CS/EE 3710 – Computer Design Lab
Lab 3 – VGA

Due Thursday September 30, 2010

Lab Objectives

The objectives of this lab are to build and demonstrate a simple VGA controller driving a VGA display. I had a colleague who liked to say: “There are two kinds of engineers – those who think VGA is hard, and those who understand VGA.” That is to say, it’s a little intimidating when you first see it, but it’s really a very simple protocol for sending output information to a video screen. The steps are:

1. Read and understand the VGA specification. I recommend breaking this down into a VGA control circuit that generates the timing signals, and a bitGen circuit that defines what color each pixel should be as the information is being sent to the display.

2. Design and implement a VGA control/timing circuit and simulate that circuit using ISE.

3. Design and demonstrate a very simple bitGen circuit that takes input from three switches on the Spartan3e board and drives the entire screen to the color represented by the values on those switches.

4. Design and demonstrate a slightly more interesting bitGen2 circuit that drives the display with the output from your TBird turn signal state machine. That is, in addition to lighting up the LEDs on the Spartan3e board, you will drive the VGA display as a TBird output device.

VGA Basics

The term VGA really means one of two things depending on how you use the acronym. It’s either a standard 15-pin connector used to drive video devices (e.g. a VGA cable) or it’s the protocol used to drive information out on that cable (e.g. a VGA interface spec.). The interface defines how information is sent across the wires from your board to the VGA device. The cable defines which pins you use on the standard connector for those signals.

There are some slides on VGA on the class web site, and there is a nice description of VGA (both the connector and the protocol) in the Spartan3E board User Guide, also linked to the class web site. The most basic thing to know about VGA is that it is a protocol designed to be used with analog CRT (cathode ray tube) output devices. On these devices the electron beam moves across the screen from left to right as you’re looking at the screen at a fixed rate (the refresh rate defines how fast the beam moves), and also moves down the screen from top to bottom at a fixed rate. While it’s moving across and down the screen, you can modify the Red, Green, and Blue values on the VGA interface to control what color is being painted to the screen at the current location. So, painting a certain color on the screen is as easy as keeping track of where the beam is, and making sure the R, G, and B signals are at the right values when the beam is over the point on the screen where you want that color.

If you don’t do anything to stop it, the beam will move to the right and bottom of the screen and get stuck there. You can force the beam to move back to the left by asserting an active-low signal called hSync (horizontal sync). You can force the beam to move back to the top of the screen by asserting an active-low signal called vSync (vertical sync). Because the beam moves at a fixed

rate (defined by the monitor's refresh rate), you can keep track of where the beam is on the screen by counting clock ticks after the hSync and vSync signals.

So, the basics of the VGA control/timer circuit are just a pair of counters to count horizontal ticks and vertical ticks of the VGA clock. How many ticks are there? That depends on how fast your clock is, and how many pixels you want to paint during the time the beam moves across the screen. The basic (ancient) standard for "plain" VGA is 640 pixels on each line, and 480 lines down the screen. This is "640x480" mode. Figure 1 shows a 640x480 screen, and the horizontal sync (hSync) timing required to make it work. After the hSync pulse, you must wait for a certain number of ticks before painting pixels to the screen. This gives the beam time to get back to the left and start moving forward again. This time is called the "back porch" because it's in back of the hSync timing pulse. Then you count 640 pixels as the beam moves. After the 640th pixel, you wait for some amount of time (this is the "front porch" because it's in front of hSync), then assert the hSync signal (asserted low) for a certain amount of time. Note that the figure in the Spartan3e User Guide has the front porch and back porch reversed. This figure is correct.

