VGA Controller

Students:

Dăian Petre-Mihai

Flanja Tudor

Coordinator: Blaj Ileana

Group: 30414

Year: 2022

Outline

1. Introduction

1.0 Requirement

Using a FPGA board implement a VGA controller. There need to be shown 4 different images , the selection been read from the board buttons. The images must prove the select ability of at least 4 different colors. Also, the position of the images on the screen must be controllable on the 2 axis with the help of buttons. Images suggestions: square, vertical lines, horizontal lines, triangle, circle, etc. The necessary time for different resolutions would be found in table 2-6 of the FPGA XUPV2P_User_Guide.pdf at the pages 37-38. Operating diagrams and others explications can be found in reference manuals of FPGA board.

1.1 Context

The aim of this project is to implement on an FPGA board, specifically basys 3, a VGA controller which is able to display 3 different shapes of 4 different colors on a screen. These 4 shapes will be selected from the basys 3 board with the help of the 2 switches from the board and the color with another 2 switches.

The shape that is shown will be moved on the two axis upward, downward, left and right using buttons placed on the board.

1.2 Specifications

The device will function according to the program created in Vivado Xillinx IDE and the program will run on a basys 3 board. Depending on the shape and on the color selected, the device will compute the coordinates of the pixels that will be shown on the screen. Then image is processed and then displayed using the VGA port of the board.

2. Reference Info

2.0 VGA connector

VGA stands for Video Graphics Array and usually refers to analog computer display standards, the DE-15 Connector (commonly known as VGA connector) or the 640x480 resolution. Shortly, it is used for controlling the analog monitors. The computing side of the interface provides the monitor with horizontal and vertical sync signals, color magnitudes and ground references.

The focus will be on the 5 signals out of 15 pins in this project. These signals are Red, Green, Blue, HS and VS. The first three specify the color of a point on the screen.

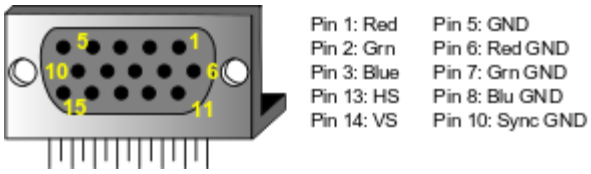

Figure 1

2.1 Signals

The horizontal and vertical sync signals are 0V/5V digital waveforms that synchronize the signal timing with the monitor. Being digital, they provide are provided directly by the Basys 3 board (3.3V is the minimum threshold for a logical high, so 3.3V can be used instead of 5V).

The color magnitudes are 0V-0.7V analog signals sent over the R, G and B wires. The three color magnitude wires are terminated with 75Ω resistors. These lines are also terminated with 75Ω inside the monitor. To crease these analog signals, the Basys 3 board outputs an 8-bit bus for each color to a video DAC converter, which requires a pixel clock to latch in these values.

3. Structure and functionality

3.0 Diagrams

According to the affirmations stated before, the diagram of the project is presented in the figure 2.

