Electronic Microsystems Project on VGA Controller

by- Vinay Garade

Table of Content:

1. Project description:

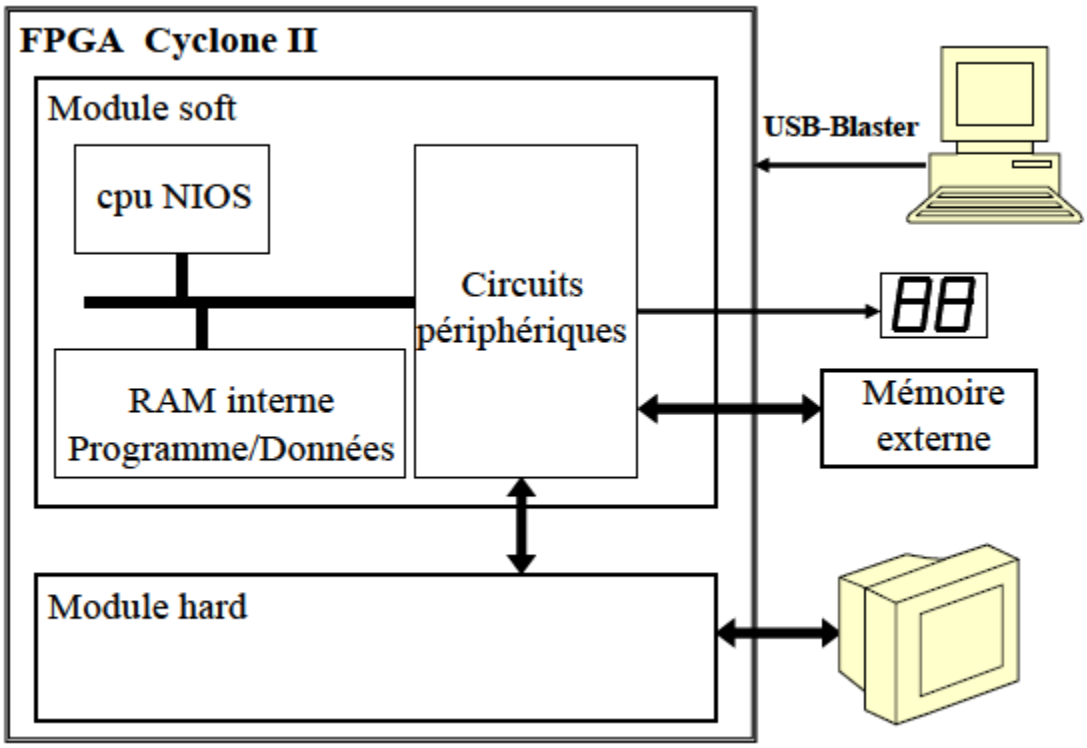
Display on an VGA video monitor an animation representing the earth describing its movement of revolution and the moon revolving around the earth. Animation management will be fully integrated into an FPGA circuit (Cyclone III or V from Altera). Binary representations of the two images (earth, moon) will be stored (linearly) in a ROM memory (cf. Annex III). The design will

be divided into two part: the first part is "hardware" to control VGA screen signals and a part “Software” (programming in C) in a processor embedded in the FPGA chip for management.

2. Methodology

Integrating a digital function in a programmable component:

- Analysis and functional breakdown of the treatment
- Partitioning of processing between programmed solution (processor core and program processing) and wired solution (specific hardware architecture)
- Global Architecture of the circuit



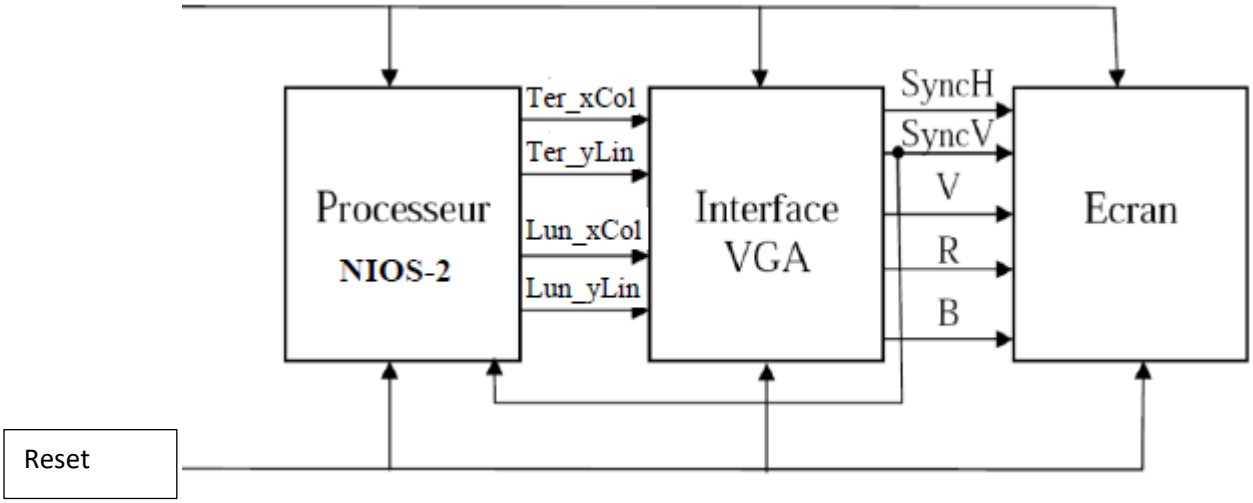
FPGA Figure 2.1

- Design of different blocks:
  - Hardware module: graphic description and synthesis of specified blocks (Quartus II software) and validation
  - Software module: hard configuration of the NIOS processor and its peripheral circuits (Software Quartus II + QSys) and writing of the processing program in C language (NIOS-II software).

- Configuration and validation of the functionality of the entire system.

