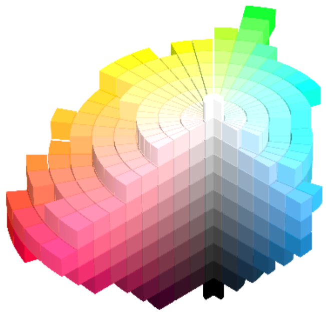
VIDEO COLOR PROCESSING

Draft version: 2.0

Sakinder Ali



Contents

INTRODUCTION

Video color processing based upon implementation of raspberry pi camera link running on a Kria KV260 board. This design capture rgb video data via MIPI interface and process video data into axi4 stream and output to ethernet /display port.

This design takes 10-bit RGB format video stream and apply various filters and color space conversions. At high level, the design comprises of three pipelines: Capture, Process and Output pipelines. During process pipeline each rgb pixel either filtered or converted into color space which is implemented video color processing (VCP) module.

VCP module contains a collection of control registers and maintain a local small buffers to store video frame lines. A VCP module takes input video pixel stream, performs computation, or adds new contents to the stream through filters, and then outputs the processed pixel stream. VCP module can be either a pixel transformation or a pixel generation.

VCP have following filters and color space conversion which can enabled before build:

- Color Correction Matrix
- K-Mean Color Clustering
- Test-Pattern
- Filters
  - SHARP
  - BLUR
  - EMBOSS
  - SOBEL EDGE DETECTION
  - CONTRAST
- Color Space Conversions
  - RGB TO HSL
  - HSL TO RGB
  - RGB TO YCBCR
  - YCBCR TO RGB
  - RGB TO CMYK
  - RGB TO YDBDR
  - RGB TO CIEXYZ
  - RGB TO CIEYUV
  - RGB TO YIQ
  - RGB TO YPBPR
  - RGB TO LMS
  - RGB TO ICTCP
  - RGB TO HED
  - RGB TO YC1C2
  - RGB TO YCBCR

Image frame resolution is set to 3840x2160 at 30 frames per second and maximum full resolution of 4056x3040 is also supported but limited to 15 frames per second.

The simplified block diagram for implemented end-to-end pipeline design is shown in Figure 1. The design is appliable to process video from both connected cameras AR1335 and IMX477 and transmit video stream to output components display and ethernet ports.

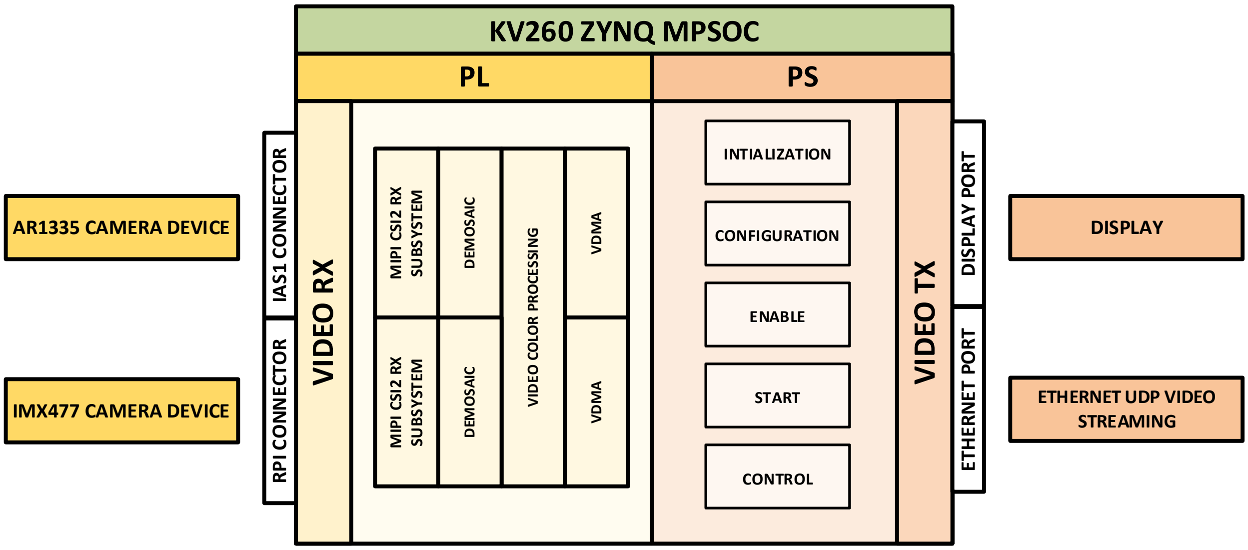


Figure 1 The overall structure of implemented design, which shows video RX and TX video data flow.

The Sony IMX477 12.5MP sensor is a 1/2.3-inch CMOS digital image sensor with an active imaging pixel array of 4056H x 3040V 30fps@max resolution. This sensor is configured by I2C interface and has MIPI output interface that is connected to the MIPI CSI-2 RX subsystem inside.

The AR 1335 13.0MP sensor is a 1/3.2-inch CMOS digital image sensor with an active imaging pixel array of 4208H x 3120V 30fps@max resolution. This sensor is configured by I2C interface and has MIPI output interface that is connected to the MIPI CSI-2 RX subsystem inside.

The MIPI CSI-2 receiver subsystem includes a MIPI D-PHY core and that connects 2 data lanes for imx477 camera device and 4 lanes for AR1335 camera device. The subsystem captures video stream frames from the AR1335 camera and IMX477 camera in RAW10 format.

The Demosaic module converts bayer pattern input frame to RGB color frame.

The VCP custom module apply various filters, color space conversion and K-Mean color clusters. It also applies image enhancement control to improve image quality which include contrast, brightness, saturation, white/black balance and rgb gain through axi4 lite configuration registers.

VDMA (Video Direct Memory Access) transfer video streaming to and from external memory and operate under software control.

The DP (display port) in PS source (live input) video data from PL.

VIVADO BLOCK DIAGRAM IP CUSTOMIZATION

ZYNQ ULTRASCALE+ MPSoC	SETTINGS
I/O CONFIGURATION	
DPAUX	MIO 27 TO MIO 30
DisplayPort Lane Selection	Dual Lower
CLOCK CONFIGURATION	
PL Fabric Output Clock PL0	IOP 100 MHz
DisplayPort Reference Frequency	27
PS-PL CONFIGURATION	
PS-PL Interface: AXI HPM0 FPD	Select
PS-PL Interface: AXI HPM0 FPD Data Width	128
PS-PL Interface: AXI HPM1 FPD	Select
PS-PL Interface: AXI HPM1 FPD Data Width	128
PS-PL Interface: Slave Interface AXI HPC0 FPD	Select
PS-PL Interface: Slave Interface AXI HPC0 FPD Data Width	128

MIPI CSI2 RX SUBSYSTEM 1	SETTINGS
Pixel Format	RAW10
Serial Data Lanes	2
Line Rate	2500 Mbps
CSI2 Controller Register Interface	Select
Filter User Defines Data Types	Select
Line Buffer Defined Data Types	4096
Allowed VC	All
Pixel Per Clock	1
TUSER Width	1
Enable CRC	Select

MIPI CSI2 RX SUBSYSTEM 2	SETTINGS
Pixel Format	RAW10
Serial Data Lanes	4
Line Rate	2000 Mbps
CSI2 Controller Register Interface	Select
Filter User Defines Data Types	Select
Line Buffer Defined Data Types	4096
Allowed VC	All
Pixel Per Clock	1
TUSER Width	1
Enable CRC	Select

RESOLUTIONS									V_CLK
WIDTH	**HEIGHT**	**MODE**	**HTOTAL**	**HBLANK**	**VTOTAL**	**VBLANK**	**HFREQ**	**VFREQ**	**CLOCK**
640	480	PROGRESSIVE	800	160	525	45.0	31.469	59.940	25.175
720	480	PROGRESSIVE	858	138	525	45.0	31.469	59.940	27.000
720	480	PROGRESSIVE	858	138	525	45.0	31.469	59.940	27.000
1280	720	PROGRESSIVE	1650	370	750	30.0	45.000	60.000	74.250
1920	1080	INTERLACED	2200	280	1125	22.5	33.750	60.000	72.250
1440	480	INTERLACED	1716	276	525	22.5	15.734	59.940	27.000
1440	480	INTERLACED	1716	276	525	22.5	15.734	59.940	27.000

1440	240	PROGRESSIVE	1716	276	262	22.0	15.734	60.054	27.000
1440	240	PROGRESSIVE	1716	276	262	22.0	15.734	59.826	27.000
2880	480	INTERLACED	3432	552	525	22.5	15.734	59.940	54.000
2880	480	INTERLACED	3432	552	525	22.5	15.734	59.940	54.000
2880	240	PROGRESSIVE	3432	552	262	22.0	15.734	60.054	54.000
2880	240	PROGRESSIVE	3432	552	262	22.0	15.734	59.826	54.000
1440	480	PROGRESSIVE	1716	276	525	45.0	31.469	59.940	54.000
1440	480	PROGRESSIVE	1716	276	525	45.0	31.469	59.940	54.000
1920	1080	PROGRESSIVE	2200	280	1125	45.0	67.500	60.000	148.500
720	576	PROGRESSIVE	864	144	625	49.0	31.250	50.000	27.000
720	576	PROGRESSIVE	864	144	625	49.0	31.250	50.000	27.000
1280	720	PROGRESSIVE	1980	700	750	30.0	37.500	50.000	74.250
1920	1080	INTERLACED	2640	720	1125	22.5	28.125	50.000	74.250
1440	576	INTERLACED	1728	288	625	24.5	15.625	50.000	27.000
1440	576	INTERLACED	1728	288	625	24.5	15.625	50.000	27.000
1440	288	PROGRESSIVE	1728	288	312	24.0	15.625	50.080	27.000
1440	288	PROGRESSIVE	1728	288	313	25.0	15.625	49.920	27.000
2880	576	INTERLACED	3456	576	625	24.5	15.625	50.000	54.000
2880	576	INTERLACED	3456	576	625	24.5	15.625	50.000	54.000
2880	288	PROGRESSIVE	3456	576	312	24.0	15.625	50.080	54.000
2880	288	PROGRESSIVE	3456	576	313	25.0	15.625	49.920	54.000
1440	576	PROGRESSIVE	1728	288	625	49.0	31.250	50.000	54.000
1440	576	PROGRESSIVE	1728	288	625	49.0	31.250	50.000	54.000
1920	1080	PROGRESSIVE	2640	720	1125	45.0	56.250	50.000	148.500
1920	1080	PROGRESSIVE	2750	830	1125	45.0	27.000	24.000	74.250
1920	1080	PROGRESSIVE	2640	720	1125	45.0	28.125	25.000	74.250
1920	1080	PROGRESSIVE	2200	280	1125	45.0	33.750	30.000	74.250
2880	480	PROGRESSIVE	3432	552	525	45.0	31.469	59.940	108.500
2880	480	PROGRESSIVE	3432	552	525	45.0	31.469	59.940	108.500
2880	576	PROGRESSIVE	3456	576	625	49.0	31.250	50.000	108.000
2880	576	PROGRESSIVE	3456	576	625	49.0	31.250	50.000	108.000
1920	1080	INTERLACED	2304	384	1250	85.0	31.250	50.000	72.000
1920	1080	INTERLACED	2640	720	1125	22.5	56.250	100.000	148.500
1280	720	PROGRESSIVE	1980	700	750	30.0	75.000	100.000	148.500
720	576	PROGRESSIVE	864	144	625	49.0	62.500	100.000	54.000
720	576	PROGRESSIVE	864	144	625	49.0	62.500	100.000	54.000
1440	576	INTERLACED	1728	288	625	24.5	31.250	100.000	54.000
1440	576	INTERLACED	1728	288	625	24.5	31.250	100.000	54.000
1920	1080	INTERLACED	2200	280	1125	22.5	67.500	120.000	148.500
1280	720	PROGRESSIVE	1650	370	750	30.0	90.000	120.000	148.500
720	480	PROGRESSIVE	858	138	525	45.0	62.937	119.880	54.000
720	480	PROGRESSIVE	858	138	525	45.0	62.937	119.880	54.000
1440	480	INTERLACED	1716	276	525	22.5	31.469	119.880	54.000
1440	480	INTERLACED	1716	276	525	22.5	31.469	119.880	54.000
720	576	PROGRESSIVE	864	144	625	49.0	125.000	200.000	108.000
720	576	PROGRESSIVE	864	144	625	49.0	125.000	200.000	108.000
1440	576	INTERLACED	1728	288	625	24.5	62.500	200.000	108.000
1440	576	INTERLACED	1728	288	625	24.5	62.500	200.000	108.000
720	480	PROGRESSIVE	858	138	525	45.0	125.874	239.760	108.000
720	480	PROGRESSIVE	858	138	525	45.0	125.874	239.760	108.000
1440	480	INTERLACED	1716	276	525	22.5	62.937	239.760	108.000
1440	480	INTERLACED	1716	276	525	22.5	62.937	239.760	108.000
1280	720	PROGRESSIVE	3300	2020	750	30	18	24.0003	59.4
1280	720	PROGRESSIVE	3960	2680	750	30	18.75	25	74.25
1280	720	PROGRESSIVE	3300	2020	750	30	22.5	30.0003	74.25
1920	1080	PROGRESSIVE	2200	280	1125	45	135	120.003	297

1920	1080	PROGRESSIVE	2640	720	1125	45	112.5	100	297
1280	720	PROGRESSIVE	3300	2020	750	30	18	24.0003	59.4
1280	720	PROGRESSIVE	3960	2680	750	30	18.75	25	74.25
1280	720	PROGRESSIVE	3300	2020	750	30	22.5	30.0003	74.25
1280	720	PROGRESSIVE	1980	700	750	30	37.5	50	74.25
1280	720	PROGRESSIVE	1650	370	750	30	45	60.0003	74.25
1280	720	PROGRESSIVE	1980	700	750	30	75	100	148.5
1280	720	PROGRESSIVE	1650	370	750	30	90	120.003	148.5
1920	1080	PROGRESSIVE	2750	830	1125	45	27	24.0003	74.25
1920	1080	PROGRESSIVE	2640	720	1125	45	28.125	25	74.25
1920	1080	PROGRESSIVE	2200	280	1125	45	33.75	30.0003	74.25
1920	1080	PROGRESSIVE	2640	720	1125	45	56.25	50	148.5
1920	1080	PROGRESSIVE	2200	280	1125	45	67.5	60.0003	148.5
1920	**1080**	**PROGRESSIVE**	**2640**	**720**	**1125**	**45**	**112.5**	**100**	**297**
1920	**1080**	**PROGRESSIVE**	**2200**	**280**	**1125**	**45**	**135**	**120.003**	**297**
1680	720	PROGRESSIVE	3300	1620	750	30	18	24.0003	59.4
1680	720	PROGRESSIVE	3168	1488	750	30	18.75	25	59.4
1680	720	PROGRESSIVE	2640	960	750	30	22.5	30.0003	59.4
1680	720	PROGRESSIVE	2200	520	750	30	37.5	50	82.5
1680	720	PROGRESSIVE	2200	520	750	30	45	60.0003	99
1680	720	PROGRESSIVE	2000	320	825	105	82.5	100	165
1680	720	PROGRESSIVE	2000	320	825	105	99	120.003	198
2560	1080	PROGRESSIVE	3750	1190	1100	20	26.4	24.0003	99
2560	1080	PROGRESSIVE	3200	640	1125	45	28.125	25	90
2560	1080	PROGRESSIVE	3520	960	1125	45	33.75	30.0003	118.8
2560	1080	PROGRESSIVE	3300	740	1125	45	56.25	50	185.625
2560	1080	PROGRESSIVE	3000	440	1100	20	66	60.0003	198
2560	1080	PROGRESSIVE	2970	410	1250	170	125	100	371.25
2560	1080	PROGRESSIVE	3300	740	1250	170	150	120.003	495
3840	**2160**	**PROGRESSIVE**	**5500**	**1660**	**2250**	**90**	**54**	**24.0003**	**297**
3840	**2160**	**PROGRESSIVE**	**5280**	**1440**	**2250**	**90**	**56.25**	**25**	**297**
3840	**2160**	**PROGRESSIVE**	**4400**	**560**	**2250**	**90**	**67.5**	**30.0003**	**297**
3840	2160	PROGRESSIVE	5280	1440	2250	90	112.5	50	594
3840	2160	PROGRESSIVE	4400	560	2250	90	135	60.0003	594
4096	**2160**	**PROGRESSIVE**	**5500**	**1404**	**2250**	**90**	**54**	**24.0003**	**297**
4096	**2160**	**PROGRESSIVE**	**5280**	**1184**	**2250**	**90**	**56.25**	**25**	**297**
4096	**2160**	**PROGRESSIVE**	**4400**	**304**	**2250**	**90**	**67.5**	**30.0003**	**297**
4096	2160	PROGRESSIVE	5280	1184	2250	90	112.5	50	594
4096	2160	PROGRESSIVE	4400	304	2250	90	135	60.0003	594
3840	**2160**	**PROGRESSIVE**	**5500**	**1660**	**2250**	**90**	**54**	**24.0003**	**297**
3840	**2160**	**PROGRESSIVE**	**5280**	**1440**	**2250**	**90**	**56.25**	**25**	**297**
3840	**2160**	**PROGRESSIVE**	**4400**	**560**	**2250**	**90**	**67.5**	**30.0003**	**297**
3840	2160	PROGRESSIVE	5280	1440	2250	90	112.5	50	594
3840	2160	PROGRESSIVE	4400	560	2250	90	135	60.0003	594
1280	720	PROGRESSIVE	2500	1220	750	30	36	48.0003	90
1280	720	PROGRESSIVE	2500	1220	750	30	36	48.0003	90
1680	720	PROGRESSIVE	2750	1070	750	30	36	48.0003	99
1920	1080	PROGRESSIVE	2750	830	1125	45	54	48.0003	148.5
1920	1080	PROGRESSIVE	2750	830	1125	45	54	48.0003	148.5
2560	1080	PROGRESSIVE	3750	1190	1100	20	52.8	48.0003	198
3840	2160	PROGRESSIVE	5500	1660	2250	90	108	48.0003	594
4096	2160	PROGRESSIVE	5500	1404	2250	90	108	48.0003	594
3840	2160	PROGRESSIVE	5500	1660	2250	90	108	48.0003	594
3840	2160	PROGRESSIVE	5280	1440	2250	90	225	100	1188
3840	2160	PROGRESSIVE	4400	560	2250	90	270	120.003	1188
3840	2160	PROGRESSIVE	5280	1440	2250	90	225	100	1188

3840	2160	PROGRESSIVE	4400	560	2250	90	270	120.003	1188
5120	2160	PROGRESSIVE	7500	2380	2200	40	52.8	24.0003	396
5120	2160	PROGRESSIVE	7200	2080	2200	40	55	25	396
5120	2160	PROGRESSIVE	6000	880	2200	40	66	30.0003	396
5120	2160	PROGRESSIVE	6250	1130	2475	315	118.8	48.0003	742.5
5120	2160	PROGRESSIVE	6600	1480	2250	90	112.5	50	742.5
5120	2160	PROGRESSIVE	5500	380	2250	90	135	60.0003	742.5
5120	2160	PROGRESSIVE	6600	1480	2250	90	225	100	1485
5120	2160	PROGRESSIVE	5500	380	2250	90	270	120.003	1485
7680	4320	PROGRESSIVE	11000	3320	4500	180	108	24.0003	1188
7680	4320	PROGRESSIVE	10800	3120	4400	80	110	25	1188
7680	4320	PROGRESSIVE	9000	1320	4400	80	132	30.0003	1188
7680	4320	PROGRESSIVE	11000	3320	4500	180	216	48.0003	2376
7680	4320	PROGRESSIVE	10800	3120	4400	80	220	50	2376
7680	4320	PROGRESSIVE	9000	1320	4400	80	264	60.0003	2376
7680	4320	PROGRESSIVE	10560	2880	4500	180	450	100	4752
7680	4320	PROGRESSIVE	8800	1120	4500	180	540	120.003	4752
7680	4320	PROGRESSIVE	11000	3320	4500	180	108	24.0003	1188
7680	4320	PROGRESSIVE	10800	3120	4400	80	110	25	1188
7680	4320	PROGRESSIVE	9000	1320	4400	80	132	30.0003	1188
7680	4320	PROGRESSIVE	11000	3320	4500	180	216	48.0003	2376
7680	4320	PROGRESSIVE	10800	3120	4400	80	220	50	2376
7680	4320	PROGRESSIVE	9000	1320	4400	80	264	60.0003	2376
7680	4320	PROGRESSIVE	10560	2880	4500	180	450	100	4752
7680	4320	PROGRESSIVE	8800	1120	4500	180	540	120.003	4752
10240	4320	PROGRESSIVE	12500	2260	4950	630	118.8	24.0003	1485
10240	4320	PROGRESSIVE	13500	3260	4400	80	110	25	1485
10240	4320	PROGRESSIVE	11000	760	4500	180	135	30.0003	1485
10240	4320	PROGRESSIVE	12500	2260	4950	630	237.6	48.0003	2970
10240	4320	PROGRESSIVE	13500	3260	4400	80	220	50	2970
10240	4320	PROGRESSIVE	11000	760	4500	180	270	60.0003	2970
10240	4320	PROGRESSIVE	13200	2960	4500	180	450	100	5940
10240	4320	PROGRESSIVE	11000	760	4500	180	540	120.003	5940
4096	2160	PROGRESSIVE	5280	1184	2250	90	225	100	1188
4096	2160	PROGRESSIVE	4400	304	2250	90	270	120.003	1188

ARCHITECTURE

The Video Color Processing module provides a modular expandable interface for video pipeline.

General architecture of VCP consists of axi4 lite for configuration and axi4 stream video input and output. It consists of filters, color space conversion, k-mean video color quantization and color gain matrix.

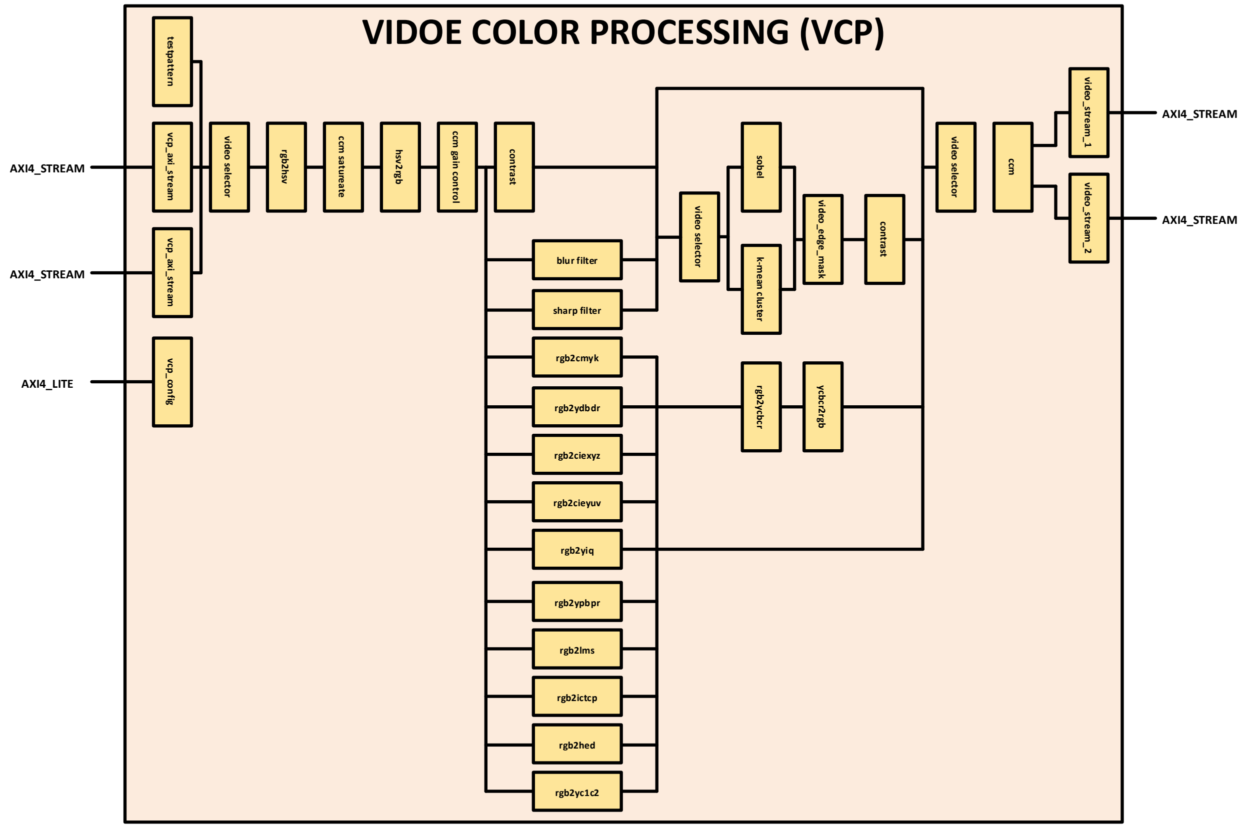


Figure 2: Video color processing architecture

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].

BANDWIDTH AND DATA RATE CALCULATION

The design is divided into three clock domains. The frequency of the output clock from ZYNQ UltraScale+ MPSoC is set to 100MHz, and it than fed to the clock generator that create all other clocks in the PL design. A 100Mhz input clock is fed to generator block which than generate three clock outputs 300MHz and 297MHz and 200MHz. Pixel clock is nominally the pixel clock rate set according to resolution and refresh rate.

Following are the clocks used in this design:

- Clock 300MHz for video Axi4 video configuration and process.
- Clock 200MHz for MIPI DPHY video input.
- Clock 297MHz for video output.

