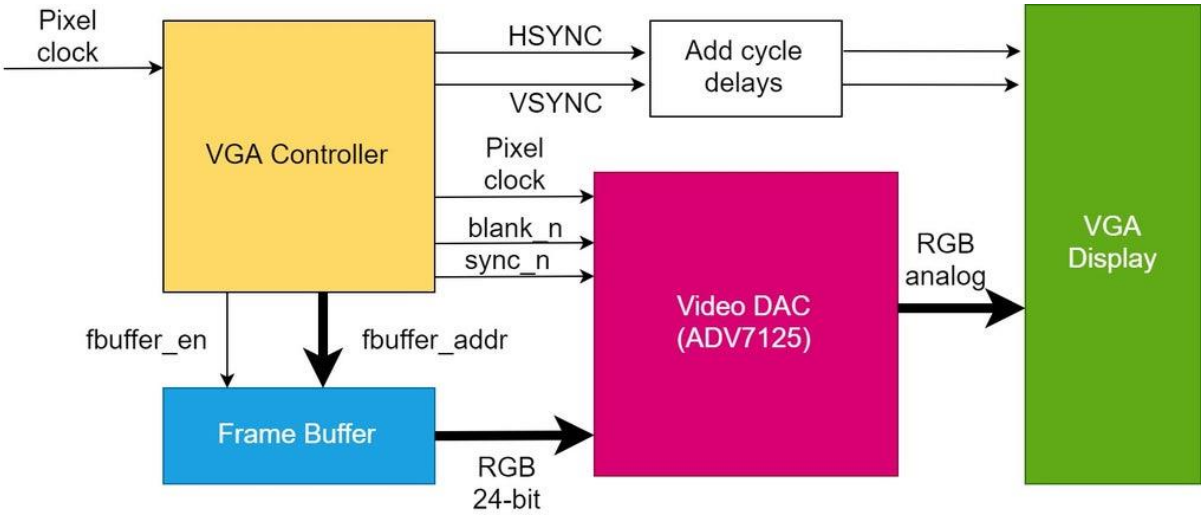
VGA Controller



GIURGIU RAZVAN

GROUP : 30432

Content :

1. Introduction

1.1. Context

The goal of this project is to implement on a FPGA board, a VGA controller which is able to display 4 different images. The images must be selected from the FPGA board using switches and they also need to show the ability to change colors according to the user's selection.

Images\Figures that are displayed can be moved on two axis, UP\DOWN and RIGHT\LEFT using buttons placed on the board.

1.2. Specifications

The device will be programmed, using ISE Design Suite, on a Basys2 board. It will be able to display one of the four programmed figures and move them using one of the four buttons on the Basys2 Board. Image is processed and then dispalyed using the VGA port.

2. Bibliographic study

VGA is a standard interface for controlling analog monitors. The computing side of the interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references.

The horizontal and vertical sync signals are 0V/5V digital waveforms that synchronize the signal timing with the monitor. Being digital, they are provided directly by the FPGA (3.3V meets 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. (Alternatively, the green wire can use 0.3V-1V signals that incorporate both the horizontal and vertical sync signals, eliminating the need for those lines. This is called sync-on-green and is not addressed here.) The three color magnitude wires are terminated with 75Ω resistors. These lines are also terminated with 75Ω inside the monitor. To create these analog signals, the FPGA outputs an 8-bit bus

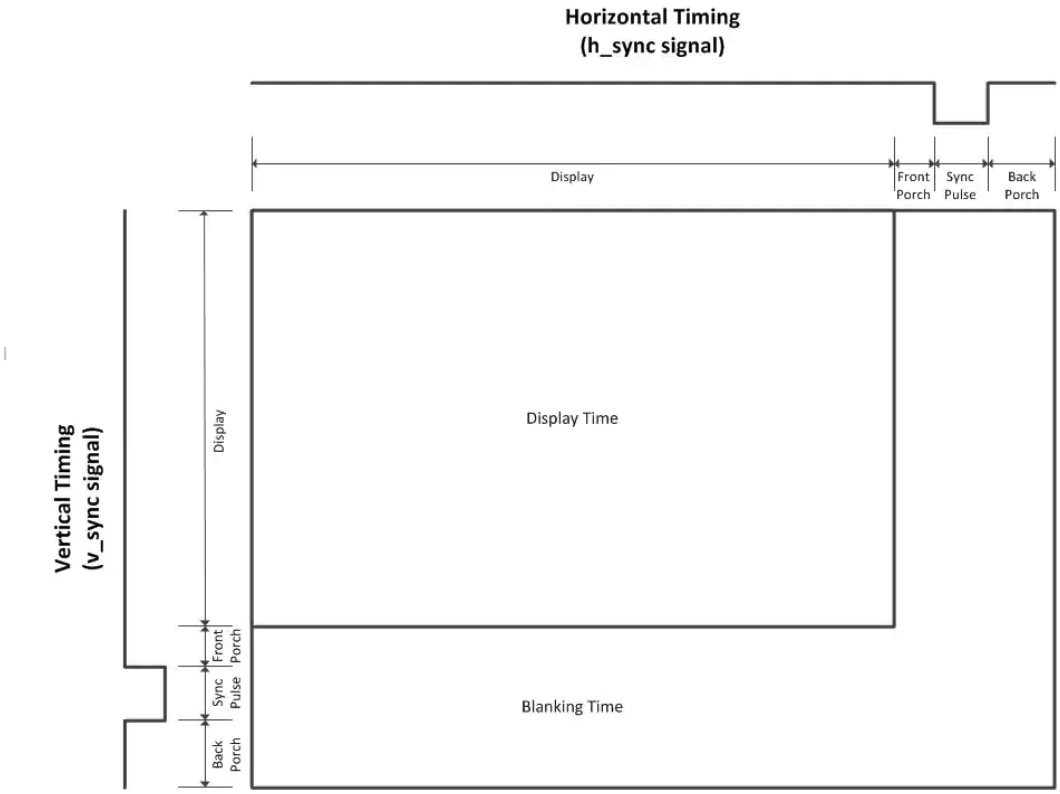
for each color to a video DAC, in this example an ADV7123 29 from Analog Devices. This video DAC also requires a pixel clock to latch in these values.

