Video Frame Processing

Contents

[1 Introduction 4](#_Toc94323607)

[2 Architecture 5](#_Toc94323608)

[3 Features 6](#_Toc94323609)

[4 Clocks 7](#_Toc94323610)

[5 Frame Time 8](#_Toc94323611)

[6 Frame Rates at Common Resolutions 9](#_Toc94323612)

[7 VFP 10](#_Toc94323613)

[8 HSL Filter 11](#_Toc94323614)

[9 HSV Filter 13](#_Toc94323615)

[10 Sharp Filter 14](#_Toc94323616)

[11 Blur Filter 15](#_Toc94323617)

[12 Emboss Filter 16](#_Toc94323618)

[13 Sobel Filter 17](#_Toc94323619)

[14 YCbCr Color Space 18](#_Toc94323620)

[15 Color Correction Matrix 19](#_Toc94323621)

[16 Camera Raw Data 20](#_Toc94323622)

[17 Image Read Interface 21](#_Toc94323623)

[18 Three Taps data 22](#_Toc94323624)

[19 Four Taps data 22](#_Toc94323625)

[20 Pixel Coordinates 22](#_Toc94323626)

[21 Camera Raw Data 24](#_Toc94323627)

[22 Histogram 25](#_Toc94323628)

[23 Testbench 27](#_Toc94323629)

[24 D5M DRIVER 28](#_Toc94323630)

[Figure 1 : Video Frame Processing Skeleton Architecture 5](#_Toc94323663)

[Figure 2 10](#_Toc94323664)

[Figure 3 11](#_Toc94323665)

[Figure 4 : HSL Filter Wave Diagram 11](#_Toc94323666)

[Figure 5 : Hue Numerator Logic 11](#_Toc94323667)

[Figure 6 : Hue Denominator Logic 12](#_Toc94323668)

[Figure 7 : Hue Degree Logic 12](#_Toc94323669)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 12](#_Toc94323670)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 12](#_Toc94323671)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 12](#_Toc94323672)

[Figure 9 13](#_Toc94323673)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 13](#_Toc94323674)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 13](#_Toc94323675)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 13](#_Toc94323676)

[Figure 10 14](#_Toc94323677)

[Figure 11 15](#_Toc94323678)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 18](#_Toc94323679)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 18](#_Toc94323680)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 18](#_Toc94323681)

[Figure 12 19](#_Toc94323682)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 19](#_Toc94323683)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 19](#_Toc94323684)

[Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. 19](#_Toc94323685)

[Figure 13 : camera raw data module 20](#_Toc94323686)

[Figure 14 : General view of camera raw data flow 20](#_Toc94323687)

[Figure 15 21](#_Toc94323688)

[Figure 16 22](#_Toc94323689)

[Figure 17 22](#_Toc94323690)

[Figure 18 22](#_Toc94323691)

[Figure 19 23](#_Toc94323692)

[Figure 20 24](#_Toc94323693)

[Figure 21 28](#_Toc94323694)

# Introduction

The purpose of this design to take raw Bayer format data and convert into video 16-bit YCbCr [4:2:2] (YUV) color space. This design reads camera input data, then transform into rgb color space of 24-bit data bus with 8 bits each for the red pixel, green pixel, and blue pixel. Each rgb pixel either filtered or converted into color space. Sharp, blur, emboss and sobel filtered are used in this design. RGB into Ycbcr and color correction space are implemented in this design.

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

# Architecture

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

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



Figure : Video Frame Processing Skeleton Architecture

# Features

Input format: Raw Bayer format

Output format: 16-bit YCbCr [4:2:2] (YUV) color space

24-bit User AXI Stream input.

