VGA Controller in Verilog

By Angelica Reyes

Abstract:

This project proposed the design of a FPGA based VGA controller. The tools needed will be the Xilinx software, a keyboard, a VGA monitor, and Nexys4DDR. The reason I chose this as a project is because I wanted to challenge myself on what would be good experience for a job in the FPGA industry. Although I am new to VHDL/Verilog this will be a fun and challenging way to learn. I will be implementing a VGA controller in FPGA for this project. This VGA controller will be the design of a programmable logic component that successfully outputs the signal timing necessary to interface with a VGA capable monitor. The user will need to provide a pixel clock as well as image source for the input. For this project we will be learning how to use a VGA capable monitor using the Nexys4DDR board. The design will implement a circuit that generates a VGA signal to display a red, blue, or green screen on your monitor in a resolution of 1800p. VGA stands for video graphic array and defines the display of hardware in a computer. The very first VGA came in the 1987 IBM computer. This spiraled off several booming industry areas such as the competitive Graphics cards market. This report will start from the ground up and walk you through how VGA works with the FPGA board. For the purpose of this project I will be designing the VGA controller using a Nexys4DDR board. This version of the board comes with a VGA port where I will then connect to a VGA monitor for the output of the design. For input, I will be using a VGA Cable that will be plugged into the VGA port of the Nexys4DDR board. This design can be better understanding on the block diagram of Figure 1 below. For demonstration purpose I will show how to display a color and letter on the VGA monitor using the controller that will be designed from this project. First, we will need a clock divider to generate the pixel clock for the timing reference to the display signals. Second, we will need two counters for horizontal and vertical lines in the frame of the monitor/display. Finally, we will test the controller on a 1800p LG 14inch LED monitor. In the end, the design will create a simple VGA controller using Verilog HDL and FPGA Nexys4DDR board using the Xilinx software.

Keywords: FPGA, HDL, Xilinx.

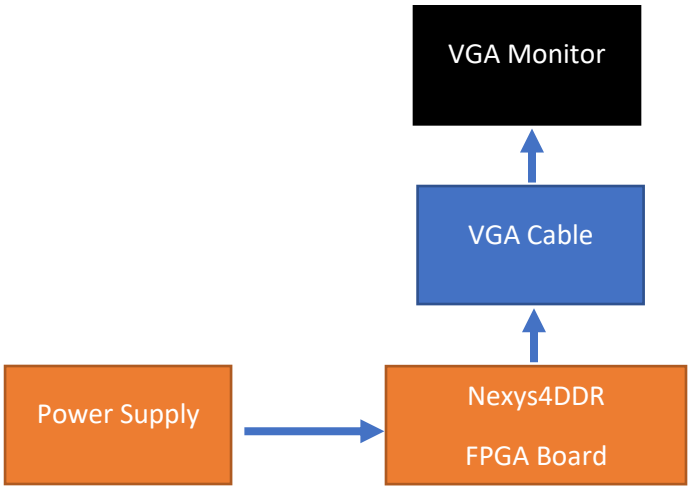


Figure 1: Flow Diagram of VGA Controller Design

1. VGA Controller: Design and Purpose

VGA Controller: What Is It why do we need it?

VGA stands for video graphics array and is a hardware interface standard introduced by IBM in the 1980s for their IBM PC. In FPGA world it is known as a programmable logic component that accomplishes the signal timing needed for a successful interface with a VGA capable monitor. VGA implementation requires a guided pixel clock and the image source. The VGA controller maintains a dual dimensional array of 15bit values in video memory. The width of the array, horizontal sync is 640 (0 to 639) pixels. The height of the array, vertical sync, is 480 (0 to 479) pixels. Referencing dimensions, each of the values represent the color of a pixels where bits 10 to 14 specify the amount of red, 5 to 9 bits specify the amount of green, and bits 0 to 4 specify the amount of blue. There are 256 colors that can be displayed on the video screen with VGA. This combination of red, blue, and green is known as RGB and is used to coordinate in several ways not just in regard to VGA. The pixel coordinates of a VGA display are seen as a graph where x and y are used to address the location of a pixel. In this project the (640 by 480) display has an upper left pixel at 0,0 and a lower right pixel at the coordinate (639, 479). The Nexys 4 DDR board uses this design of 15-bit connector.

