Video Graphics Array (VGA) Driver

CHRISTODOULOS ZERDALIS, AEM: 03531, 11/11/2024

Summary: The objective of this lab is to design and built a VGA (Video Graphics Array) driver in three key stages: The memory (VRAM), the Hsync signal that is responsible for the synchronization of the horizontal lines and the Vsync signal that controls the behavior of the whole frame. This report provides a detailed explanation of each stage, offering insights into the design process, underlying concepts, challenges encountered, and construction strategies for each component.

Introduction: This lab aims to build a comprehensive understanding of the VGA driver design by constructing each of its key components: memory, Hsync, Vsync and finally the combination of all of them. Each stage presents unique challenges and requires careful consideration of timing, data integrity, and synchronization. The first stage included building the memory and initializing its contents.

Part A – Video Ram (VRAM):

Implementation: The Video Ram is an integral part of the VGA driver and is the memory that is constantly displayed on the screen. Its size depends on the resolution of the image that is going to be displayed, the greater the resolution the greater the memory. In this case the resolution of the image is 640x480 pixels. To calculate the memory needed to store this image it has to be taken into account that every pixel needs three values red, green and blue so the total memory needed is 640x480x3 = 115.2 Kbytes.

Even though the FPGA board used for this project has sufficient memory to store this amount of information, this memory cannot be controlled all at the same time or in other words it is split into smaller blocks. The biggest block available is 1bit 16Kbytes and that allows for an image with a resolution 128x96 if the red, blue and green components are in different BRAM modules.

The target is a resolution of 640x480 pixels so the 128x96 pixels stored are going to be upscaled five times. To create the BRAM the BRAM_SINGLE_MACRO language template is used as it has only one port and it is simpler than the other approaches. After that the parameters are initialized to be only-read and the contents are initialized through the .INIT_xx parameter.

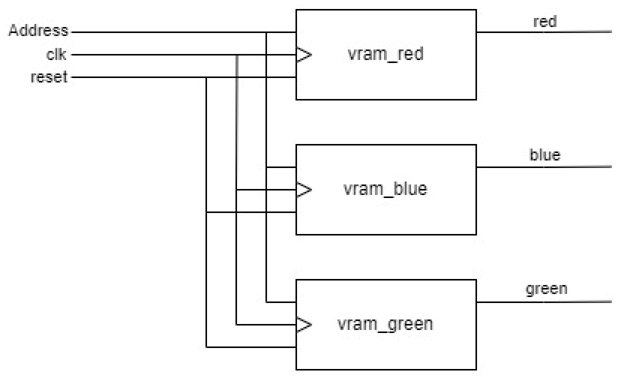
Modules:

Vram_red: stores the values for red

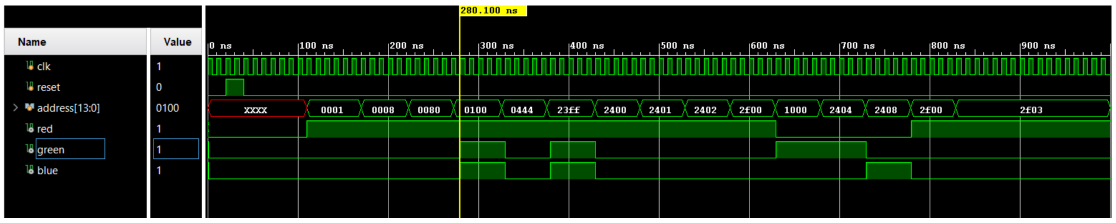
Vram_green: stores the values for green

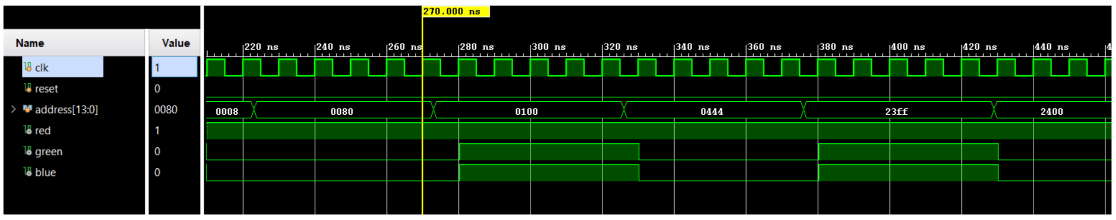
Vram_blue: stores the values for blue

Vram: combines all three modules



Simulation: In this simulation some addresses that store different colors are tested



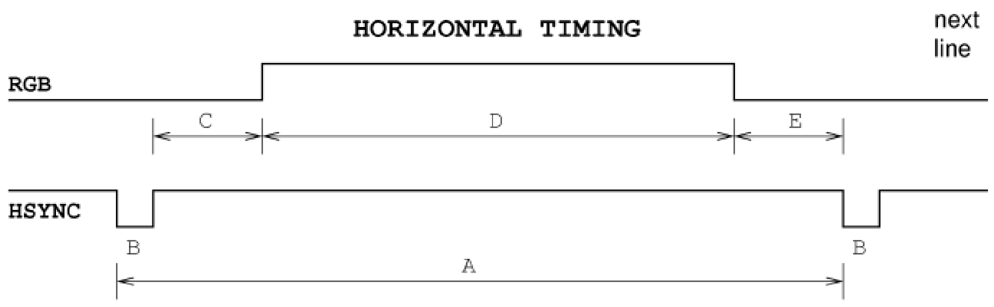


Here it is evident that every time the address changes the values of the RGB change and that the change is not asynchronous and happens at the next positive edge of the clock.

Experiment: There was no experiment on the FPGA on this part.