**Filters:**

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

# Clocks

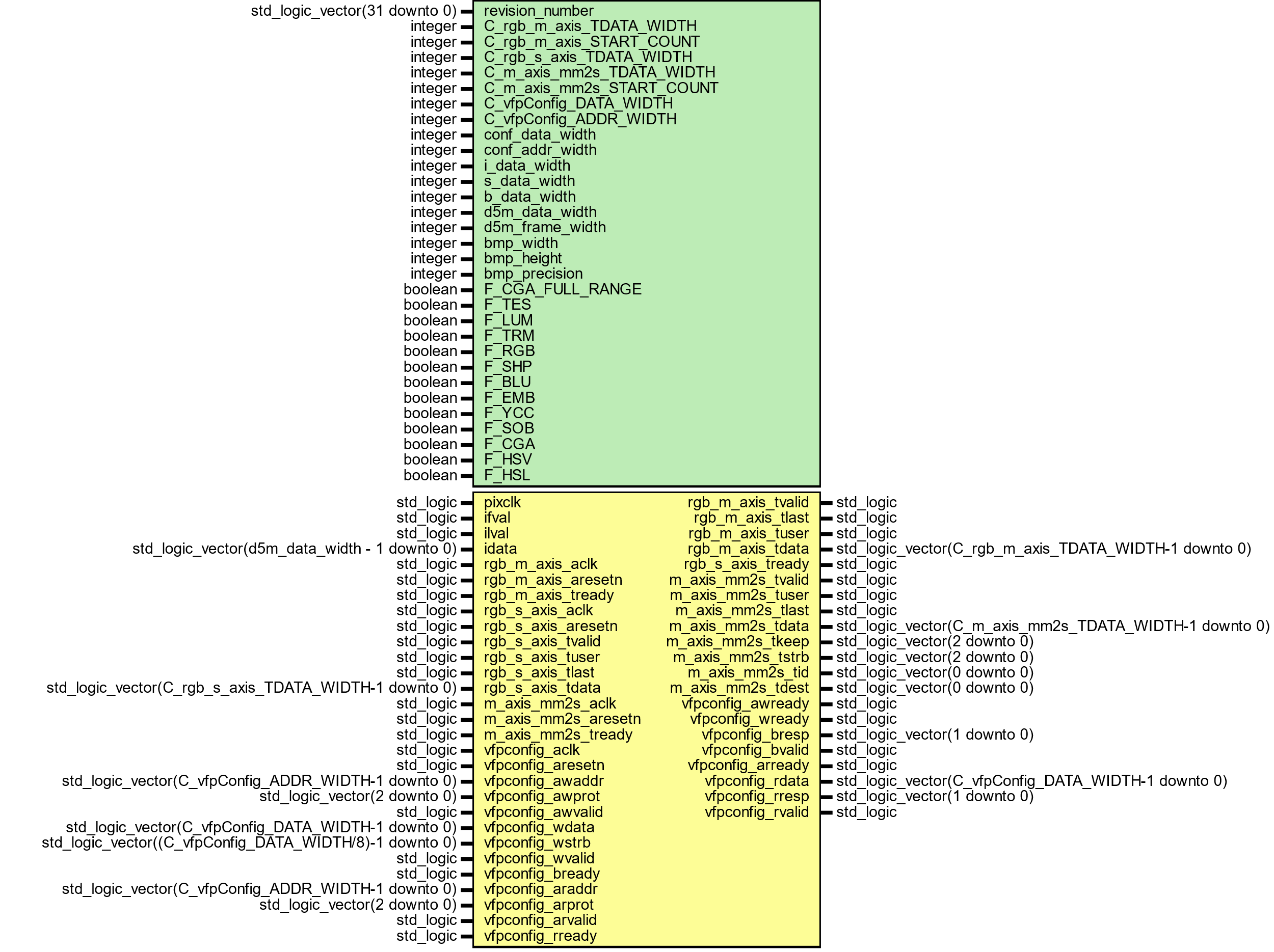
There are two clocks used in this design, pixel clock is nominally the pixel clock rate, with a

set pll programed design frequency of 96MHz or below, and system clock which is a 150 Mhz.

# Frame Time

# Frame Rates at Common Resolutions

# VFP



Figure

# HSL Filter

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.