3. Design of the display system:

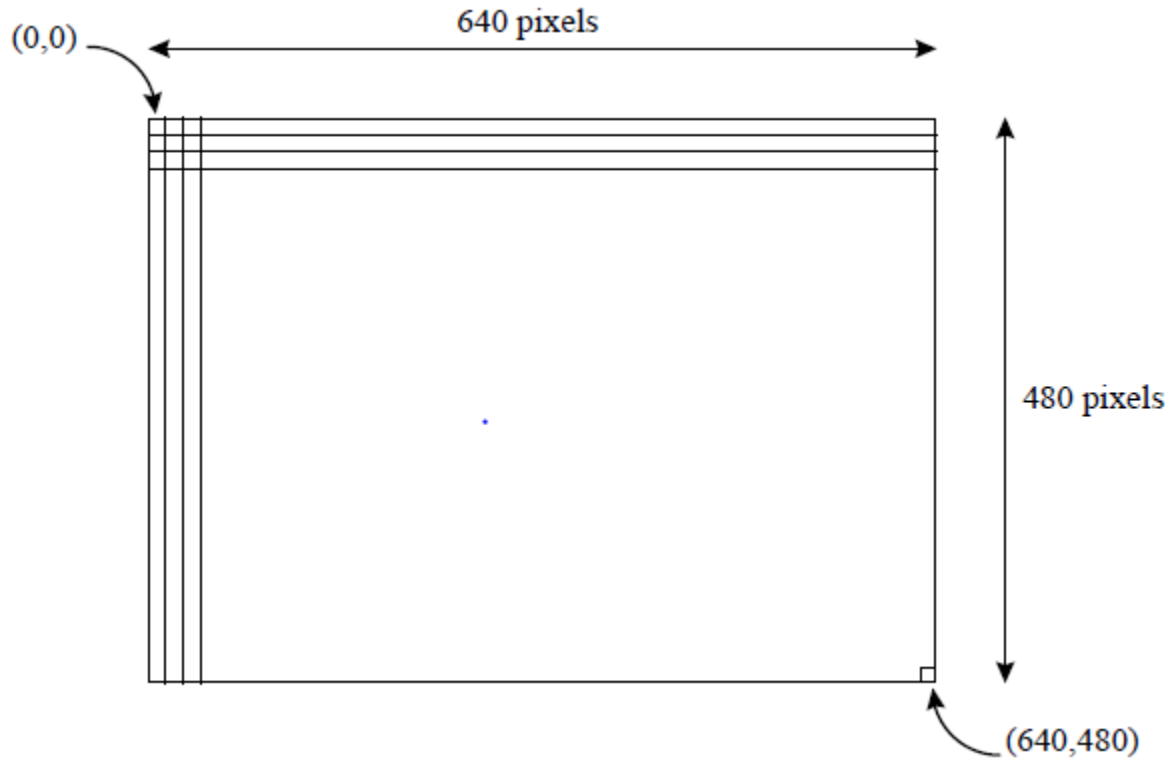
Clock



VGA Display System Figure 3.1

VGA Signal:

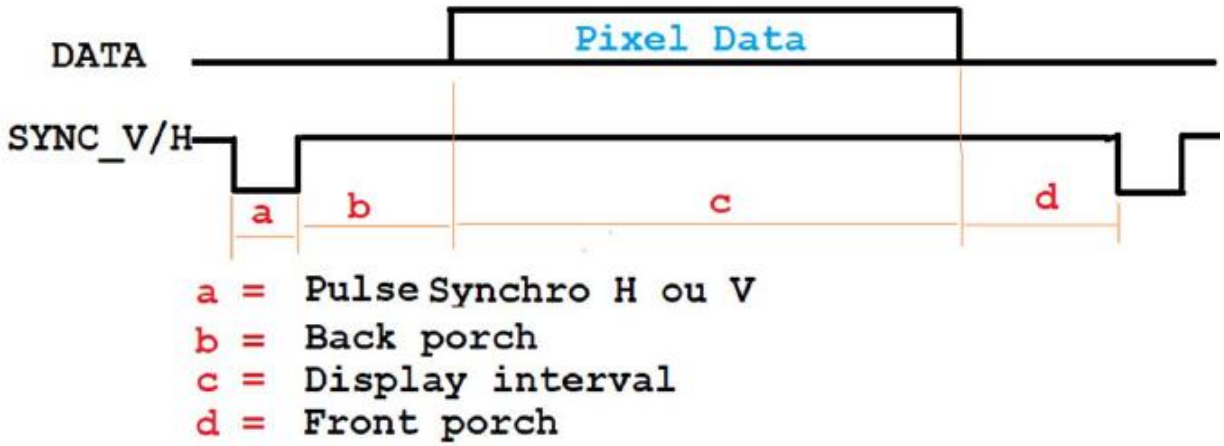
For a monitor to correctly display an image, the pixels of the image must be present at specific moments and accompanied by synchronization signals. Each of pixels is coded in RGB. The image displayed by a VGA screen typically contains 640 columns and 480 rows.



VGA screen Figure 3.2

VGA screen uses sync signals and a clock to specify when each pixel should be displayed. On the basis of the synchronization signals, the monitor refreshes the screen. He starts with the pixel at the top left of the image (coordinate point (0,0) in the figure), and refreshes all the pixels of the line. Then on the rising edge of the horizontal sync signal, the monitor updates the second line of the image; and so on until you reach the last line of the screen. When it reaches the bottom of the image, the monitor waits for the rising edge of the sync signal vertical to start

the refresh again, starting with the pixel of coordinates (0,0). As will be understood, the horizontal and vertical synchronization signals must be generated at specific moments to correctly synchronize the monitor when it receives the values of pixels.