Part B – Hsync and Horizontal pixel counter:

Implementation: In this part of the design the Hsync signal is implemented, this signal is responsible for the depiction and synchronization of the horizontal lines on the screen and looks like this:



A: *Scanline Time* 32μsec
B: *HSYNC Pulse Width* 3.84 μsec
C: *Back porch* 1.92 μsec
D: *Display time* 25.6 μsec
E: *Front porch* 0.640 μsec

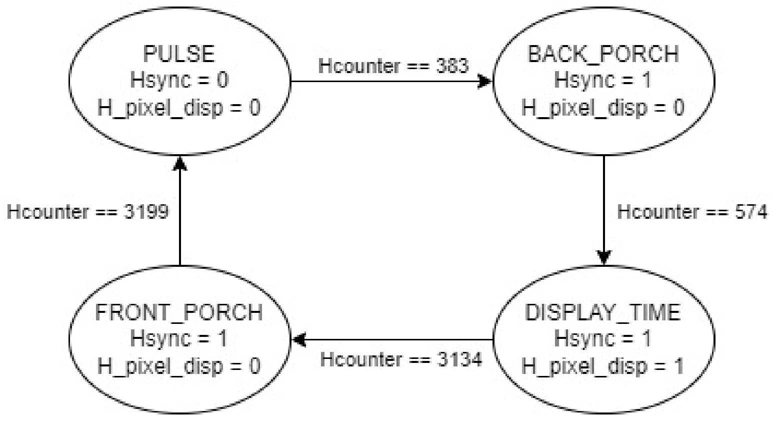
The time A is the period of the signal from pulse to pulse and is the time that the screen needs to display a line and get ready for next one. D or Display time is the time in the pulse at which the screen is ready to depict the pixel data given to it. Lastly, E and C are the times after and before the display time that are needed for the screen to get ready for showing new data.

To follow these times an 11-bit counter is designed that counts to 3200 and because the clock of the design is 10ns that takes 32μsec or a period of Hsync signal.

After that a finite state machine is created to simulate the four stages of the signal. These stages are: PUSLE, FRONT_PORCH, DISPLAY_TIME, BACK_PORCH and its outputs are: Hsync, H_pixel_disp. The FSM starts from the pulse state with both outputs at zero and when the counter is at 384 the state changes to BACK_PORCH where the Hsync is activated. At count 575 the state changes to Display time where the H_pixel_disp is also activated.

It is important to understand why the display time starts at 575 and not 576. This happens because the output H_pixel_disp is responsible for driving the address variable to the Vram and as mentioned before the Vram only gives its data at the positive edge of the clock. That results in a delay of one clock cycle (on top of the Vrams delay) in the implemented design as the activation of H_pixel_disp does not happen instantly. Although there is a chance that this small delay would not affect the output on the screen, to be sure the FSM changes its state one clock cycle earlier.

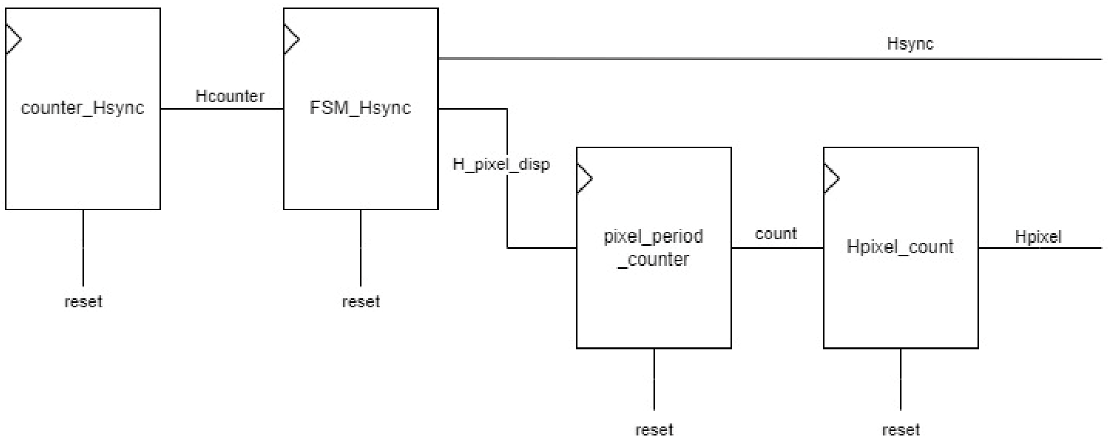
After that at count 3135 (again one earlier) the state changes to front porch the H_pixel_disp deactivates and at last at count 3200 the state changes to pulse there Hsync is also deactivated.



While the FSM controls and Hsync signal and the display time something must control the horizontal pixels that are displayed at each time or in other words when the address that controls the output of the Vram changes.

Since every line has 640 pixels and the display time is 25.6 μsec every pixel must be displayed for 40ns equal to 4 clock cycles. That means that every 40ns the address should increase by one but because the stored data has a resolution of 128x96 every line has 128 pixels and to upscale the image the address should change every 5 on screen pixels or every 20 clock cycles. This process is timed with a counter that counts for 20 clock cycles and when the count is 20 another counter increases by one until it reaches 128. Both counters start when the H_pixel_disp is activated and the output of the second one is the Hpixel signal.

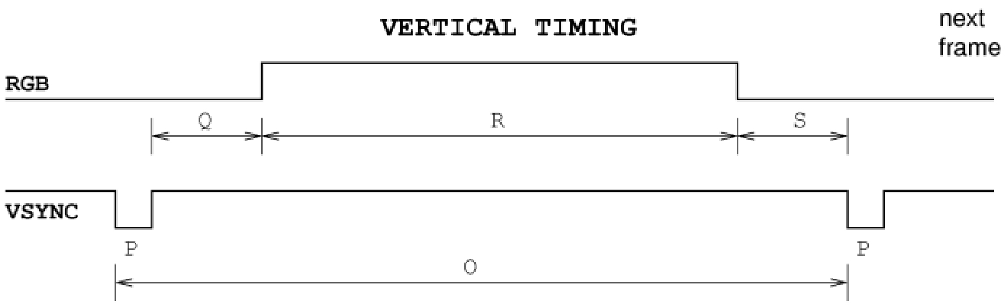
Modules:
Counter_Hsync: The counter the is used by the FSM to change its states
FSM_Hsync: The FSM that simulates the Hsync signal
pixel_period_counter: The counter that count the 20 clock cycles that every pixel should be display for
Hpixel_count: The counter that outputs the Hpixel



Simulation/Experiment: Part B and C and going to be tested combined.