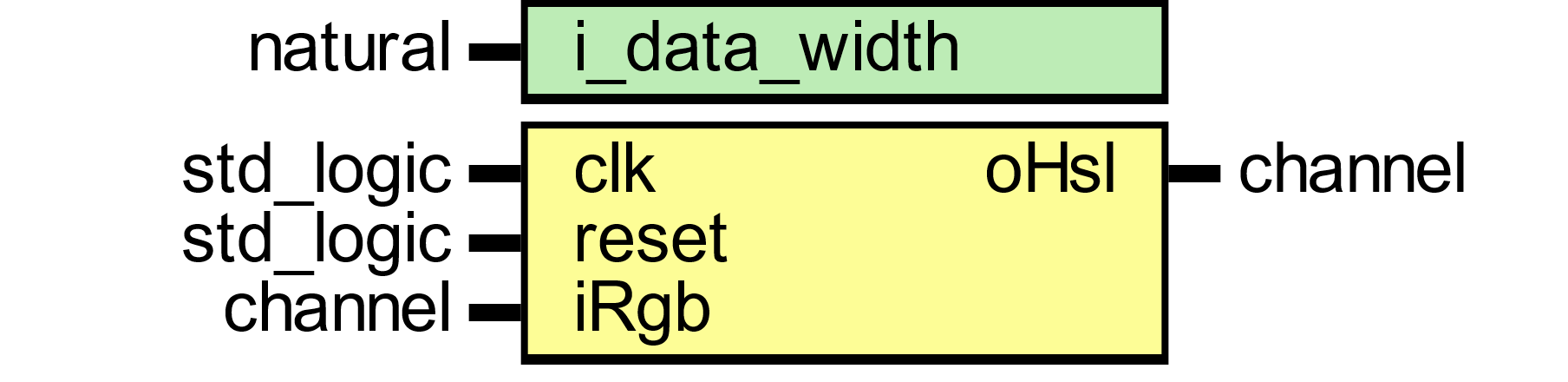


Figure 3

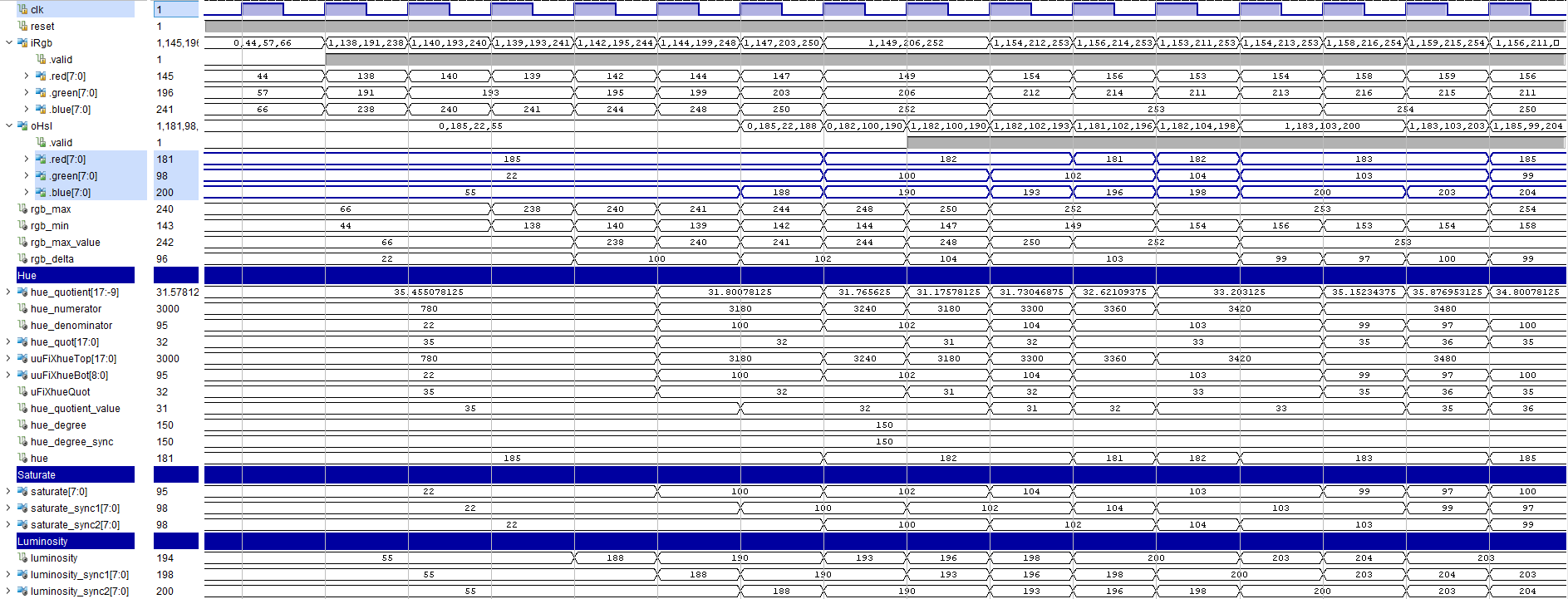


Figure 4 : HSL Filter Wave Diagram



Figure : Hue Numerator Logic



Figure : Hue Denominator Logic

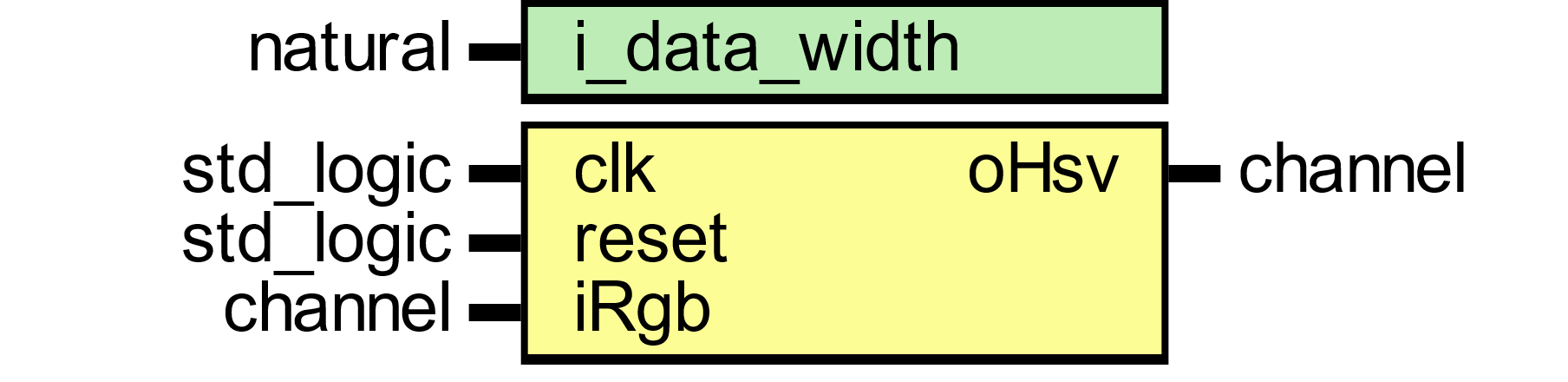


Below figures shows rgb color space conversion implementation into six regions of the hexagon images.

Figure : Hue Degree Logic

|  |  |  |  |
| --- | --- | --- | --- |
| ***RGB image*** | ***HSL converted image*** | ***RGB image*** | ***HSL converted image*** |
| Figure 8 : 1st and 3rd figure are rgb image and whereas 2nd and 4th are hsl simulated results. | | | |
| ***HSL Image*** | ***Hue channel*** | ***Saturate channel*** | ***Luminosity channel*** |
| Figure 9 : 1st image is hsl image, 2nd represent hue channel, 3rd saturate channel and 4th luminosity channel. | | | |
| ***HSL image*** | ***Hue channel*** | ***Saturate channel*** | ***Luminosity channel*** |
| Figure 10 : 1st image is hsl image, 2nd represent hue channel, 3rd saturate channel and 4th luminosity channel. | | | |

