EEE102 Lab 06 Report

VGA Controller with VHDL

Miraç Lütfullah Gülgönül / 21602357 / EEE102 - 04

2 December 2018

The Design Methodology

The purpose of this lab was to design a VGA driver using the BASYS3 board. My proposed design was to implement one VGA controller block which takes an array of values as an input and plots a bar graph of these values-where the "bar graph width" is 1 pixel. After the inital research, I have decided upon a 640x480 60Hz setup. To calculate the necessary numerical dimension values I have used the **Calculator for video timings** by Tomi Engdah. The processes proposed in the preliminary report have worked without error, so I did not have to rewrite again. *"There are 6 different processes in the architecture. The processes* `h_pos_counter` *and* `v_pos_counter` *increment the count values for the horizontal and vertical counters. The processes* `hSyncProc` *and* `vSyncProc` *generate the sync pulses according to when the counter values transition between the display area and the porches. The process* `videoProc` *toggles a* `isVideo` *signal which is* `ON` *when the pixel counter is in the"display area", as opposed to the porches. Finally the process* `drawProc` *toggles a* `isDraw` *signal which turns* `ON` *when the current pixel coincides with the shape to be drawn. The Gaussian curve that I want to draw is stored in a* `data` *signal which is an array of integers."*

Also I have decided to show a Gaussian graph on the screen, so I wrote a small `python` script for that. *"Noting that the Gaussian function is of the form* $f(x) = \alpha e^{-\frac{(x-b)^2}{2c^2}}$ *where* α *is the height of the curve's peak,* b *is the position of the center of the peak, and* c *controls the width of the"bell". Considering the display resolution of* `640x480` *I have decided to use the parameters* $\alpha = 300, b = 320, c = 100$ *which will give us a nice, centered bell curve. Plotting the curve on the screen, we can see that it is indeed optimal. (Note that the blanks don't really exists but are there because of some Moire effect.)"*

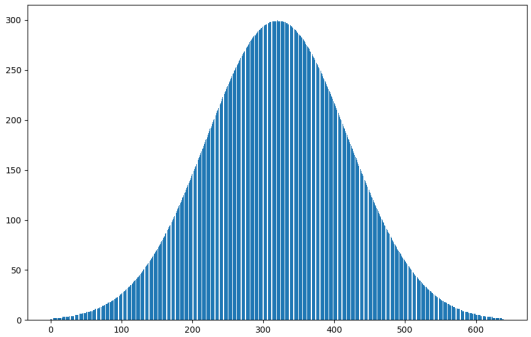

Figure 1: The plot of the Gaussian curve to be displayed.

Results

Before implementing the code on the BASYS3 board, first I had to simulate it to troubleshoot any possible problems. Even though I will not be able to directly observe the image which will be displayed on the screen, I can check the periods of the `hsync` and `vsync` signals to see if they coincide with the correct values, which are calculated using the online tool, in Figure 2.

General parameters

Pixel clock frequency 25.17 MHz

Interlace 0

Horizontal timing parameters

Resolution 640 pixels
Front porch 16 pixels
Sync pulse 96 pixels, polarity 0 *
Back porch 48 pixels

Vertical timing parameters

Resolution 480 lines
Front porch 11 lines
Sync pulse 2 lines, polarity 0 *
Back porch 31 lines

* For sync polarities 0 = NEGATIVE SYNC and 1 = POSITIVE SYNC

Numerical results

Horizontal sync frequency is 31.46 kHz.
Vertical sync frequency is 60.04 Hz.
Field rate is 60.04 Hz.

Horizontal sync pulse length is 3.81 microseconds.
Vertical sync pulse length is 63.57 microseconds.

Horizontal total is 800 pixels and vertical total is 524 lines.

Figure 2: The calculated numerical results.

We can see that the `hsync` signal bust have a frequency value of 31.46 kHz, and the `vsync` signal 60Hz. Also the pulse lenghts for these signals are given, which are respectively $3.81\mu S$ and $63.57\mu S$.

