RUTGERS
UNIVERSITY

Course Name: Embedded Systems Design

Course Number: 16:332:579

Assignment:	Lab 4– Now You See It, Now You Don't
Course Instructor:	Prof. Phillip Southard
Date Submitted:	4/11/2019
Submitted by:	Sanjana Devaraj (NetID: sd1049)

Contents

1 Purpose

The main purpose of this lab was to introduce the concept of utilizing the VGA analog video standard in order to produce a static image on a computer display screen and of using a combination of counters for driving the timing signals and ROM addressing.

CRT-based VGA displays use amplitude-modulated moving electron beams (or cathode rays) to display information on a phosphor-coated screen. Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube as shown in Figure 1.

Information is only displayed when the beam is moving in the "forward" direction (left to right and top to bottom), and not during the time the beam is reset back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking" periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass. The size of the beams, the frequency at which the beam can be traced across the display, and the frequency at which the electron beam can be modulated determine the display resolution.

LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis. Although the above mentioned description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays.

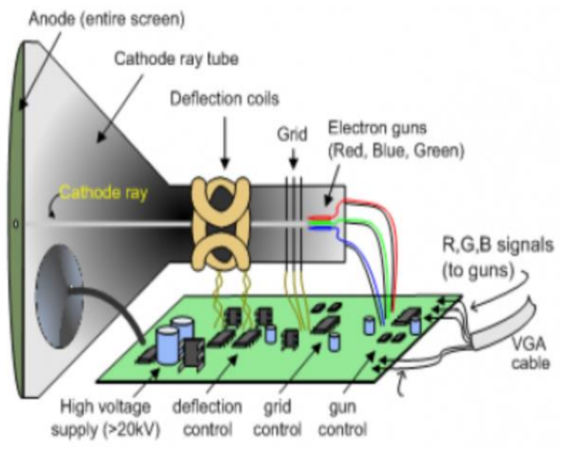


Figure 1: Color CRT display

The lab is comprised of two tasks. The first task is to design a VGA controller to generate the timing signals along with testing the VGA controller module with a testbench to check the behavior of the counters and the horizontal sync 'hs' and display 'vid' signals. The second task consisted of five sub-sections, creation of block memory with the COE file generated using the given MATLAB script, design of a pixel pusher module enables the picture be displayed on the screen by reading the pixel information stored in block memory, draw a block diagram for the image top level diagram, create a image top level design that implements the drawn block diagram and modify the XDC file to make the clock and VGA connector pins enabled.

Prelab Task:

The first prelab task is to read section 11 on VGA in the Digilent reference manual for the Zybo board. The ZYBO board uses 18 programmable logic pins to create an analog VGA output port. This translates to 16-bit color depth and two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync) as shown in Figure 2. With 5 bits each for red and blue and 6 bits for green, 65,536 (32×32×64) different colors can be displayed, one for each unique 16-bit pattern.

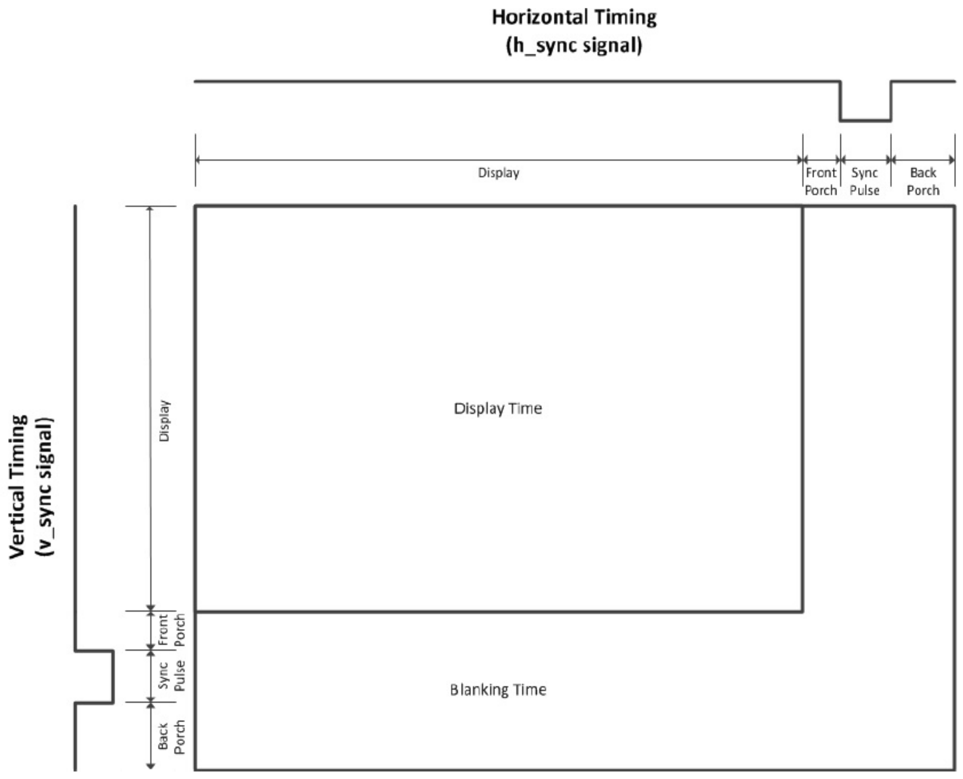


Figure 2: Off Screen Space and Sync Pulse Visualization

A video controller circuit must be created in programmable logic to drive the sync and color signals with the correct timing in order to produce a working display system. Modern VGA displays can accommodate different resolutions, and a VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. The VGA system timing information on how a VGA monitor might be driven in 640 by 480 mode, the resolution used in this lab is shown in Table 1 below.

Horizontal Timing (line)			Vertical Timing (frame)		
Scanline Part	**Pixels**	**Time (μs)**	**Frame Part**	**Lines**	**Time (μs)**
Visible Area	640	25.422045680238	Visible Area	480	15.253227408143
Front Porch	16	0.63555114200596	Front Porch	10	0.31777557100298
Sync Pulse	96	3.8133068520357	Sync Pulse	2	0.063555114200596
Back Porch	48	1.9066534260179	Back Porch	33	1.0486593843098
Whole Line	800	31.777557100298	Whole Frame	525	16.683217477656

Table 1: VGA system timing information

The second prelab task is to run the given MATLAB script which takes an input image and down-samples it to an 8-bit color space having 480x480 output resolution due to memory limitations on the Zybo board and also creating a Xilinx COE file that is used to create a ROM to hold the image data. The dimensions of my chosen image shown in Figure 3 is 450x450 in JPG format. The MATLAB version of the image is shown in Figure 4 which is padded with black below and to the right of the image as the image dimensions is smaller compared to 480x480.

Figure 3: My Image

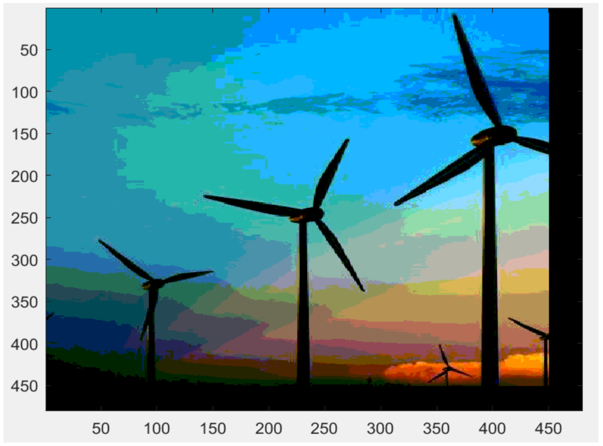


Figure 4: MATLAB version of the Image

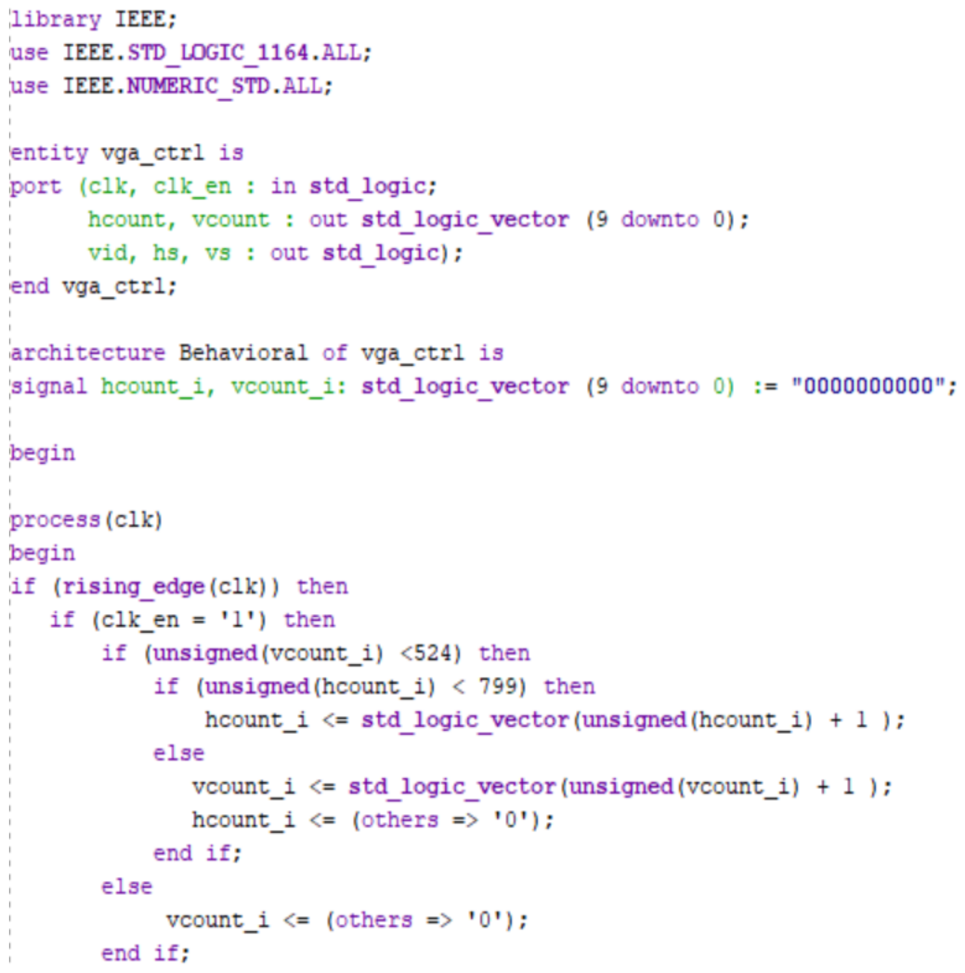
2 Lab Assignment 1: Design of VGA Controller

