An-Najah National University
Faculty of Engineering and IT

جامعة النجاح الوطنية
كلية الهندسة وتكنولوجيا المعلومات

Computer Engineering Department	
Course Name: Digital design Lab 2	**Number: 10636391**
Lab Report Grading Sheet	

Instructor: Dr Ashraf Armoush	Experiment #: 8
Academic Year: 2020/2021	Experiment Name: VGA controller and memory IP core
Semester: Summer	

Students				
1-Ashraf Habromman		2-Ra'ed Khwayerh		
Performed on: 15/7/2021		Submitted on: 29/7/2021		
Report's Outcomes				
ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %

Evaluation Criterion	Grade	Points
Abstract answers of the questions: "What did you do? How did you do it? What did you find?"	0.5	
Introduction and Theory Sufficient, clear and complete statement of objectives. In addition to Presents sufficiently the theoretical basis.	1.5	
Apparatus/ Procedure Apparatus sufficiently described to enable another experimenter to identify the equipment needed to conduct the experiment.Procedure sufficiently described.	2	
(Experimental Results and Discussion (In-Lab Worksheet Crisp explanation of experimental results. Comparison of theoretical predictions to experimental results, including discussion of accuracy and error analysis in some cases.	4	
Conclusions and Recommendations Conclusions summarize the major findings from the experimental results with adequate specificity. Recommendations appropriate in light of conclusions. Correct grammar.	1	
Appearance Title page is complete, page numbers applied, content is well organized, correct spelling, fonts are consistent, good visual appeal.	1	
Total	10	

Issue number: AD3-3

VGA controller and memory IP core

Introduction:

A standard VGA monitor consists of a grid of pixels that can be divided into rows and columns. A VGA monitor contains at least 480 rows, with 640 pixels per row.

Each pixel can display various colors, depending on the state of the red, green, and blue signals. Each VGA monitor has an internal clock that determines when each pixel is updated. This clock operates at the VGA-specified frequency of 25 MHz. The monitor refreshes the screen in a prescribed manner that is partially controlled by the horizontal and vertical synchronization signals. The monitor starts each refresh cycle by updating the pixel in the top left-hand corner of the screen, which can be treated as the origin of an X–Y plane. After the first pixel is refreshed, the monitor refreshes the remaining pixels in the row. When the monitor receives a pulse on the horizontal synchronization, it refreshes the next row of pixels. This process is repeated until the monitor reaches the bottom of the screen. When the monitor reaches the bottom of the screen, the vertical synchronization pulses, causing the monitor to begin refreshing pixels at the top of the screen.

VGA Timing For the VGA monitor to work properly, it must receive data at specific times with specific pulses. Horizontal and vertical synchronization pulses must occur at specified times to synchronize the monitor while it is receiving color data. Based on the clock frequency, these times translate to a certain number of clock cycles.

In a previous experiment we learned how to use an IP core that let us use a black box that implements some software operation so in this experiment we want to use a memory block to store image information to display it on the monitor.

Abstract:

In this experiment you will develop a controller that would operate a VGA to display some pictures to PC monitor.

Apparatus:

1-Vivado Design Suite HL WebPACK™ Edition.
2-ZedBoard.
3-Pc Monitor.
4-VGA cable

Procedure:

Part 1 Color display:

First horizontal display:

VGA Driver:

```
---------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity VGA_Driver is
   Port ( clk : in STD_LOGIC;
          red : out STD_LOGIC_VECTOR (3 downto 0);
          green : out STD_LOGIC_VECTOR (3 downto 0);
          blue : out STD_LOGIC_VECTOR (3 downto 0);
          HS : out STD_LOGIC;
          VS : out STD_LOGIC);
end VGA_Driver;

architecture Behavioral of VGA_Driver is
signal counter_v,counter_h : integer :=0;
begin

process(clk)
begin
   if(clk'event and clk = '1')then
      counter_h <= counter_h + 1;
      if(counter_h > 0 and counter_h <= 640)then
         HS <= '1';
         if(counter_v > 0 and counter_v <= 160)then
            red <= "1111";
            green<= "0000";
            blue <= "0000";
         elsif(counter_v > 160 and counter_v <= 320)then
            red <= "0000";
            green<= "1111";
            blue <= "0000";
         elsif(counter_v > 320 and counter_v <= 480)then
            red <= "0000";
            green<= "0000";
            blue <= "1111";
            end if;

        elsif(counter_h > 640 and counter_h <= 656)then
            red <= "0000";
            green<= "0000";
            blue <= "0000";
        elsif(counter_h > 656 and counter_h <= 752)then
            HS <= '0';
        elsif(counter_h > 752 and counter_h <= 800)then
            HS <= '1';
        elsif(counter_h > 800)then
            counter_h <= 0;
            counter_v <= counter_v + 1;
        end if;

        if(counter_v > 0 and counter_v <= 480)then
            VS <= '1';
        elsif(counter_v > 480 and counter_v <= 490)then
            red <= "0000";
            green<= "0000";
            blue <= "0000";
        elsif(counter_v > 490 and counter_v <= 492)then
            VS <= '0';
        elsif(counter_v > 492 and counter_v <= 525)then
            VS <= '1';
        elsif(counter_v > 525)then
            counter_v <= 0;
        end if;
    end if;

end process;

end Behavioral;
```

Clock divider:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clock_divider is
    Port ( CLK_IN : in STD_LOGIC;
           CLK_OUT : out STD_LOGIC);
end clock_divider;
```

```
architecture Behavioral of clock_divider is
signal count: integer := 1;
signal tmp : std_logic :='0';
begin
process(CLK_IN)
begin
   if(CLK_IN'event and CLK_IN = '1') then
      count <= count+1;
     if (count = 2) then
      tmp <= not(tmp);
      count <= 1;
     end if;
   end if;
     CLK_OUT <= tmp;
end process;
end Behavioral;
```

Top-level of the driver:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity VGA_Top_level is
   Port ( clk : in STD_LOGIC;
          red : out STD_LOGIC_VECTOR (3 downto 0);
          green : out STD_LOGIC_VECTOR (3 downto 0);
          blue : out STD_LOGIC_VECTOR (3 downto 0);
          HS : out STD_LOGIC;
          VS : out STD_LOGIC);
end VGA_Top_level;

architecture Behavioral of VGA_Top_level is

component clock_divider Port ( CLK_IN : in STD_LOGIC;
          CLK_OUT : out STD_LOGIC);
end component;

component  VGA_Driver_Vertical
   Port ( clk : in STD_LOGIC;
          red : out STD_LOGIC_VECTOR (3 downto 0);
          green : out STD_LOGIC_VECTOR (3 downto 0);
```

```
            blue : out STD_LOGIC_VECTOR (3 downto 0);
            HS : out STD_LOGIC;
            VS : out STD_LOGIC);
end component;
signal clk_out_s : std_logic := '0';
begin
clock_divider_1 : clock_divider port map(clk_in=>clk, clk_out=>clk_out_s );

VGA_Driver_1 : VGA_Driver_Vertical port map (clk=>clk_out_s, red=>red, green=>green,
blue=>blue, HS=>HS, VS=>VS );

end Behavioral;
```

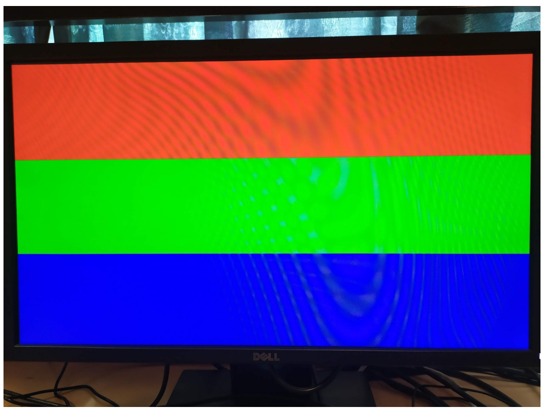
Second vertical display:

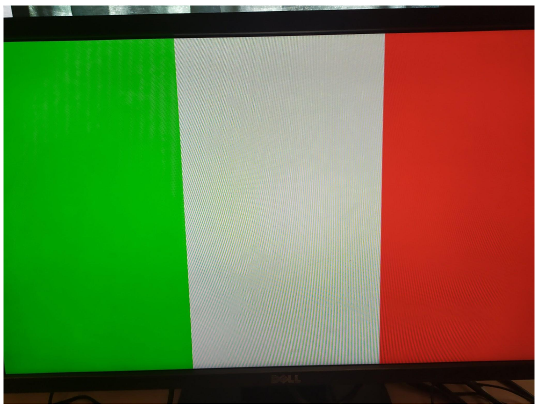
Just use the same top-level but with the following driver which colled “VGA_Driver_Vertical”:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity VGA_Driver_Vertical is
    Port ( clk : in STD_LOGIC;
            red : out STD_LOGIC_VECTOR (3 downto 0);
            green : out STD_LOGIC_VECTOR (3 downto 0);
            blue : out STD_LOGIC_VECTOR (3 downto 0);
            HS : out STD_LOGIC;
            VS : out STD_LOGIC);
end VGA_Driver_Vertical;

architecture Behavioral of VGA_Driver_Vertical is
signal counter_v,counter_h : integer :=0;
begin

process(clk)
begin
    if(clk'event and clk = '1')then
        counter_h <= counter_h + 1;
        if(counter_h > 0 and counter_h <= 213)then
            HS <= '1';
            red <= "0000";
            green<= "1111";
            blue <= "0000";
        elsif(counter_h > 213 and counter_h <= 427)then
```

```
            HS <= '1';
            red <= "1111";
            green<= "1111";
            blue <= "1111";
        elsif(counter_h > 427 and counter_h <= 640)then
            HS <= '1';
            red <= "1111";
            green<= "0000";
            blue <= "0000";

        elsif(counter_h > 640 and counter_h <= 656)then
            red <= "0000";
            green<= "0000";
            blue <= "0000";
        elsif(counter_h > 656 and counter_h <= 752)then
            HS <= '0';
        elsif(counter_h > 752 and counter_h <= 800)then
            HS <= '1';
        elsif(counter_h > 800)then
            counter_h <= 0;
            counter_v <= counter_v + 1;
        end if;

        if(counter_v > 0 and counter_v <= 480)then
            VS <= '1';
        elsif(counter_v > 480 and counter_v <= 490)then
            red <= "0000";
            green<= "0000";
            blue <= "0000";
        elsif(counter_v > 490 and counter_v <= 492)then
            VS <= '0';
        elsif(counter_v > 492 and counter_v <= 525)then
            VS <= '1';
        elsif(counter_v > 525)then
            counter_v <= 0;
        end if;
    end if;

end process;