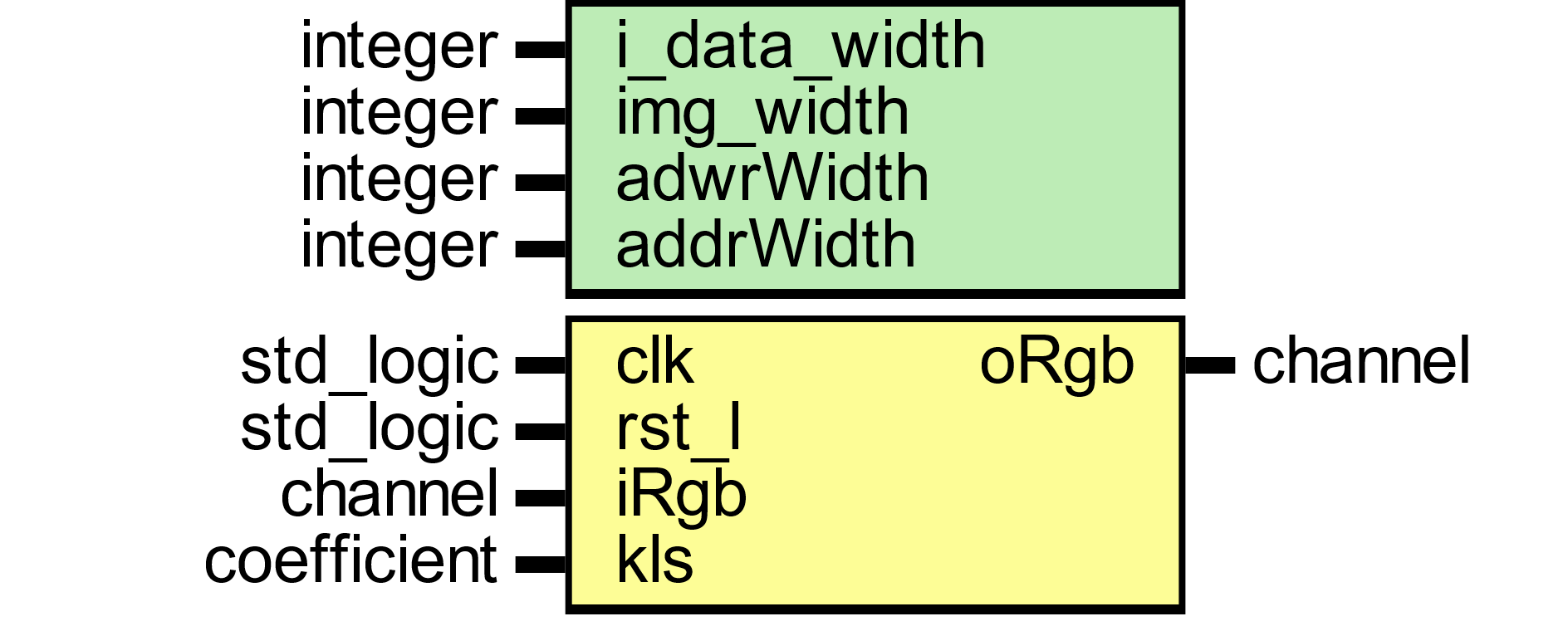
# HSV Filter



Figure

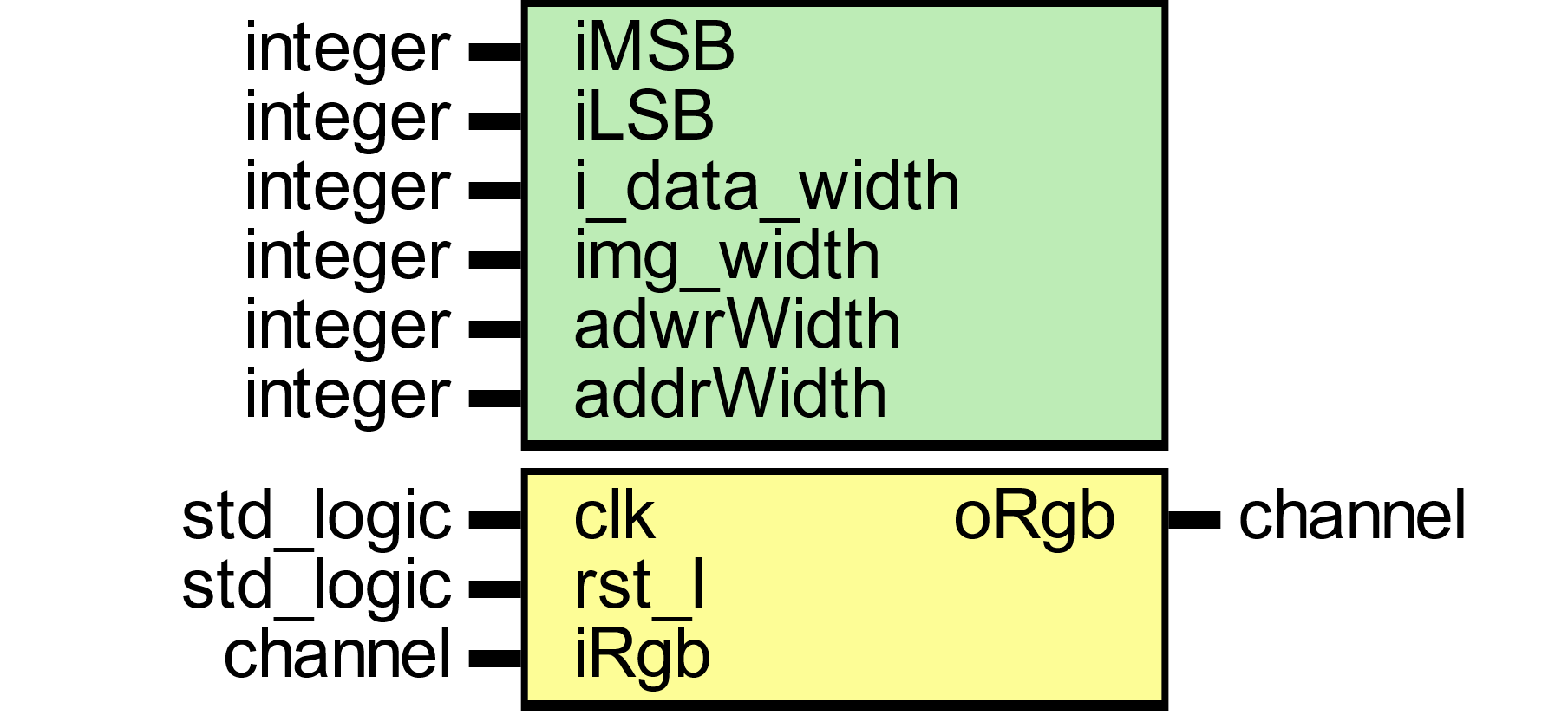
|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  |  |  |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  | A picture containing stationary  Description automatically generated | Shape  Description automatically generated |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |

# Sharp Filter



Figure

# Blur Filter



Figure

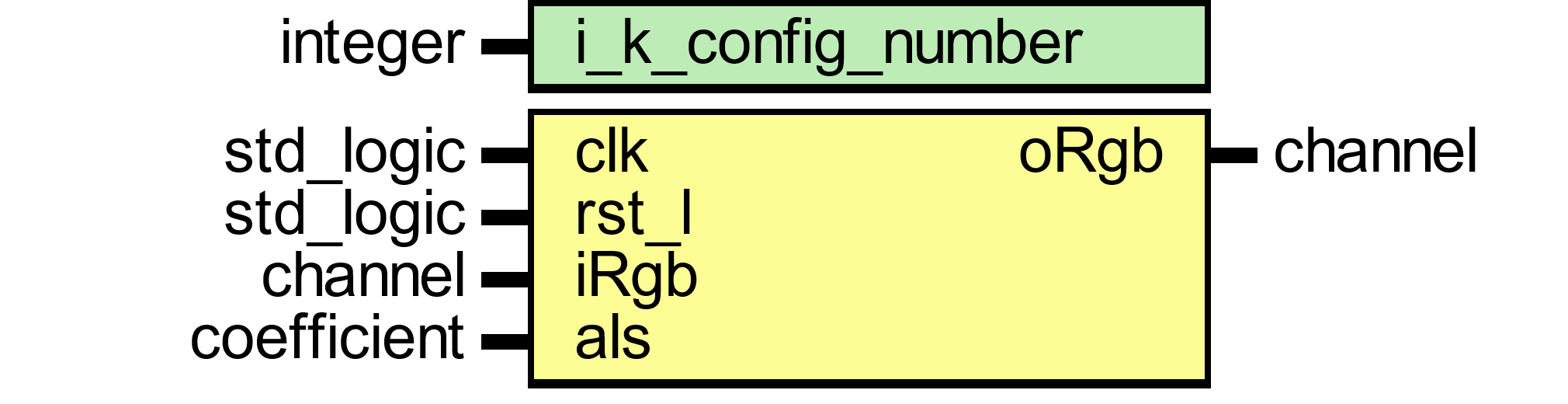
# Emboss Filter

# Sobel Filter

# YCbCr Color Space

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  | A picture containing envelope, stationary  Description automatically generated | A picture containing envelope  Description automatically generated |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  | A picture containing stationary  Description automatically generated | Shape  Description automatically generated |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |

# Color Correction Matrix



Figure

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  | A picture containing envelope, stationary  Description automatically generated | A picture containing envelope  Description automatically generated |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |
|  | A picture containing stationary  Description automatically generated | Shape  Description automatically generated |  |
| Figure 8 : 1st and 2nd figure are rgb image and whereas 3rd and 4th are hsl simulated results. | | | |

# Camera Raw Data

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

Buffersize is auto size supported which is controlled by input line valid from host camera and maximum is set to default value of 3071.

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.

