National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science

EE-421 Digital Systems Design

Assignment-1: VGA Controller and Drawing Circles

Learning Objectives:

1. Independently design a finite-state machine and datapath system in Verilog.
2. Build complex arithmetic functions in Verilog.
3. Gain experience of using third-party IP in the design.
4. Get familiar with using mostly used VGA interface.

Overview:

In this assignment, you will be designing your own hardware to draw circles on the VGA screen. At first glance, you might think that drawing circles is really boring or you might think that it's not very useful for you to learn about. Turns out though, it is one of the most fundamental operations in graphics technology. And because it's so fundamental, circle-drawing (along with line drawing and other basic geometric shapes) is often implemented with dedicated hardware. Anytime you look at a screen with any sort of rendered graphics, be it your desktop PC, smart phone, smart TV, Xbox, or even futuristic virtual reality technologies like the Oculus Rift or Microsoft HoloLens, you will see hardware implemented shape-drawing in action.

The top-level diagram of your lab is shown below. The VGA Adapter Core is a component that is given to you! This is common practice in industrial design – taking predesigned components that are either purchased or written by another group and incorporating them into your own design.

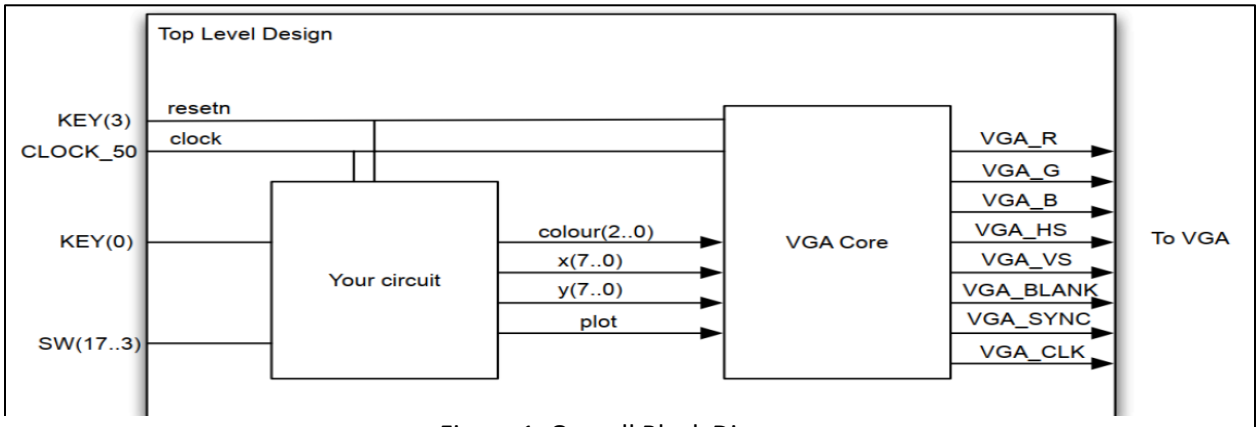

Figure 1: Overall Block Diagram

Task 1: Understand the VGA Adapter Core

The VGA Adapter core was created at the University of Toronto for a course similar to ours. The following describes enough for you to use the core; more details can be found on University of Toronto's web page: https://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 core has been setup to display a grid of 160x120 pixels, with the interface shown in Figure 2:

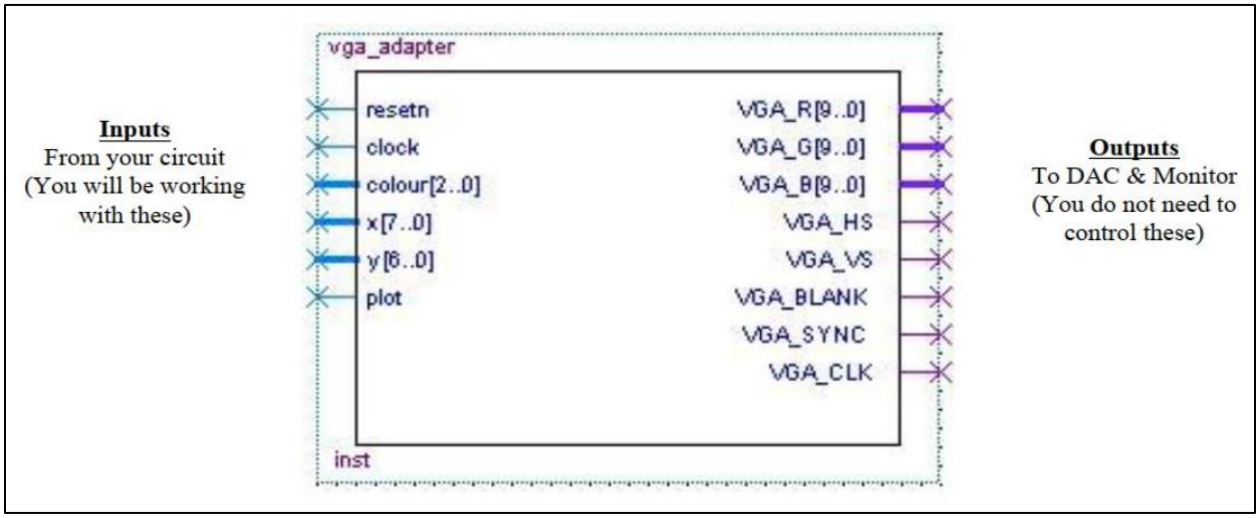

Figure 2: VGA Adapter core as a black box

Inputs:

Resetn	Active low reset signal (does not reset the screen buffer). Digital circuits with state elements should always contain a reset.
Clock	Clock signal. The VGA Adapter core must be fed with a 50MHz clock to function correctly.
Colour (2 down to 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 down to 0)	X coordinate of pixel to be drawn (8 bits) – supported values $0 \leq x < 160$.
y (6 down to 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:

Note: You shouldn't have to worry about the outputs except that they need to be properly connected to your top-level ports.

VGA_CLK	VGA clock signal.
VGA_R (9 down to 0) **VGA_G (9 down to 0)** **VGA_B (9 down to 0)**	Red, Green, Blue components of display (10 bits). These signals are connected to the Digital-to-Analog Converter (DAC) on the DE2 board (or whichever you have) 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 Adapter core directly to appropriate output pins of the FPGA (please see the datasheet for your particular board).

You can picture the VGA screen as a grid of pixels shown in Figure 3. The X/Y position (0,0) is located on the top-left corner and (159,119) pixel located at the bottom-right corner. The role of the VGA Adapter core is to continuously draw the same thing on the screen at the monitor's refresh rate, e.g. 60 Hz. To do this, it has an internal memory that stores the colour of each pixel. Your circuit will write pixel colours to the VGA Adapter core.

