

# Microelectronic Systems

## WS16/17

---

### Lab 3: HDMI Design on ZedBoard

Report submitted by :

**Dayana Jose**

Matrikelnr : 752492

**Kajal Poovaiah Nellira**

Matrikelnr : 752494

**13-January-2017**

## Content

1. Introduction .....	3
2. Implementation.....	4
Part 1 Study VHDL Design of HDMI project.....	4
2.1 Task 1.....	4
2.2 Task 2.....	5
2.3 Task 3.....	5
Part 2 Design of camera emulation.....	6
2.4 Task 1.....	6
2.5 Task 2.....	13
2.6 Task 3.....	14

## 1 Introduction

The objective of the lab is to test the HDMI display controller with the required outputs.

The project needs are tested based on the description on how to get the project run. Data captured by a camera is displayed on the HDMI output of the ZedBoard. The camera needs to get connected to the Pmods of the ZedBoard, the screen to the HDMI transmitter of the ZedBoard. Both interfaces are connected to the PL side of Zynq device.

The camera is not available to perform a real test scenario here. The testing setup is available but the test pattern has to be provided.

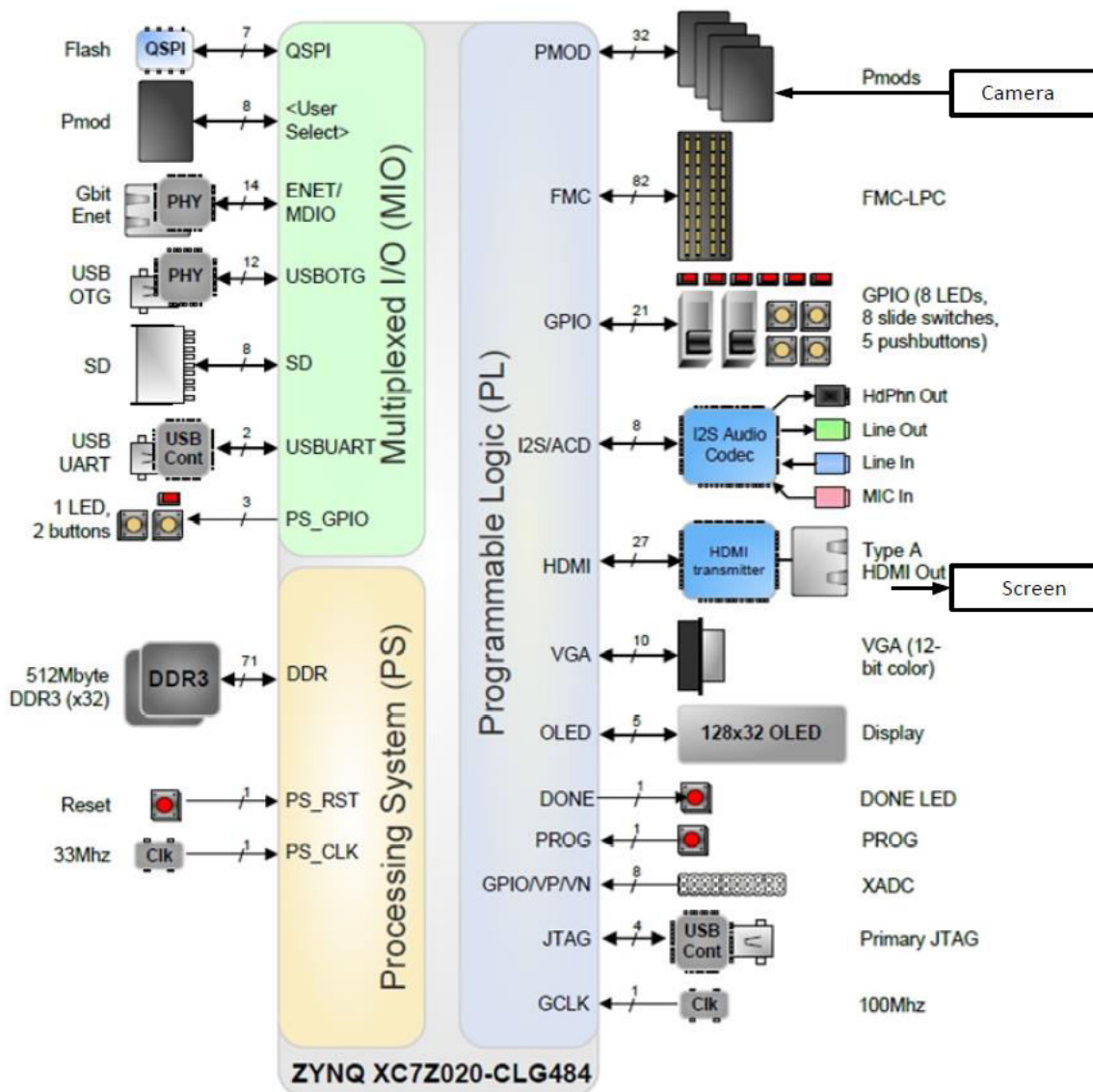


Figure 1. ZedBoard connections:

## 2 Implementation

### Part1: Study VHDL Design of HDMI project

#### 2.1 : Task1

##### Task Description:

Draw a block diagram of the top level RTL design. Name all input and output ports and also all internal signals which connect the two components HDMI\_V1 and IMG\_GENERATOR (RTL Analysis -> Open Elaborated Design).