3.1. Timing

The figure illustrates the timing signals produced by the VGA controller. The controller contains two counters. One counter increments on pixel clocks and controls the timing of the h_sync (horizontal sync) signal. By setting it up such that the display time starts at counter value 0, the counter value equals the pixel's column coordinate during the display time. The horizontal display time is followed by a blanking time, which includes a horizontal front porch, the horizontal sync pulse itself, and the horizontal back porch, each of specified duration. At the end of the row, the counter resets to start the next row.

The other counter increments as each row completes, therefore controlling the timing of the v_sync (vertical sync) signal. Again, this is set up such that the display time starts at counter value 0, so the counter value equals the pixel's row coordinate during the display time. As before, the vertical display time is followed by a blanking time, with its corresponding front porch, sync pulse, and back porch. Once the vertical blanking time completes, the counter resets to begin the next screen refresh.

Using these counters, the VGA controller outputs the horizontal sync, vertical sync, display enable, and pixel coordinate signals. The sync pulses are specified as positive or negative polarity for each VGA mode.



Horizontal timing (line)

Scanline part	Pixels
Visible area	640
Front porch	16
Sync pulse	96
Back porch	48
Whole line	800

Vertical timing (frame)

Frame part	Lines
Visible area	480
Front porch	10
Sync pulse	2
Back porch	33
Whole frame	525

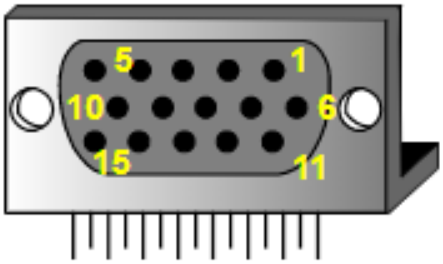
General timing

Screen refresh rate	60 Hz
Vertical refresh	31.46875 kHz
Pixel freq.	25.175 MHz

3.2. VGA Connector (DE – 15)

VGA stands for Video Graphics Array. Initially, it refers specifically to the display hardware first introduced with IBM® PS/2 computer in 1987. With the widespread adoption, it now usually refers to the analog computer display standards the DE-15 Connector (commonly known as VGA connector).

We will only concentrate on the 5 signals out of 15 pins in this project. These signals are Red, Grn, Blue, HS, and VS. Red, Grn, and Blue are three analog signals that specify the color of a point on the screen

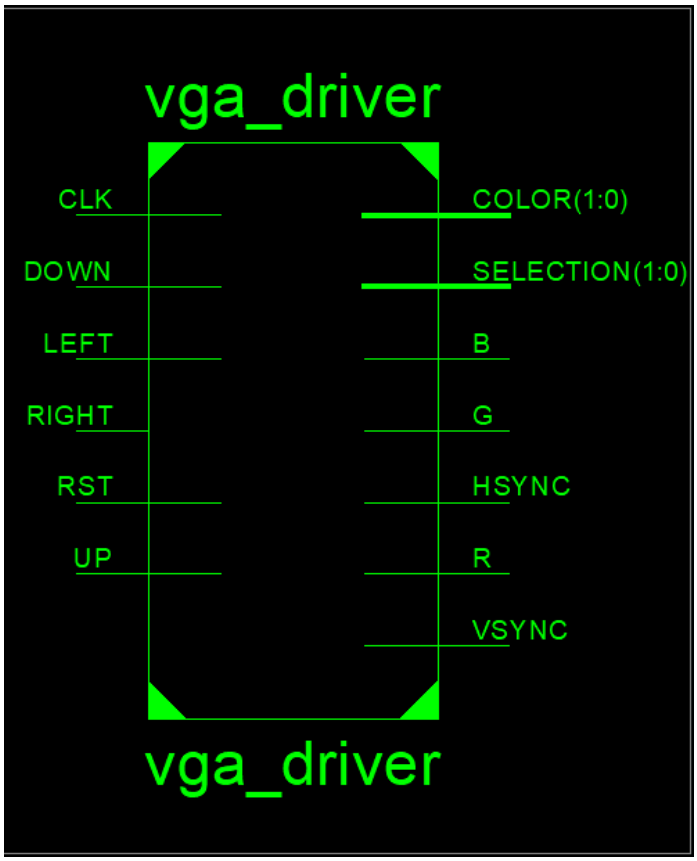


Pin 1: Red	Pin 5: GND
Pin 2: Grn	Pin 6: Red GND
Pin 3: Blue	Pin 7: Grn GND
Pin 13: HS	Pin 8: Blu GND
Pin 14: VS	Pin 10: Sync GND

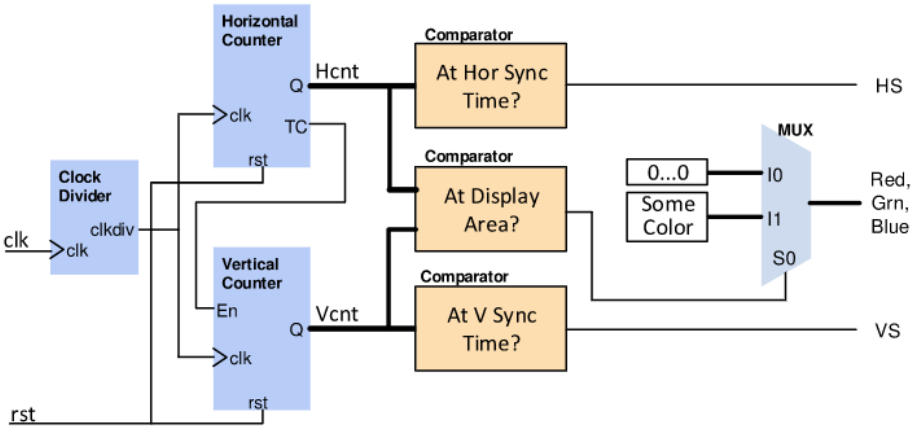
Figure 1. VGA connector.