The calculator states that I need an input clock with a frequency of 25.17 Mhz. In the implemented design I will use the built-in Vivado Clock Wizard module for precision purposes, but for the testbench I will have to have a process do generate this clock. Note that a square clock wave of 25.17 Mhz has an approximate period of $40ns$ which means the process will looks

like:

```
p_CLK_GEN : process is
        begin
            wait for 20 ns;
            r_25MHz <= '1';
            wait for 20 ns;
            r_25MHz <= '0';
    end process;
```

Simulation Results

After simulating the VGA controller with the appropriate testbench, I have checked the frequencies of `hsync` and `vsync` signals. According to the calculator (as shown in Figure 2) `hsync` must have a frequency of 31.46KHz and `vsync` 60Hz. These give us period values of respectively $32\mu S$ and $16.67ms$. As can be seen on Figure 3-5, the simulation results agree with these values.

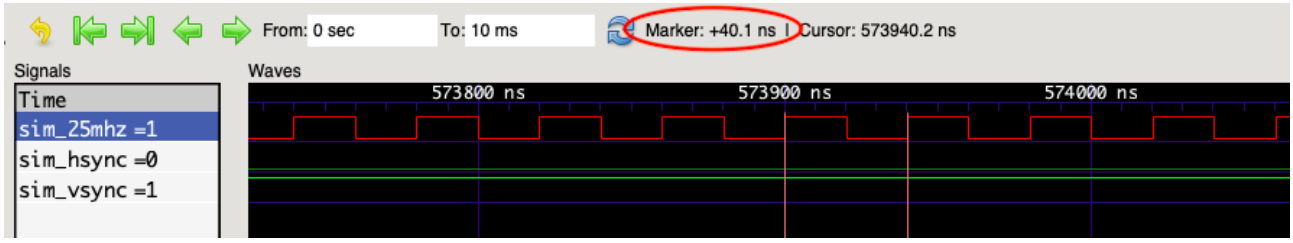

Figure 3: The simulated clock of 40ns = 25MHz

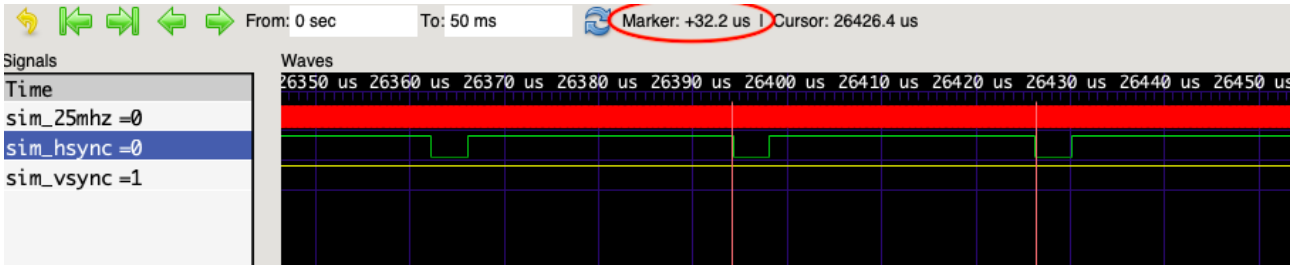

Figure 4: The hsync signal with a period of 32.2 us.

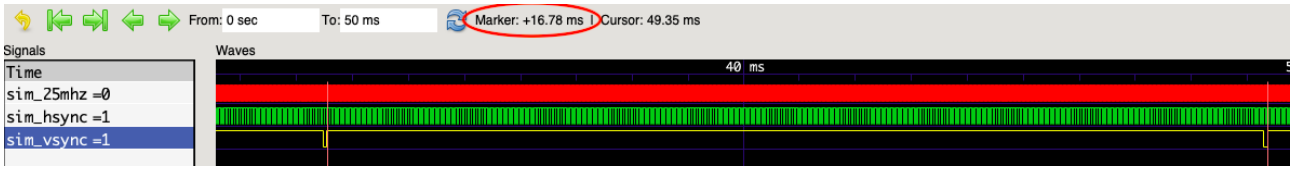

Figure 5: The vsync signal with a period of 16.78ms.