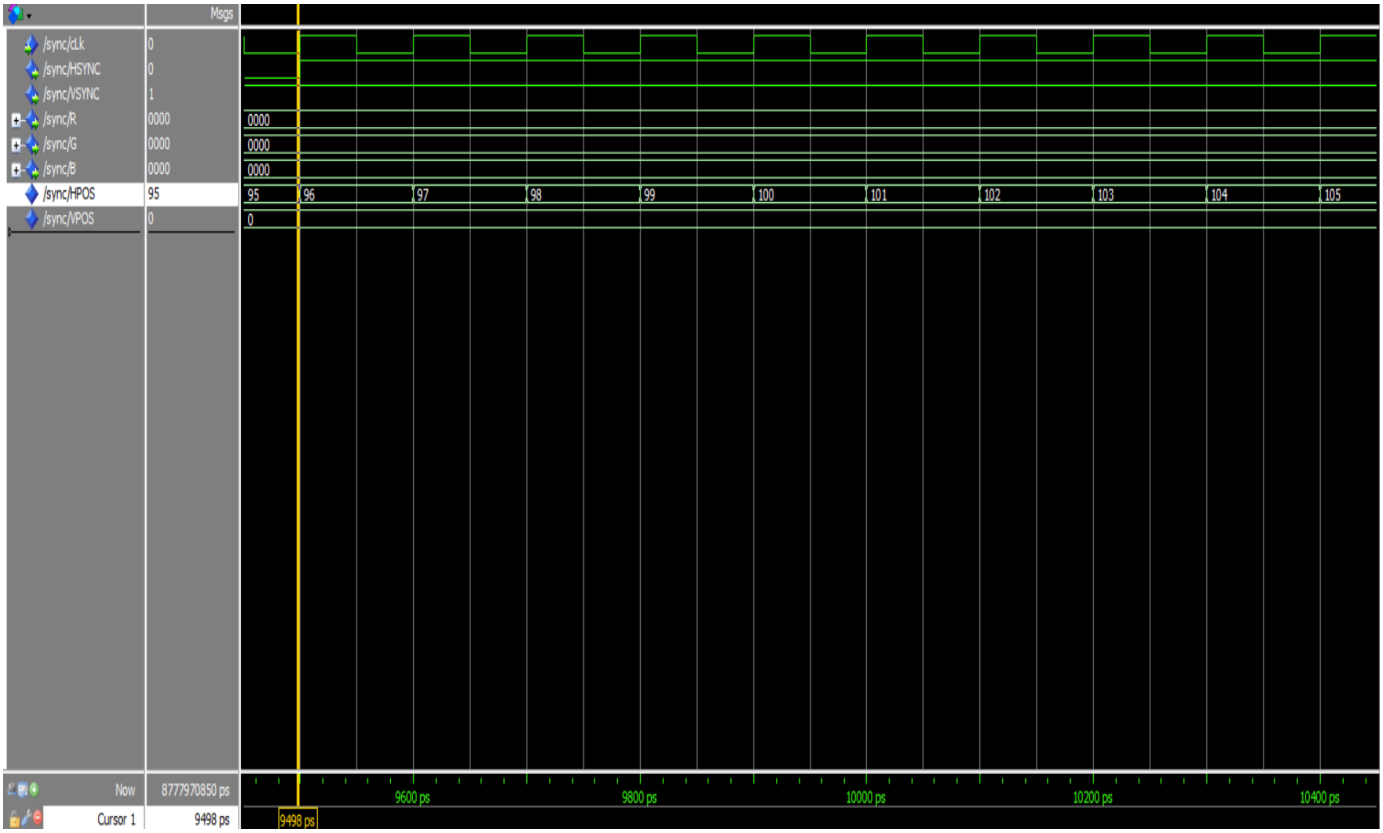
4. Displaying of the Color stripes

VGA 640x480 @60Hz

	Horizontal Timing			Vertical Timing	
Fréquence pixel = 25 MHz	Time (µs)	Pixels		Time (ms)	Lines
a : pulse synchro	3.8	95		0.063	2
b : back porch	1.76	44		1.048	33
c : display interval	25.6	640		15.245	480
d : front porch	0.6	15		0.318	10
Total ligne/trame	31.76	794		16.674	525

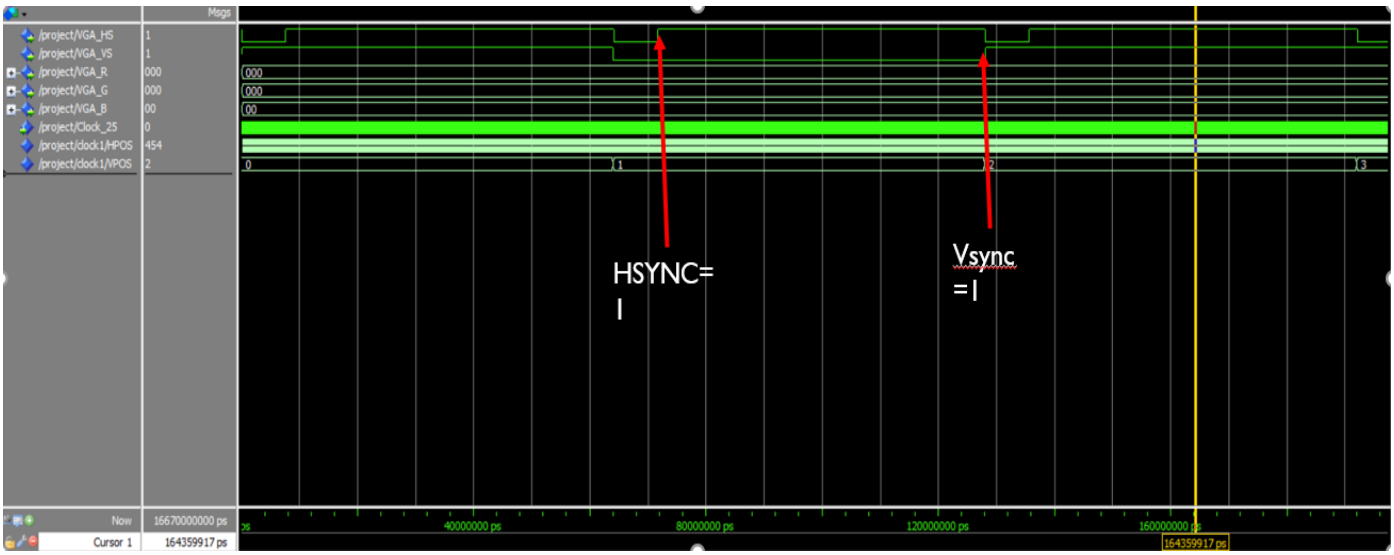
4.1 simulation in MODELSIM

The four colors were produced in the Modelsim by splitting the screen into 4 equal horizontal timing. Horizontal timing values for pulse synchro, back porch, and front porch will be changed in order to get 800 pixels. First, the pulse synchronization for horizontal timing will be 96 (as shown in the below figure), while the back porch will be modified as 46 and the front porch will be changed to 18. This produce VGA of 800 *525 pixels, every pixel is is 0.04 µs. Time for the whole Frame is 16.67 µs (800*525). A counter Vpos is used to count the vertical lines and Hpos counter to count the pixels in every line. The counters are used in code for determining the limits while displaying the image.