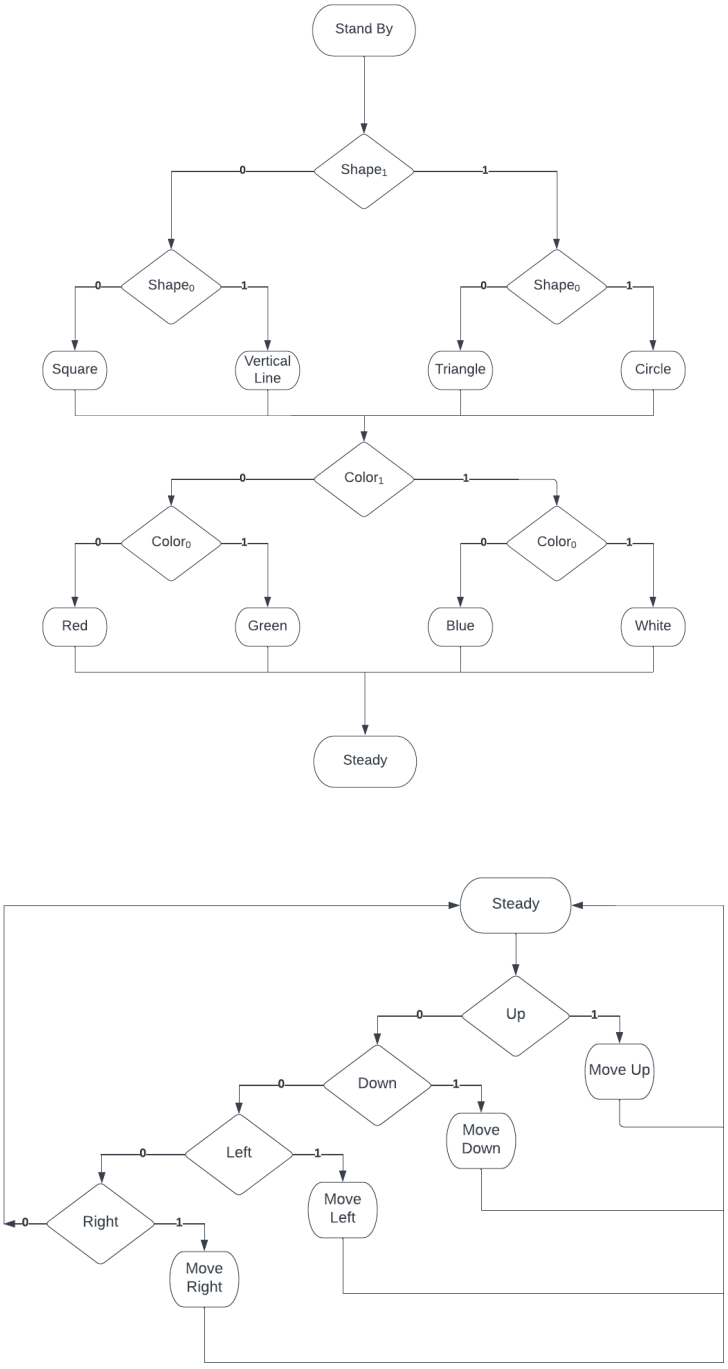

Figure 2

3.1 Timing

The controller must produce synchronizing pulses at 3.3V to set the frequency at which the current flows through the liquid crystals displays, known as LCD (figure 3).