end Behavioral;
```

Results:





Part 2 Image Display:

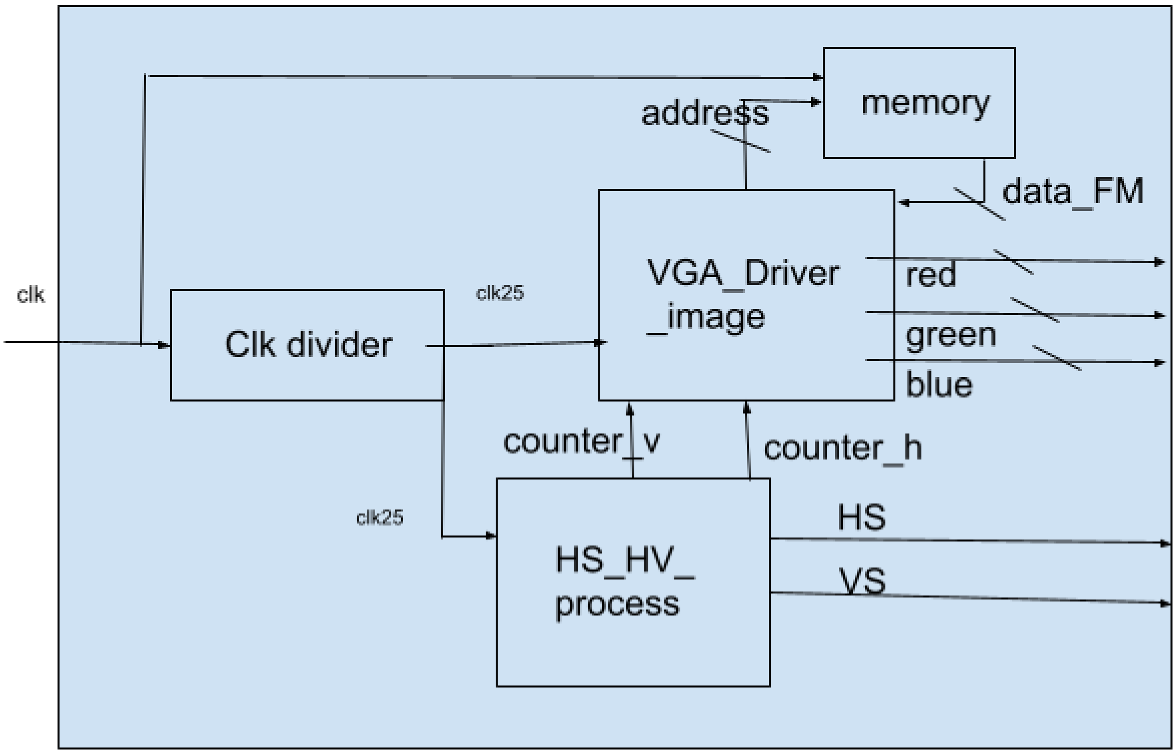


Fig1: top level of the image display

Memory:

Use IP core to build memory. And use this to make an instance from it:

```
COMPONENT image_memory
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(18 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
  );
END COMPONENT;
```

Instance_name: image_memory PORT MAP (clka => clk, addra => address_s, douta => data_FM_s);

VGA_Driver_image :

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;



entity VGA_Driver_image is
    Port ( clk25 : in STD_LOGIC;
           counter_h : in integer;
           counter_v : in integer;
           data_FM : in STD_LOGIC_VECTOR (11 downto 0);
              address : out STD_LOGIC_VECTOR (18 downto 0);
           Red : out STD_LOGIC_VECTOR (3 downto 0);
           Green : out STD_LOGIC_VECTOR (3 downto 0);
           Blue : out STD_LOGIC_VECTOR (3 downto 0));
end VGA_Driver_image;

architecture Behavioral of VGA_Driver_image is
signal address_s : std_logic_vector(18 downto 0);
begin

process(clk25)
begin
    if(clk25'event and clk25 = '1')then
           if(address_s >= x"4B000")then
```

```
                    address_s <= (others =>'0');
            end if;

         if(counter_h > 0 and counter_h <= 640)then
                  if(counter_v > 0 and counter_v <= 480)then
                           address_s <= address_s + 1;
                           Blue <= data_FM(11 downto 8);
                           Green <= data_FM(7 downto 4);
                           Red <= data_FM(3 downto 0);
                  end if;
         else
                  Red <= "0000";
                  Green <= "0000";
                  Blue <= "0000";
         end if;
   end if;

address <= address_s;
end process;

end Behavioral;
```