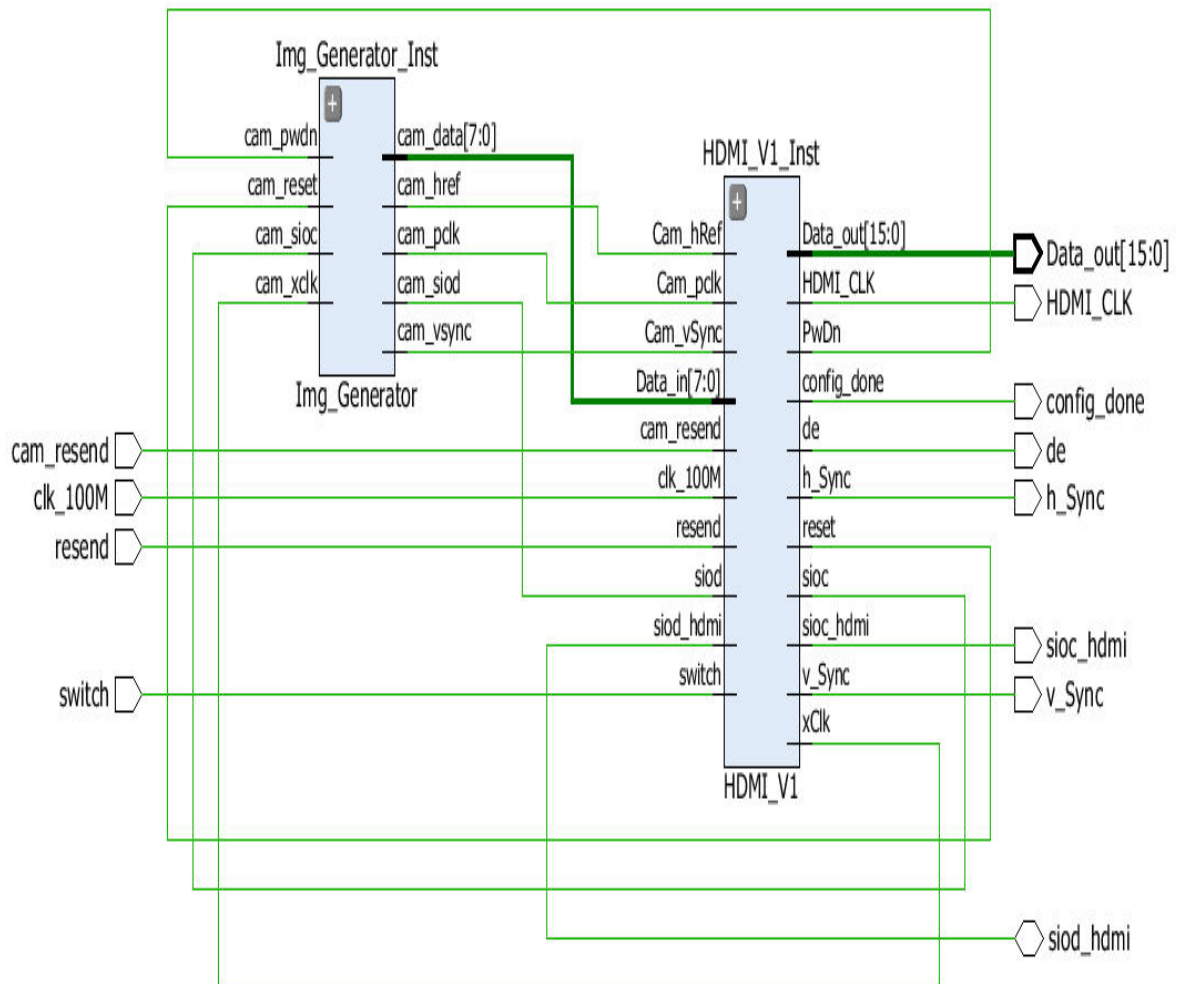


Figure 2: Top level RTL design

### Analysis:

The top level RTL schematic can be seen in Figure 2. The input as well as the output pins of the modules is highlighted, as well as the input and output pins of the top level design. The interconnection signals between the two modules (IMG\_GENERATOR and HDMI\_V1) are also highlighted on each connection. This can explain much more clearly how the entire design works.