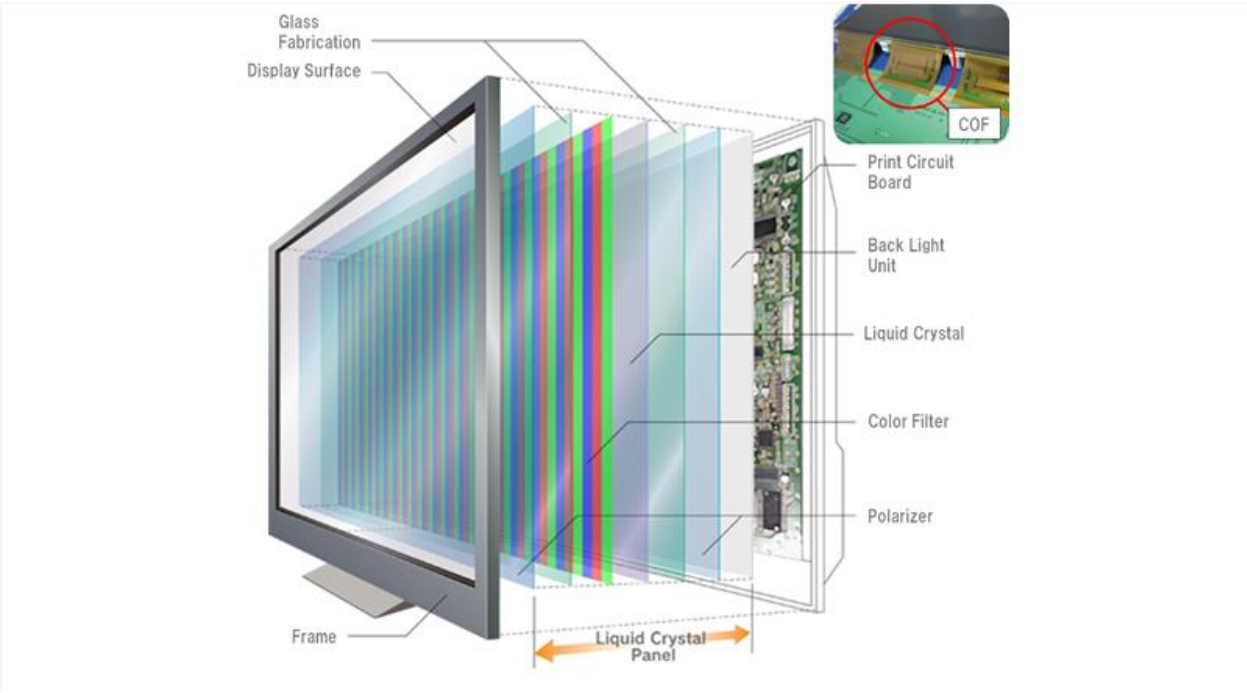

Figure 3

Raster video displays (figure 4) define a number of “rows” that corresponds to the number of horizontal passes the cathode makes over the display area, and a number of “columns” that corresponds to an area on each row that is assigned to one “picture element”, or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns. The overall size of a display and the number of rows and columns determines the size of each pixel. Video data typically comes from a video refresh memory.

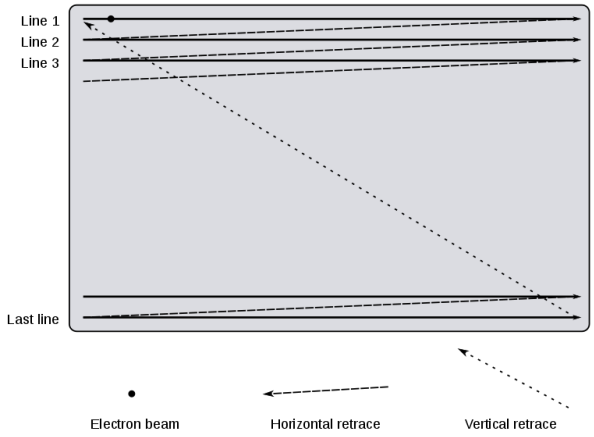

Figure 4

A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the “refresh” frequency of the display, or the frequency at which all information on the display is redrawn. The number of lines to be displayed at a given refresh frequency defines the horizontal “retrace” frequency.

Figure 5 and figure 6 provide the timing specification for 640x480 resolution display at 60Hz frame rate.

