ECE333 - Digital Systems Lab

– Lab Report –

Lab 3 - Video Graphics Array (VGA) Driver

submitted by

Liaskonis Spyridon

December 17, 2022

Liaskonis Spyridon
sliaskonis@uth.gr
Student ID: 03381

Lab Report Summary

The following report is divided into five sections. The first section is an introduction to the aim of this project and how it was accomplished. The sections two, three and four are more detailed paragraphs about each part of the project. The project was divided into three parts and so each paragraph about these parts focuses on the implementation of the VGA Driver, how we verified the implementation and also the experiments that were conducted on the FPGAs. The last section is the conclusions in which we talk about the progress of the project in fields such as design, verification and testing as well as difficulties that may occurred and how we overcame them.

Contents

1 Introduction

The aim of this project is the implementation of a Video Graphis Array (VGA) Driver for the Nexys A7 board. The implementation is divided into three parts, each one developed separately. Each part focuses on a different objective, however all the parts are needed for the full implementation of the VGA Driver. Here is a quick summary of the objective of each part and how this objective was achieved:

- Part A: Implementation of a Video RAM, where the data (photo) that we want to display is stored. The video RAM was implemented using an 18K-bit Configurable Synchronous Block RAM (RAMB12E1) provided by Xilinx. The block ram here was instantiated 3 times, one for each colour (Red, Green, Blue).
- Part B: Implementation of the HSYNC Controller and the Horizontal Pixel Counter. The HSYNC Controller was responsible for the horizontal synchronization of the screen, meaning that its used to cue the monitor that a new line of pixels should be displayed. The controller here was implemented using an FSM and a counter that was responsible for providing the correct timing to the HSYNC signal. Next the Horizontal Pixel Counter was used for displaying the currently active pixel of the VRAM by providing a 7-bit value called HPIXEL. This value is representing the horizontal coordinate of the pixel on the screen and it used (together with the VPIXEL value from the vertical pixel counter) to find the address of that active pixel on the VRAM. The horizontal pixel counter here was implemented using a 5-bit counter that counted the time each pixel should stay on the display.
- Part C: Implementation of the VSYNC Controller and the Vertical Pixel Counter. The VSYNC Controller is responsible for the synchronization of the refresh rate of the monitor, meaning that its used to cue the monitor that a new frame should be displayed. The implementation of the VSYNC controller is the same as the one for the HSYNC controller since they do the same job with the only difference being that they have different timings (which leads to using different size counters for each controller). Then the vertical pixel counter is also used and implemented the same way as the horizontal pixel counter. Combining the two pixel counters together we get the address of the active pixel from the VRAM.

2 Part A - Video RAM (VRAM) implementation

In this part we talk about the implementation of the VRAM. Here is the dataflow for the VRAM module that is used to store the data we want to display on the monitor:

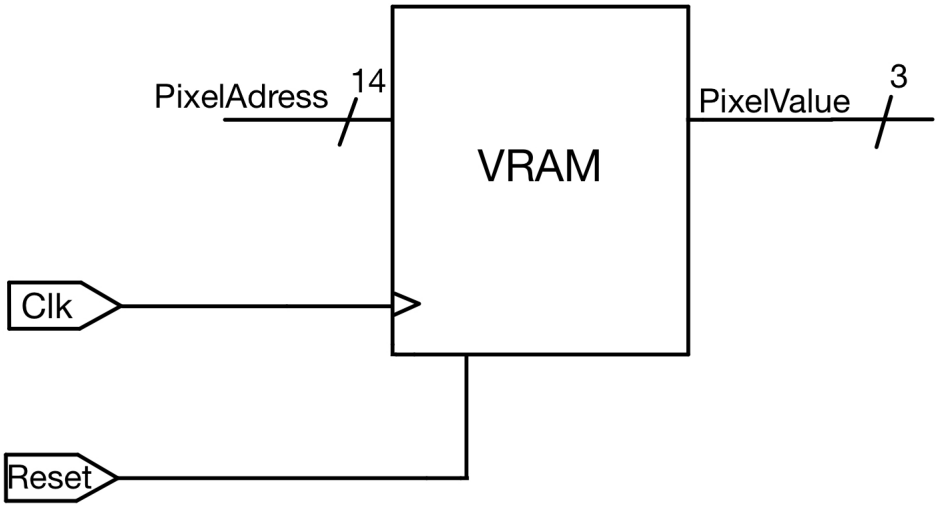


Figure 1: Video RAM (VRAM) Dataflow

The current module has 3 inputs: Clock signal, Reset signal and PixelAdress, a 14-bit value representing the input address from which we read from the VRAM. The output of the VRAM is a 3-bit value called PixelVaue where each bit represents the values of VGA_RED, VGA_GREEN and VGA_BLUE signals respectively.

2.1 Implementation

The resolution of the monitor (and the VGA standard) is 640x480, meaning that we need $640 \times 480 = 307,200$ memory bits to represent a monochrome pixel. However in our case each pixel consists of three different colours, meaning that for a 640x480 resolution we would need $3 \times 307200 = 115.2KBytes$ of memory. For simplicity we re-scaled our photo to a resolution of 128x96 (which only needs 12,288 memory bits for each colour) and used three $16K \times 1$ block RAMs (BRAM), one for each colour. However when displaying the photo later we would re-scale the resolution to 640x480 by sending the same pixels of the photo 5 times.

For the BRAM we used the RAMB12E1 block RAM primitive provided by xilinx. We instantiated and initialized this primitive three times, one for each colour. The block RAMs had the following inputs and outputs that were set according to the colour whose values they were containing:

- DO(output): 1-bit output data containing the value written from the BRAM.
- ADDR(pixelAddress): Input Address from which we read our data.
- CLK(clk): 1-bit input clock.
- DI(0): Input data. Here is set to a fixed value since we only use the BRAM to read data and not write.
- EN(1): 1-bit input RAM Enable. Here it is set to 1 since we always need the BRAM to be enabled.
- REGCE(0): 1-bit input, output register enable. Set to 0 since its not used in the current implementation.
- Reset(reset): 1-bit reset signal.
- WE(0): Input write enable. Also set to a fixed value (0) since we dont use the BRAM for writing but only reading.

2.2 Verification of implementation

In order to verify the implementation of the video RAM we used a testbench in which we instantiated the module VRAM and fed to it different addresses and the checked if the output values of the PixelValue corresponded to the values we were expecting.

The VRAM in this part contained the following photo which is also the photo we want to display:

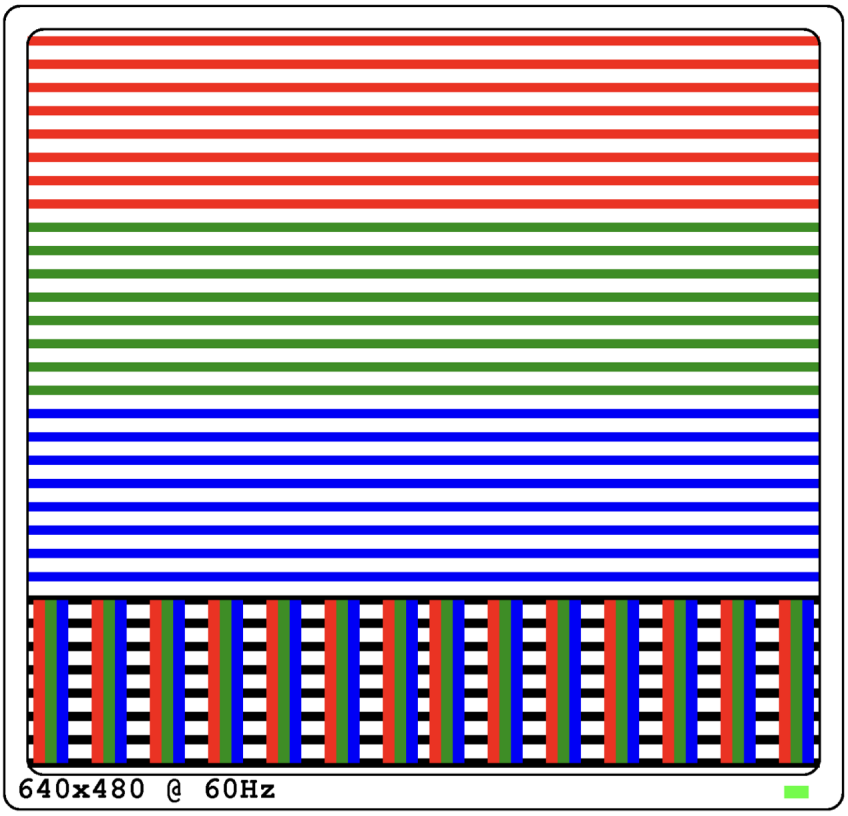


Figure 2: Photo used for the verification of the VGA Driver

Here are the results of the testbench for different addresses:

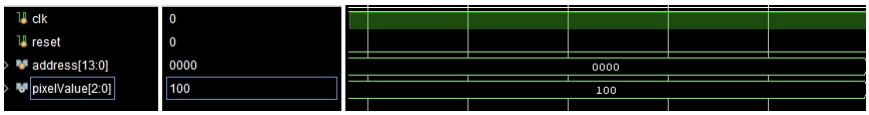


Figure 3: Address = 0 - Colour: RED

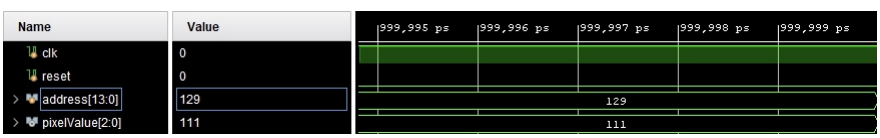


Figure 4: Address = 129 - Colour: WHITE

3 Part B - Implementation of HSYNC controller and Horizontal Pixel Counter

The second part of the project is the implementation of the HSYNC Controller and the Horizontal Pixel Counter. Here is the dataflow of the second part:

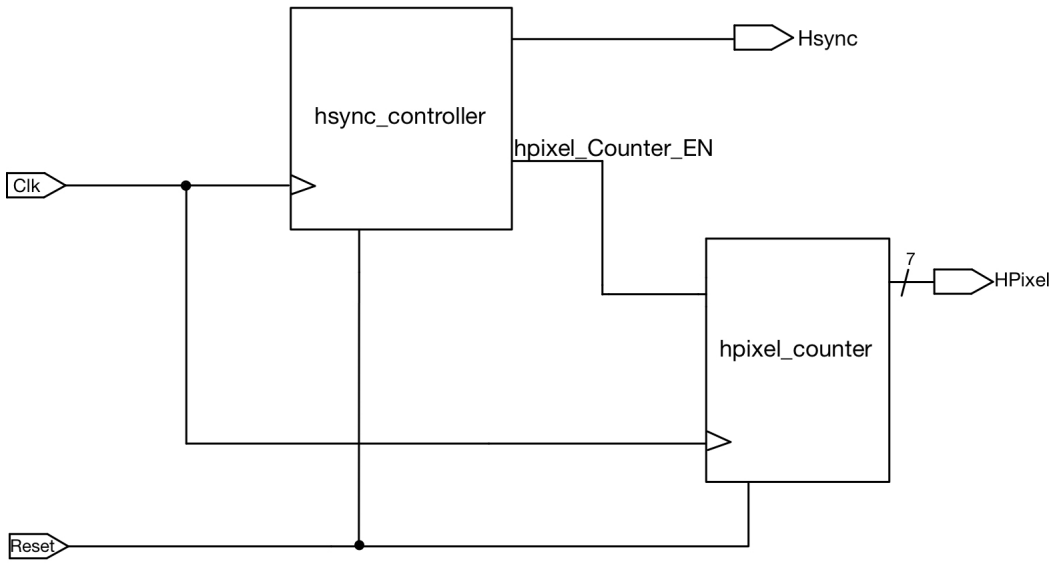
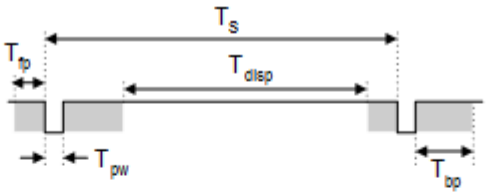


Figure 5: Part B - Dataflow

In this part we only have two inputs: clock and reset signal, and two outputs: HSYNC signal and HPixel.

3.1 HSYNC Controller

The HSYNC Controller is one of the most important parts of the VGA Driver since its responsible for the horizontal synchronization of the screen. Here is a summary of the timing of the HSYNC (as well as VSYNC) according to the VGA standard:



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 6: HSYNC and VSYNC Timings for the VGA Standard

Except the output HSYNC signal the controller also has a 1-bit HPIXEL_CounterEN output signal that's responsible for starting the pixel counter when the controller is at the Display state. In order to provide the right timing to the HSYNC signal we had to create a mealy machine (fsm) with 4 states, one for each interval shown in the Figure 6 (except for the T_s which is the total time of each sync pulse). The fsm here is used to control the HSYNC signal as well as the HPIXEL_CounterEN signal. More specifically the machine starts at the first state which is the Pulse state. There the HSYNC signal is set to 0 (since it's active low). A 12-bit counter is used in order to count the time that the machine should stay at each state. Once the counter counts the right amount of clock cycles that the HSYNC should be 0 then the machine goes to the next state which is the back porch and the hsync singal becomes one. There the counter has already been restarted and is counting again until the machine has to go to the next state. The next state is the Display in which state the data are getting displayed at the monitor. There the FSM sets the HPIXEL_CounterEN signal to 1 which then start the horizontal pixel counter. After the display state the machine goes to the front porch state where it disables the Horizontal pixel counter. The hsync signal is still set to 1 (and its only set to 0 at the pulse state). After the front porch state the machine returns to the pulse state and starts over.

Here is the FSM:

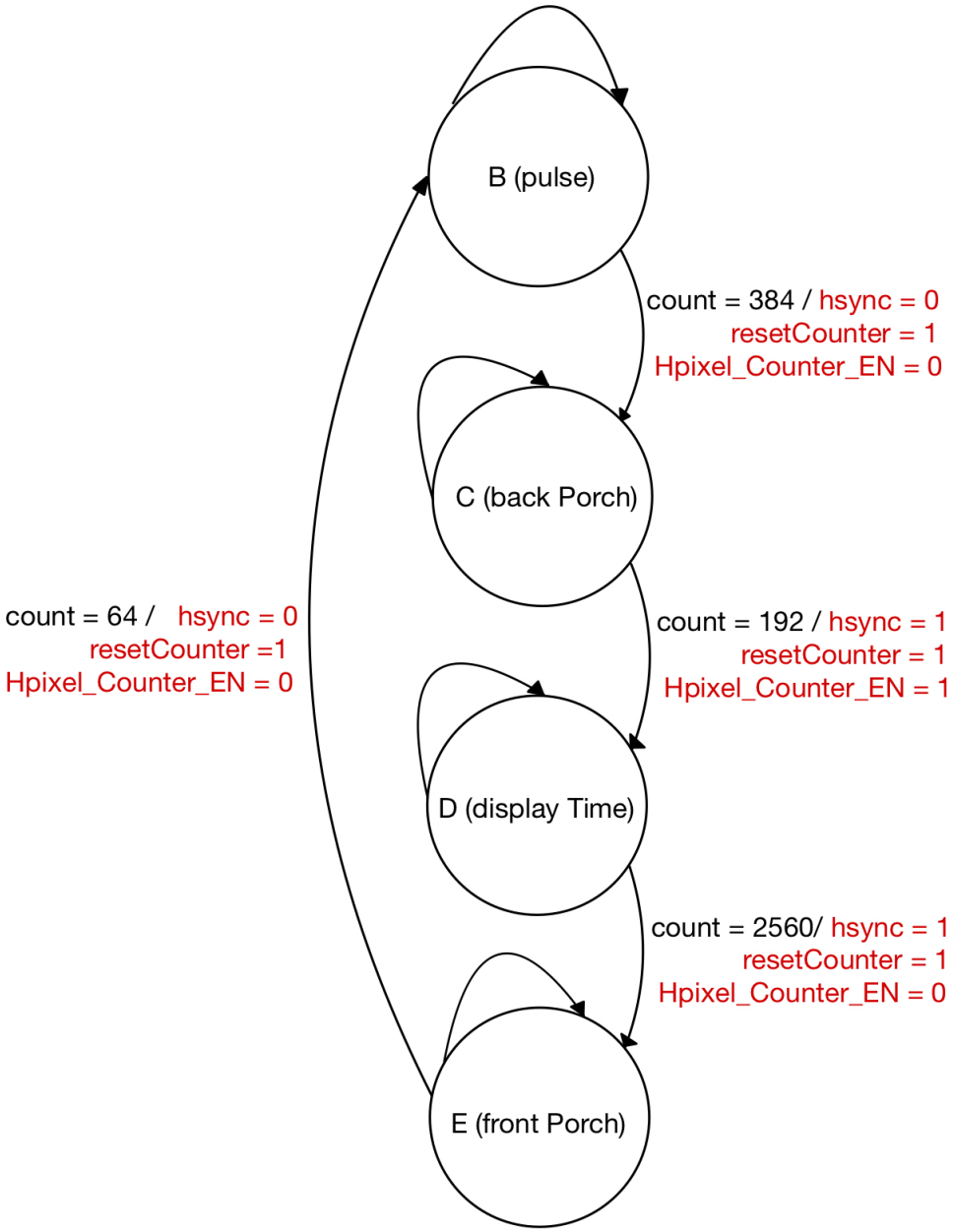


Figure 7: FSM - Part B

The count value is the input of the fsm and the rest (red) signals are the outputs of each state (when the FSM stays at the same state the outputs stay the same so that's why they aren't presented in the graph).

For the implementation of the FSM, two always blocks were used. One sequential that was responsible for assigning the next state to the current state at each posedge clock signal and one combinational that was responsible for controlling the next state's value according to the counter's value.

The 12-bit counter that, as said above, was used for counting clock cycles providing together with the FSM the correct timing to the HSYNC signal, was implemented using one sequential always block. Also, something worth mentioning is that when the counter got the resetCounter signal from the FSM, it got reset with the value 1 and not 0 since the resetCounter signa would come to the counter at the next posedge clock and not on the one where the counter should've been reset.

3.2 Horizontal Pixel Counter

The horizontal pixel counter, as said before, is used for displaying the currently active pixel from the VRAM to the monitor. The pixel counter here has the two inputs mentioned above as well as one more called HPIXEL_CounterEN that is used as an enable signal to the pixel counter. When that signal is set to 1 the counter starts counting.

The pixel counter is active only at the display time state of the hsync signal, thus its active only for 25.6us. In this time we need to send 640 pixels (one line) to the monitor. Thus each pixel should be sent to the monitor for $\frac{25.6}{640} = 0.04us = 40ns$. However, since we have 128 pixels in each line (in the VRAM) we need to send each pixel 5 times so that we re-scale the resolution from 128 to 640. For this reason we send each pixel for a time of $40 \times 5 = 200$ns.

The counter consists of 3 always blocks, two sequential and one combinational. The first always block is a 5-bit counter that is used to count clock cycles. In particular it counts every time 20 clock cycles (200ns, the time we sent each pixel). The second always block implements two multiplexers that control two signals according to the counter's value. The nextPixel signal, a signal that when the counter counts 20 clock cycles becomes one and indicates that we should send the next pixel, and the resetCount signal, that resets the counter to 0 when again the counter counts 20 clock cycles. Finally, the last sequential block implements a second, 7-bit counter, which is the HPIXEL. The HPIXEL value is first set at 0 (when reset is active) and then whenever the signal nextPixel becomes one HPIXEL's value increases by one, indicating the next active pixel from the VRAM. Also whenever the HPIXEL_CounterEN signal is set to 0, the HPIXEL value is also set to 0.

3.3 Verification of implementation

In order to verify the implementation of the hsync controller as well as the horizontal pixel counter we used a module in which we instantiated the top level module called vga_driver that contained the two modules, plus the module vram. Then we reset the circuit and let it run for a certain amount of time. After that we had to check the waveforms and see if the hsync timing was correct. Here are the results of the testbench showing the timing of each state of the hsync counter:

Figure 8: Scanline time

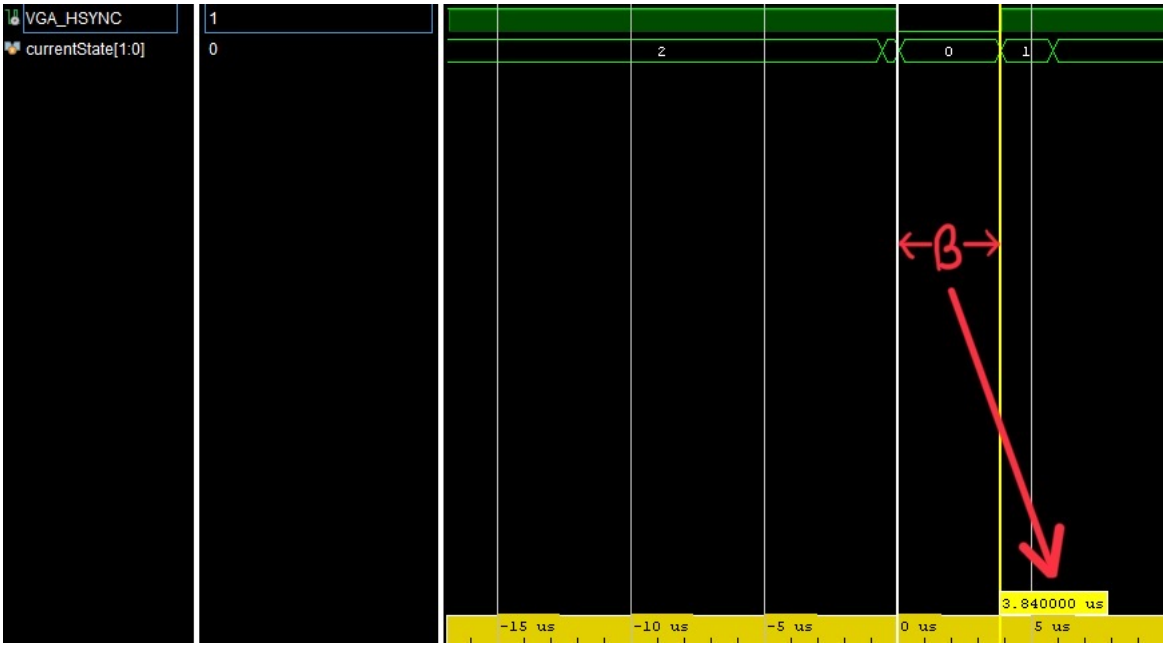


Figure 9: HSYNC pulse

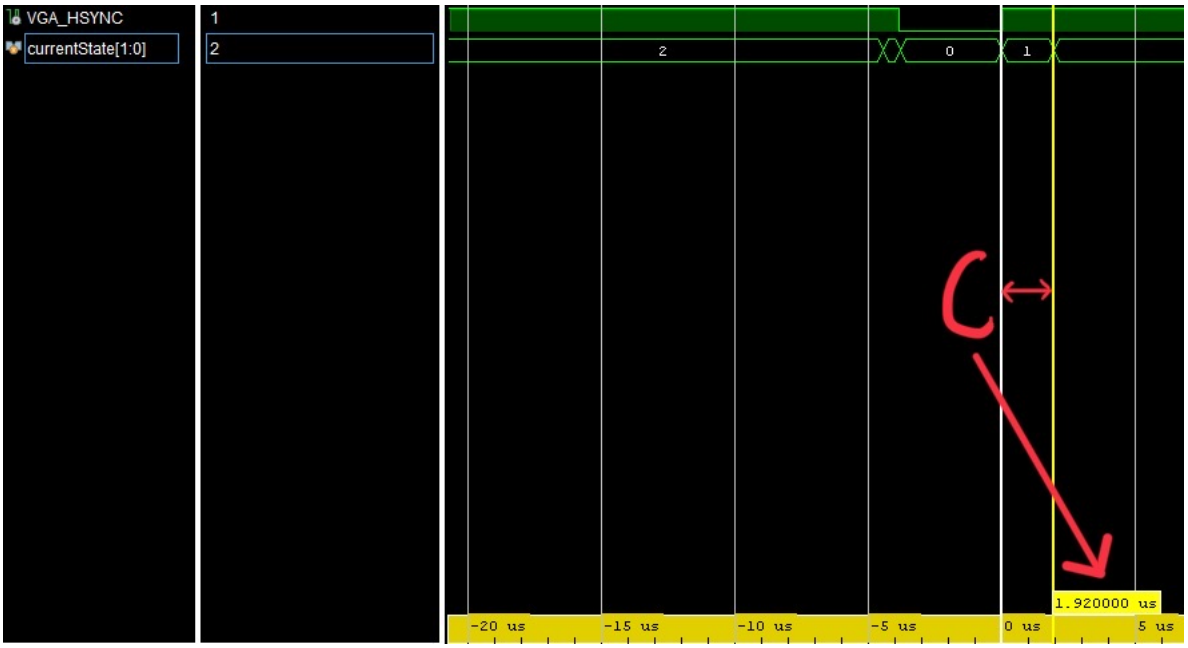


Figure 10: Back porch

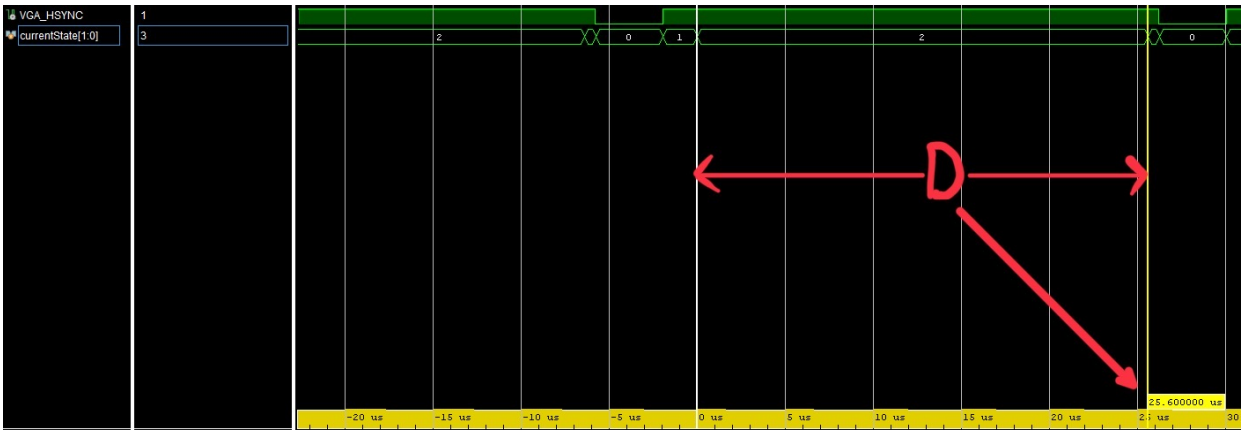


Figure 11: Display time

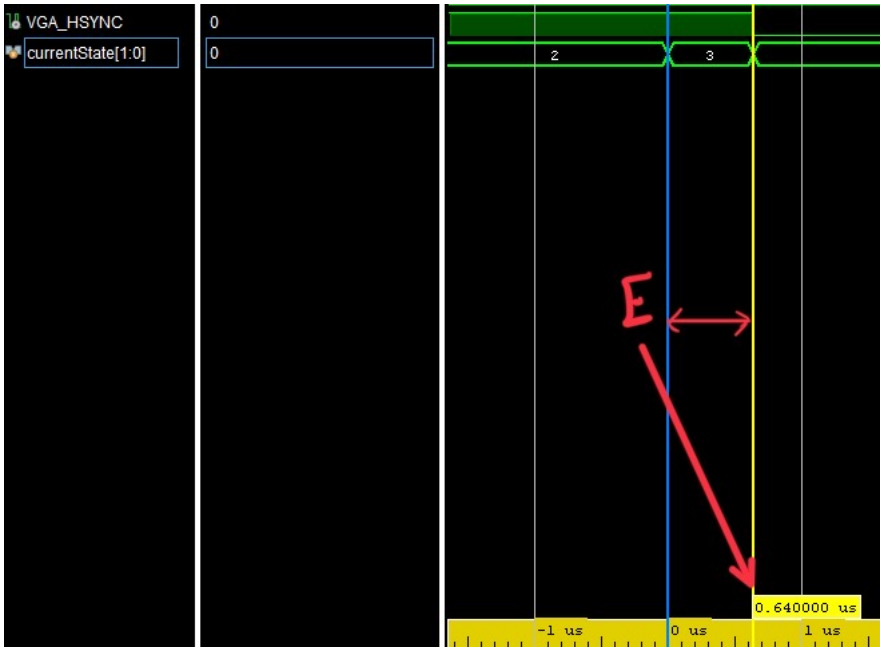


Figure 12: Front porch time

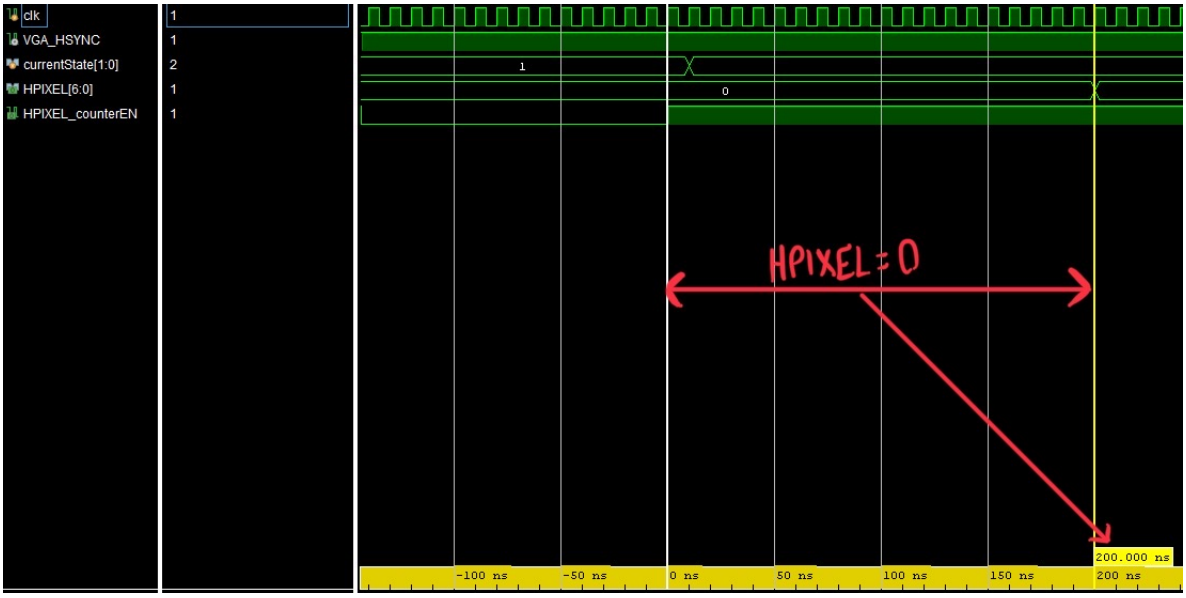


Figure 13: Horizontal Pixel Counter: HPIXEL = 0

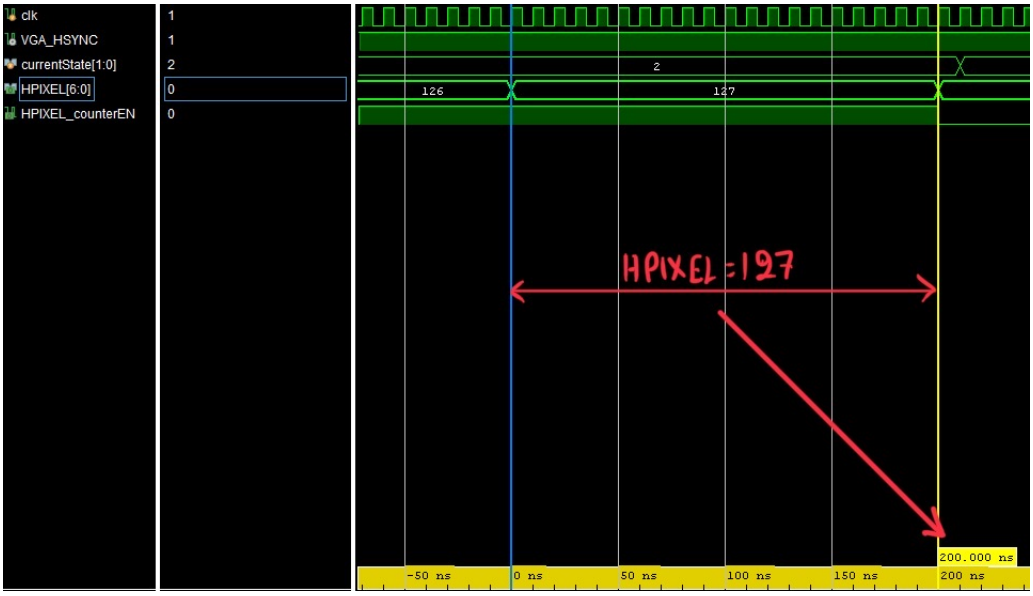


Figure 14: Horizontal Pixel Counter: HPIXEL = 127

From the testbench we see that all the timings are correct. Also it is worth mentioning that since we use a mealy machine the hsync goes to 0 (or 1) one cycle before the machine changes its state. Thus, that's why the HPIXEL = 0 starts one cycle before the HSYNC gets at state two and also why the HPIXEL = 127 finishes one cycle before HSYNC gets at state three (front porch). Even thought they start and finish one cycle before the state two starts and finishes, the timing is still correct since that is happening between all the states and the HSYNC.

4 Part C - Implementation of VSYNC controller, Vertical pixel counter and completion of the VGA driver

The third part of the project is the implementation of the VSYNC Controller and the vertical pixel counter as well as the completion of the VGA Driver, that's achieved by combining everything implemented in the current and previous parts together.Here is the dataflow for part C (the dataflow for the final implementation of the vga driver is presented in section 4.3):

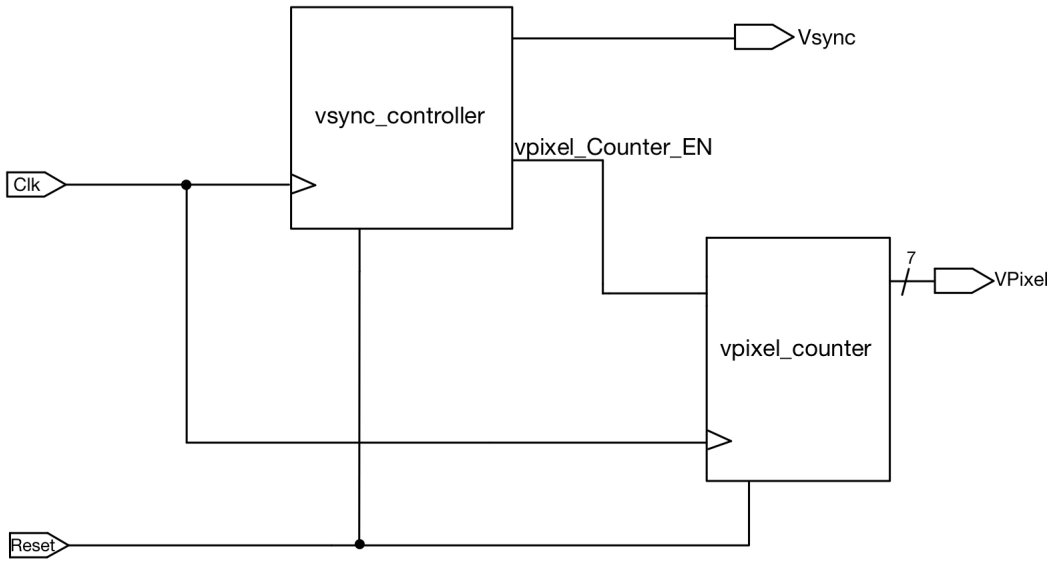


Figure 15: Part C - Dataflow

As you can tell by the dataflow, part C is basically the same modules as part B only this time we needed different (bigger) counters since we had different timings for the VSYNC (than the HSYNC) signal. The code for the part C in the VSYNC Controller as well as in the Vertical Pixel Counter is also the same as part B, thus we wont explain the whole functionality of these two modules again. However, it is worth mentioning that in my implementation the horizontal and vertical pixel counters aren't synchronized as it should be. Meaning that the Vpixel counter is not getting increased every time the hpixel counter sends a whole line to the display. The way the Vpixel counter is getting increased though is the same way the HPixel counter is getting increased in part B. Using the VSYNC timings. More information about the vertical pixels timings will be given in section 4.2.

4.1 VSYNC Controller

The VSYNC Controller is also one of the most important parts of the VGA Driver since its responsible for the synchronization of the refresh rate of the monitor. As said above though, the implementation of the VSYNC Controller is exactly the same with the HSYNC Controller, with the only difference being the timings (which lead to different sized counter). Thus for the functionality and the implementation of the vsync controller refer to the section 3. Also the timings for the VSYNC are shown in Figure 6.

Here is the fsm (mealy machine) that implements the VSYNC Controler (which is also the same one as part b only this time the count input has different values):

