The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 541 Digital Logic and Computer Design
Fall 2014

Lab #7: VGA Display Timing Generator
Issued Wed 10/1/14; Due Wed 10/8/14 (submit by 11:59pm)

You will learn the following in this lab:

- Designing counters that count to bounds that are not powers of 2
- 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 using 8 bits
- Understanding how `parameter` and `` `define `` are used in Verilog to specify parameters and constants

---

Part 0: Modify the counter from Lab 6 to display 4-digit decimal numbers

This modified version should still have four digits, but each digit wraps around to 0 once it reaches 9. Thus, the behavior of a digit should be as follows:

- A digit is incremented on the next positive edge of clock *only if* all of the digits to its right are currently at '9'. The rightmost digit is incremented on each clock tick.
- Incrementing is done as follows: If a digit is at '9', its new value is '0', else one is added to its value.

Name the Verilog file `bcdcounter.v`. For this part, we are not interested in counting down, only counting up is required. Also, pause/resume behavior is not needed (but you can leave it in if you'd like to).

Parts 1-3 below involve building a VGA Display interface.

Part 1: Designing a VGA Timing Circuit.

Study the VGA timing specification in the board manual (Nexys 3 manual pp. 15-17, or Nexys 4 manual pp. 13-17). Also, look at the website for VGA specifications whose link is provided on the class website.

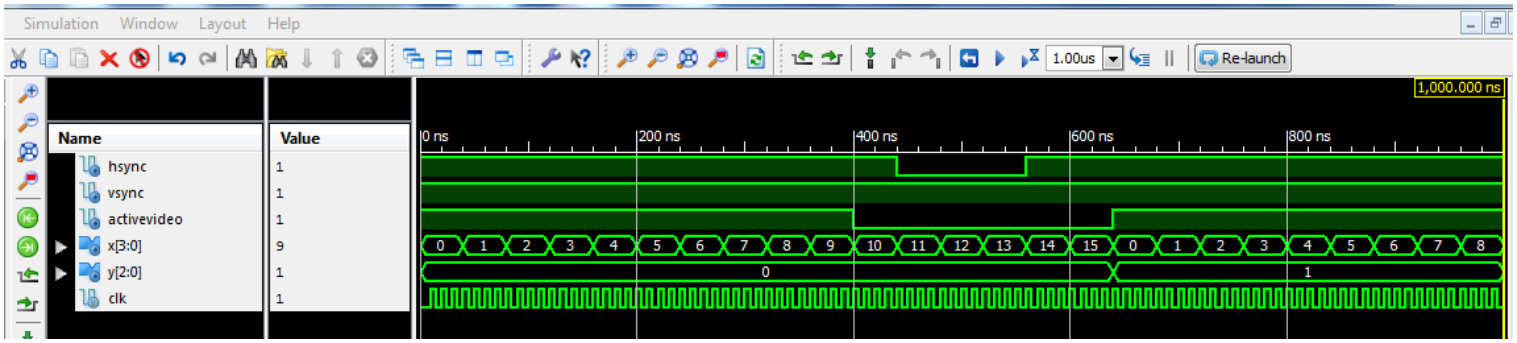
Use the template provided to design a VGA timer. First, let's design a "toy" display that has 10 columns and 4 rows. The specifications for this are given in the file **`display10x4.v`**. Note the following:

- Understand the use of `` `include `` to include another source file.
- 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!).
- Understand the use of the `parameter` statement, and how default values are specified, as well as new values specified during instantiation can override the defaults.