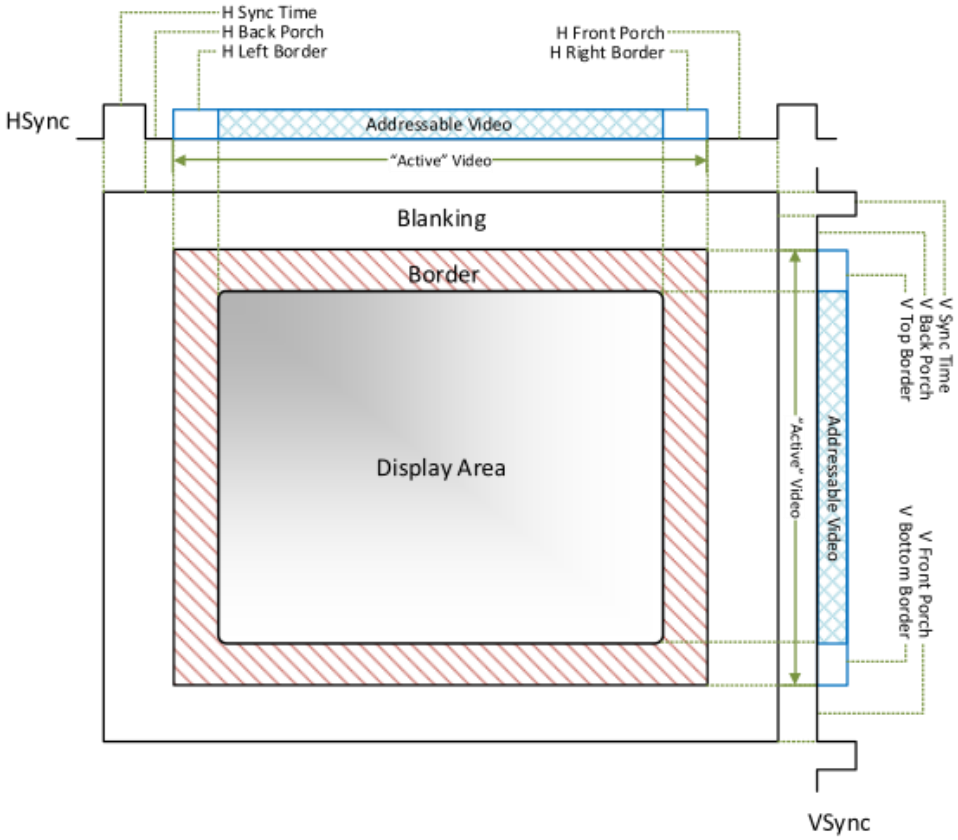

Figure 5

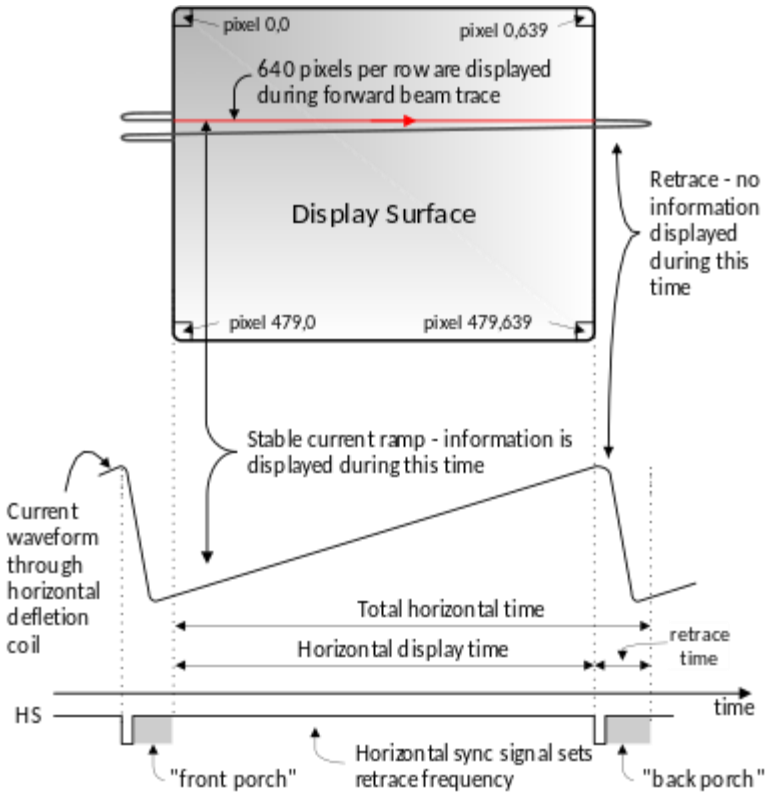

Figure 6

3.2 Block scheme with main components

In order to synchronize the processes, the device needs a clock divider to generate the pixel clock, which provides a timing reference to HS and VS signals. The pixel clock frequency is 25.175MHz in the specification. However, with ±0.5% accuracy, 25MHz can be acceptable as well. And obviously, 25MHz is easy to generate on the Basys 3 board with the clock divider from a frequency of 100Hz using 2 frequency dividers in series.

After this, it will need two counters, a counter (horizontal counter) to count pixels in each line and another counter (vertical counter) to count lines in a frame. The horizontal counter needs to reset itself when it reaches the end of the line (799 in this case) and when it resets itself, it needs to provide a Terminal Count signal to Enable input of the vertical counter so that vertical counter can add 1 when new line begins. Similarly, vertical counter needs to reset itself when it reaches the end of the frame.

Based on the counter values, the device can compare them to the constant defined in the specification to generate HS and VS signals. Note that it has to drive Red, Green and Blue to GND outside the display area. Figure 7 shows the HS and VS generation based on the counter values.

Also, depending on the horizontal and vertical counter, the video comparator states when the image must be shown on the display. A 4:1 multiplexer is needed for color selection. Finally, in function of horizontal and vertical counter, video on signals the image is generated. The color is given by the multiplexer and the position or the movement is given by the up, down, left and right buttons. Now, all the signals for the VGA adaptor are generated, i.e. hsync, vsync, r, g and b.

