



# **UNIVERSITATEA TEHNICĂ**

**DIN CLUJ-NAPOCA**

**PROJECT “VGA CONTROLLER”**

**CLASS: DIGITAL SYSTEM DESIGN**

**STUDENTS: PATRICK BUDEA,**

**CRISTIAN-CLAUDIU CHIRA**

**PROFESSOR: VLAD CRISTIAN MICLEA**

## **TABLE OF CONTENTS**

- **SPECIFICATION**
- **BLOCK DIAGRAM**
- **DESIGN**
- **UTILIZED COMPONENTS**
- **CODE PRESENTATION & EXPLANATION**
- **SIMULATION**
- **USER INSTRUCTIONS**
- **SOLUTION JUSTIFICATION AND IMPROVEMENT IDEAS**

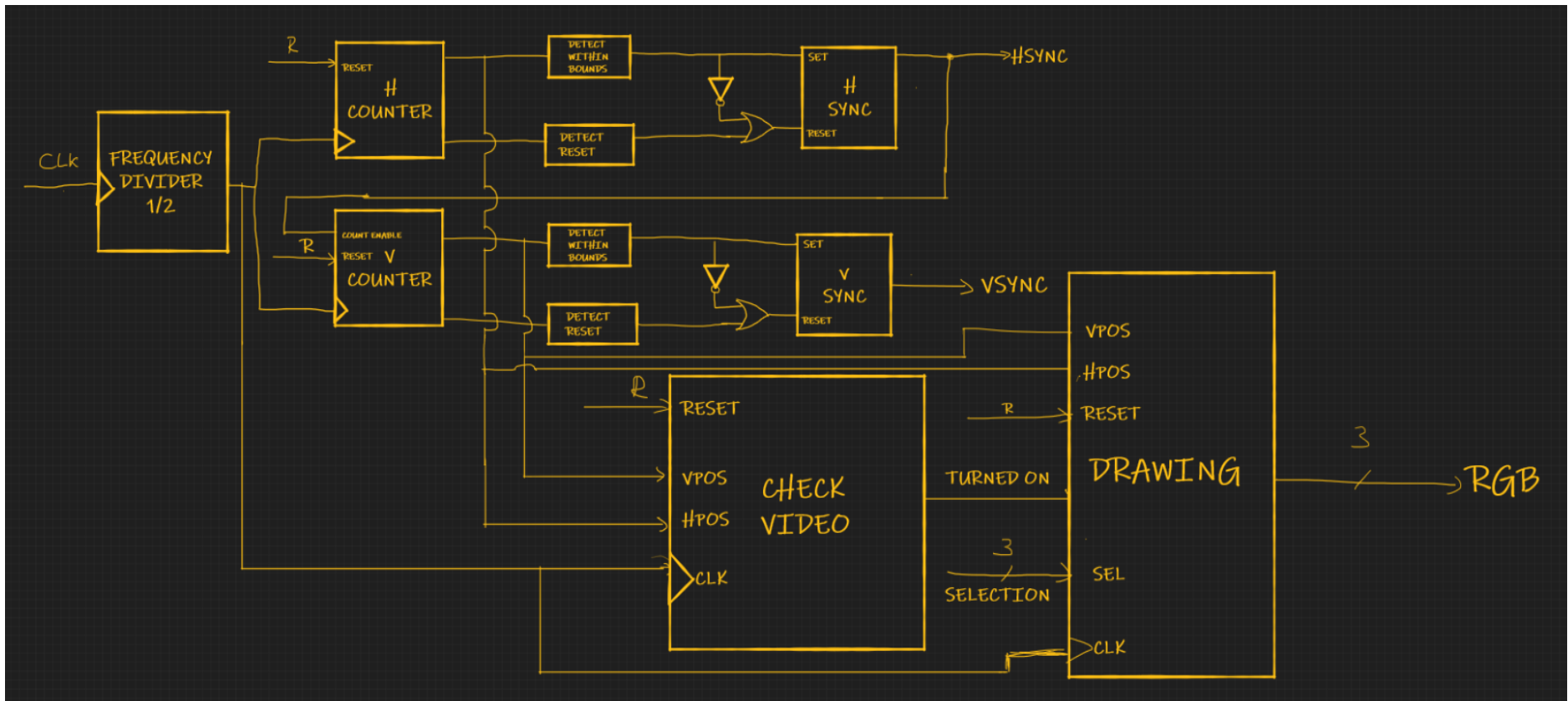
## **SPECIFICATION**

### **VGA CONTROLLER**

This project's aim is to describe, using VHDL, how a VGA controller functions. The controller is designed to draw several different shapes in a multitude of different colors, on a 640x480 resolution display.