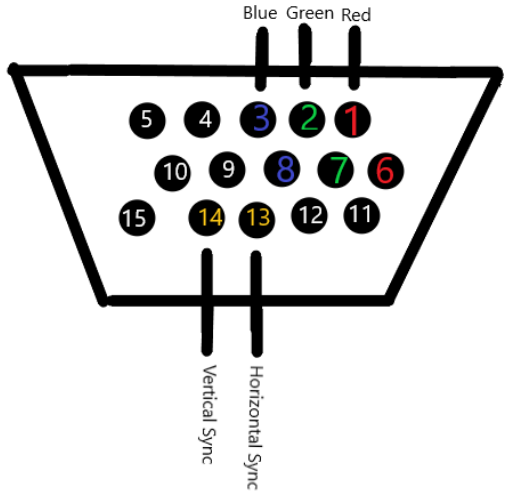


Figure 2: Pin diagram of NEXYS 4 DDR VGA.

The most important pins are pins 1,2, and 3 which are analog outputs for the red, green and blue as seen on Figure 2. Following the same figure, pins 14 and 13 are the vertical sync and horizontal sync respectively. The 15-pin connecter is connected to the video monitor using a male to male VGA cable connector. For the purpose of this project design we will be using the Nexys 4 DDR Diligent CRT display timing that is a 640 by 480 display. The coordinates scan down starting from the top and going to the bottom of the screen monitor display. This is all done at a uniform speed using a clock divider. In this project you will notice the clock divider which pulses through the pixels of the monitor to allow for a VGA display from the FPGA board to the LG Monitor. The design of this project is compatible with any FPGA that is 640 by 480 resolution and is in Vivado. Since VGA is a standard interface for controlling analog monitors its capabilities are wide. For the purpose of this project we will see how to build a VGA Controller and we will preform a simple task to display on a LED Monitor. This project will show you an implementation for the necessary horizontal and vertical sync signals expected to be received by the VGA receiver in order to successfully display a colored screen on the monitor. The rows are the vertical coordinates of the pixels on the monitor. The column are the horizontal coordinates of the pixels being drawn on the monitor. The combination of the horizontal and vertical sync along with the clock and counters determine the RGB color outputs given a certain high and low instruction at the top level.

2. Clock Divider

The VGA controller needs a reliable clock divider signal to provide stable time for operation to count the pixels in each line. This requirement needs to be successful in order for the Vertical Sync and Horizontal Sync timing references to work. We will go more in depth on Vertical Sync and Horizontal Sync in this project report. The clock divider in this project design has an input and it then divides the clock input. All you need to do is set the clock output equal to 0 at the time of reset, clk <= 1'b0. Most clock dividers in FPGA happen through Phase Lock Loop PPL or DCM, digital clock manager. Once the clock is available it is providing opportunity for simple synchronous clock division through gates and D Flip Flops. This is a core implementation principle in FPGA as it can be used for several projects in both VHDL and Verilog. When you create a clock divider you are actually creating a new clock for the purpose of the project. Which in most cases the clock will be divided by an even number so an inverter can be placed at the dividers output. You can use inverters to avoid having to use combinational logic for timing which can be lengthy and complicated for big projects.

Switching the focus to the design of the clock divider for this specific project where we are designing a VGA controller, we have one input wire for the clock and one output register for the clock divider. We set the output divided_clk = 0 and the module clock divider value to div_value = 1 in order to reduce the clock signal to 25 MHz We need it to be 25MHz because the pixels change at that frequency for the 640 by 480 VGA resolution that we are implementing for this project. This results in half of the frequency of a 50MHz clock that is provided by the Nexys 4 DDR FPGA board. Essentially creating the clock to ignore every other clock cycle. The pixel rate can be found using the formula below. See the results for this project on the 2nd line for = 25MHz for a 640x480 pixel screen.
Pixel Rate = (Total Horizontal Pixels * Total Vertical Lines * Number of screens / second)
= 800 * 525 * 60 = 25 MHz

3. Horizontal and Vertical Counters

In VGA Controller you also need a horizontal and vertical counter. These counters work with the clock divider discussed earlier to provide accurate timing for the controller. The horizontal counter counts pixels in each line and the vertical counter counts lines in a frame. The counting starts in the upper left-hand corner of the 640 by 480 display at the coordinate of (0,0) and ends at the bottom right hand corner at the coordinate of (639, 479). The code needs to be designed so that the horizontal counter resets when it reaches the line 799. Once the reset is high the input of the vertical counter enable is triggered so that the vertical count can start and add 1 per new line. The vertical counter needs to reset once it reaches the end of the frame after 524.

