**Lab Report 1**

**VGA**

**Team Members:-**

**Mina Ibrahim Henien 49-1743**

**Ali Samy Aboelkhair 52-10440**

**This report includes:**

* **Implementation of 640x480 VGA control using VHDL and basys 3 board**
* **Display a color on a screen based on user input using basys 3 board and VGA**
* **Display a fixed image on the screen using the block ram of the basys 3 board and VGA**

**Introduction for VGA Control:-**

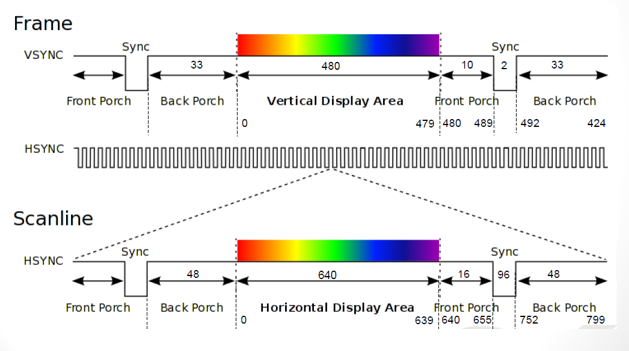
VGA, known as Video Graphics Array, originally emerged alongside the IBM® PS/2 computer in 1987, defining the standards for analog computer display adopted by VESA® and characterized by the DE-15 Connector, colloquially known as the VGA connector. Its significance has transcended mere hardware, encompassing the iconic resolution of 640×480 and becoming emblematic of visual computing. While CRT-based VGA displays employed modulated electron beams to illuminate screens coated with phosphor, LCD displays revolutionized the paradigm by leveraging a matrix of switches to control the passage of light through liquid crystal at the pixel level. In contemporary contexts, VGA displays exhibit remarkable versatility, accommodating various resolutions to cater to diverse application needs. Integral to this adaptability is the VGA controller circuit within FPGAs, tasked with orchestrating the determination of display resolution by generating precise timing signals to regulate the displayed patterns. These signals, emitted at 3.3V or 5V, synchronize the flow of current through deflection coils, ensuring coherence in electron beam movement. Moreover, the controller meticulously coordinates the delivery of video data to the electron guns, aligning it with the timing signals. The concept of raster video displays remains fundamental to VGA technology, delineating the "rows" and "columns" comprising the display area. The number of rows, typically ranging from 240 to 1200, and columns, spanning 320 to 1600, directly influence the size of each pixel and the overall display dimensions. At the heart of VGA image display in FPGAs lies the VGA controller circuit, responsible for generating Horizontal Sync (HS) and Vertical Sync (VS) timing signals and managing the delivery of video data in concert with the pixel clock. The pixel clock governs the time allocated for displaying each pixel, while the VS signal dictates the refresh frequency, indicating how frequently the entire display undergoes refreshment. Refresh frequencies, typically ranging from 50Hz to 120Hz, are contingent upon factors such as phosphor properties and electron beam intensity. Conversely, the horizontal retrace frequency is determined by the number of lines displayed at a given refresh frequency, embodying the intricacies of VGA image display in FPGA environments.

**Task 1:-**

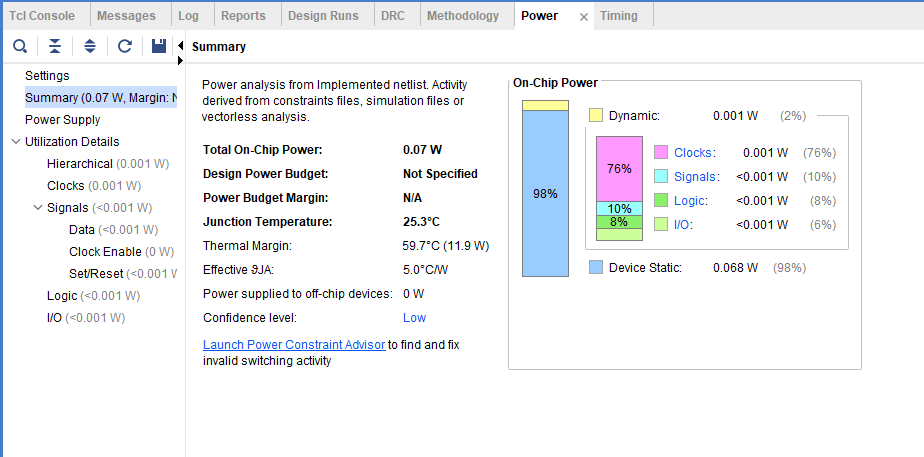
**Methodology:-**

* Controlling the LCD screen by changing the screen colour based on the user selection through 3 switches inputs (8 colour combinations)