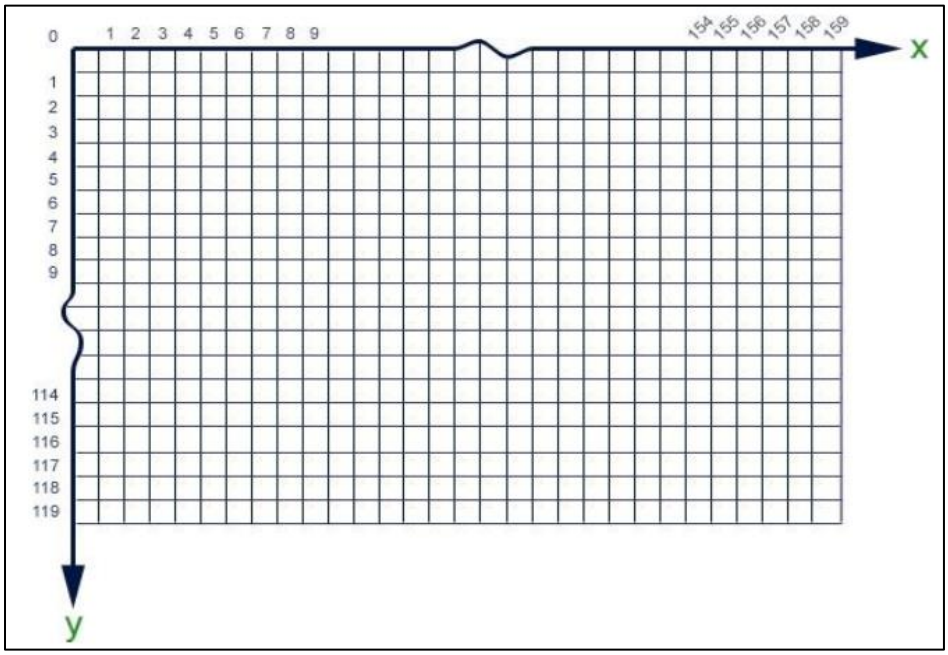

Figure 3: VGA Adapter core's display grid

To set the colour of a pixel, you first drive the VGA Adapter Core's X, Y and COLOUR inputs with the pixels' x coordinate, y coordinate, and desired color value, respectively. You then raise the PLOT input to high. You must keep these values driven until the next rising clock edge. At the next rising clock edge, the pixel colour is accepted by the VGA Adapter core's memory. Then, starting on the next screen redraw, the pixel will take on the new colour. In the following timing diagram (from the UofT Website), two pixels are changed: one at (15, 62) and the other at (109,12). As you can see, the first pixel drawn is green (rgb = 010) 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 changed on each cycle*. **If you want to change the colour of m pixels, you need m clock cycles.**

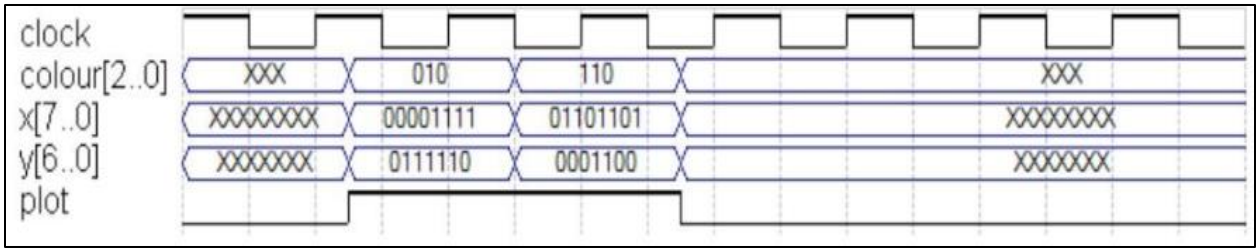

Figure 4: Timing Diagram

Download the VGA core from UofT website. The Verilog files describing the VGA Adapter core can be included into Altera Quartus II project. *You shouldn't be modifying the VGA Adapter Core code at all!!!*

In order to understand the VGA Adapter core, create a top level vga_demo.v file. This file does nothing but connect the VGA Adapter core I/O to switches on the board so you can experiment. It will let you understand how the inputs of the core work.

This task is not worth any marks, but you should do it to ensure that everything else is working (e.g. your VGA Cable is good) before starting the main task below.

Make sure you include the Adaptor Core files in your project: vga_adapter.v, vga_controller.v, vga_address_translator.v and vga_pll.v. And remember to set up your pin assignments for your board.

Task 2: Fill the Screen

You will create a new component that interfaces with the VGA Adaptor Core. It will contain a simple FSM to fill the screen with colours. This is done by writing to one pixel at a time in the VGA Adapter core. Each row will be set to a different colour (repeating every 8 rows). Since you can only set one pixel at a time, you will need a FSM that does something like this:

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

Create an FSM that implements the above algorithm. 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.

Test your design on the DE board. You need your DE board with a USB cable, a VGA cable, and a VGA-capable display. Most new LCD displays have multiple inputs, including DVI (digital) and VGA (analog). Note: the VGA connection on your laptop is an **OUTPUT**, so **do not connect your laptop's VGA port to your DE board.**

Hint: Modelsim will be very useful for debugging your component's outputs.

