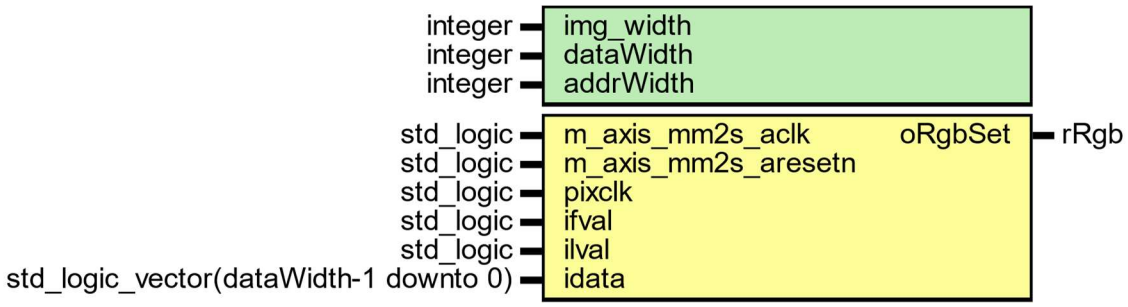


Figure 131 : camera raw data module

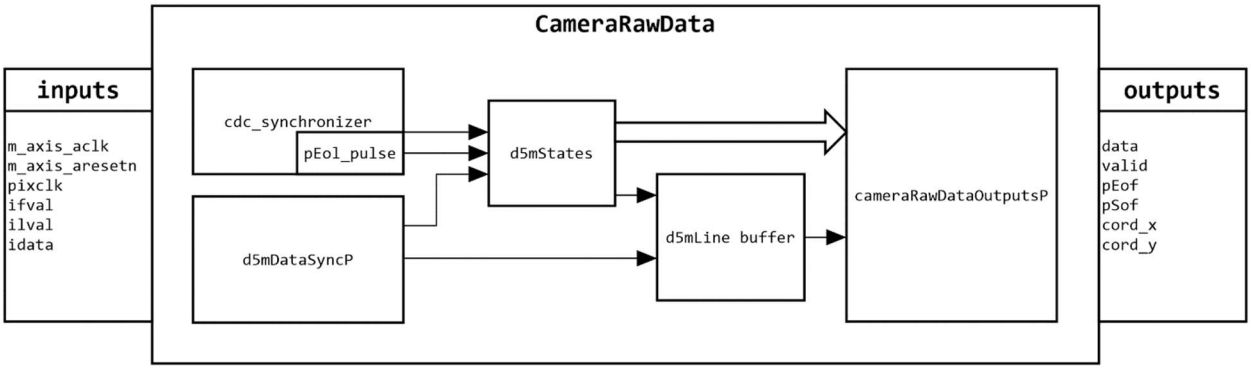


Figure 132 : General view of camera raw data flow

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

IMAGE READ INTERFACE

Figure 133

Image processed are shown below using ccm1.