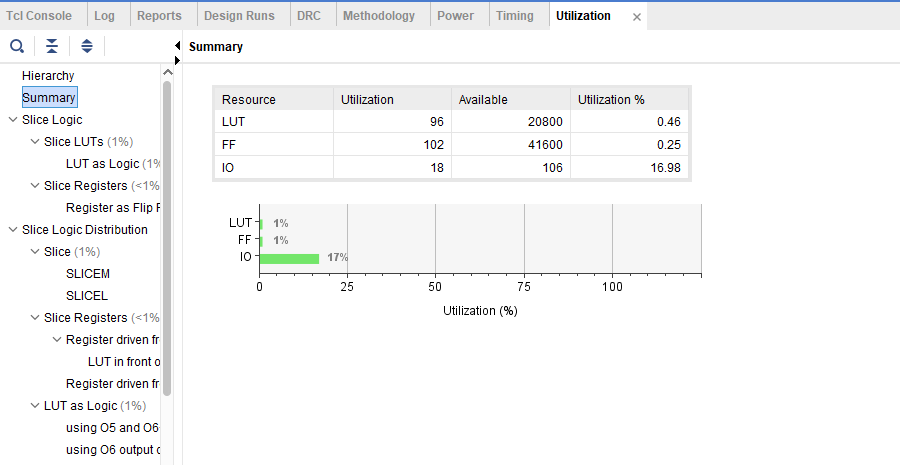
**VGA Timing implemented in the code:-**



**Power Utilization:-**



**Area Utilization:-**



**Implementation:-**

<https://github.com/AliAboelkhair/VGA/blob/main/VGA_Display_Colors.vhd>

**Task 2:-**

**Methodology:-**

* Program the FPGA board using VHDL to display a fixed image on the screen
* We used python to convert the image to RGB binary values
* We uploaded the .coe file to the block ram of the basys 3 board
* We have displayed only 300x300 pixels because the storage of the basys 3 block ram is not enough to store 640x480 pixels