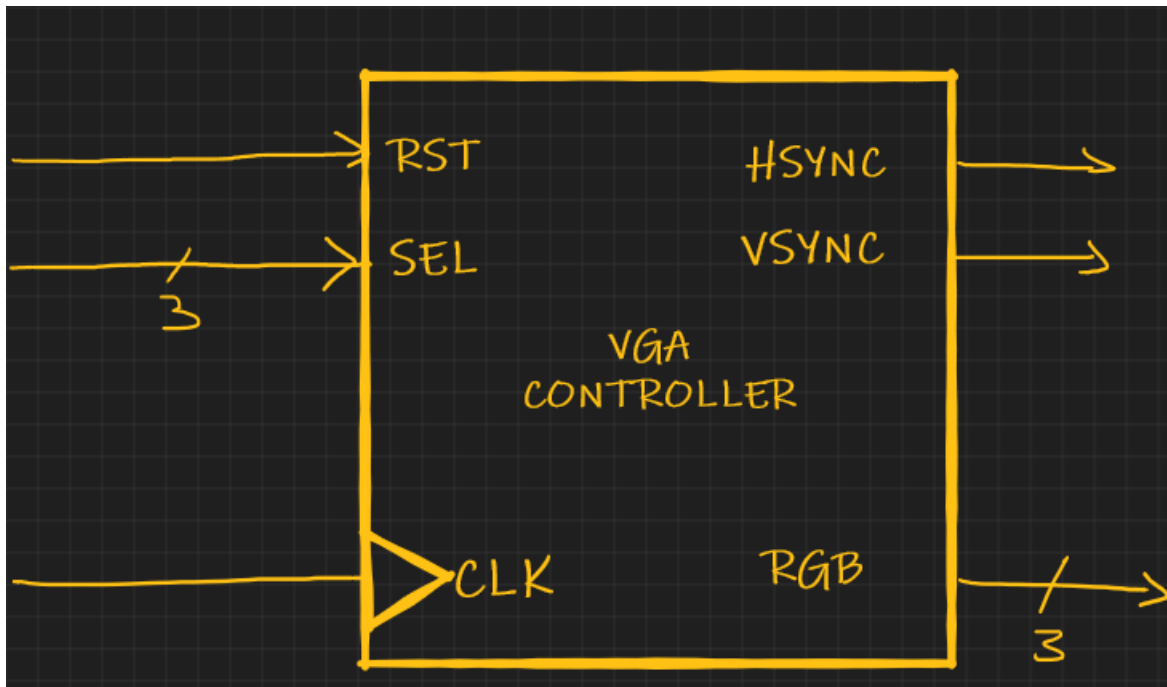
The controller was designed in a synchronous manner, meaning the whole operation is led by a singular clock pulse. The project was designed top-down, using a behavioral architecture, resulting in an easily synthesizable program.

## BLOCK DIAGRAM



## DESIGN

The main idea behind the controller consists of a cursor which traverses the display left-right and top-bottom through different areas which are called porches. The purpose of the porches is to synchronize horizontal and vertical iteration of the drawing. The controller adheres to the industry standards, having the clock pulse at 25Hz with a refresh rate of 50Hz.



Once the user selects the desired shape, the cursor will start moving across the screen and each pixel will be colored either black (background) or one of the available colors (the desired shape). If a reset is needed, the user will activate the Reset input which will stop the drawing and reset the cursor position.

The controller receives 3 inputs and provides 3 outputs for the FPGA board.

### INPUTS

- 50Hz clock pulse (CLK)
- Reset (RST)
- 3 bit selection (SEL)

### OUTPUTS

- HSYNC
- VSYNC
- 3 bit RGB

## UTILIZED COMPONENTS

The controller consists of:

- 1 frequency divider (FREQUENCY DIVIDER  $\frac{1}{2}$ )
- 2 synchronous up counters (V COUNTER & H COUNTER)
- 7 comparators (2×DETECT WITHIN BOUNDS, 2×DETECT RESET, HSYNC, VSYNC, CHECK VIDEO)
- 1 multiplexer (DRAWING)
- 2 inverters
- 2 OR gates

## CODE PRESENTATION & EXPLANATION

In this section of the documentation the code's meaning and functionality is going to be explained, linking the components to the VHDL code.