3.3. Black box

Using Xilinix ISE Design Suite after the synthesis has been done we can see the Black Box that is going to be implemented :



3.4. Block scheme



The block scheme may seem complicate but it is very simple. First we are are using a clock divider because we are havig a 50 mHz Clock signal and we need to work with a 25 mHz Clock signal. We start the sync process to generate our horizontal sync (HSYNC) and our vertical sync (VSYNC) as described above and we use a signal (videoOn) to keep track of the moments when we can draw something on the screen or not.

In the drawing process we are moving our initial points from which the drawing of our figure started up/down or left/rigt in a procedure name „Position”. At any particular moment we can press the reset button and our screen will become black.

Next, if we are allowed to draw, based on our selection we will draw one of 4 predefined figures. We can choose from a sqaure, lines, a triangle, and a spehere

4. List of components and code

For this project we used the following components :

- Frequency divider
- 4 Counters

- Comparators
- Multiplexers

• Frequency divider :

used to transform a 50 mHz Clock to a 25 mHz Clock

```
clk_div:process(CLK)
begin
    if(CLK'event and CLK = '1')then
        clk25 <= not clk25;
    end if;
end process;
```

• Counters used for HSYNC and VSYNC

Used to generate the lines and the frames

```
procedure Horizontal_position_counter (signal clk25 : in std_logic;
begin
if(clk25'event and clk25 = '1')then
    if (hPos = (HD + HFP + HSP + HBP)) then
        hPos <= 0;
    else
        hPos <= hPos + 1;
    end if;
end if;
end procedure;
```

```
procedure Vertical_position_counter  (signal clk25 : in
begin
if(clk25'event and clk25 = '1')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 procedure;
```

• Comparators

Used to check if the screen is in blinking time or we can draw

```
procedure Horizontal_Synchronisation  (signal clk25 : in std_logic;
begin
if(clk25'event and clk25 = '1')then
    if((hPos <= (HD + HFP)) OR (hPos > HD + HFP + HSP))then
        HSYNC <= '1';
    else
        HSYNC <= '0';
    end if;
end if;
end procedure;
```

```
procedure Vertical_Synchronisation   (signal clk25 : in std_logic,
begin
if(clk25'event and clk25 = '1')then
    if((vPos <= (VD + VFP)) OR (vPos > VD + VFP + VSP))then
        VSYNC <= '1';
    else
        VSYNC <= '0';
    end if;
end if;
end procedure;
```

```
procedure video_on(signal clk25 : in std_logic ; s
begin
if (clk25'event and clk25 = '1') then
    if(hPos <= HD and vPos <= VD)then
        videoOn <= '1';
    else
        videoOn <= '0';
    end if;
end if;
end procedure;
```

Next, the drawing process begins, we are changing the position of the figure acording or reset it according to the user input

```
drawing : process(clk25, RST, hPos, vPos, videoOn, UP, DOWN, LEFT, RIGHT, startH, startV)
begin
position (clk25, UP, DOWN, LEFT, RIGHT,startV,startH);
if(RST = '1') then
    R<='0';
    G<='0';
    B<='0';
```

If the reset is not on we start drawing the figure that we want based on SELECTION

```
 elsif(clk25'event and clk25 = '1')then
    if(videoOn = '1') then
        if(SELECTION = "00") then
            square(hPos, vPos, startH, startV,R,G,B,COLOR);
        elsif (SELECTION = "01") then
            lines(hPos, vPos, startH, startV, R, G, B, COLOR);
        elsif (SELECTION = "10") then
            ball(hPos, vPos, startH, startV, R, G, B, COLOR);
        elsif (SELECTION = "11") then
            triangle(hPos, vPos, startH, startV, R, G, B, COLOR);
        end if;
    end if;
end if;
end process;
```

The procedure that is used for movement, check at each step if we need to move the figure on one of the 2 axis

```
procedure  position (signal clk25 : in std_logic; signal UP, DOWN, LEFT, RIGHT
begin

if(UP'event and UP = '1') then
    x <= x - 5 ;
elsif (DOWN'event and DOWN = '1') then
    x <= x + 5 ;
elsif (LEFT'event and LEFT = '1') then
    y <= y - 5 ;
elsif(RIGHT'event and RIGHT = '1') then
    y <= y + 5 ;
end if;
end procedure;
```

We are using procedure to draw each figure, an example of how the drawing is done is showd here :

```
procedure square( signal hPos, vPos, hSet, vSet : in integer; signal R,G,B : out std_logic; signal COLOR
    begin
        if((hPos > hSet - 50) and (hpos <hSet + 50) and (vPos > vSet - 50) and (vPos < vSet + 50)) then
            if(COLOR = "00")  then
                R<='1';
                G<='0';
                B<='0';
         elsif(COLOR = "01")  then
                R<='0';
                G<='1';
                B<='0';
        elsif(COLOR = "10")  then
                R<='0';
                G<='0';
                B<='1';
        elsif(COLOR = "11")  then
                R<='1';
                G<='1';
                B<='1';
        end if;
         else
           R<='0';
           G<='0';
           B<='0';
        end if;
    end procedure square;
end sq;
```

5. Inputs / Outputs

- Inputs :
    - CLK
    - UP
    - DOWN
    - LEFT
    - RIGHT
    - RST
    - SELECTION
    - COLOR

- Outputs :
    - R
    - G
    - B
    - HSYNC
    - VSYNC

Description of each input and output :

CLK - a clock signal based on which all the components work

UP/DOWN – we move the figure by pushing the buttons assigned on an x axis up and down

RIGHT/ LEFT – we move the figure by pushing the buttons assigned on an y axis left and right

SELECTION – we have 4 figures from which we can select, encoded as

- SQUARE (”00”)
- LINES (”01”)