Part C – Vsync and Vertical pixel counter:

Implementation: In this part of the design the Vsync signal is implemented, this signal is responsible for the depiction and synchronization of the hole frame on the screen and looks like this:

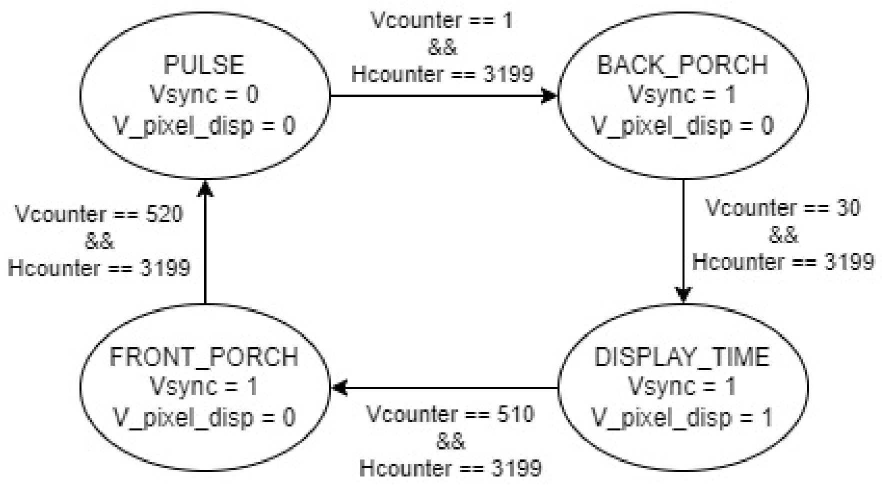


O: *Total Frame Time* 16.672 msec
P: *VSYNC Pulse Width* 64 µsec
Q: *Back Porch* 928 µsec
R: *Active Video Time* 15.36 msec
S: *Front porch* 320 µsec

These times follow the same principle as the times from the Hsync and It should be noted that each one of them is divisible by the Hsync Scanline time.

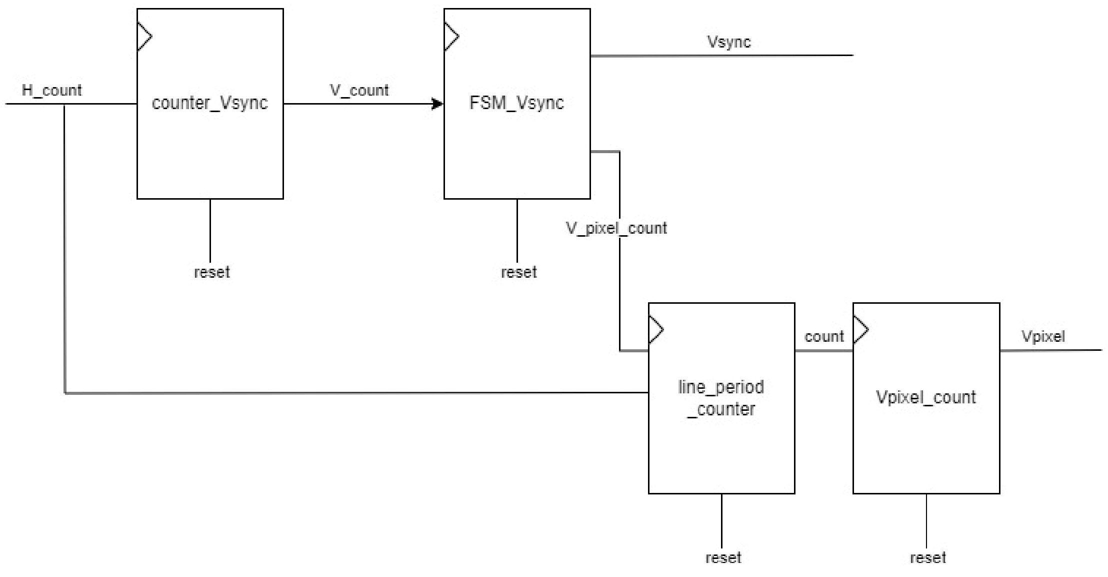
The total time of the frame is 16.672ms and if divided by 32µs equals to 521, that means that a Vsync period or a frame is equal to 521 Hsync periods. Knowing that a 9bit-counter that counts to 521 and that increases every time Hcounter is equal to 3199 is created to time the four stages of the signal and similarly to part B an FSM is designed with four stages and two outputs the VSYNC and V_pixel_disp.

O: 521x**A**	**R:** 480x**A**	(*Vsync times expressed by scanline period)*
P: 2x**A**	**S:** 10x**A**	
Q: 29x**A**		



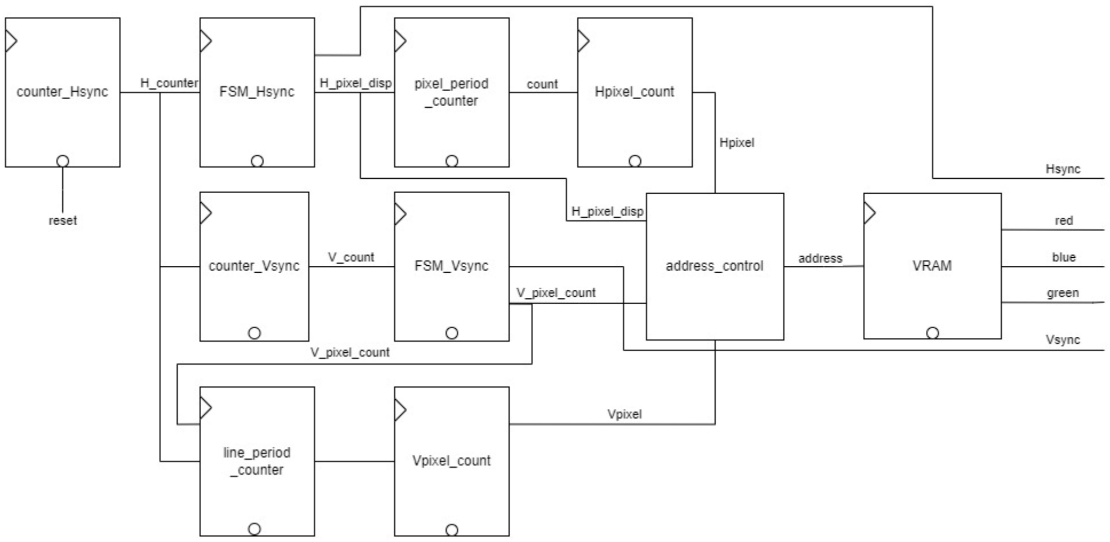
Like the Hpixel that controls the address of the horizontal pixels a Vpixel is need for the vertical ones. For that a counter that counts to 5 is created (5 because of the upscaling) and is increased every time a line is scanned (Hcounter == 3199) and every time the count is equal to 5 another counter increases the Vpixel value until it reaches 96. These counters start when the V_pixel_disp is enabled.