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

These are the libraries that have been imported. The controller utilizes standard logic components, as well as standard integer signals and variables.

```
entity vga_driver is
    Port ( CLK: in STD_LOGIC;
          RST: in STD_LOGIC;
          sel: in STD_LOGIC_VECTOR( 2 downto 0);
          HSYNC: out STD_LOGIC;
          VSYNC: out STD_LOGIC;
          RGB: out STD_LOGIC_VECTOR (2 downto 0));
end vga_driver;
```

The entity consists of, as presented above, 3 inputs (*CLK*, *RST*, *sel*) and 3 outputs (*HSYNC*, *VSYNC*, *RGB*). The outputs are transmitted to the FPGA and give instructions regarding the position and color of each pixel on the display.

```
architecture Behavioral of vga_driver is
    signal clk25: std_logic:='0';

    constant HD: integer :=640;           -- horizontal display
    constant HFP: integer :=16;           -- front porch
    constant HSP: integer :=96;           -- sync pulse
    constant HBP: integer :=48;           -- back porch

    constant VD: integer :=480;           -- vertical display
    constant VFP: integer :=10;           -- front porch
    constant VSP: integer :=0;            -- sync pulse !!!
    constant VBP: integer :=33;           -- back porch

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

    signal offset : integer :=0;

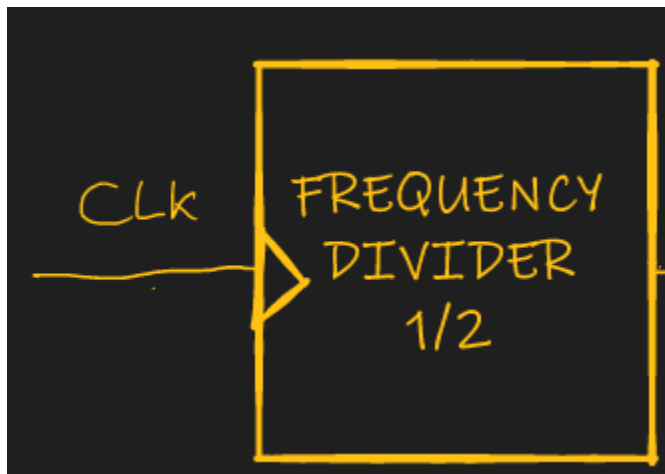
    signal turnedOn: std_logic:= '0';
```

The constants which the controller uses are integers and have values according to the display and the porches.

The signals *hpos* & *vpos* are also integers and they are incremented during the counting process.

The *offset* signal is used when drawing a triangle, while the *turnedOn* signal changes value if the display is on or off.

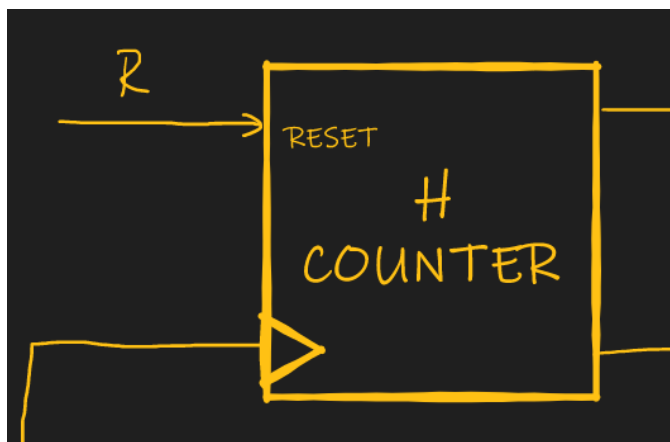
## The frequency divider



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

The frequency divider  $\frac{1}{2}$  is used for decreasing the clock pulse from 50Hz to 25 Hz, respecting market standards.