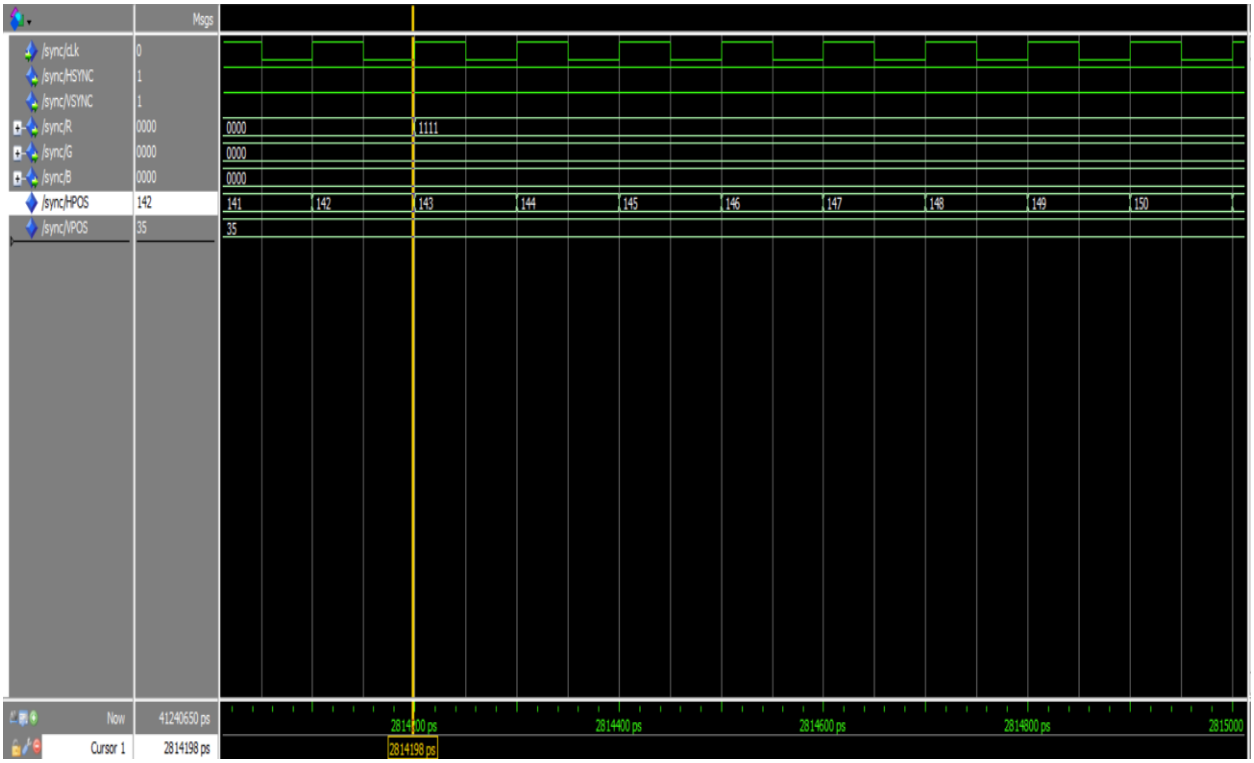
Synchro horizontal timing figure 4.1.1

Horizontal synchronization is shown together with Vertical in below figure 4.1.2



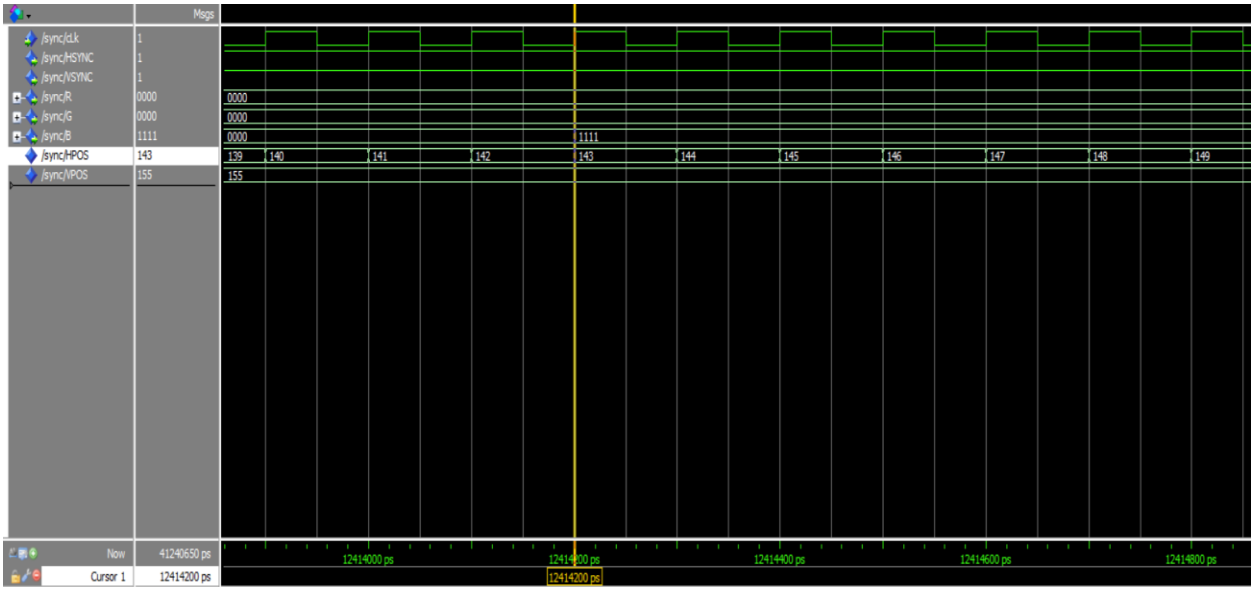
Synchro horizontal and vertical timing figure 4.1.2

The four colors are split on the 480 vertical lines, so Each color will account for 120*640 pixels. The first color(**RED**) appears after the horizontal timing reached this eqn **synchro+ back porch =96+46=142** and vertical timing reached this eqn **synchro+ back porch= 2+33=35**.These values are shown in the below graph 4.1.3