Modules:
Counter_Vsync: Counts the number of complete scanlines
FSM_Vsync: The FSM that controls the Vsync and V_pixel_disp
Line_period_counter: Counts 5 complete scanlines
Vpixel_counter: Increases the Vpixel every time the Line_period_count is equal to 5

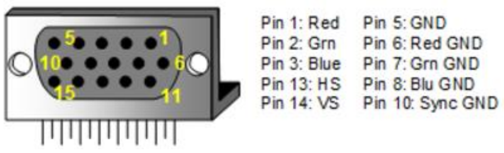


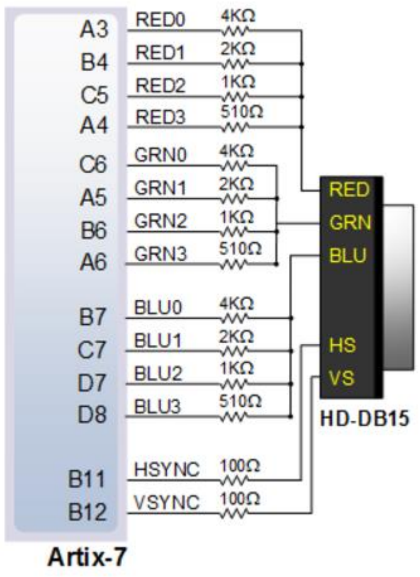
To control the address given to the Vram a module called address_control is designed. This module assigns to the addresses 7 MSB the Vpixel and to the 7 LSB Hpixel only if H_pixel_disp and V_pixel_disp are both activated (which means that the video is activated) else it assigns the address to an address that stores a black pixel because when the video is not on the values red, green, blue should be zero.

Top module diagram:



Constraints: The constraints of the design where initialized based on this image from the manual

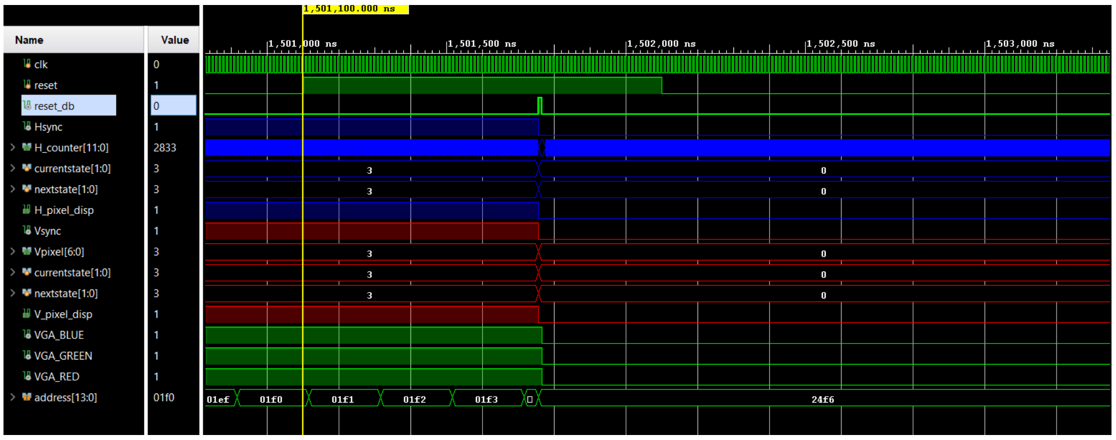




Simulations: In this simulation the vga_controller is reset to show that the reset debounce works and after that only the clock changes and the module is generating frames until the simulation is stopped.

In the following images, examples of times and why hey work will be shown.

Debounce:

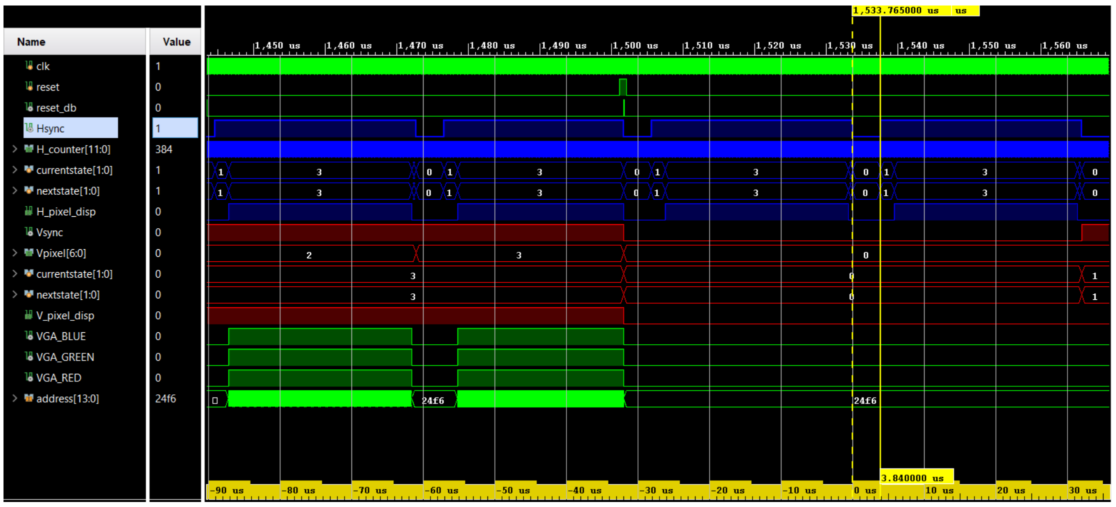


Name	Value
clk	0
reset	1
reset_db	0
Hsync	1
H_counter[11:0]	2833
currentstate[1:0]	3
nextstate[1:0]	3
H_pixel_disp	1
Vsync	1
Vpixel[6:0]	3
currentstate[1:0]	3
nextstate[1:0]	3
V_pixel_disp	1
VGA_BLUE	1
VGA_GREEN	1
VGA_RED	1
address[13:0]	01f0