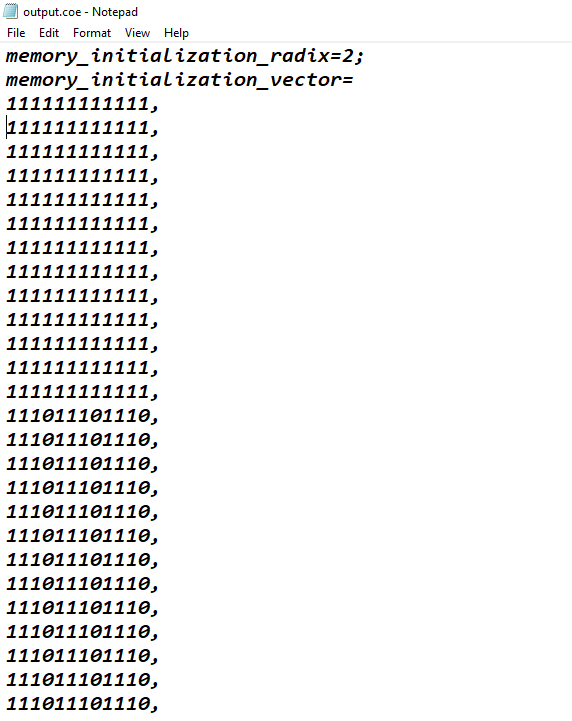
This is the image we used:-



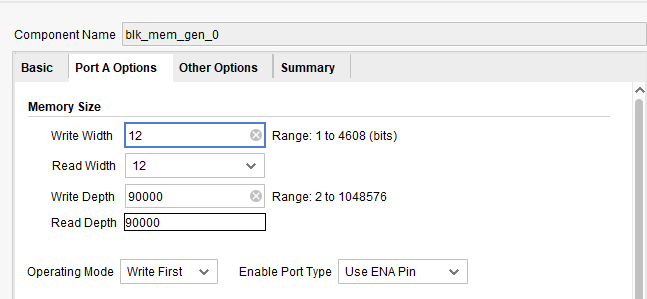
**Python code:-**

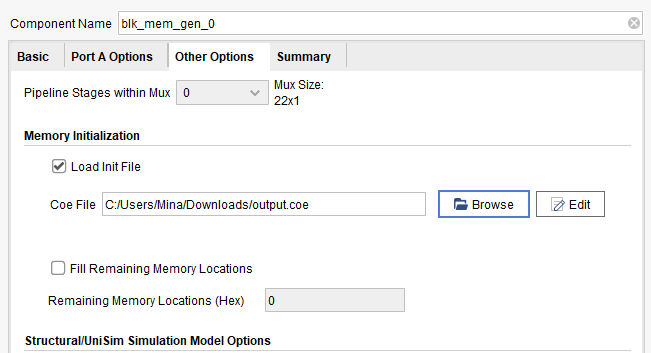
<https://github.com/AliAboelkhair/VGA/blob/main/COE%20Generation.py>  
  
<https://github.com/AliAboelkhair/VGA/blob/main/COE%20Convertor%20and%20comma.py>

**Part of the .coe file generated from python:-**

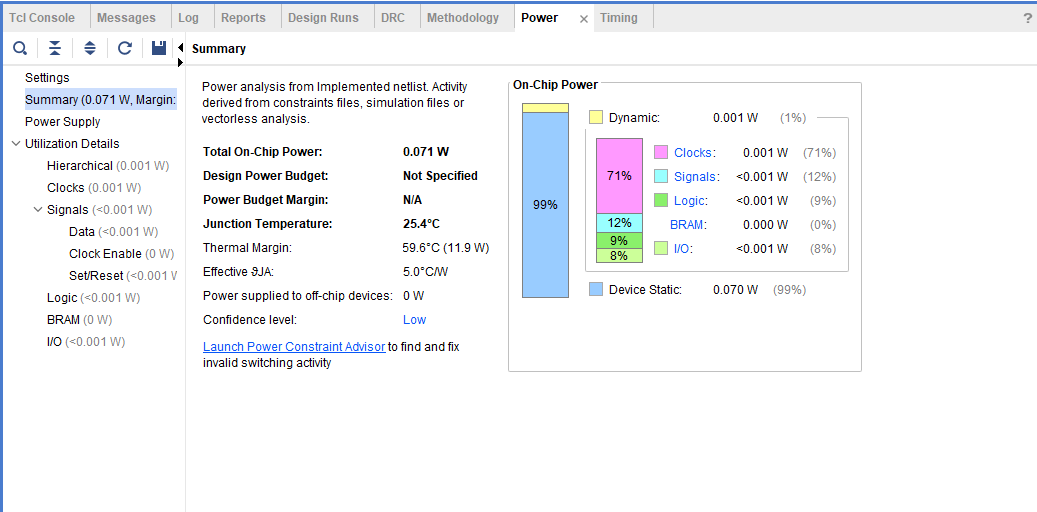


**Block Ram configuration:-**

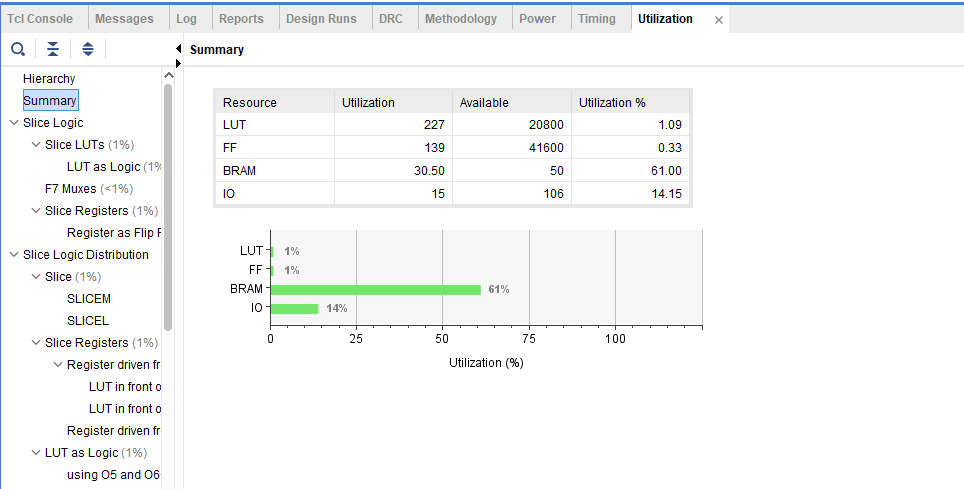




**Power Utilization:-**



**Area Utilization:-**



**Output:-**



**Implementation:-**

[**https://github.com/AliAboelkhair/VGA/blob/main/VGA\_Display\_Image.vhd**](https://github.com/AliAboelkhair/VGA/blob/main/VGA_Display_Image.vhd)

