School of Engineering Science
Simon Fraser University
ENSC 350 – Digital Systems Design
Spring 2017

Lab 3: VGA Controller / Drawing Lines

Working week: February 6-10, 2017
Marking week: February 13-17, 2017

In this lab, you will get more experience creating datapaths and state machines. Compared to Lab 2, the algorithm you will implement will be slightly more complex. If you understood Lab 2, you should have no problem "stepping up" to this lab. You will also learn how to use an embedded core that we will give you; this is common practice in industrial design – taking cores that are either purchased or written by another group and incorporating them into your design. The embedded core we will give you is a VGA adapter, which will allow you to create a circuit that draws to a VGA screen.

The top level diagram of your lab is as follows. The VGA Core is the part given to you, so all the excitement will be in the block labeled "your circuit". This handout first explains how to use the VGA Core, and then specifies what your circuit should do.

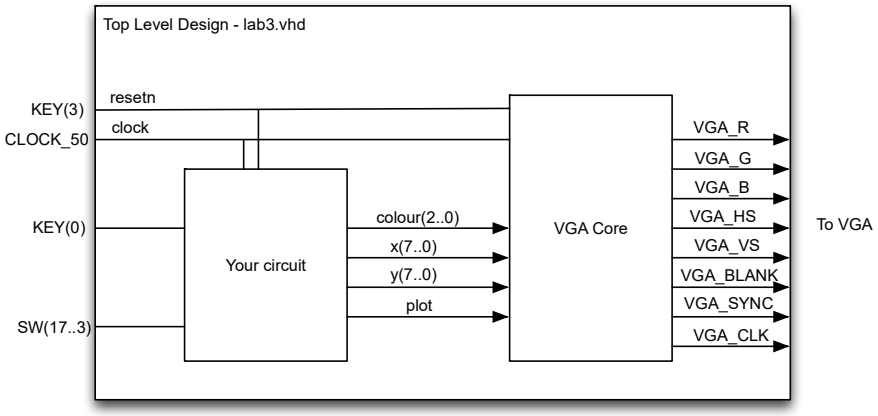


Figure 1: Overall block diagram

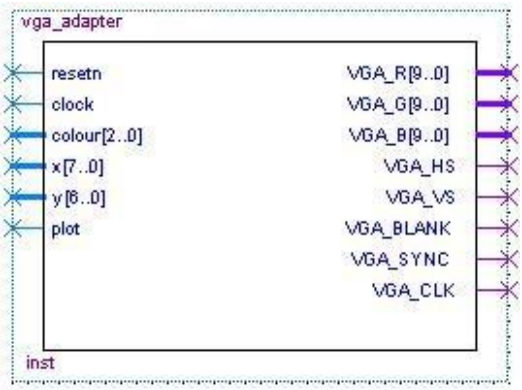
Task 1: Understand the VGA Adapter Core:

The VGA Adapter was created at the University of Toronto for a course similar to EECE 353. The following describes enough for you to use the core; more details can be found on University of Toronto's web page: http://www.eecg.utoronto.ca/~jayar/ece241_07F/vga Some of the following figures have been taken from that website (with permission!).

In order to save on the limited memory on DE2 board, the VGA adapter has been setup to display a grid of 160x120 pixels, with the interface shown in Figure 2:

Commented [LS1]: 1010 0000-> 1001 1111
0111 1000-> 0111 0111

Inputs
From your circuit
(You will be working with these)



Outputs
To DAC & Monitor
(You do not need to control these)

Figure 2: VGA adapter as a black box

Inputs:

Resetn	Active low reset signal. Digital circuits with state elements should always contain a reset.
Clock	Clock signal. The VGA core must be fed with a 50MHz clock to function correctly.
colour(2 downto 0)	Pixel colour (3 bits). Sets the colour of the pixel to be drawn. The three bits indicate the presence of Red, Green and Blue components for a total of 8 colour combinations.
x(7 downto 0)	X coordinate of pixel to be drawn (8 bits) – supported values $0 \leq x < 160$.
y(6 downto 0)	Y coordinate of pixel to be drawn (7 bits) – supported values $0 \leq x < 120$.
Plot	Active high plot signal. Raise this signal to cause the pixel at (x,y) to be set to the specified colour on the next rising clock edge.