Implementation Results

After validating my code with simulation, it was time to implement it on the board. I have added a *Clock Wizard IP Module* which takes the board clock of 100MHz and outputs 25.17 MHz which is necessary for the VGADRIVER. Finally I have assigned the outputs of the VGADRIVER module to the corresponding pins on the constraints file for the VGA output of the board. I have used the LCD monitor to display the results; and as can be seen, they are as expected: a cyan Gaussian curve.

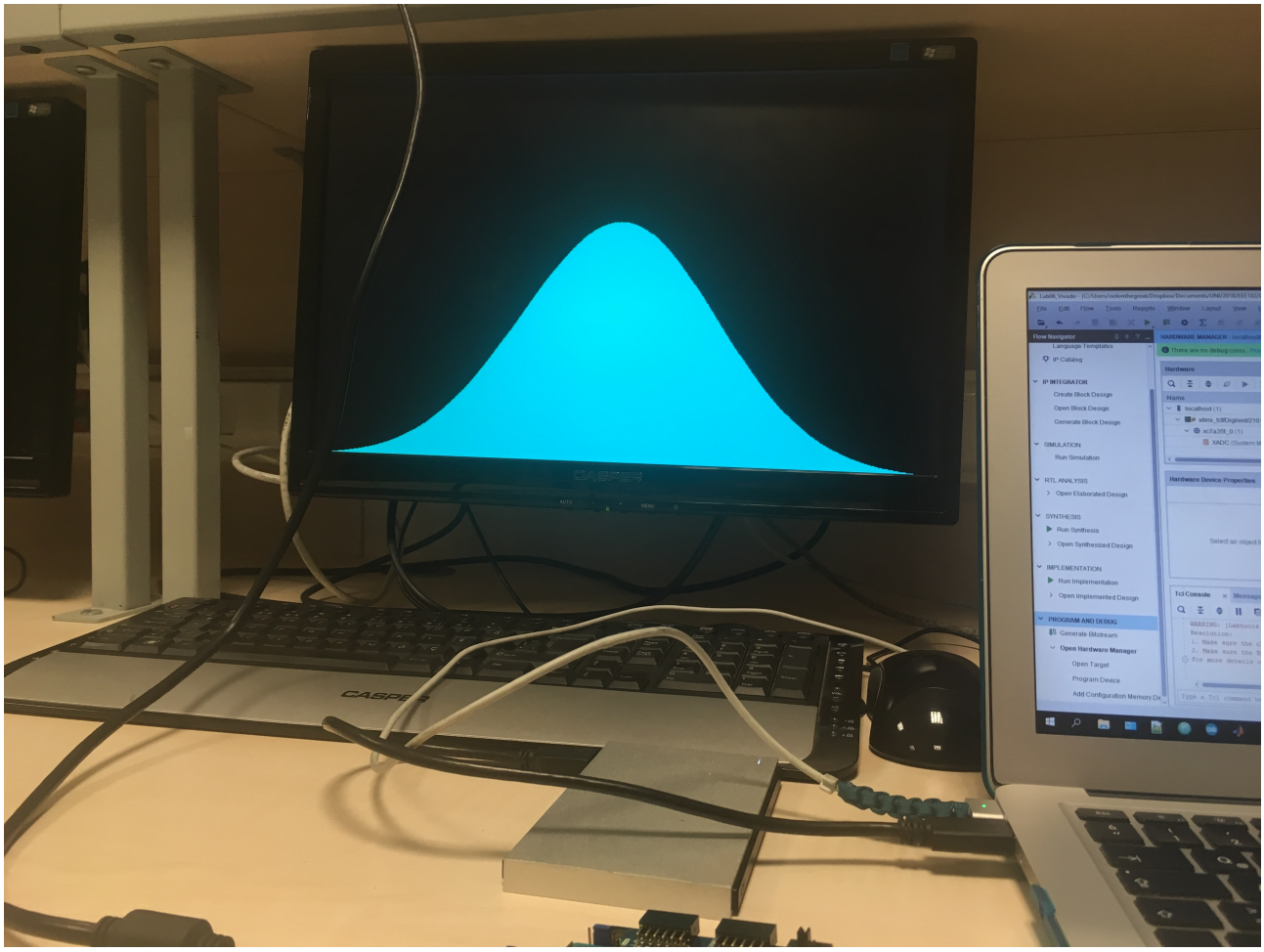

Figure 6: The general setup and the displayed curve.