- SPHERE (”10”)
- TRIANGLE (”11”)

COLOR – we can select from 4 different colors encoded as

- RED (”00”)
- GREEN (”01”)
- BLUE (”10”)
- WHITE (”11”)

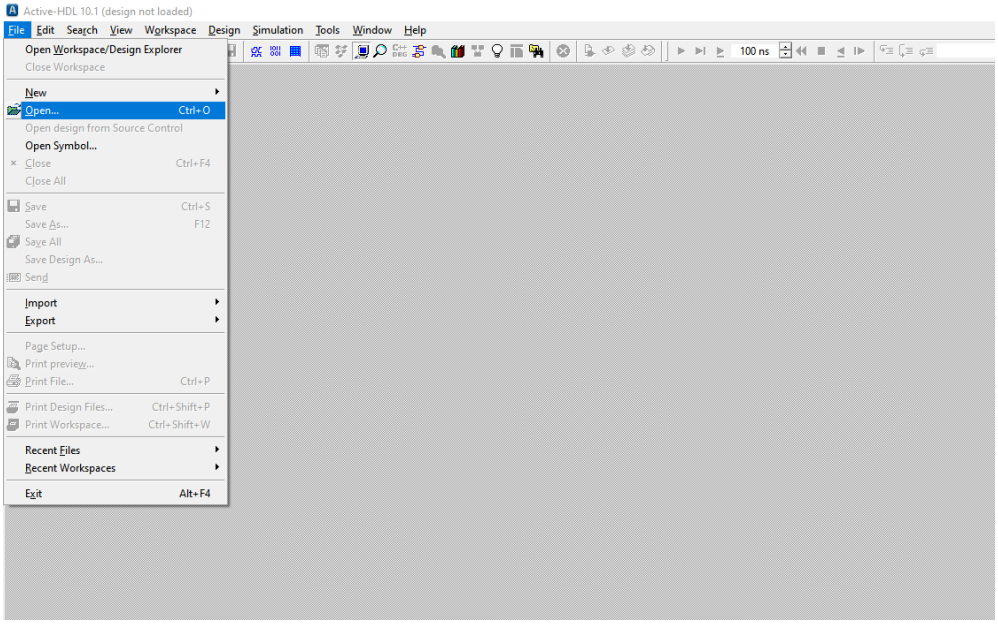
R/G/B – outputs that decide the color of the pixel at each step (RED, GREEN, BLUE)

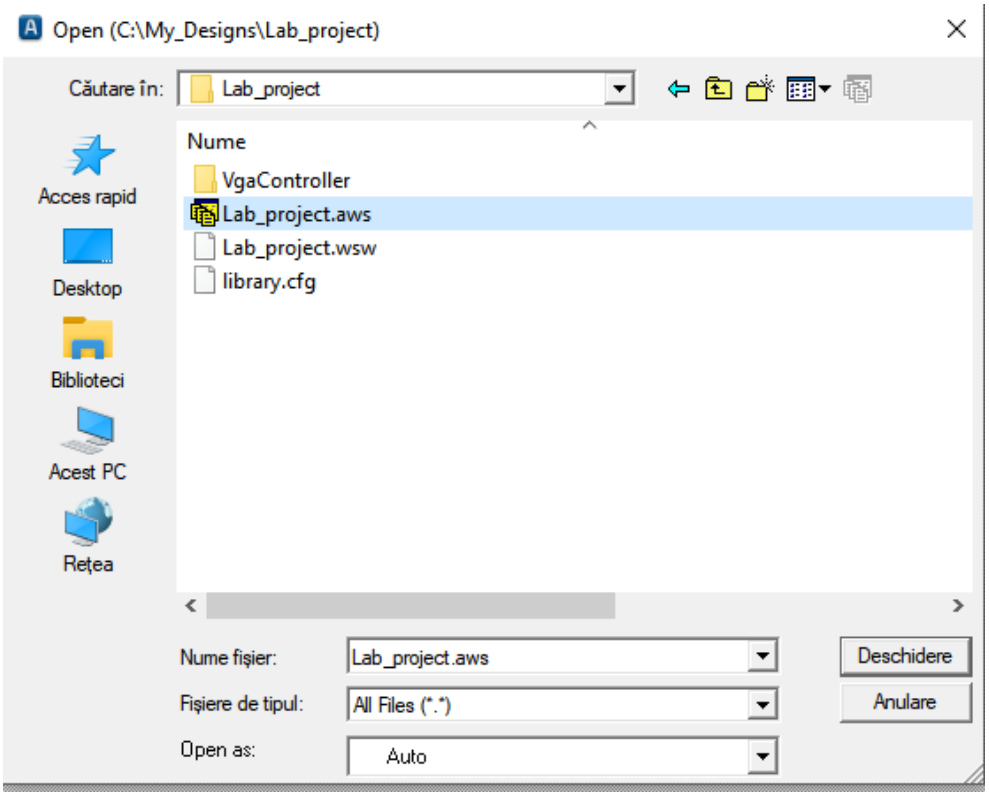
6. User manual

6.1. Active-HDL Simulation

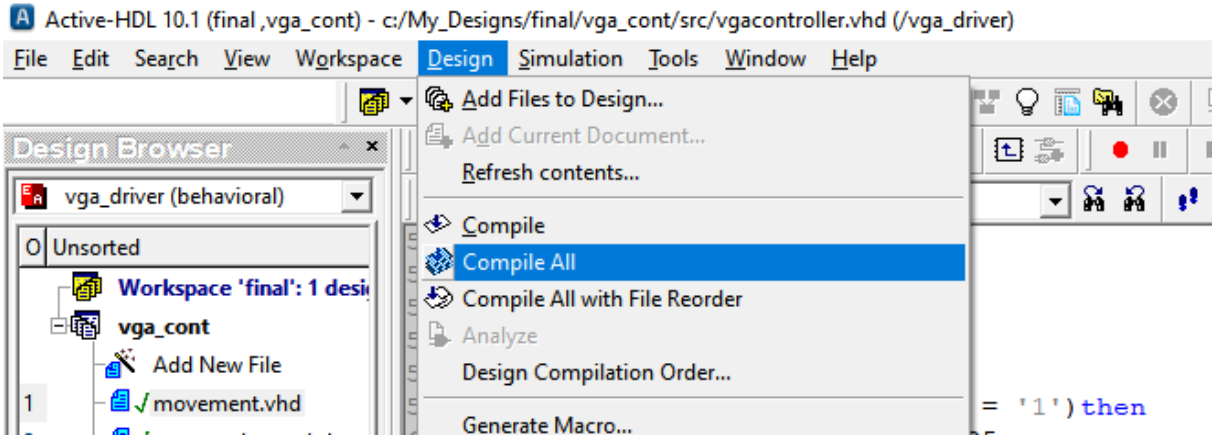
In order to simulate our VGA Controller in Active-HDL we need to follow some steps after running the program