(a)

```
if(enable_V == 1) begin
    if (V_Count < 524)
    V_Count <= V_Count +1;
```

(b)

```
if (H_Count < 799) begin
H_Count <= H_Count +1;
enable_V <=0;
```

Figure 3. (a) The if loop verilog code for the Vertical counter with the enable on. (b) The if loop verilog code for the Horizontal counter where enable is off until H is reset.

Once the Horizontal and Vertical timing signals are complete they will coordinate the delivery of the image or video processing based on pixels that will be transferred via VGA and displayed on a VGA capable monitor. As the counter moves through each pixel the data is stored into memory where each bite assigned to each pixel location is used for the display. The VGA controller must use indexing

memory as each pixel is processed across the display Area. You can see in detail exactly what the Display Area is and how it relates to vertical and horoizontal dimensions that are so often used to cordinate the vertical and horizontal sync and counters of this project. Each pixel coordinate will go through the timing code to define the display of each pixel on the monitor. To better understand how the display of the image is controlled by the three counters we have gone over thus far, clock divider, horizontal divider, and vertical divider, we will now take a look at the timing specification for a 640 by 480 resolution display.

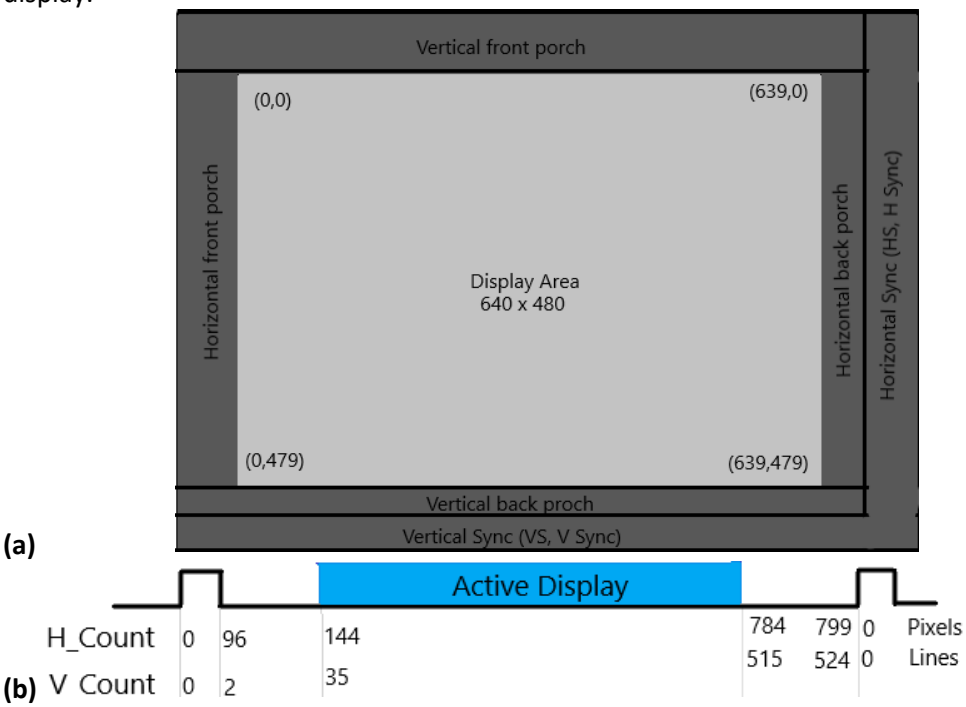


Figure 4. (a) This diagram shows the horizontal and vertical sync as well as available. You can see the timing specifications per addressable video of the VGA controller.Display Area. (b) Shows the Horizontal and Vertical counter values in the Active display area.

Horizontal counter is designed to count through all the sequence of pixel columns which includes a count from 0 to 799 (640 display pixels, 16 front porch pixels, 48 back porch pixels, and 96 pulses greater than 639.The Vertical counter counts the rows. It is used to sequence through the vertical sync phases and indicate the current line being displayed. This counter will count all rows/ lines until the value of 479 giving indication of the last visible displayed line.