THREE TAPS DATA

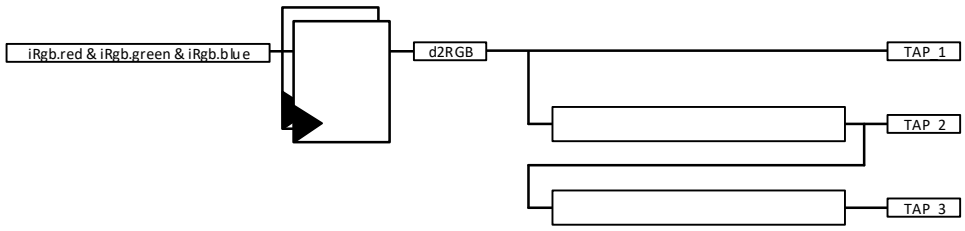


Figure 134

FOUR TAPS DATA

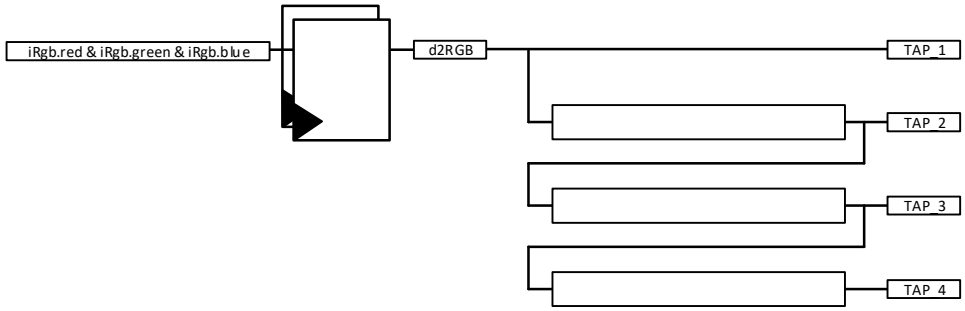


Figure 135

PIXEL COORDINATES

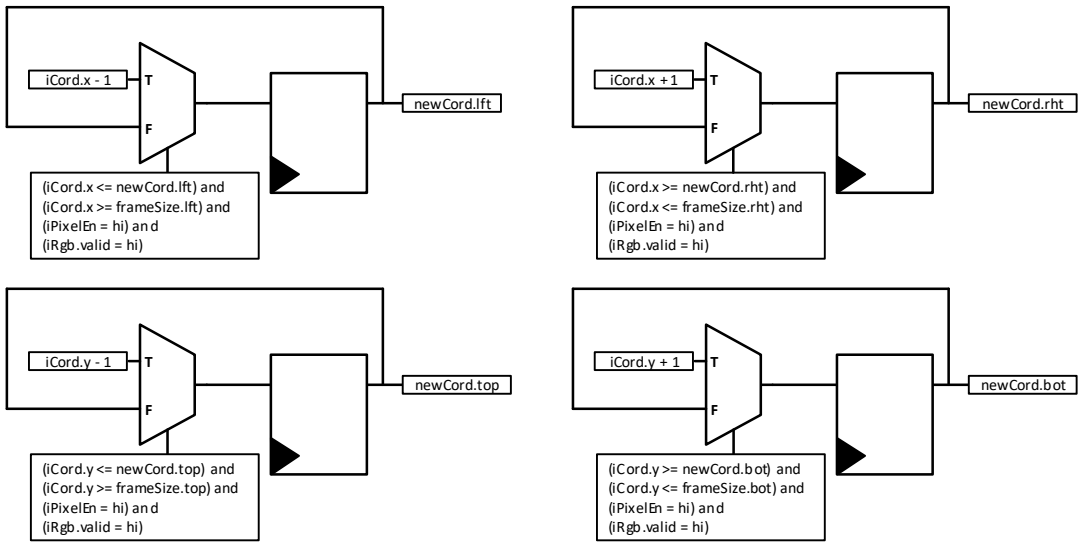


Figure 136

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