## 2.2: Task 2

### Task-Description:

Describe the meaning of all output signals of HDMI\_V1. For this do research on HDMI ADV7511 transmitter. Use a table format with two columns: signal name and description!

### Elaboration:

The output signals of HDMI\_V1 are presented in the table below:

<i><b>Signal Name</b></i>	<i><b>Signal Description</b></i>
Data_out [15:0]	Video Data Output
HDMI_CLK	Video Clock Output
h_Sync	Horizontal Sync Output
v_Sync	Vertical Sync Output
config_done	Configuration Done (to LED)
de	Data Enable Signal for Digital Video
sioc_hdmi	I2C Interface (clock signal)
siod_hdmi	I2C Interface (data) - bidirectional

### Analysis:

All the output signals of the HDMI\_V1 module are presented in the previous table, along with a description for each of these signals. From these the overall functioning of the module can be deduced.

## 2.3: TASK 3

### Task-Description:

The component IMG\_GENERATOR emulates the camera. Do research on OV7670 camera (data sheet). Describe each signal provided by the camera (assume a resolution setting of 640x480 and an output format of Y/Cb/Cr 4:2:2) and check with block diagram of task1! Use a table format with two columns: signal name and description.

Elaboration:

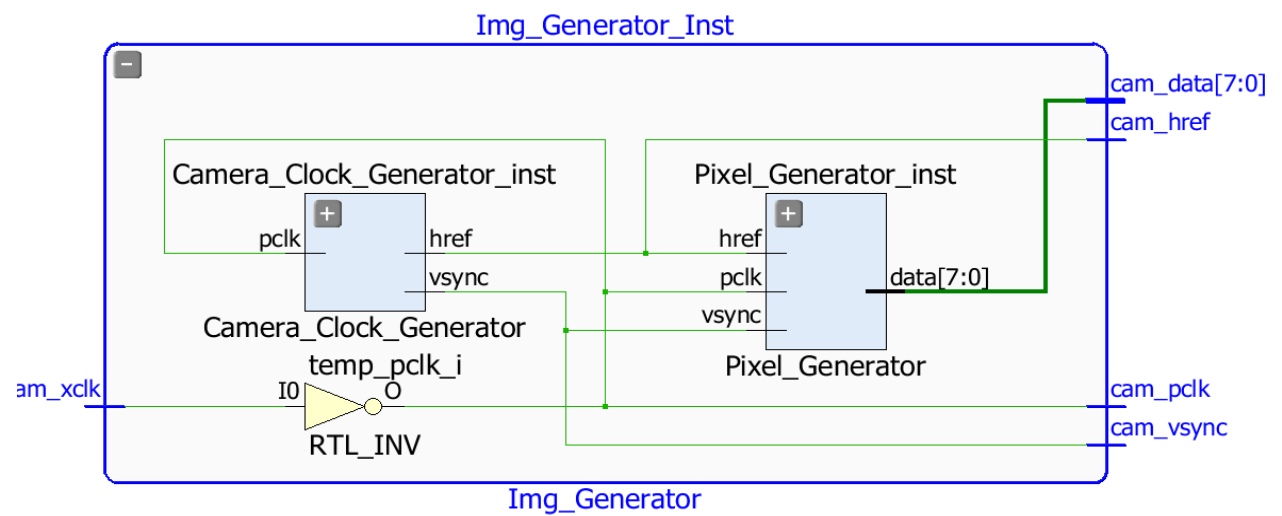


Figure 2: Elaborated design of Image Generator

Analysis:

Signal Name	Description
Cam_data[7:0]	YUV/RGB video component output data bits from image generator
cam_href	Horizontal camera sync signal
cam_pclk	Camera pixel clock output
cam_siod	SCCB (Serial Camera Control Bus) serial data I/O
cam_vsync	Vertical camera sync signal

## Part 2: Design of camera emulation

### 2.4 Task 1:

#### Red screen

#### Task-Description:

When investigating the component `Img_Generator` you will find two subcomponents, `Camera_Clock_Generator` and `Pixel_Generator`. Complete those VHDL models so that data is generated to display red pixels on your screen! The pixel output format is: 8Bit Y/8Bit Cb/8Bit Y/8Bit Cr.

### Elaboration:

The format in which the pixel color is formed is: 8Bit Y/8Bit Cb/8Bit Y/8Bit Cr. This uses a luminance component (Y) and two chrominance components (CB and CR). From these three components we can form a color pixel. We can also deduce the RGB values of the exact same pixel. The format 4:2:2 means that for each two pixels, they will have different luminance Y components, but they will share the same chrominance CB and CR components.