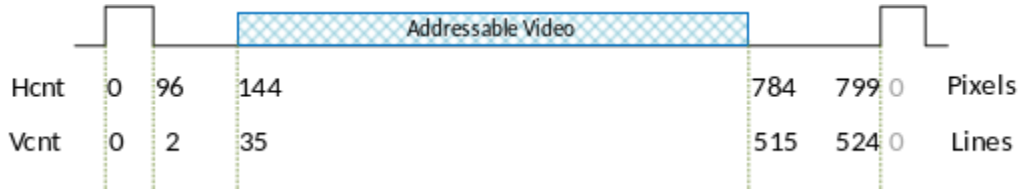

Figure 7

The reference block diagram is provided in figure 8 below

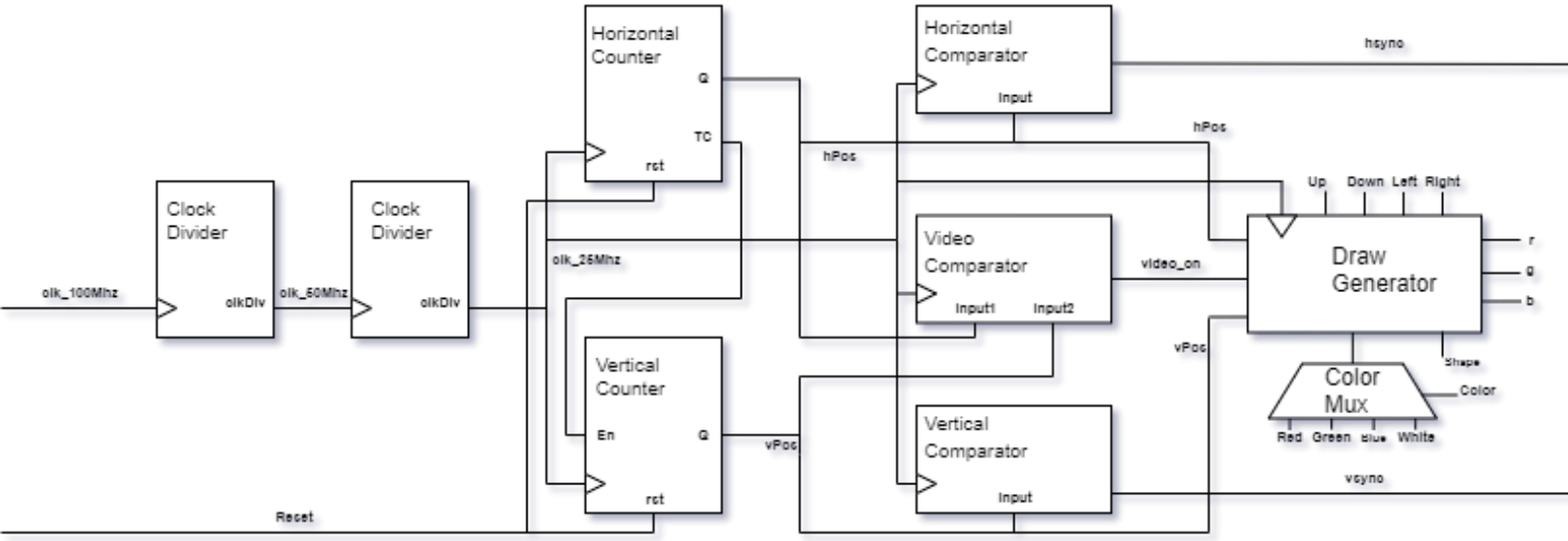

Figure 8

3.3 Black Box

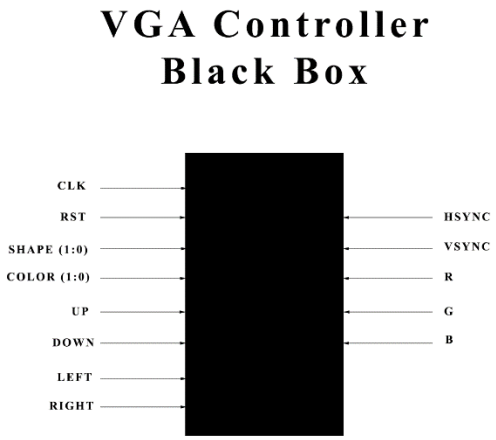

Figure 9

4. List of components

After the previously discussion, the components that this project will have are the following:

- 2 frequency dividers(one form 100Mhz to 50 Mhz and another from 50Mhz to 25Mhz)
- 2 counters (horizontal and vertical counters)
- 4 comparators (HS, VS, video, draw)
- 1 mux 4:1 for color selection

5. Inputs and Outputs

The inputs are:

- Clk – a clock signal based on which all the components work given by the basys3 board and it is equal with 100Mhz
- Up – by pushing the button, the figure moves up on the x axis
- Down - by pushing the button, the figure moves down on the x axis
- Left - by pushing the button, the figure moves left on the y axis
- Right - by pushing the button, the figure moves right on the y axis
- Rst – resets the display
- Shape – 2 switches with which we can select one of the 4 shapes encoded as:
    - Square (“00”)
    - Vertical Line (“01”)
    - Triangle (“10”)
    - Circle (“11”)
- Color - 2 switches with which we can select one of the 4 colors encodes as:
    - Red (“00”)
    - Green (“01”)
    - Blue (“10”)
    - White (“11”)

The outputs are:

- R/ G/ B – output that decide the color of the pixel at each step
- HSync
- VSync

6. Utility and results

This controller is well met at old displays such as monitors or projectors which had a VGA port. In a world dominated by computers or others devices, at that time, the VGA port was the connection between the real world and the digital one. The video devices were and are everywhere in this world, so this technology is an important component of modern digital systems. After some time and improvements, the VGA port was replaced by the HDMI port, more faster and offering more quality images.

The results obtained from this project are as expected. These can be seen in the pictures below.

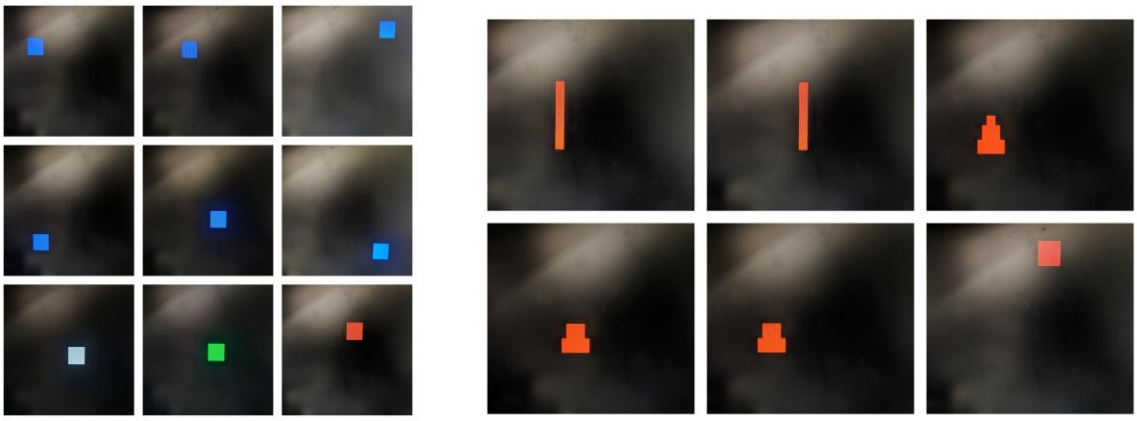

Figure 10

7. Further development

For further development can be used a rom memory to store more complex images or figures, better than actual shapes, and we can transform the whole project in a game that we can be controlled from a board or maybe a keyboard.

We can tweak the signals to get a better resolution and a higher refresh rate so that the things would look more real.

8. More explanations

- Why were used 2 frequency dividers?
  - The Basys 3 board has an internal clock of 100 MHz and the pixel clock in this project that controls the whole process needs 25Mhz and so there were used 2 dividers, one from 100MHz to 50 MHz and another from 50 MHz to 25 MHz.