The long reset signal is reduced to one clock cycle

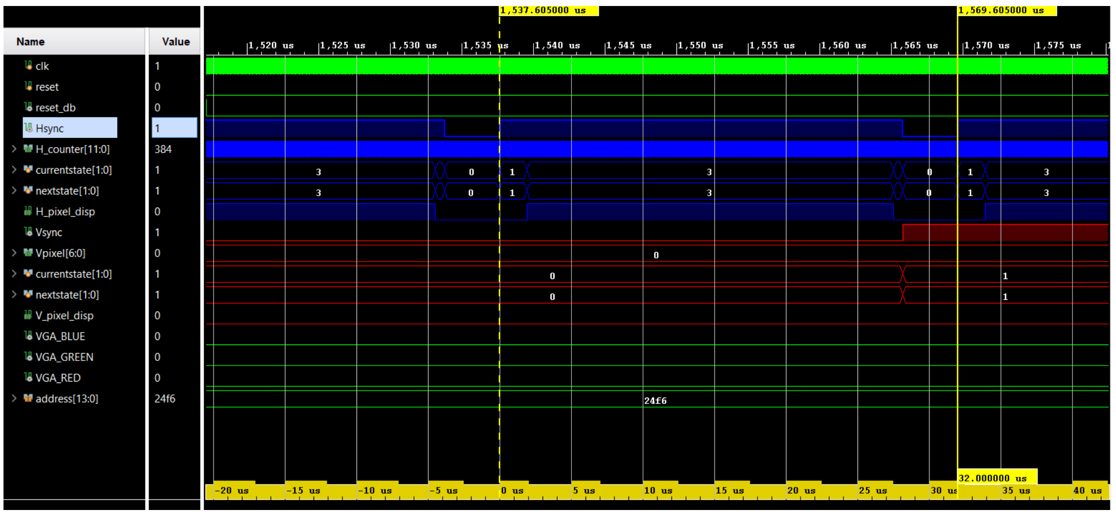
Hsync times:

HSYNC Pulse Width:



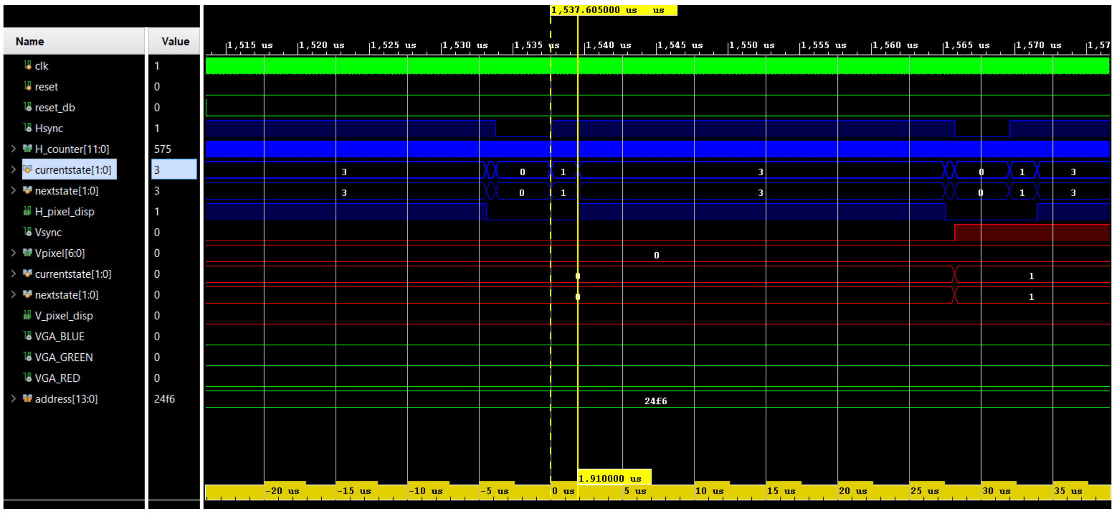
Name	Value
clk	1
reset	0
reset_db	0
Hsync	1
H_counter[11:0]	384
currentstate[1:0]	1
nextstate[1:0]	1
H_pixel_disp	0
Vsync	0
Vpixel[6:0]	0
currentstate[1:0]	0
nextstate[1:0]	0
V_pixel_disp	0
VGA_BLUE	0
VGA_GREEN	0
VGA_RED	0
address[13:0]	24f6

Scanline Time:



Name	Value
clk	1
reset	0
reset_db	0
Hsync	1
H_counter[11:0]	384
currentstate[1:0]	1
nextstate[1:0]	1
H_pixel_disp	0
Vsync	1
Vpixel[6:0]	0
currentstate[1:0]	1
nextstate[1:0]	1
V_pixel_disp	0
VGA_BLUE	0
VGA_GREEN	0
VGA_RED	0
address[13:0]	24f6

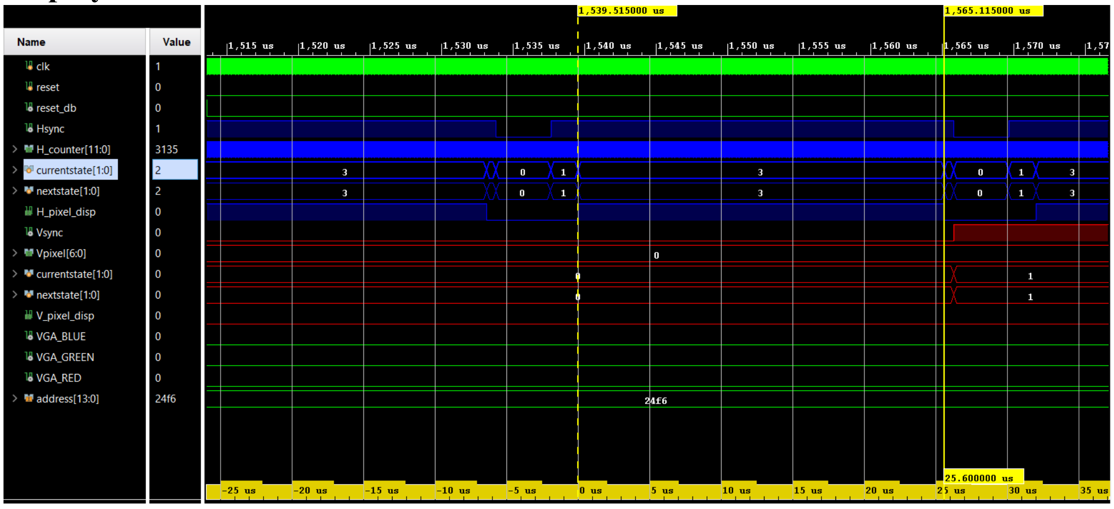
Back Porch:



Name	Value
clk	1
reset	0
reset_db	0
Hsync	1
H_counter[11:0]	575
currentstate[1:0]	3
nextstate[1:0]	3
H_pixel_disp	1
Vsync	0
Vpixel[6:0]	0
currentstate[1:0]	0
nextstate[1:0]	0
V_pixel_disp	0
VGA_BLUE	0
VGA_GREEN	0
VGA_RED	0
address[13:0]	24f6