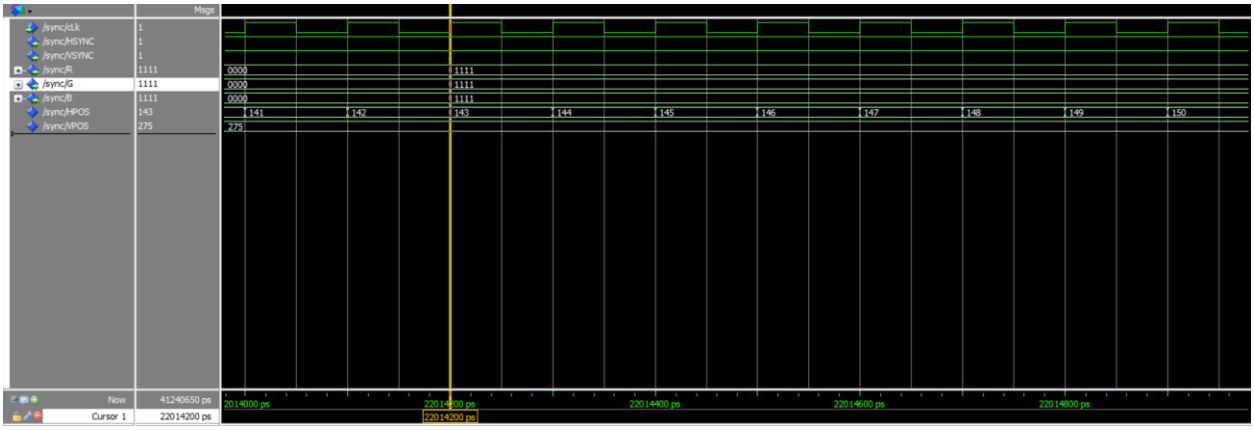
Red Figure 4.1.3

The second color(**BLUE**) appears after the horizontal timing reached this eqn **synchro+ back porch =96+46=142** and vertical timing reached this eqn **synchro+ back porch= 2+33+120=155**. These values are shown in the below graph 4.1.4



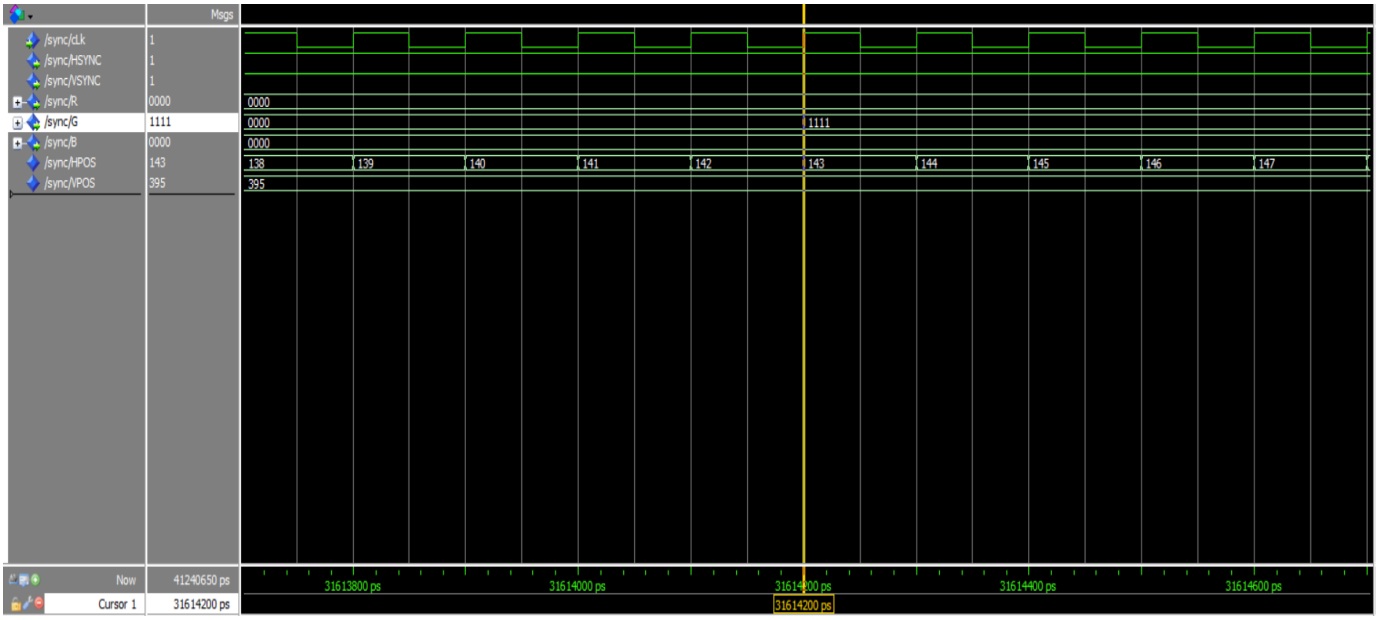
Blue Figure 4.1.4

The third color color(**white**) appears after the horizontal timing reached this eqn synchro+ back porch =96+46=142 and vertical timing reached this eqn **synchro+ back porch= 2+33+120+120=275**. These values are shown in the below graph 4.1.5