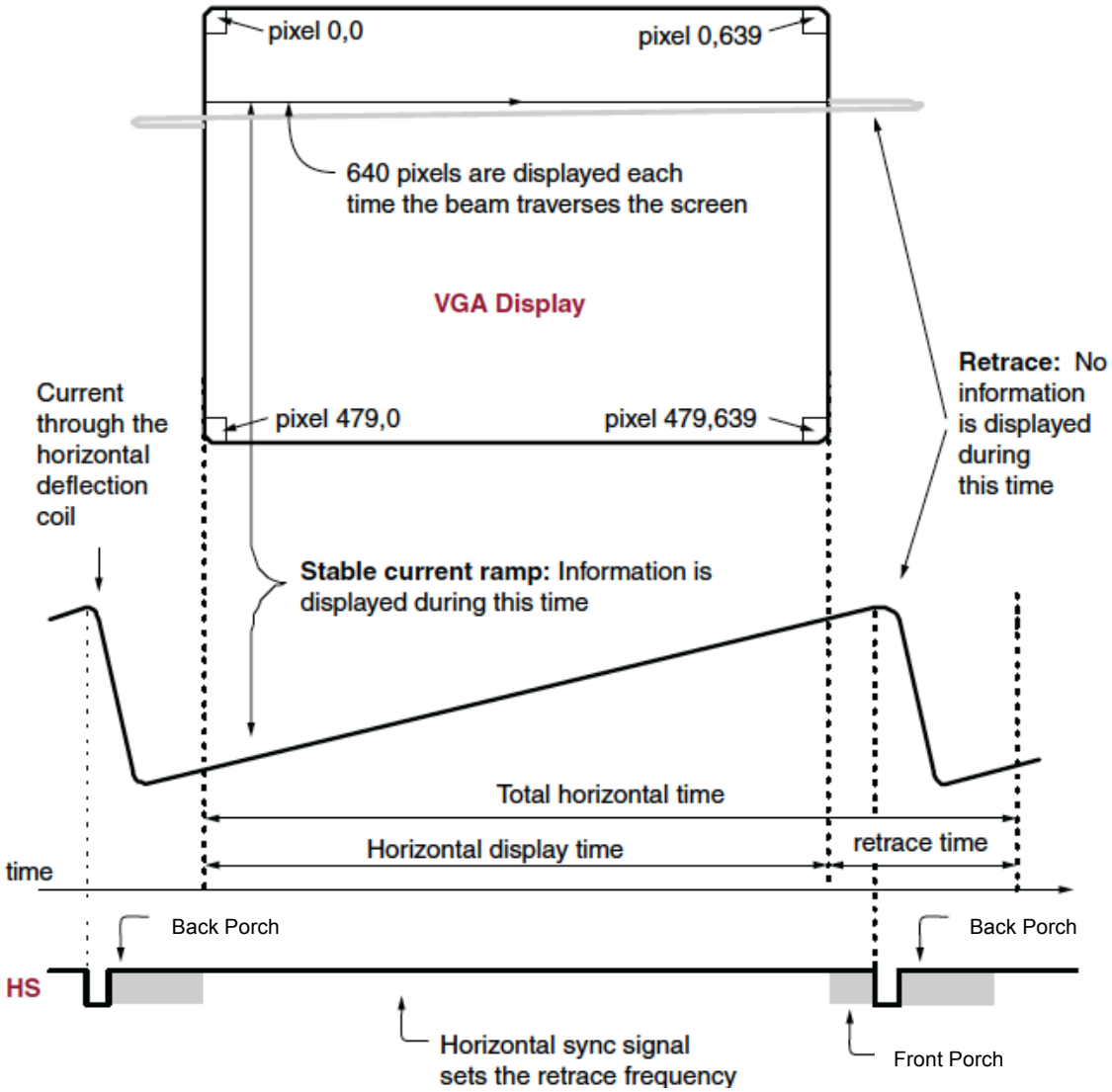

Figure 1: CRT VGA Horizontal Timing Example

The timing for all this depends on the monitor refresh rate. For a monitor with 60Hz refresh in 640x480 mode and a 25MHz pixel clock, you can use the timings in Figure 2. The timings in this figure define the display time (the time when the pixel is one of the 640 visible pixels in a line), pulse width (hSync or vSync), and the front porch and back porch timings. These times (in μs or ms) can also be measured in terms of the number of ticks of the 25MHz pixel clock, or in terms of the number of horizontal lines. That is, the vertical timing can be measured in terms of how many hSync pulses have been seen. The bottom line is that both the horizontal and vertical timing for the vgaControl are just counters. You may have to enable things or reset things or change things when the counters get to a certain value, but basically they're just counters.

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 μs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 μs	640
T_{PW}	Pulse width	64 μs	1,600	2	3.84 μs	96
T_{FP}	Front porch	320 μs	8,000	10	640 ns	16
T_{BP}	Back porch	928 μs	23,200	29	1.92 μs	48