HS_VS_process:

```
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity HS_VS_process is
   Port ( clk25 : in STD_LOGIC;-- the input from clock divider
         HS : out STD_LOGIC;
         VS : out STD_LOGIC;
            counter_v : out INTEGER;
            counter_h : out INTEGER);
end HS_VS_process ;

architecture Behavioral of HS_VS_process is
signal counter_v_s,counter_h_s : integer :=0;
begin

process(clk25)
begin
```

```
    if(clk25'event and clk25 = '1')then
        counter_h_s <= counter_h_s + 1;
        if(counter_h_s > 0 and counter_h_s <= 656)then
            HS <= '1';
        elsif(counter_h_s > 656 and counter_h_s <= 752)then
            HS <= '0';
        elsif(counter_h_s > 752 and counter_h_s <= 800)then
            HS <= '1';
        elsif(counter_h_s > 800)then
            counter_h_s <= 0;
            counter_v_s <= counter_v_s + 1;
        end if;

        if(counter_v_s > 0 and counter_v_s <= 490)then
            VS <= '1';
        elsif(counter_v_s > 490 and counter_v_s <= 492)then
            VS <= '0';
        elsif(counter_v_s > 492 and counter_v_s <= 525)then
            VS <= '1';
        elsif(counter_v_s > 525)then
            counter_v_s <= 0;
        end if;
    end if;

counter_v <= counter_v_s;
counter_h <= counter_h_s;

end process;

end Behavioral;
```

Clock divider:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity clock_divider is
    Port ( CLK_IN : in STD_LOGIC;
           CLK_OUT : out STD_LOGIC);
end clock_divider;
```

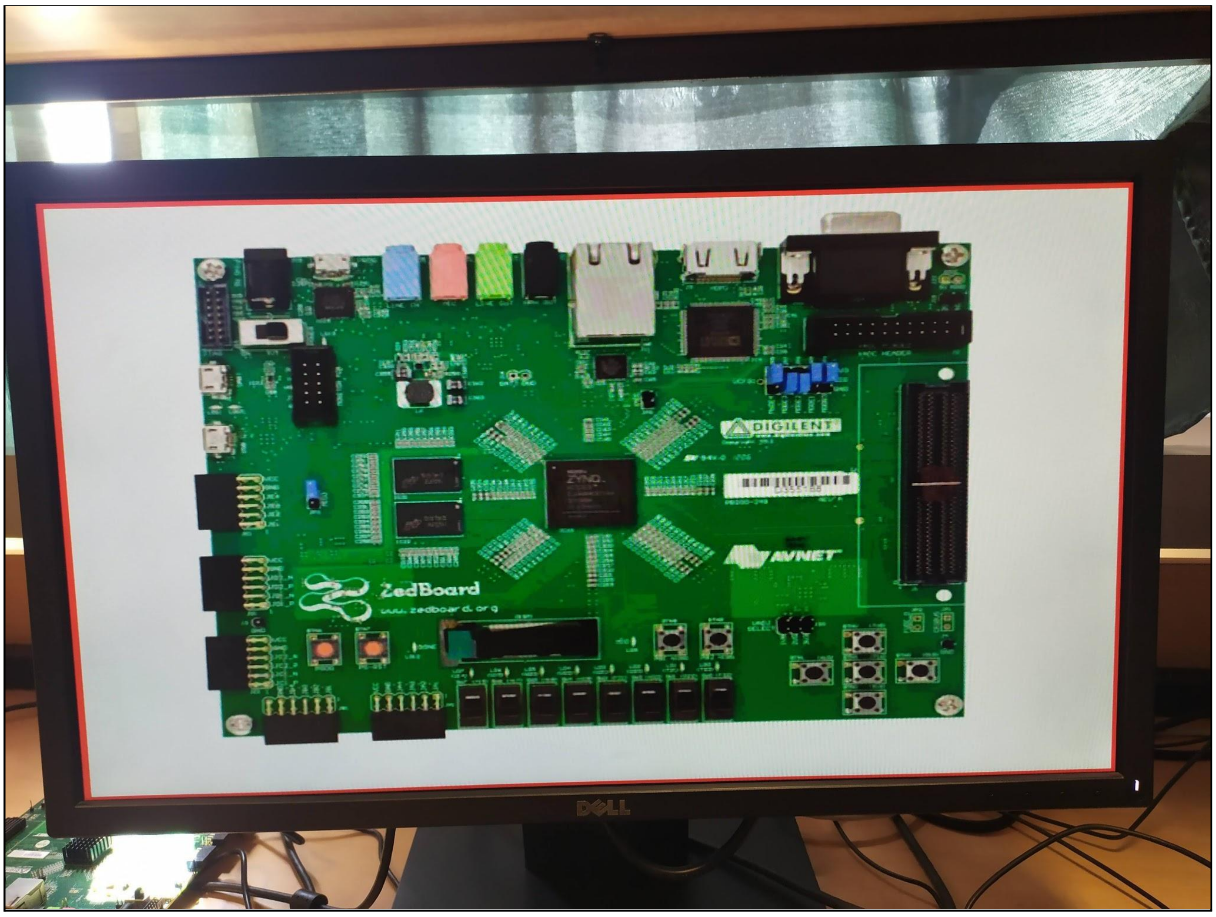
```
architecture Behavioral of clock_divider is
signal count: integer := 1;
signal tmp : std_logic :='0';
begin
process(CLK_IN)
begin
   if(CLK_IN'event and CLK_IN = '1') then
      count <= count+1;
     if (count = 2) then
      tmp <= not(tmp);
      count <= 1;
     end if;
   end if;
     CLK_OUT <= tmp;
end process;
end Behavioral;
```

Image_driver_top_level :

