HY430 – Digital Systems Lab

Lab assignment 3

4/12/2019

Βαγενάς Αναστάσης

This report describes the implementation of the 3rd laboratory work where it is divided into 3 parts and in each of them we show its implementation in code / logic, the verification of its operation and the results of the experiment on the FPGA.

Prelude

The goal of the lab exercise is to implement a VGA driver, so that a screen is driven and an image is displayed on it. This image will be stored in FPGA RAM and the driver will display it continuously on the screen.

Part A

Implementation/Verification

In Part A, the goal is to implement FPGA BRAM as VRAM for VGA Drive. To achieve this we will use ready-made Xillinx primitives for Spartan 3E and all we have to do is declare the connections as shown below:

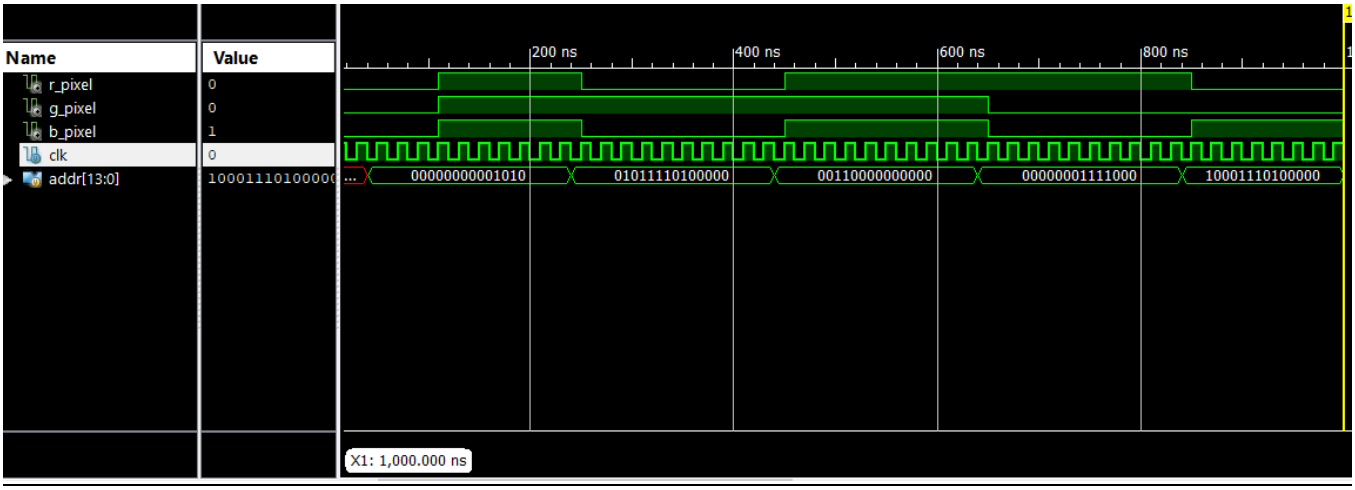
```
module vram_unit (clk,addr,r_pixel,g_pixel,b_pixel);
.....
   red_vram (
     .DO(r_pixel),       // 1-bit Data Output
     .ADDR(addr),       // 14-bit Address Input
     .CLK(clk),        // Clock
     .EN(1'b1),        // RAM Enable Input
     .SSR(1'b0) ,      // Synchronous Set/Reset Input
     .WE(1'b0)
.....
endmodule
```

Above is the 16kB BRAM for red (respectively for blue and green). Since we only want to read from memory, lower WE and SSR to 0, leave EN at 1 to be ready to read.

With addr as input to the memory we declare which address we want to read and as output we have the corresponding color of the pixel (if it is ΄red΄ RAM then r_pixel).

In addition, we use the .INIT command to initialize the memory data and initialize it with the test image.

For verification, which is done only at the behavioral level, we check the pixel values for different addresses.



In the image of the simulation it seems that when we give different values to the address (10,6048,3072,120,9120) and we have white, green, white, red and blue.

Part B/C

Implementation

In the second part and the third part we implement HSYNC and VSYNC which otherwise have the same form with the only difference being the duration times. In addition we calculate the times when the screen is active for color display and the HPIXEL, VPIXEL counters that show at any time which memory address is active.

