Comp 541 Digital Logic and Computer Design
Prof. Montek Singh
Spring 2019

Lab #4: VGA Display Timing Generator

Issued Fri 12/31/20; Due Fri 2/7/19 (11:59pm)

You will learn the following in this lab:

- Designing counters that count at a fraction (power of 2) of clock frequency
- Understanding how VGA displays work
- Generating timing signals that can drive a VGA display monitor (using a 2D *xy*-counter)
- Understanding how color values are encoded
- Understanding how `parameter` and `` `define `` are used in Verilog to specify parameters and constants

---

In this assignment, you will design a VGA Display interface.

Part 1: Designing a VGA Timing Circuit.

Study the VGA timing specification in the board manual (Nexys 4 manual pp. 13-17, Nexys 4 DDR manual pp. 14-17). Also, look at the website for VGA specifications whose link is provided on the class website.

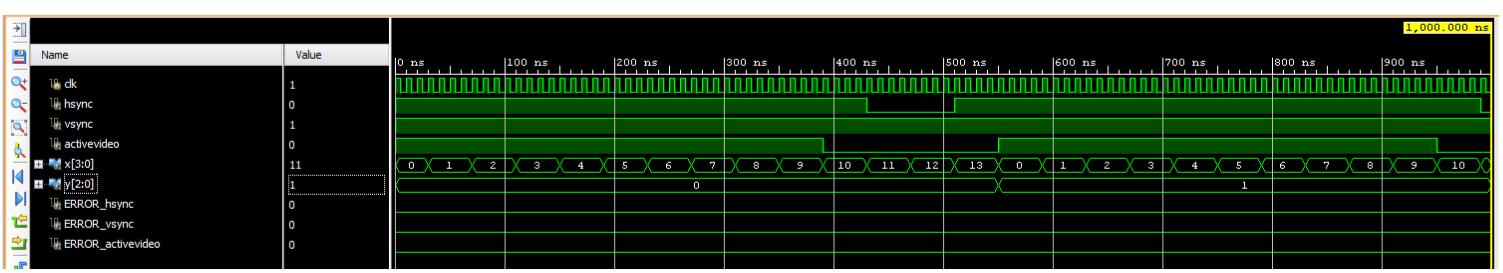
Use the template provided to design a VGA timer. First, let's design a "toy" display that has 10 columns and 4 rows. The specifications for this are given in the file **`display10x4.vh`**. Note the following:

- Understand the use of `` `include `` to include another source file.
- Files like **`display10x4.vh`**, which are used to provide parameters for your design, are called "Verilog Header" files, and are named with the "**`.vh`**" extension. When you create them or add them in Vivado, be sure to select their type as Verilog Header, or else the tool will not be able to locate them during simulation or synthesis.
- Understand the use of `` `define `` to define text substitutions. The right-hand side of a `` `define `` does a literal text substitution for the value being defined (like a search-and-replace!).

A Verilog template for `vgatimer` is provided on the website, and copied below:

```
`timescale 1ns / 1ps
`default_nettype none
`include "display10x4.vh"

module vgatimer(
   input wire clk,
   output wire hsync, vsync, activevideo,
   output wire [`xbits-1:0] x,
   output wire [`ybits-1:0] y
   );

   // These lines below allow you to count every 2nd clock tick and 4th clock tick
   // This is because, depending on the display mode, we may need to count at 50 MHz or 25 MHz.

   logic [1:0] clk_count=0;
   always_ff @(posedge clk)
      clk_count <= clk_count + 2'b 01;

   wire Every2ndTick = (clk_count[0] == 1'b 1);
   wire Every4thTick = (clk_count[1:0] == 2'b 11);


   // This part instantiates an xy-counter using the appropriate clock tick counter
   // xycounter #(`WholeLine, `WholeFrame) xy(clk, Every2ndTick, x, y);  // Count at 50 MHz

   xycounter #(`WholeLine, `WholeFrame) xy(clk, Every4thTick, x, y);  // Count at 25 MHz


   // Generate the monitor sync signals
   assign hsync = [redacted]
   assign vsync = [redacted]
   assign activevideo = [redacted]

endmodule
```

Use the test fixture provided (`vgatimer_10x4_test.sv`) to test your VGA timing circuit. When you first run the simulator, it will simulate only until 1 microsecond (shown as `1,000.000 ns` above the waveforms). This time is sufficient only to see the first horizontal sync pulse, so we will lengthen the simulation time. But for now, your simulation output should look *exactly* like the following.