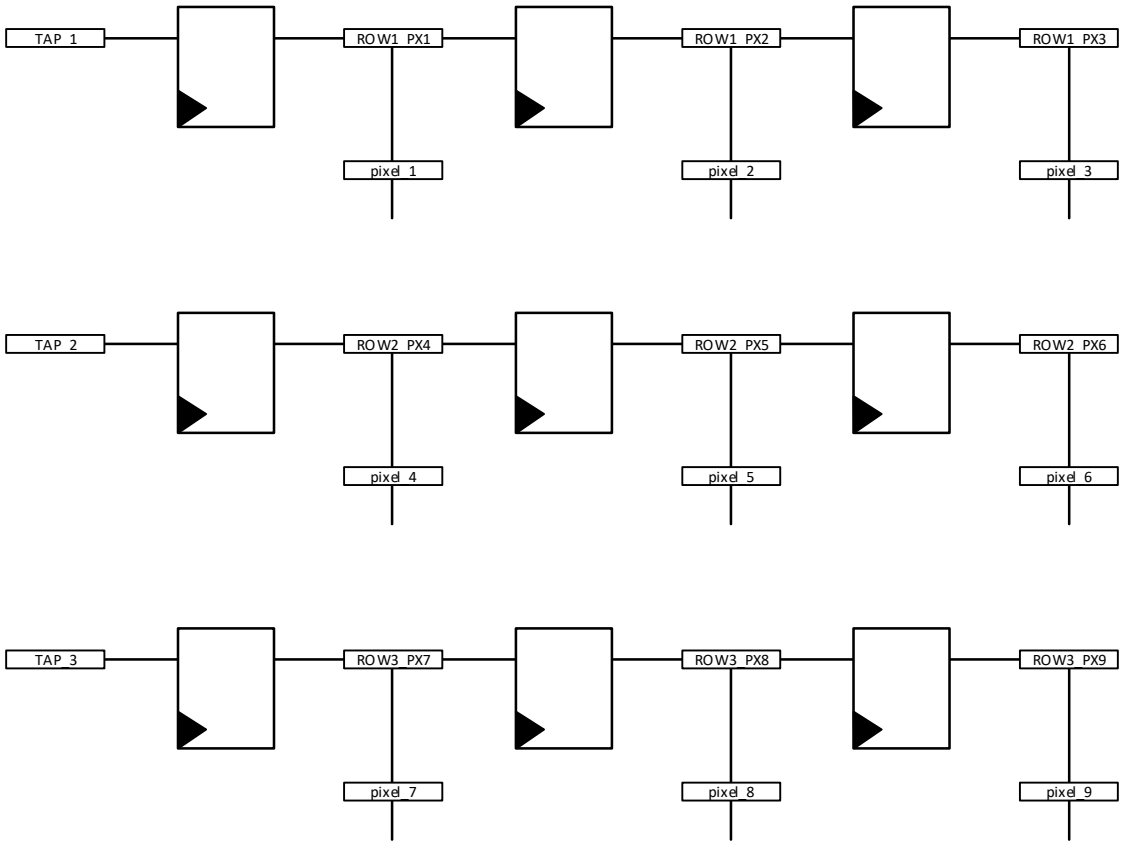
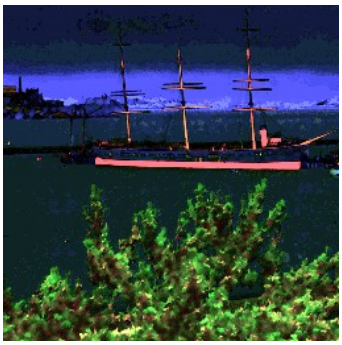


Figure 137

K1	K2	K3
K4	K5	K6
K7	K8	K9

Figure 138

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.

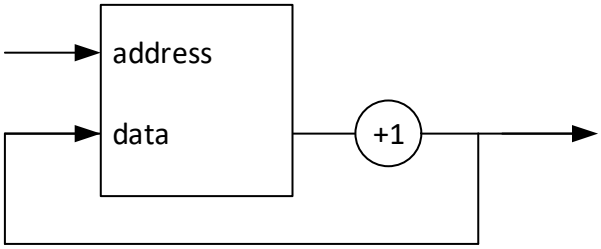
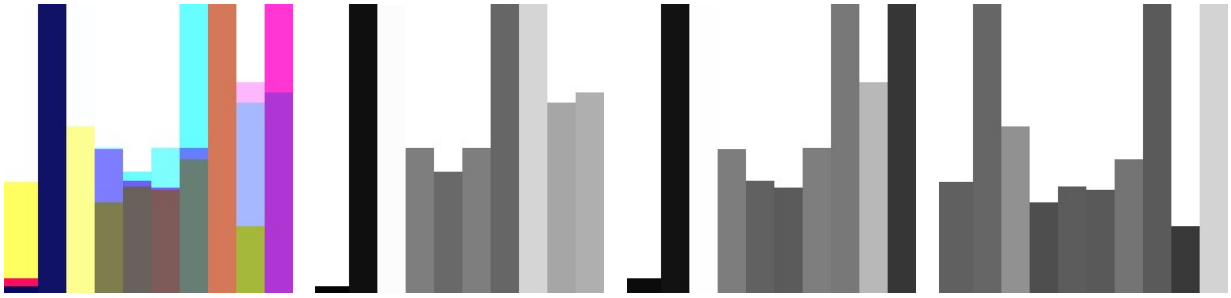