**Conclusion:-**

Throughout our project, we successfully used VHDL to control a VGA 640x480 display, displaying a fixed image. We used clock and reset inputs to create the timing signals needed for the VGA display. By setting the right RGB values for each pixel, we made sure the image showed up correctly on the VGA screen. But, it wasn't all smooth sailing. We faced challenges that needed careful fixing and testing. For example, making sure the timing signals were generated at the right times was tricky. Also, getting the RGB values just right for each pixel was tough. Plus, matching the VGA signal levels to display colours accurately was important.

**Appendix:-**

**Task 1 VHDL code:-**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity VGA\_Display is

Port (

clk : in std\_logic;

R\_in : in std\_logic;

G\_in : in std\_logic;

B\_in : in std\_logic;

Hsync : out std\_logic := '1';

Vsync : out std\_logic := '1';

R\_out : out std\_logic\_vector(3 downto 0);

G\_out : out std\_logic\_vector(3 downto 0);

B\_out : out std\_logic\_vector(3 downto 0)

);

end VGA\_Display;

architecture Behavioral of VGA\_Display is

--ClockScaling

shared variable counter : integer := 0;

signal clk25: std\_logic := '0';

--HorizontalValues

constant H\_RES : integer := 640;

constant H\_FP : integer := 16;

constant H\_SYNC : integer := 96;

constant H\_BP : integer := 48;

--VerticalValues

constant V\_RES : integer := 480;

constant V\_FP : integer := 10;

constant V\_SYNC : integer := 2;

constant V\_BP : integer := 33;

begin

--ClockScaling process

clock\_generation : process (clk)

begin

if rising\_edge(clk) then

if counter = 1 then

counter := 0;

clk25 <= not clk25;

else

counter := counter + 1;

end if;

end if;

end process;

process (clk25)

variable HC : integer := 1; --Horizontal Counter

variable VC : integer := 1; --Vertical Counter

begin

if rising\_edge(clk25) then

if HC < H\_RES + H\_FP + H\_SYNC + H\_BP then--HorizontalActiveArea---800

if HC <= H\_RES then--VisibleArea

-----------------------RED---------------------------------------------------------

if (R\_in = '1') then

R\_out <= "1111";

elsif (R\_in = '0') then

R\_out <= "0000";

end if;

-----------------------GREEN-------------------------------------------------------

if (G\_in = '1') then

G\_out <= "1111";

elsif (G\_in = '0') then

G\_out <= "0000";

end if;

-----------------------BLUE--------------------------------------------------------

if (B\_in = '1') then

B\_out <= "1111";

elsif (B\_in = '0') then

B\_out <= "0000";

end if;

else

R\_out <= "0000";

G\_out <= "0000";

B\_out <= "0000";

end if;

-----------------------RESET------------------------------------------------------

if HC > H\_RES + H\_FP and HC < H\_RES + H\_FP + H\_SYNC then --Hsync only 0 at sync pulse region

Hsync <= '0';

else

Hsync <= '1';

end if;

HC := HC + 1;

-----------------------IncrementVC------------------------------------------------

else

HC := 1;

if (VC < V\_RES + V\_FP + V\_SYNC + V\_BP) then--VerticalActiveArea

if VC > V\_RES + V\_FP and VC < V\_RES + V\_FP + V\_SYNC then --Vsync only 0 at sync pulse region

Vsync <= '0';

else

Vsync <= '1';

end if;

VC := VC + 1;

else--ResetVC

VC := 1;

end if;

end if; --HorizontalActiveArea

end if; -- risingEdge

end process;

end Behavioral;

**Task 2 Python code to convert RGB image into binary format:-**

from PIL import Image

# Load the image

image = Image.open("AMD.png")

# Resize the image to 300x300

image = image.resize((300, 300))

# Convert image to RGB mode (if not already in RGB)

image = image.convert("RGB")

# Get the pixel data

pixel\_data = list(image.getdata())

# Function to convert RGB values to 12-bit representation

def convert\_to\_12\_bit(pixel):