## The horizontal position counter

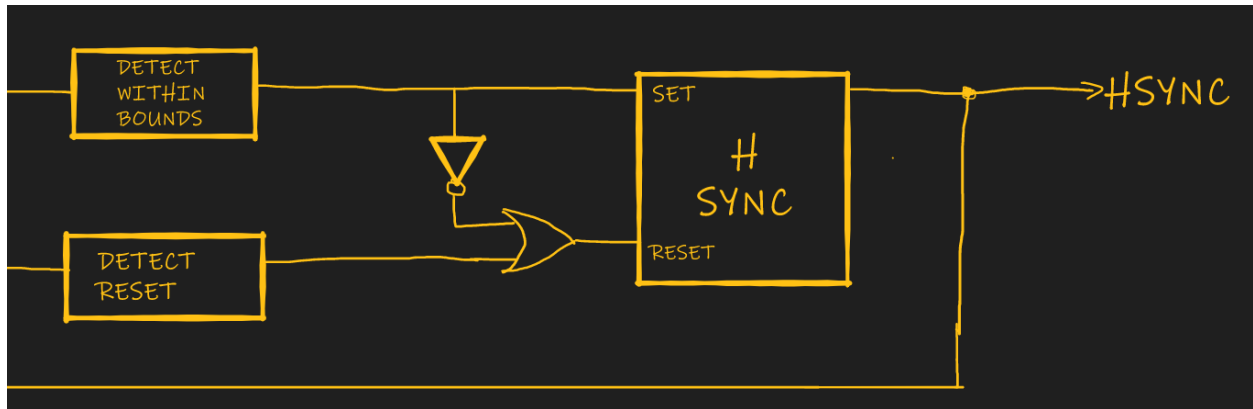


```
Horizontal_pos_counter:process (clk25, RST)
begin
    if (RST = '1') then
        hpos <= 0;
    elsif (clk25'event and clk25='1') then
        if (hpos = HD + HFP + HSP + HBP) then
            hpos <= 0;
        else
            hpos <= hpos+1;
        end if;
    end if;
end process;
```

This counter increments the *hpos* signal according to the 25Hz clock's rising edge, while also resetting if the end of the display has been met.



## Horizontal sync

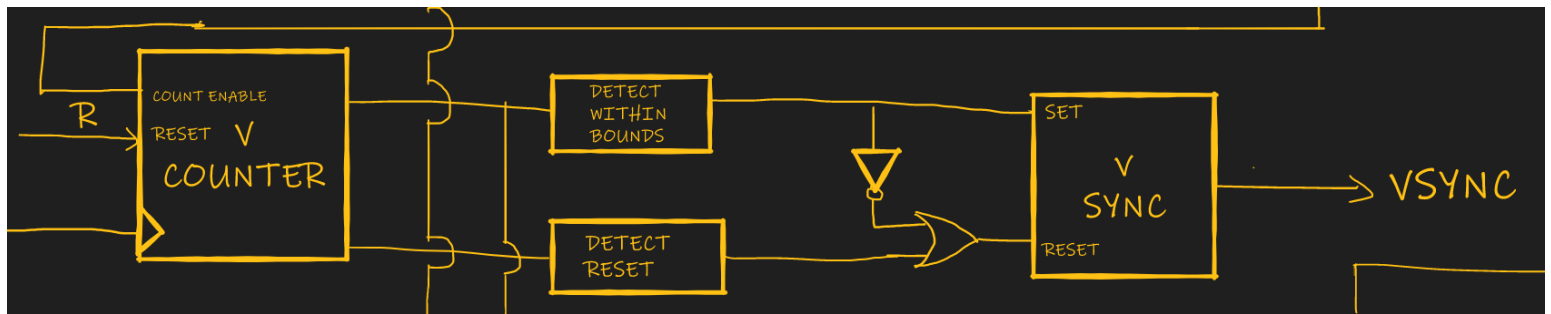


```

Horizontal_sync: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;
  
```

This component changes the value of the *HSYNC* signal. It becomes '0' if *RST* has been applied or if the counter is out of bounds, according to the porch values. The *HSYNC* is then transmitted to the FPGA.