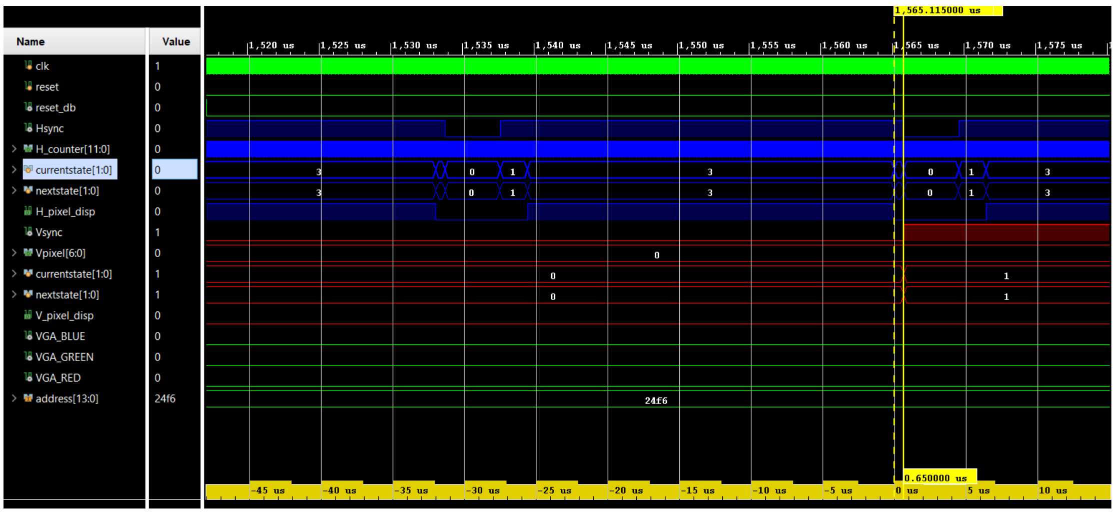
It should be 1.92 but it was explained why it is one clock cycle shorter*.

Display Time:



Name	Value
clk	1
reset	0
reset_db	0
Hsync	1
H_counter[11:0]	3135
currentstate[1:0]	2
nextstate[1:0]	2
H_pixel_disp	0
Vsync	0
Vpixel[6:0]	0
currentstate[1:0]	0
nextstate[1:0]	0
V_pixel_disp	0
VGA_BLUE	0
VGA_GREEN	0
VGA_RED	0
address[13:0]	24f6

Front Porch:

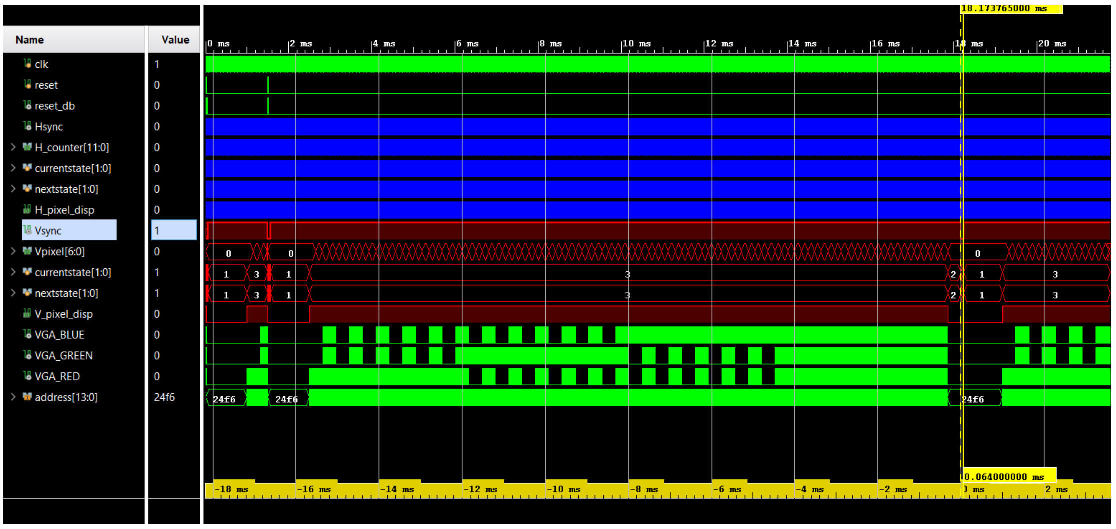


Name	Value
clk	1
reset	0
reset_db	0
Hsync	0
H_counter[11:0]	0
currentstate[1:0]	0
nextstate[1:0]	0
H_pixel_disp	0
Vsync	1
Vpixel[6:0]	0
currentstate[1:0]	1
nextstate[1:0]	1
V_pixel_disp	0
VGA_BLUE	0
VGA_GREEN	0
VGA_RED	0
address[13:0]	24f6

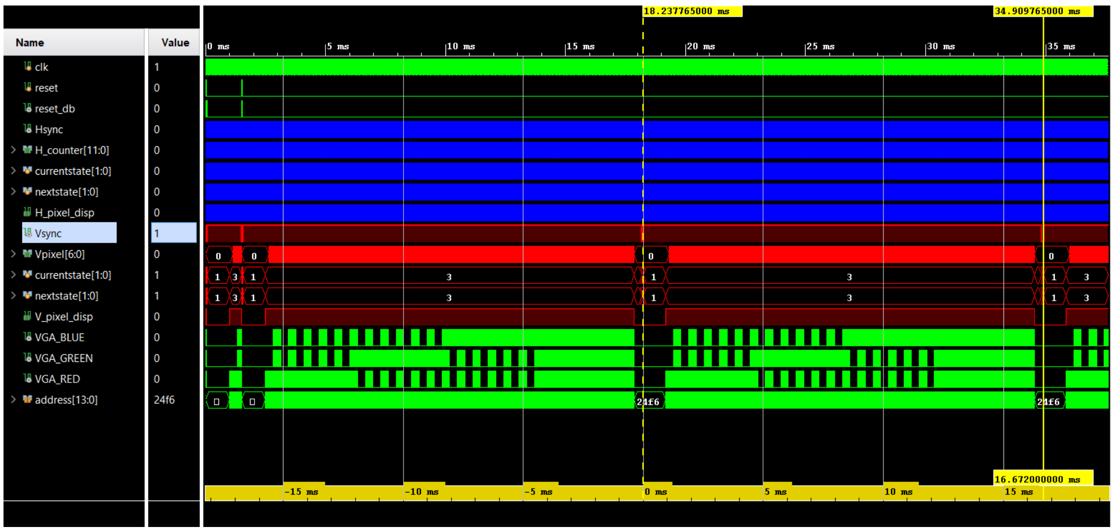
One clock cycle longer as a result of the change above*.