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

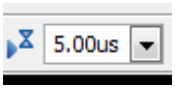
```
`include "display10x4.v"

module vgatimer(
   input clk,
   output hsync,
   output vsync,
   output activevideo,
   output [`xbits-1:0] x,
   output [`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 100MHz
   // or 50 MHz or 25 MHz.

   reg [1:0] clk_count=0;
   always @(posedge clk)
      clk_count <= clk_count + 2'b01;
   assign Every2ndTick = (clk_count[0] == 1'b1);
   assign Every4thTick = (clk_count[1:0] == 2'b11);


   // This part instantiates an xy-counter using the appropriate clock tick counter
   // xycounter xy(clk,          1'b1, x, y);  // Count at 100 MHz
   // xycounter 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_test.v`) to test your VGA timing circuit. When you first run the simulator, it will simulate only until 1 microsecond (shown as `1.00us` 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.

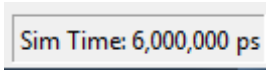


Observe carefully the start and stop times of the `hsync` and `activevideo` pulses in terms of the `x` values of the counter, and make sure you do not have an off-by-one error!

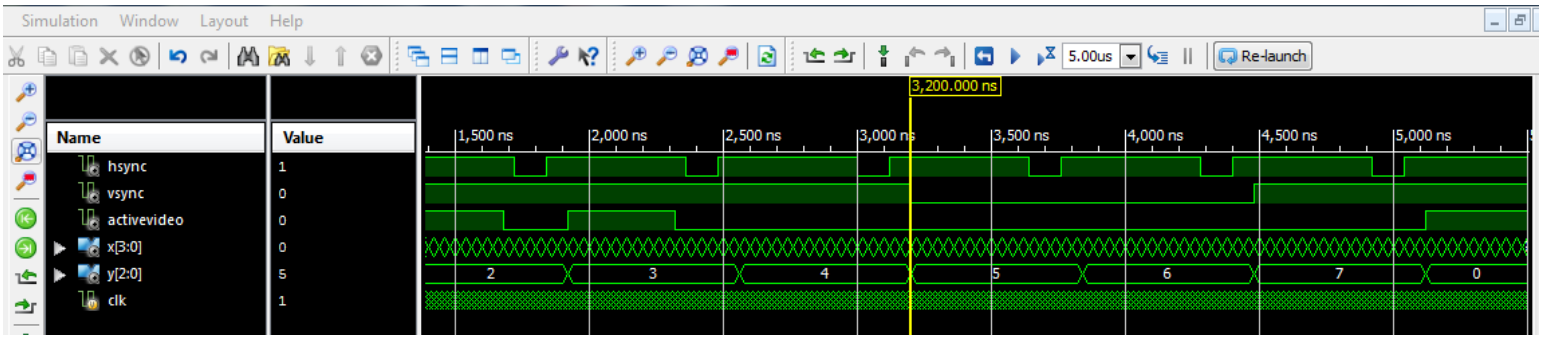
Next, we will lengthen the simulation duration to 6 microseconds. Change the simulation time shown above the waveforms to show `5.00us`, and then click the "play-hourglass" icon



immediately to its left, which tells the simulator to *add* another 5 microseconds to the simulation. You should see the simulation time shown at the bottom right of the screen change to 6 microseconds (6 million picoseconds).



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 "find 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 "find 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!

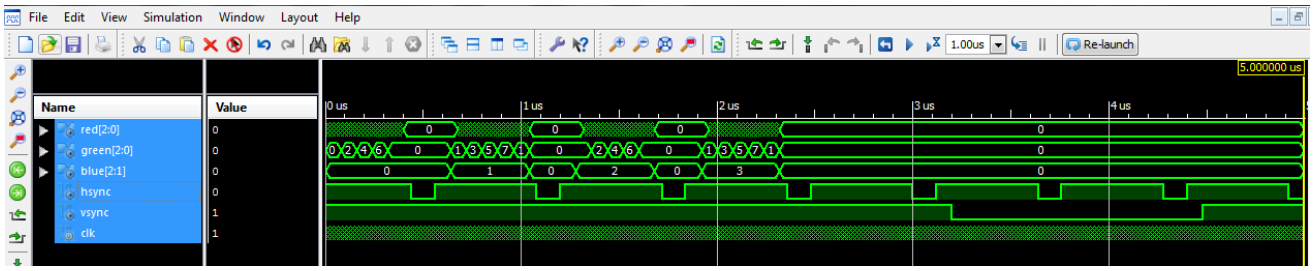
Part 2: Driving the display.

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. Be sure to edit it to match the board you are using.

```
module vgadisplaydriver(
    input clk,
    // output [2:0] red,              // Nexys 3
    // output [2:0] green,            // Nexys 3
    // output [2:1] blue,             // Nexys 3
    // output [3:0] red,              // Nexys 4
    // output [3:0] green,            // Nexys 4
    // output [3:0] blue,             // Nexys 4
    output hsync,
    output vsync
    );

   wire [10:0] x;
   wire [10:0] y;

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

   // For Nexys 3, use the following for 8-bit color
   assign red[2:0]   = (activevideo == 1) ? x[2:0] : 3'b0;
   assign green[2:0] = (activevideo == 1) ? {x[2:1],y[0]} : 3'b0;
   assign blue[2:1]  = (activevideo == 1) ? y[1:0] : 2'b0;

   // For Nexys 4, use the following for 12-bit color
   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[3:0] : 4'b0;

endmodule
```

Simulate using the test fixture provided on the website, `vgadisplaydriver_test.v`. Verify that the simulation output is exactly as expected, according to the values in the **`display10x4.v`** file. If you simulate to 5 microseconds, you should see *exactly* the following output waveforms.



Do not proceed if your waveforms do not match the above figure *exactly*. An off-by-one error could easily cause of lack of synchronization between your circuit and the monitor.

Once your 10x4 display driver is working correctly, select a *real* set of timing values by using the file **`display640x480.v`** (or **`display800x600.v`**) instead. Program the design onto the board, connect the VGA monitor and see if everything works!

Part 3: Show an interesting pattern,

Modify the `vgadisplaydriver` to show a different pattern, something interesting. Lines, stripes, checkerboards and diagonals are all fine. Try higher-order bits of *x* and *y* to derive RGB values. You can also use arithmetic and Boolean operators.

What to submit:

- **The four Verilog sources for Parts 0, 1, 2 and 3: bcdcounter.v, xycounter.v, vgatimer.v, and vgadisplaydriver.v.**
- **Show a working demo of your design for Part 0 and Part 4 during the lab session on Friday, October 10.**

How to submit: **Please submit your work by email by 11:59pm, Oct 8, 2014 as follows:**

- **Send email to: `comp541submit-cs@cs.unc.edu`**
- **Use subject line: Lab 7**
- **Include the four attachments as specified above**