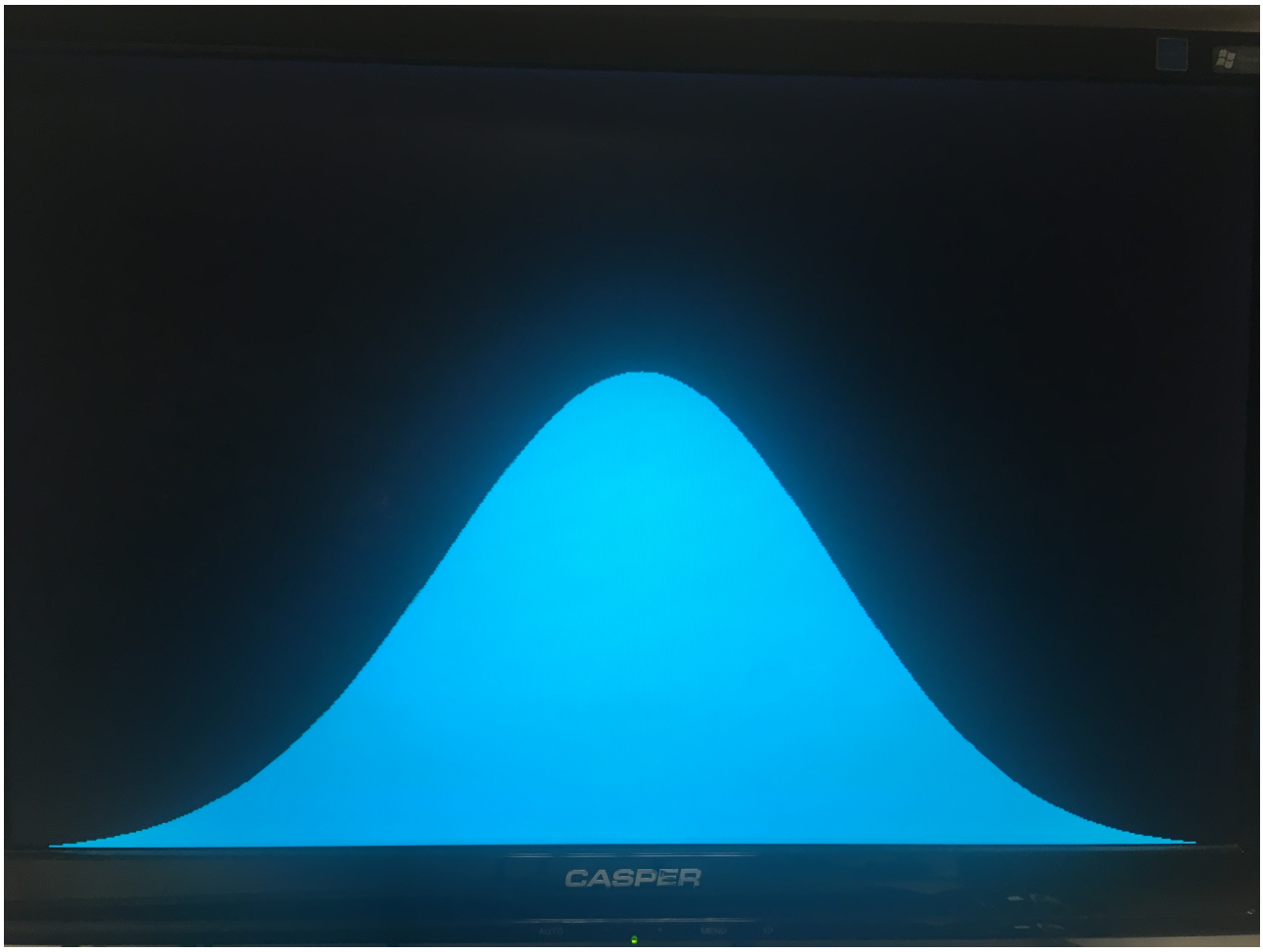

Figure 7: A closeup op the displayed curve.

Conclusion

As we can see I was able to directly design the system according to my proposed methodology, and the results were acceptable. It had surprised me that the control of VGA is based on very old CRT monitors with actual electron guns and such, but I guess why rewrite something when it works as it is.

Also, I had implemented a VGA display in my term project (the first time I took the course), so it was really no surprise that the algorithms worked again.