PIXEL CLOCK FREQUENCY

Pixel Clock Frequency = Total Horizontal Samples * Total Vertical Lines * Refresh Rate

VIDEO CLOCK

The following equation is used to calculate the minimum required Video Clock:

$$\text{video\_aclk (MHz)} = \frac{\text{Line Rate}\left(\frac{\text{Mb}}{\text{s}}\right) * \text{Data Lanes}}{\text{Pixels per Clock} * \text{ Number of Bits Per Pixel}}$$

TOTAL DATA RATE

The following equation is used to calculate the minimum required Total Data Rate:

$$\text{Total Data Rate (Bandwidth)} = \text{ Pixel Clock Frequency } * \text{ Pixel Size (in bits)}$$

BANDWIDTH

The bandwidth of a given video format is a product of the Pixel Clock Frequency and Pixel Size in bits.

Bandwidth = Pixel Clock Frequency * Pixel Size

LINE RATE

The Line Rate is the total data rate (bandwidth) divided by the number of lanes.

Line Rate = Total Data Rate (Bandwidth)/Number of Data Lane.

1920x1080p@60Hz, RAW10, 2-lane

Total Horizontal Samples = 2200,
Total Vertical Lines = 1125
Refresh Rate = 60
Number of Data Lanes = 2
Pixel per clock = 1
Video Clock = 2200 x 1125 x 60 = 148.5 MHz

Bandwidth (Total Data Rate) = 148.5 MHz * 10-bit = 1485 Mbps
Line Rate (Data Rate per Lane) = 1485 Mbps /2-lane = 742.5 Mbps
Video Clock = 742.5 Mbps * 2 / 1*10 = 148.5 MHz

2560x1080p@30Hz, RAW10, 2-lane

Total Horizontal Samples = 3520
Total Vertical Lines = 1125
Pixel Clock Frequency = 3520 x 1125 x 30 = 118.8 MHz
Bandwidth (Total Data Rate) = 118.8 MHz * 10-bit = 1188 Mbps
Line Rate (Data Rate per Lane) = 1188 Mbps /2-lane = 594.0 Mbps

2560x1080p@60Hz, RAW10, 2-lane

Total Horizontal Samples = 3520
Total Vertical Lines = 1125
Pixel Clock Frequency = 3520 x 1125 x 60 = 237.6 MHz
Bandwidth (Total Data Rate) = 237.6MHz * 10-bit = 2376 Mbps
Line Rate (Data Rate per Lane) = 2376 Mbps /2-lane = 1188.0 Mbps

3840x2160@30Hz, RAW8, 4-lane

Total Horizontal Samples = 4400,
Total Vertical Lines = 2250
Pixel Clock Frequency = 4400 x 2250 * 30 = 297 MHz
Bandwidth (Total Data Rate) = 297 MHz * 8-bit = 2376 Mbps
Line Rate (Data Rate per Lane) = 2376 Mbps /4-lane = 594 Mbps

In above calculation, resolution “p” signifies “progressive scan” which display both odd and even lines and where as “i” denotes the use of interlaced scanning, with quality being lower which displays odd and even scan lines as individual fields.

CMOS PIXEL CAPTURE

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

VCP

Prior to the overview of the various filters, it necessary to overview top module and io connections to internal modules. Generic defined in VCP 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.

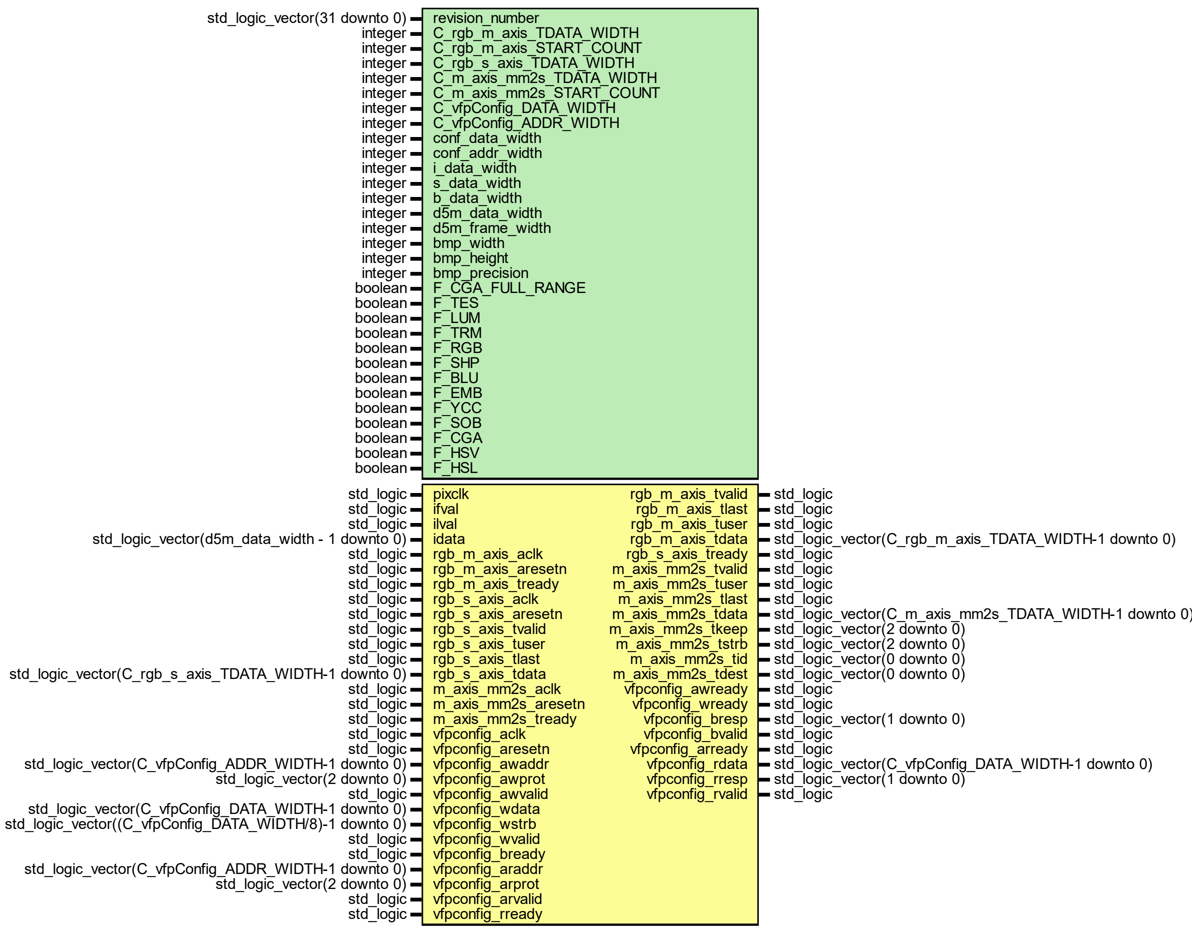


Figure 3

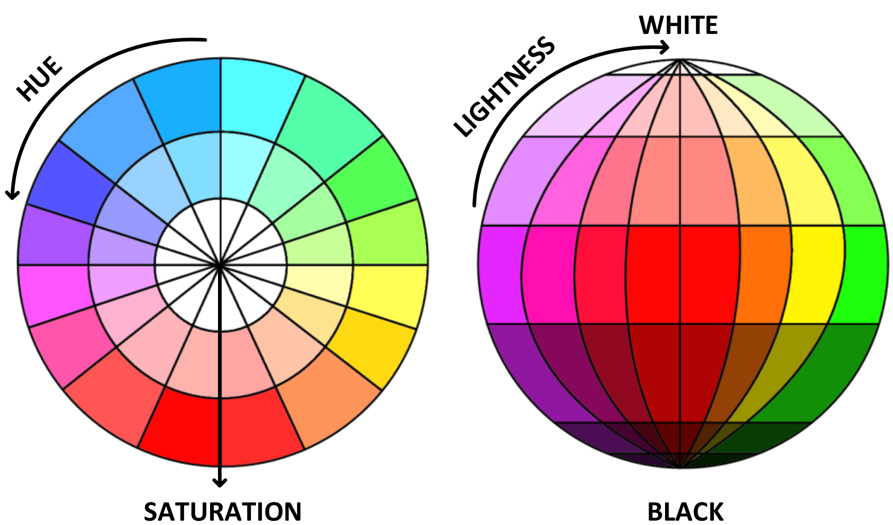
To configure video stream register, axi4-lite protocol is implemented which present a memory map register interface to the processor.

HSL COLOR SPACE

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. Hue color describe the pure color whereas saturate values determine pure color range strength and luminosity describe pure color range. A color with 100% **s**aturation yields purest color and 0% yields grayscale. A color with 0% luminosity is black and 100% luminosity yields bright color.

Value of angles below are hue values on the color wheel.

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



From RGB triplet saturation equation shown below where max and min is calculated of between red, green, and blue channel. It represents the strength of the color and the radius of the cone. RGB consist of equal amount of white light. When max rgb component is subtract with minimum rgb component and divided by max rgb component than grayscale intensity is eliminated. Thus, this new color has no white light and saturated and represent a single wavelength.

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

The rgb colors made stronger by increasing the saturation in hsv color space when converted from rgb to hsv and hsv to rgb. The hue, saturation, and intensity elements control wider color range for image enhancement.

RGB TO HSV

The implemented rgb to hsv color space in video color processing module uses hsv 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.

This module converts rgb color space to hsl color space. First logic calculates maximum and minimum value of rgb values. Hue is calculated by 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.

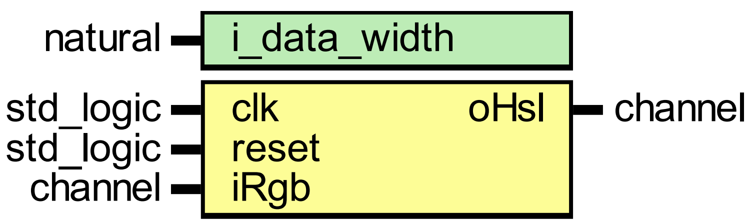


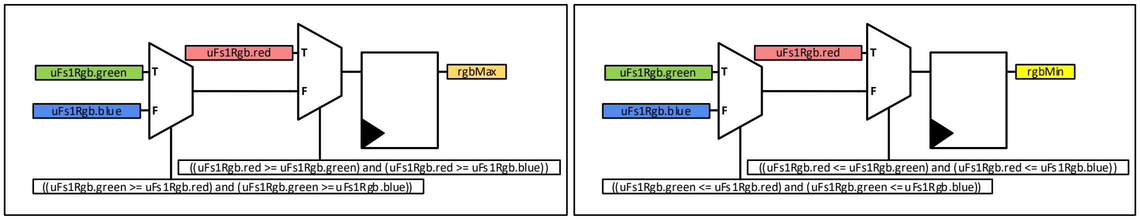
Figure 4

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.

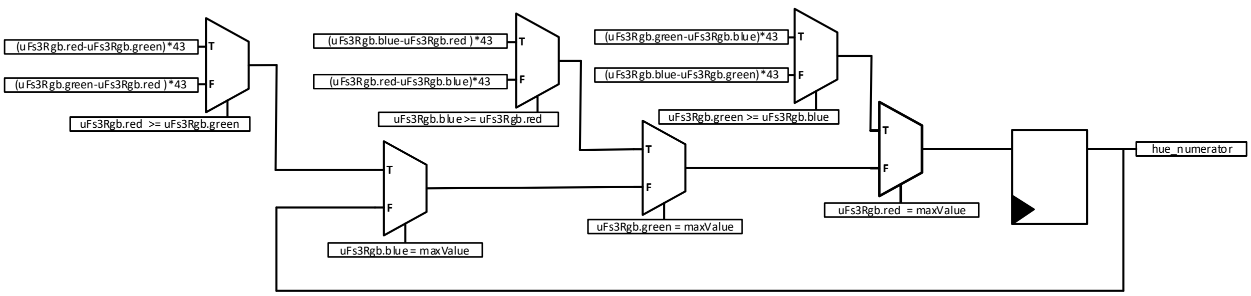


Figure 5: Hue Numerator Logic

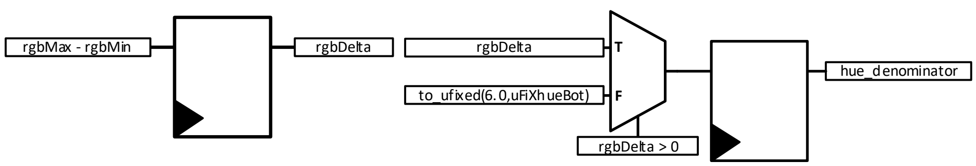


Figure 6: Hue Denominator Logic

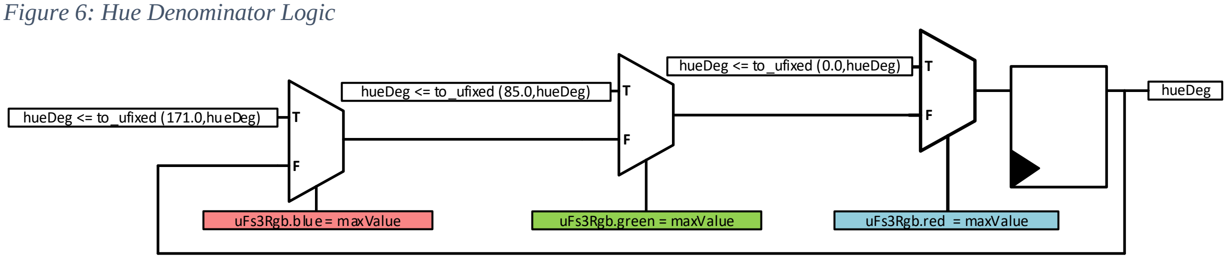


Figure 7: Hue Degree Logic

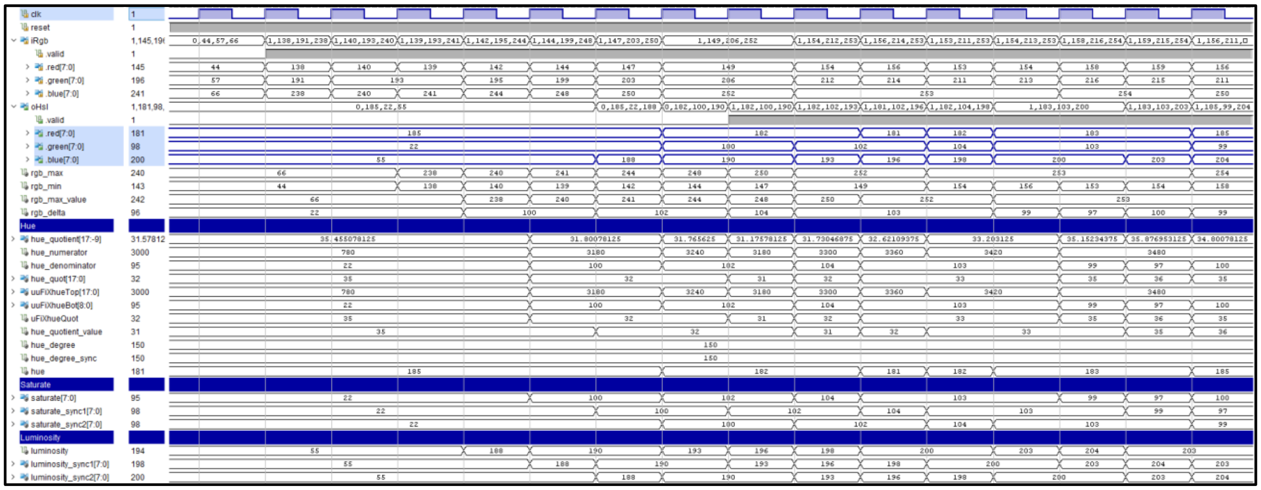


Figure 8 : HSL Filter Wave Diagram

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

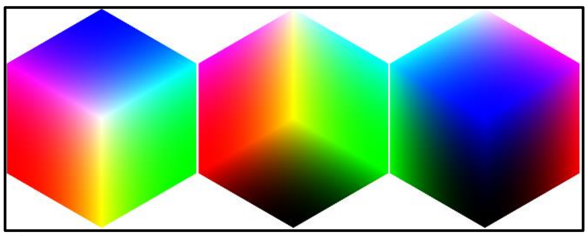


Figure 9

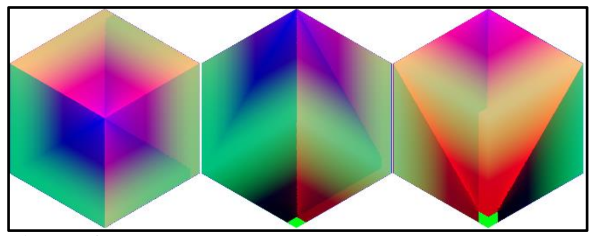


Figure 10

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.

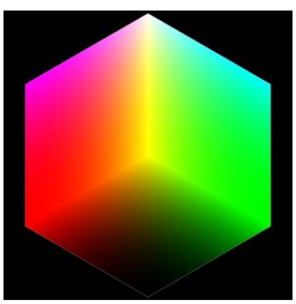


Figure 11: RGB image

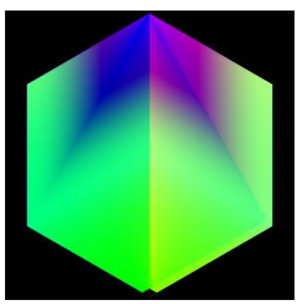


Figure 12: HSL converted image

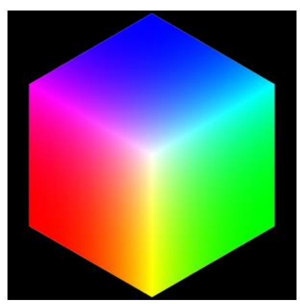


Figure 13: RGB image

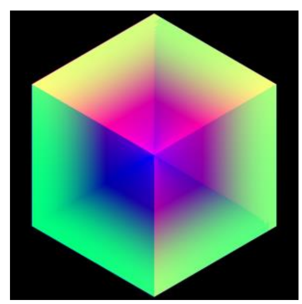


Figure 14: HSL converted image

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

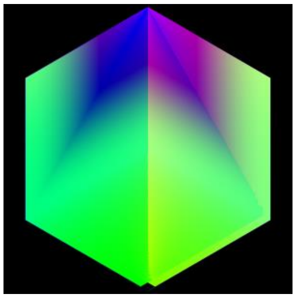


Figure 15: HSL Image

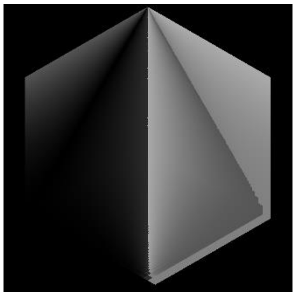


Figure 16: Hue channel

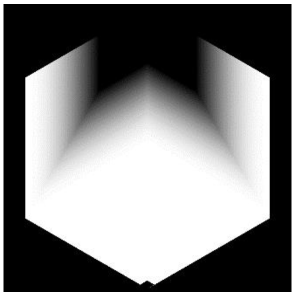


Figure 17: Saturate channel

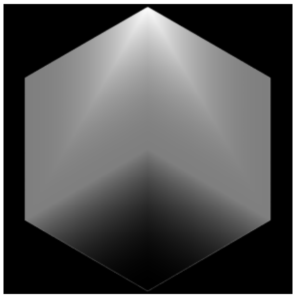


Figure 18: Luminosity channel

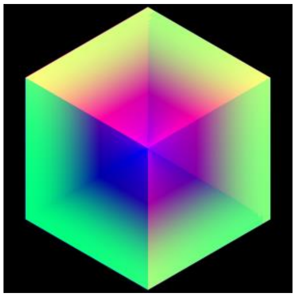


Figure 19: HSL Image

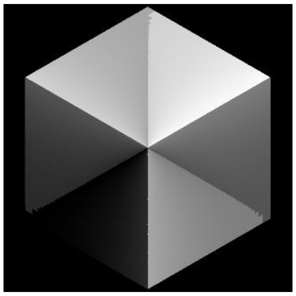


Figure 20: Hue channel

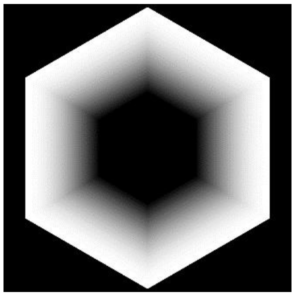


Figure 21: Saturate channel

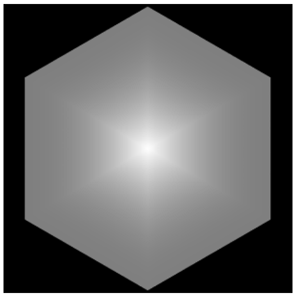
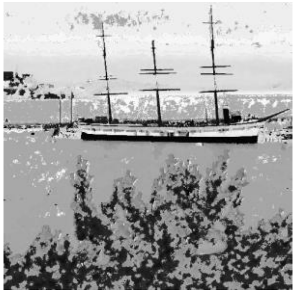
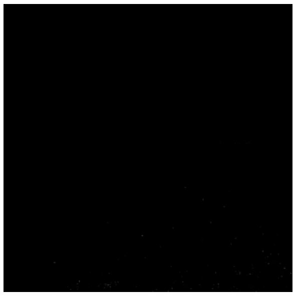


Figure 22: Luminosity channel

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.

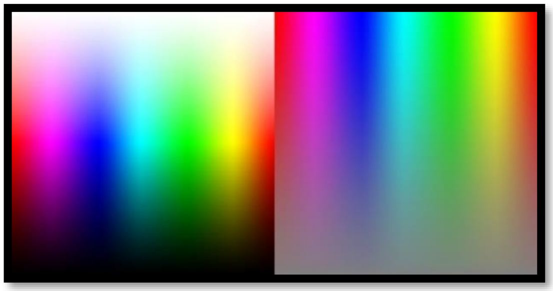


Figure 23

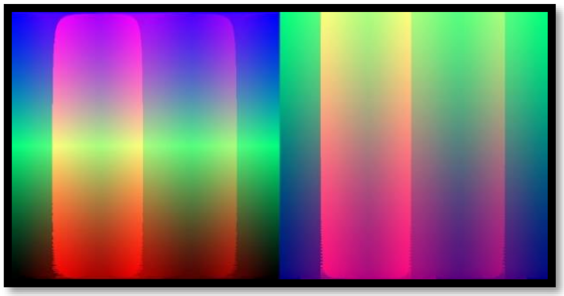


Figure 24

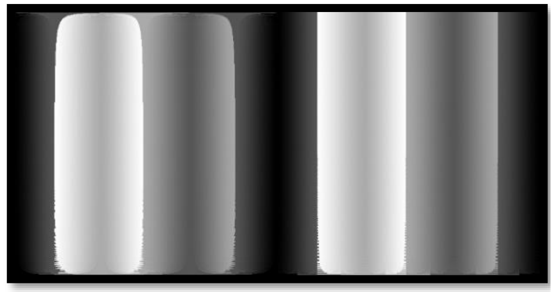


Figure 25

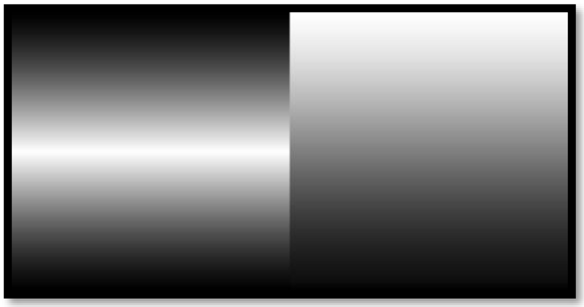


Figure 26

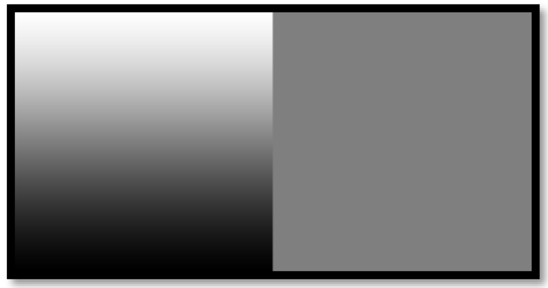


Figure 27

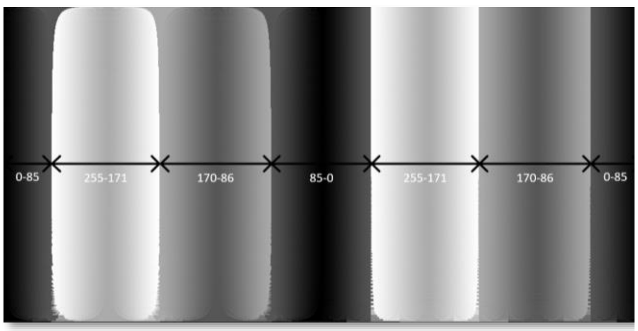


Figure 28

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 TO RGB

The implemented hsv rgb color space in video color processing module uses hsv to rgb conversion 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.

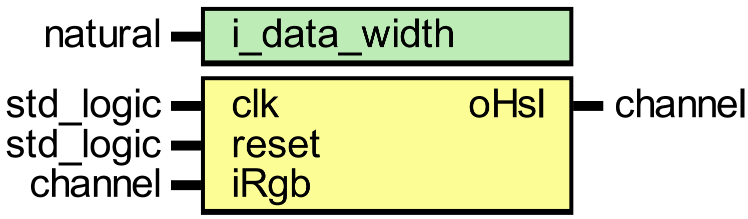


Figure 29

HSV to RGB module convert input hsv 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.

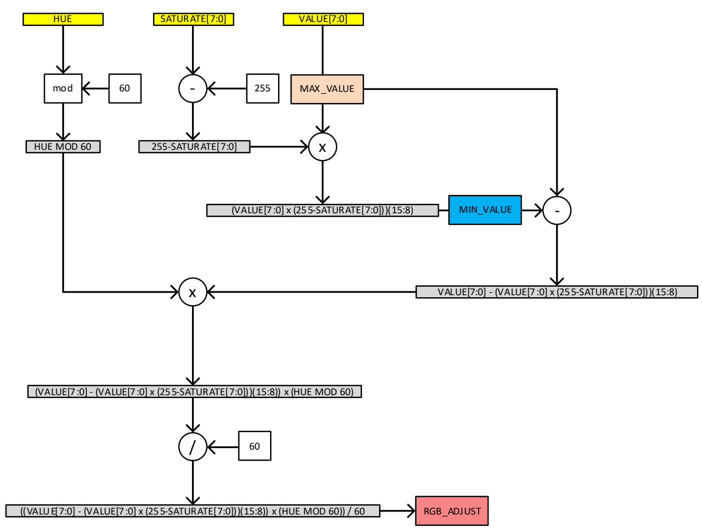


Figure 30

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.