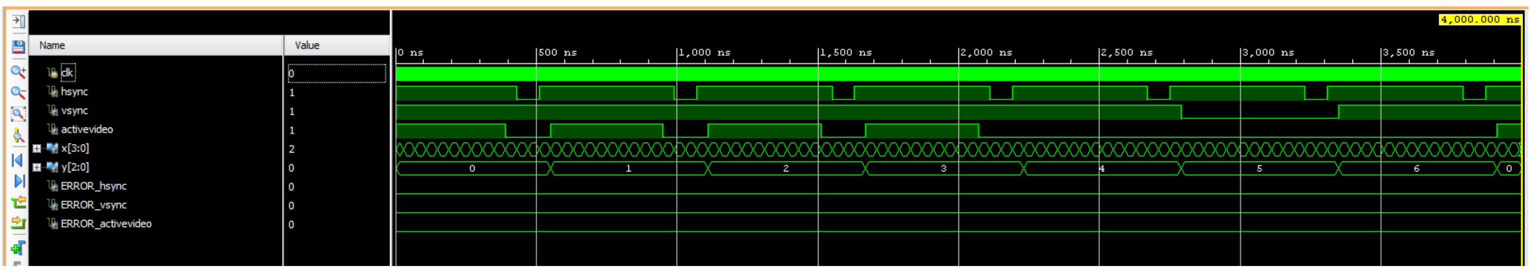


Observe carefully the start and stop times of the `hsync` and `activevideo` pulses in terms of the `x` values of the counter, and make sure you do not have an off-by-one error! This is a "self-checking" tester, with the correct expected outputs built into it! If there is an error, one of the ERROR* signals will turn red.

Next, we will lengthen the simulation duration to 4 microseconds. From the top menu bar, select *Run* → *Run All,* which runs the simulation until it encounters the `$finish` at time 4000 nanoseconds (or 4 microseconds).

Now let us try to find the vertical sync pulse. Click on `vsync` signal (either in the *Name* column or on its waveform), and then clock on the "next transition" icon, which should automatically scroll to the start of the vertical sync pulse. You can click on this icon again to find the end of the pulse. You can also click on the "previous transition" icon immediately to its left to go back to the start of the pulse. Use the zoom in and zoom out buttons so you see the entire vertical sync pulse within the window. You should see the following waveforms.

Once again, observe carefully the start and stop times of the `vsync` and `activevideo` pulses in terms of the `x` and `y` values of the counter, and make sure you do not have an off-by-one error! If there is an error, one of the ERROR* signals will turn red.



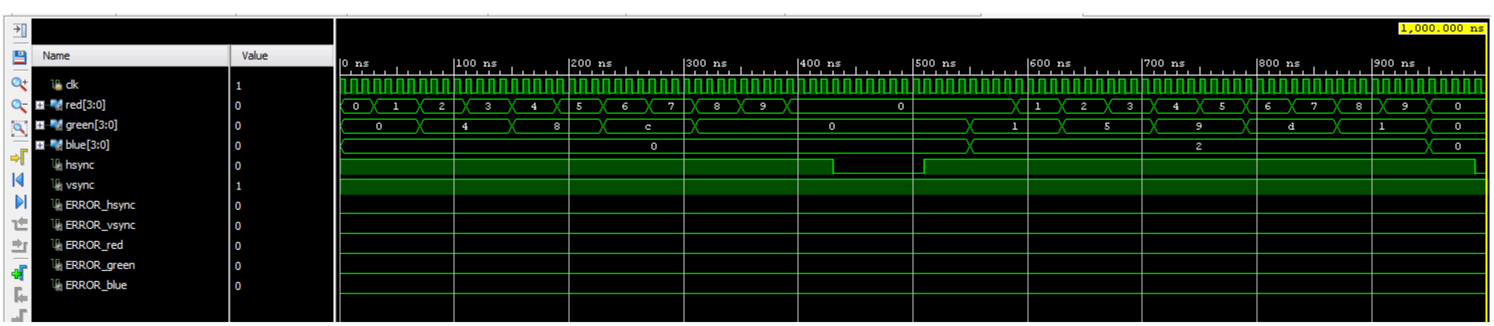
Part 2: Driving the display.