First step : After the aplication has started running we will open our project from **File -> Open**, or we can use a shortcut **CTRL + 0 .**

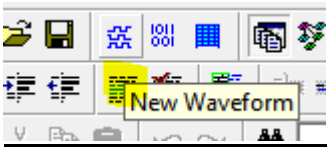




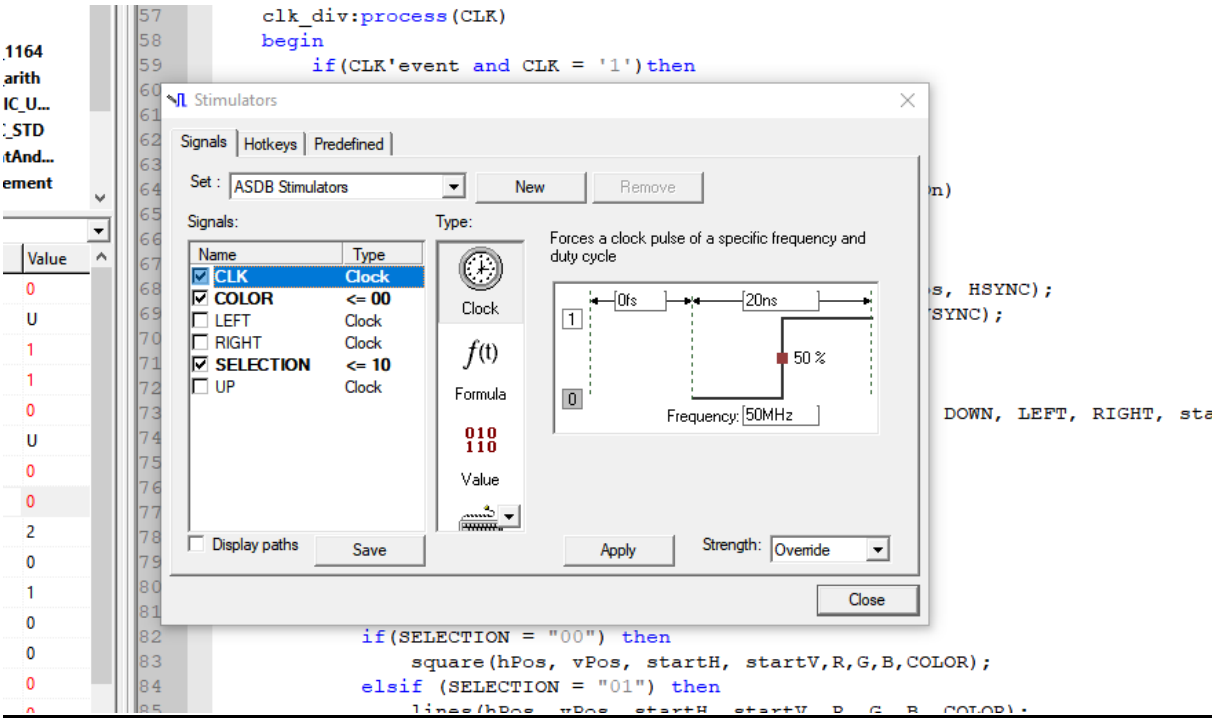
Second step : The code wil be displayed on the screen. To initialize the simulation we must first compile our project **Design -> Compile all**