4. Top Level and Testbench: Testing and Results

Indication The next part of this project is to create a top level circuit and a testbench. This top level code witll use the VGA timing controller to generate the display on the monitor. It will use the signal from that VGA timing controller to generate the proper RBG signal to the display. The first step once the top level is created is to instance the VGA controller so that the design can run correctly. Initiate a reset that will be immplemented in the clock cycle to initialize the VGA timing module that was created from the horizontal and vertical counters. Add code that will decode the pixels/column and lines/rows for the appropriate Red Green and Blue values to be displayed on the display monitor. If the values are empty or zero then the RGB values and the counters will fail and there will not be a successful display. Essentially the top module will tie all the code together. The top module interface has one input for

clock, two outputs for Hsync and Vsync, and three additional outputs for red, green, and blue. The next declatations are the wires between the Controller and the data bath. The three wires going back and forth between the controller and that datapath are the 25M clock, the enable for vertical counter, and the Horizontal and Vertical counter. It is important to secify that the Horizontal and Vertical counters are going to be 16bit values. The rest are single bit inputs. The controller will take the wires as inputs and the datapath will take inputs from the top module as inputs. The clock, reset, and enable will ideal to managing the data that is being processed between the Controller verilog code and the datapath verilog code. The clock and the reset will go into the controller to initiate the design. As mentioned, the top module is combining the clock divider, the horizontal counter and the vertical counter. Finally, what is now left of the top module is to write the code for the outputs using assign. In this project design we are assigning the Horizontal and Vertical sync. Assigning the hosizontal sync signals will be coded to go high only when the Horizontal counter value is less than 96 (H_Counter < 96). Once the Horizontal counter reaches 96 it will reset to 0. Why 96? Remember from our earlier disussion we can look back at the 4b and see that the Horizontal sync time width/frequency is 96 pixels. Similarly we will assign the Vertical counter to go to high only when the Vertical counter value is less than 2.Once it reaches two it will reset to zero and again 2 is the vga standards for the 640*480 vga display on the nexys 4 ddr board. Final three assign code left is for red, green and blue (RGB). For the purpose of demonstrating this project we will be assigning the colors to display a white screen. We do this by assigning all the colors to go high (1,1,1). You can look up colors for vga simply by going to the website of your board and making sure to search for VGA colors to be sure you are within the addressable range when you are assigning your color to fit the purpose of your project. You will view a diagram of all the colors available to be dispalyed on the screen. Moving on with assigning the colors we will start with red. In this project we are assigning the red to allways be displayed when the Horizontal counter value is between 143 and 784 and when the Vertical counter is greater than 34 and less than 515. And you can take another look at Figure 4b for a refresher on why we are choosing those two values. It is because those are the pixels that are addressable in the available Display area on our monitor. Take a look at Figure 4a for a better demonstration of what exactly the Display Area is on the diagram. Next we will assign the green. We are assigning the green to allways be displayed when the Horizontal counter value is between 143 and 784 and when the Vertical counter is greater than 34 and less than 515. Third we assign to allways be high when the Horizontal counter value is between 143 and 784 and when the Vertical counter is greater than 34 and less than 515. Ending the module is important to avoid any syntax error before saving your code. Make sure to end module at the end of the top level verilog code.

The test bench instantiates the unit under test which then stimulus is applied from the top level code test bench to the lower level code of the design being tested. Testbench is important part of the design because it is the section of the design that will verify the functional correctness of the hardware design. The test bench code will check all possible outcomes of your desing and wether the hardware model does what it was designed to do. And likewise, will show you if your hardware code is not doing what it was designed to do. In short terms, you are generating code to compare the generated outputs against the expected outputs. This is similar to debugging process in normal C, Python, and other programming languages. The test bench for this VGA Controller will only requre the default library at the very start of the code, `timescale 1ns / 1ps. Timescale in verilog is used for testbench in top level to instruct the delays during processing. In this design we will use #5 which will tell the process to wait for 50ns before continuting. The time scale is as you can see 1ns delay unless otherwise specified usign the correct hashtag format to multiply that amount of delay to a longer delay. Next on the test bench code is to type in all the necessary signals and you can do this by looking at the top code as those need to be included. To simulate the design you will need the unit under test UUT and the stimulus by the test bench. In the unit under test we need to include all the inputs and outputs for the VGA Controller. You

can cross check against the top level code to makesure all inputs and outputs are accounted for. The last step to do in the test bench is to create the actual test bench sim of what we want to verify the design against. For this desing we will be simply running a clock that will be timed at #5 or 50ns. This will be ran and tested against all the hardware module we have made on the clock divider, horizontal counter, vertical counter, and the top module. Similarly like the top module make sure to always end the module of the test bench using the correct verilog ending for the test bench called endmodule with out any spaces.