## The vertical position counter & vertical sync

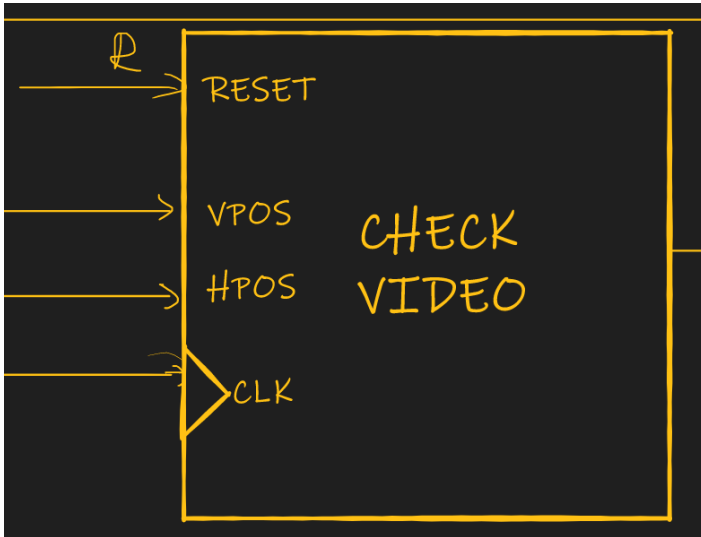


```
Vertical_pos_counter:process (clk25, RST, hpos)
begin
    if(RST = '1')then
        vpos <= 0;
    elsif(clk25'event and clk25='1') then
        if(hpos = HD + HFP + HSP + HBP)then -- count enable
            if(vpos = VD + VFP + VSP + VBP)then
                vpos <= 0;
            else
                vpos <= vpos+1;
            end if;
        end if;
    end if;
end process;
```

```
Vertical_sync: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;
```

The vertical position counter and the vertical sync work in a similar manner as their horizontal counterparts, but also having a selection statement which enables the *V\_COUNTER* if that condition has been met. The *VSYNC* is then transmitted to the FPGA.

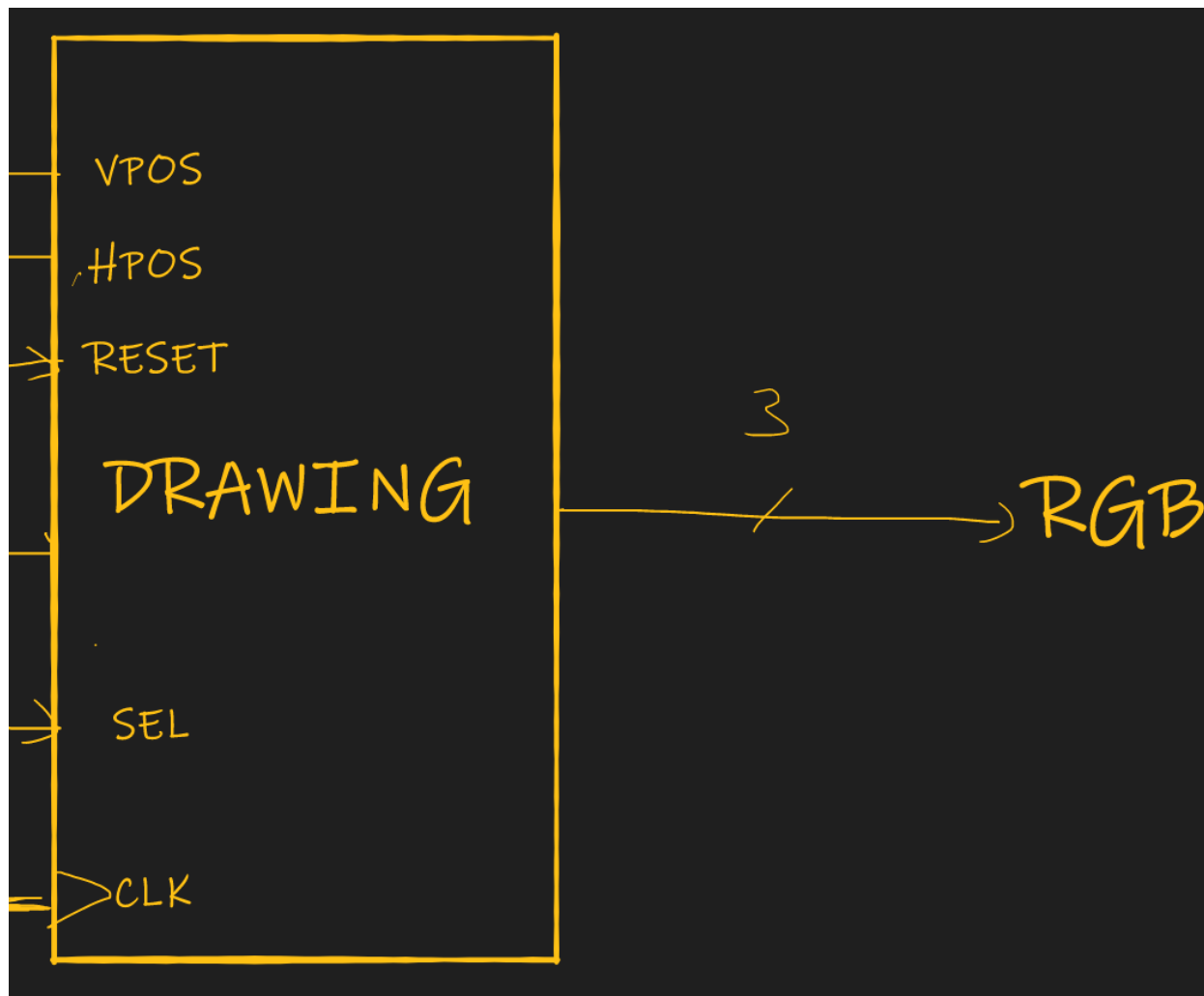
## Check video



```
checkVideo: process(clk25, RST, hpos, vpos)
begin
    if(RST = '1')then
        turnedOn <= '0';
    elsif(clk25'event and clk25='1')then
        if(hpos <= HD and vpos <=VD)then
            turnedOn <= '1';
        else
            turnedOn <= '0';
        end if;
    end if;
end process;
```

