DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
UNIVERSITY OF BRITISH COLUMBIA
CPEN 311 – Digital Systems Design
2016/2017 Term 1

Lab 2: VGA Controller / Drawing Circles

Working week: Sept 26-30, 2016
Marking week: Oct 3-7, 2016

In this lab, you will get more experience creating datapaths and state machines. 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.

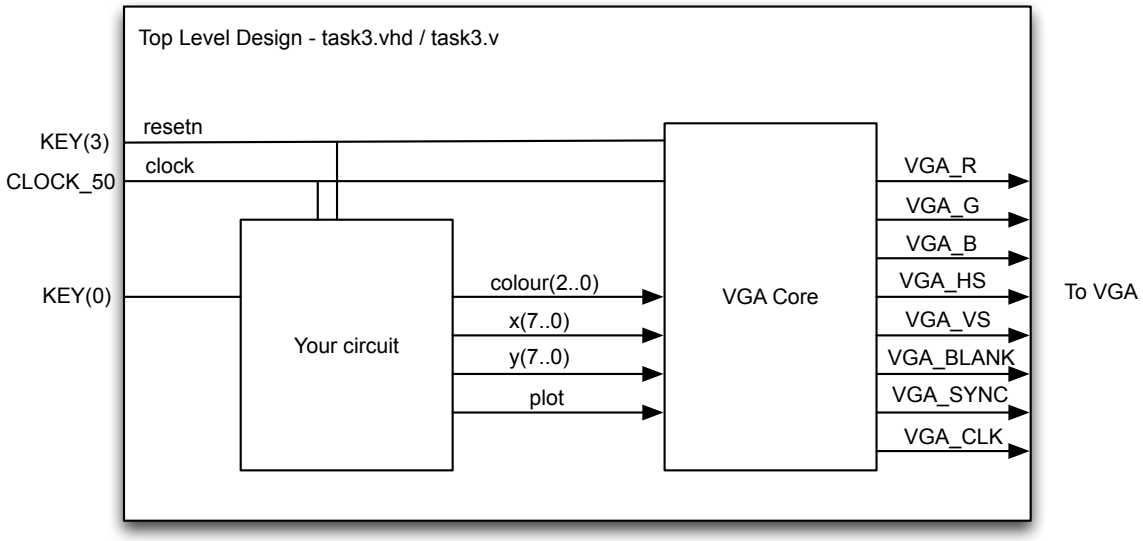


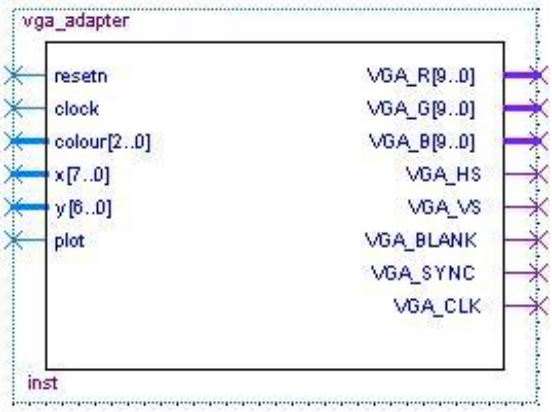
Figure 1: Overall block diagram

Task 1: Understand the VGA Adapter Core:

The VGA Adapter was created at the University of Toronto for a course similar to CPEN 311. 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 Altera board, the VGA adapter has been setup to display a grid of 160x120 pixels, with the interface shown in Figure 2:

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 Altera 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.

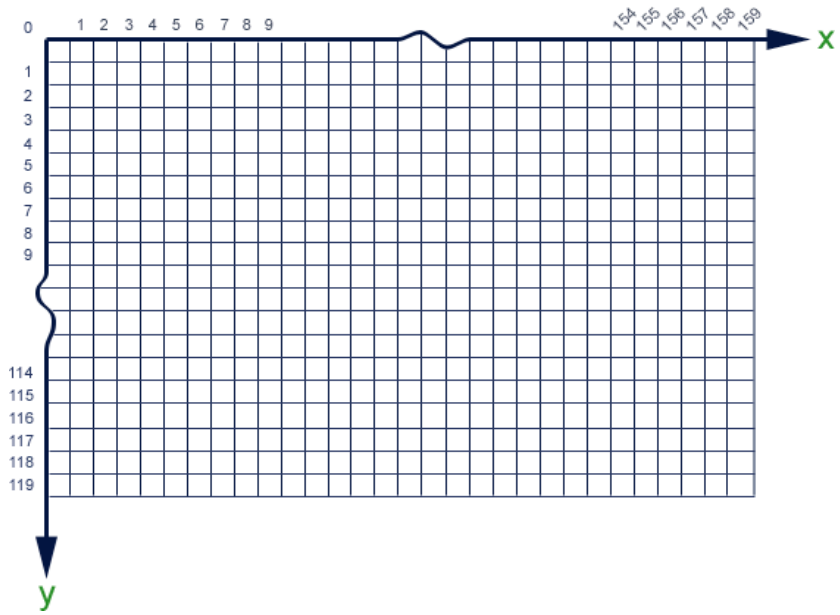


Figure 3: VGA adapter's display grid

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 colour 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 fix 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.