- Why we need to counters and not one for the whole pixels?
  - It would be hard to use a single counter for all the pixels on the screen. By doing so, we can manage easier the pixels, taking line by line and coloring the pixel in the corresponding manner.
  - The whole problem is divided into subproblems, on the horizontal and vertical parts that are gathered at final the when constructing the shape.
  - The two counters travers step by step each pixel.

- Why do need 4 comparators?
    - The horizontal and the vertical comparators determines the hsync and vsync which generates the signals for timing refresh. Next, the video comparator establish whether the current pixel is in the display area and the draw comparator moves the figure on the screen. All these comparators create the signals that are needed for displaying the figures. Fewer than 4 would be impossible to do this task.

9. Annexes

Here is the code of this project written in VHDL language and Vivado IDE.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_textio.all;
use std.textio.all;
use IEEE.STD_LOGIC_ARITH.ALL;

entity vgaDriver is
    port( clk: in STD_LOGIC;
    rst: in STD_LOGIC;
    hsync: out STD_LOGIC;
    vsync: out STD_LOGIC;
    up: in STD_LOGIC;
    down: in STD_LOGIC;
    left: in STD_LOGIC;
    right: in STD_LOGIC;
    shape: in STD_LOGIC_VECTOR( 1 downto 0);
    color: in STD_LOGIC_VECTOR ( 1 downto 0);
    r: out STD_LOGIC;
    g: out STD_LOGIC;
    b: out STD_LOGIC);
end vgaDriver;

architecture controller of vgaDriver is

signal clk50: std_logic := '0'; -- initializing the clock will be obtain
after division
signal clk25: std_logic := '0';

--horizontal parameters

constant hd: integer := 639; -- horizontal display( 640)
constant hfp: integer := 16; -- right border or front porch
constant hsp: integer := 96; -- sync pulse or retrace
constant hbp: integer := 48; -- left border or back porch
```

```
--vertical parameters

constant vd: integer := 479; -- vertical display(480)
constant vfp: integer := 10; -- right border or front porch
constant vsp: integer := 2; -- sync pulse or retrace
constant vbp: integer := 33; -- left border or back porch

signal hPos: integer := 0;
signal vPos: integer := 0;

signal videoOn: std_logic := '0';

signal R1, G1, B1: std_logic; -- color coordinates buffer

signal x: integer := 320;-- horizontal reference position for the shape
signal y: integer := 240;-- vertical reference position for the shape

constant step: integer := 5;

signal lastUp, lastDown, lastLeft, lastRight: std_logic := '0';

begin

    clkDiv50: process( clk ) -- frequency divider from 100MHz to 50Mhz
    begin
        if ( clk = '1' and clk'event ) then
            clk50 <= not ( clk50 );
        end if;
    end process;

    clkDiv25: process( clk50 ) -- frequency divider from 50MHz to 25Mhz
    begin
        if ( clk50 = '1' and clk50'event ) then
            clk25 <= not ( clk25 );
        end if;
    end process;

    horizontalPositionCounter: process( clk25, rst ) -- the counter for
horizontal traversal
    begin
        if( rst = '1') then
            hPos <= 0;
        elsif(clk25 = '1' and clk25'event ) then
            if( hPos = ( hd + hfp + hsp + hbp ) ) then
                hPos <= 0;
            else hPos <= hPos + 1;
            end if;
        end if;
    end process;

    verticalPositionCounter: process( clk25, rst, hPos ) -- the counter for
vertical traversal

    begin
        if( rst = '1') then
            vPos <= 0;
        elsif(clk25 = '1' and clk25'event ) then
```