2.1 Theory of Operation

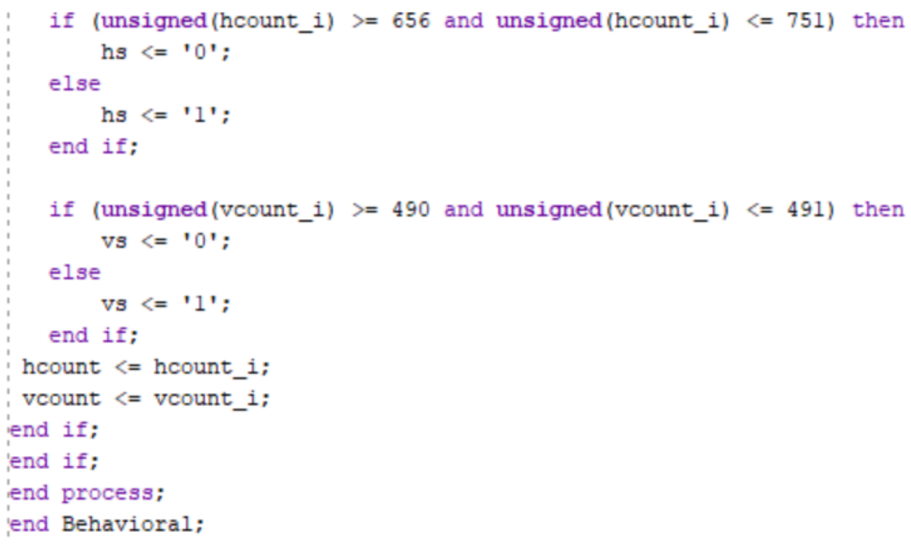
The VGA controller module takes as input a clock and a clock enable and outputs two counter variables, ‘hcount’ and ‘vcount’, a display variable ‘vid’, horizontal sync variable ‘hs’ and vertical sync variable ‘vs’. This module behaves in the following described manner. The ‘hcount’ variable counts from [0, 799] and it increments at every rising edge and when clock enable is 1. The ‘vcount’ variable counts from [0, 524] and it increments at every rising edge and when clock enable is 1 and when ‘hcount has reset to 0. The display variable ‘vid’ is ON when ‘hcount’ is between [0, 639], and ‘vcount’ is between [0, 479], otherwise it is OFF. The ‘hs’ variable is 0 when ‘hcount’ variable is between [656, 751], otherwise it is 1. The ‘vs’ variable is 0 when ‘vcount’ is between [656, 751], otherwise it is 1.

2.2 Design

Listing 1: vga_ctrl.vhd



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

entity vga_ctrl is
port (clk, clk_en : in std_logic;
      hcount, vcount : out std_logic_vector (9 downto 0);
      vid, hs, vs : out std_logic);
end vga_ctrl;

architecture Behavioral of vga_ctrl is
signal hcount_i, vcount_i: std_logic_vector (9 downto 0) := "0000000000";

begin

process(clk)
begin
if (rising_edge(clk)) then
   if (clk_en = '1') then
       if (unsigned(vcount_i) <524) then
           if (unsigned(hcount_i) < 799) then
               hcount_i <= std_logic_vector(unsigned(hcount_i) + 1 );
           else
              vcount_i <= std_logic_vector(unsigned(vcount_i) + 1 );
              hcount_i <= (others => '0');
           end if;
       else
            vcount_i <= (others => '0');
       end if;
```

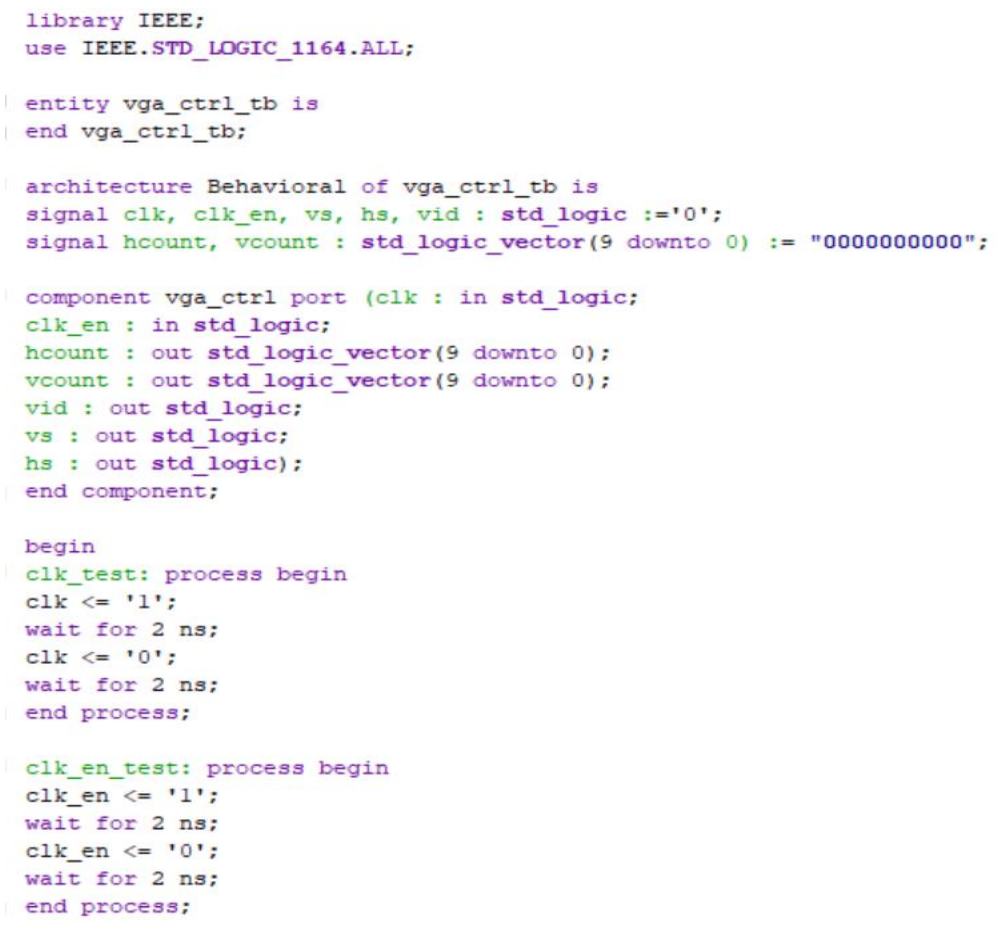


```
    if (unsigned(hcount_i) >= 656 and unsigned(hcount_i) <= 751) then
        hs <= '0';
    else
        hs <= '1';
    end if;

    if (unsigned(vcount_i) >= 490 and unsigned(vcount_i) <= 491) then
        vs <= '0';
    else
        vs <= '1';
    end if;
  hcount <= hcount_i;
  vcount <= vcount_i;
end if;
end if;
end process;
end Behavioral;
```

2.3 Test

The VGA Controller design is tested with a testbench which showed satisfactory simulation results.

Listing 2: vga_ctrl_tb.vhd



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

entity vga_ctrl_tb is
end vga_ctrl_tb;

architecture Behavioral of vga_ctrl_tb is
signal clk, clk_en, vs, hs, vid : std_logic :='0';
signal hcount, vcount : std_logic_vector(9 downto 0) := "0000000000";

component vga_ctrl port (clk : in std_logic;
clk_en : in std_logic;
hcount : out std_logic_vector(9 downto 0);
vcount : out std_logic_vector(9 downto 0);
vid : out std_logic;
vs : out std_logic;
hs : out std_logic);
end component;

begin
clk_test: process begin
clk <= '1';
wait for 2 ns;
clk <= '0';
wait for 2 ns;
end process;

clk_en_test: process begin
clk_en <= '1';
wait for 2 ns;
clk_en <= '0';
wait for 2 ns;
end process;
```

```
vga_ctrl_test: vga_ctrl port map(
clk => clk,
clk_en => clk_en,
hcount => hcount,
vcount=> vcount,
hs => hs,
vs => vs,
vid => vid);

end Behavioral;
```

Simulations results:

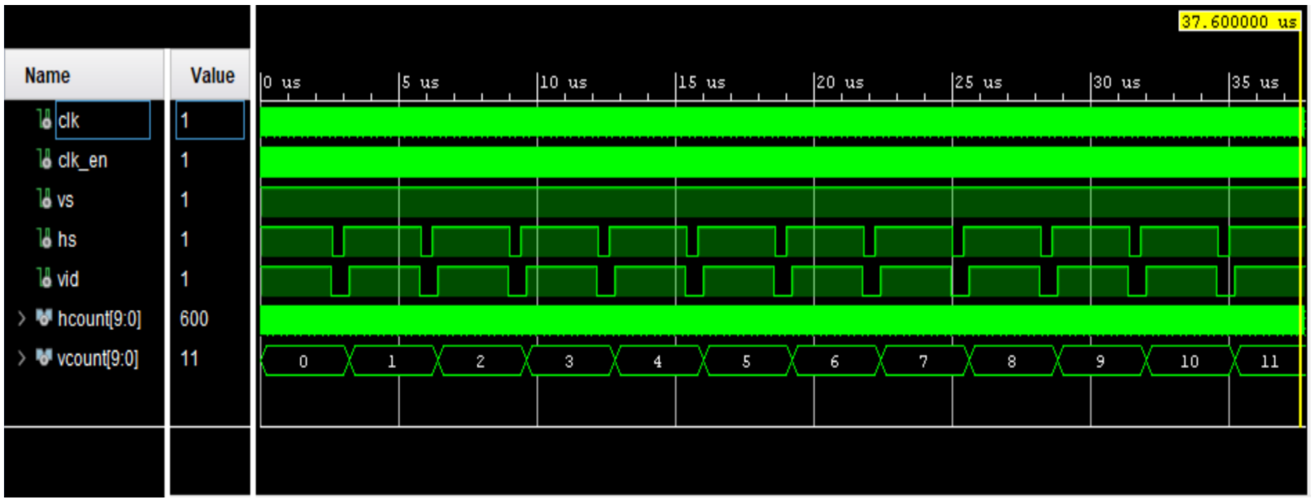


Figure 5: VGA Controller Simulation

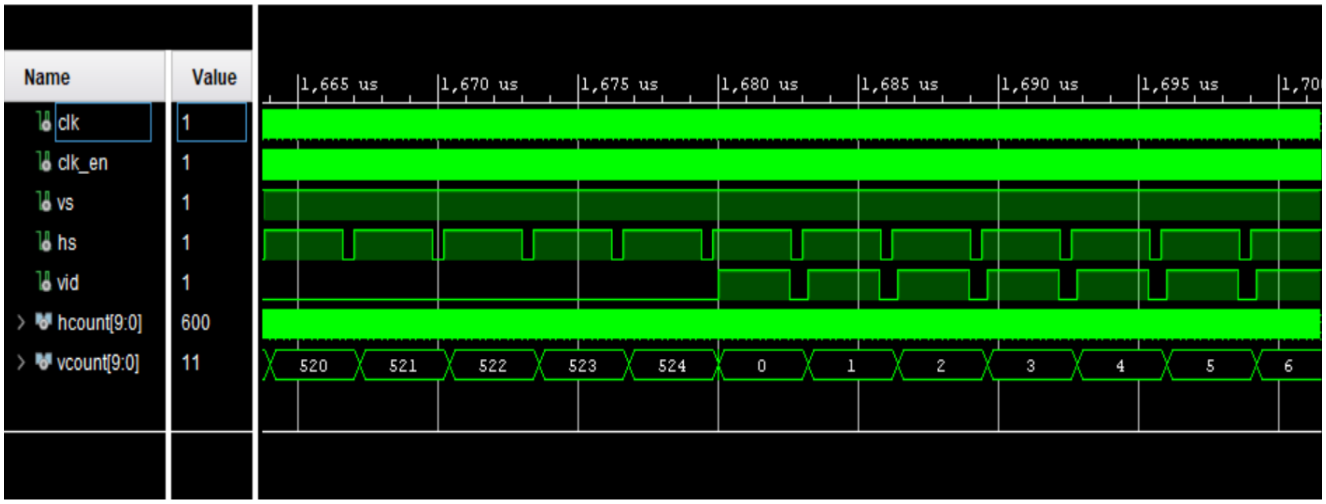


Figure 6: VGA Controller Simulation showing ‘vcount’ counts till 524 and ‘vid’ transition to 1

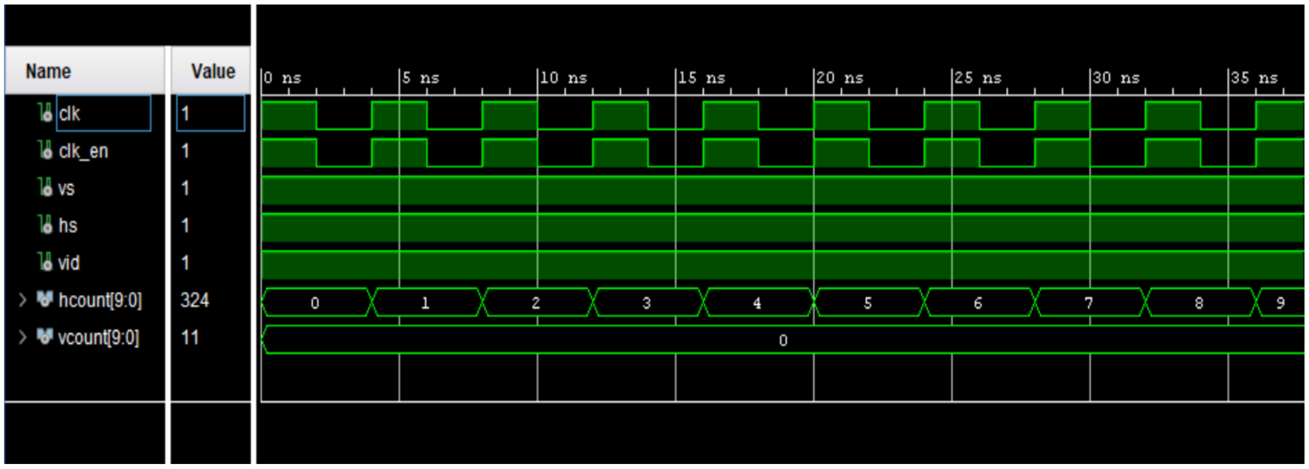


Figure 7: VGA Controller Simulation, Zoomed in to show ‘hcount’

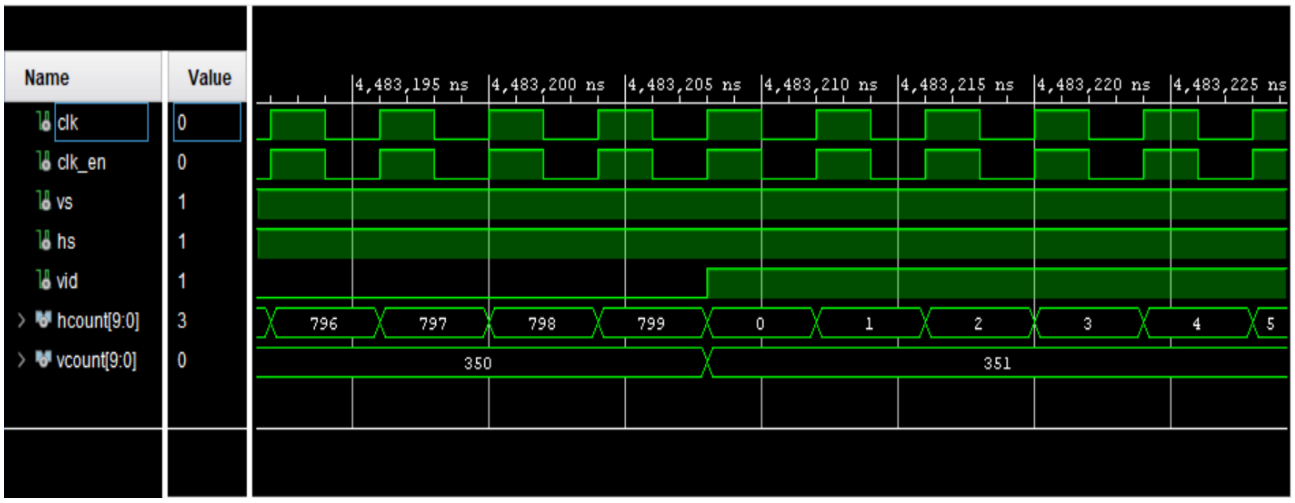


Figure 8: VGA Controller Simulation, Zoomed in to ‘hcount’ counts till 799 and ‘vid’ transition to 1