```
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity Image_driver_top_level is
   Port ( clk : in STD_LOGIC;
          red : out STD_LOGIC_VECTOR (3 downto 0);
          green : out STD_LOGIC_VECTOR (3 downto 0);
          blue : out STD_LOGIC_VECTOR (3 downto 0);
          HS : out STD_LOGIC;
          VS : out STD_LOGIC);
end Image_driver_top_level;

architecture Behavioral of Image_driver_top_level is

component clock_divider
   Port ( CLK_IN : in STD_LOGIC;
          CLK_OUT : out STD_LOGIC);
end component clock_divider;
```

```
component HS_VS_process
   Port ( clk25 : in STD_LOGIC; -- the input from clock divider
        HS : out STD_LOGIC;
        VS : out STD_LOGIC;
           counter_v : out INTEGER;
           counter_h : out INTEGER);
end  component HS_VS_process ;

component VGA_Driver_image
   Port ( clk25 : in STD_LOGIC;
        counter_h : in integer;
        counter_v : in integer;
        data_FM : in STD_LOGIC_VECTOR (11 downto 0);
              address : out STD_LOGIC_VECTOR (18 downto 0);
        Red : out STD_LOGIC_VECTOR (3 downto 0);
        Green : out STD_LOGIC_VECTOR (3 downto 0);
        Blue : out STD_LOGIC_VECTOR (3 downto 0));
end component VGA_Driver_image;

COMPONENT image_memory
  PORT (
   clka : IN STD_LOGIC;
   addra : IN STD_LOGIC_VECTOR(18 DOWNTO 0);
   douta : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
  );
END COMPONENT;

signal clk25_s : std_logic;
signal counter_h_s, counter_v_s : integer;
signal address_s : std_logic_vector(18 downto 0);
signal data_FM_s : std_logic_vector(11 downto 0);

begin
clk_div : clock_divider port map (CLK_IN => clk, CLK_OUT => clk25_s);
hs_vs_process_1 : HS_VS_process port map (clk25 => clk25_s, HS=>HS, VS=>VS,
counter_v => counter_v_s, counter_h => counter_h_s );
vga_driver_1 : VGA_Driver_image port map (clk25 => clk25_s, counter_h => counter_h_s,
counter_v => counter_v_s, data_FM => data_FM_s,
 address => address_s, Red => red, Green => green, Blue => blue);
memory : image_memory PORT MAP (clka => clk, addra => address_s, douta =>
data_FM_s);

end Behavioral;
```

Results:



Conclusion:

We learned in this experiment how to display some pictures to monitor, in the first part we display 3 colors vertically and horizontally, in part 2 we do things more advanced that display image that has details. In this image each pixel has a different color.