This component is designed to change the value of the *turnedOn* signal if the cursor that draws each pixel is inside or outside the display area, sending this information to the FPGA.

## Drawing component



```

drawing: process (clk25, RST, hpos, vpos, turnedOn, sel)
begin
    case sel is
        when "000" => -- square
            if(RST = '1') then
                RGB <= "000"; -- black
            elsif (clk25'event and clk25 = '1') then
                if(turnedOn = '1') then
                    if((hpos >= 1 and hpos <= 60) and (vpos >= 0 and vpos <= 60)) then
                        RGB <= "111"; -- white
                    else
                        RGB <= "000";
                    end if;
                else
                    RGB <= "000";
                end if;
            end if;
        when "001" => -- rect
            if(RST = '1') then
                RGB <= "000";
            elsif (clk25'event and clk25 = '1') then
                if(turnedOn = '1') then
                    if((hpos >= 0 and hpos <= 300) and (vpos >= 0 and vpos <= 250)) then
                        RGB <= "101"; -- pink
                    else
                        RGB <= "000";
                    end if;
                else
                    RGB <= "000";
                end if;
            end if;
    end case;
end process;

```

This component is essentially a type of MUX which allows the user to select the shape that needs to be drawn by using the *sel* input (3 bit vector). This process utilizes the *case expression* when selecting the drawing, which makes the code readable and well-formatted, but also eases the inclusion of additional shapes or colors. For example during the *square* mode (*sel* = "000") the RGB output changes values from "000" to "111" (black → white). The square will be drawn in the area marked by the conditions put on *hpos* and *vpos* signals, while the *turnedOn* signal is active.







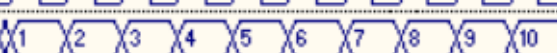




In a similar way will the other shapes be drawn when the user chooses another value for *sel*.

Sel	Drawing
000	White square
001	Pink rectangle
010	Green triangle
011	Blue horizontal parallel lines
100	Cyan vertical parallel lines

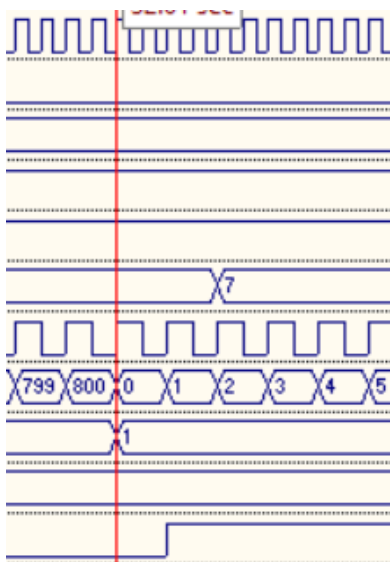
## SIMULATION

In this part of the documentation the simulation of the shapes will be explained. The simulation uses a step with a value of 1000ms.