```
          if( hPos = ( hd + hfp + hsp + hbp )) then
            if( vPos = ( vd + vfp + vsp + vbp ) ) then
                vPos <= 0;
            else vPos <= vPos + 1;
            end if;
            end if;
        end if;
    end process;

    horizontalSynchronisation: process( clk25, rst, hPos)      -- setting the
horizontal timing
    begin
        if( rst = '1' ) then
            hsync <= '0';
        elsif( clk25 = '1' and clk25'event) then
            if( (hPos <= ( hd + hfp )) or (hPos > ( hd + hfp + hsp ))) then
                hsync <= '1';
            else
                hsync <= '0';
            end if;
        end if;
    end process;

    verticalSynchronisation: process( clk25, rst, vPos)     -- setting the
vertical timing
    begin
        if( rst = '1' ) then
            vsync <= '0';
        elsif( clk25 = '1' and clk25'event )then
            if( (vPos <= ( vd + vfp) ) or ( vPos > ( vd + vfp + vsp ))) then
                vsync <= '1';
            else
                vsync <= '0';
            end if;
        end if;
    end process;

    videoOnSet: process( clk25, rst, hPos, vPos ) -- setting the display area
    begin
        if rst = '1' then
            videoOn <= '0';
        elsif( clk25 = '1' and clk25'event ) then
            if ( hPos <= hd and vPos <= vd ) then
                videoOn <= '1';
            else
                videoOn <= '0';
            end if;
        end if;
    end process;

    colorSet: process( clk25, color ) -- setting the color of the shape
    begin
        if( clk25 = '1' and clk25'event ) then
            case color is
                when "00" =>  -- red
                R1 <= '1';
                G1 <= '0';
```

```
                B1 <= '0';

                when "01" =>  -- green
                R1 <= '0';
                G1 <= '1';
                B1 <= '0';

                when "10" =>  -- blue
                R1 <= '0';
                G1 <= '0';
                B1 <= '1';

                when others =>  -- white
                R1 <= '1';
                G1 <= '1';
                B1 <= '1';
            end case;
        end if;
    end process;


    move: process( clk25, up, down, left, right, x, y )  --- moving the shape
    begin
        if( clk25 = '1' and clk25'event ) then
            if( up = '1' and lastUp = '0' ) then
                x <= x - step;
            elsif( down = '1' and lastDown = '0' ) then
                x <= x + step;
            elsif( left = '1' and lastLeft = '0' ) then
                y <= y - step;
            elsif( right = '1' and lastRight = '0' ) then
                y <= y + step;
            end if;
            lastUp <= up;
            lastDown <= down;
            lastLeft <= left;
            lastRight <= right;
        end if;
    end process;

   draw: process( clk25, rst, hPos, vPos, videoOn, x, y )
   begin
    if rst = '1' then
        r <= '0';
        g <= '0';
        b <= '0';
    elsif( clk25 = '1' and clk25'event ) then
        if videoOn = '1' then -- we are in the display area
            case shape is

                when "00" =>  -- square
                if( ( hPos >= 10 + x and hPos <=60 + x) and (vPos>=10 + y and
vPos<=60 + y)) then
                    r <= R1;
                    g <= G1;
                    b <= B1;
                else
```

```
                    r <= '0';
                    g <= '0';
                    b <= '0';
                end if;

                when "01" => -- line
                if( (hPos > x - 10) and (hpos < x + 10) and (vPos > y - 70)
and (vPos < y + 70)) then
                    r <= R1;
                    g <= G1;
                    b <= B1;
                else
                    r <= '0';
                    g <= '0';
                    b <= '0';
                end if;

                when "10" => -- triangle
                if  ((hPos > y - 5) and (hpos < y + 5) and (vPos > x - 30)
and (vPos < x + 10)) or
                    ((hPos > y - 20) and (hpos < y + 20) and (vPos > x - 50)
and (vPos < x + 10)) or
                    ((hPos > y - 30) and (hpos < y + 30) and (vPos > x - 20)
and (vPos < x + 10)) then
                    r <= R1;
                    g <= G1;
                    b <= B1;
                else
                    r <= '0';
                    g <= '0';
                    b <= '0';
                end if;

                when others => -- circle
                if  ((hPos > y - 10) and (hpos < y + 10) and (vPos > x - 70)
and (vPos < x + 10)) or
                    ((hPos > y - 20) and (hpos < y + 20) and (vPos > x - 50)
and (vPos < x + 10)) or
                    ((hPos > y - 30) and (hpos < y + 30) and (vPos > x -20 )
and (vPos < x + 10)) then
                    r <= R1;
                    g <= G1;
                    b <= B1;
                else
                    r <= '0';
                    g <= '0';
                    b <= '0';
                end if;

            end case;
        end if;
    end if;

   end process;


end controller;
```