2.4 Implementation

2.4.1 VGA Controller Design

2.4.1.1 Elaboration Schematic:

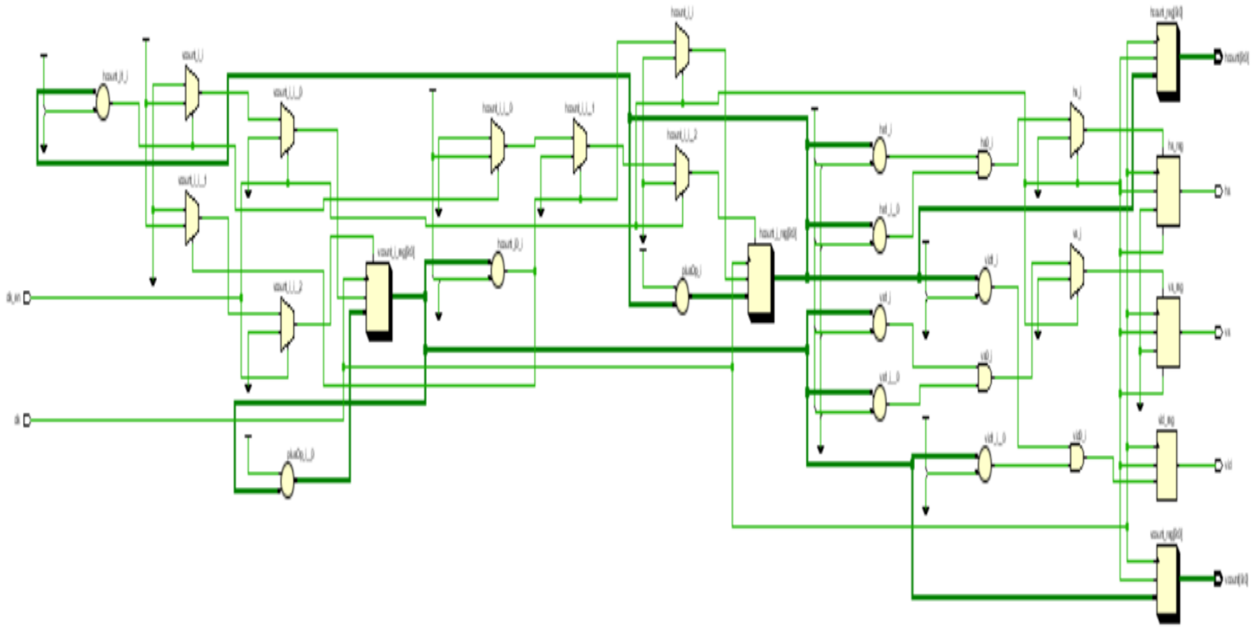


Figure 9: VGA Controller RTL Schematic

2.4.1.2 Synthesis Schematic:

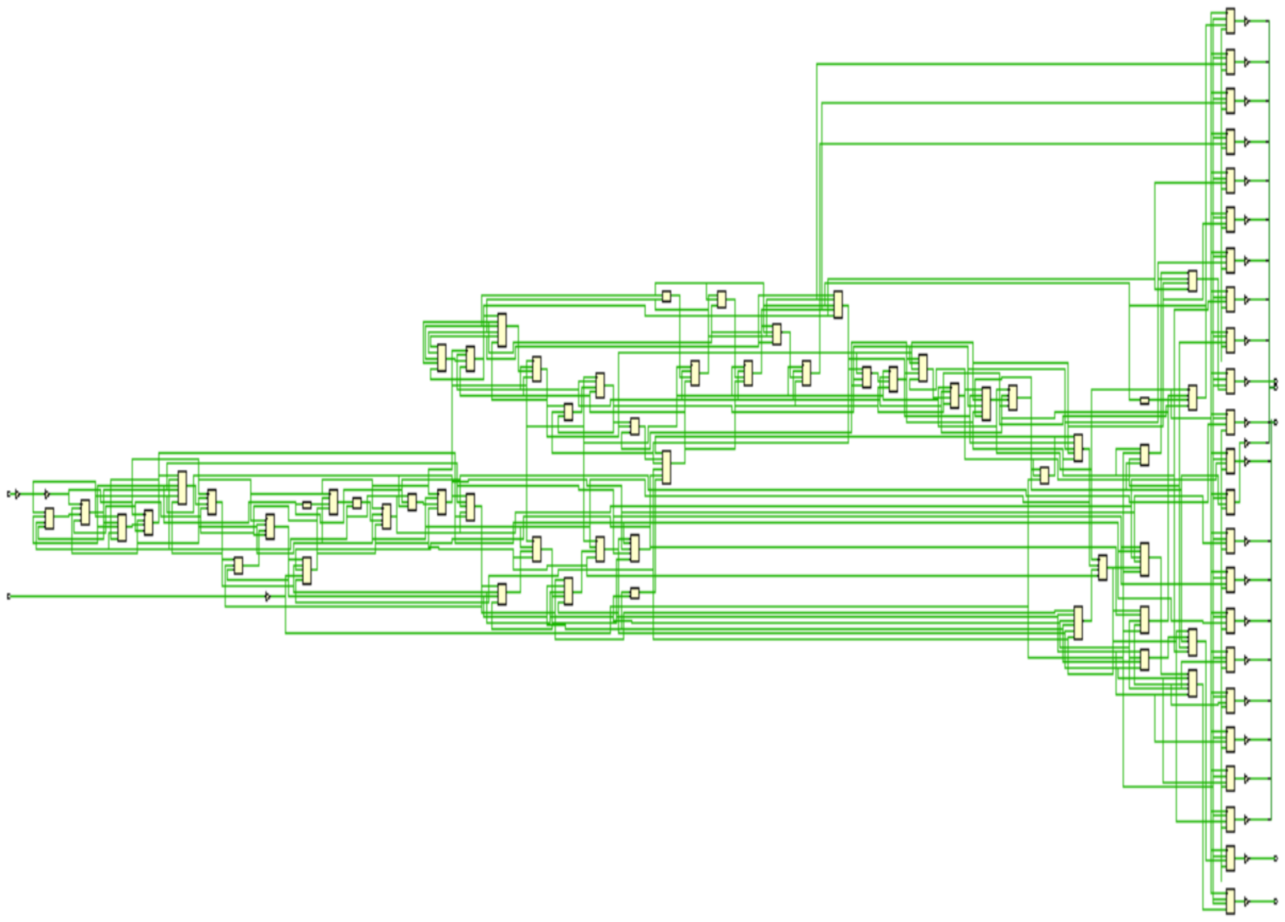
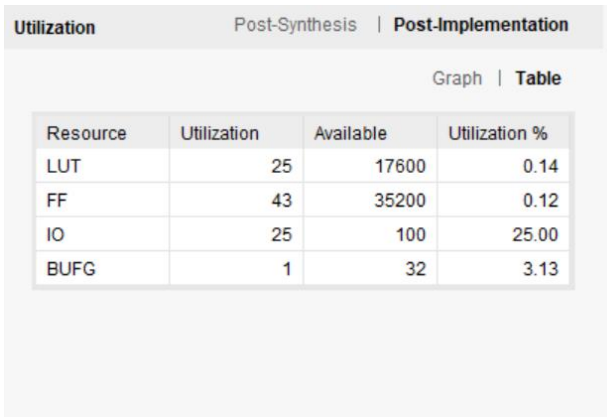


Figure 10: VGA Controller Synthesis Schematic

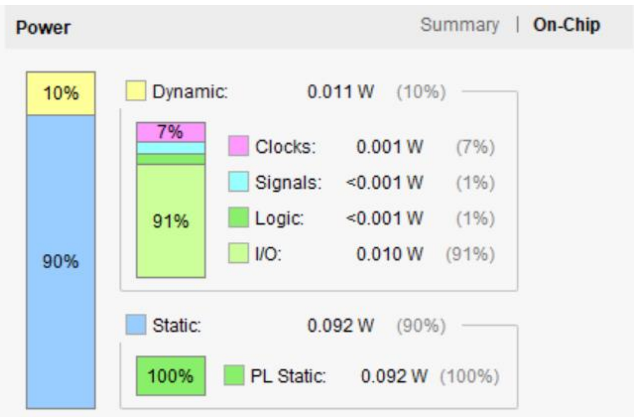
2.4.1.3 Power Graph and Utilization Table:



Utilization — Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	25	17600	0.14
FF	43	35200	0.12
IO	25	100	25.00
BUFG	1	32	3.13



Power — Summary | **On-Chip**

Dynamic:	0.011 W	(10%)	
	Clocks:	0.001 W	(7%)
	Signals:	<0.001 W	(1%)
	Logic:	<0.001 W	(1%)
	I/O:	0.010 W	(91%)
Static:	0.092 W	(90%)	
	PL Static:	0.092 W	(100%)

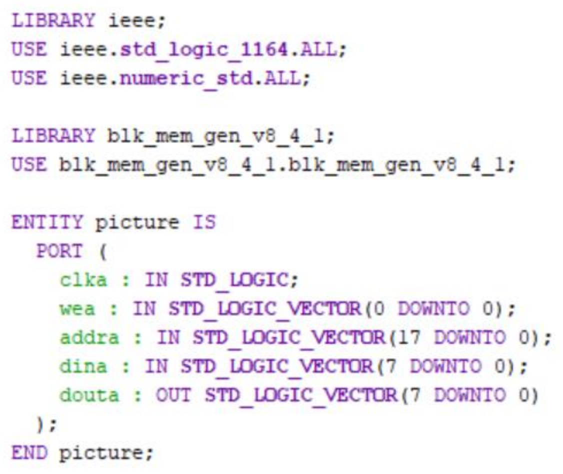
Figure 11: VGA Controller Project Summary

3 Lab Assignment 2:

3.1 Block Memory Creation

By following the given instructions in the report, a block memory was generated using IP Catalog. The main VHDL file created has the below shown entity declaration which we instantiate later as a component in the image top level design.

Listing 3: picture.vhd (only entity declaration is shown here)



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY blk_mem_gen_v8_4_1;
USE blk_mem_gen_v8_4_1.blk_mem_gen_v8_4_1;

ENTITY picture IS
  PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END picture;
```

3.2 Pixel Pusher

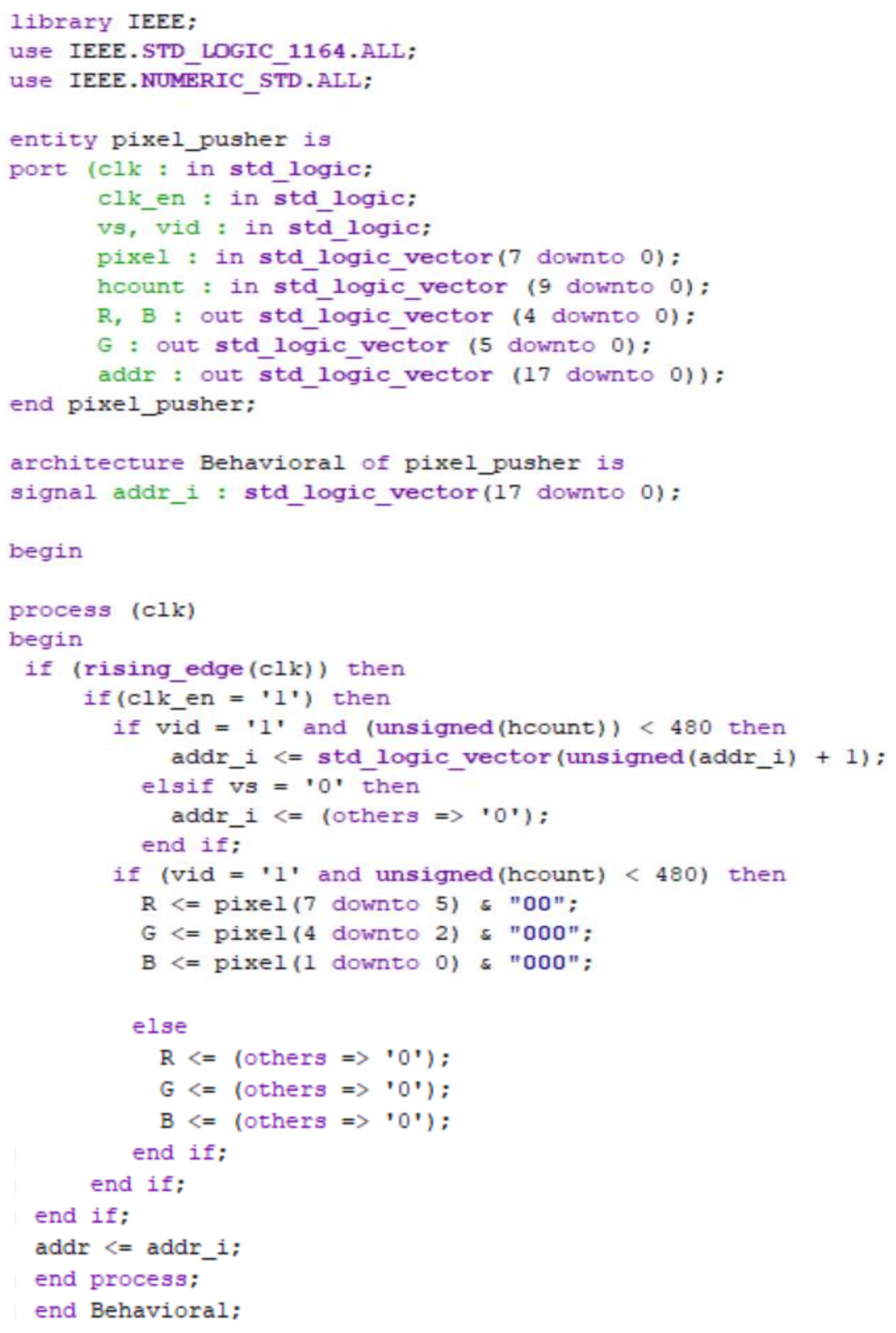
3.2.1 Theory of Operation

The pixel pusher module that takes as input a clock, a clock enable, a 1-bit ‘vs’, an 8-bit ‘pixel’ signal, a 10-bit ‘hcount’ signal, and a ‘vid’ signal and outputs two 5-bit ‘R’ and ‘B’ signals and a 6-bit ‘G’ signal and an 18-bit ‘addr’. The module behaves in the following manner. The internal 18 bit counter ‘addr’ increments at every rising clock edge when enable is 1, ‘vid’ is 1, and ‘hcount’ < 480 and resets synchronously when ‘vs’ is 0. At every rising clock edge when enable is 1, ‘vid’ is 1, and ‘hcount’ < 480, the output is as shown below, otherwise R, G, and B are 0.

R <= pixel(7 downto 5) & "00"
G <= pixel(4 downto 2) & "000"
B <= pixel(1 downto 0) & "000"

3.2.2 Design

Listing 4: pixel_pusher.vhd



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

entity pixel_pusher is
port (clk : in std_logic;
      clk_en : in std_logic;
      vs, vid : in std_logic;
      pixel : in std_logic_vector(7 downto 0);
      hcount : in std_logic_vector (9 downto 0);
      R, B : out std_logic_vector (4 downto 0);
      G : out std_logic_vector (5 downto 0);
      addr : out std_logic_vector (17 downto 0));
end pixel_pusher;

architecture Behavioral of pixel_pusher is
signal addr_i : std_logic_vector(17 downto 0);

begin

process (clk)
begin
 if (rising_edge(clk)) then
     if(clk_en = '1') then
       if vid = '1' and (unsigned(hcount)) < 480 then
           addr_i <= std_logic_vector(unsigned(addr_i) + 1);
         elsif vs = '0' then
           addr_i <= (others => '0');
         end if;
       if (vid = '1' and unsigned(hcount) < 480) then
         R <= pixel(7 downto 5) & "00";
         G <= pixel(4 downto 2) & "000";
         B <= pixel(1 downto 0) & "000";

        else
          R <= (others => '0');
          G <= (others => '0');
          B <= (others => '0');
        end if;
     end if;
 end if;
 addr <= addr_i;
 end process;
 end Behavioral;
```