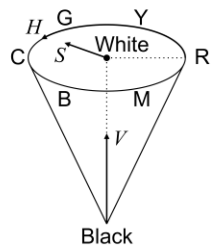


Figure 31

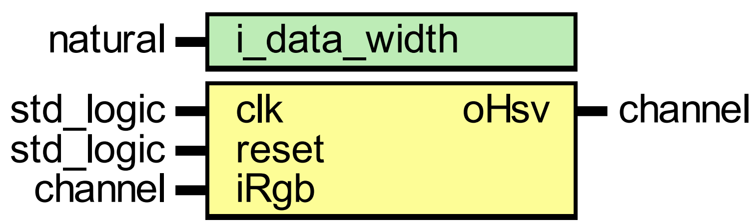


Figure 32

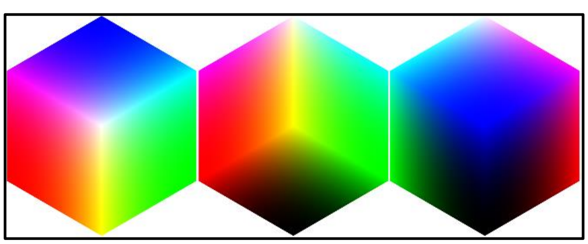


Figure 33

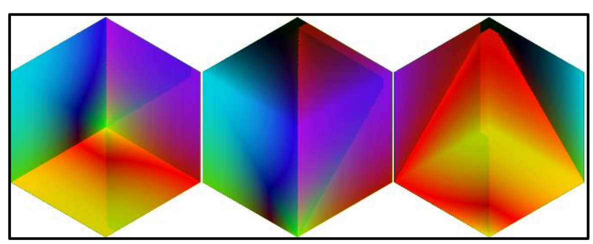


Figure 34

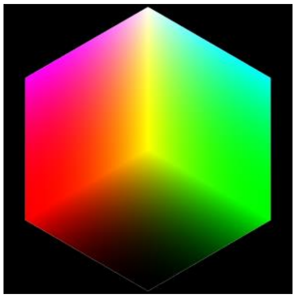


Figure 35

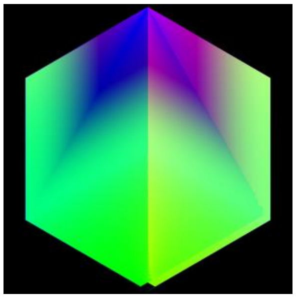


Figure 36

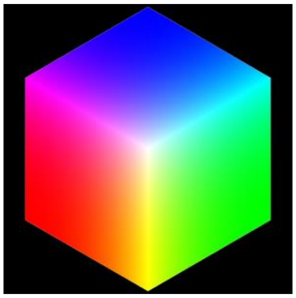


Figure 37

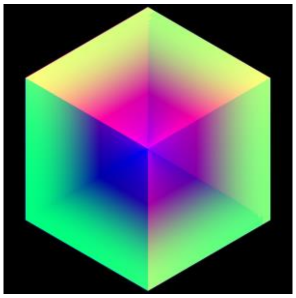


Figure 38

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

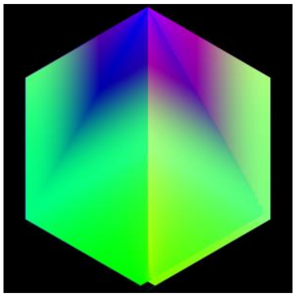


Figure 39

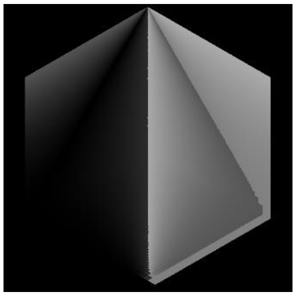


Figure 40

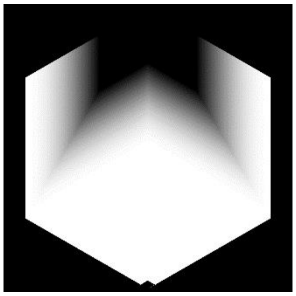


Figure 41

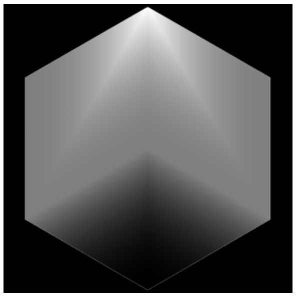


Figure 42

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

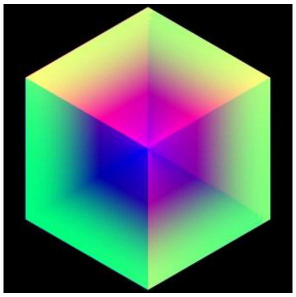


Figure 43

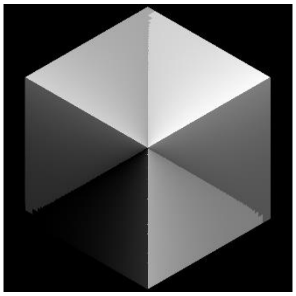


Figure 44

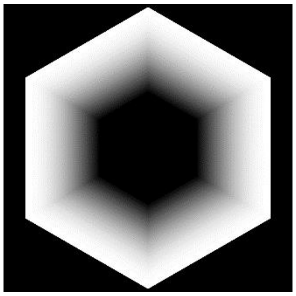


Figure 45

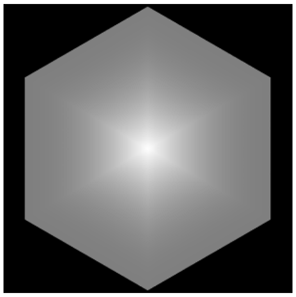


Figure 46

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

Figure 47

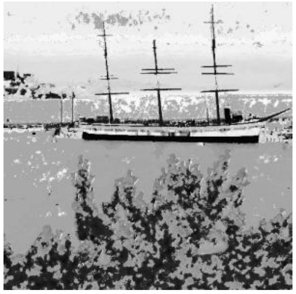


Figure 48

Figure 49

Figure 50

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 51


Figure 52


Figure 53


Figure 54

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

COLORHSL

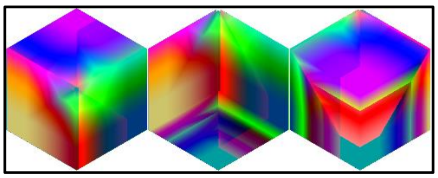



Figure 55


Figure 56


Figure 57


Figure 58

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

HSL_1RANGE

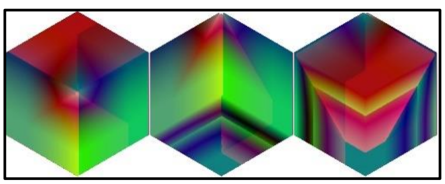


Figure 59

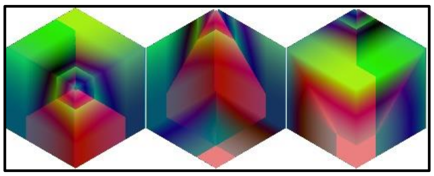
Figure 60

Figure 61

Figure 62

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

HSL_2RANGE



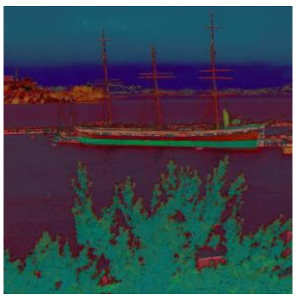


Figure 63

Figure 64

Figure 65

Figure 66

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

HSL_3RANGE

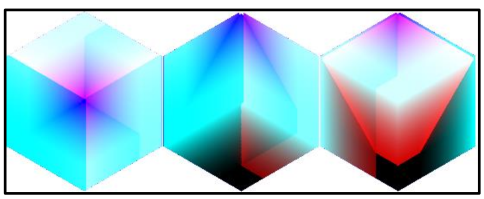


Figure 67

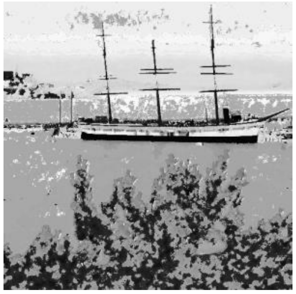


Figure 68

Figure 69

Figure 70

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

HSVL_1RANGE

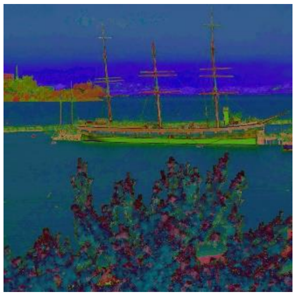


Figure 71

Figure 72

Figure 73

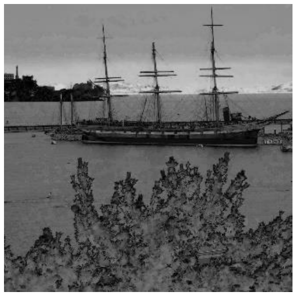
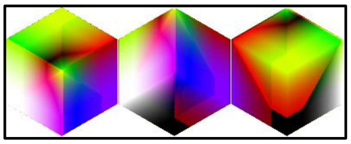


Figure 74

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

HSVL_2RANGE



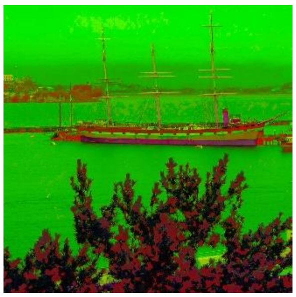


Figure 75

Figure 76

Figure 77

Figure 78

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

HSVL_3RANGE

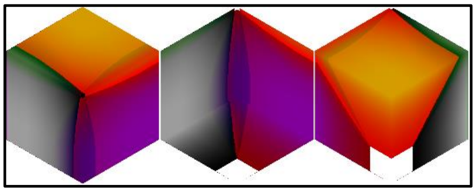


Figure 79

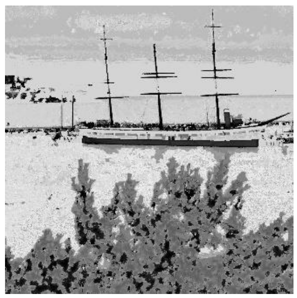


Figure 80

Figure 81

Figure 82

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

HSVL_4RANGE

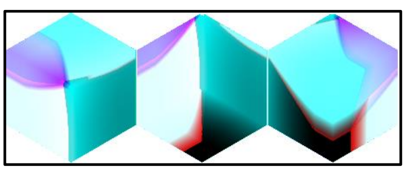


Figure 83

Figure 84

Figure 85

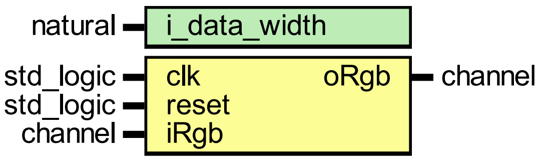
Figure 86

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

CMYK COLOR SPACE

The implemented cmyk to rgb color space in video color processing module uses cmyk to rgb conversion 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.

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.

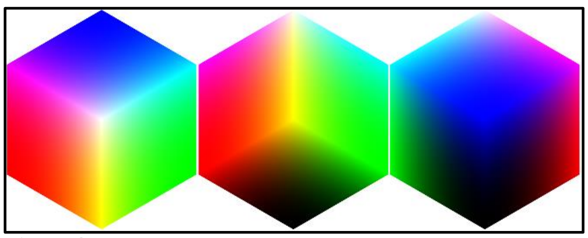


Figure 87

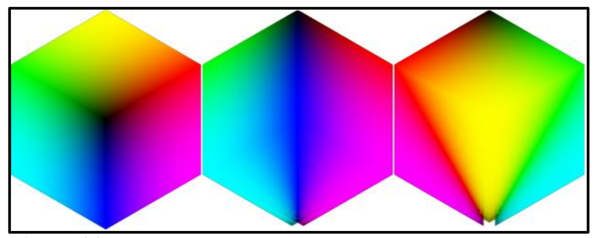


Figure 88

Figure 89

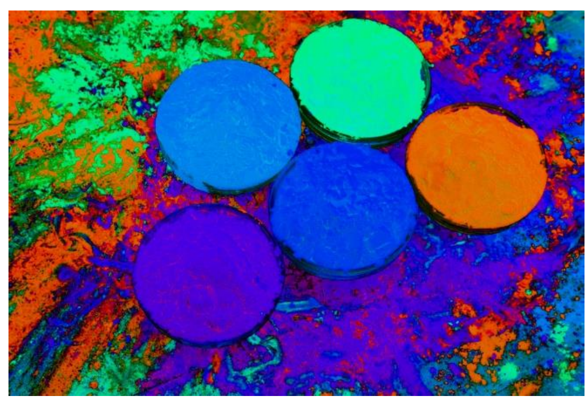


Figure 90

Figure 91

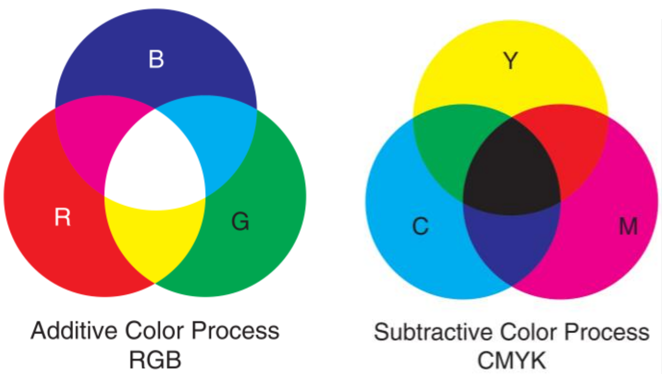
Figure 92

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

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

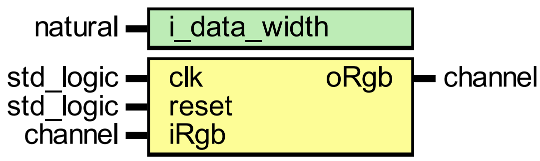
$$C = (1 - R - K)(1 - K)$$
$$M = (1 - G - K)(1 - K)$$
$$Y = (1 - B - K)(1 - K)$$

CMYK color space has four dimensions, and it 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.



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.



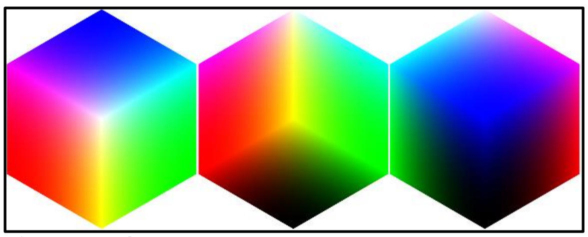


Figure 93 RGB Image

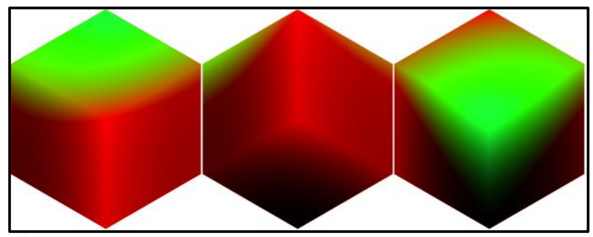


Figure 94 YDBDR Image


Figure 95 RGB Image

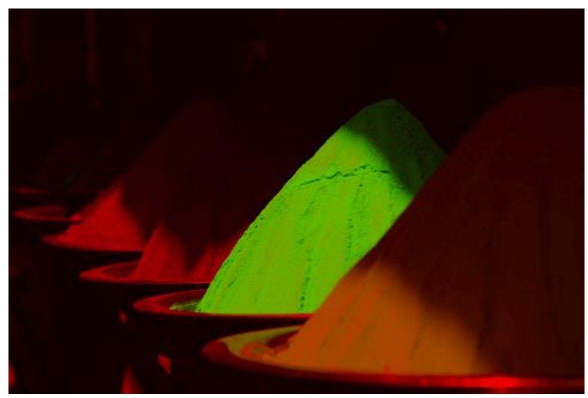


Figure 96 YDBDR Image

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

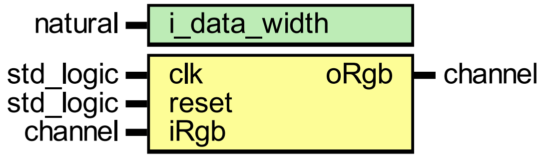
The R, G, B tristimulus values converted to Y, D_B, D_R tristimulus values as follows:

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

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.



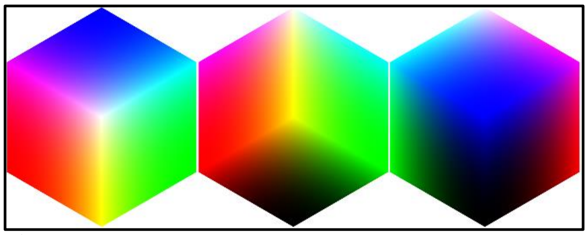


Figure 97

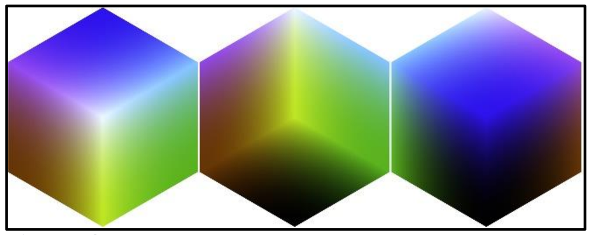


Figure 98

Figure 99

Figure 100

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

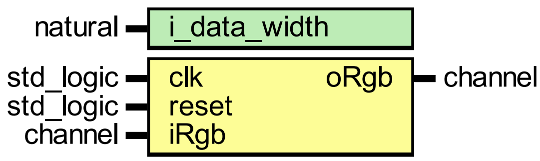
The R, G, B tristimulus values converted to X, Y, Z tristimulus values. The relationship between RGB and the XYZ values expressed by a 3x3 matrix as shown in equation below.

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

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.



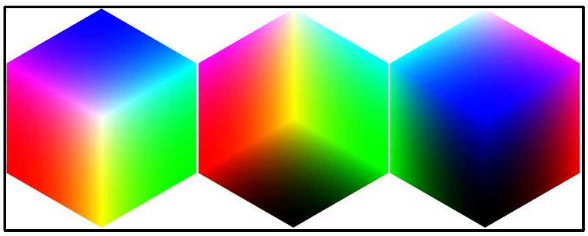


Figure 101

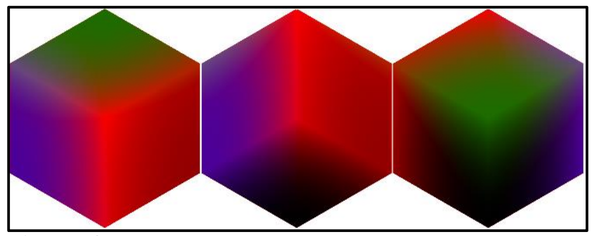


Figure 102

Figure 103

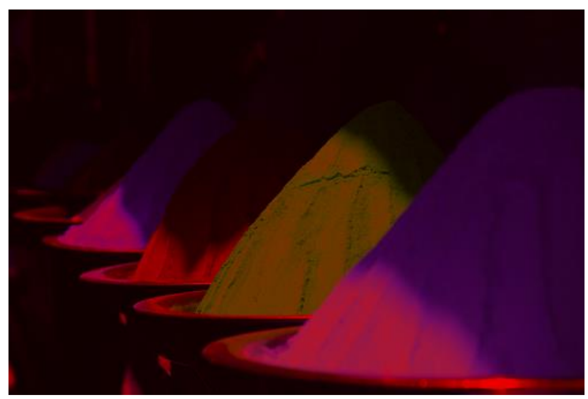


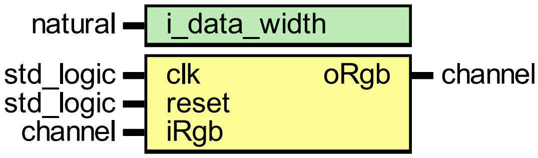
Figure 104

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.



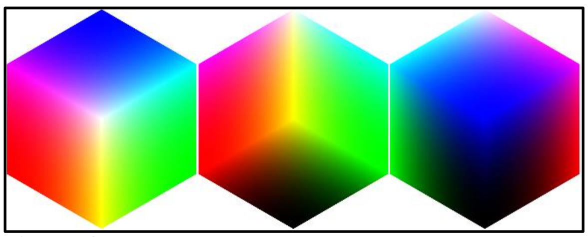


Figure 105

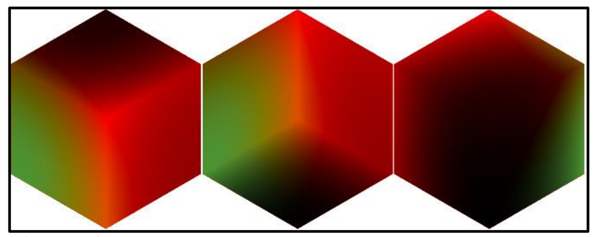


Figure 106


Figure 107


Figure 108

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:

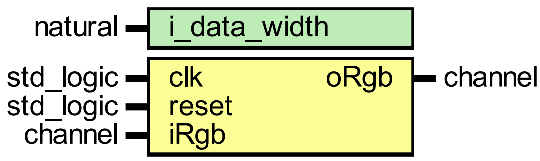
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595 & -0.274 & -0.321 \\ 0.211 & -0.522 & -0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

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.



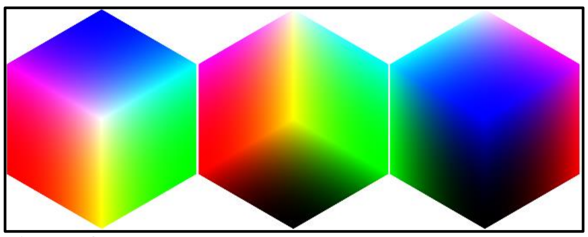


Figure 109

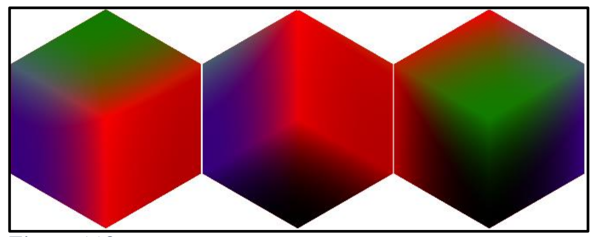


Figure 110


Figure 111

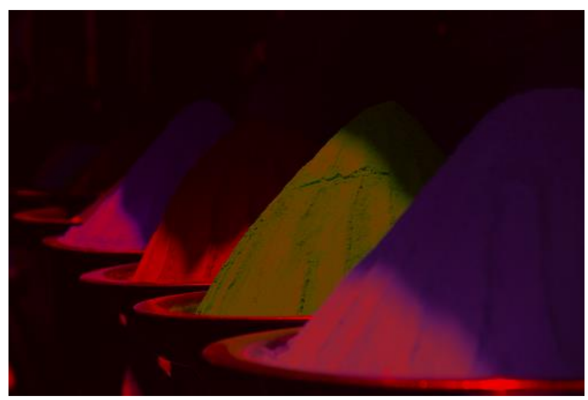


Figure 112

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.

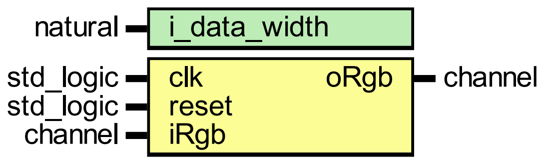


Figure 113

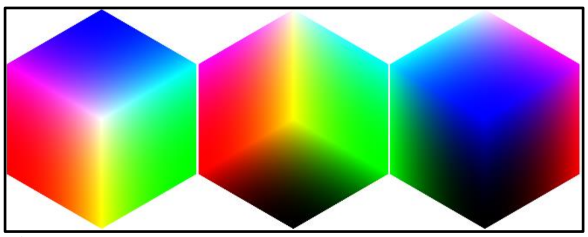


Figure 114

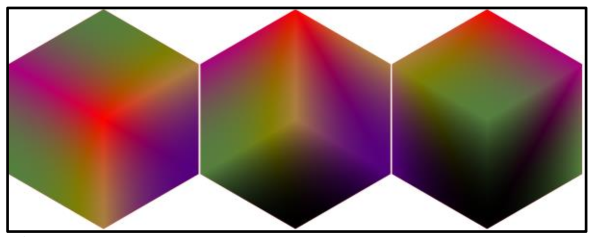


Figure 115

Figure 116

Figure 117

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

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

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

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.

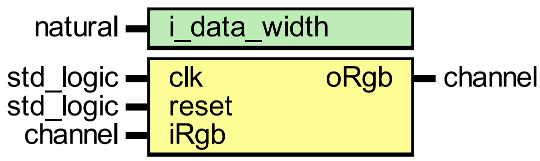


Figure 118

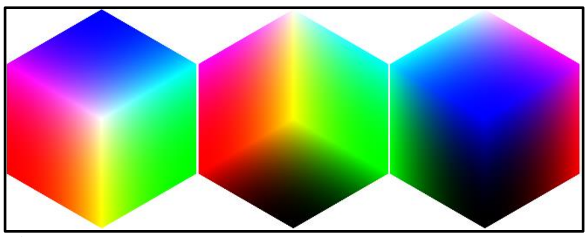


Figure 119

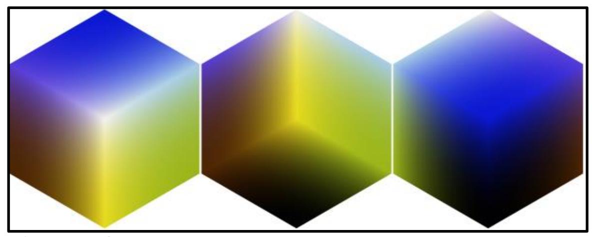


Figure 120

Figure 121

Figure 122

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.