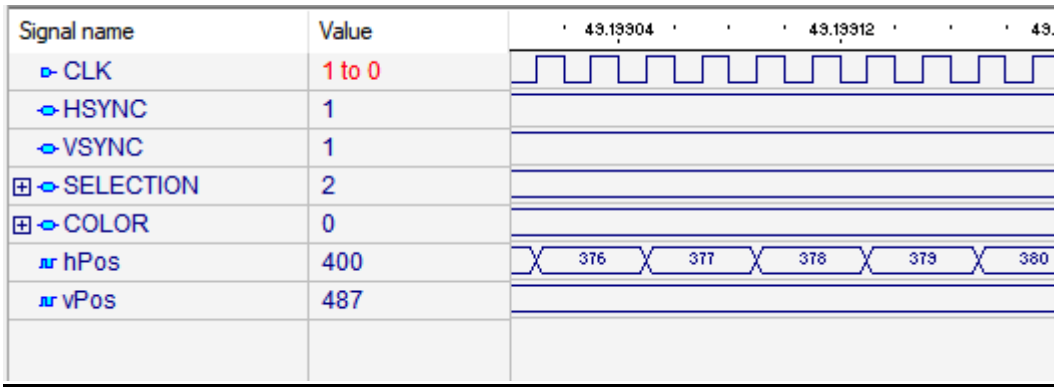


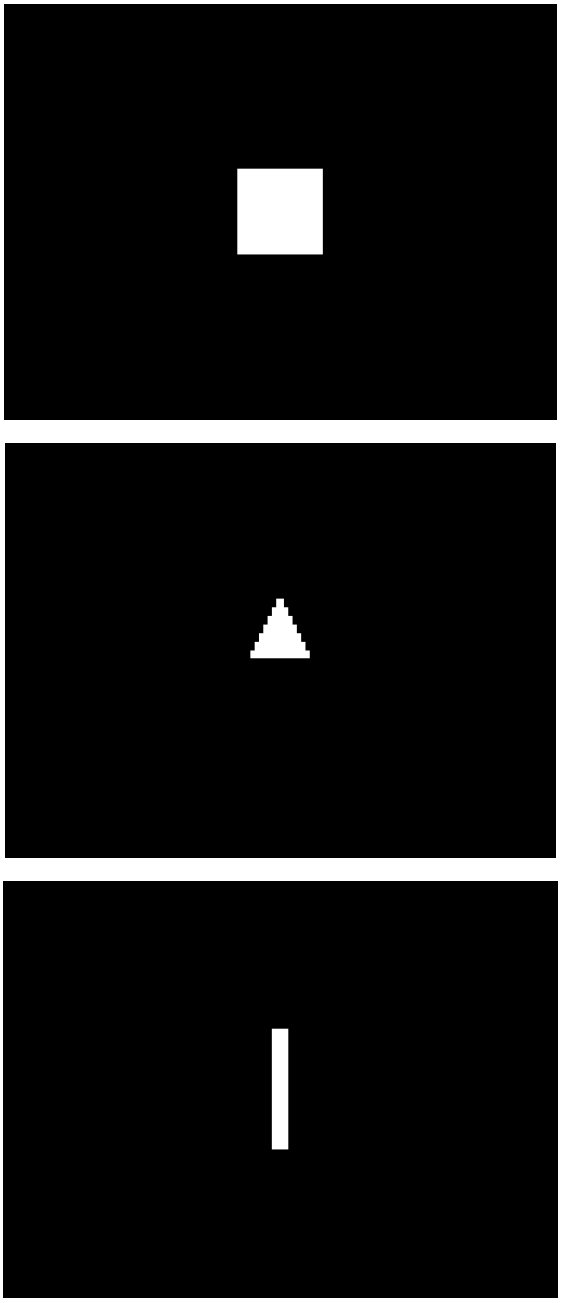
Third step : We need now to initialize our simulation from the menu **Simulation -> Initialize Simulation.** In order to display our simulation window we need no press the icon **New Waveform**


New Waveform

Step four : We will select all signals that we are using in the project and we will go to **Stimulators** Tab. From the stimulator tab we can select values for our simulation. We must first assign a 50 MHz Clock to signal **CLK** and select which figure we want to display **SELECTION.** We can also change the color by assigning a value to **COLOR.**