Appendices

VGADRIVER.vhd

```
----------------------------------------------------------------------------------
package myPack is
     type INT_ARRAY is array (0 to 700) of integer range 0 to 500 ;
end package myPack ;

----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.myPack.all;


entity VGADRIVER is

    Port (
    CLK : in STD_LOGIC ;
    RESET: in STD_LOGIC ;
    HSYNC : out STD_LOGIC ;
    VSYNC : out STD_LOGIC  ;

    RED0 : out STD_LOGIC ;
    RED1 : out STD_LOGIC ;
    RED2 : out STD_LOGIC ;
    RED3 : out STD_LOGIC ;

    GRN0 : out STD_LOGIC ;
    GRN1 : out STD_LOGIC ;
    GRN2 : out STD_LOGIC ;
    GRN3 : out STD_LOGIC ;


    BLU0 : out STD_LOGIC ;
    BLU1 : out STD_LOGIC ;
    BLU2 : out STD_LOGIC ;
    BLU3 : out STD_LOGIC
    );

end VGADRIVER;


architecture Behavioral of VGADRIVER is

signal DATA : INT_ARRAY :=
( 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 8, 8,
8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 11, 11, 11, 11, 12, 12, 12, 13, 13, 13, 14, 14,
```

```
14, 15, 15, 16, 16, 16, 17, 17, 18, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23,
23, 24, 24, 25, 26, 26, 27, 27, 28, 29, 29, 30, 31, 31, 32, 33, 33, 34, 35, 35,
36, 37, 38, 39, 39, 40, 41, 42, 43, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68, 69, 70, 71, 73, 74, 75,
76, 78, 79, 80, 82, 83, 84, 86, 87, 88, 90, 91, 93, 94, 95, 97, 98, 100, 101,
103, 104, 106, 107, 109, 111, 112, 114, 115, 117, 118, 120, 122, 123, 125, 127,
128, 130, 132, 133, 135, 137, 139, 140, 142, 144, 146, 147, 149, 151, 153, 154,
156, 158, 160, 162, 163, 165, 167, 169, 171, 172, 174, 176, 178, 180, 181, 183,
185, 187, 189, 191, 192, 194, 196, 198, 200, 201, 203, 205, 207, 209, 210, 212,
214, 216, 217, 219, 221, 223, 224, 226, 228, 229, 231, 233, 234, 236, 238, 239,
241, 242, 244, 246, 247, 249, 250, 252, 253, 255, 256, 257, 259, 260, 262, 263,
264, 266, 267, 268, 269, 271, 272, 273, 274, 275, 276, 278, 279, 280, 281, 282,
283, 284, 285, 285, 286, 287, 288, 289, 290, 290, 291, 292, 292, 293, 294, 294,
295, 295, 296, 296, 297, 297, 297, 298, 298, 298, 299, 299, 299, 299, 299, 299,
299, 299, 300, 299, 299, 299, 299, 299, 299, 299, 299, 298, 298, 298, 297, 297,
297, 296, 296, 295, 295, 294, 294, 293, 292, 292, 291, 290, 290, 289, 288, 287,
286, 285, 285, 284, 283, 282, 281, 280, 279, 278, 276, 275, 274, 273, 272, 271,
269, 268, 267, 266, 264, 263, 262, 260, 259, 257, 256, 255, 253, 252, 250, 249,
247, 246, 244, 242, 241, 239, 238, 236, 234, 233, 231, 229, 228, 226, 224, 223,
221, 219, 217, 216, 214, 212, 210, 209, 207, 205, 203, 201, 200, 198, 196, 194,
192, 191, 189, 187, 185, 183, 181, 180, 178, 176, 174, 172, 171, 169, 167, 165,
163, 162, 160, 158, 156, 154, 153, 151, 149, 147, 146, 144, 142, 140, 139, 137,
135, 133, 132, 130, 128, 127, 125, 123, 122, 120, 118, 117, 115, 114, 112, 111,
109, 107, 106, 104, 103, 101, 100, 98, 97, 95, 94, 93, 91, 90, 88, 87, 86, 84,
83, 82, 80, 79, 78, 76, 75, 74, 73, 71, 70, 69, 68, 67, 66, 64, 63, 62, 61, 60,
59, 58, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48, 47, 46, 45, 44, 43, 43, 42, 41,