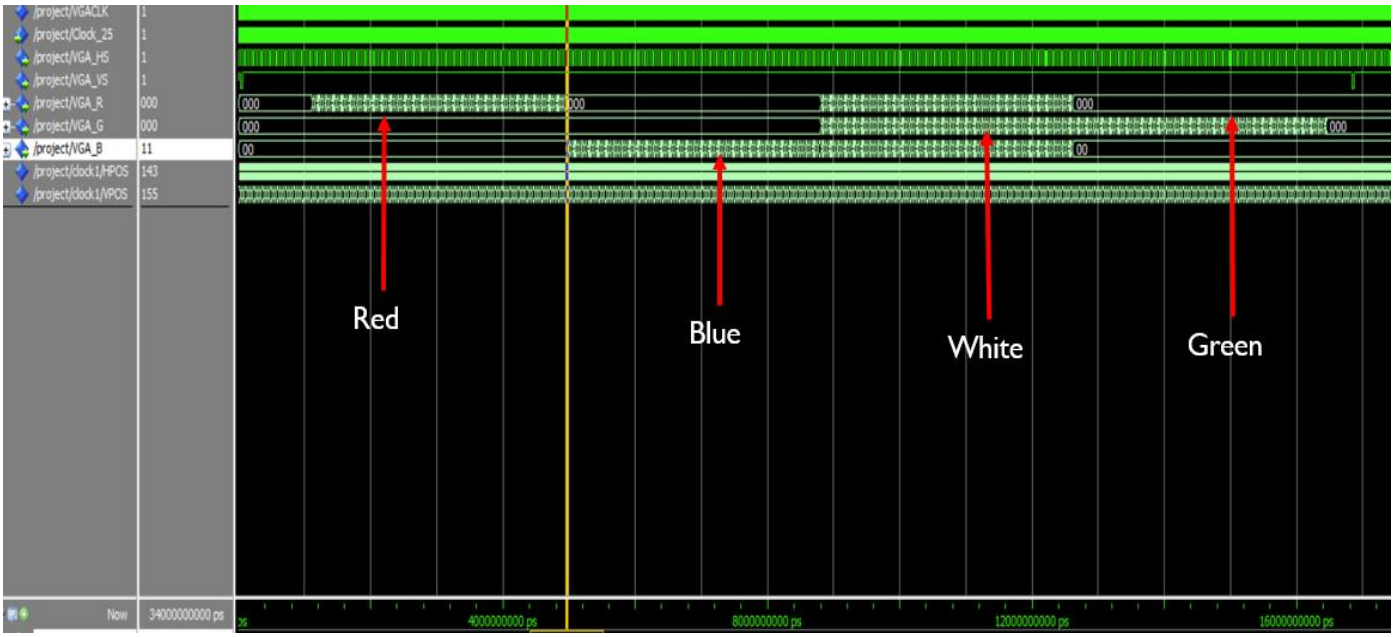
White Figure 4.1.5

The fourth color color(**green**) appears after the horizontal timing reached this eqn synchro+ back porch =96+46=142 and vertical timing reached this eqn **synchro+ back porch= 2+33+120+120+120=395.** These values are shown in the below graph 4.1.6



Green Figure 4.1.6

All the colors are shown in one frame cycle in below figure 4.1.7.



RED, BLUE, WHITE and GREEN color simulation result Figure 4.1.7

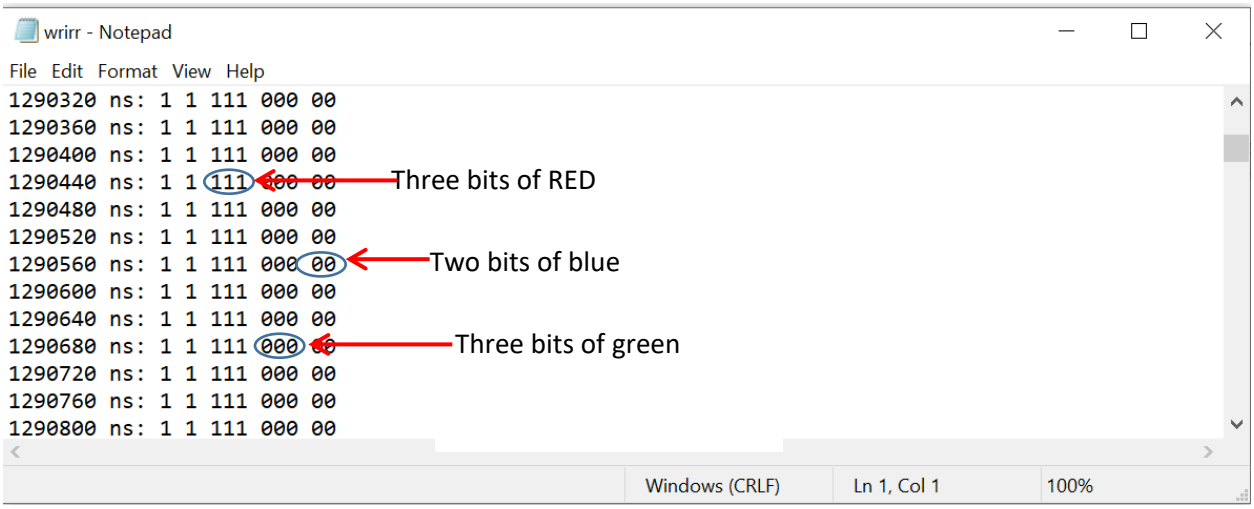
4.2 Simulation on online Simulator

The RGWB stripes were displayed on an online VGA simulator. The LOG file of the VHDL code is generated using the Textio library. The log file is uploaded on the online simulator for simulating.

The log files consist of the time and the respective 10 bit data for that time instance. The 10 bit data is as follows.

online simulator data structure

Parameters	HSYNC	VSYNC	3 bits of RED			3bits of GREEN			2 bits of BLUE	
Bit no	9	8	7	6	5	4	3	2	1	0



Screen shot of log file for printing the color stripes. Figure 4.2.1

Online Simulation of color Stripes on VGA simulator Figure 4.2.2

5. Displaying the Earth revolution around the Moon.

5.1 Creating ROM memory

The 12 bit RGB format image of Moon and Earth from the PC was uploaded into the 2 separate ROM block memory of FPGA board by using the **'MegaWizard Plug-in Manager'** tool of **QUARTUS.** The ROM block has input as clock pulse, memory address and gives output as a 12 bit RGB data. The output data structure is as follows.

	ROM memory output data structure											
Parameters	4 bits of BLUE				4 bits of GREEN				4 bits of RED			
ROM data Bit no	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

The online simulator used works on 8bit RGB format hence we converted the 12bit RGB to 8 bit RGB as follows:

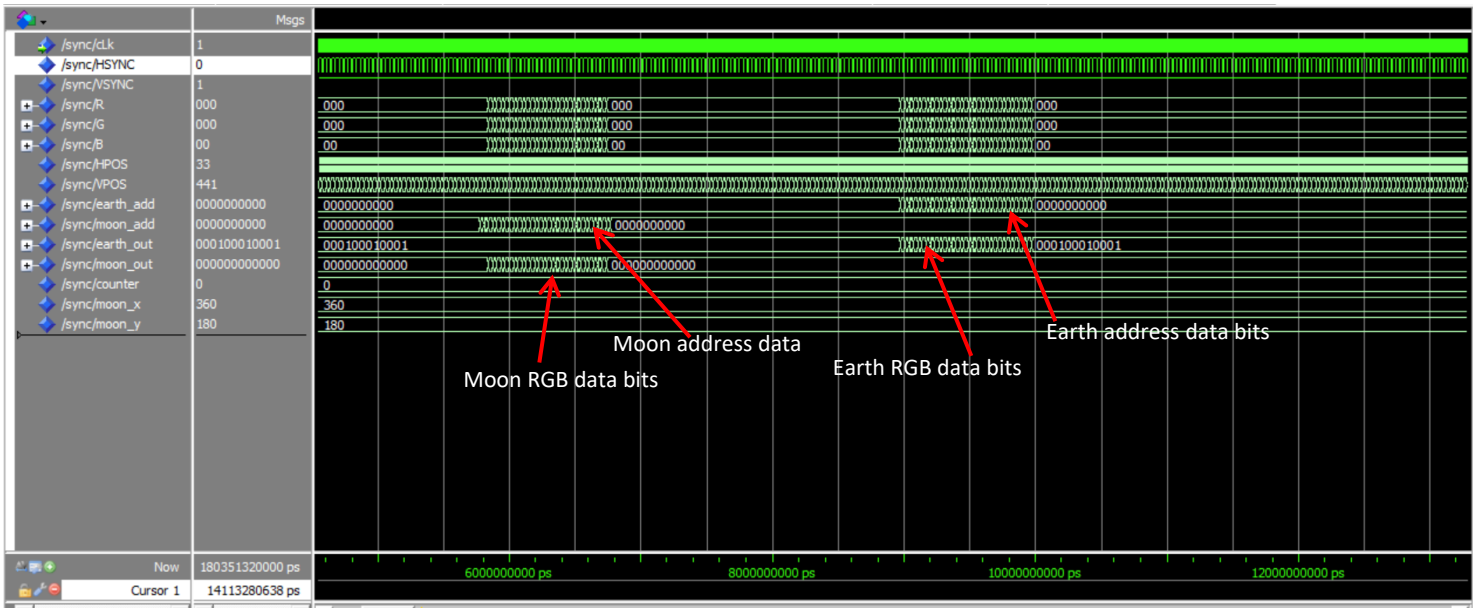
12 bit to 8 bit converted RGB data structure for online simulator								
Parameters	3 bits of RED			3bits of GREEN			2bits of BLUE	
Bit no	b2	b1	b0	b6	b5	b4	b9	b8