First, we use counters with which we measure the necessary times needed. For HSYNC and active screen time we use the h_count counter. Respectively for VSYNC v_count.

```
//hsync is active low
assign HSYNC = ((h_count<HSYNC_PULSE_WIDTH_B))?1'b0:1'b1;
//Enable RGB for DISPLAY_TIME_D
assign h_rgb_enable = (h_count >LOW_RGB && h_count<UPPER_RGB)?1'b1:1'b0;endmodule
```

H_count starts and counts when VSYNC is active. When it has counted time A of HSYNC then it will return to 0 and start counting again.

```
always @ (posedge clk or posedge reset) begin
.....
    if(h_count == SCANLINE_TIME_A || VSYNC==0)
        h_count<=0;
    else
        h_count<=h_count+1;
.....
```

In a similar way we have the function of VSYNC which has the v_count counter, with the only difference being that it has different times which it counts but also that it is reset only when it has reached its maximum value (time O).

For addressing, ie how to access the memory we use the HPIXEL and VPIXEL counters, which measure the vertical and horizontal pixels that we display at any time.

To change the counters themselves we use the auxiliary counters h_pix and v_pix.

Initially h_pix is an auxiliary counter that counts to 9 and returns to 0. It helps us to separate the part where rgb is active inside the HSYNC pulse. Since we want to show 128 pixels in a hsync and the width of rgb is 25.6us we have:

25.6 us/128 =200ns =10 cycles (hence the fact that we count to 9)

```
always @ (posedge clk or posedge reset) begin
.....
else if(h_rgb_enable == 0)//set to 0 when rgb inactive
    h_pix<=0;
else begin
    if(h_pix == 9) //max value of h_pix
        h_pix<=0;
    else
        h_pix<=h_pix+1;//increment
.....
```

The v_pix counter is the counter that multiplies the line we show by 5, that is, v_pix just counts 5 hsync pulses. We keep it at 0 as long as the rgb of VSYNC is at 0.

```
always @ (posedge clk or posedge reset) begin
.....
 else if(v_rgb_enable == 0 )begin
      v_pix<=0;
 end
 else begin//do something after every hsync
   if(h_count == SCANLINE_TIME_A) begin
       if(v_pix == 4)
           v_pix<=0;
       else
           v_pix<=v_pix+1;
.....
```

Now using the auxiliary counters we make the shifts in HPIXEL and VPIXEL.

```
always @ (posedge clk or posedge reset) begin
 if(reset == 1)
  HPIXEL<=0;
 else if(v_rgb_enable == 0 ||h_rgb_enable == 0)//deacttivate when video is inactive
           HPIXEL<=0;
 else if(h_pix ==9)
           HPIXEL<=HPIXEL+1;
```

```
end
```

HPIXEL changes when h_pix becomes 9 or when we have counted 10 cycles.

```
always @ (posedge clk or posedge reset) begin
  if(reset == 1)
    VPIXEL<=0;
  else if(v_rgb_enable == 0)//reset when vsync rgb is down
            VPIXEL<=0;
  else if(HPIXEL == H_RESOLUTION && v_pix==4 && h_pix==9) begin
     if(VPIXEL==V_RESOLUTION)//we reached the end line
               VPIXEL<=0;
             else
               VPIXEL<= VPIXEL+1;//else increment
      end
end
```

VPIXEL changes when HPIXEL has reached the last position and is ready to return to 0 (hence h_pix = 9) and v_pix is equal to 4, ie we have reached the point where we need to show the next line.

If we are not in the last line or we are out of the active rgb then we add one to VPIXEL and go to the next line.

After that if we have HPIXEL and VPIXEL the address we read from memory is addr = {VPIXEL, HPIXEL}

In the last module, vga_driver, which is also the top level module after we export the addr based on the above, we set the ram_enable signal which is also the signal which activates the color projection from the ram.

Based on this we assign the output signals VGA_RED, VGA_GREEN, VGA_BLUE, which are the ram outputs based on the address we give it.

Below is a piece of code:

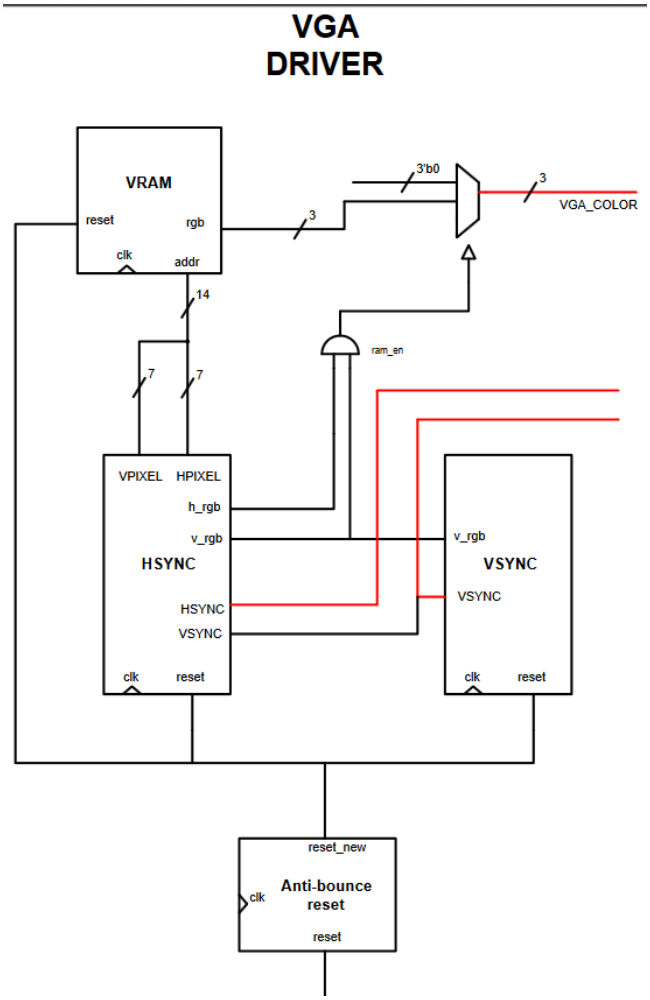
```
//ram enable => when to read from ram (active during display time)
assign ram_enable = v_rgb_enable && h_rgb_enable;
//ram addr is the concat of VPIXEL and HPIXELS
assign addr = {VPIXEL,HPIXEL};
assign VGA_VSYNC = VSYNC;
//assigning color output based on ram enable signal
assign VGA_RED = (ram_enable==1)?r_pixel:1'b0;
assign VGA_GREEN = (ram_enable==1)?g_pixel:1'b0;
assign VGA_BLUE = (ram_enable==1)?b_pixel:1'b0;
```

Below is the table with the parameters-times used for the HSYNC and the VSYNC module.

HSYNC			VSYNC	
SCANLINE_TIME_A	1.599		TOTAL_FRAME_TIME_O	833.499
HSYNC_PULSE_WIDTH_B	192		VSYNC_PULSE_WIDTH_P	3.200
BACK_PORCH_C	96		BACK_PORCH_Q	46.400
DISPLAY_TIME_D	1.280		ACTIVE_VIDEO_TIME_R	768.000
FRONT_PORCH_E	32		FRONT_PORCH_S	16.000
H_RESOLUTION	127		LOW_RGB	46.400
V_RESOLUTION	95		UPPER_RGB	817.601
LOW_RGB	287			
UPPER_RGB	1.568			

All parameters have been calculated based on the FPGA clock which is at 50 MHz and is essentially the number of cycles needed to have each time period based on the requirements of the exercise (the table with the times given to us in the assignment paper).

Below is a simplified schematic of the implementation described (in red we symbolize the outputs of vga_driver) :



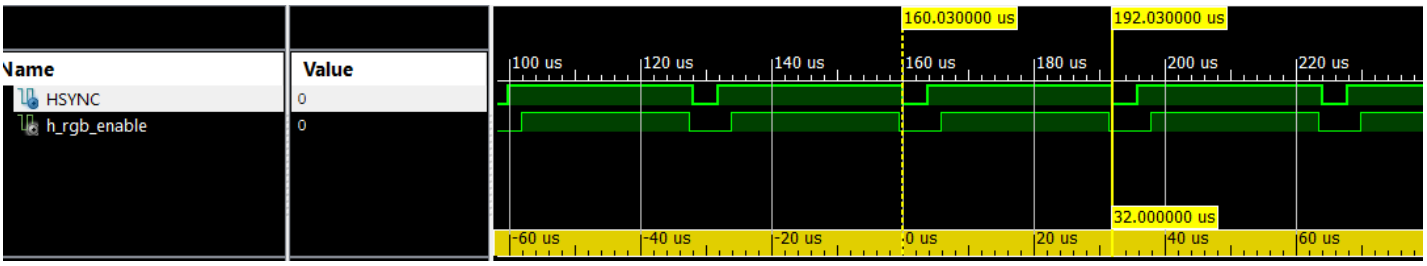
Verification

To verify the correct operation of the circuit we use 3 testbench of which:

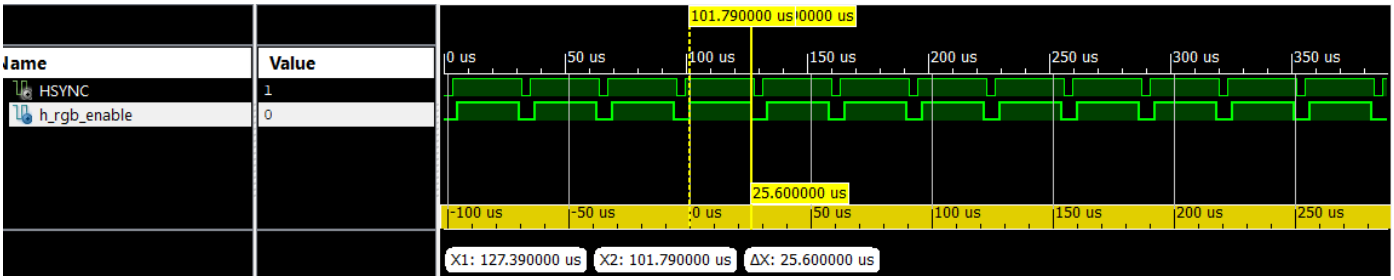
- The 2 control HSYNC and VSYNC separately and we measure manually if the duration times are correct
- The 3rd controls the entire operation of the vga_driver circuit where we have instantiate all the modules

HSYNC/VSYNC check

Indicatively we count some times for HSYNC.

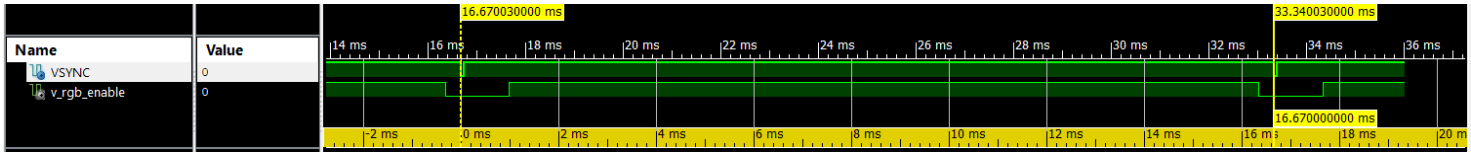


In the 1st image we measure the time SCANLINE_TIME_A which must be 32μs as shown.

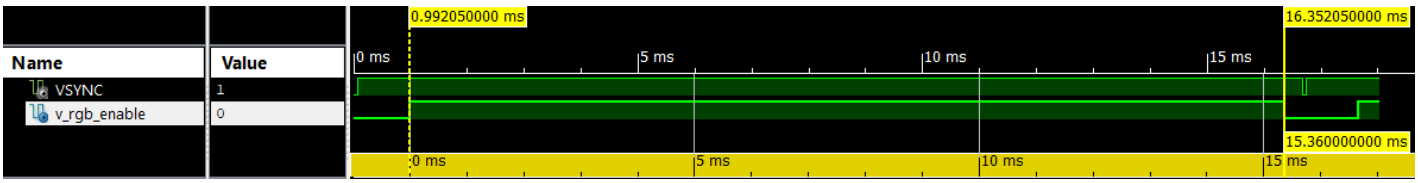


In the 2nd image we measure the time DISPLAY_TIME_D which should be 25.6μs as shown.

We also count some times for VSYNC.



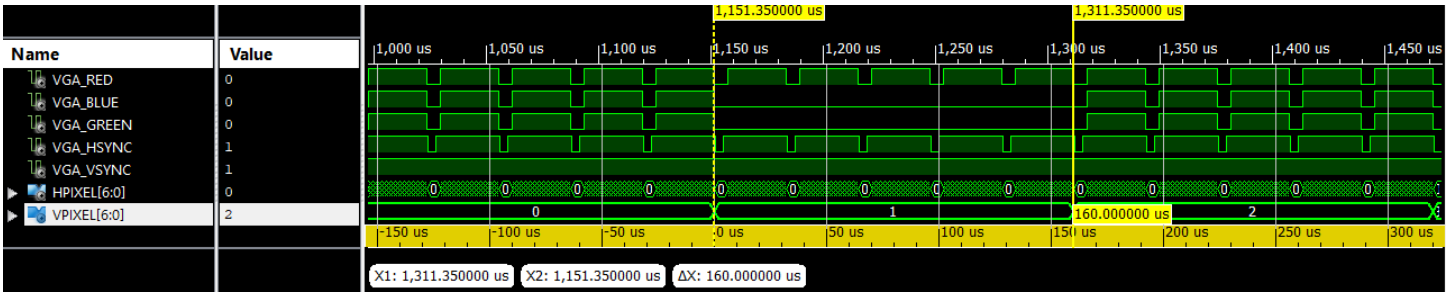
In the 1st image we measure the time TOTAL_FRAME_TIME_O which should be 16.67ms as shown.