40, 39, 39, 38, 37, 36, 35, 35, 34, 33, 33, 32, 31, 31, 30, 29, 29, 28, 27, 27,
26, 26, 25, 24, 24, 23, 23, 22, 22, 21, 21, 20, 20, 19, 19, 18, 18, 18, 17, 17,
16, 16, 16, 15, 15, 14, 14, 14, 13, 13, 13, 12, 12, 12, 11, 11, 11, 11, 10, 10,
10, 9, 9, 9, 9, 8, 8, 8, 8, 8, 7, 7, 7, 7, 7, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5,
5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 1, 1, 1, others => 0);


constant H_DSP : integer := 639 ; -- 640 horizontal pixels
constant H_FP : integer := 16 ;
constant H_SP : integer := 96 ;
constant H_BP : integer := 48 ;

constant V_DSP : integer := 479 ; -- 480 vertical pixels
constant V_FP : integer := 10 ;
constant V_SP : integer := 2 ;
constant V_BP : integer := 33 ;

signal hPos : integer := 0 ;
signal vPos : integer := 0 ;
signal isVideo : STD_LOGIC := '0' ;
signal isDraw : STD_LOGIC := '0' ;

begin
```

```
RED0 <= '0' ;
RED1 <= '0' ;
RED2 <= '0' ;
RED3 <= '0' ;

BLU0 <= 'isDraw' ;
BLU1 <= 'isDraw' ;
BLU2 <= 'isDraw' ;
BLU3 <= 'isDraw' ;

GRN0 <= isDraw ;
GRN1 <= isDraw;
GRN2 <= isDraw ;
GRN3 <= isDraw ;

h_pos_counter : process (CLK , RESET)
begin
    if ( RESET = '1' ) then
        hPos <= 0 ;

    elsif (rising_edge(CLK)) then
        if (hPos = H_DSP + H_FP + H_SP + H_BP) then
            hPos <= 0 ;
        else
            hPos <= hPos + 1;
        end if ;
    end if ;
end process ;


v_pos_counter : process ( CLK , RESET, hPos )
begin
    if ( RESET = '1' ) then
        vPos <= 0 ;

    elsif (rising_edge(CLK)) then
        if (hPos = H_DSP + H_FP + H_SP + H_BP) then -- last horizontal
            if (vPos = V_DSP + V_FP + V_SP + V_BP) then
                vPos <= 0 ;
            else
                vPos <= vPos + 1;
            end if ;
        end if ;
    end if ;
end process ;


hSyncProc : process (CLK , RESET , hPos )
begin
```

```
    if ( RESET = '1' ) then
        HSYNC <= '0' ;
    elsif ( rising_edge(CLK)) then
        if ( (hPos <= H_DSP + H_FP) OR (hPos > H_DSP + H_FP + H_SP ) ) then
            HSYNC <= '1' ;
         else
            HSYNC <= '0' ;
         end if ;
    end if ;
end process ;

vSyncProc : process (CLK , RESET , vPos)
begin
    if ( RESET = '1' ) then
        VSYNC <= '0' ;
    elsif ( rising_edge(CLK)) then
        if ( (vPos <= V_DSP + V_FP) OR (vPos > V_DSP + V_FP + V_SP ) ) then
            VSYNC <= '1' ;
         else
            VSYNC <= '0' ;
         end if ;
    end if ;
end process ;

videoProc : process (CLK , RESET, hPos, vPos)
begin
    if (RESET = '1') then
        isVideo <= '0' ;
    elsif ( rising_edge (CLK) ) then
        if ( (hPos <= H_DSP) AND (vPos <= V_DSP) ) then
            isVideo <= '1' ;
        else
            isVideo <= '0' ;
         end if ;
     end if ;
end process ;

drawProc : process (CLK , RESET , hPos , vPos , isVideo)
begin
    if (RESET = '1') then
        isDraw <= '1' ;
     elsif (rising_edge (CLK) ) then
        if (isVideo = '1') then
            if ( vPos >= ( V_DSP - (DATA(hPos))) ) then
                isDraw <= '1' ;
            else
                isDraw <= '0' ;
            end if ;
        end if ;
```

```
     end if ;
end process ;
end Behavioral;
```