Default: MAX\_LINE\_IMAGE\_WIDTH=3071

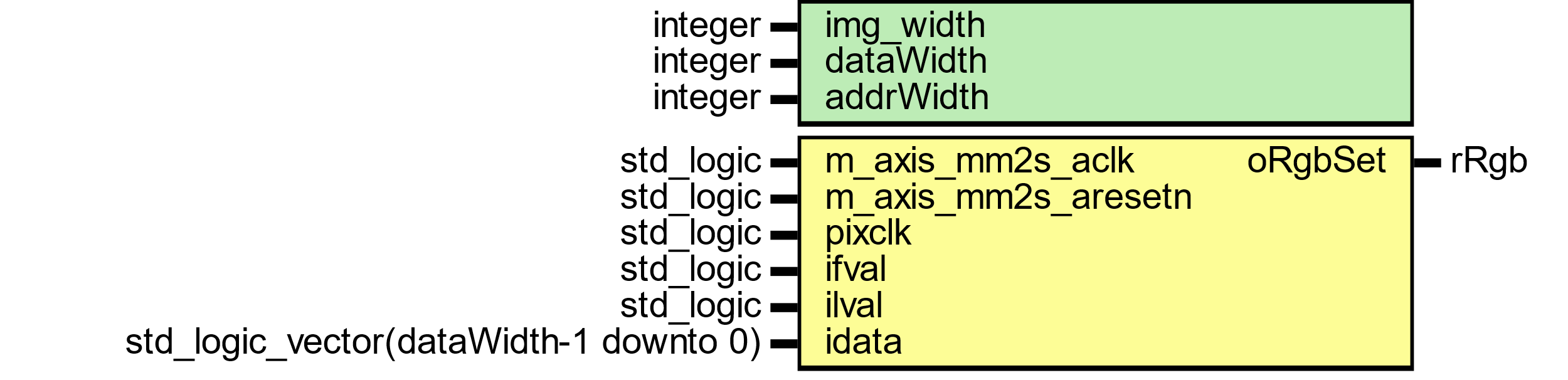


Figure : camera raw data module

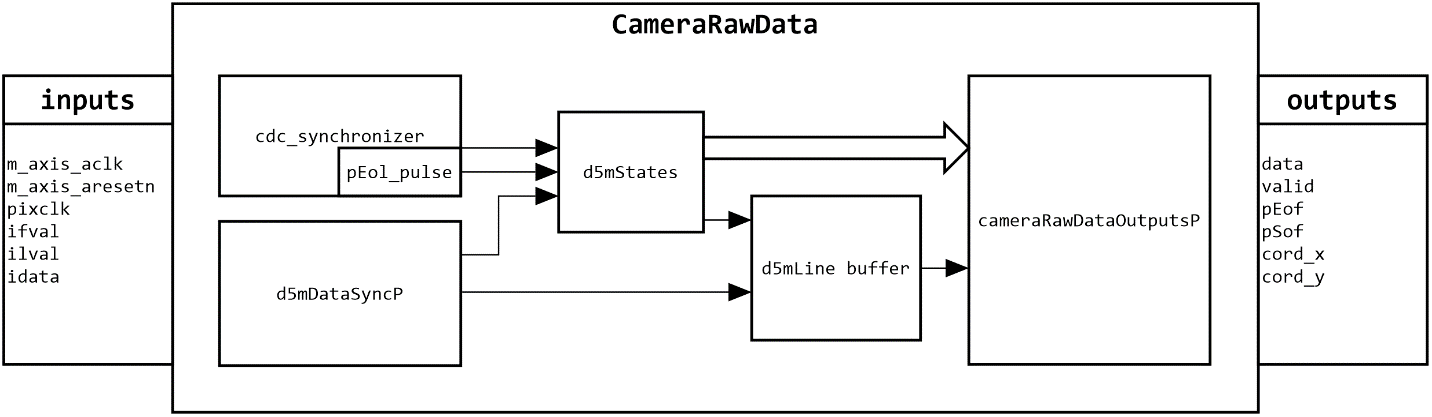


Figure : General view of camera raw data flow

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

# Image Read Interface

Figure

# Three Taps data



Figure

# Four Taps data



Figure

# Pixel Coordinates



Figure





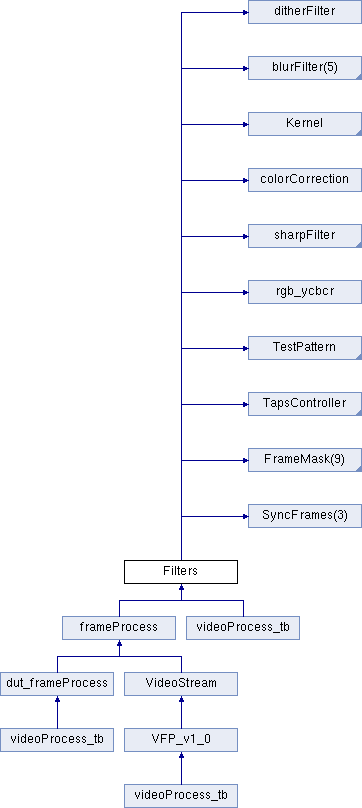
Figure



Figure

# Camera Raw Data

# Histogram

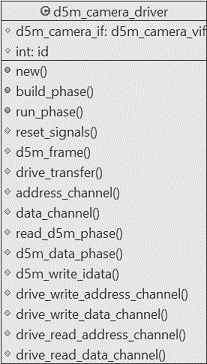


# Testbench



# D5M 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.



Figure

**DECLARATION:**

This uvm driver class is derived from uvm component.