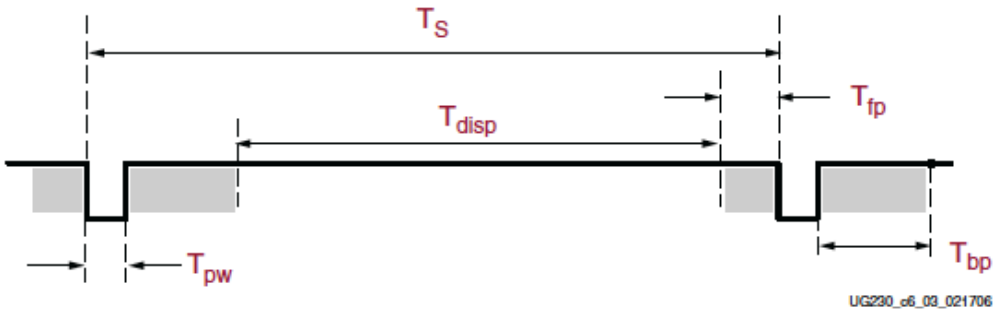

Figure 2: VGA Control Timing at 60Hz refresh and 25MHz pixel clock

I recommend designing a circuit that has two counters: one to keep track of horizontal location and one that keeps track of vertical location. The horizontal counter should count up in 25MHz ticks, and the vertical counter should count once for each full horizontal line.

Note that if you're going to use a purely synchronous design method (which I highly recommend!), you should NOT expect to have a 25MHz clock at your disposal. The system clock on the Spartan3e board is 50MHz. You could divide that by two and use that output as the 25MHz clock, but that would NOT be purely synchronous because you would be using a derived signal as your clock. Instead you should generate an enable signal that is asserted every two 50MHz clocks, and use that enable signal in your horizontal pixel counter. For example, the horizontal pixel counter might look like:

```
// An example of a counter that only counts when En is high.
// This particular En is assumed to be high for a single clock
// cycle when it's time to count
always@(negedge clr, posedge clk50MHz)
       if (clr == 0) count = 0;          // clear count if clr is asserted
       else if (En) count = count + 1;   // If En is high, count
              // otherwise, hold previous value (default)
```

Of course, there are potentially lots of details missing here (like when to wrap around back to 0, for example), but this is an example of how you might count at 25MHz if there was a 25MHz En signal available.

For your vertical counter, I recommend generating an enable signal from your horizontal counter once per line and letting your vertical counter count up by one only when it's enabled.

The main thing your VGA control circuit should do is generate hSync and vSync at the appropriate times based on the VGA timing. However, if you're going to know where the beam is on the screen so that your bitGen signal knows what color to put there, you also need to know which pixel and which line the beam is currently over. So, you also need to export the hCount and vCount signals so you can figure out which pixel is being painted. Finally, it's a good idea to generate a signal from your VGA control circuit that is asserted whenever the beam is in a "pixel bright" area of the screen. That is, you are NOT in a front porch, back porch, or sync section of the timing.

I recommend that your VGA control circuit have the following interface:

```
module vgaControl (
  input clk50MHz, clr,           // system clock and clear
  output hSync, vSync,           // sync signals to the VGA interface
  output bright,                 // asserted when the pixel is bright
  output reg [X:0] hCount, vCount // horiz and vertical count (you pick the X!)
                                 // these tell you where you are on the screen
);
```

The body of this module is, of course, up to you. It should generate the outputs based on the 640x480 60Hz refresh timing in Figure 2. You can validate the timing by simulating the vgaControl circuit in the ISE simulator.

Some things to consider when you define this module:

- Remember that hSync and vSync are asserted low
- hCount and vCount are used by your bitGen circuit so that it knows where you are on the screen. It's handy if the value of hCount is the value of the pixel. That is, if you start counting at 0 when you start the Horizontal Display Time from Figure 1, then the number on the counter will be the pixel number in the line (between 0 and 639).
- In the same way, if you start counting vertical lines in the Vertical Display Time portion of the vertical timing, then the value on the counter will be the line number (between 0 and 479)
- The bright signal can be either asserted high or low – it's used by your bitGen circuit to tell that you're in a section of the screen where you want to turn on the pixel. If you're "not bright" you should force the RGB output to all zeros so that you're not driving the screen.

The bitGen Circuit

This is the circuit that takes the hCount, vCount, and bright signals, and decides for each pixel what color should be on the screen. At a minimum a bitGen circuit would have the following interface:

```
module bitGen_minimum (
   input bright,          // asserted if the pixel is in the bright region
   input [X:0] hCount, vCount, // the horizontal and vertical counts
   output [2:0] rgb           // the RGB value of the (hCount,vCount) pixel
```

```
// there may be other inputs to help define what the pixel color is...
);
```

How does your bitGen circuit know what the color should be? There are lots of ways, but the three general techniques are:

1. **Bitmapped graphics:** in this technique you have a "frame buffer" that holds the RGB values for every pixel on the screen. You can address this buffer using the hCount and vCount signals to retrieve the color at each of the 640x480 locations on the screen. Generally a frame buffer is dual-ported so that, for example, a CPU can be putting data into the frame buffer and the VGA bitGen circuit can be reading it out at the same time. This is the most memory-intensive technique. 640x480 is 307,200 pixel locations on the screen. If you pack two three-bit rgb pixel values in each byte, that's still 153,600 bytes of storage to hold the whole 640x480 screen (150kBytes).