In order to obtain the color red, the values for each component, as well as the equivalent RGB values and the obtained color are shown in the next table:

<u>COLOR</u>	Y	Y_HEX	Y_DEC	CB	CB_HEX	CB_DEC	CR	CR_HEX	CR_DEC	COLOR RGB			DISPLAY COLOR
										RED	BLUE	GREEN	
RED	01010001	51	81	01011010	5A	90	11110000	F0	240	238	14	14	

The camera sends the components for each pixel in a specific order. It must be known in order to be able to construct the image on the screen. This order in which the camera will send the Y, CB and CR components is presented in the table below:

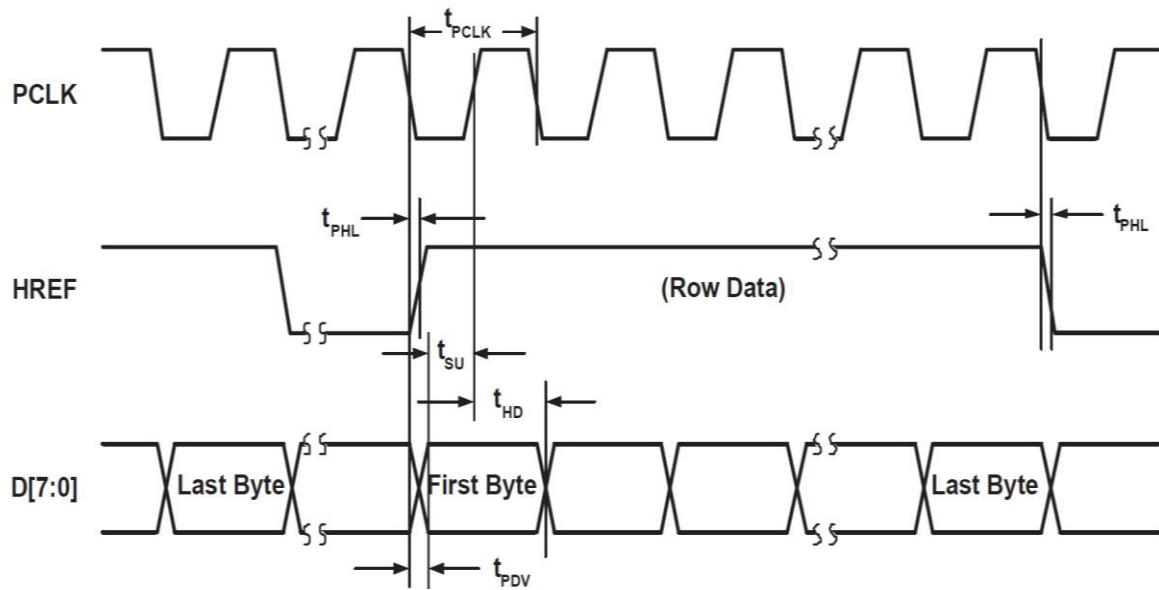
N	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>	...
Byte	CB0	Y0	CR0	Y1	CB2	Y2	CR2	Y3	...

So it needs two bytes for each pixel!

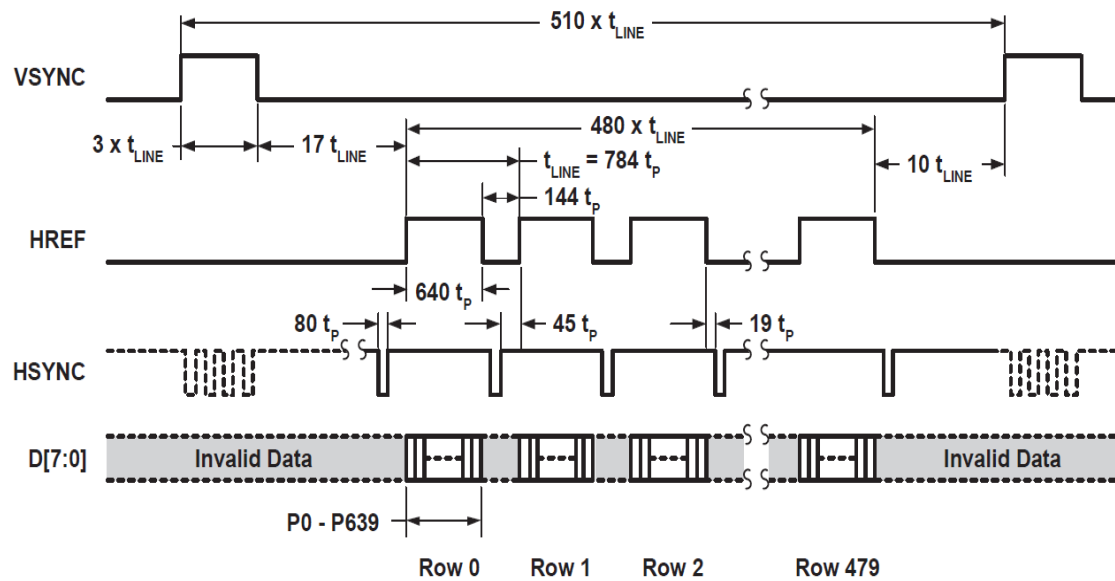
Where each individual pixel is format by:

PIXEL	Y Byte	CB Byte	CR Byte
Pixel 0	Y0	CB0	CR0
Pixel 1	Y1	CB0	CR0
Pixel 2	Y2	CB2	CR2
Pixel 3	Y3	CB2	CR2
Pixel 4	Y4	CB4	CR4

There are several control signals that dictate how the pixel components are sent. The most important are VSYNC, responsible for synchronizing an entire image frame on the screen, and HREF, responsible for synchronizing each line of the image frame. How the HREF signals is handled according to the clock signal is seen in Figure 3. How both the VSYNC signal and HREF signal are handled can be seen in Figure 4. We will need to replicate these signals in order to emulate the camera behavior.



**Figure 3. Horizontal timing:**



**NOTE:**

For Raw data,  $t_p = t_{PCLK}$

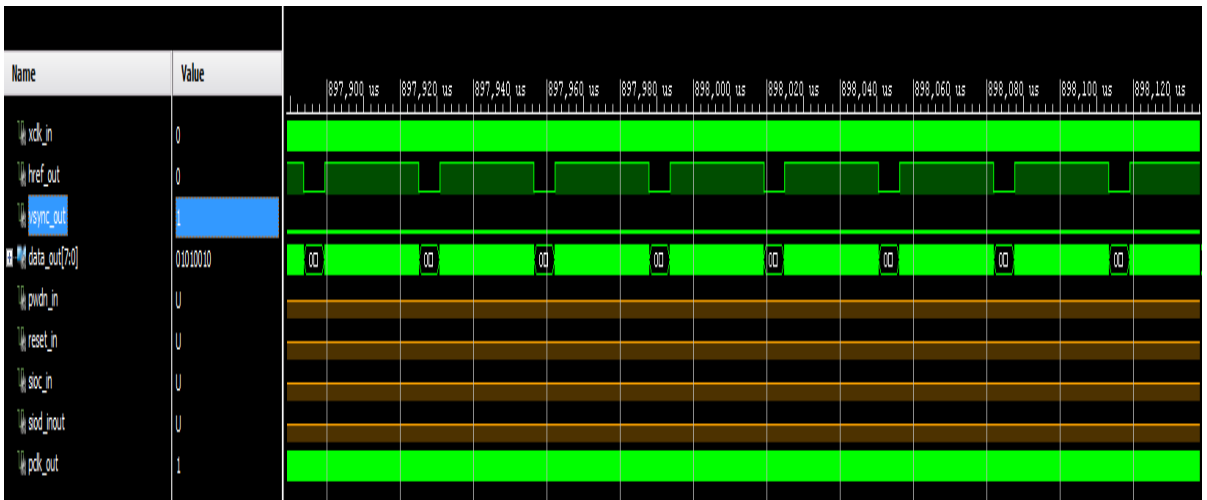
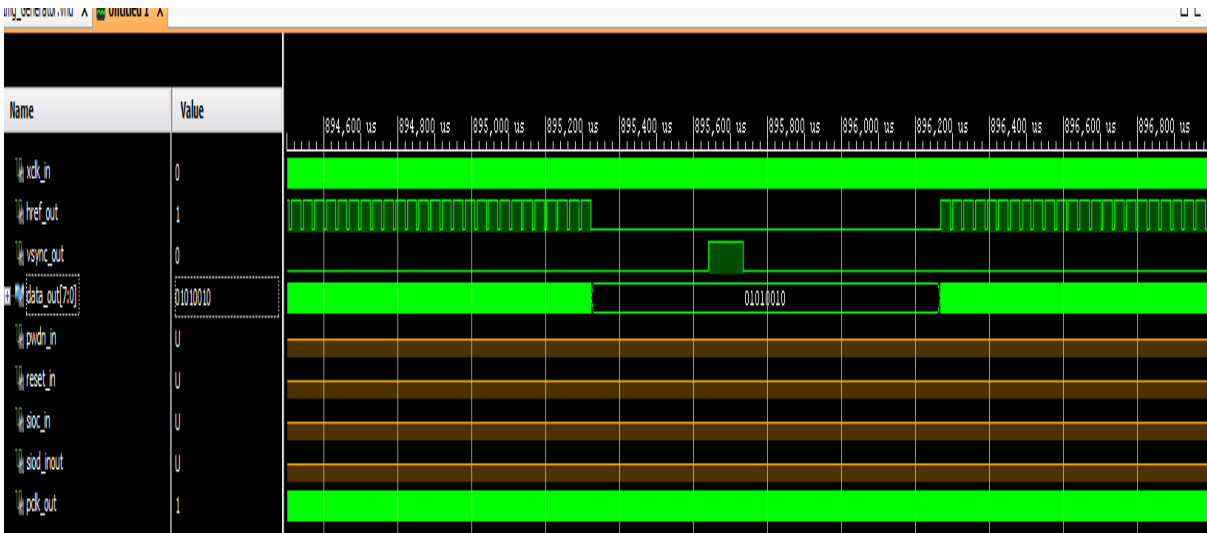
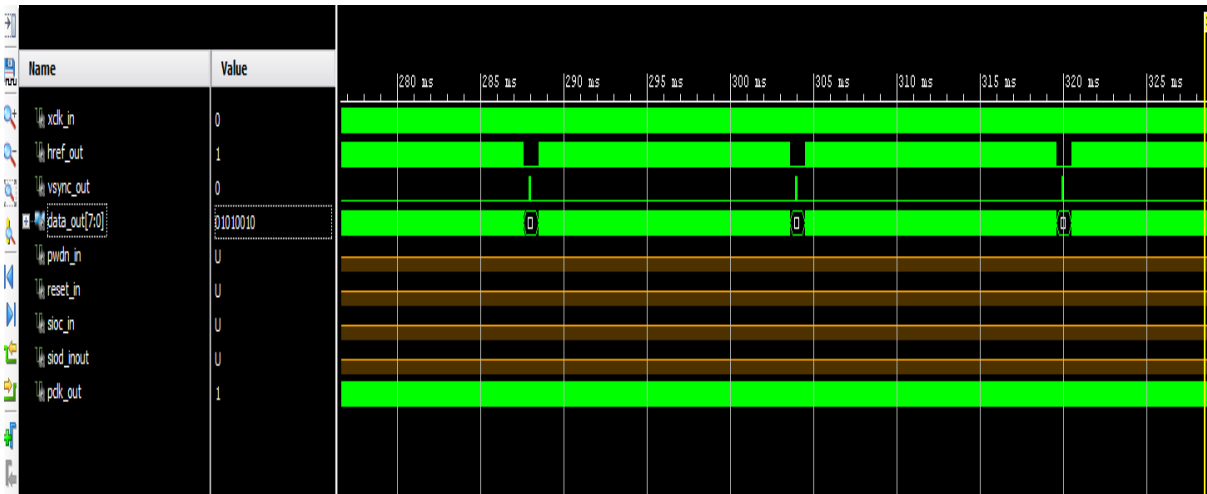
For YUV/RGB,  $t_p = 2 \times t_{PCLK}$