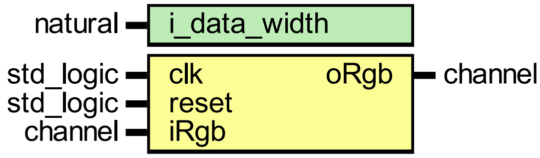


Figure 123

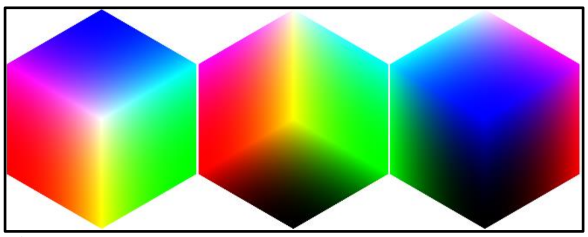


Figure 124

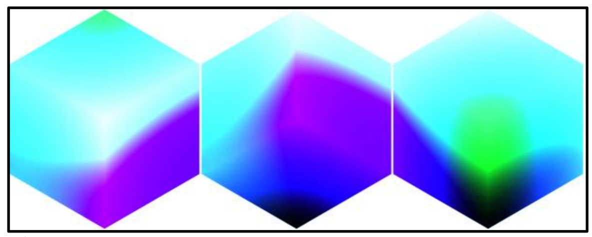


Figure 125

Figure 126

Figure 127

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.

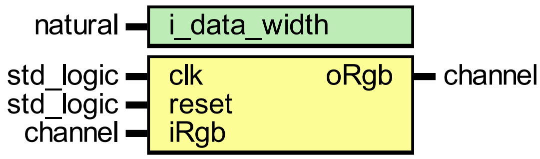


Figure 128

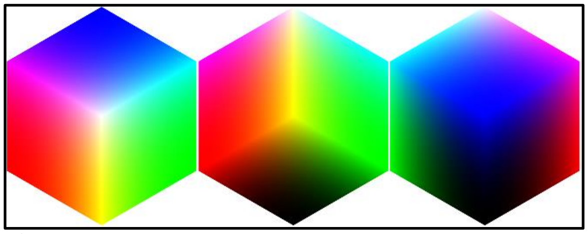


Figure 129

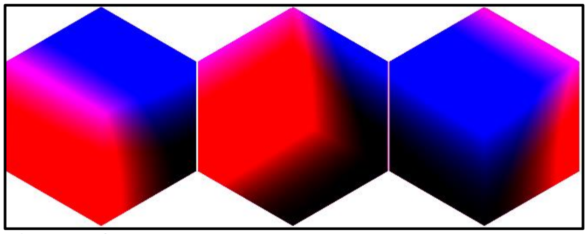


Figure 130

Figure 131

Figure 132

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.

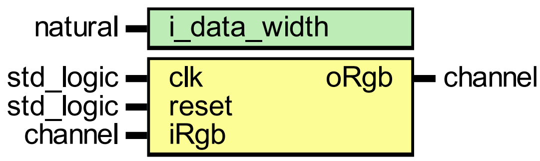


Figure 133

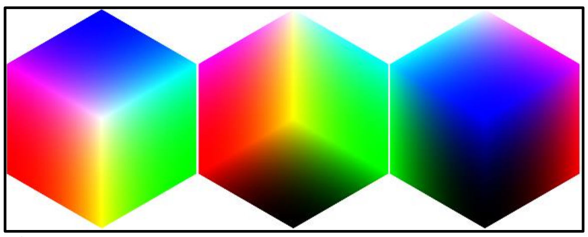


Figure 134

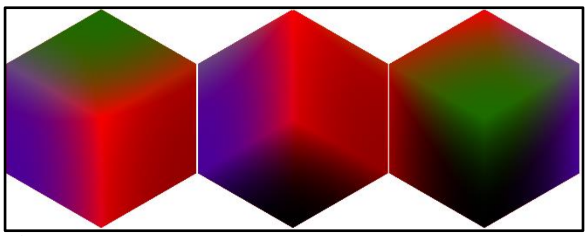


Figure 135

Figure 136

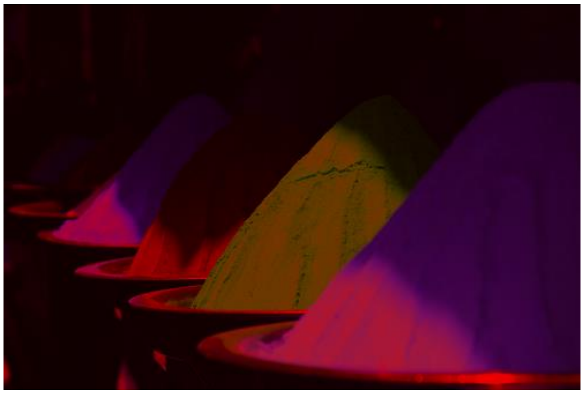


Figure 137

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

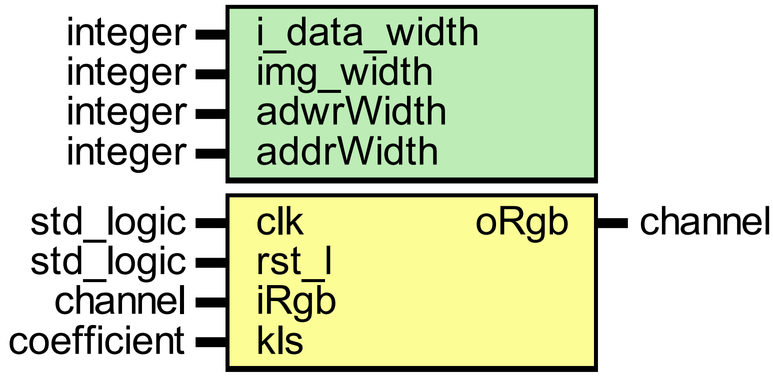
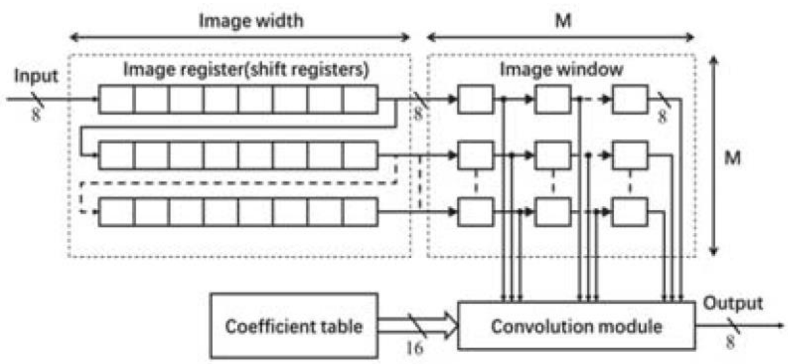


Figure 138



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

BLUR FILTER

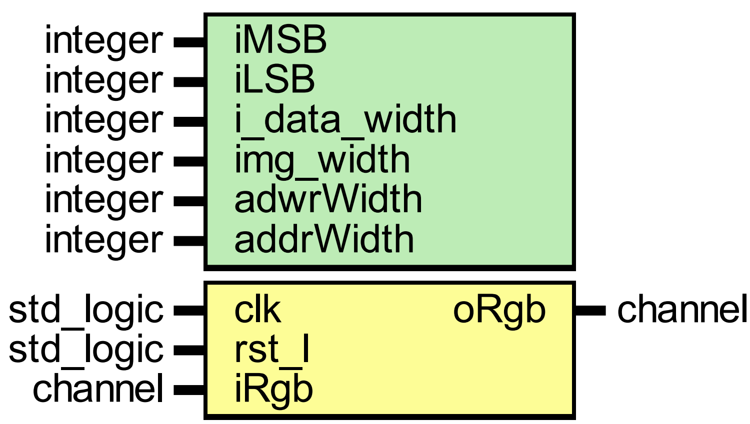
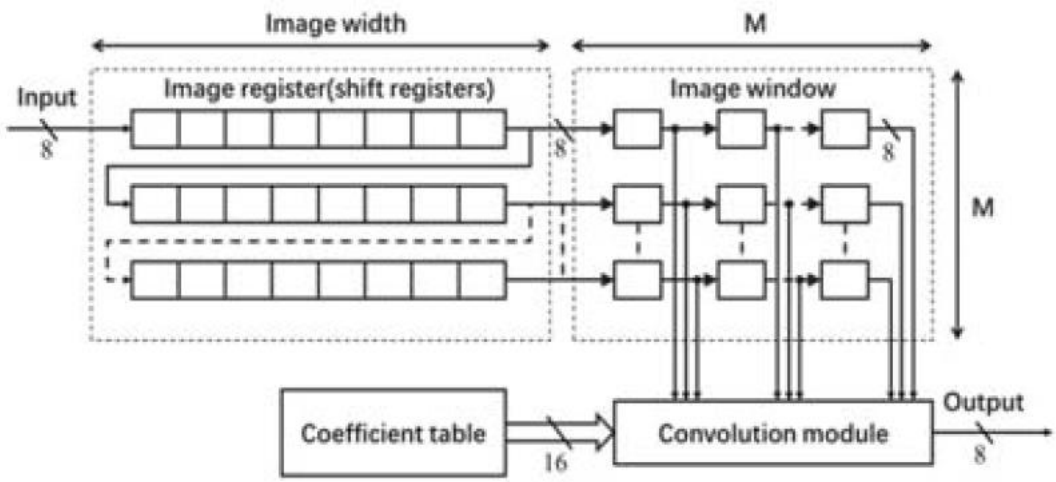
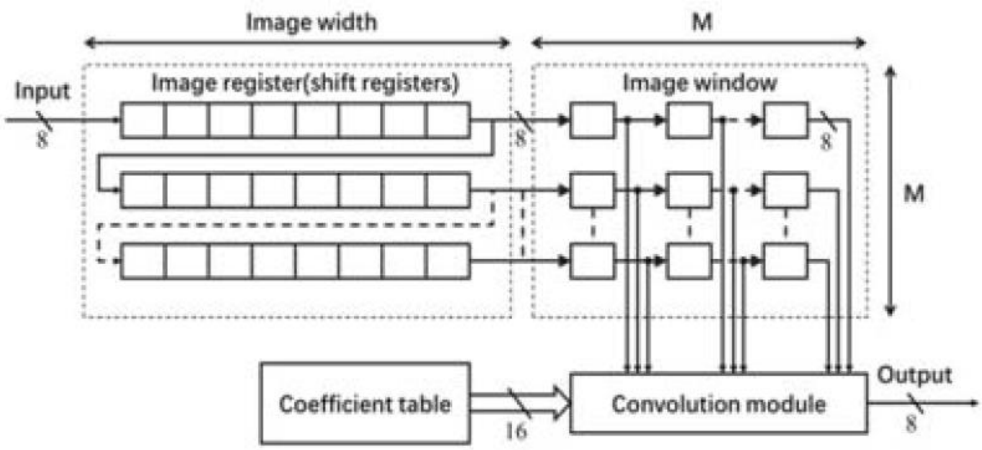


Figure 139



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	∑

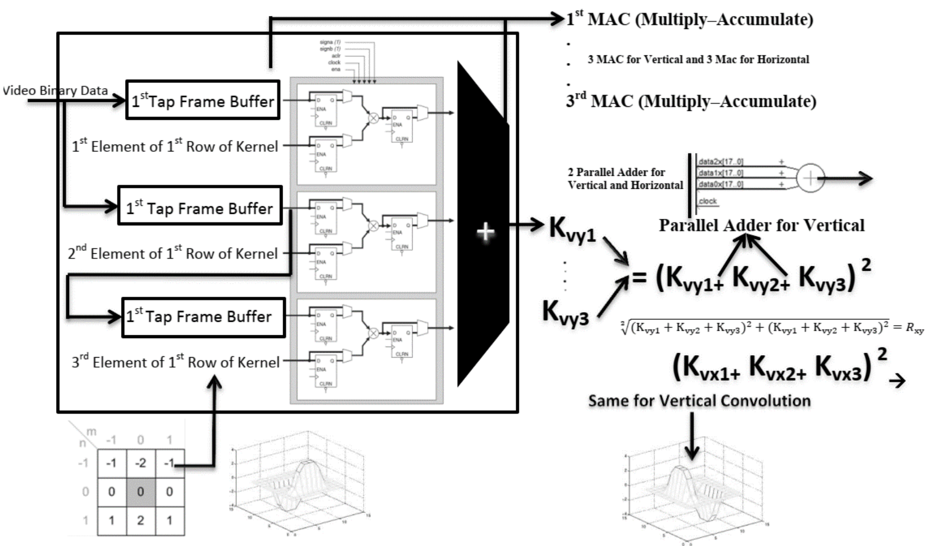
EMBOSS FILTER



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.

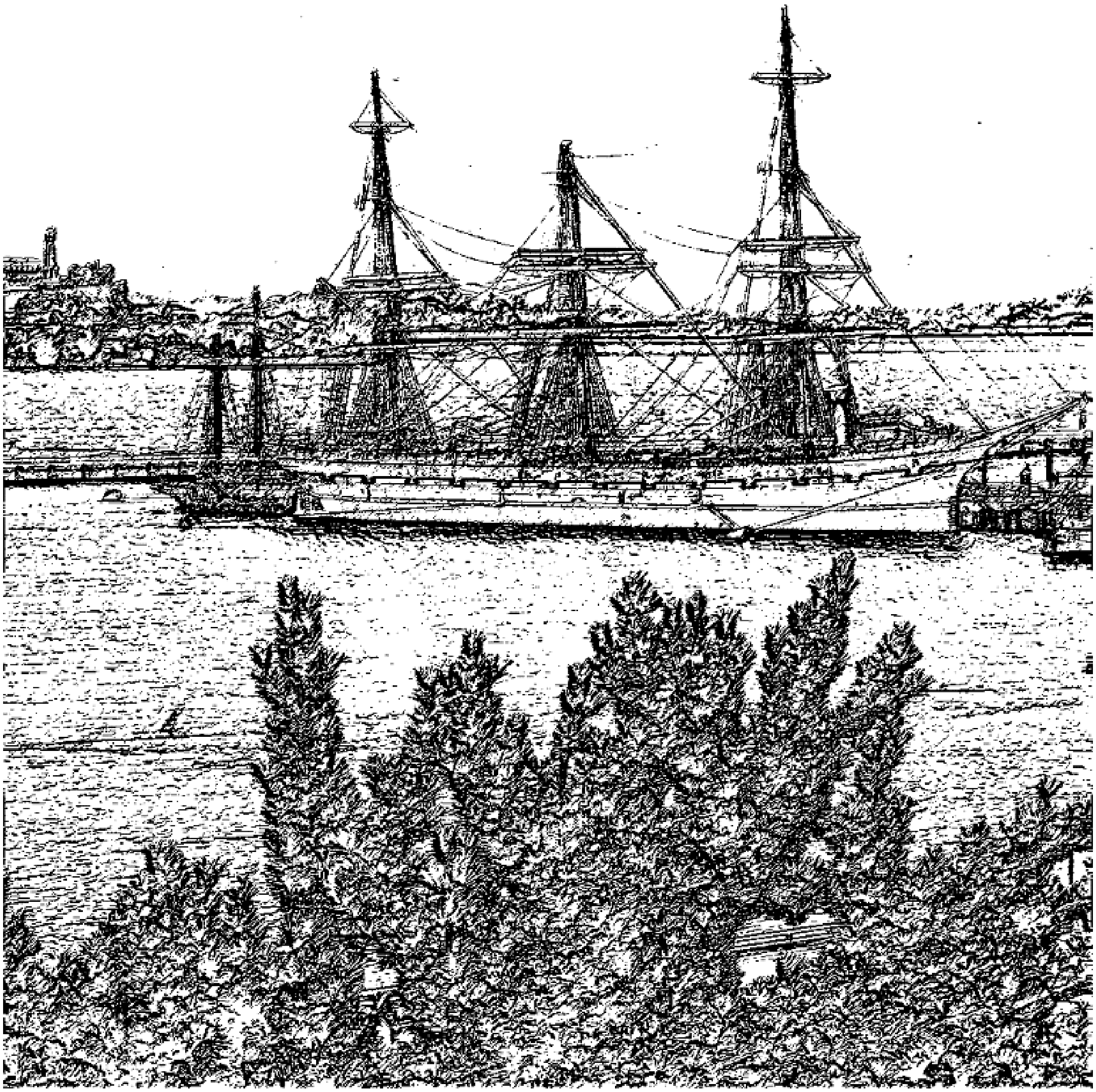


Figure 140

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.

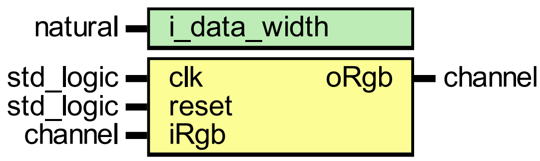


Figure 141

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

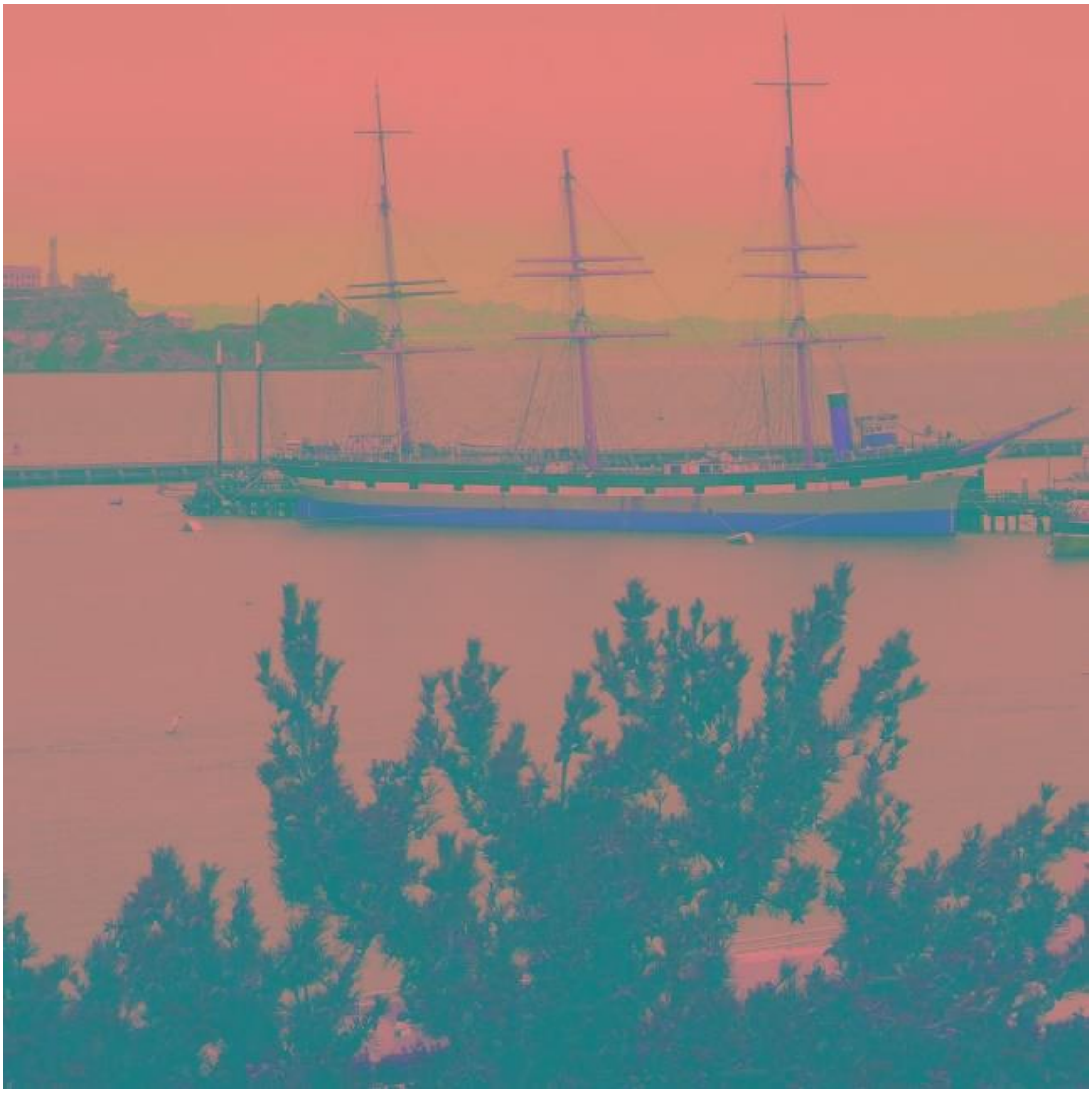


Figure 142

COLOR ADJUST MATRIX

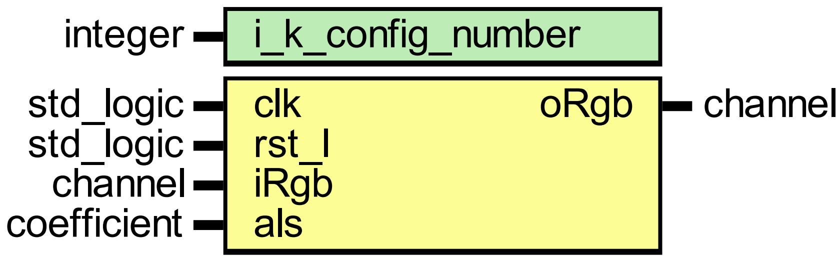
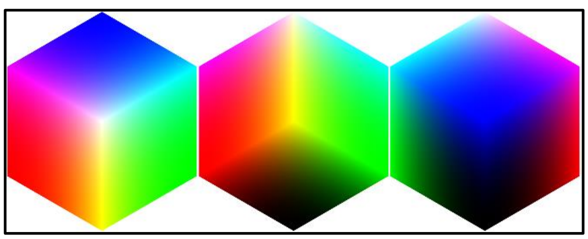


Figure 143

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.

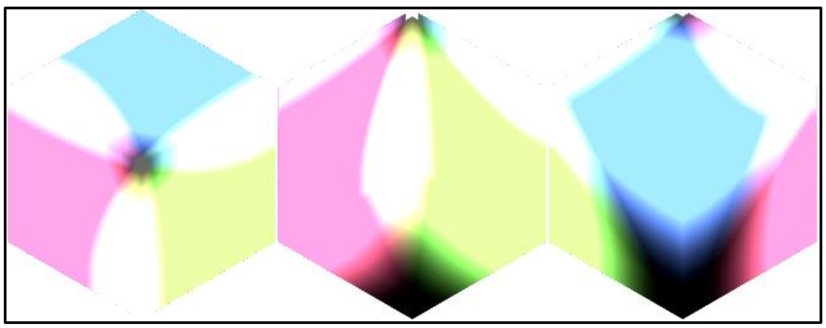
$$\begin{bmatrix} R_{CAM} \\ G_{CAM} \\ B_{CAM} \end{bmatrix} = \begin{bmatrix} K1 & K2 & K3 \\ K4 & K5 & K6 \\ K7 & K8 & K9 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



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.

