Digital System Design
SS 2023

Lab 1: Design and Test of VHDL IP
(HDMI Display Controller + Camera Emulator)

Report submitted by :

GeethuSagar Ajitha
Matrikelnr : 1119258

Riya Alias
Matrikelnr : 1119262

Najmeddine Bouzambila

Matrikelnr: 738345

May/19/2023

Content

1 Introduction

In this lab we are testing a HDMI display controller, designed to display images captured by a camera. Data of the HDMI display controller is send to the on-board HDMI transmitter chip. The camera is getting emulated by another VHDL model.

2 Implementation

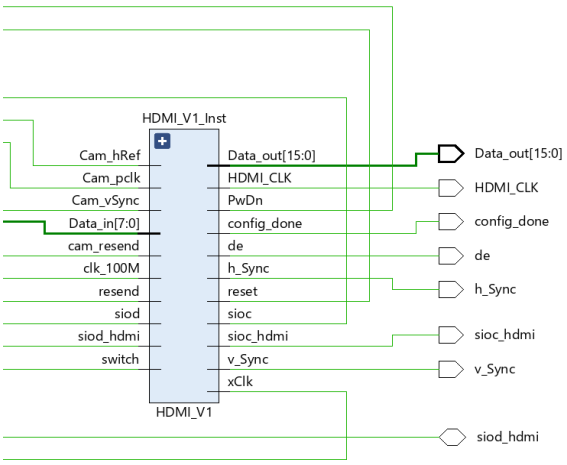
2.1 Topic1: Study top level RTL design

(a): Describe the meaning of all output signals of HDMI_V1!! Consider that those output signals are inputs to the HDMI transmitter ADV7511. Do research on ADV7511 of ZedBoard. Use a table format with two columns: signal name and description!

Task-Description: The component HDMI_V1 is the HDMI display controller, the component IMG_GENERATOR emulates the camera. And doing research on ADV7511

Analysis:

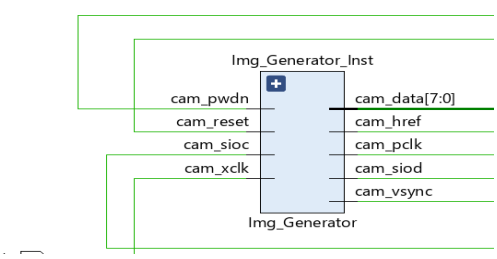
SIGNAL NAME	DESCRIPTION
Data_out[15:0]	16 bit pixel data output
HDMI_CLK	Clock for the HDMI module
config_done	Configuration completion signal
de	Data Enable signal input for Digital Video
h_Sync	Horizontal sync
sioc	SCCB serial interface clock output
v_Sync	Vertical sync
sioc_hdmi	SCCB serial interface data I/O
xClk	Transmit Clock to OV7670
PwDn	Power Down Mode Selection
reset	Clears all registers and resets them to their default values 0: Normal mode 1: Reset mode



(b): 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 compare with the output ports of IMG_GENERATOR! Use a table format with two columns: signal name and description!

Task-Description: Here instead of using a camera to test the HDMI controller we create input data normally provided by the camera. For emulation of the camera component Img_Generator.vhd is already designed, and we calculate the pixels to display (Pixel_Generator.vdh) as well as the camera clock (Camera_Clock_Generator.vhd).

Analysis:



SIGNAL NAME	DESCRIPTION
cam_data[7:0]	YUV/RGB video component output bits
cam_href	href output
cam_pclk	Pixel clock output
cam_siod	SCCB serial interface data I/O
cam_vsync	Vertical sync output
cam_xclk	System clock input
cam_sioc	SCCB serial interface clock input
cam_reset	Clears all registers and resets them to their default values. 0: Normal mode 1: Reset mode
cam_pwdn	Power Down Mode Selection 0: Normal mode 1: Power down mode

2.2 Topic2 : Design of camera emulator

(a): Here instead of using a camera to test the HDMI controller we are creating input data normally provided by the camera. For emulation of the camera a component *Img_Generator.vhd* is already designed, but to provide the pixels we want to display we have (*Pixel_Generator.vdh*) as well as the camera clock (*Camera_Clock_Generator.vhd*). These are present as subcomponents in Img_Generator.