**Figure 4. VGA Frame Timing:**

For the purposes of designing the timing module Camera\_Clock\_Generator.vhd, a simulation was done to see that the obtained waveforms match those in the camera datasheet. The results can be



seen in Figure 5. The input is only the clock, while the outputs are the two control signals, VSYNC and HREF. Below can be seen the results done for a clock signal of 100MHz and at different ranges to highlight certain portions of the waveforms.



### Figure 5. Simulation results:

The code implement for the modules Camera\_Clock\_Generator.vhd and Pixel\_Generator.vhd, necessary for the generation of the color red on the display can be seen below. Certain important parts of the code are highlighted with additional comments to explain the basic functionality of these modules.

Camera\_Clock\_Generator.vhd is necessary for generating the control signals for displaying the image on the screen through the HDMI connection. It simply generates the horizontal and vertical synchronization signals. Pixel\_Generator.vhd will generate the actual pixel, meaning sending the Y, CB, CR components, in the same manner in which the actual camera would. It uses a simple counter to cycle through all these components. The final designs can be seen in the next text boxes, along with important comments.

#### Camera Clock Generator.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Camera_Clock_Generator is
    Port ( pclk : in STD_LOGIC;
          href : out STD_LOGIC;
          vsync : out STD_LOGIC
        );
end Camera_Clock_Generator;

architecture Behavioral of Camera_Clock_Generator is

    SIGNAL vsync_copy : STD_LOGIC;
    SIGNAL href_copy : STD_LOGIC;
begin

    PROCESS(pclk)

        VARIABLE line_time : integer range 0 to 2047 := 0; -- counts till tline= 784 * 2 pclk's
        VARIABLE t_line : integer range 0 to 1023:=0; -- counts till 510 tline's
        BEGIN

            IF pclk'event and pclk = '0' THEN

                IF (line_time = 1568) THEN
                    line_time := 0;
                
```

```

        t_line := t_line + 1;
        IF (t_line = 510) THEN
            t_line := 0;
        else
            -- retain old values
        END IF;
    else
        -- retain old values
    END IF;

