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

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

1.2.1 Task-1

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;

entity Pixel_Generator is Port (
pclk : in  STD_LOGIC; -- Pixel clock input
href : in STD_LOGIC; -- Horizontal synchronization input
vsync : in  STD_LOGIC; -- Vertical synchronization input
data : out STD_LOGIC_VECTOR (7 downto 0)  -- Output pixel data
);
end Pixel_Generator;

architecture Behavioral of Pixel_Generator is
    -- Constants representing blue pixel values in YCbCr format
    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";

    -- Counter for horizontal pixel position
    signal count : integer range 0 to 639 := 0;

begin
    -- Process for generating blue pixels on falling edge of pixel clock
    blue_pixel_generation : process(pclk)
    begin
        if falling_edge(pclk) then
        -- Switch case to determine which component of YCbCr to output
            case count mod 4 is
                when 0 =>
                    data <= cb_blue; -- Output Cb component
                when 1 | 3 =>
                    data <= y_blue; -- Output Y component
                when others =>
                    data <= cr_blue; -- Output Cr component
            end case;

            -- Increment horizontal pixel position counter
            if count = 639 then
                count <= 0; -- Reset counter at end of line
            else
                count <= count + 1; -- Increment counter
```

```
          end if;
        end if;
      end process blue_pixel_generation;
    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)

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]