Task-Description: In Camera_Clock_Generator.vhd we need to generate both, VSYNC and HREF inorder to emulate the camera. VSYNC is responsible for synchronizing an entire image frame on the screen, and HREF is responsible for synchronizing each line of the image frame.

(b): In *Pixel_Generator.vhd* we need to specify Y,CB and CR. Use VHDL *constant* with data type STD_LOGIC_VECTOR (7 downto 0)! Generate blue pixels!

Task-Description: Generating blue pixels using Y/Cb/Cr 4:2:2 format in VHDL

Analysis:

```
library IEEE;
use IEEE.std_logic_1164.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
        constant y_blue : STD_LOGIC_VECTOR (7 downto 0) := x"29";
        constant cb_blue : STD_LOGIC_VECTOR (7 downto 0) := x"F0";
        constant cr_blue : STD_LOGIC_VECTOR (7 downto 0) := x"6E";
        signal cnt : integer range 0 to 639 := 0;
        begin
        pixel_gen : process(pclk) begin
        if(falling_edge(pclk)) then
        if(href = '1') and (vsync = '0') then
        if((cnt rem 4) = 0) then
                data <= cb_blue;
```

```
        elsif ((cnt rem 4) = 1) then
                data <= y_blue;
        elsif ((cnt rem 4) = 2) then
                data <= cr_blue;
        elsif ((cnt rem 4) = 3) then
                data <= y_blue;
        end if
        if(cnt = 639) then
                cnt <= 0;
        else
                cnt<= cnt + 1;
        end if;
end if;
end if;
end process;
end Behavioral;
```

2.3 Topic3: Simulation of Camera Emulator

(a): Run simulation of Camera_Clock_Generator.vhd using ModelSim. No test bench required, because one input signal only. Compare your output with fig.2 and fig.3!

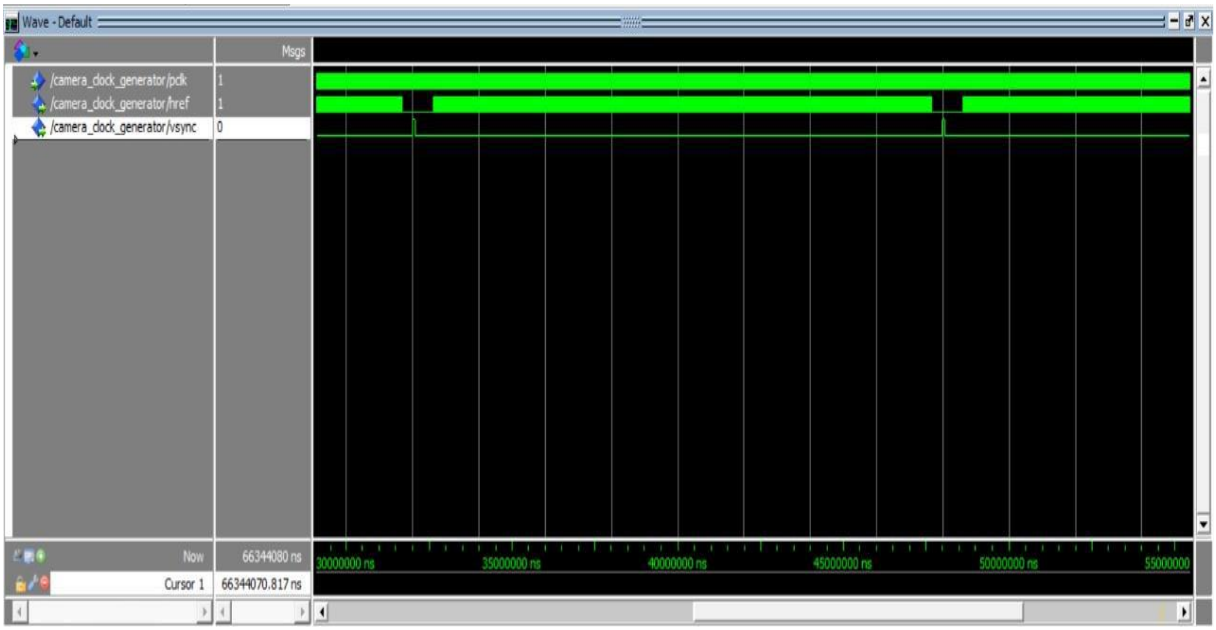
Task-Description: Here we are simulating the Camera_clock_Generator.vhd using ModelSim and is comparing our generated HREF and VSYNC signals.