Vsync times:

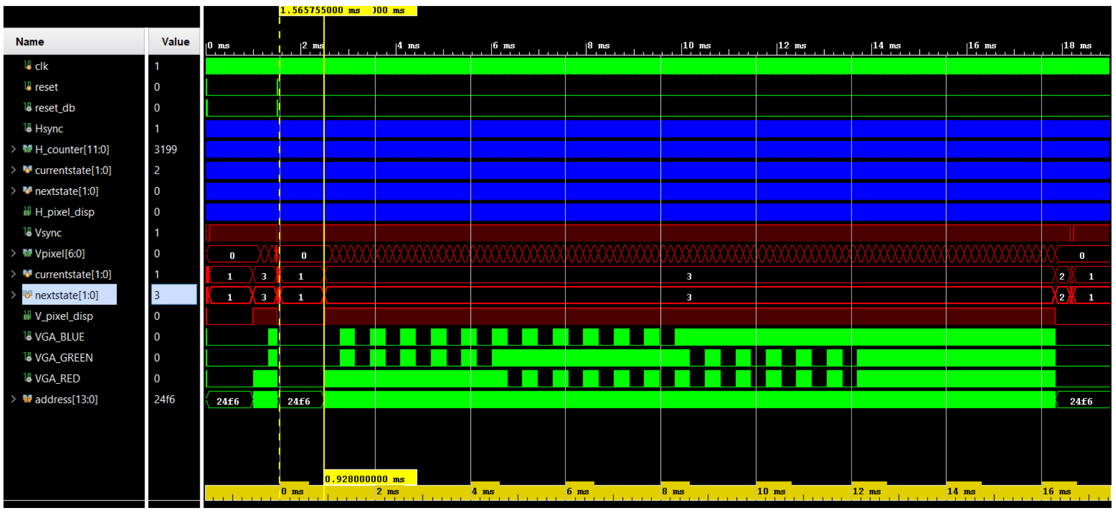
VSYNC Pulse Width:



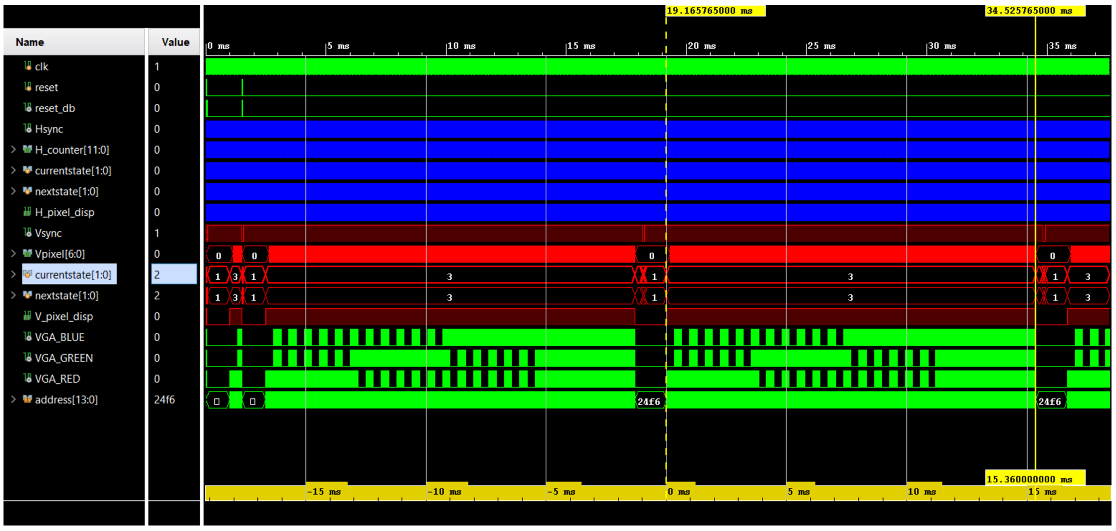
Total Frame Time:



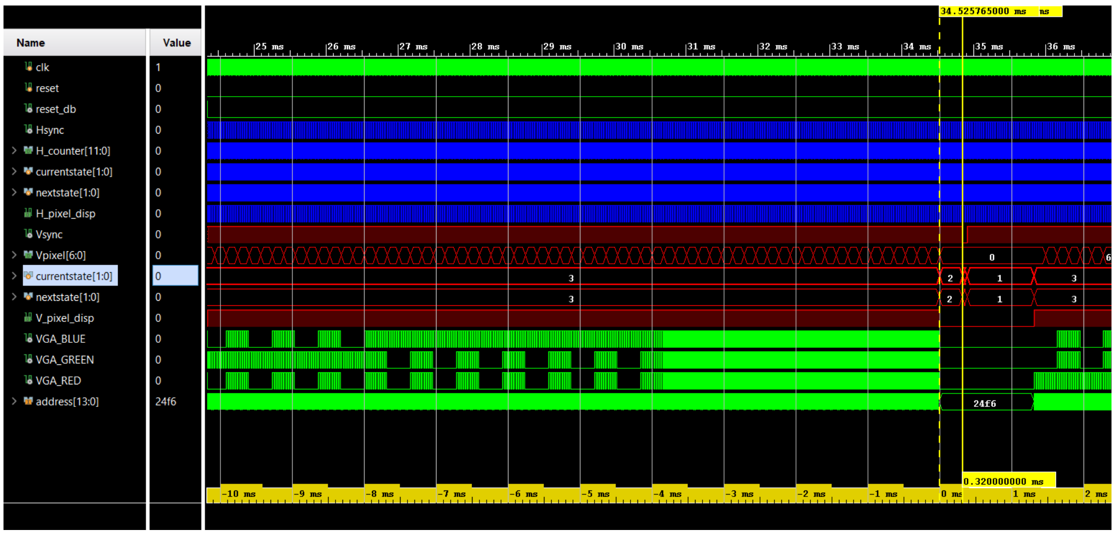
Back Porch:



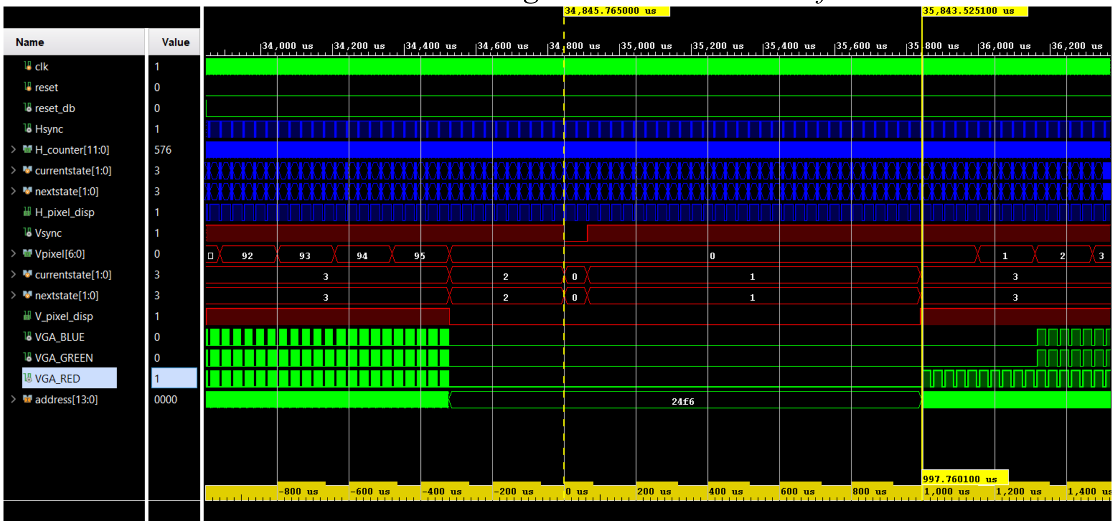
Active Video Time:



Front Porch:



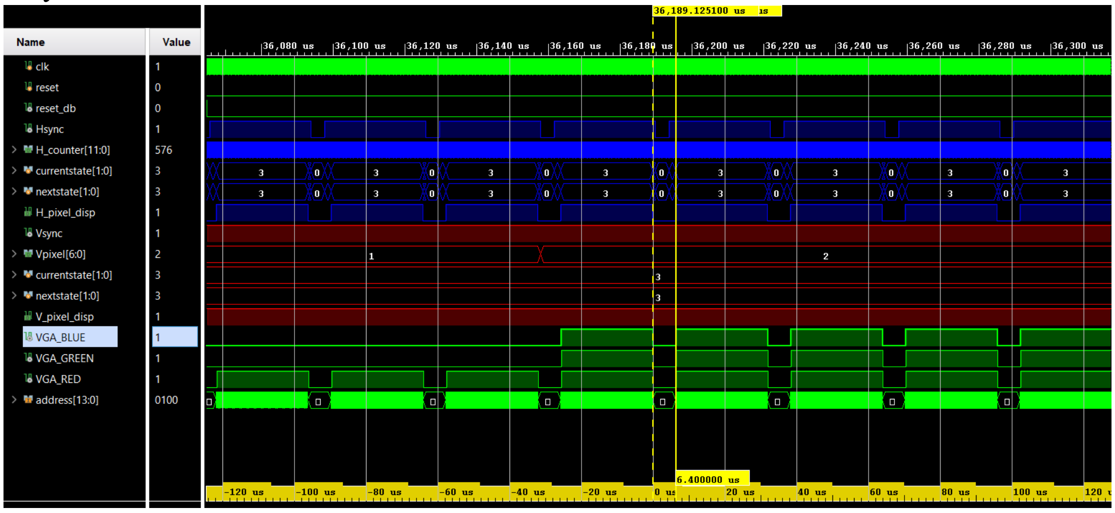
Now lets see the time when the screen begin to show the video after activation:



It is equal to 997.7601µs because the Vpulse, Vbackporch, Hpulse, Hbackporch and the latency of the video RAM all have to happen before the video starts.

64(P) + 928(Q) + 3.84(B) + 1.92(C) + 0.0001(latency) = 997.7601μs

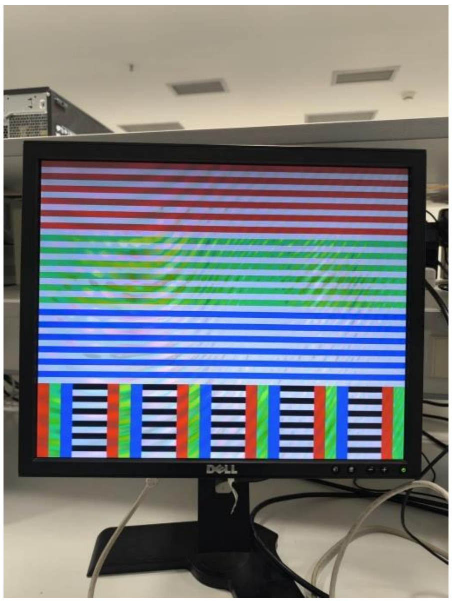
Lastly let's see the time period that the pixels ,between every line, are deactivated and why:



Name	Value
clk	1
reset	0
reset_db	0
Hsync	1
H_counter[11:0]	576
currentstate[1:0]	3
nextstate[1:0]	3
H_pixel_disp	1
Vsync	1
Vpixel[6:0]	2
currentstate[1:0]	3
nextstate[1:0]	3
V_pixel_disp	1
VGA_BLUE	1
VGA_GREEN	1
VGA_RED	1
address[13:0]	0100

3.84(B) + 1.92(C) + 0.64(E) = 6.4
In order to get from one H_pixel_disp activation to another the front porch, back porch and HSYNC Pulse Width are between.

Experiment: The design was loaded on the FPGA and the results were correct from the start. There was no stuttering of misalignment of the image



Optional: For the optional part three more Vrams with different images were created and a counter that every 16 frames changed the output of the memory from one Vram(image) to another. This implementation is not perfect, but the idea is correct, sadly I did not have enough time to fully debug this design as the FPGA was needed and the implemented design needed 25 minutes to generate one second of video(60 frames). Also, the Verilog code is not up to the standards of this class but since it worked, I did not make any changes because I would not be able to test it afterwords. Lastly the images in the video have some weird artifacts that could be caused by the wrong initialization of the memories.

Conclusion: This project presented several significant challenges, including timing and synchronization between the two signals, Vsync and Hsync and it proved that even the smallest timing error could result to a completely wrong result.