Run the simulation to view the wave diagram and make sure the project is workign as it is desined to. Sometimes the format of the inputs and outputs are defulated to binary so you want to make sure and change the counter inputs for Horizonal and Vertical to decimal. Rnning the simulation you can see that the Horizontal counter counts until 799 and the vertical counter counts until 524. This matches our expected outputs as you can see again from Figure 4 of the report. If it passes this coundry the coordinates will stride off the available display are and into the unavailable area which will create an issue when we are tryining to get a full display when uploading the code into the FPGA board. You can also see that once the horizontal counter reaches the max it will reset the enable triggering the vertical coounter. Similarly you will test the Hsync and Vsync using the expected values. The vertical sync is high on the wave diagram when the vertical reached 524. The horizontal sync Hsync stays high until the horizontal counter reaches 96 and then resets itself to 0 as expected. Looking at the RGB wave diagram we want to test and make sure that the pixel values are being followed and are within the Addressable Avaialble Video Time which is between 144 and 784 for Hosizontal and 35 to 515 for Vertical. The horizontal availablility is 640 pixels and the Vertical availability of addressable video is at 480 pixels.

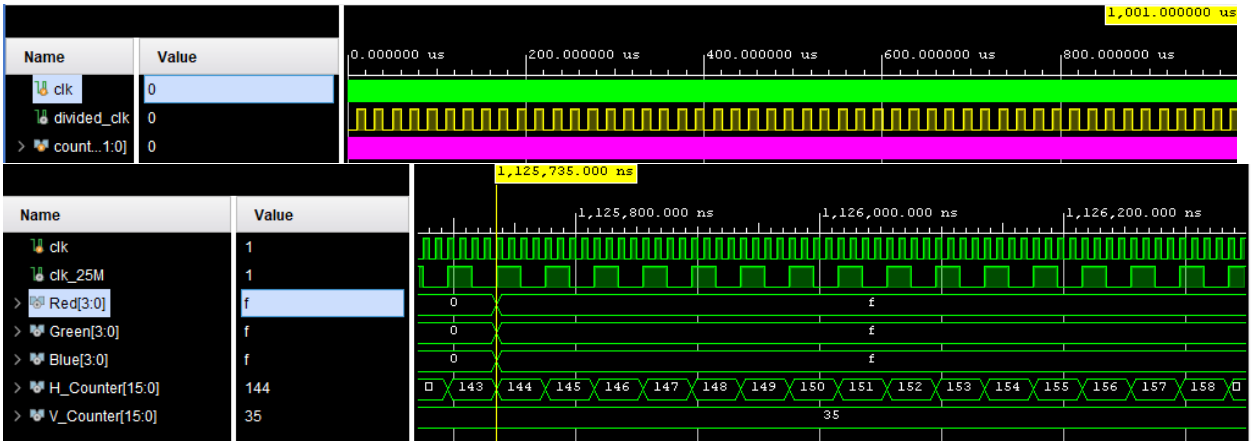
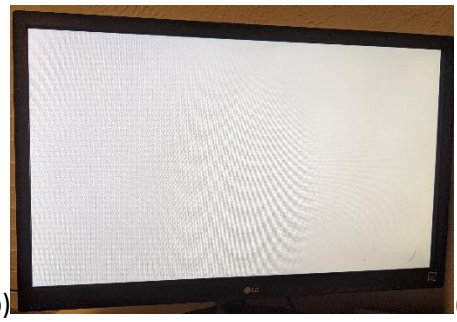


Figure 5 Simulation of clock divider and VGA controller in Vivado. You can see from the wave that RGB is high when H counter reaches 144 and V counter is at 35.