Outputs:

VGA_CLK	VGA clock signal.
VGA_R(9 downto 0) **VGA_G(9 downto 0)** **VGA_B(9 downto 0)**	Red, Green, Blue components of display (10 bits). These signals are connected to the Digital-to-Analog Converter (DAC) on the DE2 board before transmitting to the monitor.
VGA_HS **VGA_VS** **VGA_SYNC** **VGA_BLANK**	VGA control signals.

Note that you will connect the outputs of the VGA core directly to appropriate output pins of the FPGA.

You can picture the VGA pixel grid as shown in Figure 3. The X/Y position (0,0) is located on the top-left corner and (159,119) pixel located at the other extreme end. (Note: Depending on your display, the upper row of the display may be "clipped" so you can only partially view the top row of pixels on your screen.)

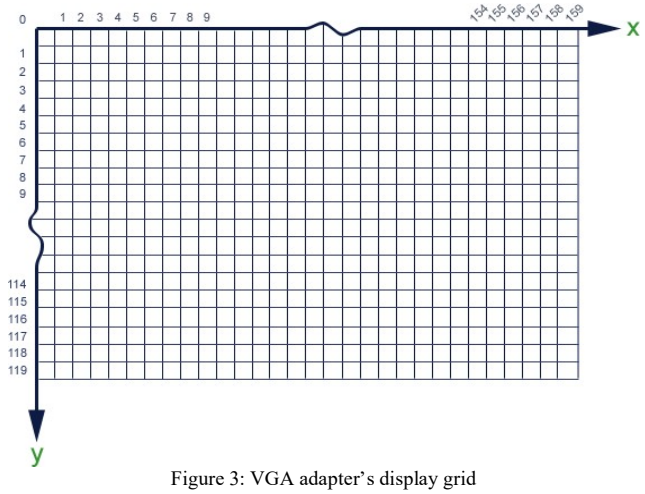


Figure 3: VGA adapter's display grid

To turn on a pixel, you drive the "x" input with the x position of the pixel, drive the "y" input with the y position of the pixel, and the "colour" input with the colour you wish to use. You then raise input plot. At the next rising clock edge, the pixel turns on. In the following timing diagram (from the UofT Website), two pixels are turned on: one at (15, 62) and the other at (109,12). As you can see, the first pixel drawn is red and is placed at (15, 62). The second is a yellow pixel at (109, 12). It is important to note that, at most, *one pixel can be turned on each cycle*. Thus, if you want to turn on *m* pixels, you need *m* cycles.

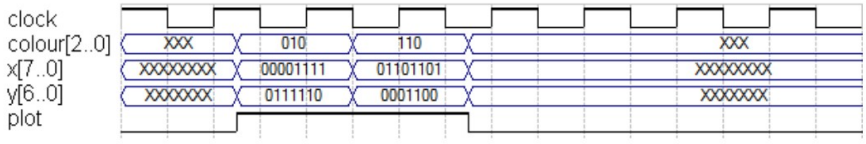


Figure 4: Timing Diagram

The source code for the VGA adapter is included as part of the lab zip file. This core is written in Verilog, not VHDL. Remember from our class discussions that this is ok; when making a structural description, you can include Verilog modules just as you can VHDL modules (this "mixed language" design approach is common in industry). The files are "open"; so for those of you interested in learning Verilog, this is possibly an opportunity.

The Verilog files describing the VGA adapter can be included into Altera Quartus II project just like the VHDL files. If you read them, you would notice that the "module" definition in Verilog is similar to "entity" in VHDL. This means you can instantiate modules in the Verilog files as components in your VHDL files.

To help you understand the VGA core, check out and instantiate the `vga_demo.vhd` file along with the VGA controller files (`vga_adapter.v`, `vga_controller.v`, `vga_address_translator.v`, and `vga_pll.v`.). This file does nothing but connect the core I/O to switches so you can experiment. We suggest you download this file, understand it, and try it out to make sure you understand how the inputs

of the core work. Note that the `vga_demo.vhd` file is *only* used to help you understand the core; you will *not* use it when constructing your circuit in Tasks 2, 3, or 4.