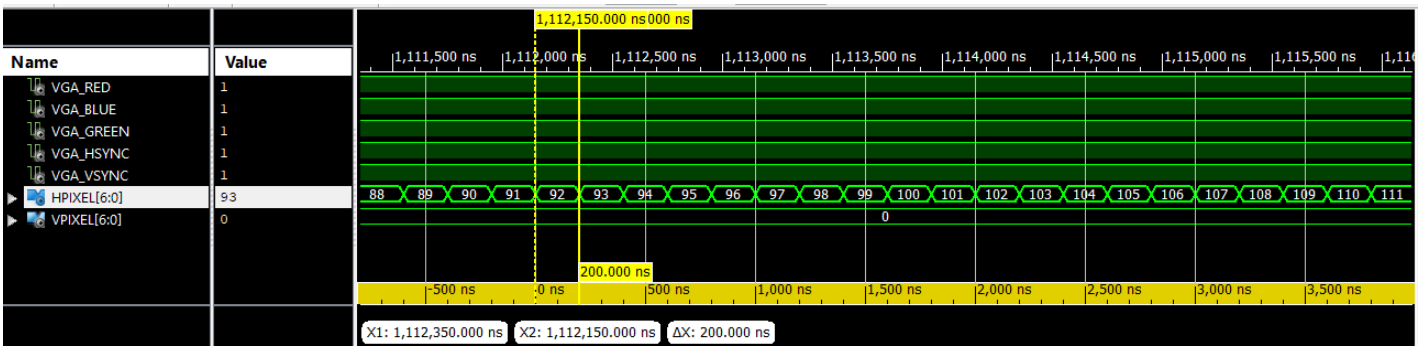
In the 2nd image we measure the time ACTIVE_VIDEO_TIME_R which should be 15.36ms as it is.

Vga_Driver check

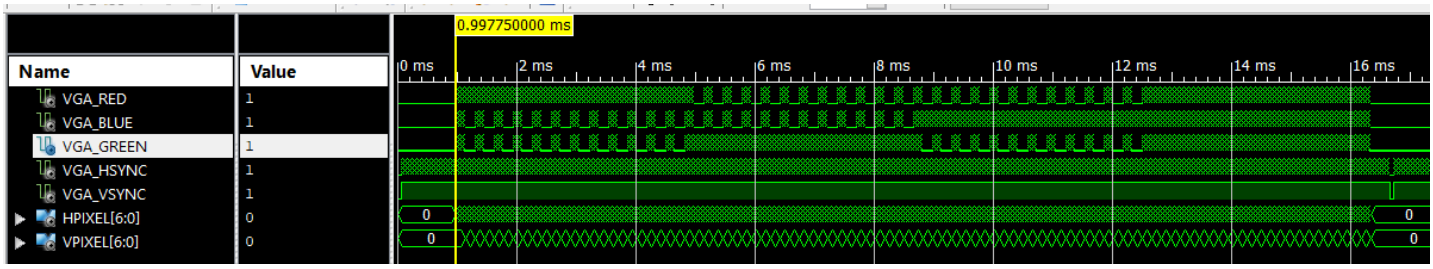
In vga_driver we check if the pixels change in the right way and we display the appropriate color on the screen at the right time.



In the 1st picture we show how the VPIXEL is switched per 5 HSYNC pulses (ie we multiply the horizontal resolution 96 by 5).



In the 2nd image we show how the HPIXEL is refreshed every 10 cycles (200ns duration) and how they are shared within the active display time of HSYNC.