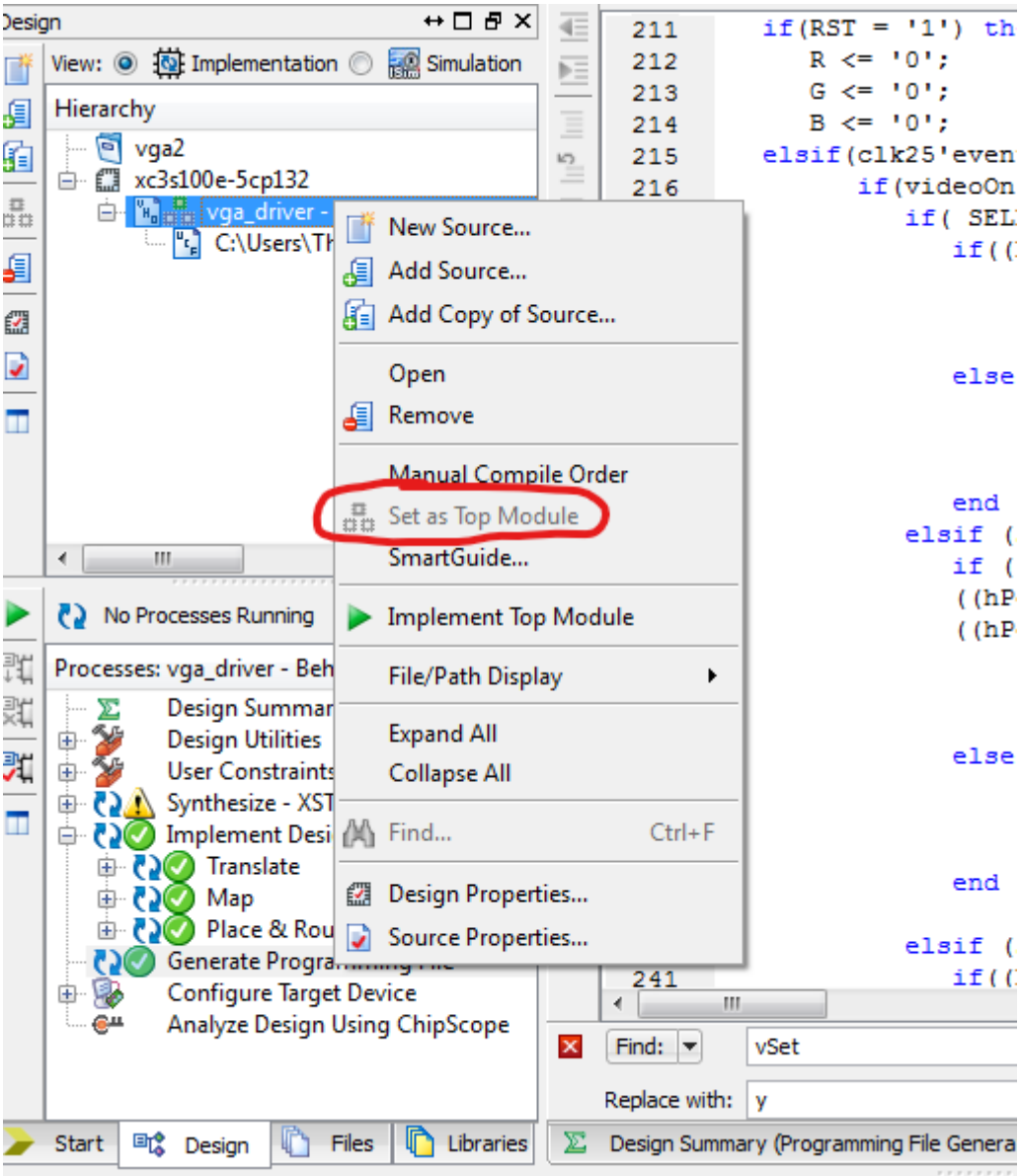




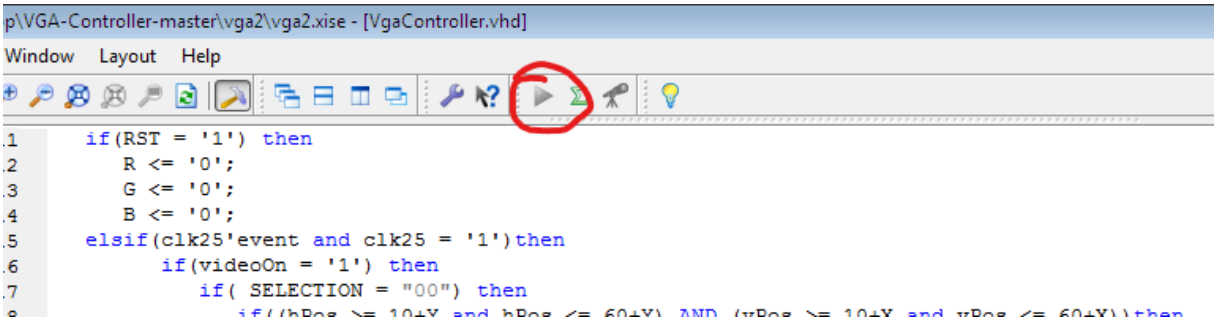
6.2. Ise Design Suite – Basys2

For this example ISE Design Suite 14.7 was used. We need to create a new project by clicking

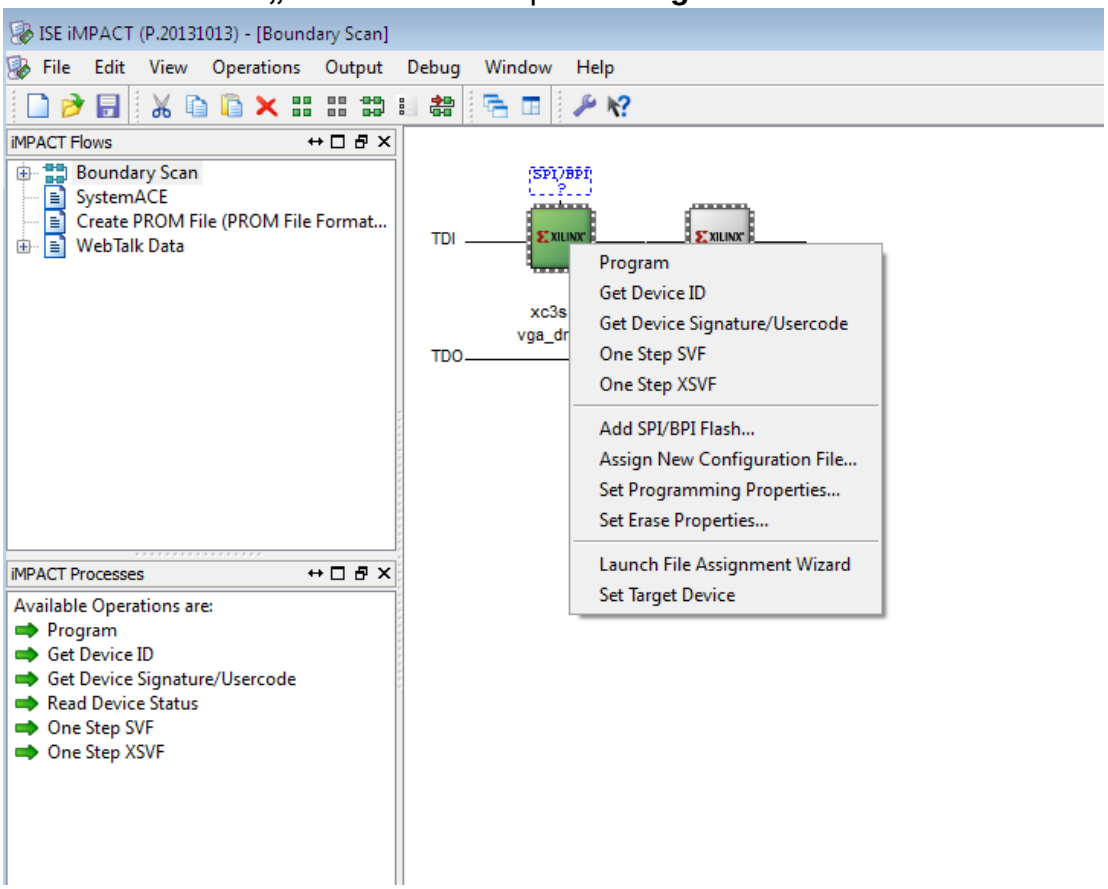
1. File → New Project → Save.
2. Next step is to import the existing vhdl file named „VgaController.vhd" by right clicking on the board name and selecting „Add source".
3. Make sure that the „vga_driver" file is implemented as a top module



4. Next step is to Implement the Design by pressing the Green Arrow



5. Next step is to program the generated bitstream to our board, using Ise Design Suite Impact tool. Open IMPACT by double click „Configure target Device”
6. Right-click on Boundary Scan and select Initalize chain, You will be asked if tou want to assign a new configuration file to the board. Select YES.
7. Last step to assign the new configuration to our board is to right click board name. For me it is **„XC3S100E”** and press **Program.**



7. Further Development

For furher development we can use a ROM memory to store more complex images/figures and we can transform the whole project in a game that can be controlled from the board or maybe from a keyboard. We can tweak the signals to get a better resolutin and a higher refresh rate so things can look more real.