|  |
| --- |
| class d5m\_camera\_driver extends uvm\_driver #(d5m\_trans); |

**HANDLES**

|  |
| --- |
| protected virtual d5m\_camera\_if d5m\_camera\_vif;  protected int id; |

**NEW CONSTRUCT**

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

|  |
| --- |
| function new (string name, uvm\_component parent);  super.new(name, parent);  endfunction: new |

**BUILD\_PHASE**

Build method run top-down and the rest of the phases run bottom-up. In this phase, config the dut interface handle through get method. The uvm\_config\_db parameterized class provides a convenience interface on top of uvm\_resource\_db is used to for reading resource database. The uvm\_config\_db is derived from uvm\_resource\_db class. Get value of field\_name “d5m\_camera\_vif”, using component cntxt ”this” as starting search point.

|  |
| --- |
| function void build\_phase (uvm\_phase phase);  super.build\_phase(phase);  if (!uvm\_config\_db#(virtual d5m\_camera\_if)::get  (this, "", "d5m\_camera\_vif", d5m\_camera\_vif))  `uvm\_fatal("NOVIF", {"virtual interface must be set for:  ",get\_full\_name(), ".d5m\_camera\_vif"});  endfunction: build\_phase |

**RUN\_PHASE**

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

|  |
| --- |
| virtual task run\_phase (uvm\_phase phase);  fork  reset signals();  d5m\_frame();  join  endtask: run\_phase |

**RESET SIGNALS**

Reset the dut axi4 lite and d5m\_cam\_mod input signals when system reset is asserted from low to high.

|  |
| --- |
| virtual protected task reset\_signals();  forever begin  @(posedge d5m\_camera\_vif.ARESETN);  d5m\_camera\_vif.axi4.AWADDR <= 8'h0;  d5m\_camera\_vif.axi4.AWPROT <= 3'h0;  d5m\_camera\_vif.axi4.AWVALID <= 1'b0;  d5m\_camera\_vif.axi4.WDATA <= 32'h0;  d5m\_camera\_vif.axi4.WSTRB <= 4'h0;  d5m\_camera\_vif.axi4.WVALID <= 1'b0;  d5m\_camera\_vif.axi4.BREADY <= 1'b0;  d5m\_camera\_vif.axi4.ARADDR <= 8'h0;  d5m\_camera\_vif.axi4.ARPROT <= 3'h0;  d5m\_camera\_vif.axi4.ARVALID <= 1'b0;  d5m\_camera\_vif.axi4.RREADY <= 1'b0;  d5m\_camera\_vif.d5p.iImageTypeTest <= 1'b0;  d5m\_camera\_vif.d5p.iReadyToRead <= 1'b0;  d5m\_camera\_vif.d5p.fvalid <= 1'b0;  d5m\_camera\_vif.d5p.lvalid <= 1'b0;  end  endtask: reset\_signals |

**D5M FRAME**

In this method, drive the signals from defined seq in uvm\_sequence.

|  |
| --- |
| virtual protected task d5m\_frame();  forever begin  @(posedge d5m\_camera\_vif.clkmm);  seq\_item\_port.get\_next\_item(req);  drive\_transfer(req);  seq\_item\_port.item\_done();  end  endtask: d5m\_frame |

**DRIVE TRANSFER**

This method which is master to dut axi4lite interface write/read data at given address using bus handshaking protocol. First valid address is transmitted and then wait for valid response in given time of 61 clock cycles in axi4\_address method. Timeout accord on 62 clock cycle, if no response is asserted high on bvalid signal from dut and timeout is flagged using uvm\_error macro. If valid response is asserted then axi4\_data method write/read data depending on case statement. Once axi4 bus config the video process module in dut then write/read operation can be initiated by calling the d5m\_pixel method.

|  |
| --- |
| virtual protected task drive\_transfer (d5m\_trans d5m\_tx);  axi4\_address(d5m\_tx);  axi4\_data(d5m\_tx);  d5m\_pixel(d5m\_tx);  endtask: drive\_transfer |

**AXI4 ADDRESS CHANNEL**

In this method, write/read axi4 address channel.

|  |
| --- |
| virtual protected task axi4\_address (d5m\_trans d5m\_tx);  case (d5m\_tx.d5m\_txn)  AXI4\_WRITE : axi4\_write\_address(d5m\_tx);  AXI4\_READ : axi4\_wread\_address(d5m\_tx)  endcase  endtask: axi4\_address |

**AXI4 DATA CHANNEL**

In this method, write/read axi4 data channel.

|  |
| --- |
| 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 |