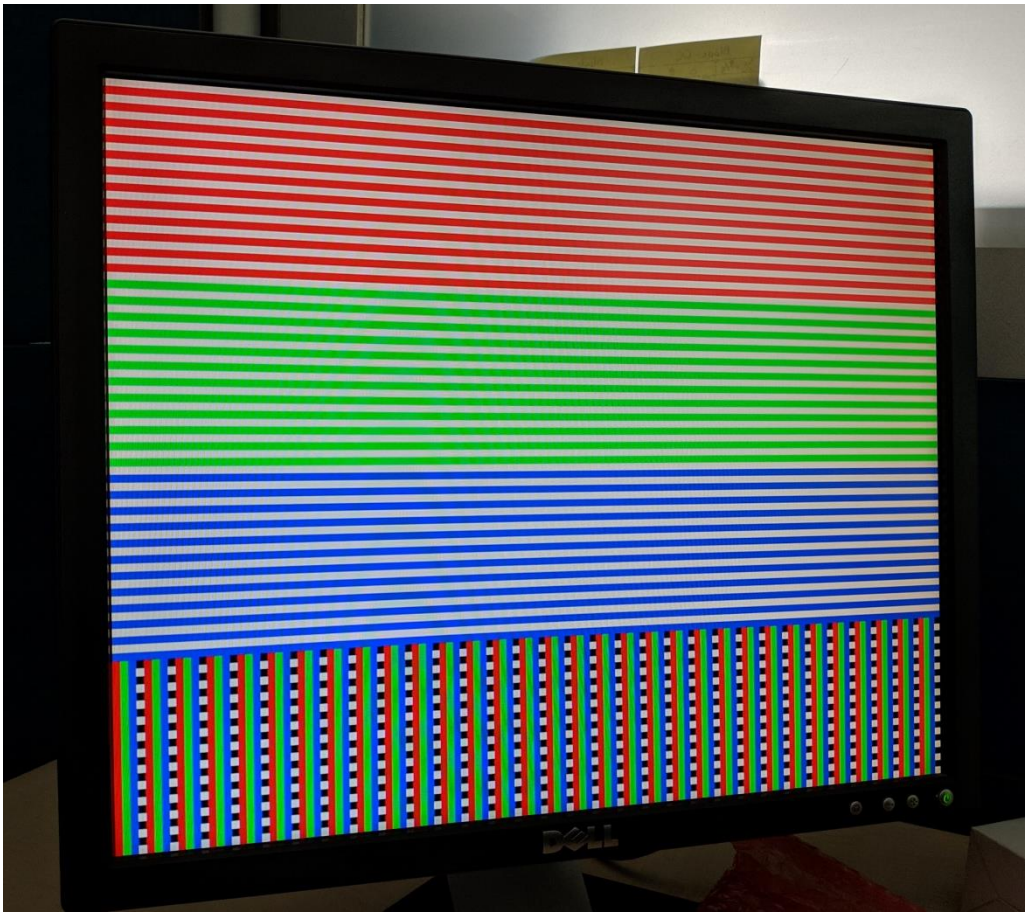


In the 3rd image we show how the color changes throughout the frame display time of VSYNC. As it seems we start from white-red stripes, we go to white-green stripes, then to white-blue and finally we go to the vertical stripes that are not easy to see in the picture (they will be seen in the picture of the experimental implementation.

Experiment/Final Implementation

For the experiment we use the Spartan 3 board of Xillinx and after we have assigned the signals to the appropriate pins of FPGA we proceed to the export of bitfile with which we will program the FPGA.

For the successful operation of the laboratory exercise 3 repetitions were needed. All were done due to the non-synchronization of HSYNC and VSYNC resulting in the appearance of a video (greater explanation in the conclusions).



The result of the successful final experiment, which is the appearance of the above pattern on the screen.

Conclusion

In this lab exercise we wanted to implement a VGA driver in circuit form in Verilog.

The successful operation of the circuit required 2 laboratories and several tests to confirm the.

The main issue presented was the asynchronization of HSYNC in relation to VSYNC in the long run (after the 2nd frame) with the result that in each change of VSYNC the 1st line of the frame is different. At the beginning and the end of the frame the HSYNC pulses were essentially intermittent and none at all.

This resulted, based on the entire implementation of HPIXEL and VPIXEL addressing, to have a video instead of a static image during the experiment.

Thus, with behavioral simulations we tried to find the cause mentioned above and what are the appropriate conditions for correct change of HPIXEL and VPIXEL addresses in combination with the synchronization of HSYNC with VSYNC.

The correct change of addresses was checked with hpixel and vpixel, while the synchronization of HSYNC was done by resetting it as long as VSYNC is also 0. Thus, in this way we have synchronization in each iteration of VSYNC and correction of the problem.

Finally, with the help of the laboratory assistants, our theoretical knowledge and the practical test on the board, the 3rd laboratory exercise was successful..