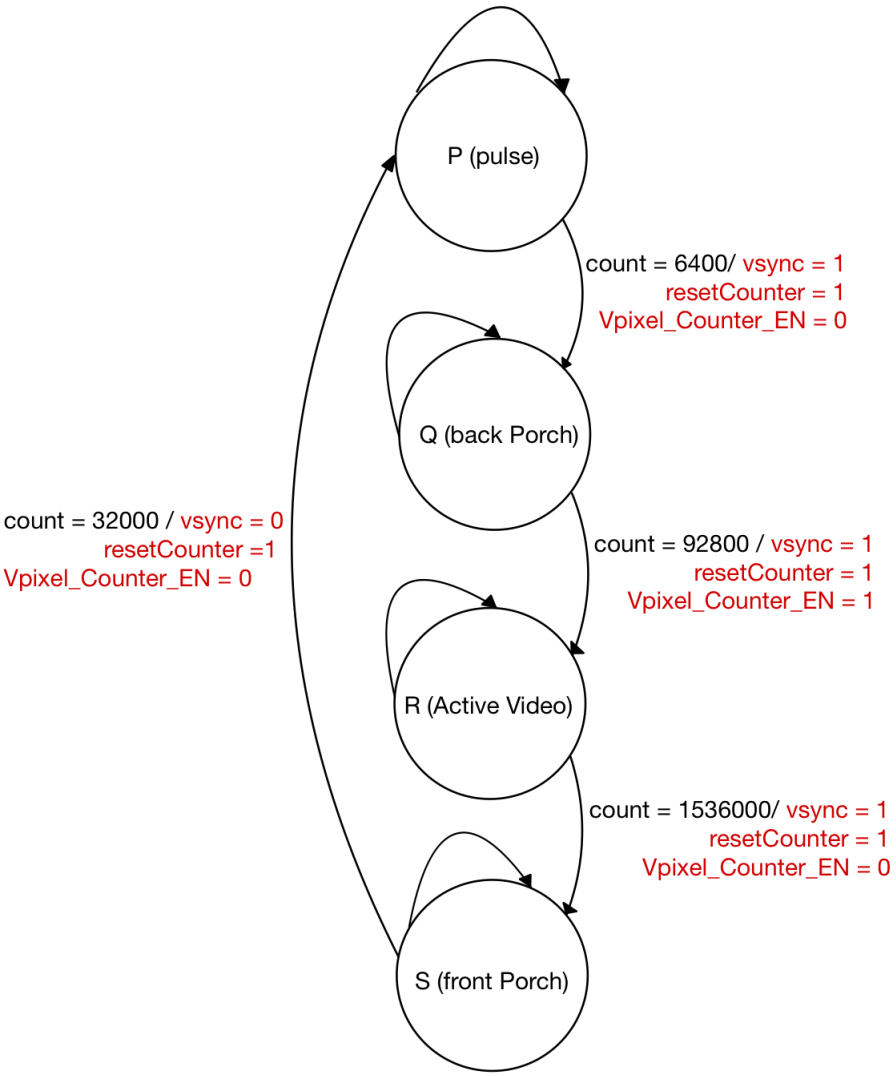


Figure 16: FSM - Part C

4.2 Vertical Pixel Counter

The vertical pixel counter is also used (alongside the horizontal pixel counter) for displaying the currently active pixel from the VRAM to the monitor. The implementation of this pixel counter is the same as the one in subsection 3.2, thus here we are only going to talk about how the timings were calculated in order to be synchronised with the horizontal pixel counter.

The vertical pixel counter is only active when the VSYNC is at the Active Video state. Thus, it's active for 15.36ms. In this time we need to send 480 lines (one frame) to the monitor. So each line should be sent to the display for a time of $\frac{15.36}{480} = 0.032ms = 32,000ns$. However, since we have 96 lines of 128 pixels each in the VRAM we need to send each line 5 times so that we re-scale the resolution from 96 to 480. For this reason we send each line for a time of $32,000 \times 5 = 160,000ns$.

4.3 VGA Driver

The completion of the VGA Driver was achieved by combining all the previous part on the top module called vga_controller. Here is the dataflow of the final implementatio that shows all the inputs (the clk and reset) as well as all the outputs of the vga:

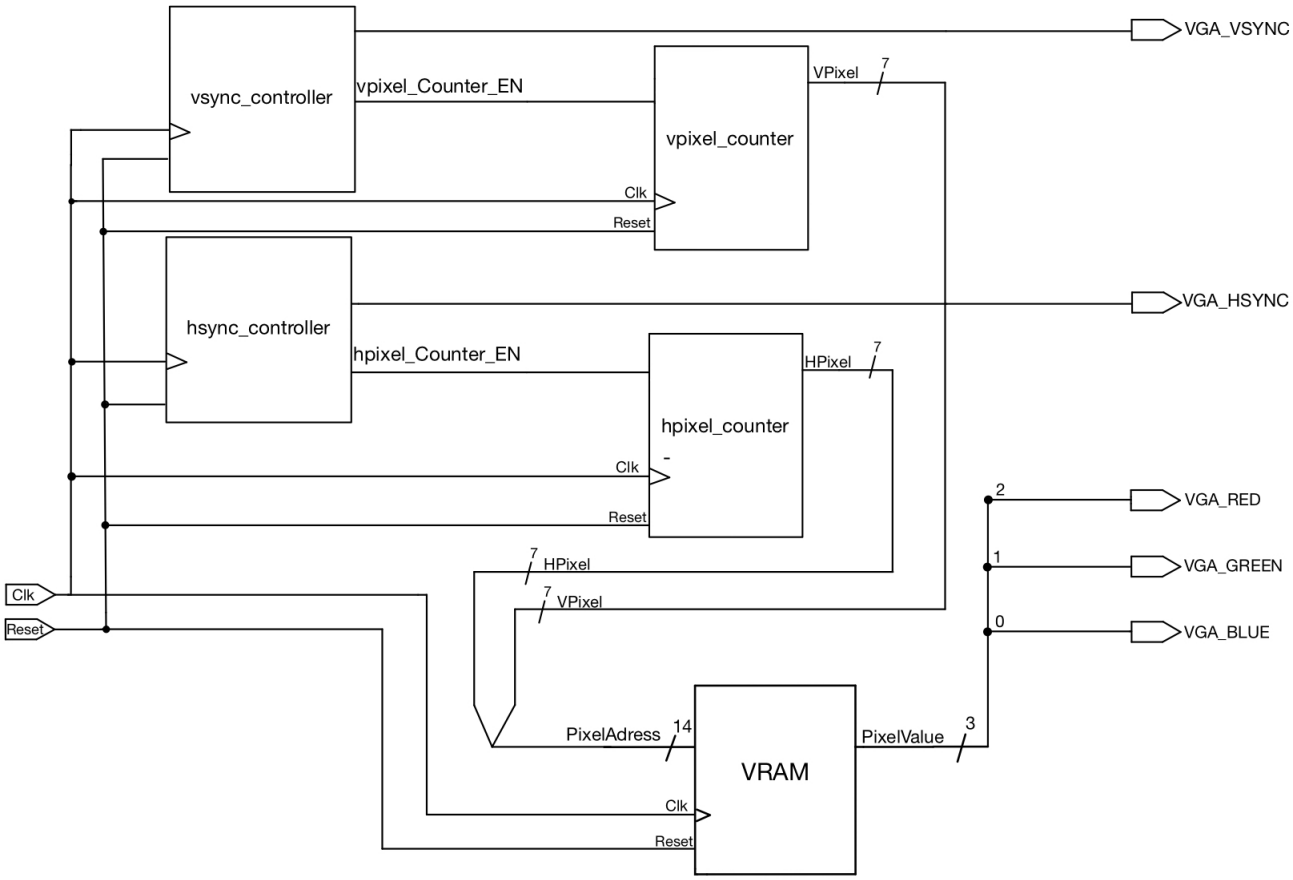


Figure 17: VGA Driver - Dataflow

As you can see on the dataflow above the pixelAddress value is calculated by using the following method:

$$\text{PixelAddress} = \{\text{VPIXEL, HPIXEL}\}$$

4.4 Verification of implementation

In order to verify the implementation of the vsync controller, the vertical counter and the VGA driver in general we used the exact same testbench that we used in part B. Here are the result of that testbench:

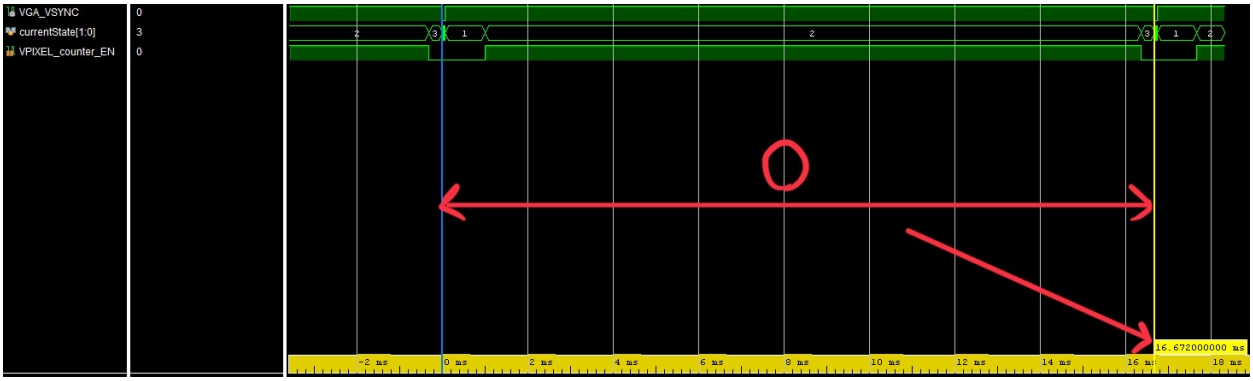


Figure 18: Total frame time

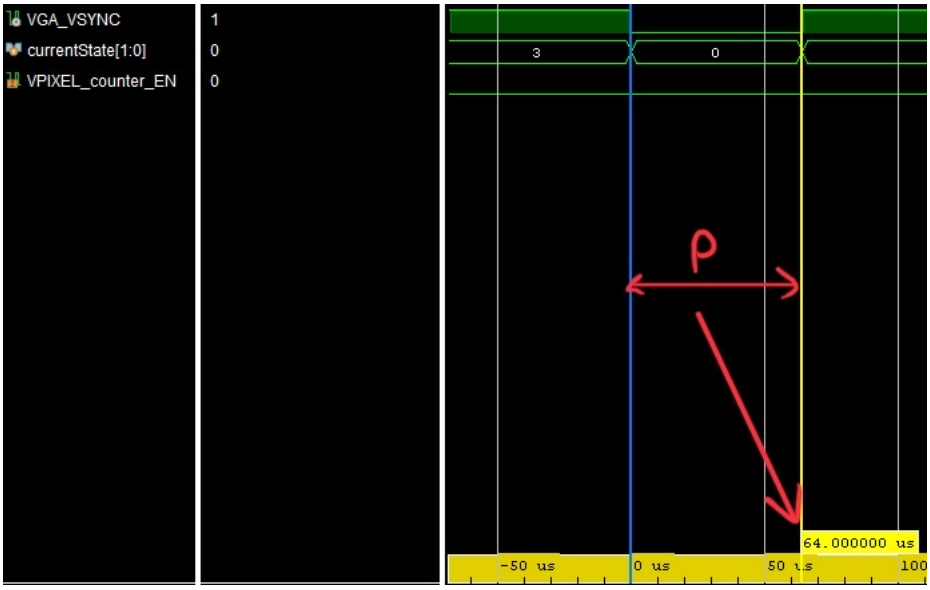


Figure 19: VSYNC pulse

Figure 20: Back porch

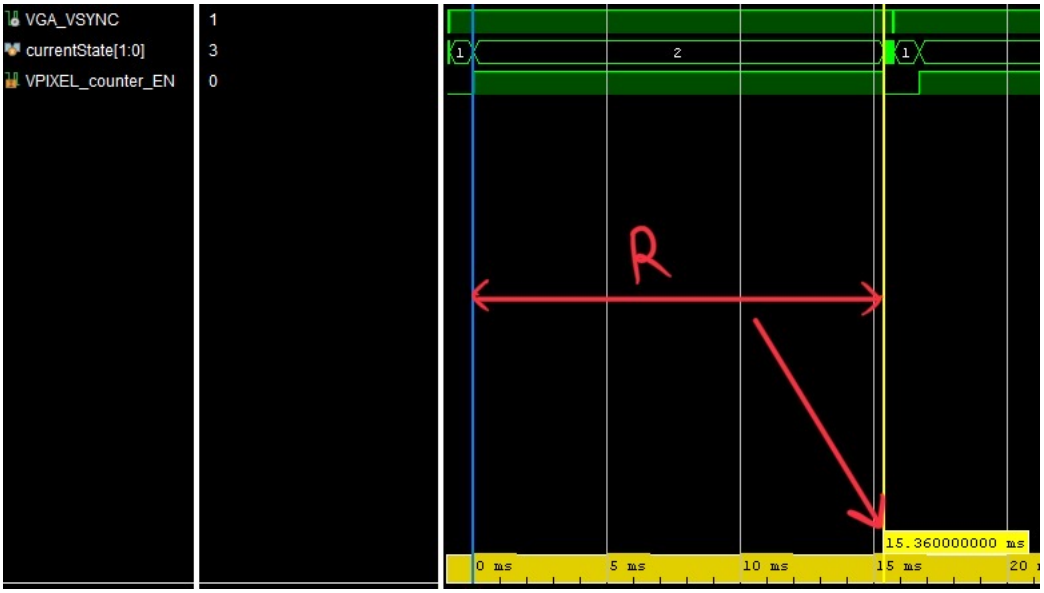


Figure 21: Active video time

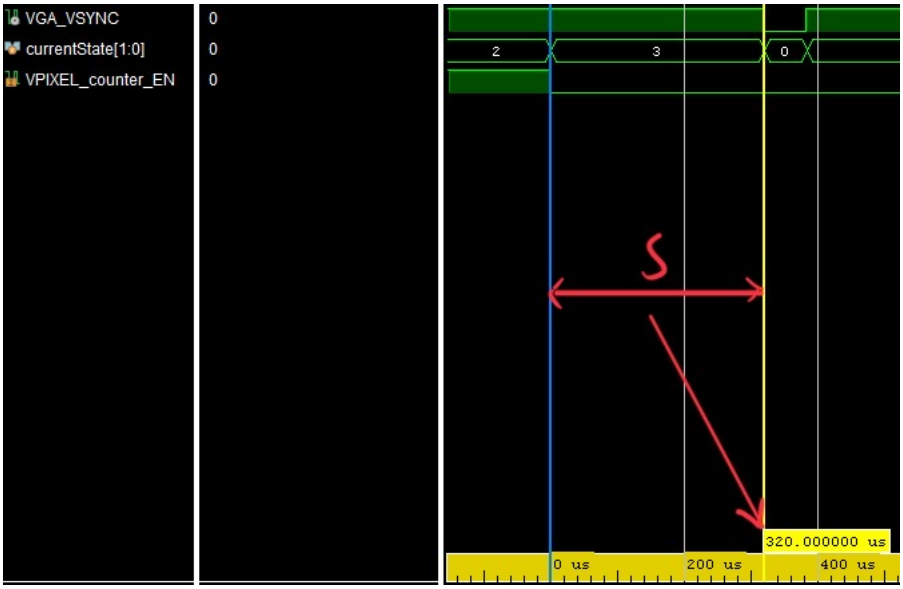


Figure 22: Front Porch

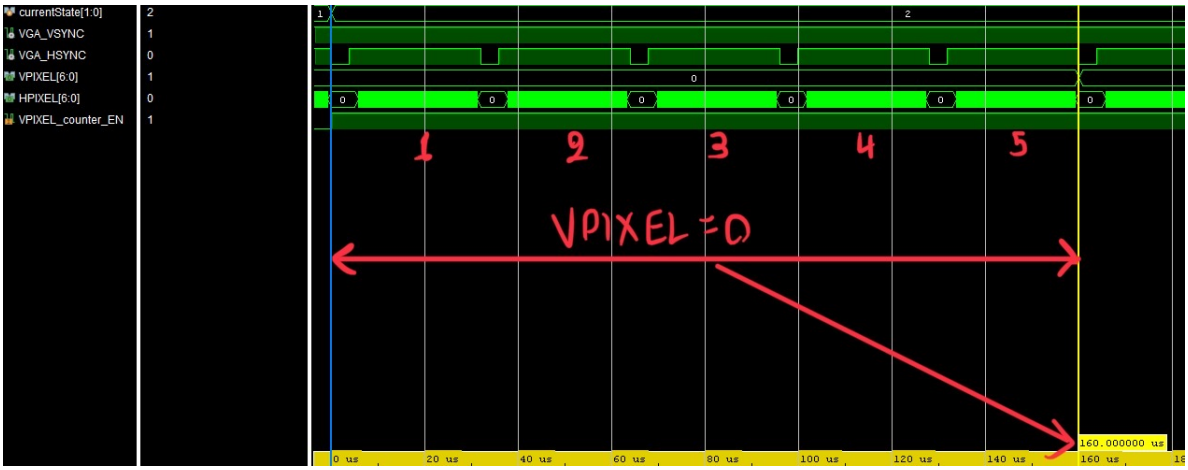


Figure 23: Vertical Pixel Counter: VPIXEL = 0

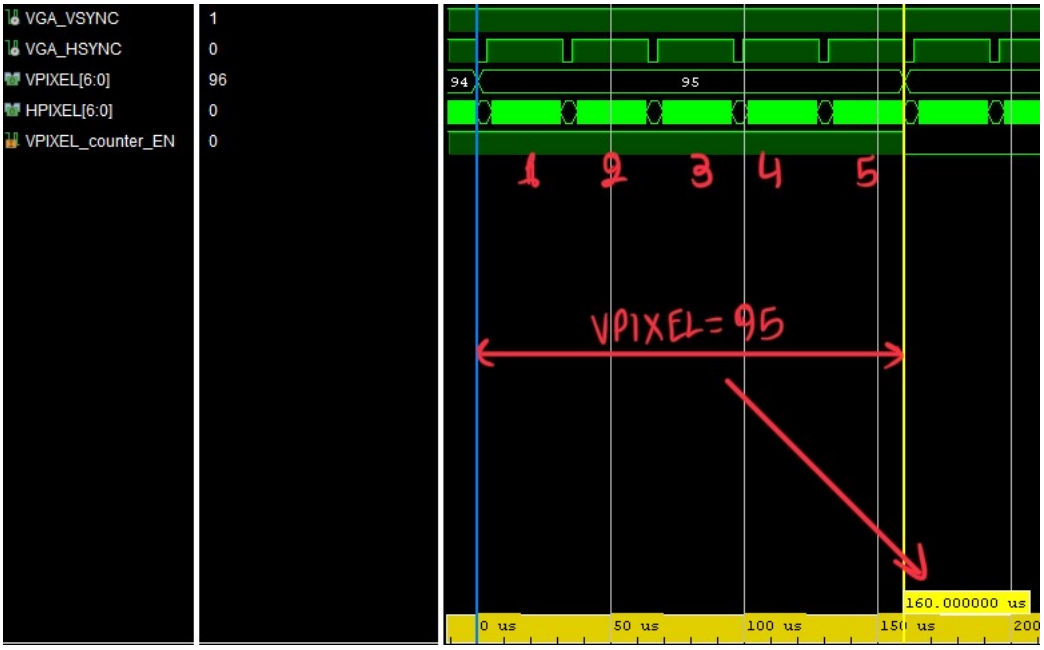


Figure 24: Vertical Pixel Counter: VPIXEL = 95

4.5 Experiment and final implementation

When it came to programming the FPGA, the results weren't the ones i was expecting. The bistream was generated without any errors and the display turned on, however the colours on the 3/4 of the screen weren't right. Here's how the picture on the screen looked:

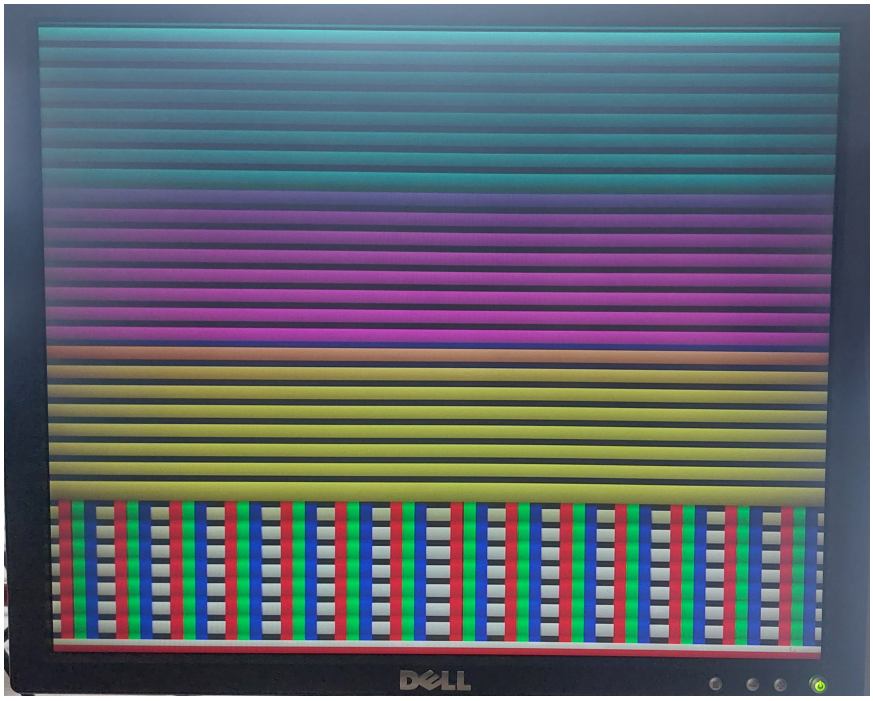


Figure 25: Picture displayed on the screen

In the picture, the white lines on the first 1/4 of the screen became cyan, the red/green/blue became black (in the first 3/4 of the screen), the white lines on the second part became purple and the white lines on the third part became yellow. Also instead of starting with a red line it started with two cyan (white) lines and ended with a red line when it was supposed to end with a white line. My assumption for the cyan, purple and yellow colours was that two colours were overlapping since if we take a look on the colour table we see that red (100) and white (111) if overlap they would give us cyan (011). Same things happens with the other colours as well. After checking all the counters, hsync and vsync signals i couldn't find anything that would cause this since every timing was correct. Also the xdc file was also fine after i checked all the pins and compared them with the ones shown on the nexy A7 manual. The only thing i noticed (from the testbench) was the following:

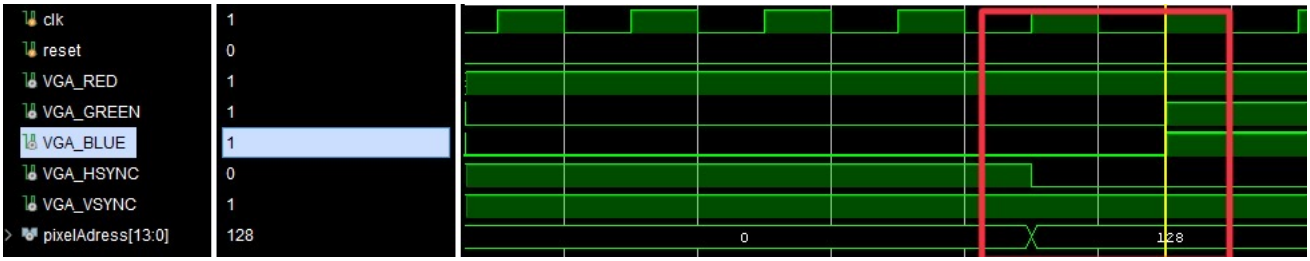


Figure 26: Testbench - Part C