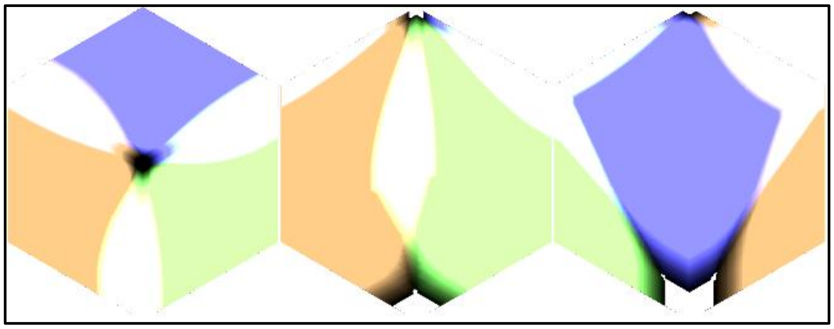
COLOR MANIPULATION INDEX # 01

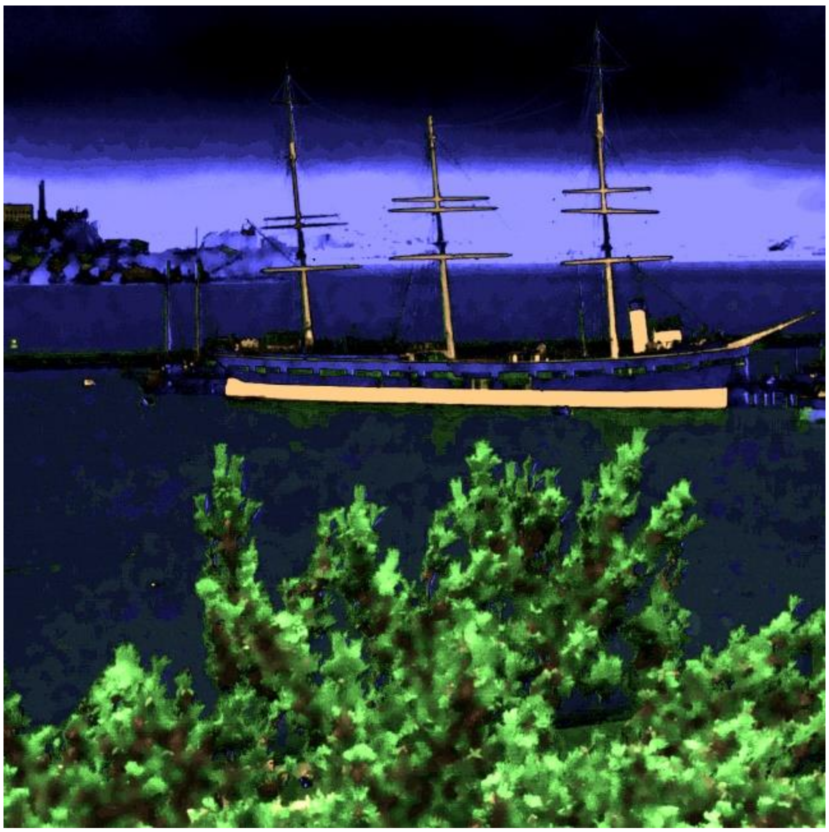
Color Adjust Matrix				
CAM-1	5.000	-3. 313	-1. 687	0.00
	-1. 687	5.000	-3. 313	0.00
	-3. 313	-1. 687	5.000	0.00
	0.00	0.00	0.00	
CAM-2	1.200	-0.435	-0.165	0.600
	-0.165	1.200	-0.435	0.600
	-0.435	-0.165	1.200	0.600
	0.600	0.600	0.600	
CAM-3	0.900	0.300	0.350	1.650
	0.350	0.900	0.300	1.650
	0.300	0.350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

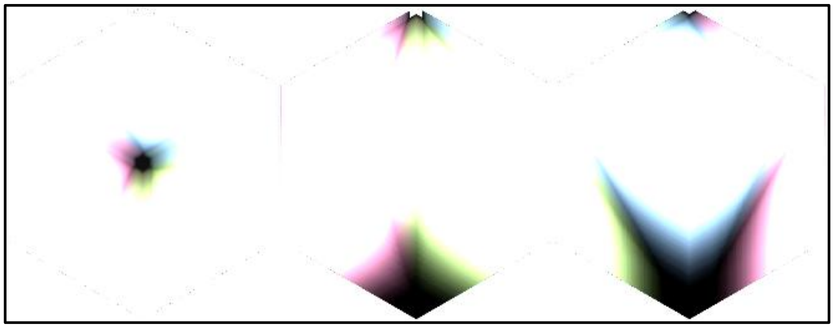




Color Manipulation Index # 03

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

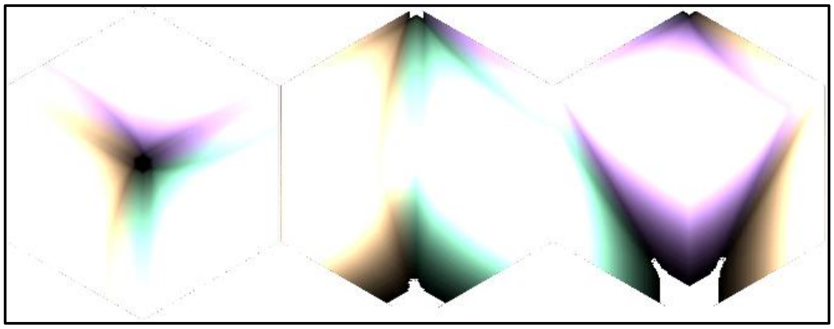
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



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

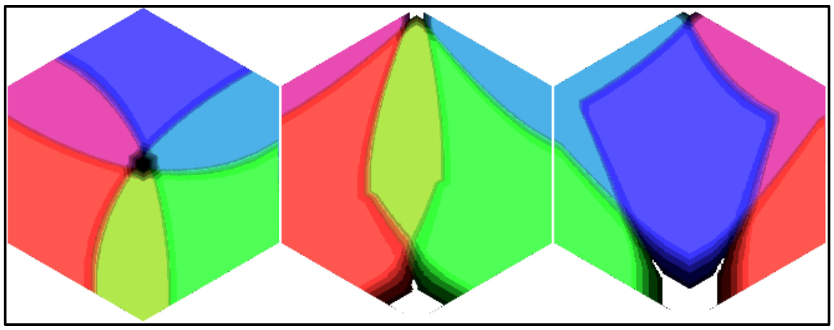
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	

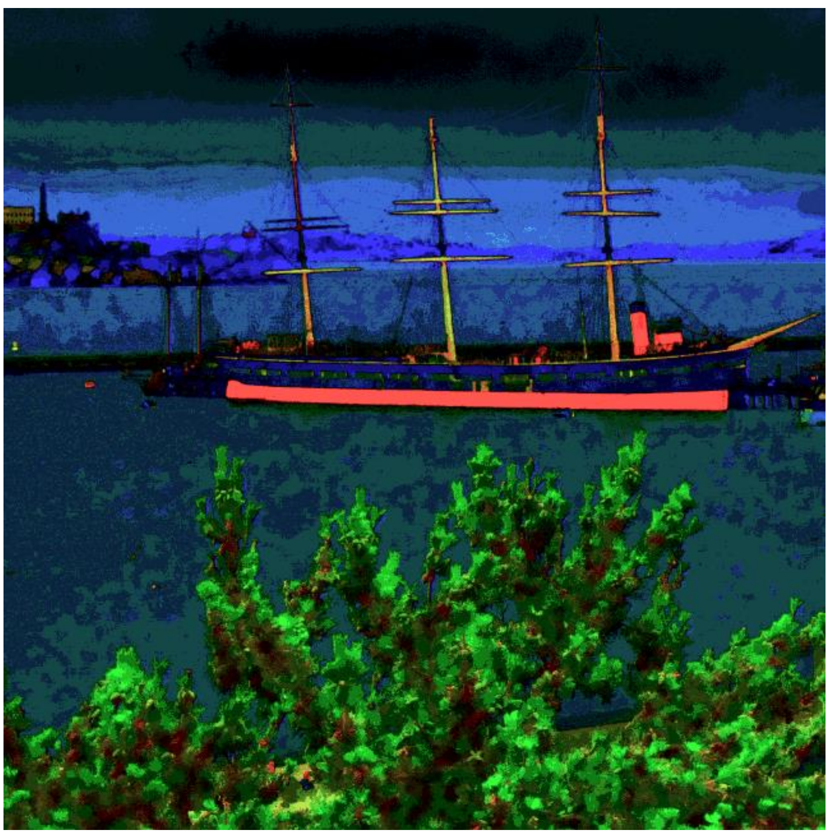


Color Manipulation Index # 05

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

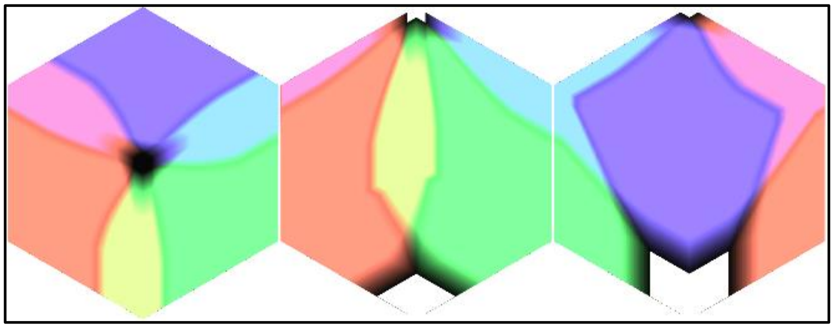
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



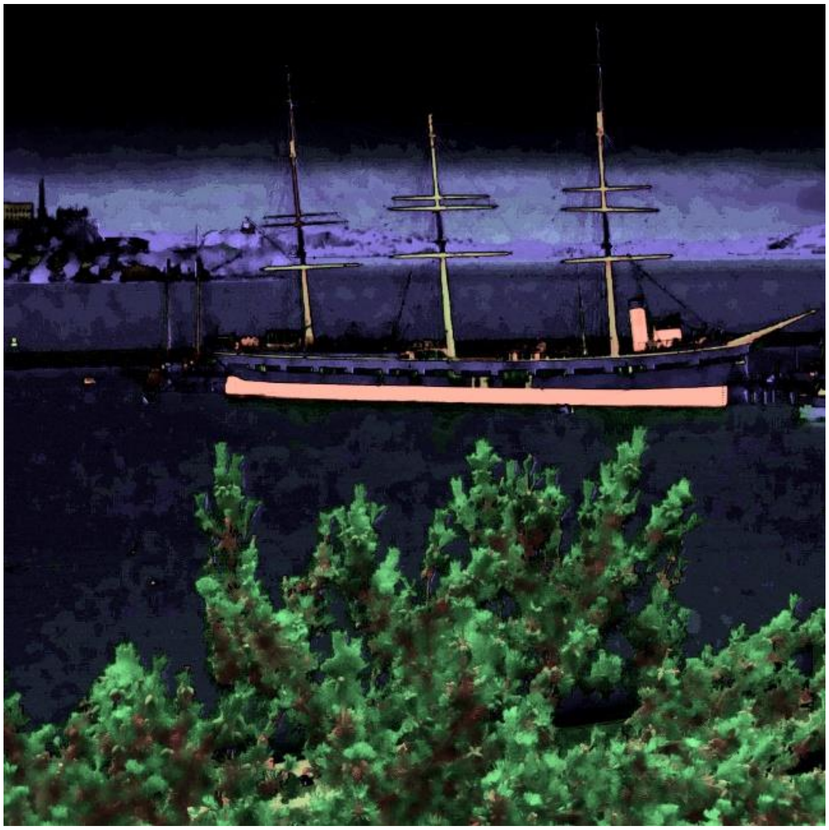


Color Manipulation Index # 06

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	

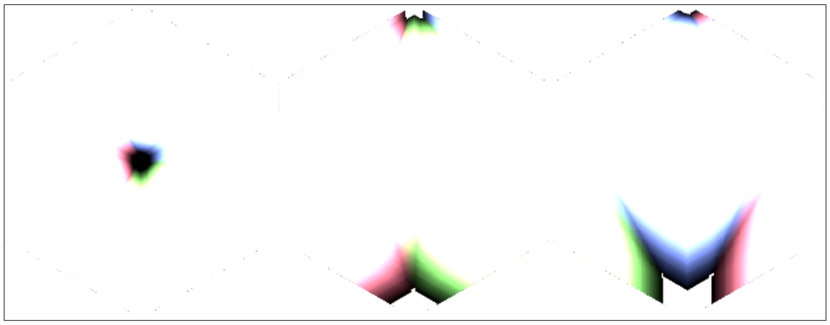


pg. 63



Color Manipulation Index # 07

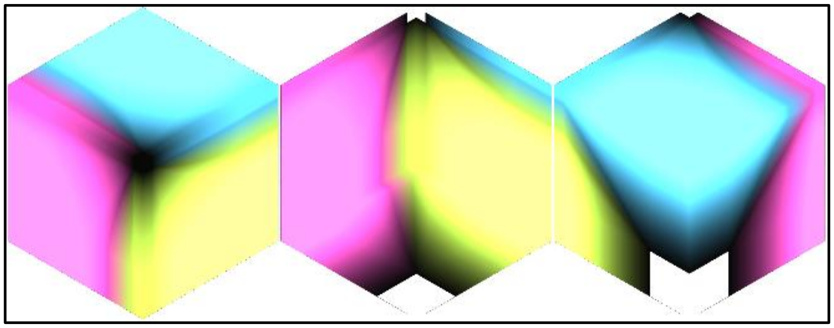
Color Adjust Matrix				
CAM-1	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-2	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	
CAM-3	1.5	-0.250	-0.250	1.00
	-0.250	1.5	-0.250	1.00
	-0.250	-0.250	1.5	1.00
	1.00	1.00	1.00	



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

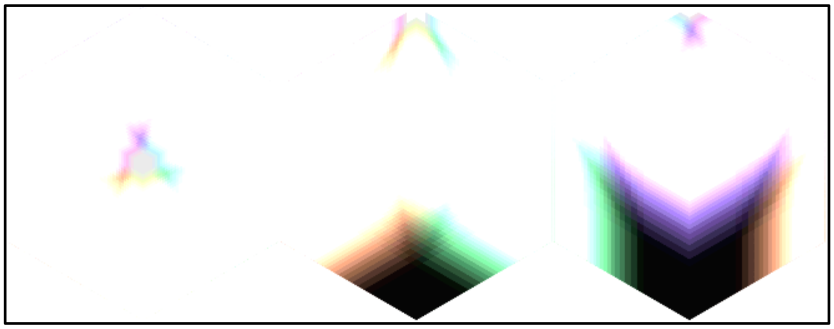
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



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

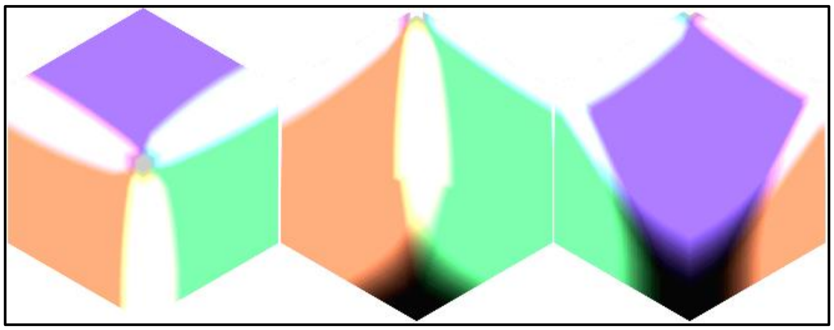
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	



Color Manipulation Index # 10

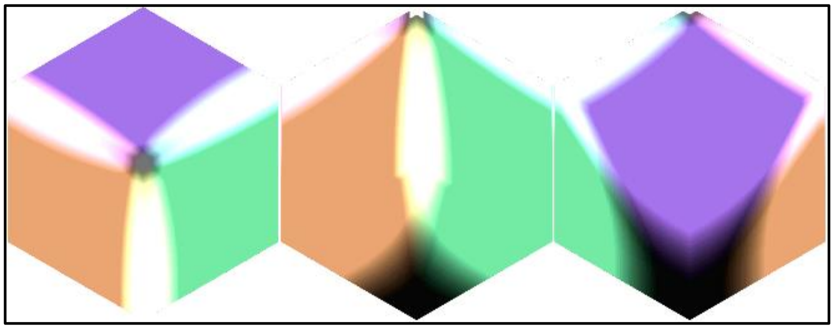
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00

	-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	



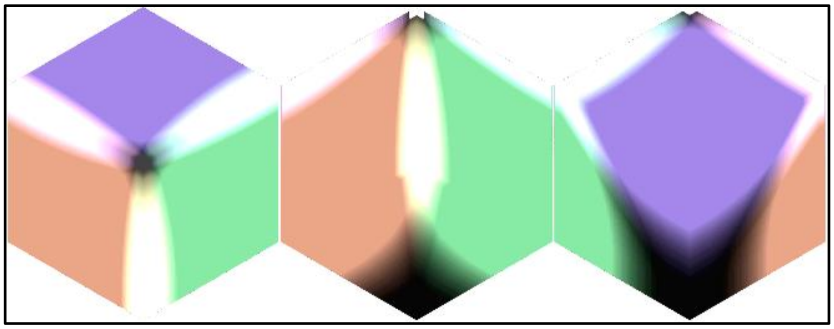
Color Manipulation Index # 11

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

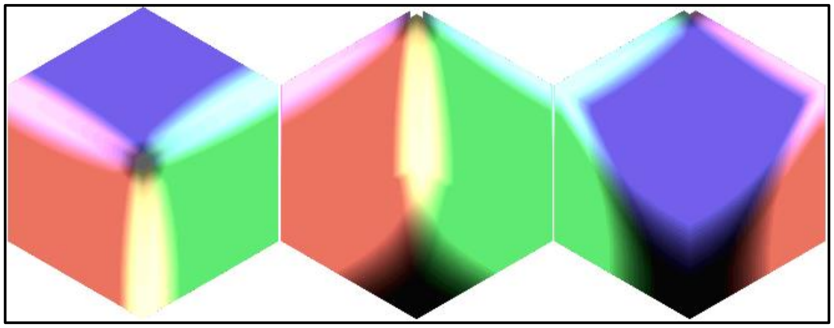
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00
	-0.850	-0.400	1.250	0.00
	0.00	0.00	0.00	
CAM-2	1.00	-0.600	-0.400	0.00
	-0.400	1.00	-0.600	0.00
	-0.600	-0.400	1.00	0.00
	0.00	0.00	0.00	
CAM-3	1.5	1.200	0.500	3.20
	0.500	1.5	1.200	3.20
	1.200	0.500	1.5	3.20
	3.20	3.20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

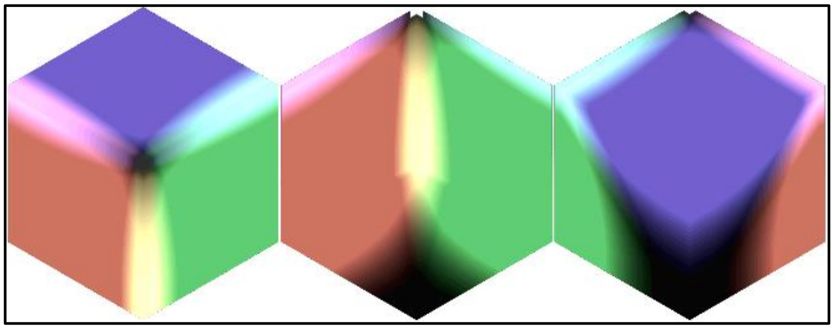
	1.200	0.500	1.5	3.20
	3.20	3.20	3.20	

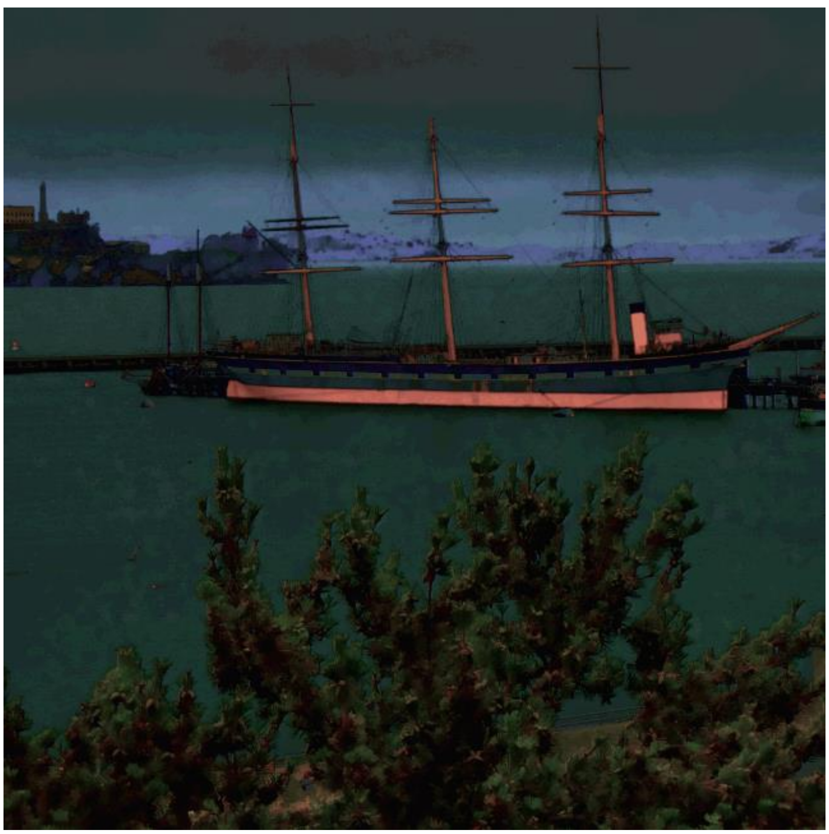


Color Manipulation Index # 14

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	

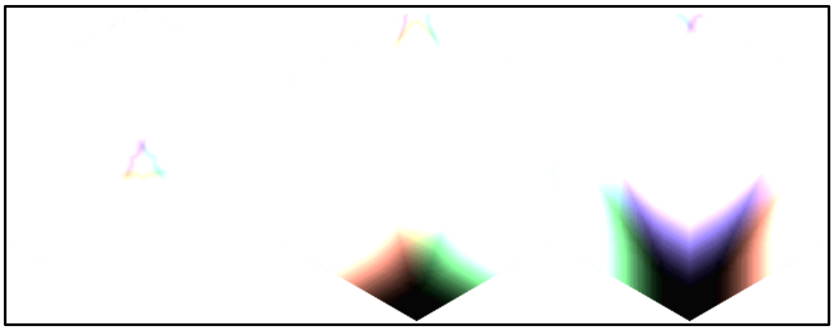




Color Manipulation Index # 15

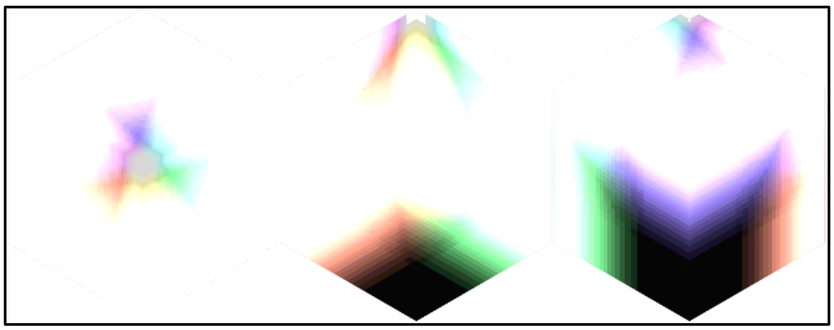
Color Adjust Matrix				
CAM-1	1.250	-0.850	-0.400	0.00
	-0.400	1.250	-0.850	0.00

	-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	



AN FPGA IMPLEMENTATION OF 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.

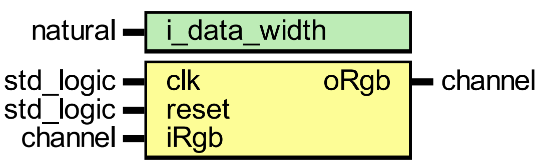


Figure 144

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.

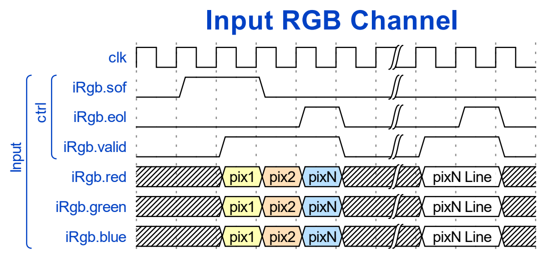


Figure 145

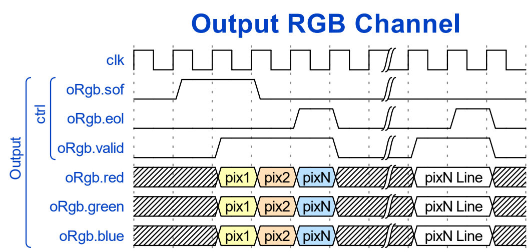
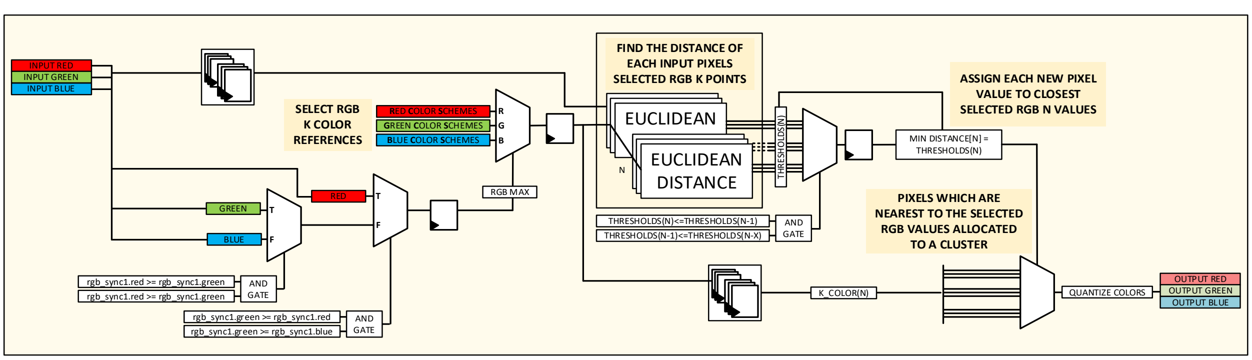


Figure 146

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.

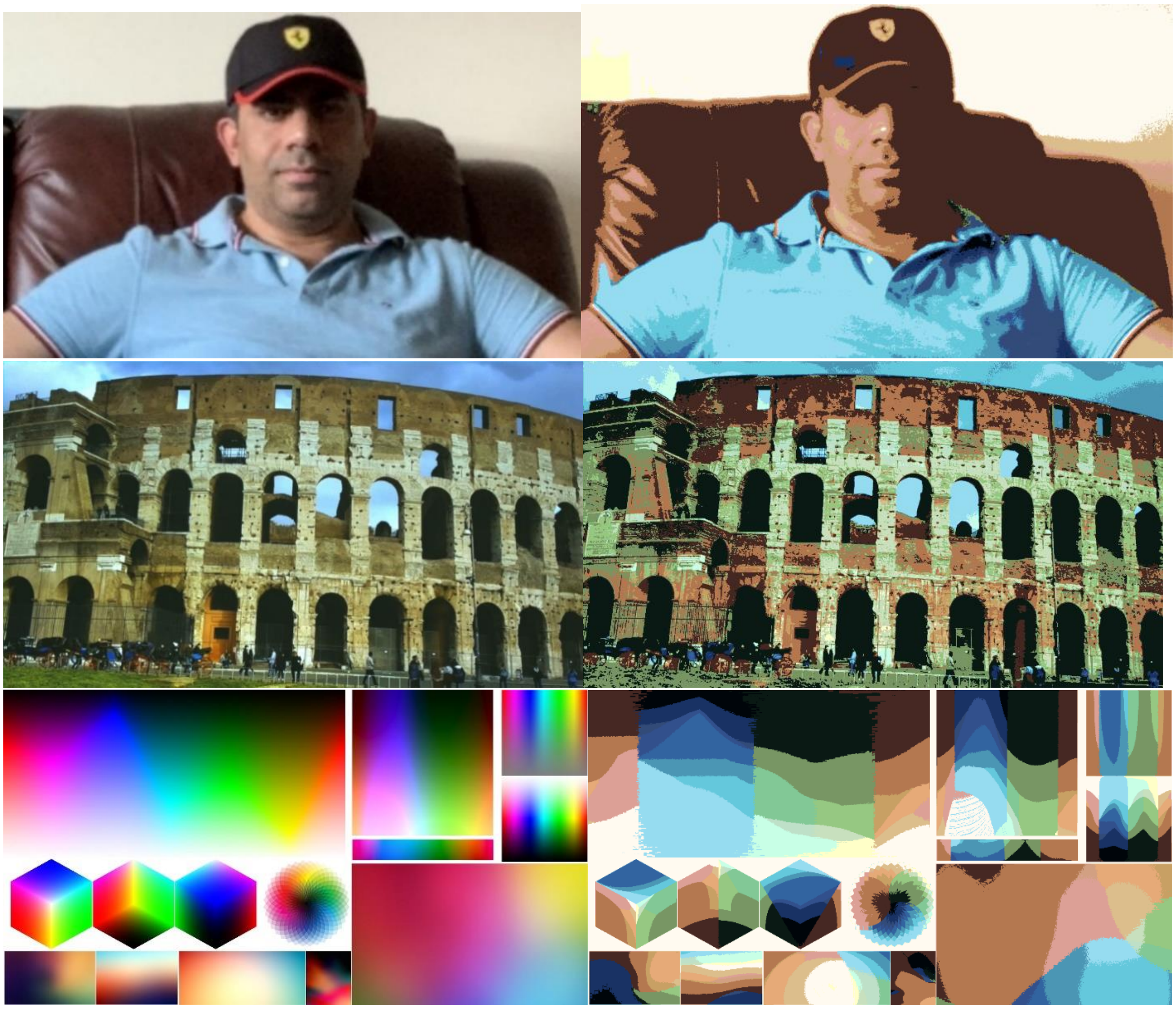
#	Red	Green	Blue
1	230	170	120
2	70	40	35
3	150	200	130
4	20	25	10
5	75	150	180
6	15	30	60

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.



