Digital System Design SS2024

Lab 1: Design and Test of VHDL IP

Report submitted by:

Soumya Ranjan Sabat
Matrikelnr: 1127993

Azaz Hassan Khan
Matrikelnr: 1128032

May/17/2024

Content

1 Introduction

Test a HDMI display controller, designed to display images captured by a camera. Data of the HDMI display controller is finally sent to the on-board HDMI transmitter chip. The camera is getting emulated by another VHDL model.

Objectives:

- Study of the top-level RTL design
- VHDL design of the Camera Emulator
- Simulation of the Camera Emulator
- Test of HDMI Display Controller on ZedBoard

1.1 Part-1: Study top level RTL design

1.1.1 Task-1

Q1. 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!

Elaboration:

Signal Name	Description
Data_out [15:0]	• 16-bit Data output of frame buffer
HDMI_CLK	• A clock of 25MHz from vga_pll_zedboard
config_done	• CEC data signal. Supports CMOS logic levels from 1.8V to 5V • One of the outputs of ov7670_controller • A LED to show when config is finished
PwDn	• Power saver mode. One of the outputs of ov7670_controller (transfers registers to the camera over an I2C like bus) • A LED shows if it is at the power saver mode
de	• Data Enable signal input for Digital Video. Supports typical CMOS logic levels from 1.8V up to 3.3V • One of the outputs of vga_controller and acts as input to Video data capture of the HDMI transmitter ADV7511 • Display enables where '1' is the display time and '0' is the blanking time
h_sync	• Horizontal Sync Pulse input. Supports typical CMOS logic levels from 1.8V up to 3.3V • One of the outputs of vga_controller and acts as input to Video data capture of the HDMI transmitter ADV7511
v_sync	• Vertical Sync Pulse input. Supports typical CMOS logic levels from 1.8V up to 3.3V • One of the outputs of vga_controller and acts as input to Video

	data capture of the HDMI transmitter ADV7511
siod_hdmi	• Serial Port Data I/O. Supports CMOS logic levels from 1.8V to 3.3V • This pin serves as the serial port data I/O slave for register access
sioc_hdmi	• Serial Port Data Clock input. Supports CMOS logic levels from 1.8V to 3.3V • Can write to the registers
xClk	• Clk Driver for OV7670 camera • One of the outputs of ov7670_controller
Reset	• Always '1' for normal mode • One of the outputs of ov7670_controller

1.1.2 Task-2

Q1. 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!

Elaboration:

Signal Name	Description
cam_pwdn	• Power Down Mode Selection • 0: Normal mode 1: Power down mode
cam_reset	• Clears all registers and resets them to their default values. • 0: Normal mode 1: Reset mode
cam_sioc	• SCCB serial interface clock input
cam_xclk	• System clock input
cam_data[7:0]	• 8-bit output of Pixel_Generator • YUV video component output
cam_href	• HREF output • It is responsible for synchronizing each line of the image frame
cam_pclk	• Pixel clock output
cam_siod	• SCCB serial interface data I/O
cam_vsync	• Vertical sync output

	• It is responsible for synchronizing an entire image frame on the screen

1.2 Part-2: Design of camera emulator

1.2.1 Task-1

Q1. In *Camera_Clock_Generator.vhd* we need to generate both, VSYNC and HREF in order to emulate the camera.

```
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 temp_vsync : std_ulogic := '1';
    signal href_ena : std_ulogic := '0';
    signal cnt_vsync_pclk : integer range 0 to 799680 := 0;
    signal cnt_href_pclk : integer range 0 to 784 := 0;
begin
    HREF_VSYNC_gen: process(pclk)
    begin
        if falling_edge(pclk) then
            if (cnt_vsync_pclk = ((3*1568)-1)) and (temp_vsync = '1') then
                temp_vsync <= '0';
            elsif (cnt_vsync_pclk = ((510*1568)-1)) and (temp_vsync = '0') then
                temp_vsync <= '1';
            end if;

            if (cnt_vsync_pclk = ((20*1568)-1)) and (temp_vsync = '0') then
                href_ena <= '1';
            elsif (cnt_vsync_pclk = ((500*1568)-1)) and (temp_vsync = '0') then
                href_ena <= '0';
            end if;
```

```
            if (cnt_href_pclk = 0) and (href_ena = '1') then
                href <= '1';
            elsif (cnt_href_pclk = 640) and (href_ena = '1') then
                href <= '0';
            end if;

            vsync <= temp_vsync;
            cnt_vsync_pclk <= cnt_vsync_pclk + 1;
            cnt_href_pclk <= cnt_href_pclk + 1;

            if cnt_href_pclk = 783 then
                cnt_href_pclk <= 0;
            end if;

            if cnt_vsync_pclk = 799679 then
                cnt_vsync_pclk <= 0;
            end if;
        end if;
    end process;
end Behavioral;
```

1.2.2 Task-2

Q1. 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!