Whenever the address of the active pixel changed, the red, green and blue signals didn't get the value of the active pixel immediately and kept their previous value for 10ns (1 clock cycle). Although i don't think that was the reason for the colours to overlap, this could be what made the lines have shadows.

5 Conclusions

The purpose of the assignment was the implementation of a VGA driver. The progress of the implementation was divided into 3 parts. First, we had to create a video RAM (VRAM) in which we would keep the data of the colours that would make up the picture we would later display on the monitor. After that we had to make an hsync controller that would be responsible for the horizontal synchronization of the monitor. Basically the controller would provide the hsync signal with the correct timing. At the same time we had to create a horizontal pixel counter that would be synchronized with the HSYNC controller and would be used for displaying the active pixel from the VRAM to the monitor. Then, we had to implement the VSYNC controller that was responsible for the synchronization of the refresh rate of the monitor, alongside with the vertical pixel counter that was also synchronized with the VSYNC controller. The vertical pixel counter was then used together with the horizontal pixel counter, to provide us the address of the active pixel that we would display on the screen. Finally by combining all the previous modules together, we implemented the VGA Driver. In terms of testing we used pretty much the same testbench for all the parts (except the first one) and all the debugging was done by observing the waveforms. Finally, after the simulation a problem occured on the experiment with the FPGA but it wasnt resolved. We ended up having a partially working VGA Driver that can turn the screen on but cannot display the colours right due to colours overlapping in each line.

6 References

References

[1] Xilinx. Nexys A7 Reference Manual

[2] Xilinx. Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide (UG953)

[3] Digilent. Master XDC Files

[4] Engdahl, Tomi. VGA Timing Information

[5] Christos Sotiriou, Christos. Lab 3 - Video Graphics Array Driver

[6] Wikipedia. Video Graphics Array