Taking a look into the simulation of the clock divider you should verify that the clock divider is pulsing on and off every nano second. This clock should be runing at 25MHz. Now to verify that RGB is working. Take your courser to the very front of the wave. Adjust the the vertical counter to 35 and make sure the horizontal counter is within the range at 144 you will see that here Red, Gree, and Blue will be set to high. Now take the courser to the end and verify similiarty the Vertical counter ends at 515 and horizontal counter at 784 and the RGA is then set to low. Since all the colors are high this will result in a white screen. Similarly you can make an adjustment to change the color. For example, the color Black is when all the colors are low. Simply adjust the top mudle to set the colors to low inistead of high and you will get a black screen. Another example is to turn the Blue to low and keep Red and Green at high and you will get a Yellow screen. Take a look at Figure 6 below for more guidance on the bits per color.

(a)

RED	GREEN	BLUE	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

(b) 

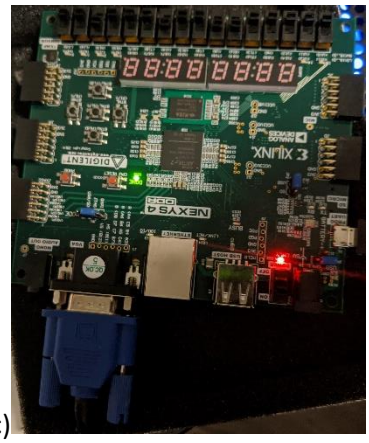
(c) 

Figure 6: (a)This is the VGA Color truth table that is standard for VGA and can be used for several image processing projects.; (b) This project outputs a white screen.; (c) This is a photo of the Nexys 4 DDR Board when project is running on Hardware.

5. Constraints

When using any switches, LEDS, ports or other components in an FPGA board you need to use the xdc file which maps all the components on the board. You will need to adjest the lines necessary for your project to be able to be processed successfully. In this VGA Controller project we use Clock signals. As you can see in Figure 7 we use the set_property to our clock and the create_clock to our new clock divider. In addition we will need to use the VGA connector for our outputs Hsync, Vsyng and RGB.

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk}];

##VGA Connector

set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { Red[0] }];
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { Red[1] }];
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { Red[2] }];
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { Red[3] }];

set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { Green[0] }];
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { Green[1] }];
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { Green[2] }];
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { Green[3] }];

set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { Blue[0] }];
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { Blue[1] }];
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { Blue[2] }];
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { Blue[3] }];