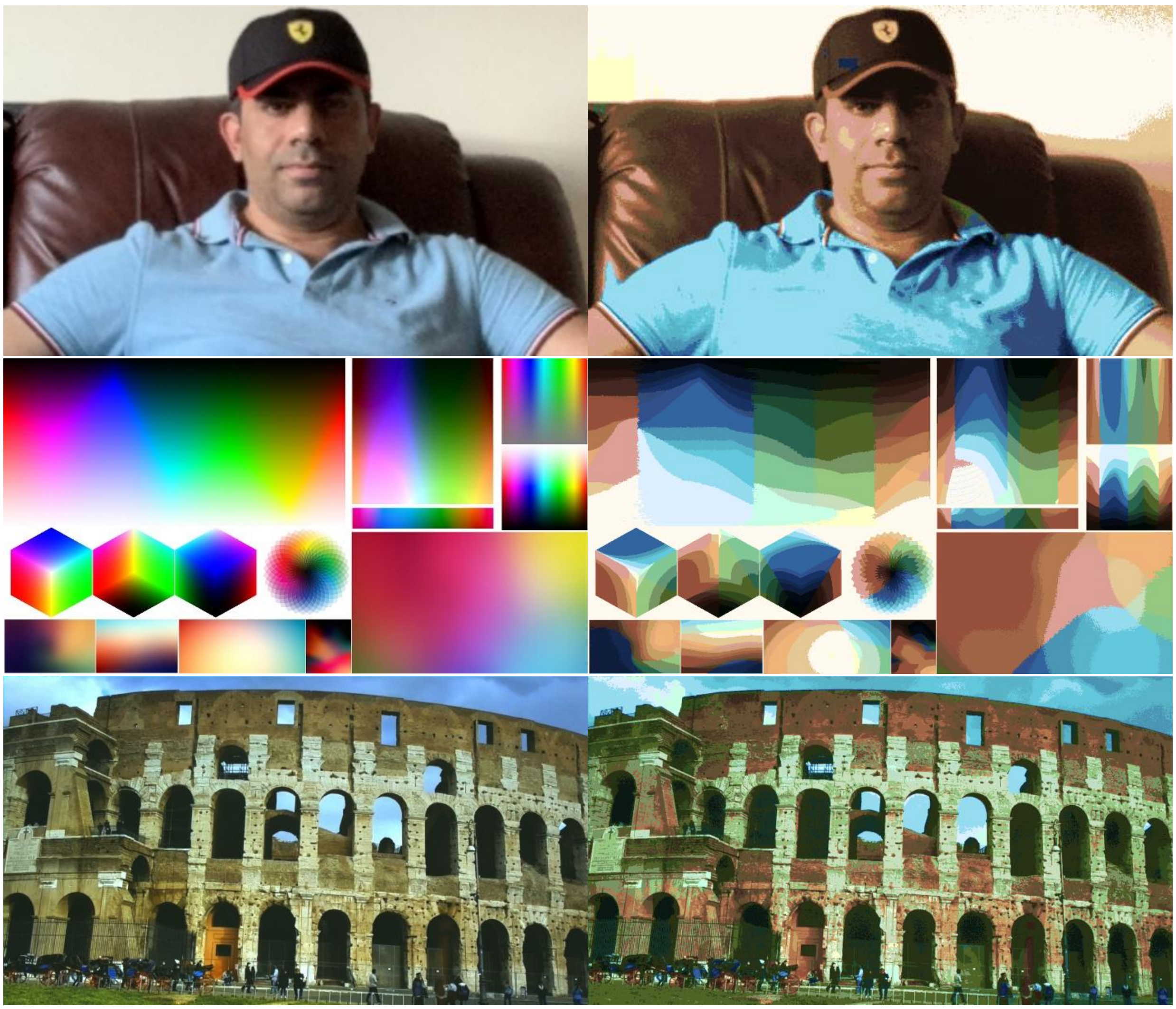
K EQUALTO 51 REFERENCE COLOR SCHEMES

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	#	R
1	255	250	240	11	150	80	60	21	255	255	200	31	5
2	230	200	150	12	140	80	60	22	200	255	230	32	3
3	220	190	140	13	120	80	55	23	200	224	190	33	4
4	240	180	120	14	100	80	50	24	190	224	200	34	2

5	230	170	120	*15*	80	60	40	*25*	150	200	130	*35*	1	
6	220	160	150	*16*	70	40	35	*26*	130	200	150	*36*	2	
7	210	160	130	*17*	60	40	30	*27*	120	150	100	*37*	1	
8	200	160	100	*18*	50	30	20	*28*	100	150	120	*38*	0	
9	180	120	80	*19*	40	20	15	*29*	80	100	40	*39*	2	
10	160	100	70	*20*	25	15	10	*30*	40	100	80	*40*	1	

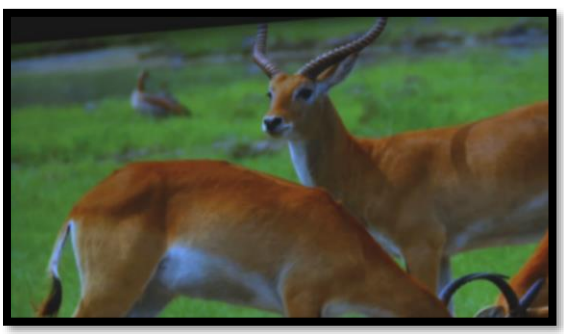


IMX477 CAMERA: K EQUALTO 90 REFERENCE COLOR SCHEMES

In this reference, k parameter set to k = 90, where k is the number of clusters. K-mean color quantization quantizes input image to number of colors into 90 clusters from 90 refences of color schemes.







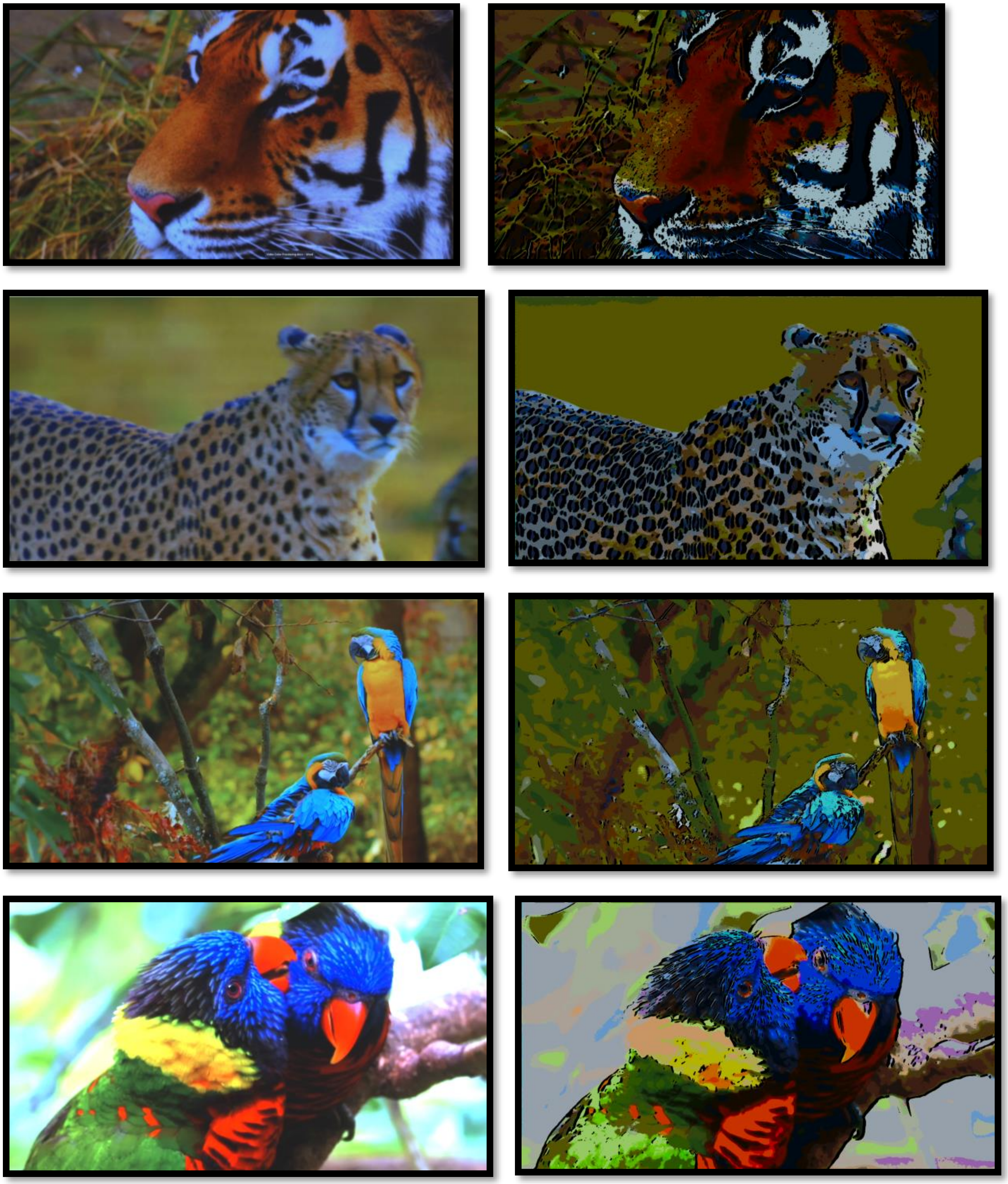
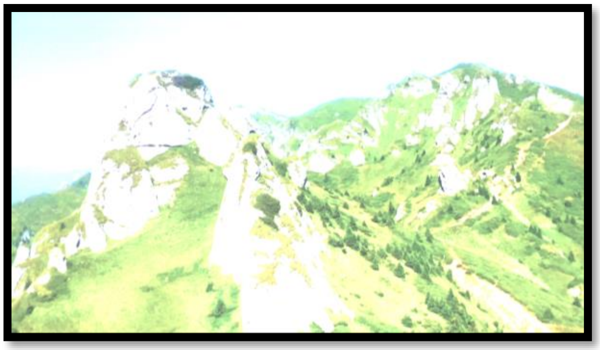
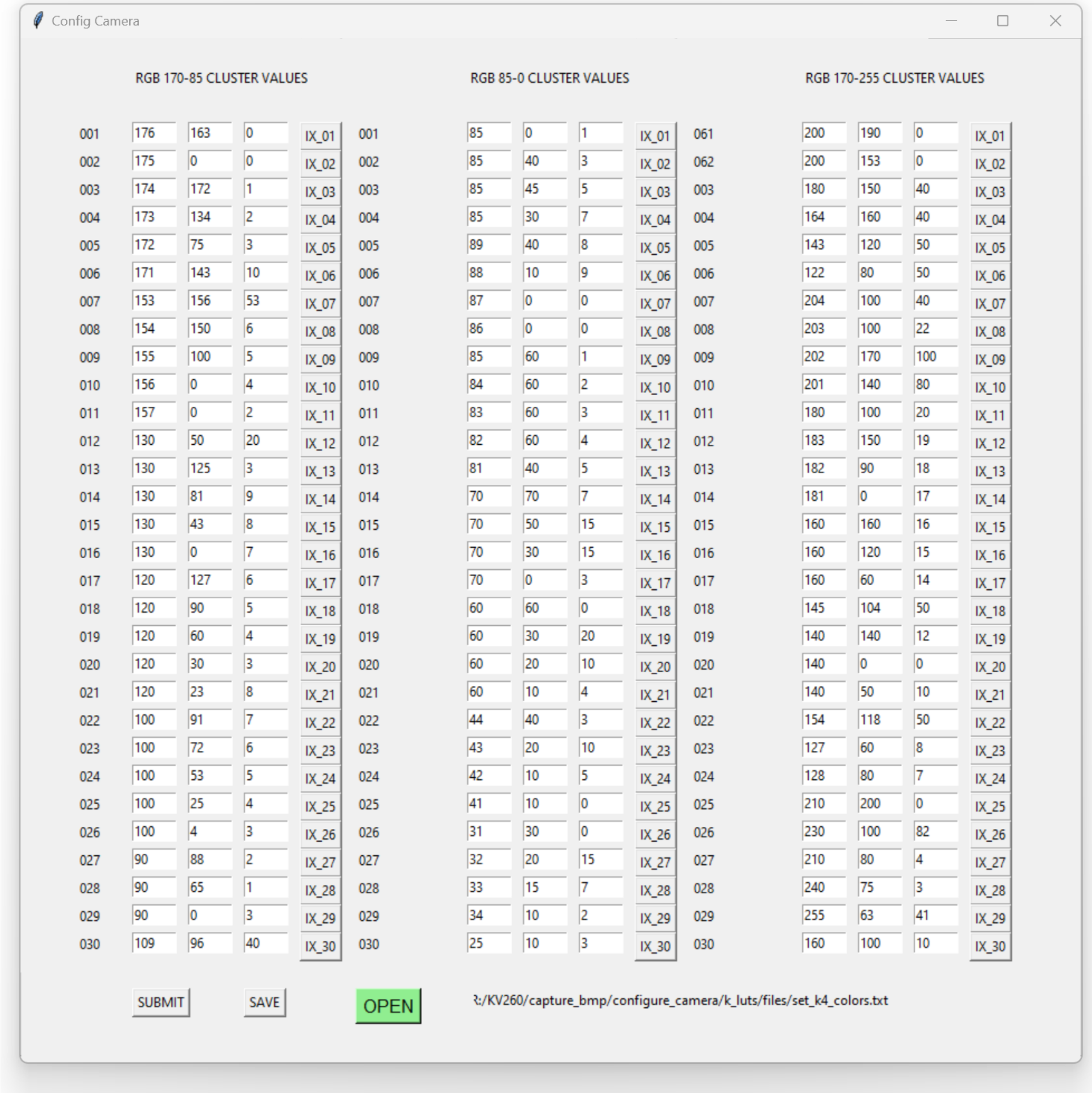



Figure 29. (Left) Original image; (Right) results obtained using the K-Mean Clustering.



90 REFERENCE PROGRAMABLE COLOR SCHEMES

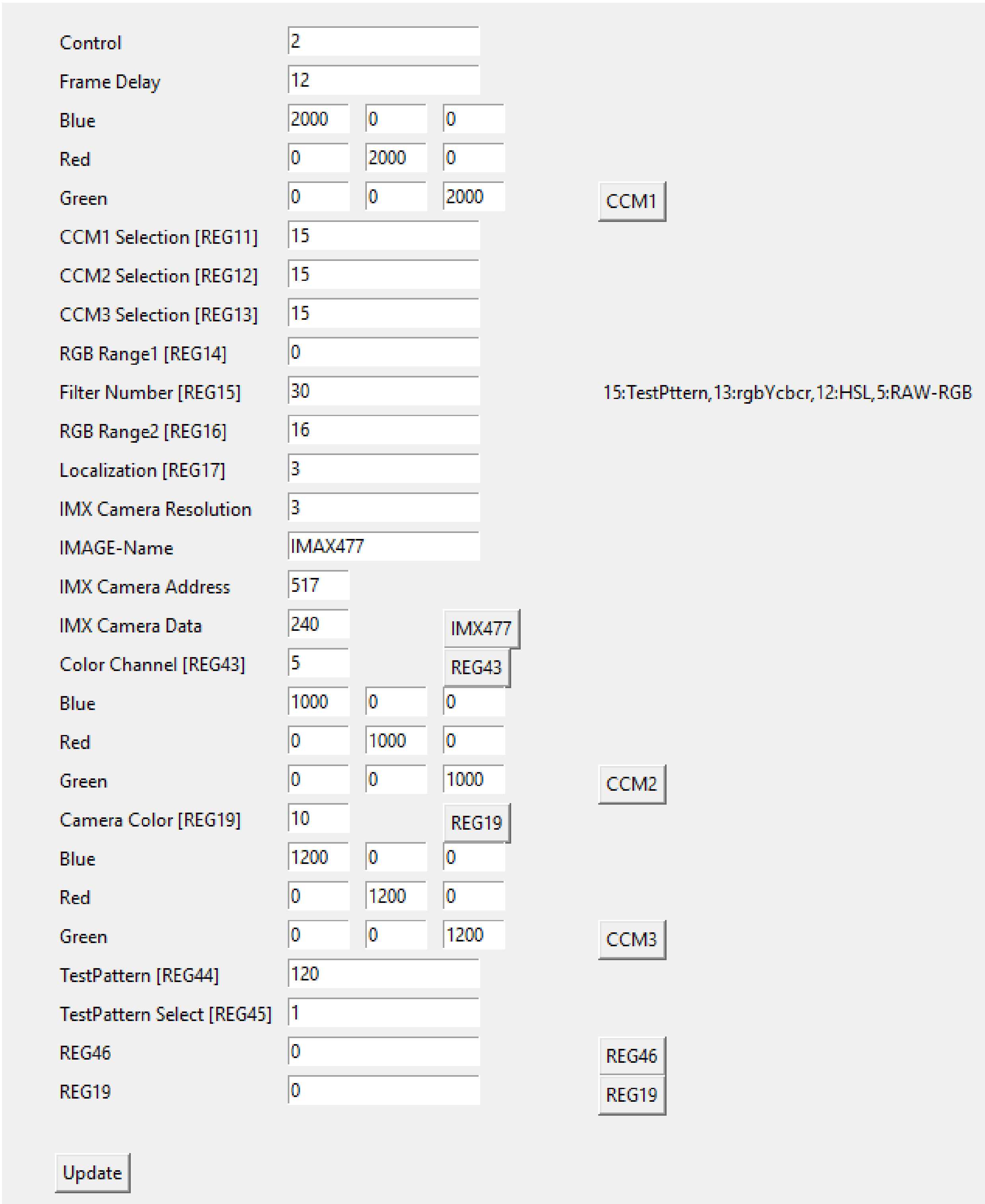


Config Camera

	RGB 170-85 CLUSTER VALUES					RGB 85-0 CLUSTER VALUES					RGB 170-255 CLUSTER VALUES			
001	176	163	0	IX_01	001	85	0	1	IX_01	061	200	190	0	IX_01
002	175	0	0	IX_02	002	85	40	3	IX_02	062	200	153	0	IX_02
003	174	172	1	IX_03	003	85	45	5	IX_03	003	180	150	40	IX_03
004	173	134	2	IX_04	004	85	30	7	IX_04	004	164	160	40	IX_04
005	172	75	3	IX_05	005	89	40	8	IX_05	005	143	120	50	IX_05
006	171	143	10	IX_06	006	88	10	9	IX_06	006	122	80	50	IX_06
007	153	156	53	IX_07	007	87	0	0	IX_07	007	204	100	40	IX_07
008	154	150	6	IX_08	008	86	0	0	IX_08	008	203	100	22	IX_08
009	155	100	5	IX_09	009	85	60	1	IX_09	009	202	170	100	IX_09
010	156	0	4	IX_10	010	84	60	2	IX_10	010	201	140	80	IX_10
011	157	0	2	IX_11	011	83	60	3	IX_11	011	180	100	20	IX_11
012	130	50	20	IX_12	012	82	60	4	IX_12	012	183	150	19	IX_12
013	130	125	3	IX_13	013	81	40	5	IX_13	013	182	90	18	IX_13
014	130	81	9	IX_14	014	70	70	7	IX_14	014	181	0	17	IX_14
015	130	43	8	IX_15	015	70	50	15	IX_15	015	160	160	16	IX_15
016	130	0	7	IX_16	016	70	30	15	IX_16	016	160	120	15	IX_16
017	120	127	6	IX_17	017	70	0	3	IX_17	017	160	60	14	IX_17
018	120	90	5	IX_18	018	60	60	0	IX_18	018	145	104	50	IX_18
019	120	60	4	IX_19	019	60	30	20	IX_19	019	140	140	12	IX_19
020	120	30	3	IX_20	020	60	20	10	IX_20	020	140	0	0	IX_20
021	120	23	8	IX_21	021	60	10	4	IX_21	021	140	50	10	IX_21
022	100	91	7	IX_22	022	44	40	3	IX_22	022	154	118	50	IX_22
023	100	72	6	IX_23	023	43	20	10	IX_23	023	127	60	8	IX_23
024	100	53	5	IX_24	024	42	10	5	IX_24	024	128	80	7	IX_24
025	100	25	4	IX_25	025	41	10	0	IX_25	025	210	200	0	IX_25
026	100	4	3	IX_26	026	31	30	0	IX_26	026	230	100	82	IX_26
027	90	88	2	IX_27	027	32	20	15	IX_27	027	210	80	4	IX_27
028	90	65	1	IX_28	028	33	15	7	IX_28	028	240	75	3	IX_28
029	90	0	3	IX_29	029	34	10	2	IX_29	029	255	63	41	IX_29
030	109	96	40	IX_30	030	25	10	3	IX_30	030	160	100	10	IX_30

SUBMIT SAVE OPEN ꝛ:/KV260/capture_bmp/configure_camera/k_luts/files/set_k4_colors.txt

VIDEO CONTROLLER INTERFACE



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.

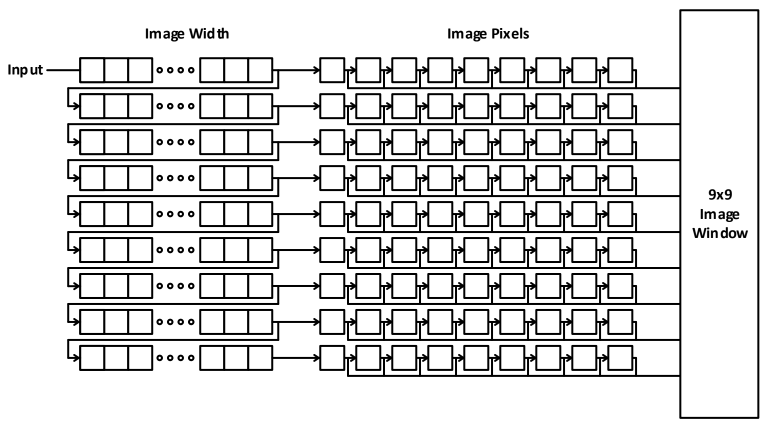


IMAGE CONTRAST AND BRIGHTNESS

CAMERA RAW DATA

This module read 12-bit RGB data from Demosaic Xilinx IP. 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.