Task 2: Fill the Screen (2 marks):

You will first create a simple circuit that fills the screen. Each column will be a different colour (repeating every 8 columns). Remember that you can only turn on one pixel at a time, so you would need an algorithm something like:

```
for x = 0 to 159 {
        for y = 0 to 119 {
                turn on pixel (x, y) with colour ( x mod 8)
        }
}
```

(Note that you do not need the Bresenham algorithm here, you can just cycle through all pixels as in the above algorithm).

You are to create a circuit that implements the above algorithm. A skeleton file **lab3.vhd** is available as part of the lab zip file. Your design should have an asynchronous reset that will be driven by KEY(3). You don't need to use KEY(0) or any of the switches in this task as your circuit will be clocked by the CLOCK_50 (MHz) source.

Test your design on the DE2-115. You need your DE2-115 board with a USB cable, a VGA cable, and a VGA-capable display. In the Digital Design Lab, a VGA cable and LCD display are provided. Use the following files from the lab 3 folder as the basis of Task 2: `vga_adapter.v`, `vga_controller.v`, `vga_address_translator.v`, `and vga_pll.v`. Add these files to a new Quartus II project, import your pin assignments, then compile and program the design onto your DE2-115 board.

Use the VGA cable to connect your DE2-115 board to the VGA display. Most new LCD displays have multiple inputs, including DVI (digital) and VGA (analog). The LCD displays in Digital Design Lab have both; use the DVI input for the PC, while the VGA input is connected to a VGA cable for you to use. You can switch between the inputs using the display's input select button. Note: the VGA connection on your laptop is an **OUTPUT**, so **do not connect your laptop's VGA port to your DE2 board.**

Hint: Modelsim simulation will be very useful while debugging this task. Start by looking at your x and y counters.

(continued on next page…)

Task 3: Bresenham Line Algorithm (4 marks) :

The Bresenham Line algorithm is a hardware (or software!) friendly algorithm to draw lines between arbitrary points on the screen. The basic algorithm is as follows (taken from Wikipedia):

```
function line(x0, y0, x1, y1)
    dx := abs(x1-x0)
    dy := abs(y1-y0)
    if x0 < x1 then sx := 1 else sx := -1
    if y0 < y1 then sy := 1 else sy := -1
    err := dx-dy
    loop
        setPixel(x0,y0)
        if x0 = x1 and y0 = y1 exit loop
        e2 := 2*err
        if e2 > -dy then
            err := err – dy
            x0 := x0 + sx
        end if
        if e2 < dx then
            err := err + dx
            y0 := y0 + sy
        end if
    end loop
end function
```

In this task, you are to implement a circuit that does the following:

As in Task 2, the switch KEY(3) is an asynchronous reset. When the machine is reset, it will do the following:

1. The circuit will start clearing the screen (hint: the circuit from Task 2 can be used to clear the screen by changing the colour of each pixel to Black). Of course, clearing the screen will take 160*180 cycles.

2. Once the screen is cleared, perform the following algorithm:

    ```
    for i = 1 to 14 {
        draw line from ( 0, i*8) to ( 159, 120-(i*8) ) using colour gray(i mod 8)
    }
    ```

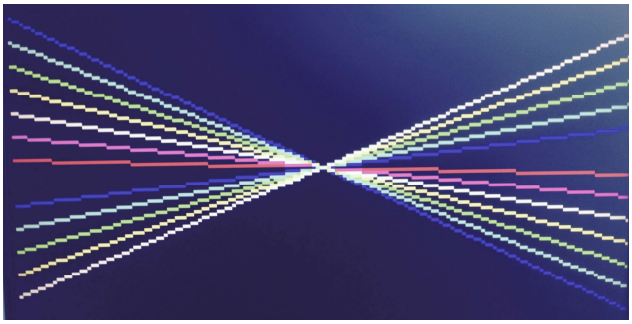
    The function *gray(i)* is a combinational logic function implementing the following:

    Gray(0) = 000
    Gray(1) = 001
    Gray(2) = 011
    Gray(3) = 010
    Gray(4) = 110
    Gray(5) = 111
    Gray(6) = 101
    Gray(7) = 100

    This "gray code" is such that each consecutive value differs in exactly one bit from the previous value (you can look up Gray codes online). In our design, this means each drawn line will be drawn in a colour that differs exactly one bit from the colour of the previous line and the next line (see the example screenshot below). Remember that each of the 3 bits of the colour

indicates the presence of red, green, and blue, so this technique will ensure that colours that are "similar" are drawn next to each other. For example, Red is encoded at 100 which would be drawn nearby "purple" (which is somewhat similar) which would be encoded as 101 (red+blue).