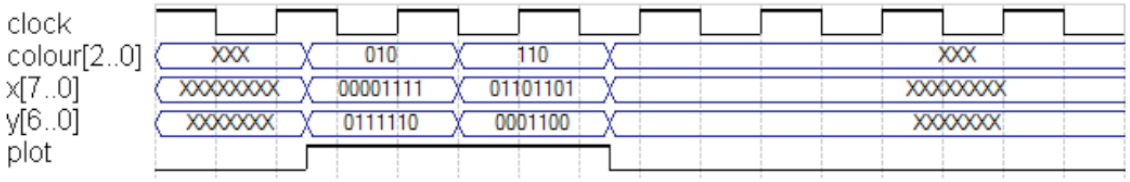


Figure 4: Timing Diagram

The source code for the VGA adapter is on the CPEN 311 Connect site (Lab 2->Task1). This core is written in Verilog, not VHDL. That is ok, even if you are writing your labs using VHDL, when making a VHDL structural description, you can include Verilog modules just as you can VHDL modules (this "mixed language" design approach is common in industry).

To help you understand the VGA core, we have created an `vga_demo.vhd` file (no Verilog version, but it should be straightforward what this file does). 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 this file is *only* used for you to understand the core; you will *not* use it when constructing your circuit in Tasks 2 and 3.

Task 2: Fill the Screen: (3 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)
        }
}
```

You are to create a circuit that implements the above algorithm. A skeleton file **task2.vhd** is available on the Connect site. Your design should have an asynchronous reset which will be driven by KEY(3). You don't need to use KEY(0) or any of the switches in this task. Note that your circuit will be clocked by CLOCK_50 .

Note that in class we describe two ways to describe a datapath circuit:

1. Explicit State Machine / Datapath design method: In this method, you manually design the datapath and associated state machine, and implement each in VHDL or Verilog. This involves identifying all the control and status signals you require, and determining exactly what operation is supposed to happen each cycle (in order to create the controller). We looked at simple examples of this in Slide Set 4.
2. Implicit Datapath design method: In this method, you create a combined datapath/controller description. Your description here would look like a state machine, but with operations specified for each state rather than just the values of control signals. By the time you are doing this lab, we haven't talked about this method in class yet.

My suggestion for Lab 2 is to use the first approach. Create a separate datapath and controller for the algorithm. Start by identifying the storage units, the computation units, and connect them as in the examples in class. In later labs, you can try the combined approach if you like. [If you try the combined approach here and get it to work, you will receive full marks, but you will likely find it harder to write correct synthesizable code this way].

Test your design on your board. You will need a VGA cable and a VGA-capable display. In the MCLD358/348 lab, a VGA cable and LCD display are provided. Download the files containing: `vga_adapter.v`, `vga_controller.v`, `vga_address_translator.v.` `vga_pll.v` from the Connect site (they are the same ones you just downloaded for Task 1). A stub file for Task 2 is also available to get you started (both VHDL and Verilog). Add these files to a new Quartus II project, import your pin assignments, then compile and program the design onto your board. (don't forget your pin assignments!)

Use the VGA cable to connect your board to the VGA display. Most new LCD displays have multiple inputs, including DVI (digital) and VGA (analog). The LCD displays in MCLD358/348 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 Altera 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 Circle Algorithm: (4 marks)

The Bresenham Circle algorithm is a hardware (or software!) friendly algorithm to draw circles on the screen. The basic algorithm is as follows (modified from Wikipedia): Note that this is pseudo-code of the behavior of the circuit, it is not meant to be illustrative of the synthesizable VHDL you will write.