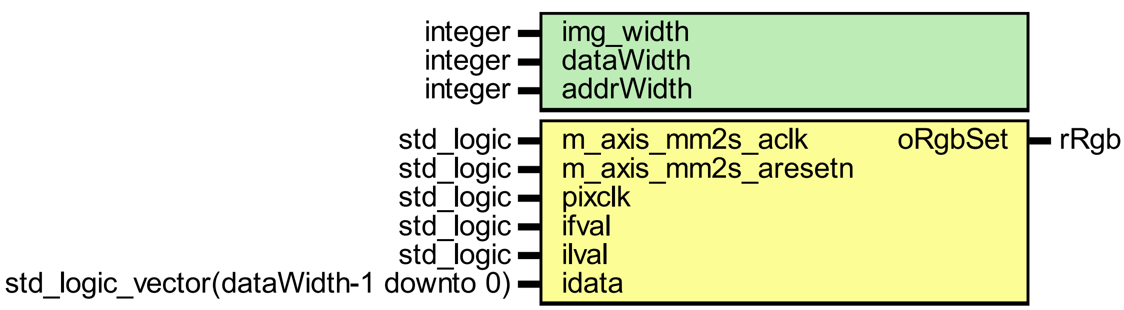


Figure 147 : camera raw data module

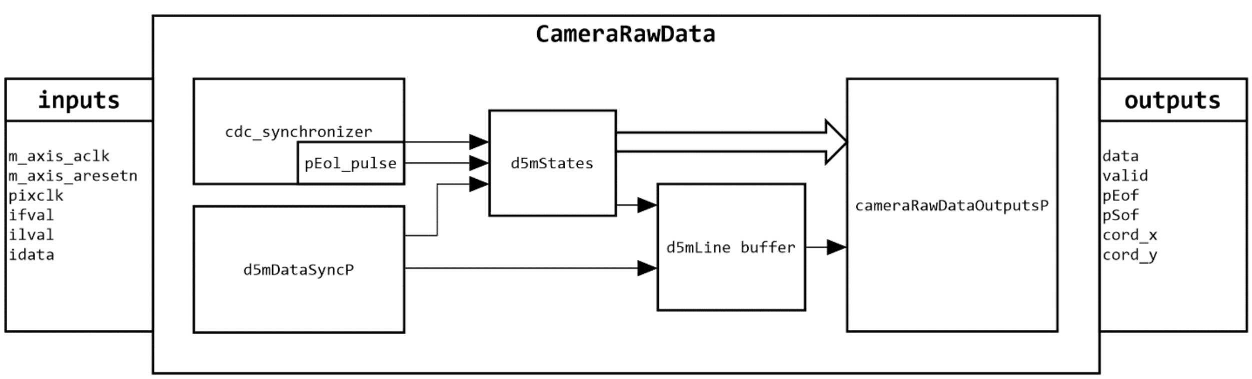


Figure 148 : General view of camera raw data flow

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

THREE TAPS DATA

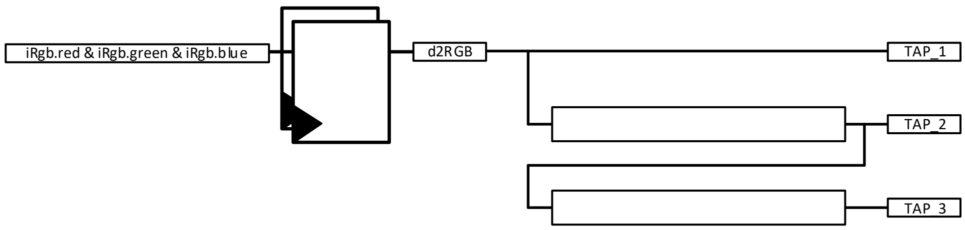


Figure 149

FOUR TAPS DATA

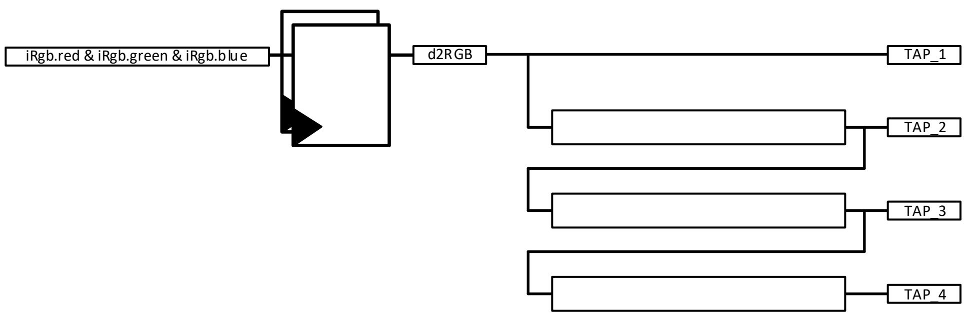


Figure 150

PIXEL COORDINATES

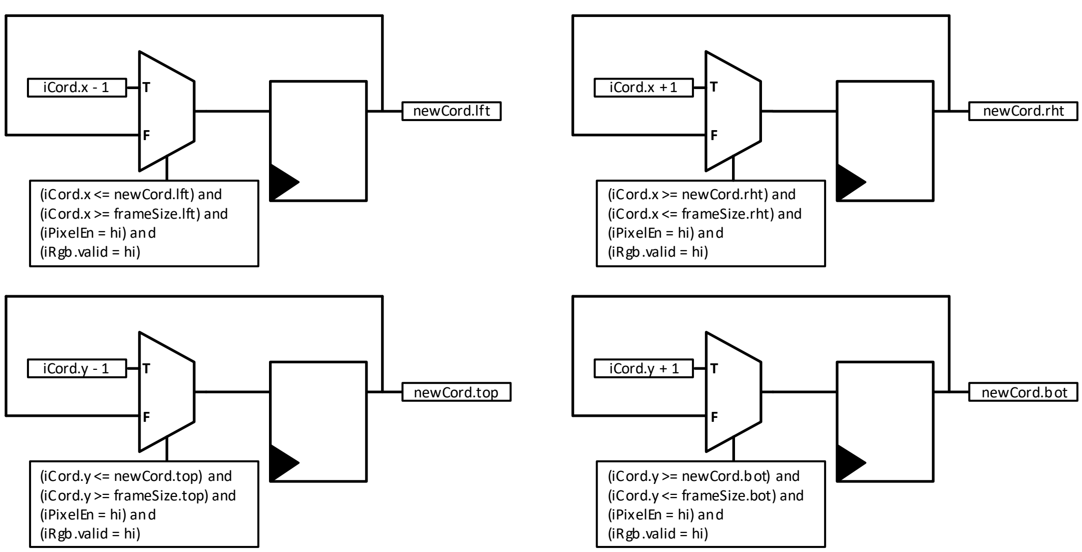


Figure 151

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

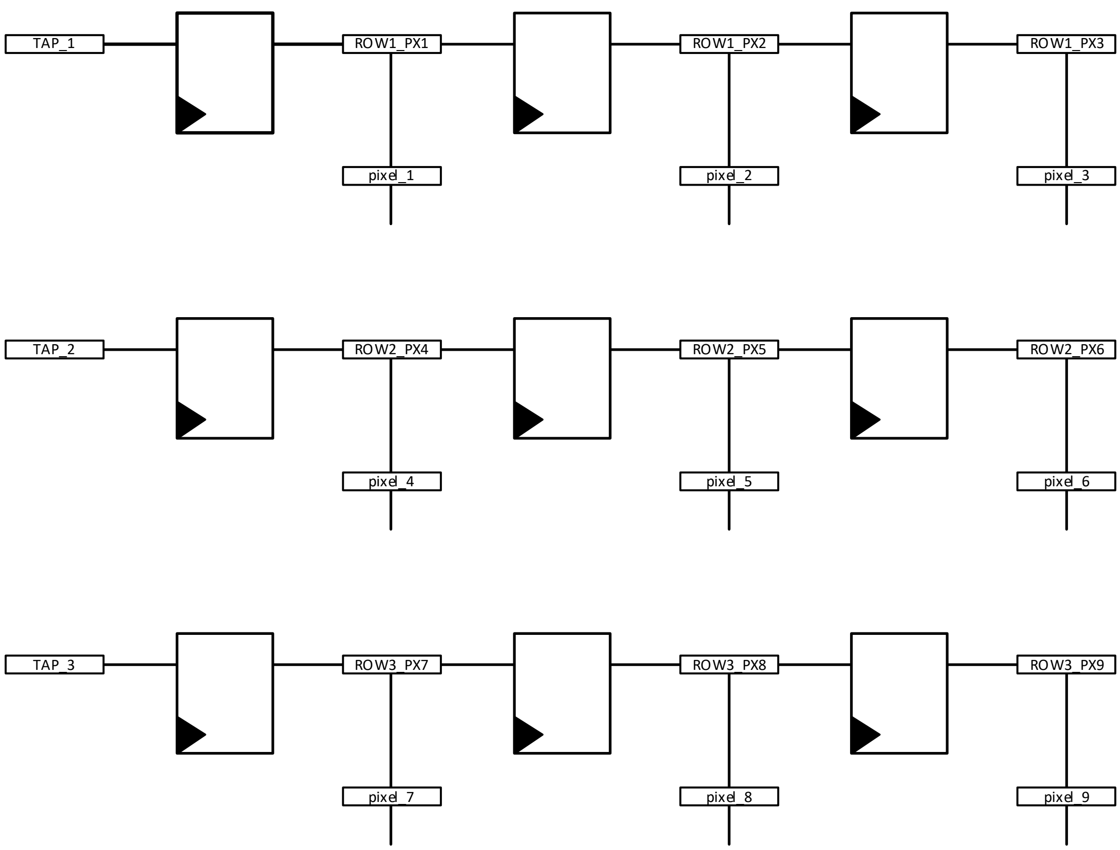
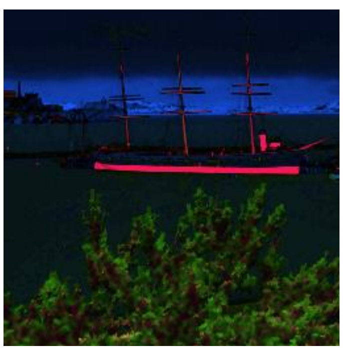


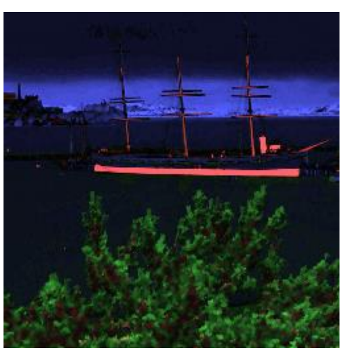
Figure 152

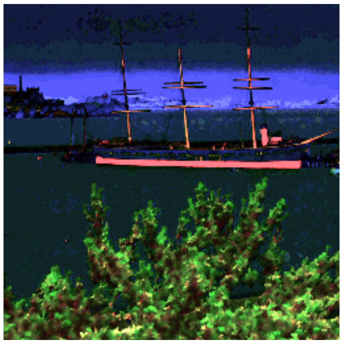
K1	K2	K3
K4	K5	K6
K7	K8	K9

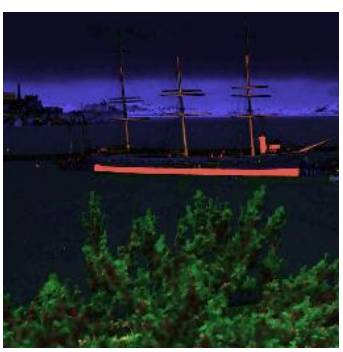
Figure 153

CAMERA RAW DATA









HISTOGRAM

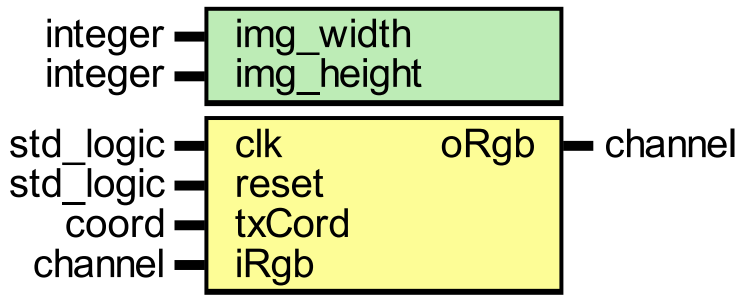


Figure 154

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.

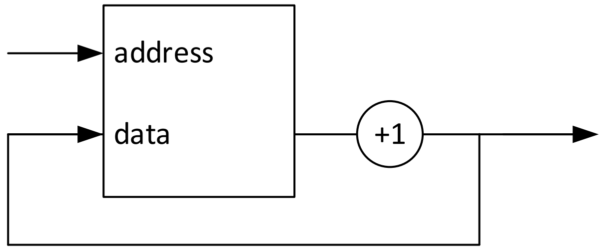
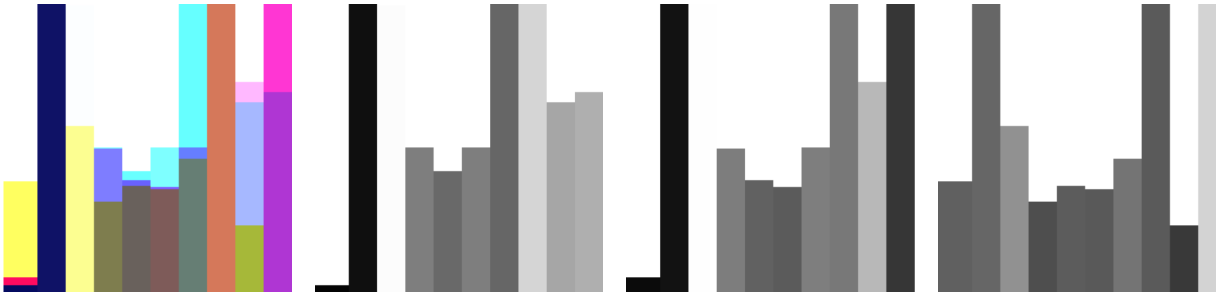


Figure 155

Histogram Equation



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

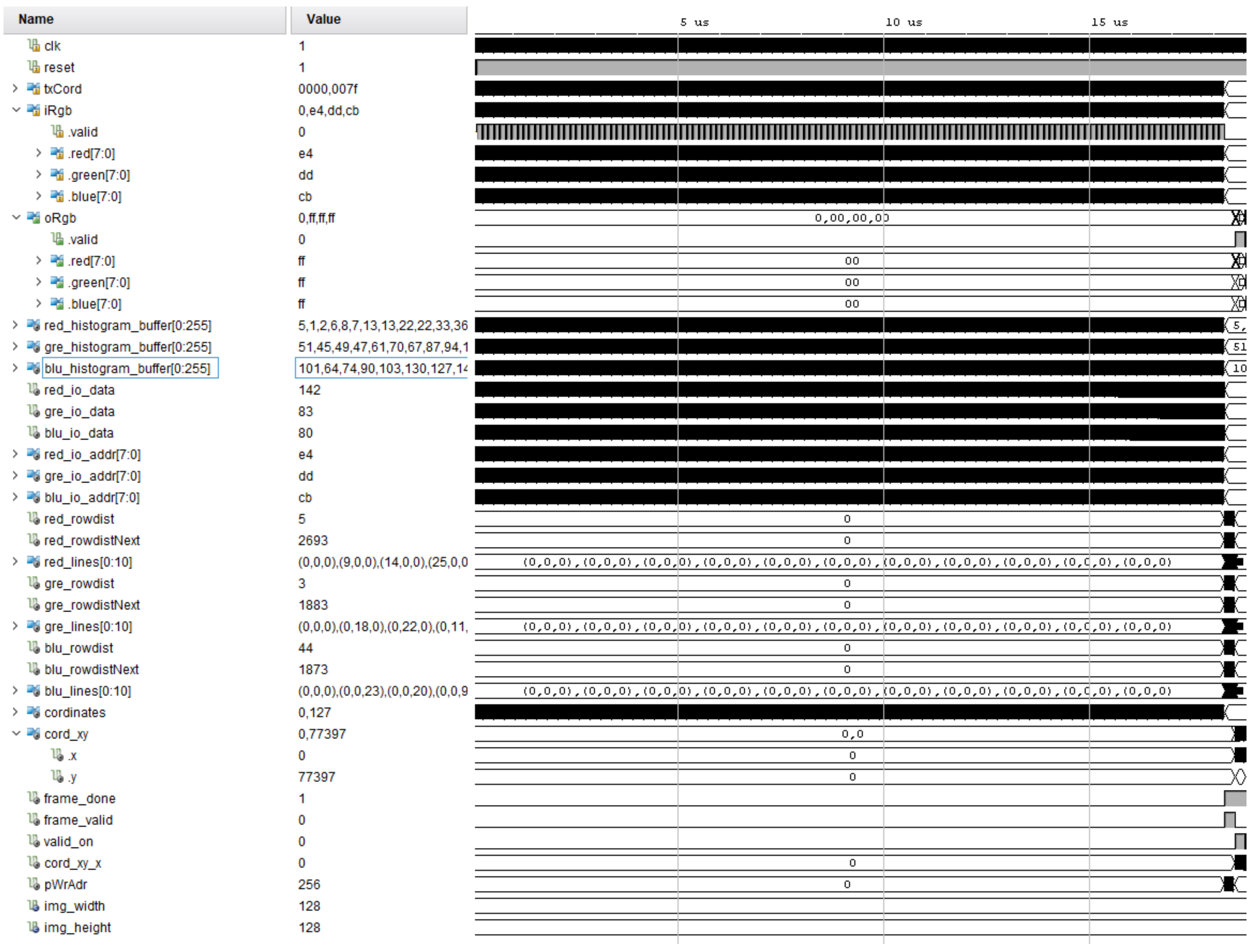


Figure 156

ETHERNET UDP VIDEO STREAMING

To successfully run the real-time network UDP (User Datagram Protocol) video streaming, this reference design uses the KRIA KV260 development board developed by Xilinx.

The design is composed of two platforms: the transmitter platform and the receiver platform. The receiver platform which receives input video stream from imx477 camera implements PL logic design of device. The transmitter platform which is PS Side of device implements a UDP/IP hardware protocol stack that enables high speed communication over a LAN or a point-to-point connection.

UDP/IP video streaming is implemented by using LWIP (**L**ight**W**eight **I**nternet **P**rotocol) which is open-source networking stacked design for embedded system. The main purpose usage of lwip in this design is portability, as well as small code and data size, making it especially suited for network video transmission.

LWIP supports both TCP and UDP at transport layer and application-level protocols. To get better performance, UDP protocol is used as a transport layer protocol. UDP is a connectionless protocol and faster than TCP since there is no error checking for packets. UDP protocol is used as transport layer in this design since it is suitable for video transmission that require fast transportation.

CONFIGURATION

KRIA KV260 board connect directly to the host PC and not running a DHCP server and host PC need to configure to an IP address of 192.168.0.42 and subnet mask of 255.255.255.0. Default IP address for FPGA development board is configured to “192.168.0.10”.

NETWORK

At startup LWIP fetch an IP address from DHCP server. If there no DHCP server than DHCP server request will timeout and the board will default to preset default to an IP address of 192.168.0.10.

The FFMPEG application on the host PC will connect with the video transmitter application on board and video data transmission will commence. When the video images are received, the FFMPEG invoke ffplay.exe which decode first 54 bytes source video type. Below figure shows 54 bytes of video header format.

```
static const BmpMode BMODE_1920x1080 = {
    .bm_header      = {0x42, 0x4d},
    .bm_len         = {0x36, 0xa4, 0x1f, 0x00}, /* file length 8294400+54 bytes */
    .reserved       = {0x00, 0x00, 0x00, 0x00},
    .offset         = {0x36, 0x00, 0x00, 0x00}, /* 54 bytes */
    .bm_infolen     = {0x28, 0x00, 0x00, 0x00}, /* 40 bytes */
    .bm_width       = {0x80, 0x07, 0x00, 0x00}, /* width 1920 */
    .bm_height      = {0x38, 0x04, 0x00, 0x00}, /* height 1080 */
    .color_plane    = {0x01, 0x00},
    .pixel_width    = {0x18, 0x00},             /* pixel 32 bit true color */
    .compress       = {0x00, 0x00, 0x00, 0x00}, /* not compressed */
    .bm_bytes       = {0x00, 0xa4, 0x1f, 0x00}, /* frame length 8294400 bytes */
    .meter_width    = {0x00, 0x00, 0x00, 0x00},
    .meter_height   = {0x00, 0x00, 0x00, 0x00},
    .color_index    = {0x00, 0x00, 0x00, 0x00},
    .index_num      = {0x00, 0x00, 0x00, 0x00}
};
```

The BSP required to run the lwip application and it includes many settings that impact ethernet performance. Below figure shows lwip BSP settings.

Name	Value	Default	Type	Description
api_mode	RAW API (RAW_API)	RAW_API	enum	Mode of operation for lwIP (RAW API/Sockets API)
lwip_tcp_keepalive	false	false	boolean	Enable keepalive processing with default interval
no_sys_no_timers	true	true	boolean	Drops support for sys_timeout when NO_SYS==1
socket_mode_thread_prio	2	2	integer	Priority of threads in socket mode
use_axieth_on_zynq	0	1	integer	Option if set to 1 ensures axiethernet adapter being used in Zynq. Valid only for Zynq
use_emaclite_on_zynq	0	1	integer	Option if set to 1 ensures emaclite adapter being used in Zynq. Valid only for Zynq
arp_options	true	true	boolean	ARP Options
arp_queueing	1	1	integer	If enabled outgoing packets are queued during hardware address resolution.
arp_table_size	10	10	integer	Number of active hardware address IP address pairs cached.
debug_options	true	true	boolean	Turn on lwIP Debug?
icmp_debug	false	false	boolean	Debug ICMP protocol
igmp_debug	false	false	boolean	Debug IGMP protocol
ip_debug	false	false	boolean	Debug IP layer
lwip_debug	false	false	boolean	Turn on lwIP Debug?
netif_debug	false	false	boolean	Debug network interface layer
pbuf_debug	false	false	boolean	Debug pbuf layer
sys_debug	false	false	boolean	Debug sys arch layer
tcp_debug	false	false	boolean	Debug TCP layer
udp_debug	false	false	boolean	Debug UDP layer
dhcp_options	true	true	boolean	Is DHCP required?
dhcp_does_arp_check	true	false	boolean	ARP check on offered addresses?
lwip_dhcp	true	false	boolean	Is DHCP required?
icmp_options	true	true	boolean	ICMP Options
icmp_ttl	255	255	integer	ICMP TTL value
igmp_options	false	false	boolean	IGMP Options
lwip_ip_options	true	true	boolean	IP Options
ip_default_ttl	255	255	integer	Global default TTL used by transport layers
ip_forward	0	0	integer	Enable forwarding IP packets across network interfaces.
ip_frag	1	1	integer	Fragment outgoing IP packets if their size exceeds MTU
ip_frag_max_mtu	1500	1500	integer	Assumed max MTU on any interface for IP frag buffer
ip_options	0	0	integer	1 = IP options are allowed (but not parsed). 0 = packets with IP options are dropped
ip_reass_max_pbufs	128	128	integer	Reassembly PBUF Queue Length
ip_reassembly	1	1	integer	Reassemble incoming fragmented IP packets
ipv6_enable	false	false	boolean	IPv6 enable value
ipv6_options	true	true	boolean	IPv6 Options
lwip_memory_options				Options controlling lwIP memory usage
mem_size	51998488	131072	integer	Size of the heap memory (bytes).
memp_n_pbuf	16384	16	integer	Number of memp struct pbufs. Set this high if application sends lot of data out of ROM
memp_n_sys_timeout	8	8	integer	Number of simultaneously active timeouts
memp_n_tcp_pcb	32	32	integer	Number of active TCP PCBs. One per active TCP connection
memp_n_tcp_pcb_listen	8	8	integer	Number of listening TCP connections
memp_n_tcp_seg	256	256	integer	Number of simultaneously queued TCP segments
memp_n_udp_pcb	4	4	integer	Number of active UDP PCBs. One per active UDP connection
memp_num_api_msg	16	16	integer	Number of api msg structures (socket mode only)
memp_num_netbuf	8	8	integer	Number of struct netbufs (socket mode only)
memp_num_netconn	16	16	integer	Number of struct netconns (socket mode only)
memp_num_tcpip_msg	64	64	integer	Number of tcpip msg structures (socket mode only)
mbox_options	true	true	boolean	Mbox Options
default_tcp_recvmbox_size	200	200	integer	Size of TCP receive mbox queue.
default_udp_recvmbox_size	100	100	integer	Size of UDP receive mbox queue.
lwip_tcpip_core_locking_input	false	false	boolean	TCPIP input core locking
tcpip_mbox_size	200	200	integer	Size of TCPIP mbox queue.
pbuf_options	true	true	boolean	Pbuf Options
pbuf_link_hlen	16	16	integer	Number of bytes that should be allocated for a link level header.
pbuf_pool_bufsize	115200	1700	integer	Size of each pbuf in pbuf pool.
pbuf_pool_size	16384	256	integer	Number of buffers in pbuf pool.
stats_options	true	true	boolean	Turn on lwIP statistics?
lwip_stats	false	false	boolean	Turn on lwIP statistics?
tcp_options	true	true	boolean	Is TCP required ?
lwip_tcp	false	true	boolean	Is TCP required ?
tcp_maxrtx	12	12	integer	TCP Maximum retransmission value
tcp_mss	1460	1460	integer	TCP Maximum segment size (bytes)
tcp_queue_ooseq	0	1	integer	Should TCP queue segments arriving out of order. Set to 0 if your device is low on memory
tcp_snd_buf	8192	8192	integer	TCP sender buffer space (bytes)
tcp_synmaxrtx	4	4	integer	TCP Maximum SYN retransmission value
tcp_ttl	255	255	integer	TCP TTL value
tcp_wnd	2048	2048	integer	TCP Window (bytes)
temac_adapter_options	true	true	boolean	Settings for xps-ll-temac/Axi-Ethernet/Gem lwIP adapter
emac_number	0	0	integer	Zynq Ethernet Interface number
n_rx_coalesce	1	1	integer	Setting for RX Interrupt coalescing.Applicable only for Axi-Ethernet/xps-ll-temac.
n_rx_descriptors	2048	64	integer	Number of RX Buffer Descriptors to be used in SDMA mode
n_tx_coalesce	1	1	integer	Setting for TX Interrupt coalescing. Applicable only for Axi-Ethernet/xps-ll-temac.
n_tx_descriptors	2048	64	integer	Number of TX Buffer Descriptors to be used in SDMA mode
phy_link_speed	1000 Mbps (CONFIG_LIN...	CONFIG_LINKSPEED_AUT...	enum	link speed as negotiated by the PHY
tcp_ip_rx_checksum_offload	false	false	boolean	Offload TCP and IP Receive checksum calculation (hardware support required).Applicable only for Axi-Et...
tcp_ip_tx_checksum_offload	false	false	boolean	Offload TCP and IP Transmit checksum calculation (hardware support required).Applicable only for Axi-E...
tcp_rx_checksum_offload	false	false	boolean	Offload TCP Receive checksum calculation (hardware support required).Applicable only for Axi-Ethernet...
tcp_tx_checksum_offload	false	false	boolean	Offload TCP Transmit checksum calculation (hardware support required).Applicable only for Axi-Etherne...
temac_use_jumbo_frames	false	false	boolean	use jumbo frames
udp_options	true	true	boolean	Is UDP required ?
lwip_udp	true	true	boolean	Is UDP required ?
udp_ttl	255	255	integer	UDP TTL value
udp_tx_blocking	false	false	boolean	Application sending a UDP packet blocks till the pkt is txed

VIDEO STREAM DATA AND FORMAT

The GUI design on the remote computer provide the following features:

- Send the commands to the remote FPGA development board IP address “192.168.0.10”.
- Receive the image data from the development board.
- FFMPEG “ffplay.exe” decode the received bmp images to video stream at specified frame rate.

SONY IMX477/IMX682/IMX519/IMX219 AND ONSEMI AR1335 CAMERAS VIDEO STREAMING

Sony Cameras are used on Raspberry Pi interface the Kria KV260 Development Board.