If you have done it correctly, the screen will look something like this:



It is important to note that you are using CLOCK_50, the 50Mhz clock, to clock your circuit. This is different than Lab 2 where you used a pushbutton switch for your clock.

Note that unlike Lab 2, we are not giving you a datapath here. With enough thinking, you should be able to come up with a datapath to implement the Bresenham Line algorithm. Remember what we talked about in Slide Set 7 and take a step by step (iterative) approach. It will greatly improve your likelihood of success.

Task 4: Adding a delay and looping the display (3 marks)

In Task 3, all fourteen lines are drawn without any delay between them. Modify your implementation such that the Bresenham Line Algorithm is drawing one line to the screen at a time with a delay between each of these lines. This is reminiscent of some of the original screen savers. Here's some pseudocode for what you want to do.

```
forever {
        for i = 1 to 14 {
                draw line from ( 0, i*8) to ( 159, 120-(i*8) ) using colour gray(i mod 8)
                wait 1 second
                erase the line that was just drawn
        }
}
```

In this implementation, only one line appears on the screen at a time. You can't use any keys or switches (other than a key to implement a reset signal if you like). ***Remember: no "for" loops should actually used to build this hardware***.

Challenge Task: (1 mark)

Challenge tasks are tasks that you should only perform if you have extra time, are keen, and want to show off a little bit. This challenge task is only worth 1 mark. If you don't demo the challenge task, the maximum score you can get on this lab is 9/10 (which is still an A+).

Finally, use Task 4, which draws one line to the screen at a time with a delay in between as the basis of a right angle triangle drawing algorithm. Instead of lines, you will generate triangles that will appear one at a time (using the delay from the previous task), where the Bresenham Line Algorithm generates the hypotenuse and the base of the triangle will always be the "central" horizontal line that bisects the y-axis (vertical).

As in Task 1, use Switches 7 downto 0 to specify the length of the base of the triangle from the center of the screen (horizontally) and Key 0 to "select" this new value for the base dimension of the triangles. As the base is remaining constant while the length and angle of the hypotenuse changes for each triangle, and the height of the triangle will also change (note: that half of the will be "negative" as they will be below the central line) to properly intersect with the defined "base" length. The length of the base will be fixed until the user selects a new value. On reset, the value for the base should be set to79 (remember that the "max" value for the base will be 80 because this is half of the x-axis dimension of 160).

To accomplish this task you should: 1) create an updated version of the pseudocode that reflects these algorithmic changes, and 2) use the pre-existing components from other tasks to minimize the number of new modules you need to create.

Hints: The screen has an even number of pixels both vertically and horizontally. As such, you can make the base line of the triangle two pixels wide if you want to create that horizontal line bisecting the vertical. The alternative is to use the upper row for the triangles with a positive vertical, and the bottom row for the triangles with a negative vertical. Either is an option.

What to demo:

Each section of this lab builds on the previous section. Because you have limited demo time with your TA, you don't need to demo each task individually. For the beginning of your demo slot, you should already have Task 2 loaded for evaluation. Once your TA has verified its functionality, you should immediately upload either the challenge task, Task 4 (if you have not completed the challenge task), or Task 3 (if you have not completed the challenge task or Task 4). Note that the programmer should be open and ready program the FPGA with the challenge task/Task 4/Task 3 ***before*** your demo begins. If you successfully demo the challenge task, we can infer that Task 4 and Task 3 also worked. If you did not successfully complete the challenge task, you can demo Task 4 (or Task 3 if you don't have Task 4 working). In short, demo what you have and explain what you think is wrong with task(s) you failed to get working. In all cases, for full marks, you must demo your working circuit on the FPGA board (simulation will get you part marks).

Submission:

Submit your ***lab3 VHDL files ONLY*** for each task on Canvas (not the project). Make sure you complete a header for all files you designed (similar to the template provided in Lab 1). This header should be included on all files you submit for review. You should submit your code at the end of your demo slot on your lab marking day and have the file you are submitting open for review by your TA on your desktop (Note: the website submission link "closes" at 11:59:59pm on Friday the 17th).