Figure 139

Histogram Equation



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

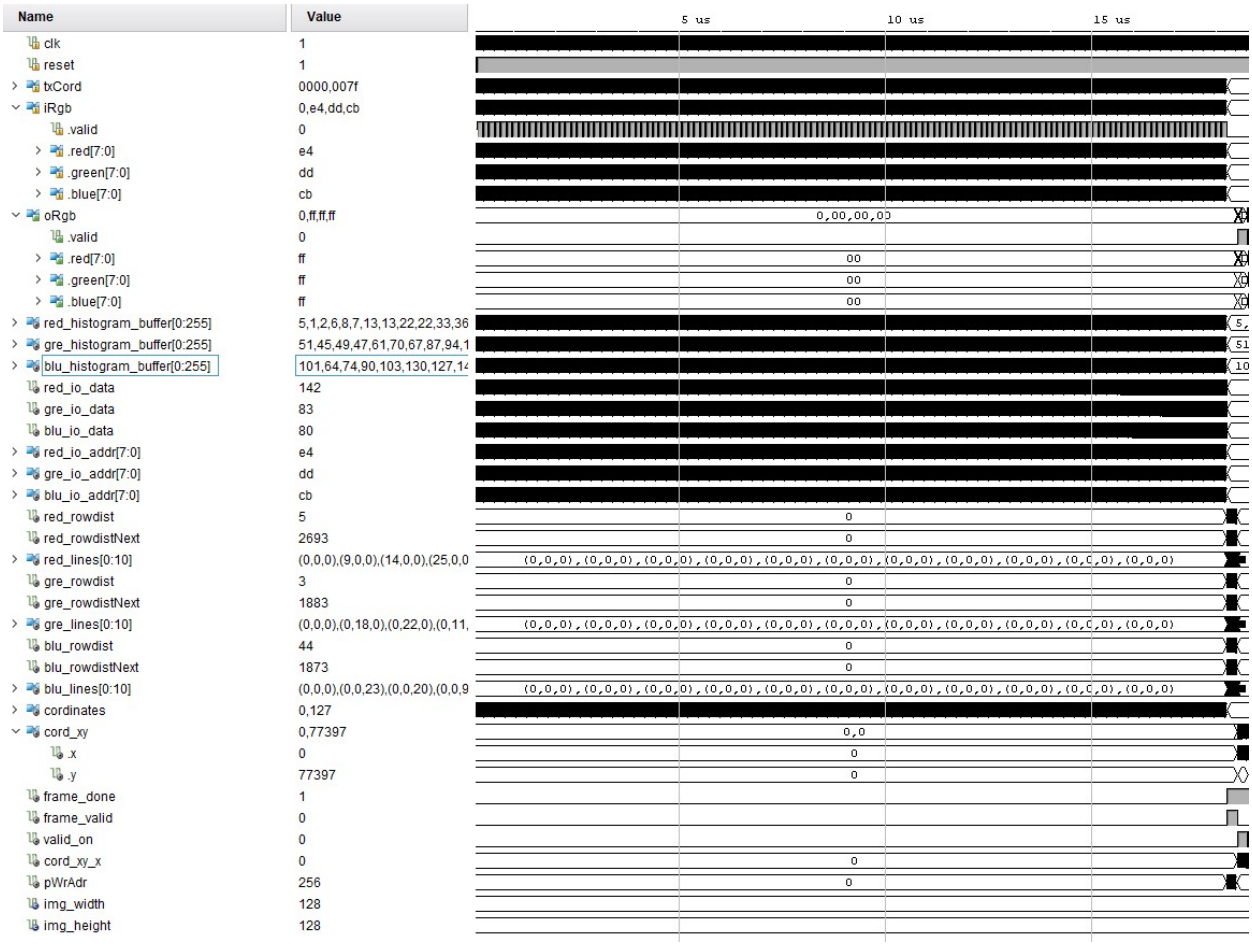


Figure 140

TESTBENCH COMPONENTS/OBJECTS

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, **1**=128x125, **2**=255x255, **3**=1920x1080	
sharp		
blur		
embross		
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	