3.2.3 Implementation

3.2.3.1 Elaboration Schematic:

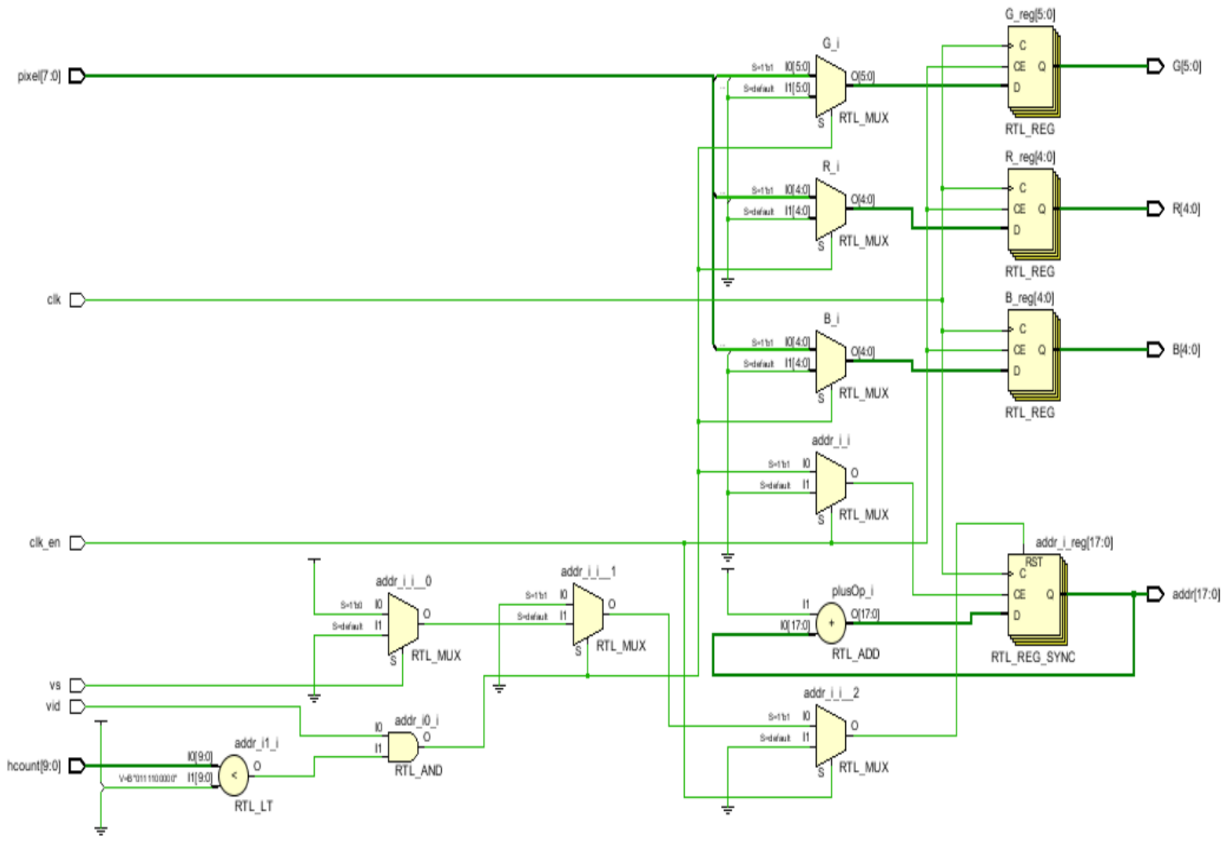


Figure 12: Pixel Pusher RTL Schematic

3.2.3.2 Synthesis Schematic:

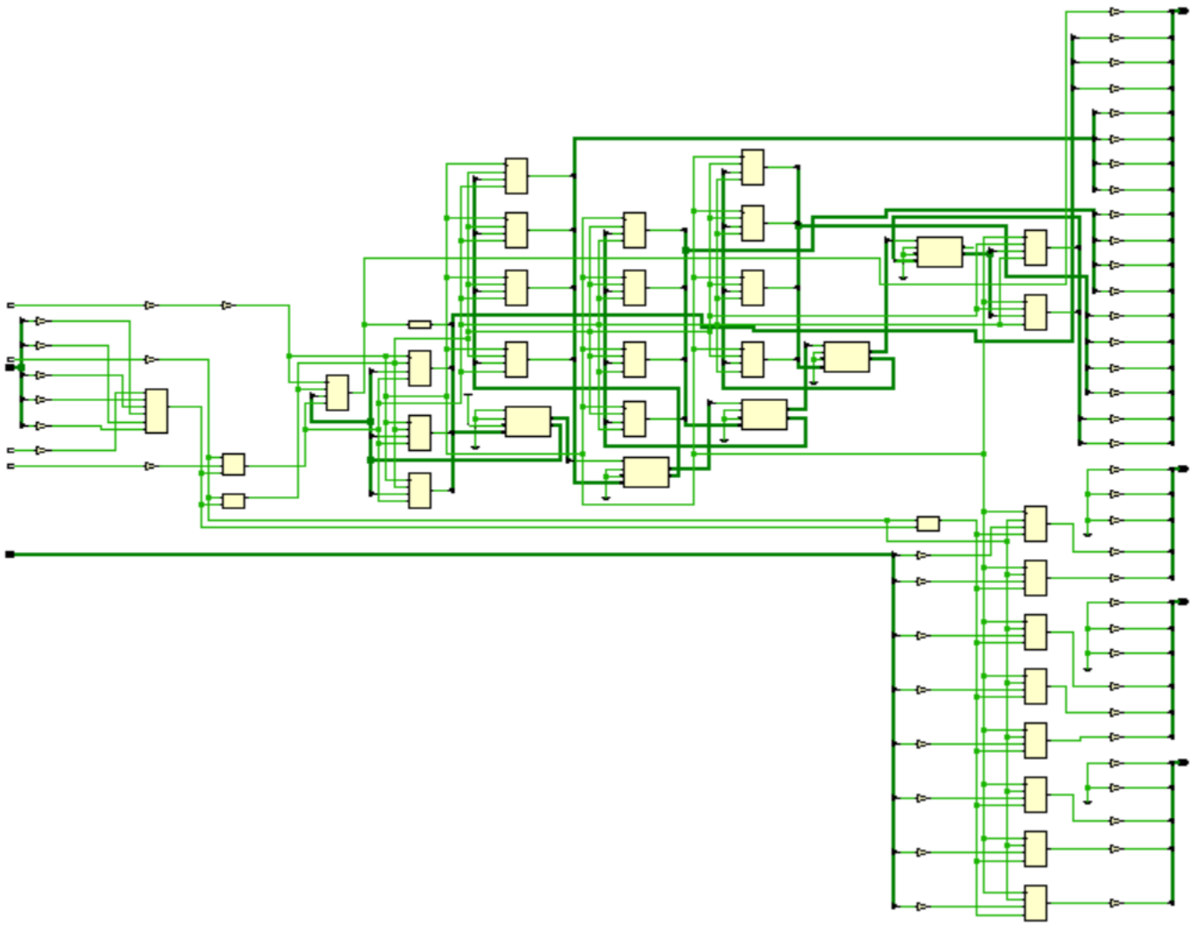
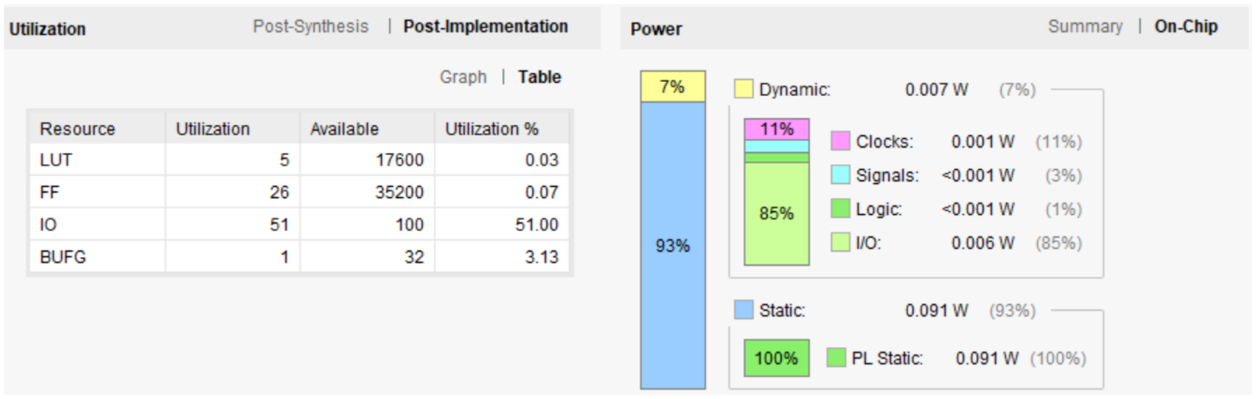


Figure 13: Pixel Pusher Synthesis Schematic

3.2.3.3 Power Graph and Utilization Table:



Resource	Utilization	Available	Utilization %
LUT	5	17600	0.03
FF	26	35200	0.07
IO	51	100	51.00
BUFG	1	32	3.13

Dynamic:	0.007 W	(7%)
Clocks:	0.001 W	(11%)
Signals:	<0.001 W	(3%)
Logic:	<0.001 W	(1%)
I/O:	0.006 W	(85%)
Static:	0.091 W	(93%)
PL Static:	0.091 W	(100%)

Figure 14: Pixel Pusher Project Summary

3.3 Block Diagram

The block diagram for the image top level design contains instances of block memory, pixel pusher, VGA controller, and a clock divider to produce a 25 MHz clock enable. It takes as input a clk signal and produces as output single bit vga_hs and vga_vs signals, 5-bit vga_r and vga_b signals, and a 6-bit vga_g signal.