8. Code

```
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 work.sq.all;
use work.line.all;
use work.b.all;
use work.tr.all;
use work.countAndSync.all;
use work.movement.all;

entity vga_driver is
    Port ( CLK : in  STD_LOGIC;
           RST : in  STD_LOGIC;
           HSYNC : inout  STD_LOGIC;
           VSYNC : inout  STD_LOGIC;
                       UP : in STD_LOGIC;
                       DOWN : in STD_LOGIC;
                       LEFT : in STD_LOGIC;
                       RIGHT : in STD_LOGIC;
                       SELECTION : in STD_LOGIC_VECTOR ( 1 downto 0);
                       COLOR : in STD_LOGIC_VECTOR ( 1 downto 0);
                               R : inout STD_LOGIC ;
                               G : inout STD_LOGIC ;
                               B : inout STD_LOGIC);

end vga_driver;

architecture Behavioral of vga_driver is

            signal clk25 : std_logic := '0';

            constant HD : integer := 639;  --  639   Horizontal Display (640)
            constant HFP : integer := 16;        --   16   Right border (front porch)
            constant HSP : integer := 96;       --   96   Sync pulse (Retrace)
            constant HBP : integer := 48;       --   48   Left boarder (back porch)

            constant VD : integer := 479;   --  479   Vertical Display (480)
            constant VFP : integer := 10;          --   10   Right border (front porch)
            constant VSP : integer := 2;                                        --    2   Sync pulse (Retrace)
            constant VBP : integer := 33;       --   33   Left boarder (back porch)

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

            signal videoOn : std_logic := '0';

            signal startH : integer := 320;
            signal startV : integer := 240;



            signal R1,G1,B1 : std_logic;
```

```
	signal x : integer :=320;
	signal y : integer :=240;
	signal lastUP, lastDOWN, lastLEFT, lastRIGHT : std_logic :='0';

begin

clk_div:process(CLK)
begin
	if(CLK'event and CLK = '1')then
		clk25 <= not clk25;
	end if;
end process;

Horizontal_position_counter:process(clk25, RST)
begin
	if(RST = '1')then
		hpos <= 0;
	elsif(clk25'event and clk25 = '1')then
		if (hPos = (HD + HFP + HSP + HBP)) then
			hPos <= 0;
		else
			hPos <= hPos + 1;
		end if;
	end if;
end process;

Vertical_position_counter:process(clk25, RST, hPos)
begin
	if(RST = '1')then
		vPos <= 0;
	elsif(clk25'event and clk25 = '1')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;

Horizontal_Synchronisation:process(clk25, RST, hPos)
begin
	if(RST = '1')then
		HSYNC <= '0';
	elsif(clk25'event and clk25 = '1')then
		if((hPos <= (HD + HFP)) OR (hPos > HD + HFP + HSP))then
			HSYNC <= '1';
		else

			HSYNC <= '0';

		end if;

	end if;

end process;



Vertical_Synchronisation:process(clk25, RST, vPos)

begin

	if(RST = '1')then
```

```
                VSYNC <= '0';
        elsif(clk25'event and clk25 = '1')then
                if((vPos <= (VD + VFP)) OR (vPos > VD + VFP + VSP))then
                        VSYNC <= '1';
                else
                        VSYNC <= '0';
                end if;
        end if;
end process;


video_on:process(clk25, RST, hPos, vPos)
begin
        if(RST = '1')then
                videoOn <= '0';
        elsif(clk25'event and clk25 = '1')then
                if(hPos <= HD and vPos <= VD)then
                        videoOn <= '1';
                else
                        videoOn <= '0';
                end if;
        end if;
end process;

        colorSet : process(clk25, COLOR)
        begin
        if(clk25'event and clk25 = '1')then
                        if(COLOR = "00")  then
                                R1<='1';
                                G1<='0';
                                B1<='0';
                        elsif(COLOR = "01")  then
                                R1<='0';
                                G1<='1';
                                B1<='0';
                        elsif(COLOR = "10")  then
                                R1<='0';
                                G1<='0';
                                B1<='1';
                        elsif(COLOR = "11")  then
                                R1<='1';
                                G1<='1';
                                B1<='1';
                        end if;
        end if;
        end process;

        position : process(clk25, UP, DOWN, LEFT, RIGHT, X, Y)
```

```
begin
	if(clk25'event and clk25 = '1')then

		if(UP = '1' and lastUP = '0') then
			X <= X - 5 ;
		elsif (DOWN = '1' and lastDOWN = '0') then
			X <= X + 5 ;
		elsif (LEFT = '1' and lastLEFT = '0') then
			Y <= Y - 5 ;
		elsif(RIGHT = '1' and lastRIGHT = '0') then
			Y <= Y + 5 ;
		end if;
		lastUP <= UP;
		lastDOWN <= DOWN;
		lastLEFT <= LEFT;
		lastRIGHT <= RIGHT;
	end if;
end process;



drawing : process(clk25, RST, hPos, vPos, videoOn, startH, startV, X, Y)
begin

if(RST = '1') then
	R <= '0';
	G <= '0';
	B <= '0';
elsif(clk25'event and clk25 = '1')then
		if(videoOn = '1') then
			if( SELECTION = "00") then
	if((hPos >= 10+Y and hPos <= 60+Y) AND (vPos >= 10+X and vPos <= 60+X))then
				R <= R1 ;
				G <= G1 ;
				B <= B1 ;
			else
				R <= '0';
				G <= '0';
				B <= '0';
			end if;
	elsif (SELECTION = "01") then
	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;

	elsif (SELECTION = "10") then
	if((hPos > y - 10) and (hpos < y + 10) and (vPos > x - 70) and (vPos < x + 70)) then
					R <= R1 ;
					G <= G1 ;
					B <= B1 ;
				else
					R <= '0';
					G <= '0';
					B <= '0';
```

```
                                        end if;

                        elsif (SELECTION = "11") then
                                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 if;
                                end if;
        end if;
        end process;
end Behavioral;
```