LWIP ETHERNET INTERFACE

The KRIA KV260 development board has 1 Gigabit Ethernet which is connected through the RGMII interface.

5MP OV5640 SENSOR INTERFACE

The image sensor in the PCAM 5C Module is a CMOS type digital sensor model OV5640 which is 5 Megapixel (MP) color image sensor.

Image pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to KRIA KV260 development via 15-pin flat flexible cable.

5MP OV5647 SENSOR INTERFACE

The image sensor in the PCAM 5C Module is a CMOS type digital sensor model OV5640 which is 5 Megapixel (MP) color image sensor.

Image pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to KRIA KV260 development via 15-pin flat flexible cable.

8MP IMX219 SENSOR INTERFACE

The image sensor in the SONY IMX219 Module is a CMOS type digital sensor which is 8 Megapixel (MP) color image sensor. It’s support 1080p30,720p60 and 640x480p90 video resolutions and still image of max resolution of 3280x2464 pixels.

Image pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to KRIA KV260 development via 15-pin flat flexible cable.

SPECIFICATIONS

Sensor	Sony IMX219
Optical Format	Type 1/4(Diagonal 4.60mm)
Pixel	8MP
Pixel Size	1.12μm×1.12μm
Active Pixels	3280 (H) × 2464 (V)
CSI-2 Data Output	2-lane mode
Data Format	Raw Bayer 10bit
Frame Rates	1080p30, 720p60 and VGA90 video modes
Shutter Type	Rolling Shutter
FOV (Field of View)	75°(H)
Camera Board Size	24mm x 25mm

SONY 12.5MP IMX477 SENSOR

IMX477-MIPI-CS is a high-resolution digital camera. It incorporates a Sony 1/2.3" CMOS digital image sensor with an active imaging pixel array of 4056H x3040V.

SPECIFICATIONS

Sensor	Sony IMX477
Optical Format	Type 1/4(Diagonal 4.60mm)
Pixel	12.5MP
Pixel Size	1.55μm×1.55μm
Active Pixels	4056 (H) × 3040 (V) approx. 12.33 M pixels
CSI-2 Data Output	2/4-lane mode
Data Format	Raw Bayer 10/12 bits
Frame Rates	4K2Kp60, 1080p240 and VGA90 video modes
Shutter Type	Rolling Shutter
Camera Board Size	24mm x 25mm

DRIVE MODE

Drive Mode	Number of active pixels	Maximum frame rate [frames/s]	Output Interface	ADC [bit]
Full (4:3) [Normal]	4056(H) × 3040(V) 12.3MP	60	CSI-2	10
Full (4:3) [Normal]	4056(H) × 3040(V) 12.3MP	40	CSI-2	12
Full (4:3) [DOL-HDR]	4056(H) × 3040(V) 12.3MP	30/15 (DOL 2/DOL 3)	CSI-2	10
Full (16:9) 4K2K [Normal]	4056(H) × 2288(V) 9.3MP	79	CSI-2	10
Full (16:9) 4K2K [DOL-HDR]	4056(H) × 2288(V) 9.3MP	39/19 (DOL 2/DOL 3)	CSI-2	10
Full (4:3) Binning [Normal]	2028(H) × 1520(V) 3.1MP	179	CSI-2	10
Full (16:9) Binning 1080P [Normal]	2028(H) × 1128(V) 2.3MP	240	CSI-2	10
Full (16:9) Binning 720P [Normal]	1348(H) × 750(V) 1MP	240	CSI-2	10
Full (16:9) Scaling 1080P [Normal]	2024(H) × 1142(V) 2.3MP	79	CSI-2	10
Full (16:9) Scaling 720P [Normal]	1348(H) × 762(V) 1MP	79	CSI-2	10

FEATURES

- Back-illuminated and stacked CMOS image sensor Exmor RS
- Digital Overlap High Dynamic Range (DOL-HDR) mode with raw data output.
- High signal to noise ratio (SNR).
- Full resolution @60 frame/s (Normal), 4K2K @60 frame/s (Normal), 1080p @240 frame/s
- Full resolution @40 frame/s (12 bit Normal), Full resolution @30 frame/s (DOL-HDR, 2 frame)
- Output video format of RAW12/10/8, COMP8.
- Power Save Mode
- Pixel binning readout and V sub-sampling function.
- Independent flipping and mirroring.
- Input clock frequency 6 to 27 MHz
- CSI-2 serial data output (MIPI 2lane/4lane, Max. 2.1 Gbps/lane, D-PHY spec. ver. 1.2 compliant)
- 2-wire serial communication.
- Two PLLs for independent clock generation for pixel control and data output interface.
- Ambient Light Sensor (ALS)
- Fast mode transition.

- Dual sensor synchronization operation (Multi camera compatible)
- 7 K bit of OTP ROM for users.
- Built-in temperature sensor
- 10-bit/12-bit A/D conversion on chip
- 92-pin high-precision ceramic package

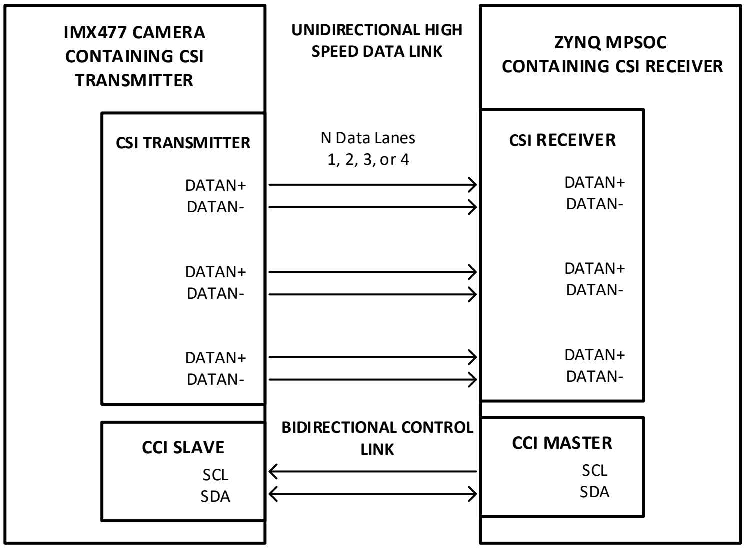
INTERFACE

IMX477 Camera module and the KRIA KV260 development board are connected through the FPC flexible flat cable. Camera pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to development board via 15-pin flat flexible cable. Raspberry Pi camera module interface only support 2 MIPI lanes which directly connected directly to the Zynq UltraScale+ MPSoC HPA bank.

The interface is defined as follows:

PIN	SIGNAL NAME	DESCRIPTION
PIN 1	Ground	Ground
PIN 2	MIPI_LAN0_N	Camera CMOS LANE0 Data Negative Output
PIN 3	MIPI_LAN0_P	Camera CMOS LANE0 Data Positive Output
PIN 4	Ground	Ground
PIN 5	MIPI_LAN1_N	Camera CMOS LANE1 Data Negative Output
PIN 6	MIPI_LAN1_P	Camera CMOS LANE1 Data Positive Output
PIN 7	Ground	Ground
PIN 8	MIPI_CLK_N	Camera CMOS Clock Negative Output
PIN 9	MIPI_CLK_P	Camera CMOS Clock Data Positive Output
PIN 10	Ground	Ground
PIN 11	CAM_PWUP	Camera CMOS Power-on Control Signal
PIN 12	CAM_CLK	Camera CMOS Input Clock Signal
PIN 13	CAM_SCL	Camera CMOS I2C Clock Signal
PIN 14	CAM_SDA	Camera CMOS I2C Data Signal
PIN 15	+3.3V	Power Supply 3.3V

The Camera Serial Interface (CSI), a division from the MIPI Alliance which have formulated set of interface standards to standardize the interface of mobile devices such as camera and display. MIPI's full name is "Mobile Industry Processor Interface", MIPI DSI correspond to video display and MIPI CSI correspond to video input standards respectively. CSI interface is divided into physical layer (D-PHY) and protocol layer (CSI-2).



CONFIGURATION

For this design application, the IMX477 camera device is configured to output 1920x1080p video frames with RGB 30 bits pixel format, at 60 frames rate. Both the camera device and MIPI CSI-2 subsystem module is configured in advance.

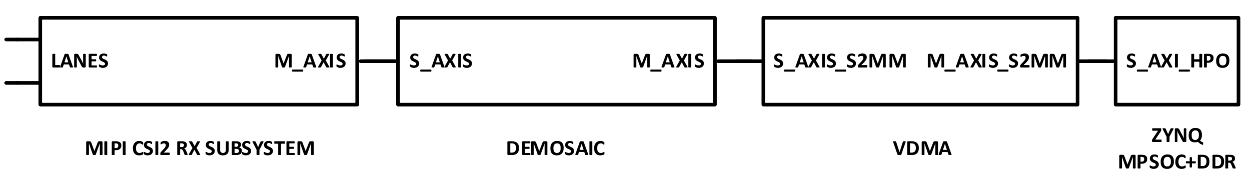
VIDEO INPUT SUBSYSTEMS

The MIPI CSI-2 subsystem module is used to captures images from IMX477 camera sensor and outputs to axi4 stream video stream. Demosaic module is used to convert raw axi4 stream video data to rgb format video stream.

VDMA IP is used convert Demosaic video stream data to AXI4 memory map format and DDR fetching decoding and execution via AXI_HP interface.

VDMA (Video Direct Memory Access) is a general DMA, specially used for video and image applications. It has an AXI4 Memory Map interface for reading (writing channel) and writing (read channel) video to memory.

The display port controller in the PS is coupled to the STDP4320 demultiplexer on the carrier which consists of dual mode output ports configured as DP/HDMI.



IMX477 CONFIGURATION REGISTERS

ADDRESS	REGISTER NAME	VALUE
0x0342	LINE_LEN_MSB	0X31
0x0343	LINE_LEN_LSB	0XC4
0x0344	X_ADD_STA_MSB	0X00
0x0345	X_ADD_STA_LSB	0X00
0x0348	X_ADD_END_MSB	0X0F
0x0349	X_ADD_END_LSB	0XD7
0x0346	Y_ADD_STA_MSB	0X01
0x0347	Y_ADD_STA_LSB	0XB8
0x034A	Y_ADD_END_MSB	0X0A
0x034B	Y_ADD_END_LSB	0X27
0x040C	DIG_CROP_WIDTH_MSB	0X0F
0x040D	DIG_CROP_WIDTH_LSB	0XD7
0x040E	DIG_CROP_HEIGHT_MSB	0X04
0x040F	DIG_CROP_HEIGHT_LSB	0XB0
0x034C	X_OUT_SIZE_MSB	0X08
0x034D	X_OUT_SIZE_LSB	0X00
0x034E	Y_OUT_SIZE_MSB	0X04
0x034F	Y_OUT_SIZE_LSB	0XB0
0x0408	DIG_CROP_X_OFFSET_MSB	0X00
0x0409	DIG_CROP_X_OFFSET_LSB	0X00
0x040A	DIG_CROP_Y_OFFSET_MSB	0X00
0x040B	DIG_CROP_Y_OFFSET_LSB	0X00
0x0381	X_ENV_INC_CONST	0X01
0x0383	X_ODD_INC_CONST	0X01
0x0385	Y_ENV_INC_CONST	0X01
0x0387	Y_ODD_INC	0X01
0x0900	BINNING_MODE	0X01
0x0901	BINNING_HV	0X12
0x0902	BINNING_WEIGHTING	0X02
0x3F0D	ADC_BIT_SETTING	0X0A
0x0401	SCALE_MODE	0X01
0x0404	SCALE_M_MSB	0X00
0x0405	SCALE_M_LSB	0X20
0x0301	IVTPXCK_DIV	0X05
0x0303	IVTSYCK_DIV	0X02
0x0305	IVT_PREPLLCK_DIV	0X04
0x0306	PLL_IVT_MPY_MSB	0X01
0x0307	PLL_IVT_MPY_LSB	0X5E
0x0309	IOPPXCK_DIV	0X0C
0x030B	IOPSYCK_DIV	0X02
0x030D	IOP_PREPLLCK_DIV	0X02
0x030E	IOP_MPY_MSB	0X00
0x030F	IOP_MPY_LSB	0X96
0x0310	PLL_MULTI_DRV	0X01
0x0820	REQ_LINK_BIT_RATE_MSB	0X07
0x0821	REQ_LINK_BIT_RATE_LMSB	0X08
0x0822	REQ_LINK_BIT_RATE_MLSB	0X00
0x0823	REQ_LINK_BIT_RATE_LSB	0X00
0x080A	TCLK_POST_EX_MSB	0X00
0x080B	TCLK_POST_EX_LSB	0X7F
0x080C	THS_PRE_EX_MSB	0X00
0x080D	THS_PRE_EX_LSB	0X4F
0x080E	THS_ZERO_MIN_MSB	0X00
0x080F	THS_ZERO_MIN_LSB	0X77
0x0810	THS_TRAIL_EX_MSB	0X00
0x0811	THS_TRAIL_EX_LSB	0X5F
0x0812	TCLK_TRAIL_MIN_MSB	0X00
0x0813	TCLK_TRAIL_MIN_LSB	0X57
0x0814	TCLK_PREP_EX_MSB	0X00
0x0815	TCLK_PREP_EX_LSB	0X4F
0x0816	TCLK_ZERO_EX_MSB	0X01
0x0817	TCLK_ZERO_EX_LSB	0X27

0x0818	TLPX_EX_MSB	0X00
0x0819	TLPX_EX_LSB	0X3F
0x3E37	PDAF_CTRL1_0	0X00
0x3F50	POWER_SAVE_ENABLE	0X00
0x3F56	LINE_LEN_INCLK_MSB	0X01
0x3F57	LINE_LEN_INCLK_LSB	0X6C
0x0204	ANA_GLOBAL_GAIN_U	0X03
0x0205	ANA_GLOBAL_GAIN_L	0X7F
0x020E	DIG_GAIN_GR_U	0X02
0x020F	DIG_GAIN_GR_L	0X00
0x0210	DIG_GAIN_R_U	0X03
0x0211	DIG_GAIN_R_L	0X00
0x0212	DIG_GAIN_B_U	0X02
0x0213	DIG_GAIN_B_L	0X00
0x0214	DIG_GAIN_GB_U	0X02
0x0215	DIG_GAIN_GB_L	0X00
0x0202	COARSE_INTEGRATION_TIME_MSB	0X02
0x0203	COARSE_INTEGRATION_TIME_LSB	0X00
0x0100	MODE_SEL	0X01

16MP IMX519 SENSOR INTERFACE

The image sensor in the PCAM 5C Module is a CMOS type digital sensor model OV5640 which is 5 Megapixel (MP) color image sensor.

Image pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to KRIA KV260 development via 15-pin flat flexible cable.

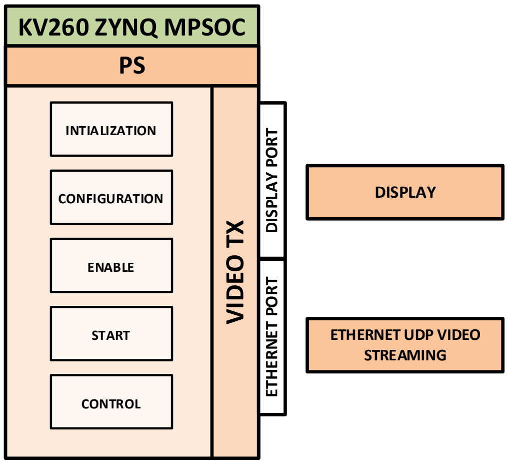
64MP IMX682 SENSOR INTERFACE

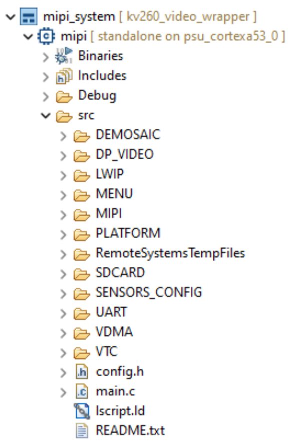
The image sensor in the PCAM 5C Module is a CMOS type digital sensor model OV5640 which is 5 Megapixel (MP) color image sensor.

Image pixels data is transferred over dual -lane MIPI CSI-2 interface which is connected to KRIA KV260 development via 15-pin flat flexible cable.

VITIS PROGRAM DEVELOPMENT

The program design is mainly divided into five parts, video initialization, configuration, control, and transmission. Below figure present the overall software structure of implemented application, which shows the program and data flow.





In the main function, the initialization of the 1 frame buffered is performed for VDMA1. The VIDEO1_MAX_FRAME macro definition is set to 1, the initial initialization of pFrame0 pointer, pFrame0 points to the cache of camera 1.

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, **1**=128x125, **2**=255x255,	
sharp		
blur		

embross	**3**=1920x1080	
hsl		
hsv		
cgain		
sobel		

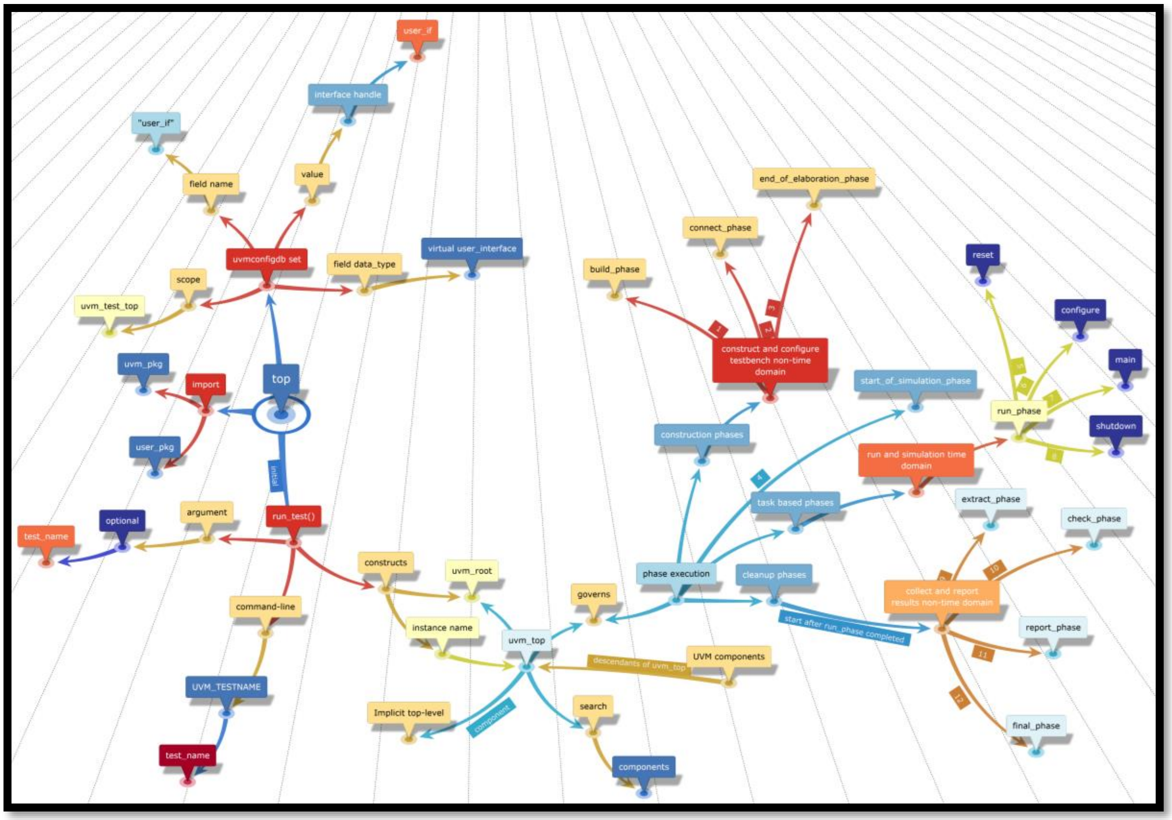
Set of parameters for image filter type and their size are assigned for current test values as shown in table 2 via defined macro in frame enable package according to table 1.

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.

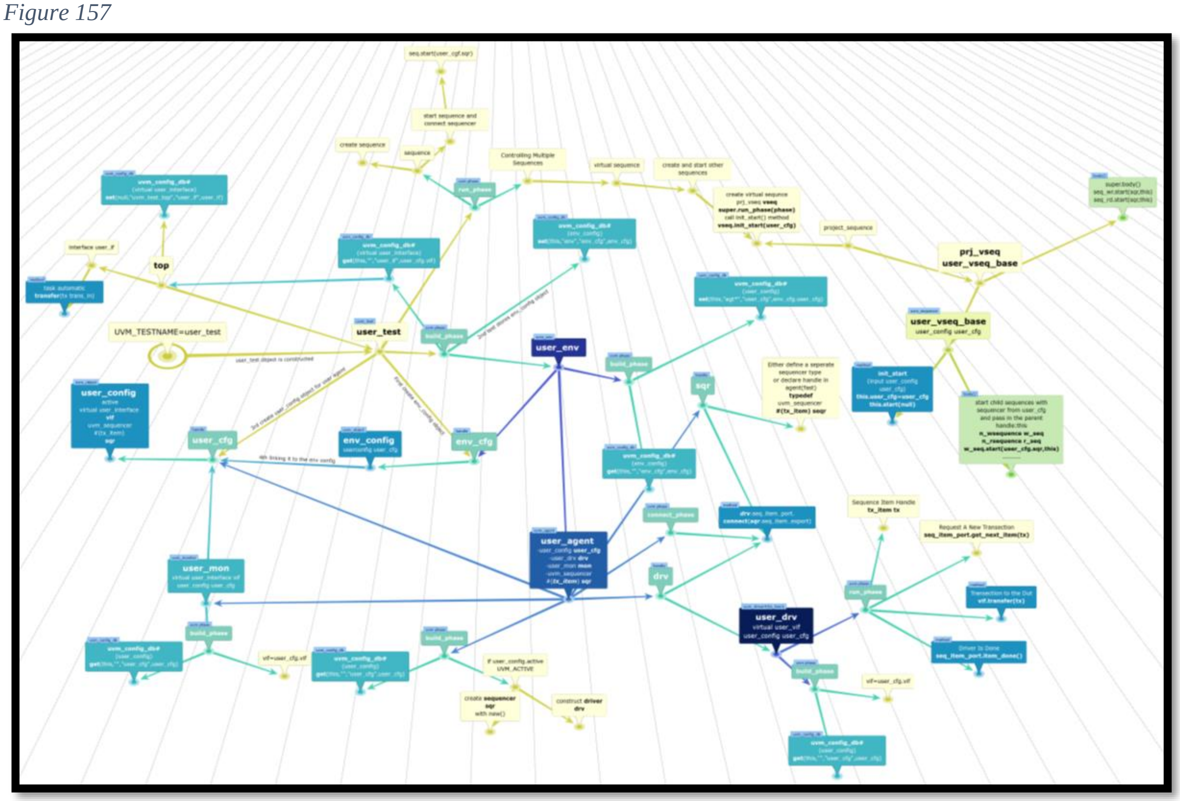
Register Name	Offet	Description
initAddr	8'h00	
oRgbOsharp	8'h00	
oEdgeType	8'h04	
filter_id	8'h08	
aBusSelect	8'h0C	
threshold	8'h10	
videoChannel	8'h14	
dChannel	8'h18	
cChannel	8'h1C	
kls_k1	8'h20	
kls_k2	8'h24	
kls_k3	8'h28	
kls_k4	8'h2C	
kls_k5	8'h30	
kls_k6	8'h34	
kls_k7	8'h38	
kls_k8	8'h3C	
kls_k9	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	



user_if
interface handle
"user_if"
value
field name
uvmconfigdb set
scope
field data_type
virtual user_interface
uvm_test_top
uvm_pkg
import
top
user_pkg
initial
run_test()
argument
optional
test_name
command-line
UVM_TESTNAME
test_name
constructs
uvm_root
instance name
uvm_top
governs
Implicit top-level
components
search
components
descendants of uvm_top
UVM components
phase execution
construction phases
task based phases
cleanup phases
start after run_phase completed
build_phase
connect_phase
end_of_elaboration_phase
construct and configure testbench non-time domain
start_of_simulation_phase
run and simulation time domain
run_phase
reset
configure
main
shutdown
extract_phase
check_phase
collect and report results non-time domain
report_phase
final_phase

Figure 157



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

Figure 158

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 159

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 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 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.

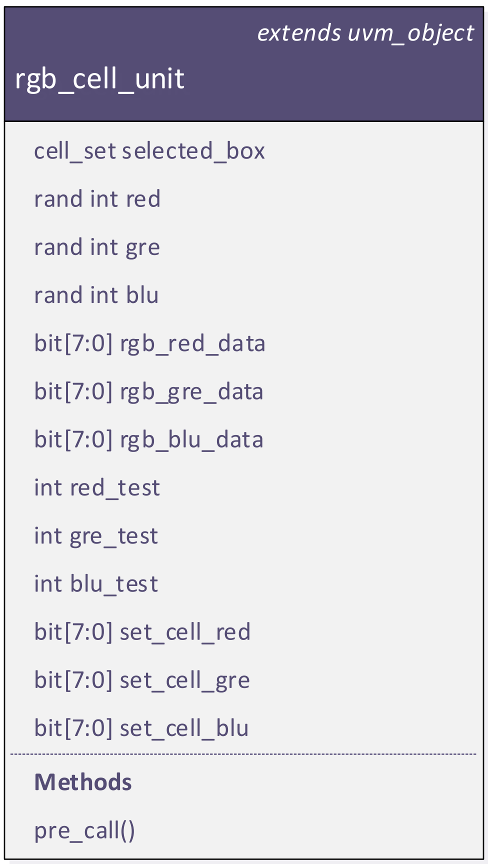
After start_of_simulation phase run phase gets initiated which is used for the stimulus generation and checking activities of the test bench and execution of all uvm_component in parallel. Each uvm_component run phase task run in parallel where main phase gets initiated, and stimulus of specified test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