    -- href generation according to the waveform
    IF (t_line >= 20) AND (t_line <= 500) THEN
        CASE line_time IS
            WHEN 0 => href_copy <= '1';
            WHEN 1280 => href_copy <= '0';
            WHEN OTHERS => NULL;
        END CASE;
    else
        href_copy <= '0';
    END IF;

    --vsync generation according to the waveform
    CASE t_line IS
        WHEN 0 => vsync_copy <= '1';
        WHEN 3 => vsync_copy <= '0';
        WHEN OTHERS => NULL;
    END CASE;

    line_time := line_time + 1;

    else
        -- retains old value

    END IF;

END PROCESS;

vsync <= vsync_copy;
href <= href_copy;

END behavioral;

```

### **Pixel Generator.vhd:**

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using

```

```

-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Pixel_Generator is
    Port ( pclk : in STD_LOGIC;
           href : in STD_LOGIC;
           vsync : in STD_LOGIC;
           data : out STD_LOGIC_VECTOR (7 downto 0));
end Pixel_Generator;

architecture Behavioral of Pixel_Generator is
begin
    -- your code
    pclk1: PROCESS(pclk)
    ---- As each pixel needs two bytes and each byte has Y and Cb/Cr components,
    -----a count of 4 is needed
        variable pdata_count : integer range 0 to 3:=0; --during synthesis initialization will be ignored
    BEGIN
        IF pclk'event and pclk = '0' THEN
            IF href = '1' and vsync = '0' THEN
                -- RED pixels
                CASE pdata_count IS
                    WHEN 0 => data <= "01011010"; -- Cb= 90
                    WHEN 1 => data <= "01010011"; -- y= 81
                    WHEN 2 => data <= "11110000"; -- Cr= 240
                    WHEN 3 => data <= "01010011"; -- y= 81
                    WHEN OTHERS => NULL;
                end case;
            end if;
        end if;
    end process pclk1;
end architecture Behavioral;

```

```

END CASE;

    pdata_count := pdata_count + 1;

    IF (pdata_count = 4) THEN

        pdata_count := 0;

    else

        --do nothing

    END IF;

    else

        data <= "01011010";

        count := 0;

        pdata_count := 0;

    END IF; -- href

    else

        -- do nothing

    END IF; --pclk

END PROCESS pclk1;

end Behavioral;

```

#### Analysis:

The code for Camera clock generator was written to generate Href and Vsync pulses required to display pixels on the screen.

The inputs were provided in the format 8Bit Y/8Bit Cb/8Bit Y/8Bit Cr, to display red pixels on your screen.

## 2.5 Task 2:

### Task-Description:

Implement this test design on the ZedBoard!

First check on file *video\_project\_constraints.xdc* if all the pins are set correctly. Provide a table for all input and output pins and their respective signals!

#### Elaboration:

Table for input and output and respective signals corresponding to those pins in the ZedBoard is shown below.

<b>Input Pin</b>	<b>Signals</b>	<b>Output Pin</b>	<b>Signals</b>
PIN T4	cam_test_data[0]	Data_out[0]	PIN Y13
PIN U4	cam_test_data[1]	Data_out[1]	PIN AA13
PIN R6	cam_test_data[2]	Data_out[2]	PIN AA14
PIN T6	cam_test_data[3]	Data_out[3]	PIN Y14
PIN Y4	cam_test_data[4]	Data_out[4]	PIN AB15
PIN AA4	cam_test_data[5]	Data_out[5]	PIN AB16
PIN AA7	cam_test_data[6]	Data_out[6]	PIN AA16
PIN AB6	cam_test_data[7]	Data_out[7]	PIN AB17
PIN AA11	Data_in[0]	Data_out[8]	PIN AA17
PIN AB10	Data_in[1]	Data_out[9]	PIN Y15
PIN Y10	Data_in[2]	Data_out[10]	PIN W13
PIN AB9	Data_in[3]	Data_out[11]	PIN W15
PIN AA9	Data_in[4]	Data_out[12]	PIN V15
PIN AA8	Data_in[5]	Data_out[13]	PIN U17
PIN W12	Data_in[6]	Data_out[14]	PIN V14
PIN V12	Data_in[7]	Data_out[15]	PIN V13
PIN V10	Cam_hRef	config_done	PIN T22
PIN W10	Cam_pclk	de	PIN U16
PIN T18	cam_resend	h_Sync	PIN V17
PIN V9	Cam_vSync	HDMI_CLK	PIN W18
PIN Y9	clk_100M	PwDn	PIN Y11
PIN F22	switch	reset	PIN AB11
PIN W8	siod	sioc	PIN V8
PIN Y16	siod_hdmi	sioc_hdmi	PIN AA18
		v_Sync	PIN W17
		xClk	PIN W11

#### Analysis:

All input and output pins in ZedBoard needed for the design implementation and their respective signals are given. From this the overall connection and flow of signals in the board are deduced.