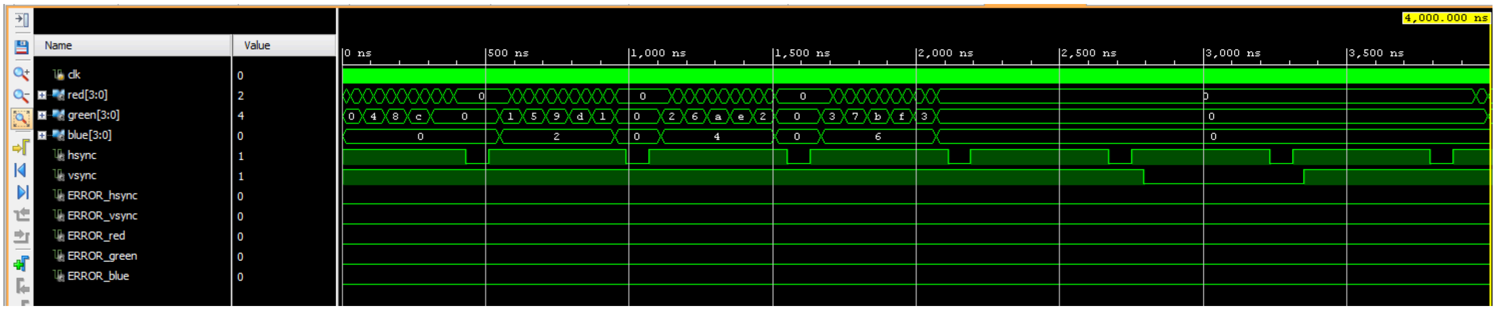
Let us now use this VGA timing generator, and drive the display. We do not yet have anything nice to display. So, let us display some random color values, in a pattern that is easy to recognize if it is correctly displayed.

A Verilog template for `vgadisplaydriver` is provided on the website and copied below.

```
`timescale 1ns / 1ps
`default_nettype none
`include "display10x4.vh"

module vgadisplaydriver(
    input wire clk,
    output wire [3:0] red, green, blue,
    output wire hsync, vsync
    );

   wire [`xbits-1:0] x;
   wire [`ybits-1:0] y;
   wire activevideo;

   vgatimer myvgatimer(clk, hsync, vsync, activevideo, x, y);

   assign red[3:0]   = (activevideo == 1) ? x[3:0] : 4'b0;
   assign green[3:0] = (activevideo == 1) ? {x[2:1],y[1:0]} : 4'b0;
   assign blue[3:0]  = (activevideo == 1) ? {y[2:0],1'b0} : 4'b0;

endmodule
```

Simulate using the test fixture provided on the website, `displaydriver_10x4_test.sv`. Verify that the simulation output is exactly as expected, according to the values in the **`display10x4.vh`** file. You should see *exactly* the following output waveforms (for 1 and 4 microseconds, respectively):





Do not proceed if your waveforms do not match the above figure *exactly*. An off-by-one error could easily cause lack of synchronization between your circuit and the monitor. Once again, if there is an error, one of the ERROR* signals will turn red.

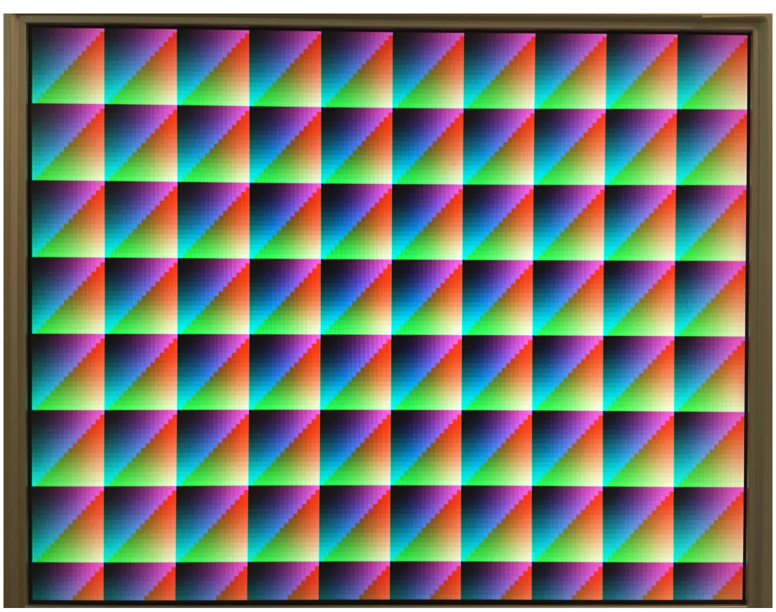
Part 3: Show an interesting pattern on an actual monitor