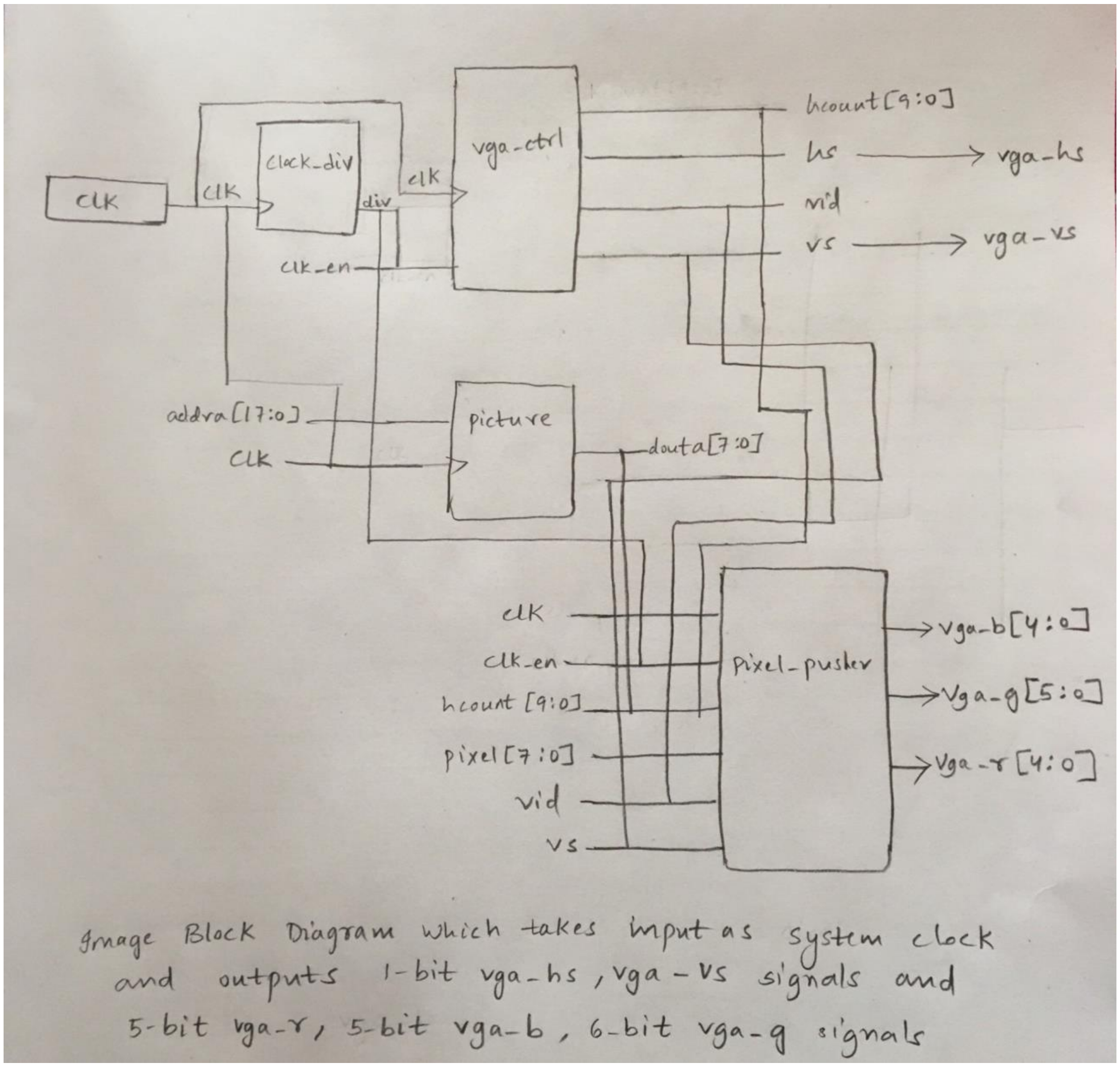


Image Block Diagram which takes input as system clock and outputs 1-bit vga_hs, vga-vs signals and 5-bit vga-r, 5-bit vga-b, 6-bit vga-g signals

Figure 15: Image Block Diagram

3.4 Image Top-Level

3.4.1 Theory of Operation

The designed, modified clock divider that produces a 25 MHz output, VGA controller module, and pixel pusher module, the created block memory called to picture to store the image data are instantiated using structural VHDL modeling in this final image top level design.

3.4.2 Design

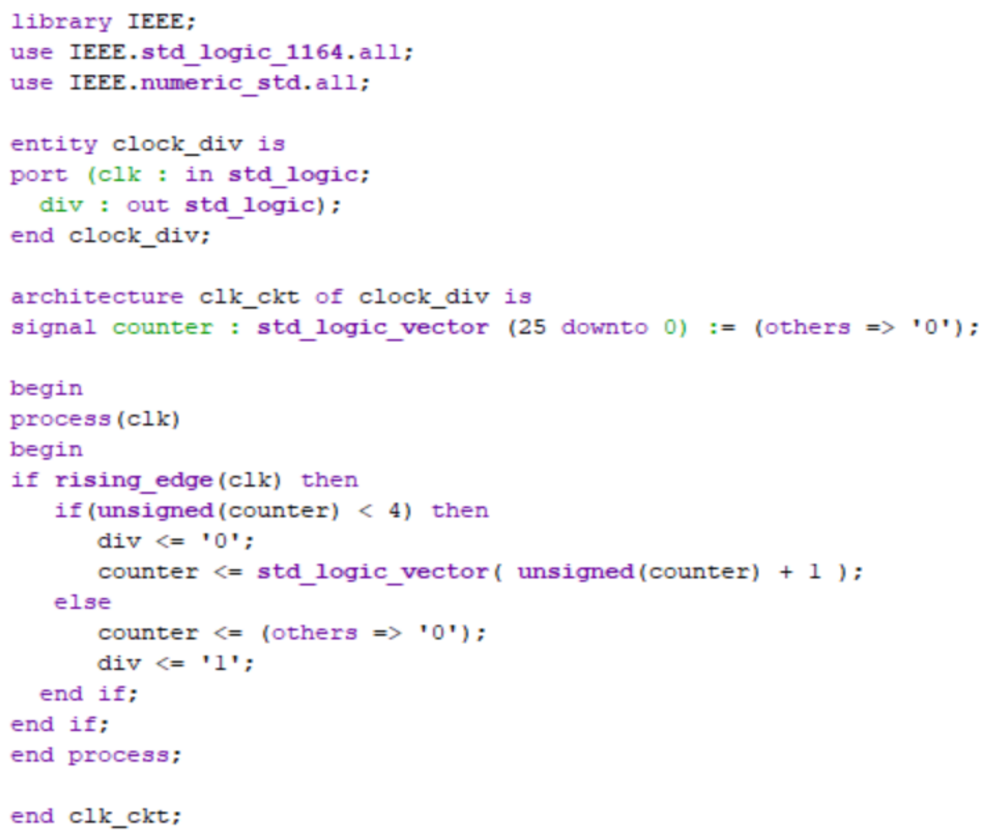
3.4.2.1 Clock Divider Design

The clock divider circuit produces the clock divided signal as output with a frequency of 25 MHz.

The input clock frequency is 125 MHz.

- ⇨ $(125*10^6)/(25*10^6) = 5$ Hz

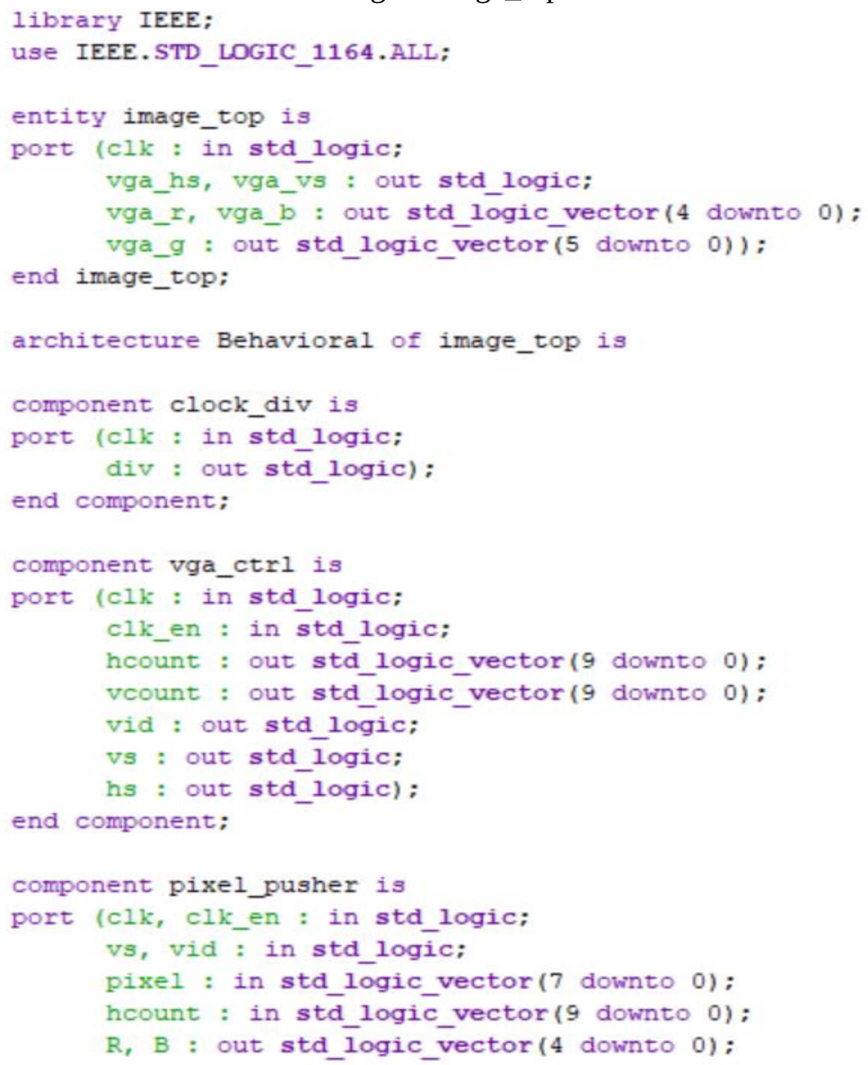
Listing 5: clock_div.vhd



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity clock_div is
port (clk : in std_logic;
  div : out std_logic);
end clock_div;