set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { Hsync }];
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { Vsync }];
```

Fiigure7: Constraints file for the VGA Controller.

6. Hardware and Software

The hardware necessary for this VGA Controller is the Nexys 4DDR board, a VGA cable, and a LED LG Monitor. The nexys 4 DDR board from Digilent is used in this project because of its capabilities to be easily reprogrammable. Some specifics on the is that it already comes equipied with a VGA female port where several previous generatoins of Digilent FPGA Boards. Digilent provides a user manual for all its components. I used the VGA port manual to get the correct VGA pin data as well as the data for the coordinates of the Available Display area for the vga controller. The Digilent Nexys 4 DDR board referance manual is also helpful for this project for the purpose of handling the correct ranges for the horizontal and vertical clock as well as for the H Sync and V Sync.

The software used in this project is Vivado 2019. Although this program can also be done with VHDL, this project was completed with Verilog. Verilog is less verbousus than VHDL which is why I decided to go with this language to the design of my project. Verilog is more widley used in FPGA when you will be needing to work with graphics or video since it is more harware modeling than VHDL. You can easily add modules for each display necessary. Here in this project we used a top level module to display a full white screen.

7. VGA Controller Capabilities

VGA Controller have many capabilities not just displaying a solid color display on the monitor. In this project we demonstrate a VGA Controller which is a very widely used controller for several projects in VHDL and Verilog. For the purpose of demonstration, we simplify this project to show how VGA Controller can be programmed with a NEXYS4DDR board in order to be able to use its VGA available port to display video or image via VGA cord to a monitor. One thing that can be done with this VGA Controller project to take it to the next level is display a graphic or image on the monitor. Since the VGA controller is based on a pixel map and color planes of horizontal sync and vertical sync you will need to find the appropriate pixel required for the image or graphic you want to display. The three color signals are red green and blue and these signals are set on and off to eventually successfully display the image being processed. The intensity of each color is another adjustment made since images can be made of several colors not only red green and blue. Using these things, the VGA Controller is capable of displaying up to 256 colors. The resolution of the screen can also vary per project and the Display area will need to be adjusted per screen pixels. The VGA image is controlled by the horizontal and vertical sync. The horizontal sync marks the beginning and end of the line of pixels for active display area. The vertical sync is very similar to the horizontal sync except that per clock pulse the vertical sync starts at a low and will go high per pulse within the addressable video time.

Another project you can make with this VGA Controller is a simple FGPA Game. You can create a Tetris game, a pong game, tic tac toe and several other games on your FPGA board. VGA controller is used to display the images and graphics required for these video games. As of this year FPGAs have not been used in gaming consoles yet but there is a possibility of markets being interested in the future. Usually the VGA Controller is used to generate the images of the game. You will need add a separate module for your gaming and might need more modules depending on the difficulty of the game. Let's see for example a pong game on the FPGA board. A single player pong game consists of two bounce pads, a ball, and a scoreboard. A module will need to be created for each image and added to the top module of the FPGA so that the VGA controller knows exactly what is being displayed at which coordinates on the display area of the monitor.

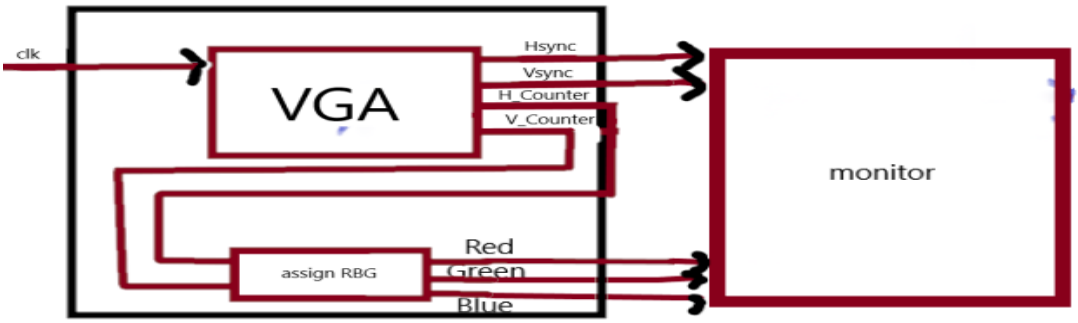


Figure8: Block Diagram of VGA Controller to monitor project.

8. Conclusion

In a VGA Controller there are four key sections: clock divider, vertical counter, horizontal counter, and top module. The Clock divider works to create the appropriate timing for the VGA controller. It is designed to display a color to the monitor in this project. VGA controller is necessary for several projects that require processing to a separate monitor or screen. VGA controller can also be successful via a VGA HDMI adapter, but pins will need to be adjusted as such. A VGAs hardware consists of 15 connectors that each serve a specific function in successfully transferring and processing display from the FPGA Nexys 4 DDR board to the LG Monitor used for this project. Pin one, two, and three are amongst the most imports pins in use since they control RGB, the red green, and blue colors that will successfully display image and video through the VGA Controller. These pins can be programmed to display the colors necessary for the project. Here we turn all the pixels to one color to display a solid colored Monitor display.