VGADRIVER_tb.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity VGADRIVER_tb is
end entity;

architecture Behavioural of VGADRIVER_tb is

    constant c_WAIT : time := 20 ns ; -- for 25.17 MHz clock

    signal sim_25MHz : std_logic := '0';
    signal sim_RST : std_logic := '0';
    signal sim_HSYNC : std_logic := '0';
    signal sim_VSYNC : std_logic := '0';

    component VGADRIVER is
            Port (
            CLK : in STD_LOGIC ;
            RESET: in STD_LOGIC ;
            HSYNC : out STD_LOGIC ;
            VSYNC : out STD_LOGIC  ;

            RED0 : out STD_LOGIC ;
            RED1 : out STD_LOGIC ;
            RED2 : out STD_LOGIC ;
            RED3 : out STD_LOGIC ;

            GRN0 : out STD_LOGIC ;
            GRN1 : out STD_LOGIC ;
            GRN2 : out STD_LOGIC ;
            GRN3 : out STD_LOGIC ;


            BLU0 : out STD_LOGIC ;
            BLU1 : out STD_LOGIC ;
            BLU2 : out STD_LOGIC ;
            BLU3 : out STD_LOGIC
            );
        end component VGADRIVER;

        begin
```

```
            UUT : VGADRIVER

            port map(
            CLK => sim_25MHz,
            RESET => sim_RST,
            HSYNC => sim_HSYNC,
            VSYNC => sim_VSYNC
            );

            p_CLK_GEN : process is
                begin
                    wait for c_WAIT;
                    sim_25MHz <= '1';
                    wait for c_WAIT;
                    sim_25MHz <= '0';
                end process;

            end Behavioural;
```

gaussian.py

```
import math
import matplotlib.pyplot as plt


def gauss(x, peak, center, width, tam=True):
    a = peak
    b = center
    c = width

    us = - (((x-b)**2) / (2*c*c))
    result = a * math.exp(us)

    if tam:
        return int(result)

    return result


PEAK = 300
CENTER = 320
WIDTH = 100
LIMIT = 640

x = []
y = []

for i in range(LIMIT):
    y.append(gauss(i, PEAK, CENTER, WIDTH))
```

```
    x.append(i)

plt.bar(x, y)

plt.show()

print(y)
```