#### 2.6 Task 3:





##### Color screen

##### Task Description:

Change Pixel\_Generator.vhd in such a way that the data is generated to display color strips on the screen

### Elaboration:

In order to obtain the colour strips on screen, the values for each component, as well as the equivalent RGB values and the obtained color are shown in the next table:

Colour	Y	Y_HEX	Y_DEC	CB	CB_HEX	CB_DEC	CR	CR_HEX	CR_DEC	Display
Red	01010011	51	81	01011010	5A	90	11110000	F0	240	
Green	10010000	90	144	00110101	35	53	00100010	22	34	
Blue	00100111	29	41	11110000	F0	240	01101101	6E	110	
Yellow	11010010	D2	210	00010000	10	16	10010010	92	146	

The code implement for the modules Camera\_Clock\_Generator.vhd is the same as above whereas the code Pixel\_Generator.vhd, has been modified for the generation of the color strips of red, green, blue and yellow on the display. This can be seen below.

### **Pixel\_Generator.vhd:**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Pixel_Generator is
  Port ( pclk : in STD_LOGIC;
        href : in STD_LOGIC;
        vsync : in STD_LOGIC;
        data : out STD_LOGIC_VECTOR (7 downto 0));
```

```

end Pixel_Generator;

architecture Behavioral of Pixel_Generator is
begin
-- your code

pclk1: PROCESS(pclk)

---- As each pixel needs two bytes and each byte has Y and Cb/Cr components,
-----a count of 4 is needed

    variable pdata_count : integer range 0 to 7:=0; --during synthesis initialization will be ignored
    variable count : integer range 0 to 3:=0;
    variable pixelpair_count : integer:=0;
BEGIN

    IF pclk'event and pclk = '0' THEN

        IF href = '1' and vsync = '0' THEN

            ----As href will be high for 1280 pclk at which horizontal output per line is processed
            ---- Pixels for each colour should be set during one fourth of the total time ie 1280pclk/2

            ----- RED pixels
            IF (pixelpair_count >= 0) AND (pixelpair_count < 320 )THEN
                CASE pdata_count IS
                WHEN 0 => data <= "01011010"; -- Cb= 90
                WHEN 1 => data <= "01010010"; -- y= 82
                WHEN 2 => data <= "11110000"; -- Cr= 240
                WHEN 3 => data <= "01010010"; -- y= 82
                WHEN OTHERS => NULL;
                END CASE;

            --Blue pixels
            ELSIF (pixelpair_count >= 320) AND (pixelpair_count < 640) THEN
                CASE pdata_count IS
                WHEN 0 => data <= "11110000"; -- Cb= 240
                WHEN 1 => data <= "00100111"; -- y= 39
                WHEN 2 => data <= "01101101"; -- Cr= 109
                WHEN 3 => data <= "00100111"; -- y= 39
                WHEN OTHERS => NULL;
                END CASE;

            ---Green pixels

            ELSIF (pixelpair_count >= 640) AND (pixelpair_count < 960) THEN
                CASE pdata_count IS
                WHEN 0 => data <= "00110101"; -- Cb= 53
                WHEN 1 => data <= "10010000"; -- y= 144
                WHEN 2 => data <= "00100010"; -- Cr= 34
                WHEN 3 => data <= "10010000"; -- y= 144
            
```



```

        WHEN OTHERS => NULL;
    END CASE;

    --yellow pixels

        ELSIF (pixelpair_count >= 960) AND (pixelpair_count < 1280) THEN
            CASE pdata_count IS
                WHEN 0 => data <= "00010000"; -- Cb= 16
                WHEN 1 => data <= "11010010"; -- y= 210
                WHEN 2 => data <= "10010010"; -- Cr= 146
                WHEN 3 => data <= "11010010"; -- y= 210
                WHEN OTHERS => NULL;
            END CASE;
        else
            END IF;

        pixelpair_count := pixelpair_count + 1;
        IF (pixelpair_count = 1280) THEN    -- 1280
            pixelpair_count := 0;
        else
            END IF;
        pdata_count := pdata_count + 1;
        IF (pdata_count = 4) THEN
            pdata_count := 0;
        else
            --do nothing
            END IF;
        else
            data <= "01011010";
            count := 0;
            pdata_count := 0;
            END IF; -- href

    else
        -- do nothing
        END IF; --pclk
    END PROCESS pclk1;

end Behavioral;

```

#### Analysis:

The code for Camera clock generator was written to generate Href and Vsync pulses required to display pixels on the screen.

The inputs were provided in the format 8Bit Y/8Bit Cb/8Bit Y/8Bit Cr, to display coloured strips of red, blue, green and yellow pixels on your screen.