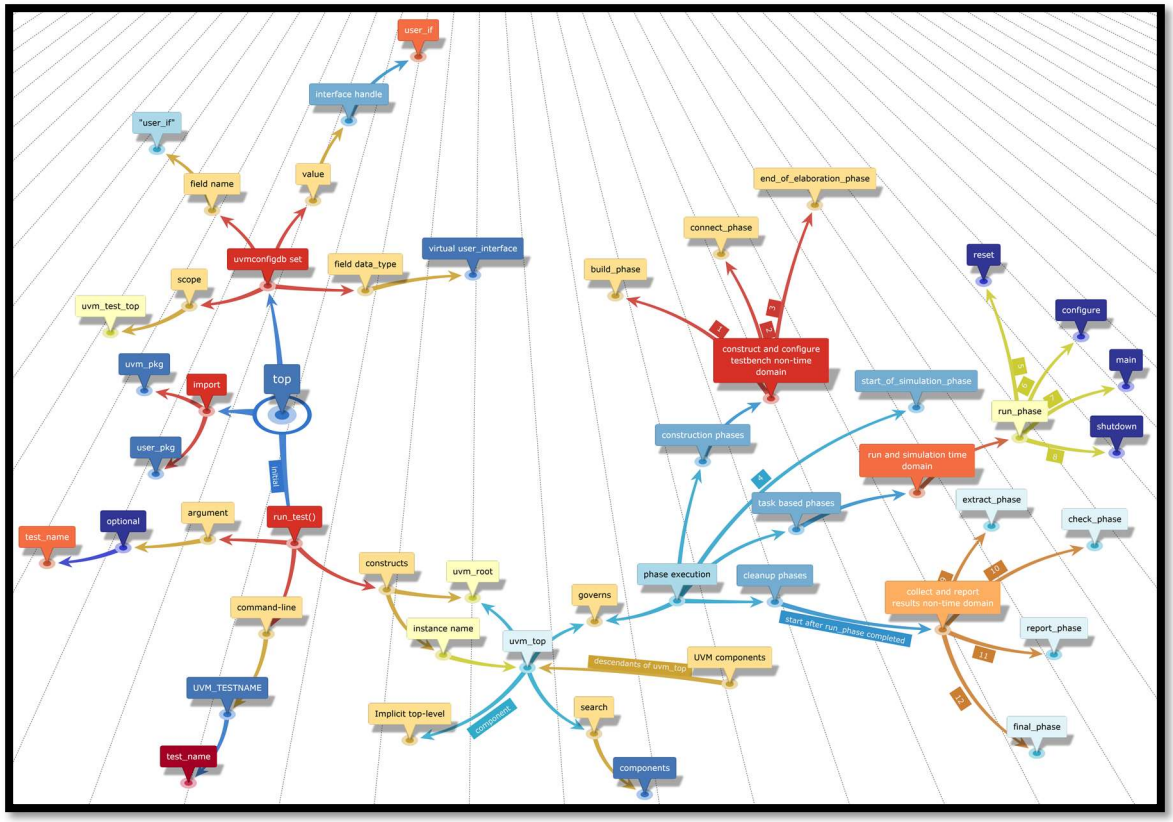


Figure 141

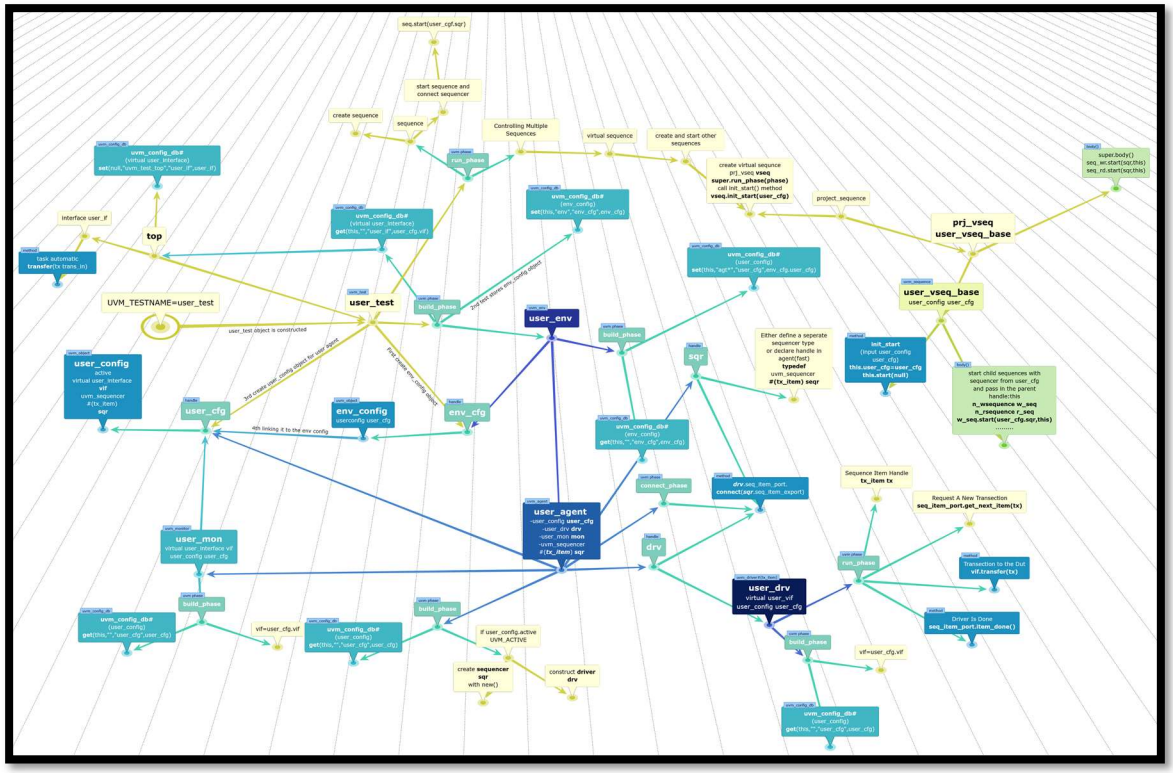


Figure 142

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 143

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

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
```

Tx monitor watches transection from dut. d5m_monitor extended from uvm monitor. 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

SCOREBOARD

ENVIRONMENT/ENV

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.

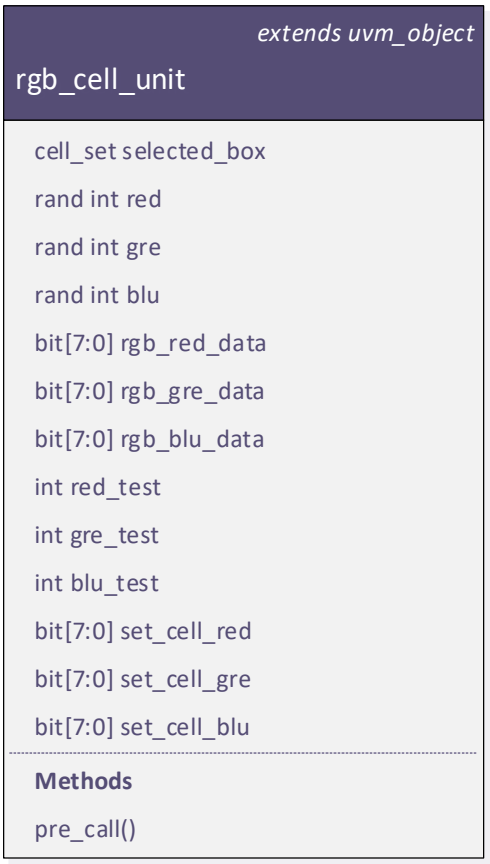
After start_of_simulation phase run phase gets initiated which is used for the stimulus generation and checking activities of the test bench and execution of all uvm_component in parallel. Each uvm_component run phase task run in parallel where main phase gets initiated, and stimulus of specified test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

TESTBENCH_TOP



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.

References:

Scratch Notes:

Color differences

The color difference between two colors is the distance between the color points of the two colors. $\mathbf{\Delta E^*_{ab}}$ is the color difference of two colors, '1' and '2', that have color coordinates L^*_1, a^*_1, b^*_1 and L^*_2, a^*_2, b^*_2, respectively. A small ΔE^*_{ab} value implies that the colors are close to one another. The color difference can be calculated by using the mathematical theorem of Pythagoras:

$$\Delta E^*_{ab} = \sqrt{\left(\Delta L^*\right)^2 + \left(\Delta a^*\right)^2 + \left(\Delta b^*\right)^2} = \sqrt{\left(L^*_1 - L^*_2\right)^2 + \left(a^*_1 - a^*_2\right)^2 + \left(b^*_1 - b^*_2\right)^2}$$

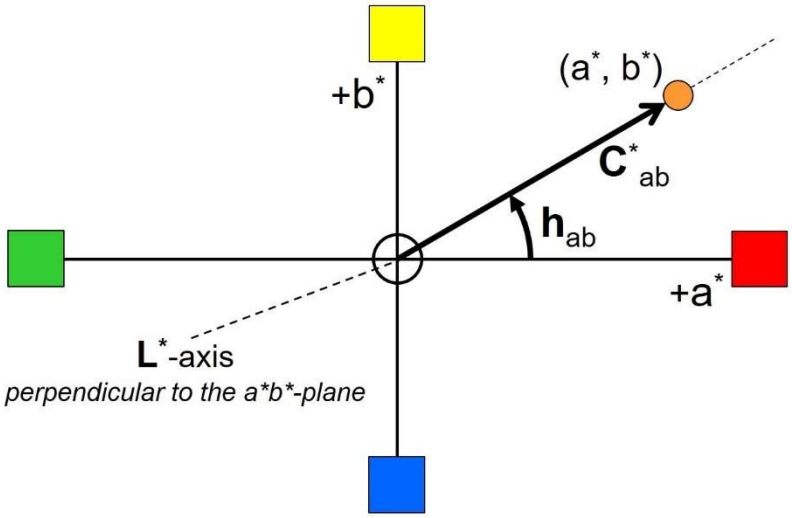
The concept and quantification of color differences is important when a color must be matched or when a new **batch** of a paint must have a color that is close to the color of the standard paint, the **reference**. But, how well must the color of the new batch match with the color of the reference? The answer to this question is a criterion in the quality control of a new batch of paint. The color coordinates of the batch and the reference are measured and the color difference $\mathbf{\Delta E^*_{ab}}$ is calculated. Depending on the industry, the criterion for color approval can be strict, for example $\Delta E^*_{ab} < 0.3$, or much milder, for example $\Delta E^*_{ab} < 1.5$.

In the daily practice of those who work with paints, the representation of a color difference is often simplified as ΔE^* or ΔE.

Hue and Chroma