Analysis:

Code Camera_Clock_Generator.vhd :

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.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 tick_timer: integer := 0;
begin
    generate_tick_timer: process(pclk)
    begin
    IF pclk = '1' then
        tick_timer <= (tick_timer + 1) mod 799680; --
    end IF;
end process;
generate_href: process(pclk)
variable count: integer := 0;
begin
    IF pclk = '1' then
        IF ((tick_timer >= 31360) AND (tick_timer < 784000)) then --
```

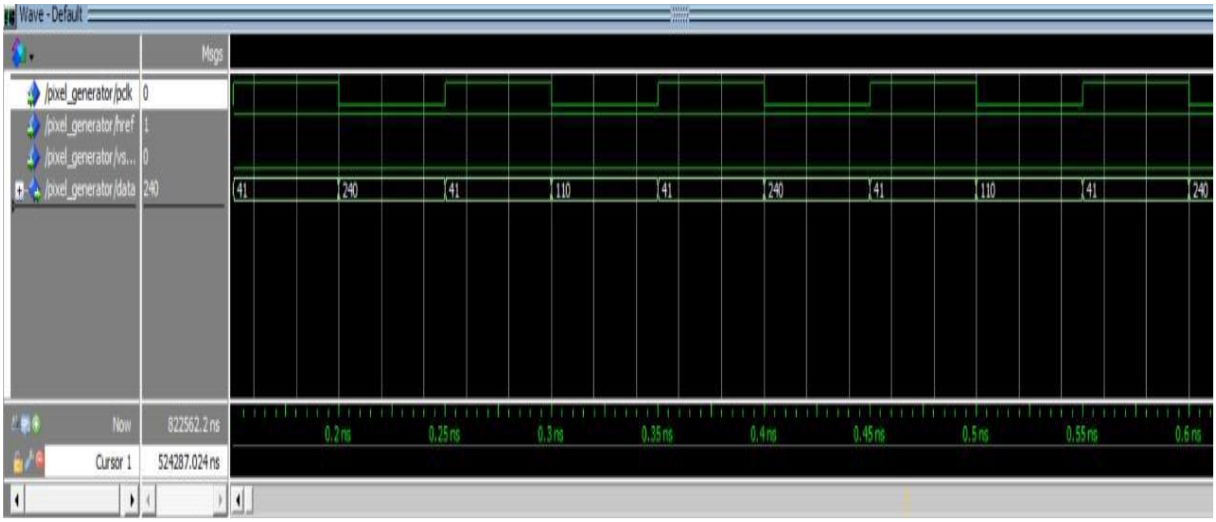
```
                IF ((count >= 0) AND (count < 1280)) then --
                        href <= '1';
                ELSE
                        href <= '0';
                end IF;
                count := (count + 1) mod 1568; --
        ELSE
        href <= '0';
        end IF;
end IF;
end process;
generate_vsync: process(pclk)
begin
IF pclk = '1' then
        IF ((tick_timer >= 0) AND (tick_timer < 4704)) then --
                vsync <= '1';
        ELSE
                vsync <= '0';
        end IF;
end IF;
end process;
end Behavioral;
```



(b): Run simulation of *Pixel_Generator.vhd* using ModelSim. No test bench required, because three input signals only. Check 8-bit output data on expected CB,Y,CR,Y,CB,Y,CR sequence for blue pixel!

Task-Description: Here we are simulating the *Pixel_Generator.vhd* using ModelSim and is comparing our generated HREF and VSYNC signals.

Analysis:



2.4 Topic4: Study top level RTL design

(a): First check on file *video_project_constraints.xdc* if all the pins are set correctly. Provide a table for all I/O pins and their respective signal names! Synthesize your VHDL design and implement it on the ZedBoard. Take photo of screen output!

Signal Name	**Port**
HDMI_CLK	W18

Data_out[15]	V14
Data_out[13]	U17
Data_out[12]	V15
Data_out[11]	W15
Data_out[10]	W13
Data_out[9]	Y15
Data_out[8]	AA17
Data_out[7]	AB17
Data_out[6]	AA16
Data_out[5]	AB16
Data_out[4]	AB15
Data_out[3]	Y14
Data_out[2]	AA14
Data_out[1]	AA12
Data_out[0]	Y13
cam_resend	T18
Clk_100M	Y9
Config_done	T22
De	U16
h_sync	V17
Resend	P16
Sloc_hdmi	AA18
Slod_hdmi	Y16
Switch	F22
V_sync	W17

(b): Change *Pixel_Generator.vhd* in a way that you get color strips on your screen as shown in fig.4! Take photo of screen output