Elaboration:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.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
    -- Blue pixel: Y: 41 (29), Cb: 240 (F0), Cr: 110 (6E)
    constant Y_BLUE : STD_LOGIC_VECTOR (7 downto 0) := x"4C";
    constant CB_BLUE : STD_LOGIC_VECTOR (7 downto 0) := x"54";
    constant CR_BLUE : STD_LOGIC_VECTOR (7 downto 0) := x"FF";

    signal cnt_col_pixel : integer range 0 to 1279 := 0;  -- Column pixel counter
begin

    -- Pixel generation process
    pixel_gen_yCbCr : process(pclk)
```

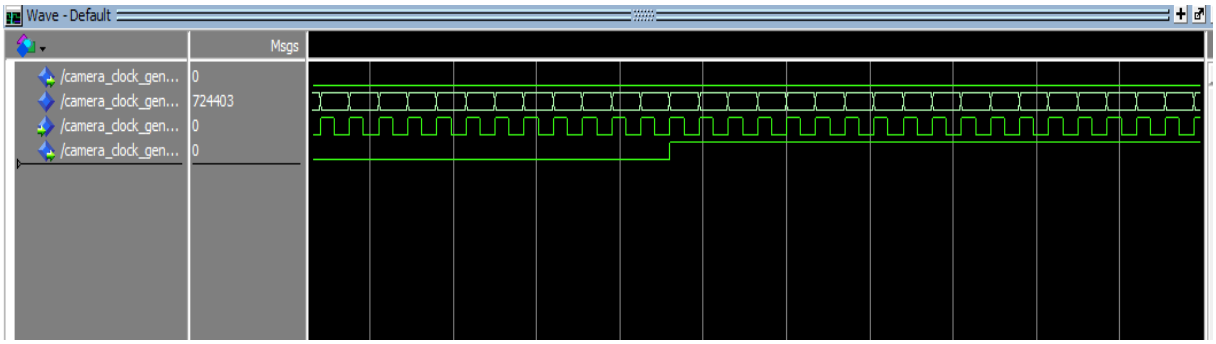
```
    begin
        if rising_edge(pclk) then
            if (href = '1') and (vsync = '0') then  -- Generate data only during active video line
                if (cnt_col_pixel mod 4 = 0) then
                    data <= CB_BLUE;  -- Cb component
                elsif (cnt_col_pixel mod 4 = 1) then
                    data <= Y_BLUE;   -- Y component
                elsif (cnt_col_pixel mod 4 = 2) then
                    data <= CR_BLUE;  -- Cr component
                else
                    data <= Y_BLUE;   -- Y component
                end if;
            end if;

            -- Reset the column pixel counter at the end of the line
            if (cnt_col_pixel = 1279) then
                cnt_col_pixel <= 0;
            else
                cnt_col_pixel <= cnt_col_pixel + 1;
            end if;
        end if;
    end process;

end Behavioral;
```

Colors	**Y-Cb-Cr Value**
Black	(16,128,128)
White	(235,128,128)
Red	(82,90,240)
Green	(145,54,34)
Blue	(41,240,110)
Yellow	(210,16,146)
Cyan	(170,166,16)
Magenta	(107,202,222)

1.3 Part-3: Simulation of camera emulator

1.3.1 Task-1

Q1. Run simulation of *Camera_Clock_Generator.vhd* using ModelSim (Copy code to a ModelSim project). No test bench required, because one input signal only. Compare your output with fig.2 and fig.3!



1.3.2 Task-2

Q1. Run simulation of *Pixel_Generator.vhd* using ModelSim (Copy code to a ModelSim project). 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!

1.4 Part-4: Test of HDMI Display Controller on ZedBoard

1.4.1 Task-1

Q1. 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!

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

1.4.2 Task-2

Q2. Synthesize your VHDL design and implement it on the ZedBoard. For this you **Run Synthesis**, then **Run Implementation**, finally click on **Generate Bitstream**.

```
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
    -- Define constants for YCbCr values
    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";

    -- Define a signal for counting pixels
    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 -- Every fourth pixel
                    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_yellow;
                    end if;
```

```
            elsif (cnt rem 4) = 1 then -- Every fourth pixel after the first
               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_yellow;
               end if;
            elsif (cnt rem 4) = 2 then -- Every fourth pixel after the second
               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_yellow;
               end if;
            elsif (cnt rem 4) = 3 then -- Every fourth pixel after the third
               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_yellow;
               end if;
               if cnt = 639 then -- Reset counter at the end of each line
                  cnt <= 0;
               else
                  cnt <= cnt + 1;
               end if;
            end if;
         end if;
      end if;
   end process;
end Behavioral;
```

1.4.3 Task-3

Q3. 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

2 References

[1] "VHDL Programming by Example" by Douglas L. Perry

[2] "VHDL: Analysis and Modeling of Digital Systems" by Zainalabedin Navabi

[3] "FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC" by Pong P. Chu

[4] "The Zynq Book" by Louise H. Crockett et al.

[5]