```
draw_circle( center_x,  center_y,  radius):
        int offset_x, offset_y, crit;   -- local variables

        offset_y = 0
        offset_x = radius
        crit = 1 - radius

        while (offset_y <= offset_x) {
                setPixel( center_x + offset_x, center_y + offset_y)   -- octant 1
                setPixel( center_x + offset_y, center_y + offset_x)   -- octant 2
                setPixel( center_x - offset_x, center_y + offset_y)   -- octant 4
                setPixel( center_x - offset_y, center_y + offset_x)   -- octant 3
                setPixel( center_x - offset_x, center_y - offset_y)   -- octant 5
                setPixel( center_x - offset_y, center_y - offset_x)   -- octant 6
                setPixel( center_x + offset_x, center_y - offset_y)   -- octant 7
                setPixel( center_x + offset_y, center_y - offset_x)   -- octant 8

                offset_y := offset_y + 1
                if ( crit <= 0 ) {
                        crit := crit + 2 * offset_y + 1
                } else {
                        offset_x := offset_x - 1
                        crit := crit + 2 * (offset_y – offset_x) + 1
                }
        }
```

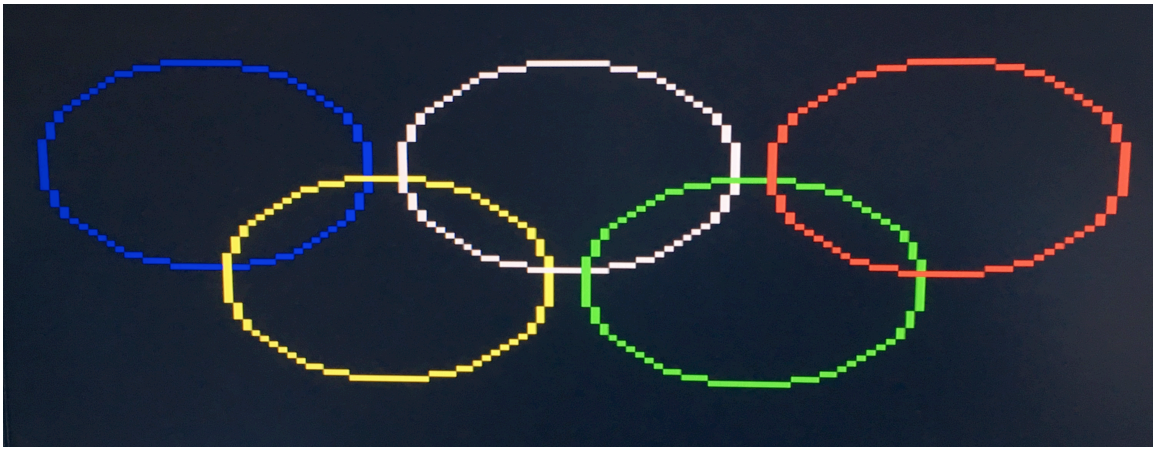
In this task, you are to implement a circuit that (a) clears the screen and (b) draws a blue circle centered at (80, 60) with radius 40. With enough thinking, you should be able to come up with a datapath to implement the Bresenham Circle algorithm. Start with your datapath from Task 2, and refine it. The slides we have worked through in class should help you here.

In designing the datapath for this algorithm, *remember that you can only update one pixel per clock cycle*. This means that each iteration through the loop would take *at least* 8 clock cycles (since you are updating 8 pixels per iteration).
The code 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.

It is important to note that you are using CLOCK_50, the 50Mhz clock, to clock your circuit. This is different than Lab 1 where you used a pushbutton switch for your clock.

Task 4: Making it a bit more interesting: (2 marks)

In order to commemorate the fact that 2016 is an Olympic year, in this task, you are modify your circuit to draw the Olympic Rings. The output will look as shown below (note that each ring is a different colour). Likely in this task, you will use your code from Task 3, implements some sort of outer loop that steps through the five rings.



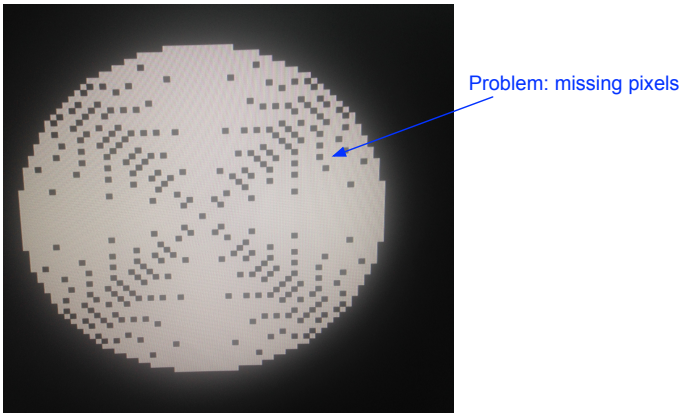
Challenge Task: (1 mark)

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+). Solutions will not be distributed.

This one is actually harder than it looks.

For the challenge task, go back to your code from Task 3. You are to modify it to draw a circle and *fill it in* with solid colour.

This sounds like it should be easy. You can imagine drawing concentric circles, all the same colour, decreasing the radius by 1 each time. The problem with this approach is that it will "miss" some pixels as shown in the following diagram.



You'll need another approach. There are several ways to do it. Be creative.