final 10 bit data structure for online simulator										
Parameters	HSYNC	VYSNC	3 bits of RED			3bits of GREEN			2bits of BLUE	
data bits	HYSNC	VYSNC	b2	b1	b0	b6	b5	b4	b9	b8
bit no	9	8	7	6	5	4	3	2	1	0

5.2 simulation on MODELSIM

VGA read the picture of the earth and the moon from the ROM. Colors of RGB were equated to Moon and Earth color as shown in the Below Graph.

One frame is of 16.67ms so we simulated for 16 frames i.e the moon revolves around the earth with a shift of 22.5 degrees then the previous position so in total 22.5*16=360 degrees. At the end of 16th frame the moon completes one revolution around earth.

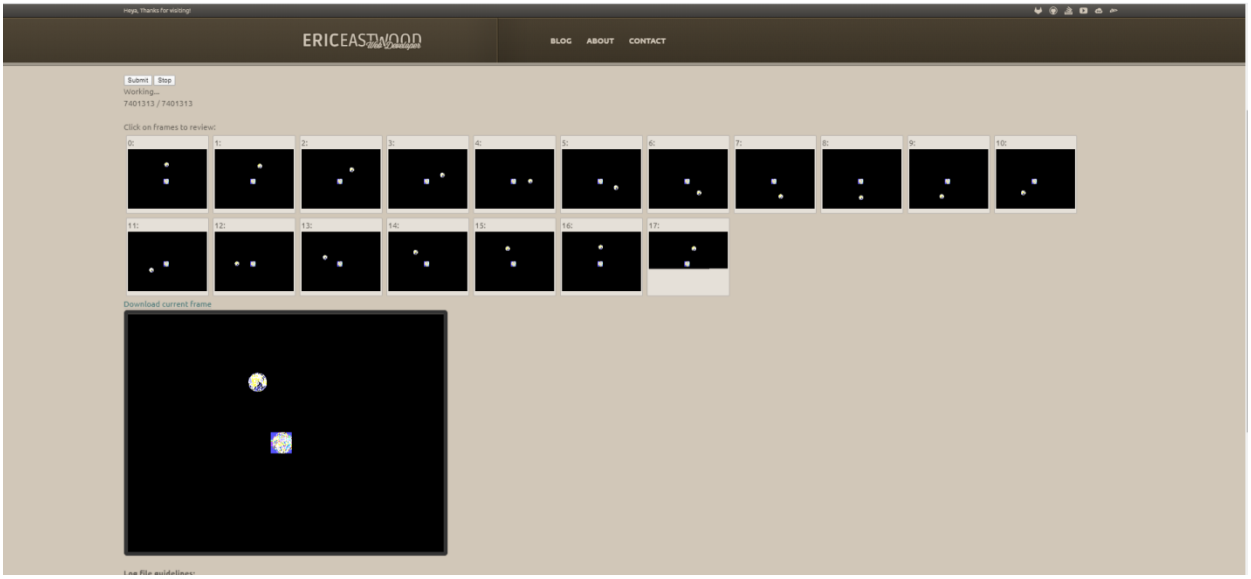


Result of VGA simulation for Earth and the moon for single frame Figure 5.2.1

Simulation displaying 5 frames figure 5.2.2

5.3 simulation on online simulator

The moon was coded to rotate around earth every time cycle as in real. The moon turns 22.5 degrees every frame cycle. The fig 5.3.1 shows the online simulation of earth revolution around moon.



Online Simulation of Moon rotating around Earth Figure 5.3.1

Code for Earth revolution around the Moon:

Sync file:

```
library IEEE;
library STD;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_logic_unsigned.all;
use ieee.std_logic_textio.all;

use std.textio.all;

Entity sync is
port(
cLk: IN STD_LOGIC
--B:OUT STD_LOGIC_vector(3 downto 0)
);
END sync;

ARCHITECTURE MAIN of sync is
signal HSYNC, VSYNC :  STD_LOGIC;
signal R,G:  STD_LOGIC_vector(2 downto 0);
signal B:  STD_LOGIC_vector(1 downto 0);
SIGNAL HPOS : INTEGER RANGE 0 TO 799:=0;
SIGNAL VPOS : INTEGER RANGE 0 TO 525:=0;
SIGNAL earth_add, moon_add: std_logic_vector(9 downto 0) := "0000000000";
SIGNAL earth_out, moon_out: std_logic_vector(11 downto 0);
constant earth_x: INTEGER RANGE 0 TO 799:=360;
constant earth_y: INTEGER RANGE 0 TO 525:=280;
signal counter: INTEGER:=0;
signal moon_x: INTEGER RANGE 0 TO 799:=360;
signal moon_y: INTEGER RANGE 0 TO 525:=180;

-----
Component results is
PORT
        (
```

```
		address		: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		q		: OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
	);
END component;
-----
Component result1 is
PORT
	(
		address		: IN STD_LOGIC_VECTOR (9 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		q		: OUT STD_LOGIC_VECTOR (11 DOWNTO 0)
	);
END component;
-----
BEGIN

R1: results port map(earth_add, clk, earth_out);
R2: result1 port map(moon_add, clk, moon_out);
PROCESS(clk)
BEGIN
 if( clk ' EVENT AND clk ='1') THEN
 if (counter =0)then
		moon_x<=360;
		moon_y<=180;
	end if;
	if (VPOS=524) THEN
			counter<= counter+1;
	end if;

	if (HPOS<799) THEN
		HPOS<= HPOS+1;
	ELSE
		HPOS<=0;
		if (VPOS<524) THEN
```

```
                VPOS<= VPOS+1;
        ELSE
                VPOS<=0;
        END IF;
    END IF;

    IF(HPOS>0 AND HPOS<95) THEN
        HSYNC<='0';
    ELSE
        HSYNC<='1';
    END IF;

    IF(VPOS>0 AND VPOS<2) THEN
        VSYNC<='0';
    ELSE
        VSYNC<='1';
    END IF;

    IF((HPOS>0 AND HPOS<141) OR(VPOS>0 AND VPOS<34)or (HPOS>781 AND
HPOS<799) OR(VPOS>514 AND VPOS<524)) THEN
        R<=(OTHERS=>'0');
        B<=(OTHERS=>'0');
        G<=(OTHERS=>'0');
    END IF;

    if(counter=1)  then
        moon_x<=399;
        moon_y<=188;
    End if;
    if(counter=2)  then
        moon_x<=430;
        moon_y<=210;
    End if;
    if(counter=3)  then
        moon_x<=453;
```

```
        moon_y<=242;
End if;
if(counter=4)  then
        moon_x<=460;
        moon_y<=280;
End if;
if(counter=5) then
        moon_x<=453;
        moon_y<=318;
End if;
if(counter=6)  then
        moon_x<=430;
        moon_y<=350;
End if;
if(counter=7)  then
        moon_x<=399;
        moon_y<=372;
End if;
if(counter=8)  then
        moon_x<=360;
        moon_y<=380;
End if;
if(counter=9)  then
        moon_x<=322;
        moon_y<=372;
End if;
if(counter=10) then
        moon_x<=290;
        moon_y<=350;
End if;
if(counter=11)  then
        moon_x<=268;
        moon_y<=318;
End if;
if(counter=12)  then
```

```
            moon_x<=260;
            moon_y<=280;
End if;
if(counter=13)  then
            moon_x<=268;
            moon_y<=241;
End if;
if(counter=14)  then
            moon_x<=290;
            moon_y<=210;
End if;
if(counter=15)  then
            moon_x<=322;
            moon_y<=188;
End if;
if (counter=16) then
counter<=0;
end if;
                        R<=(OTHERS=>'0');
                        B<=(OTHERS=>'0');
                        G<=(OTHERS=>'0');
if((VPOS >= earth_y) and (VPOS < earth_y+32)) then
            if((HPOS >= earth_x) and (HPOS < earth_x+32)) then
                        earth_add <= earth_add+1;
                        R<=earth_out(2 downto 0);
                        G<=earth_out(6 downto 4);
                        B<=earth_out(9 downto 8);
            end if;
else
            earth_add <= (OTHERS=>'0');
end if;
if((VPOS >= moon_y) and (VPOS < moon_y+32)) then
            if((HPOS >= moon_x) and (HPOS < moon_x+32)) then
                        moon_add <= moon_add+1;
                        R<=moon_out(2 downto 0);
```

```
                    G<=moon_out(6 downto 4);
                    B<=moon_out(9 downto 8);
                end if;
            else
                moon_add <= (OTHERS=>'0');
            end if;
END IF;
end process;
process (clk)
   file file_pointer: text is out "wrirr.txt";
   variable line_el: line;
begin

   if rising_edge(clk) then

      -- Write the time
      write(line_el, now); -- write the line.
      WRITE(line_el, string(":")); -- write the line.

      -- Write the hsync
      write(line_el, string(" "));
      write(line_el, (HSYNC)); -- write the line.

      -- Write the vsync
      write(line_el, string(" "));
      write(line_el, (VSYNC)); -- write the line.

      -- Write the red
      write(line_el, string(" "));
      write(line_el, ( R)); -- write the line.

      -- Write the green
      write(line_el, string(" "));
      write(line_el, (G)); -- write the line.
```

```
        -- Write the blue
        write(line_el, string(" "));
        write(line_el, (B)); -- write the line.
        --write(line_el, string(" "));
        --write(line_el, (yellow)); -- write the line.

        writeline(file_pointer, line_el); -- write the contents into the file.

    end if;
end process;
End main;
```

Project file:

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std;
use ieee.std_logic_textio.all;
use std.textio.all;
Entity project is
port(
Clock_25:IN STD_Logic;
VGA_HS,VGA_VS: out STD_LOGIC;
VGA_R,VGA_G, VGA_B: OUT STd_LOGIC_vector( 3 downto 0)
--VGA_B: OUT STd_LOGIC_vector( 1 downto 0)
);
END project;

Architecture MAIN OF project is

SIGNAL VGACLK: STD_LOGIC :='0';
--SIGNAL RESET: std_LOGIC:='0';
--SIGNAL Red,Green: STD_LOGIC_vector(2 downto 0);
--SIGNAL Blue: STD_LOGIC_vector(1 downto 0);
--SIGNAL hsync, vsync : STD_LOGIC;
--constant colon: STRING := ":";
```

```
--component PLL is
--        port (
--            clk_in_clk  : in  std_logic := 'X'; -- clk
--            reset_reset : in  std_logic := 'X'; -- reset
--            clk_out_clk : out std_logic         -- clk
--        );
--    end component PLL;

component sync is
port(
clk: IN STD_LOGIC;
HSYNC, VSYNC : OUT STD_LOGIC;
R,G,B: OUT STD_LOGIC_vector(3 downto 0)
--B: OUT STD_LOGIC_vector(1 downto 0)
);
END COMPONENT sync;

BEGIN
clock1: sync port map  ( VGACLK,VGA_HS,VGA_VS, VGA_R,VGA_G, VGA_B);
--clock2: PLL port map (Clock_25(0),RESET,VGACLK);
--clock3: sync port map(VGACLK, hsync, vsync, Red, Green,Blue);
process(Clock_25)
begin
if rising_edge(Clock_25) then
          VGACLK <= not VGACLK;
end if;
end process;

END MAIN;
```