```
library IEEE;
use IEEE.std_logic_1164.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
        constant y_red : STD_LOGIC_VECTOR (7 downto 0) := x"52";
        constant cb_red : STD_LOGIC_VECTOR (7 downto 0) := x"5A";
        constant cr_red : STD_LOGIC_VECTOR (7 downto 0) := x"F0";

        constant y_blue : STD_LOGIC_VECTOR (7 downto 0) := x"29";
        constant cb_blue : STD_LOGIC_VECTOR (7 downto 0) := x"F0";
        constant cr_blue : STD_LOGIC_VECTOR (7 downto 0) := x"6E";

        constant y_green : STD_LOGIC_VECTOR (7 downto 0) := x"91";
        constant cb_green : STD_LOGIC_VECTOR (7 downto 0) := x"36";
        constant cr_green : STD_LOGIC_VECTOR (7 downto 0) := x"22";

        constant y_yellow : STD_LOGIC_VECTOR (7 downto 0) := x"D2";
        constant cb_yellow : STD_LOGIC_VECTOR (7 downto 0) := x"F0";
        constant cr_yellow : STD_LOGIC_VECTOR (7 downto 0) := x"6E";
signal cnt : integer range 0 to 639 := 0;
```

```
begin
pixel_gen : process(pclk) begin
if(falling_edge(pclk)) then
if(href = '1') and (vsync = '0') then
        if((cnt rem 4) = 0) then
        if(cnt > 0 and cnt < 160) then
                data <= cb_red;
        elsif(cnt >= 160 and cnt < 320) then
                data <= cb_blue;
        elsif(cnt >= 320 and cnt < 480) then
                data <= cb_green;
        elsif(cnt >= 480 and cnt < 640) then
                data <= cb_yelllow;
        elsif ((cnt rem 4) = 1) then
        if(cnt > 0 and cnt < 160) then
                data <= y_red;
        elsif(cnt >= 160 and cnt < 320) then
                data <= y_blue;
        elsif(cnt >= 320 and cnt < 480) then
                data <= y_green;
        elsif(cnt >= 480 and cnt < 640) then
                data <= y_yelllow;
        elsif ((cnt rem 4) = 2) then
        if(cnt > 0 and cnt < 160) then
                data <= cr_red;
        elsif(cnt >= 160 and cnt < 320) then
                data <= cr_blue;
        elsif(cnt >= 320 and cnt < 480) then
```

```
            data <= cr_green;
        elsif(cnt >= 480 and cnt < 640) then
            data <= cr_yelllow;
        elsif ((cnt rem 4) = 3) then
        if(cnt > 0 and cnt < 160) then
            data <= y_red;
        elsif(cnt >= 160 and cnt < 320) then
            data <= y_blue;
        elsif(cnt >= 320 and cnt < 480) then
            data <= y_green;
        elsif(cnt >= 480 and cnt < 640) then
            data <= y_yelllow;
        end if;
        if(cnt = 639) then
            cnt <= 0;
        else
            cnt<= cnt + 1;
        end if;
end if;
end if;
end process;
end Behavioral;
```