Hue is the *color tone* or *color name* of a color. **Chroma** is the amount of saturation of a color. Colors of high chroma are said to be *clear*, *bright* or *brilliant*. Dull (pastel) colors have a low chroma.

Hue and chroma can be visualized and quantified by using the a^*b^*-plane of the CIELAB color space.



Hue angle and chroma in the a*b*-plane

The hue of a color is quantified by its hue angle $\mathbf{h}_{ab}$ (without * symbol) in the a^*b^*-plane, given in degrees (°). The hue angle range starts, by definition, at the positive side of the a^*-axis and goes counter-clockwise. This implies that **red** has a hue angle of 0°. A full circle goes from 0° to 360°. **Cyan blue** (an organic blue pigment with color index PB 15), for example, has a hue angle of about 240°. The hue angle of a color can be calculated from the color coordinates:

$$h_{ab} = \arctan\frac{b^*}{a^*}$$

The chroma of a color is quantified by $\mathbf{C}^*_{ab}$; it is the distance of the color point to the L^*-axis. Chroma can be calculated from the color coordinates by using Pythagoras' theorem:

$$C^*_{ab} = \sqrt{\left(a^*\right)^2 + \left(b^*\right)^2}$$

A **saturated** (or **brilliant**) color is represented by a color point that is far away from the lightness axis. The color point of a **pale** (or **dull**, or **pastel**) color, a color of low saturation, is close to the L^*-axis.