architecture clk_ckt of clock_div is
signal counter : std_logic_vector (25 downto 0) := (others => '0');

begin
process(clk)
begin
if rising_edge(clk) then
   if(unsigned(counter) < 4) then
      div <= '0';
      counter <= std_logic_vector( unsigned(counter) + 1 );
   else
      counter <= (others => '0');
      div <= '1';
  end if;
end if;
end process;

end clk_ckt;
```

3.4.2.2 Image Top Level Design

Listing 6: image_top.vhd



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

entity image_top is
port (clk : in std_logic;
      vga_hs, vga_vs : out std_logic;
      vga_r, vga_b : out std_logic_vector(4 downto 0);
      vga_g : out std_logic_vector(5 downto 0));
end image_top;

architecture Behavioral of image_top is

component clock_div is
port (clk : in std_logic;
      div : out std_logic);
end component;

component vga_ctrl is
port (clk : in std_logic;
      clk_en : in std_logic;
      hcount : out std_logic_vector(9 downto 0);
      vcount : out std_logic_vector(9 downto 0);
      vid : out std_logic;
      vs : out std_logic;
      hs : out std_logic);
end component;

component pixel_pusher is
port (clk, clk_en : in std_logic;
      vs, vid : in std_logic;
      pixel : in std_logic_vector(7 downto 0);
      hcount : in std_logic_vector(9 downto 0);
      R, B : out std_logic_vector(4 downto 0);
      G : out std_logic_vector(5 downto 0);
      addr : out std_logic_vector(17 downto 0));
end component;

component picture is
port (clka : in std_logic;
   addra : in std_logic_vector(17 downto 0);
   douta : out std_logic_vector(7 downto 0));
end component;

signal div, vid, hs, vs : std_logic;
signal pixel : std_logic_vector(7 downto 0);
signal addr : std_logic_vector(17 downto 0);
signal hcount, vcount : std_logic_vector(9 downto 0);

begin

vga_vs <= vs;

u1: clock_div port map(
clk => clk,
div => div);

u2: vga_ctrl port map(
clk => clk,
clk_en => div,
hcount => hcount,
vcount => vcount,
vid => vid,
vs => vs,
hs => vga_hs);
u3: pixel_pusher port map(
clk => clk,
clk_en => div,
vs => vs,
hcount => hcount,
R => vga_r,
G => vga_g,
B => vga_b,
addr => addr,
vid => vid,
pixel => pixel);

u4: picture port map(
clka => clk,
addra => addr,
douta => pixel);

end Behavioral;
```

3.4.3 Implementation

3.4.3.1 Elaboration Schematic:

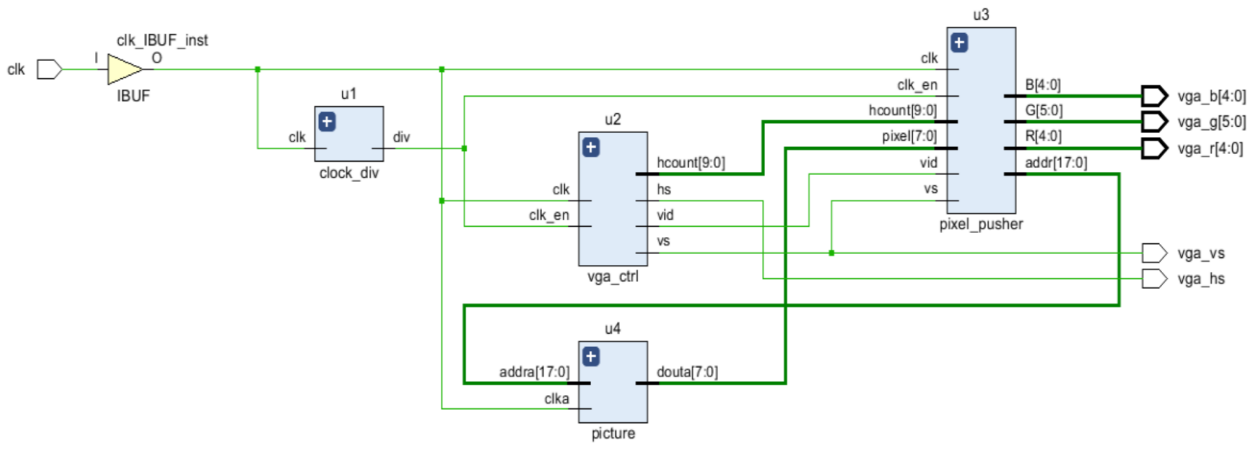


Figure 16: Image Top Level RTL Schematic

3.4.3.2 Synthesis Schematic:

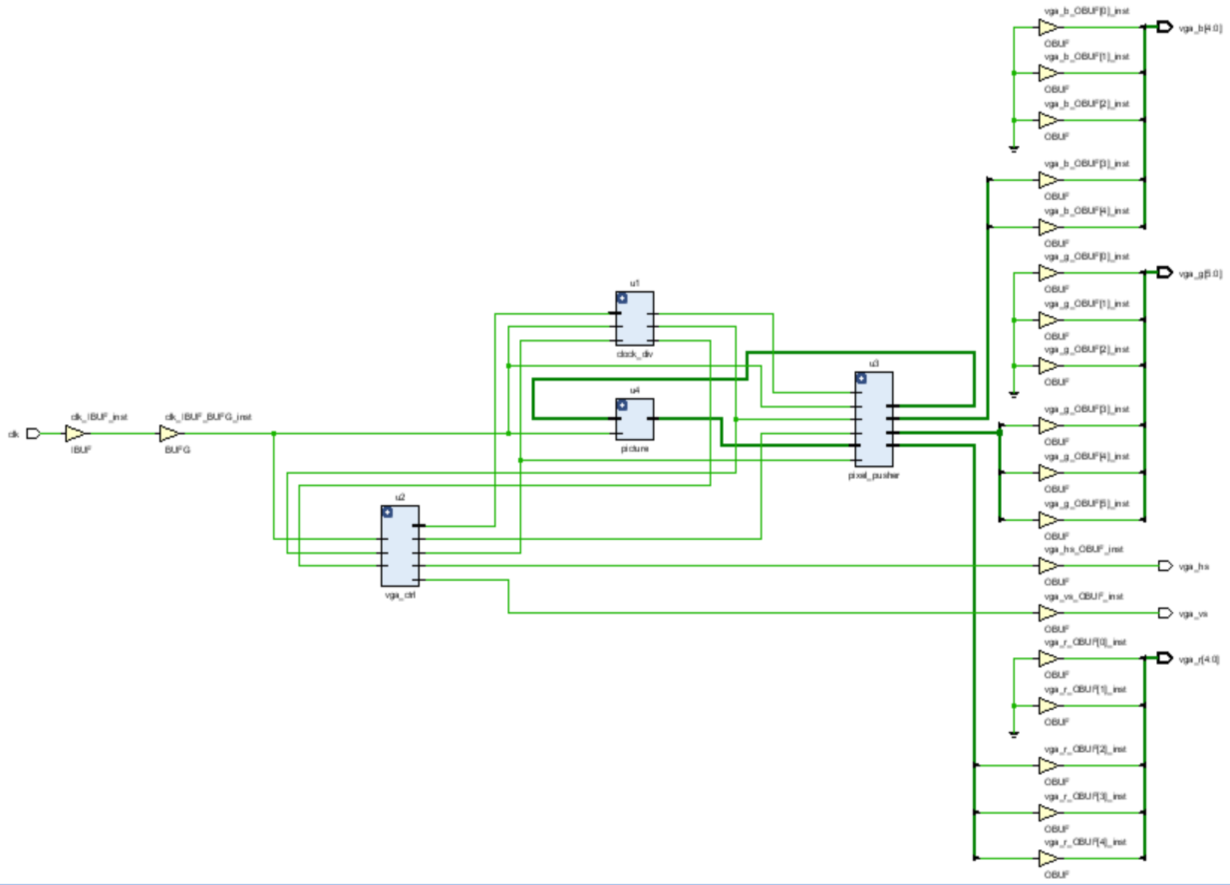
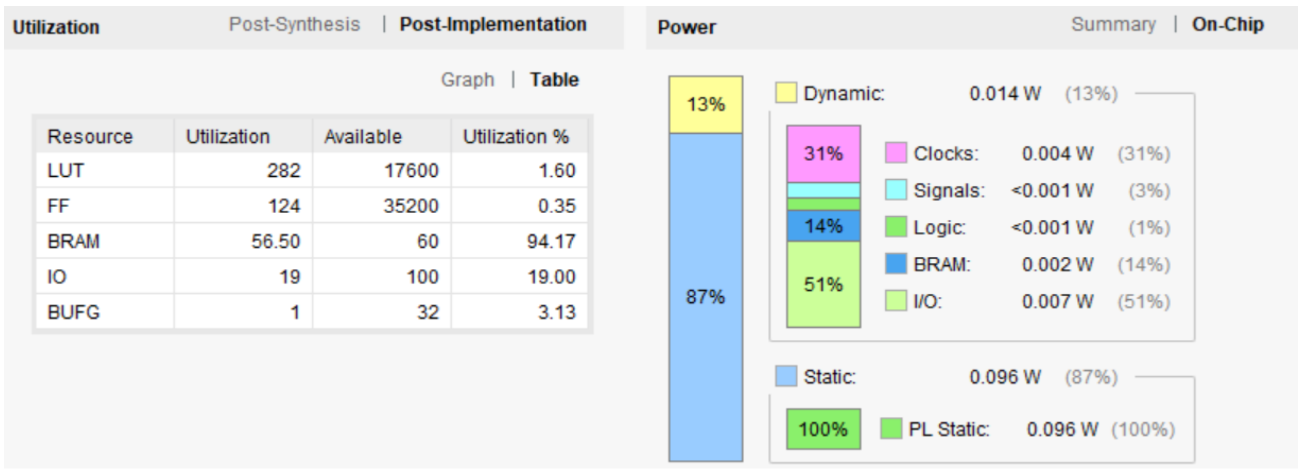


Figure 17: Image Top Level Synthesis Schematic

3.4.3.3 Power Graph and Utilization Table:



Utilization — Post-Synthesis | Post-Implementation — Graph | Table

Resource	Utilization	Available	Utilization %
LUT	282	17600	1.60
FF	124	35200	0.35
BRAM	56.50	60	94.17
IO	19	100	19.00
BUFG	1	32	3.13

Power — Summary | On-Chip

🟨	Dynamic:	0.014 W	(13%)
🟪	Clocks:	0.004 W	(31%)
🟦	Signals:	<0.001 W	(3%)
🟩	Logic:	<0.001 W	(1%)
🟦	BRAM:	0.002 W	(14%)
🟩	I/O:	0.007 W	(51%)
🟦	Static:	0.096 W	(87%)
🟩	PL Static:	0.096 W	(100%)

Figure 18: Image Top Level Project Summary

3.4.3.4 Results:

The output file generated by running the MATLAB script is shown on the top left corner of the computer screen as expected.