Task 3: Bresenham Circle Algorithm

The Bresenham Circle algorithm is a hardware (or software!) friendly algorithm to draw circles with arbitrary center and radius on the screen. The basic algorithm is as follows:

```
function circle_bresenham ( xc , yc , r )
        x := 0
        y := r
        d := 3-2*r
        loop
                if x > y exit loop

                setPixel (xc + x, yc + y)
                setPixel (xc - x, yc + y)
                setPixel (xc + x, yc - y)
                setPixel (xc - x, yc - y)
                setPixel (xc + y, yc + x)
                setPixel (xc - y, yc + x)
                setPixel (xc + y, yc - x)
                setPixel (xc - y, yc - x)

                x := x + 1

                if d < 0 then
                        d := d + (4 * x) + 6
                else
                        d := d + 4 * (x – y) + 10
                        y := y – 1
                end if-else
        end loop
end function
```

The algorithm is quite efficient: it contains no multiplication or division (multiplication by multiples of 2 can be implemented by a shift-register that shifts left). Because of its simplicity and efficiency, the Bresenham Circle Algorithm can be found in many software graphics libraries, and in graphics chips.

In this task, you will implement a circuit that behaves as follows:

1. The switch KEY[3] is an asynchronous reset. When the machine is reset, it will start clearing the screen to black. Hint: Task2 is basically clearing the screen if you set all pixels to black. Clearing the screen will take at least 160*120 cycles.
2. Once the screen is cleared, your circuit will idle. At this point, the user can set switches 7 down to 3 (SW[7:3]), which indicates the radius, and switches 2 down to 0 (SW[2:0]), which indicates one of 8 possible colours used to draw the line. **IMPORTANT: Restrict user entered radius to be within 0 to 59.** If you don't, you will see some unexpected behavior (strange patterns being drawn instead of a circle). For example, if the user set the switches to indicate a value of 70 for radius, just clip it to 59.
3. When the user presses KEY[0], the circuit will draw a circle. Centre of the circle should be the centre of the screen (location 80,60) and radius as specified by the user. Of course, this will take multiple cycles; the number of cycles depends on the radius of the circle.
4. Once the circle is done, the circuit will go back to step 3, allowing the user to choose another radius and color. Do not clear the screen between iterations. At any time, the user can press KEY[3], the asynchronous reset, to go back to the start and clear the screen. The reset signal on the VGA Core does not clear the screen. That's why you need to do it manually in step 1. But this also means that you don't have to do anything special to retain previously drawn circles on the screen.

Note that you are using CLOCK_50, the 50MHz clock, to clock your circuit. You must clearly distinguish the datapath from the FSM in your Verilog code (i.e. don't write one giant always block to do everything). The reason that this is a requirement is that we want you to practice manual partitioning of the datapath and FSM for now.

Suggestions and Hints

Here are some hints and things to think about:

1. Think about how you will partition your FSM and Datapath before you start writing any Verilog.
2. You may consider a simplified drawing algorithm first while you work out your FSM. For example, when the user presses the "draw" button (KEY[0]), you may consider having your design draw some hardcoded pattern onto the screen. This will let you focus on the FSM and screen clearing. Once those parts seem to work, you can replace your hardcoded pattern with the circle drawing algorithm.
3. Be mindful of your variable/signal vector sizes. Are they sized large enough? For example,

in the equation “e2 := 2*error”, “e2” must be 1-bit larger than “error” to not overflow. Not keeping track of signs (read up on the “signed” keyword in verilog) and sign bits can also cause overflow. Overflow can be a pain to debug unless you know to look for it.

Challenge Circuit: (10 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 10 marks. If you don’t demo the challenge task, the maximum score you can get on this assignment is 90/100 (which is still an A+).

This challenge task is actually fairly easy:

1. In the original circuit, you always use the centre of the screen (80,60) as the center of the circle and take the radius and color from the user. Modify your circuit such that, each center and color is chosen pseudo randomly (using a Linear Feedback Shift Register).
2. The radius should be the largest radius possible for the randomly generated center. For example if (100,79) is the center, the largest possible radius is 40.
3. Draw the circles successively without waiting for KEY[0] in between. Note that you will have to add some delay, or the screen will fill up too quickly for you to see.

Technical Grading:

10 marks (initializing screen black after reset)

10 marks (structure of FSM and Datapath; clear separation of code)

30 marks for Task-2

40 marks for Task-3

10 marks for challenge circuit

☺ Happy Learning, Good Luck!!!