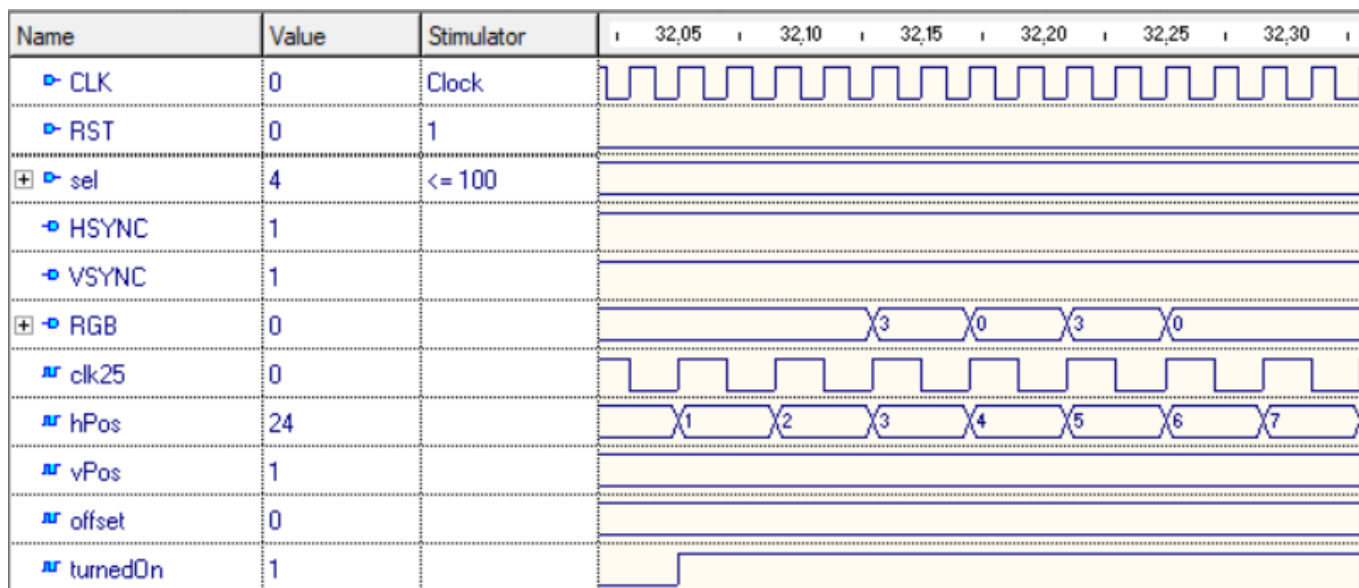
### Square

CLK	0	Clock	
RST	0	1	
sel	0	<= 000	
HSYNC	1		
VSYNC	1		
RGB	0		
clk25	0		
hPos	75		
vPos	0		
offset	0		
turnedOn	1		

As it can be seen, when the *hpos* signal reaches the value of 2, the *RGB* output will be changed to “111” and *turnedOn* is activated, which means the square will start to appear (white on black background). The *RGB* output will become “000” when the *hpos* counter reaches 62 and no shape will be drawn until *vpos* is incremented and the process will start again.



## Vertical parallel lines



In a similar way, this is the simulation of vertical parallel lines, when *sel* has the value “100”. As the simulation suggests, a pixel will be colored for a tick, then *RGB* becomes “000” for another tick, and eventually, after that small break, another pixel will be colored. This process is done for every iteration of the *vpos* counter, resulting in 2 cyan parallel lines.

## USER INSTRUCTIONS

The controller is easy to use and the simulation settings are easy to set up. The user should add a CLOCK stimulator only for the *CLK* input with the value of 50Hz. Also the *RST* should have a stimulator, for example a key from the keyboard if the user wants to reset the controller at any time during the simulation. The last input is *sel*, a 3 bit vector with which the user chooses the desired shape.

The simulation step should be assigned to 1000ms in order to simulate a drawing faster. The user should watch where the *RGB* output and the *turnedOn* signal change values, since those are the moments where the coloring of pixels takes place.

## SOLUTION JUSTIFICATION AND IMPROVEMENT IDEAS

We have chosen this solution because of the following:

- The idea is easy to understand, bearing resemblance to traversing a matrix.
- The code is intuitive and structured.
- Using the tools of VHDL, the idea is easy to implement, while also having proper feedback using the simulator.