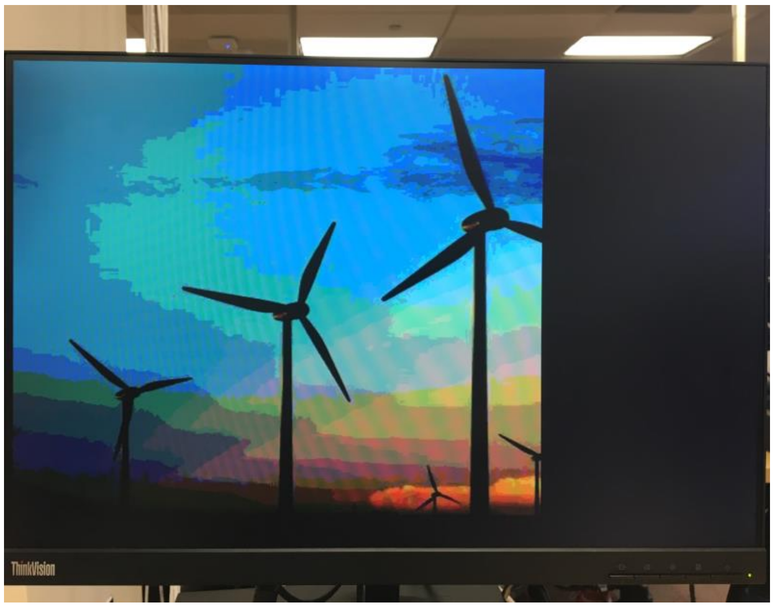
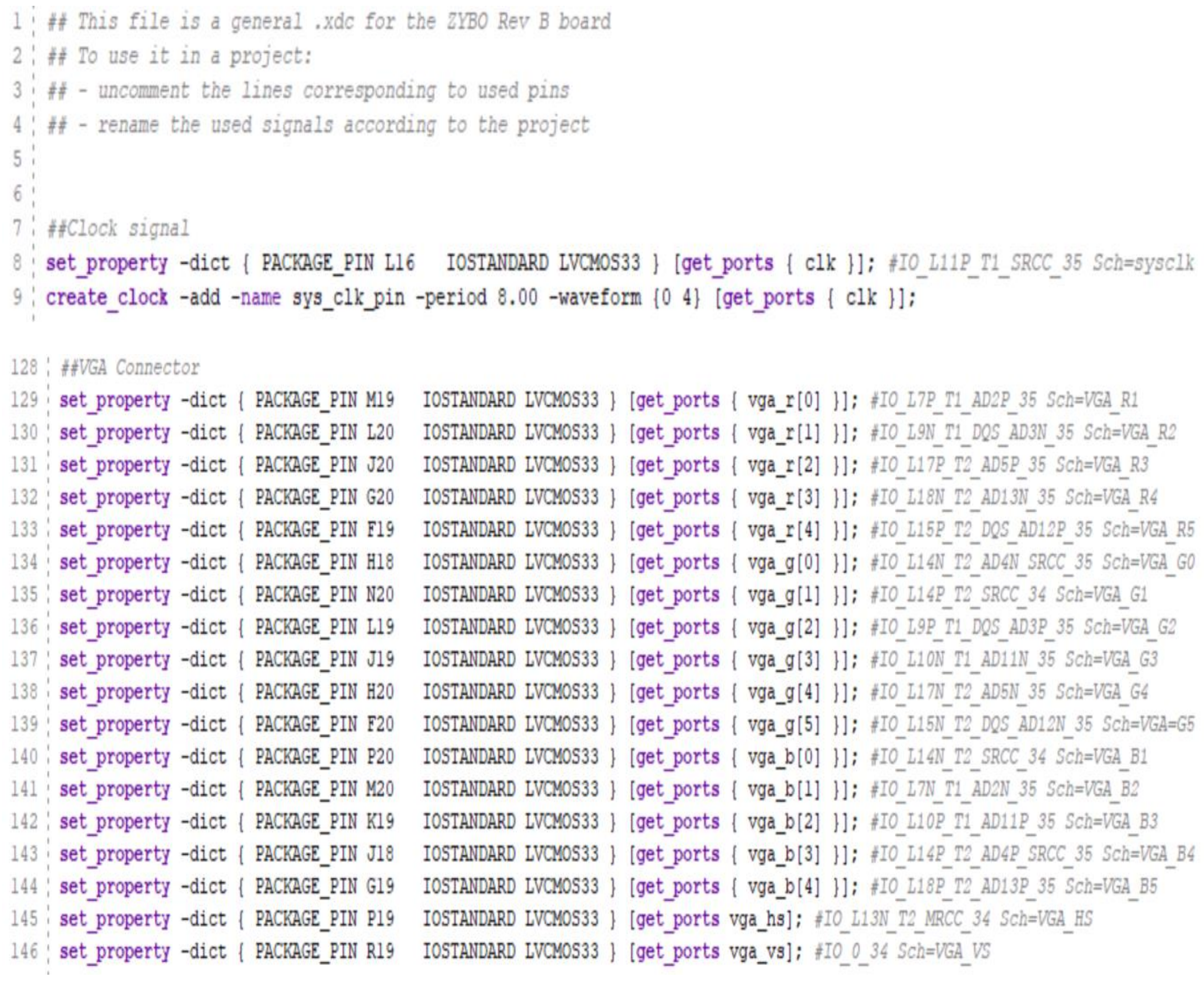


Figure 19: Image displayed on the computer screen

3.5 Constraints File

The constraints (XDC) file was changed to map the appropriate signals on the board to the final image top level design. A operating frequency of clock 125 MHz needs to sent to the Zybo board so pin corresponding to clock is enabled. The VGA Connector cable connects VGA Connector in the computer and the VGA Connector on the board. The lines of code corresponding to VGA Connector was uncommented so that the pins are enabled for the signals vga_r, vga_b, vga_g, vga_hs, vga_vs respectively. The lines uncommented in the XDC file are shown in the below figures.

Listing 7: ZYBO_Master.xdc



```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];

##VGA Connector
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { vga_r[0] }]; #IO_L7P_T1_AD2P_35 Sch=VGA_R1
set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { vga_r[1] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=VGA_R2
set_property -dict { PACKAGE_PIN J20   IOSTANDARD LVCMOS33 } [get_ports { vga_r[2] }]; #IO_L17P_T2_AD5P_35 Sch=VGA_R3
set_property -dict { PACKAGE_PIN G20   IOSTANDARD LVCMOS33 } [get_ports { vga_r[3] }]; #IO_L18N_T2_AD13N_35 Sch=VGA_R4
set_property -dict { PACKAGE_PIN F19   IOSTANDARD LVCMOS33 } [get_ports { vga_r[4] }]; #IO_L15P_T2_DQS_AD12P_35 Sch=VGA_R5
set_property -dict { PACKAGE_PIN H18   IOSTANDARD LVCMOS33 } [get_ports { vga_g[0] }]; #IO_L14N_T2_AD4N_SRCC_35 Sch=VGA_G0
set_property -dict { PACKAGE_PIN N20   IOSTANDARD LVCMOS33 } [get_ports { vga_g[1] }]; #IO_L14P_T2_SRCC_34 Sch=VGA_G1
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { vga_g[2] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=VGA_G2
set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports { vga_g[3] }]; #IO_L10N_T1_AD11N_35 Sch=VGA_G3
set_property -dict { PACKAGE_PIN H20   IOSTANDARD LVCMOS33 } [get_ports { vga_g[4] }]; #IO_L17N_T2_AD5N_35 Sch=VGA_G4
set_property -dict { PACKAGE_PIN F20   IOSTANDARD LVCMOS33 } [get_ports { vga_g[5] }]; #IO_L15N_T2_DQS_AD12N_35 Sch=VGA=G5
set_property -dict { PACKAGE_PIN P20   IOSTANDARD LVCMOS33 } [get_ports { vga_b[0] }]; #IO_L14N_T2_SRCC_34 Sch=VGA_B1
set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { vga_b[1] }]; #IO_L7N_T1_AD2N_35 Sch=VGA_B2
set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports { vga_b[2] }]; #IO_L10P_T1_AD11P_35 Sch=VGA_B3
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { vga_b[3] }]; #IO_L14P_T2_AD4P_SRCC_35 Sch=VGA_B4
set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports { vga_b[4] }]; #IO_L18P_T2_AD13P_35 Sch=VGA_B5
set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports vga_hs]; #IO_L13N_T2_MRCC_34 Sch=VGA_HS
set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports vga_vs]; #IO_0_34 Sch=VGA_VS
```

5 Discussion

5.1 Observations

- This lab assignment helped me to understand how the concept of Video Graphics Array and the creation of block memory using IP catalog was introduced for the first time in this lab.
- The dimensions of my chosen image is 450x450. Since the script generates a 480x480 output resolution due to memory limitations on the Zybo board, the image was padded with black as mentioned in the report.
- Once you create the block memory as per the given instructions in the report, you can see its instantiation template, clock details, VHDL file by going to the IP Sources tab.
- The utilization table for Image Top Level shows that BRAM was 94% utilized which means that nearly the entire block memory size that we created has been utilized not wasting any memory space.

5.2 Problem areas

- This lab though at first seemed straight forward, I spent long time in the VGA Controller getting the simulation right. At first, no image appeared on the screen. This was fixed by checking that the vid variable goes ON and OFF in the correctly in the intervals given for counter variables, hcount and vcount.
- Second problem was that my image was continuously flickering and this was correct by the way the 'if' statements were written for counter variables, hcount and vcount.
- The pixel pusher modules was fairly simple.
- In the Image top level design, I got confused whether to port map the 'clka' variable in the entity of the block memory 'picture' to the system clock or the divided clock from clock_div circuit as the simplified block diagram given in the report shows that clock and the divided clock both go to block memory 'picture'. Then I figured that since it is block memory creation, it should be connected to the system clock and not the divided clock.

5.3 Questions/Follow Up

Why is clock enable not given to block memory 'picture'?

How are the timing calculations done for the horizontal and vertical sync timings in Table 1?