The three counters on the VGA Controller are very important for this project. They control the pixel count for the VGA display. Each counter works together to successfully process every available pixel in the Display area to display the color of the screen we want which in this case is white. The counter starts at the coordinate of (0,0) and go through to (639, 479). Staying withing the pixel and timer range is imperative to avoid going into back or front porch of the Horizontal and Veridical areas of the display. The clock divider created for this display is 25MHz which is half of the usual display of 50MHz. the divider begins on the positive edge of the clock and pulses through clock cycle by cycle one by one until the end. When no more pulse is detected the module will end. The horizontal counter works with the 25MHz clock and also uses an enable that will trigger the vertical counter. The count starts at zero and will be high as long as its within the allowed range. Once the horizontal reaches 800 it will trigger the enable to high and then reset itself. If the enable is high the vertical counter will process and continue to process until it reaches 525. Once the vertical counter reaches the max of 525 it will reset itself to 0.

The top module and test bench work together to process all the hardware modules constructed in order to display a solid color on the monitor. The top level assigns the color of the screen and in this project, we are assigning all colors to high so that the result is a white screen. A different color can be displayed simply by adjusting the assign of the RGB color outputs to high and low as needed and you can take a look at Figure 6a for the truth table of colors in a VGA Controller. Another thing done in the top-level module is to kick off the vertical and horizontal counters. These counters start at 2 and 96 respectively and make sure to set the counter to high and low as it processes through all the display area. The test bench then sets off the clock for this project to get the counters processing and test the display of the VGA Controller. The constraint file is used to make available to components of the Nexys 4 DDR board that are going to be required for the purpose of this project. The available components are the clock and VGA Connector making sure to assign VGA connectors 0-3 for each RBG and one for Hsync and Vsync.

Questions for the conclusion:

1. How much have you implemented in this project?

This project is the implementation of a complete VGA Controller. It includes six separate files required to get the project to process successfully.

2. What is left to be implemented that you could not do?

Some things that can be implemented that I did not do Is create a video game with this VGA controller. I was looking into creating a FGPA Snake game which is the model of the classical Nokia snake game, but I did not have enough time to dedicate to this project nor to learn enough HDL language to be able to take on such project.

3. What are some difficulties and challenges faced for this project?

Some difficulties that I faced while creating the project was finding research on how to display a solid colored screen. Most VGA resources were for in depth projects such as Video Synthesis, Image decoding, and different gaming projects in FPGA.

9. About the Author

Angelica Reyes is a student at the University of North Texas in the college of Computer Engineering. She is currently in her last year as a senior in the university and will be getting her second degree for the focus in computer engineering and science. She will be receiving her bachelor's degree in Computer Engineering in December Fall of 2020. Her goal is to be able to get a toe into the FPGA industry and combine her 5-year knowledge of working as a Data Analyst in software company for future career possibilities.

REFERENCES

1. Encyclopaedia Britannica, The Editors of. "VGA." *Encyclopædia Britannica*, Encyclopædia Britannica, Inc., 2020, www.britannica.com/technology/VGA.
2. "Design Challenge 6." Edited by Digilent Publisher, *Learn.Digilentinc*, 2014, learn.digilentinc.com/Documents/269.
3. "Building a Video Controller: It's Just a Pair of Counters." *The ZipCPU by Gisselquist Technology*, Gisselquist Technology LLC, 29 Nov. 2018, zipcpu.com/blog/2018/11/29/llvga.html.
4. Wikipedia contributors. "Video Graphics Array." *Wikipedia*, 25 Oct. 2020, en.wikipedia.org/wiki/Video_Graphics_Array.
5. Martha. "Nexys 4 DDR Reference Manual." *Digilent Documentation*, 8 Nov. 2018, reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual.
6. Hoangtam1987. "History of Video Graphics Adapter (VGA)." *HOÀNG TÂM*, 12 July 2011, hoangtam1987.wordpress.com/2011/06/10/video-graphics-adapter-vga.
7. Thoughts, Embedded. "Driving a VGA Monitor Using an FPGA." *Embedded Thoughts*, 30 Dec. 2016, embeddedthoughts.com/2016/07/29/driving-a-vga-monitor-using-an-fpga.
8. "FPGA Open Architecture Design for a VGA Driver." *ScienceDirect*, 1 Jan. 2012, www.sciencedirect.com/science/article/pii/S2212017312002642.
9. "720p Video from an FPGA." *Development Log by Martin Atkins*, 8 Sept. 2018, log.martinatkins.me/2019/09/22/video-timing-in-verilog.
10. EETimes. "EETimes - Design Recipes for FPGAs - A Simple VGA Interface -." *EETimes*, 9 Jan. 2008, www.eetimes.com/design-recipes-for-fpgas-a-simple-vga-interface/#.
11. "VGA Controller (VHDL) - Logic - Eewiki." *Eewiki*, 2020, www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278.
12. Baumann, Steve. "FPGA to VGA with VHDL – Steve's World of Making." *Steve's World of Making*, 2016, www.steve-baumann.de/?p=64.
13. "11. Design Examples — FPGA Designs with VHDL Documentation." *Meher Krishna Patel*, 2017, vhdlguide.readthedocs.io/en/latest/vhdl/dex.html.
14. "Verilog Tutorial." *ChipVerify*, 2020, www.chipverify.com/verilog/verilog-tutorial.

15. L., Alberto. “How to Create a Testbench in Vivado to Learn Verilog.” *Mis Circuitos*, 15 Apr. 2020, miscircuitos.com/how-to-create-a-testbench-in-vivado-to-learn-verilog-or-vhdl.