2. **Character/glyph graphics:** Instead of storing each of the pixels in the frame buffer, you can break the screen into chunks and store a pointer to a glyph in each screen chunk. For example, if you break the screen into 8x8 pixel chunks, then there are 80x60 chunk locations on the screen (this is 4800 locations total). If you have 256 possible 8x8 glyphs stored somewhere else, then you only need 4800 bytes (4.69kBytes) to store the screen (one 8-bit glyph pointer for each screen chunk), but at each location on the screen you are limited to one of the 256 8x8 glyphs. The glyphs themselves need to be in a separate memory that consists of 2k locations of 8-bits each. So, the total storage requirement is 6848 bytes (6.69kBytes). A lot less than 150kBytes, but less flexible. It's common to have alphanumeric characters and some graphic glyphs in the glyph memory. You can use the hCount and vCount values not only to tell which chunk you're in, but also which pixel of the glyph you're currently in. Hints – If the glyphs are 8 pixels across, which bits of the hCount counter define the pixel within the glyph? How many bits does it take to count to 8 pixels? Which bits of hCount define which glyph you're in? You only want those bits to change once every 8 ticks of the pixel clock.

3. **Direct graphics:** Instead of storing information about what's at each screen location, you can design a circuit that checks the current hCount and vCount and draws directly on the screen when you're in the right spot. For example, if you want to draw a white box at a particular location on the screen, your bitGen circuit might have the following Verilog code:

```
parameter BLACK = 3'b 000, WHITE = 3'b111, RED = 3'b100;

// paint a white box on a red background
always@(*)
   if (~bright) rgb = BLACK;     // force black if not bright
   // check to see if you're in the box
   else if (((hCount >= 100) && (hCount <= 300)) &&
            ((vCount >= 150) && (vCount <= 350))) rgb = WHITE;
   else rgb = RED;              // background color
```

RGB Colors

The Spartan3e board has a VGA interface with one wire connected to the Xilinx part for each of the R, G, and B signals. This means you can make a generous eight colors on the screen by turning on combinations of the R, G, and B. Figure 3 shows the colors you can get with this simple interface.

VGA_RED	VGA_GREEN	VGA_BLUE	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Figure 3: Eight colors from one-bit each of R, G, and B.

VGA Connector

The standard VGA connector pin assignment can be found in the Spartan3e User Guide. See Figure 6.1 on page 55. The pin assignments for using the connector are also in the User Guide but I'll repeat them here too in Figure 4. The HSYNC and VSYNC signals come from your VGAControl module. The RED, GREEN, and BLUE come from your bitGen module.

```
NET "VGA_RED"   LOC = "H14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_GREEN" LOC = "H15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_BLUE"  LOC = "G15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_HSYNC" LOC = "F15" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
NET "VGA_VSYNC" LOC = "F14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = FAST ;
```

Figure 4: UCF Constraints for the Spartan3e board VGA connector.

Assignments Specifics

The previous text is to help you understand the VGA spec. The specifics of this assignment are as follows.

1. Design and simulate a vgaControl module to generate hSync, vSync, bright, hCount, and vCount signals. The module should take in the 50MHz system clock on pin C9 that you've used for previous labs. The VGA timing should be based on the 60Hz refresh 640x480 VGA timing described in this handout. Simulate this module with ISE.

2. Design a bitGen circuit that takes input from vgaControl, and from three switches on the Spartan3e board (use switches as follows: SW2 = Red, SW1 = Green, SW0 – Blue). Based on the value of these switches (i.e. Figure 3), paint the entire 640x480 screen in this color. Demonstrate this circuit to the instructor or TA.

3. Design a bitGen2 circuit that takes input from vgaControl, and also from your TBird FSM. Use a bitGen display technique of your own choosing, and display the L and R turn signal lights on the VGA screen. That is, something on the screen should light up when the LEDs on the board light up. It should be a VGA version of your TBird circuit from Lab1. Demonstrate this circuit to the instructor or TA.

4. Turn in all your commented Verilog code (vgaControl, bitGen, bitGen2, and any additional Verilog you generated), any testbench code you used for testing, and any schematics that you might have used to combine the vgaControl and the various bitGen modules. If you modified anything in your TBird FSM, turn in those files too.