A few ways to improve the VGA controller as a whole are:

- Additional shapes and colors (this would require extending the ranges of the *RGB* and *sel* vectors).
- Allowing user input for the positioning, size and color of the shapes.
- Creating a test-bench.
- Increasing the resolution in order to allow more complex shapes.



In this last part the code will be attached.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.NUMERIC_STD.ALL;

entity vga_driver is
    Port ( CLK: in STD_LOGIC;
           RST: in STD_LOGIC;
           sel: in STD_LOGIC_VECTOR( 2 downto 0);
           HSYNC: out STD_LOGIC;
           VSYNC: out STD_LOGIC;
           RGB: out STD_LOGIC_VECTOR (2 downto 0));
end vga_driver;

architecture Behavioral of vga_driver is

    signal clk25: std_logic:='0';

    constant HD: integer :=640;           -- horizontal display
    constant HFP: integer :=16;           -- front porch
    constant HSP: integer :=96;           -- sync pulse
    constant HBP: integer :=48;           -- back porch

    constant VD: integer :=480;           -- vertical display
    constant VFP: integer :=10;           -- front porch
    constant VSP: integer :=0;            -- sync pulse      !!!
    constant VBP: integer :=33;           -- back porch
```

```
signal hPos : integer :=0;

signal vPos : integer :=0;

signal offset : integer :=0;

signal turnedOn: std_logic:= '0';

begin

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

    Horizontal_pos_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_pos_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 -- count enable
                if(vpos = VD + VFP + VSP + VBP)then
                    vpos <= 0;
                else
                    vpos <= vpos+1;
                end if;
            end if;
        end if;
    end process;

Horizontal_sync: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_sync: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;

checkVideo: process(clk25, RST, hpos, vpos)
begin
    if(RST = '1')then
        turnedOn <= '0';
    elsif(clk25'event and clk25='1')then
        if(hpos <= HD and vpos <=VD)then
            turnedOn <= '1';
        else
            turnedOn <= '0';
        end if;
    end if;

end process;

drawing: process (clk25, RST, hpos, vpos, turnedOn, sel)
begin
    case sel is
        when "000" => -- square
            if(RST = '1')then
                RGB <= "000"; -- black
            elsif(clk25'event and clk25= '1')then
                if(turnedOn = '1')then
                    if((hpos >= 1 and hpos <= 60) and (vpos >= 0 and
vpos <=60))then

                        RGB <= "111" ; -- white
                    else

```

```

                                RGB <= "000";

                                end if;

                                else

                                RGB <= "000";

                                end if;

                                end if;

when "001" => -- rect

                                if(RST = '1')then

                                RGB <= "000";

                                elsif(clk25'event and clk25= '1')then

                                if(turnedOn = '1')then

                                if((hpos >= 0 and hpos <= 300) and (vpos >= 0 and
vpos <=250))then

                                RGB <= "101" ;           -- pink

                                else

                                RGB <= "000";

                                end if;

                                else

                                RGB <= "000";

                                end if;

                                end if;

when "010" => -- triangle

                                if(RST = '1')then

                                RGB <= "000";

                                elsif(clk25'event and clk25= '1')then

                                if(turnedOn = '1')then

                                if((hpos >=0 and hpos<=60-offset) and (vpos >= 1 and
vpos <=61))then

                                RGB <= "010"; -- green

                                else

                                RGB <= "000";

                                end if;

                                if(hpos = 60 - offset)then

```

```

                                offset <= offset+1;

                                end if;

                                else

                                RGB <= "000";

                                end if;

                                end if;

when "011" => -- hlines

    if(RST = '1')then

        RGB <= "000";

    elsif(clk25'event and clk25= '1')then

        if(turnedOn = '1')then

            if((hpos >= 2 and hpos <= 30) and (vpos = 1 or vpos
= 3))then

                RGB <= "001" ;          -- blue

            else

                RGB <= "000";

            end if;

        else

            RGB <= "000";

        end if;

    end if;

when "100" => -- vlines

    if(RST = '1')then

        RGB <= "000";

    elsif(clk25'event and clk25= '1')then

        if(turnedOn = '1')then

            if((hpos = 2 or hpos = 4) and (vpos >= 1 and vpos <=
30))then

                RGB <= "011" ; -- cyan

            else

                RGB <= "000";

            end if;

        else


```

```
        RGB <= "000";  
  
        end if;  
  
        end if;  
  
        when others => null;  
  
        end case;  
  
end process;  
  
end Behavioral;
```