# Extract RGB values

r, g, b = pixel

# Convert each channel to 4-bit representation

r\_4bit = (r >> 4) & 0b1111

g\_4bit = (g >> 4) & 0b1111

b\_4bit = (b >> 4) & 0b1111

# Combine channels into a single 12-bit representation

return (r\_4bit << 8) | (g\_4bit << 4) | b\_4bit

# Convert each pixel to 12-bit representation

pixels\_12\_bit = [convert\_to\_12\_bit(pixel) for pixel in pixel\_data]

# Open a text file in write mode

with open("output.txt", "w") as file:

# Write each pixel value in binary format to the file

for pixel in pixels\_12\_bit:

# Convert the pixel value to binary format with 12 bits

pixel\_binary = format(pixel, '012b')

# Write the binary representation to the file

file.write(pixel\_binary + '\n')

**Task 2 VHDL code:-**

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.std\_logic\_unsigned.all;

entity VGA\_Display is

Port (

clk : in std\_logic;

Hsync : out std\_logic := '1';

Vsync : out std\_logic := '1';

R\_out : out std\_logic\_vector(3 downto 0);

G\_out : out std\_logic\_vector(3 downto 0);

B\_out : out std\_logic\_vector(3 downto 0)

);

end VGA\_Display;

architecture Behavioral of VGA\_Display is

component blk\_mem\_gen\_0 is

PORT (

clka : IN STD\_LOGIC;

ena : IN STD\_LOGIC;

wea : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0);

addra : IN STD\_LOGIC\_VECTOR(16 DOWNTO 0);

dina : IN STD\_LOGIC\_VECTOR(11 DOWNTO 0);

douta : OUT STD\_LOGIC\_VECTOR(11 DOWNTO 0)

);

end component;

--ClockScaling

shared variable counter : integer := 0;

signal clk25: std\_logic := '0';

--12 Bits RGB Extraction from memory

signal RGB\_sig: std\_logic\_vector(11 downto 0);

--16 bits memory address

signal address : STD\_LOGIC\_VECTOR(16 downto 0) := (others => '0');

--HorizontalValues

constant H\_RES : integer := 640;

constant H\_FP : integer := 16;

constant H\_SYNC : integer := 96;

constant H\_BP : integer := 48;

--VerticalValues

constant V\_RES : integer := 480;

constant V\_FP : integer := 10;

constant V\_SYNC : integer := 2;

constant V\_BP : integer := 33;

begin

clock\_generation : process (clk)--ClockScaling Process

begin

if rising\_edge(clk) then

if counter = 1 then

counter := 0;

clk25 <= not clk25;

else

counter := counter + 1;

end if;

end if;

end process;

Blk\_Ram: blk\_mem\_gen\_0 port map(clk25,'1',"0", address, (others => '0'), RGB\_sig);--Block memory assignment

process (clk25)

variable HC : integer := 1; --Horizontal Counter

variable VC : integer := 1; --Vertical Counter

begin

if rising\_edge(clk25) then

if HC < H\_RES + H\_FP + H\_SYNC + H\_BP then --HorizontalActiveArea---800

if HC <= H\_RES then --VisibleArea

if(HC <= 300 and VC <= 300) then --only display the dimenstions of the image

R\_out <= RGB\_sig(11 downto 8);

G\_out <= RGB\_sig(7 downto 4);

B\_out <= RGB\_sig(3 downto 0);

address <= address + 1;

else

R\_out <= "0000";

G\_out <= "0000";

B\_out <= "0000";

end if;

end if;

-----------------------RESET------------------------------------------------------

if HC > H\_RES + H\_FP and HC < H\_RES + H\_FP + H\_SYNC then --Hsync is only 0 at the sync pulse region

Hsync <= '0';

else

Hsync <= '1';

end if;

HC := HC + 1;

-----------------------IncrementVC------------------------------------------------

else

HC := 1;

if (VC < V\_RES + V\_FP + V\_SYNC + V\_BP) then --VerticalActiveArea

if VC > V\_RES + V\_FP and VC < V\_RES + V\_FP + V\_SYNC then --Vsync is only 0 at the sync pulse region

Vsync <= '0';

else

Vsync <= '1';

end if;

VC := VC + 1;

else --ResetVC

VC := 1;

address <= (others => '0');--resetting the address after each frame

end if;

end if; --HorizontalActiveArea

end if; -- risingEdge

end process;

end Behavioral;