Once your 10x4 display driver is working correctly, select a *real* set of timing values by **changing the include file** to **`display640x480.vh`** in *both* `vgatimer.sv` and `vgadisplaydriver.sv`. **Include the constraints files** (two XDC files on the website, one for the clock input and one for VGA outputs). Program the design onto the board, connect the VGA monitor and see if everything works! You should see a tiny grayish pattern on the monitor.

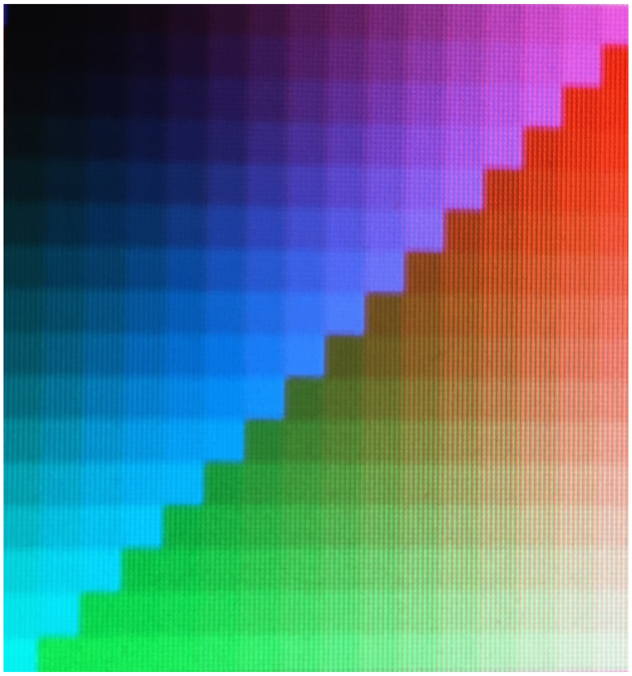
Note: If your simulation worked fine for the 10x4 driver, chances are that your hardware implementation will simply work when you make the changes specified in the previous paragraph. However, if your display does not work, you may need to debug it via simulation. A tester is available on the website (**`displaydriver_640x480_test.sv`**). Since this tester simulates the full 640x480 display resolution, the tool may feel very slow or at times non-responsive. That is why you are encouraged to thoroughly debug the design using the 10x4 resolution first, and only use the 640x480 tester as a last resort.

Now, modify the three lines in `vgadisplaydriver` that generate the RGB values, so as to show the pattern below. Each "box" in the pattern is 64x64 pixels. Within each box, the value of red increases from 0 to 15 from left to right (incremented once every four pixels); similarly the value of green increases from 0 to 15 from top to bottom (incremented once every four pixels). The value of blue increases in the diagonal direction, from top-left towards bottom-right, from 0 to 15 (incremented once every four pixels) *twice* (once from top-left to the middle, and then again from the middle to the bottom-right).

Hint: This is actually a very simple exercise! Do not overthink!



Here is an enlarged view of one box:



The website has more pictures that show the three color components separated out for your convenience.

What to submit:

- **The following Verilog sources for Part 3: xycounter.sv, vgatimer.sv, and vgadisplaydriver.sv.**
- **Show a working demo of your design for Part 3 during the lab session on Fri Feb 7 if completed by that time. If you complete it after the lab session, please attach a photo/video (phone camera is fine) to your submission.**

How to submit: **Please submit your work by email by 11:59 pm on Fri Feb 7 as follows:**

- **Send email to: `comp541-submit-s20@cs.unc.edu`**
- **Use subject line: Lab 4**
- **Include the attachments as specified above**
