SFU

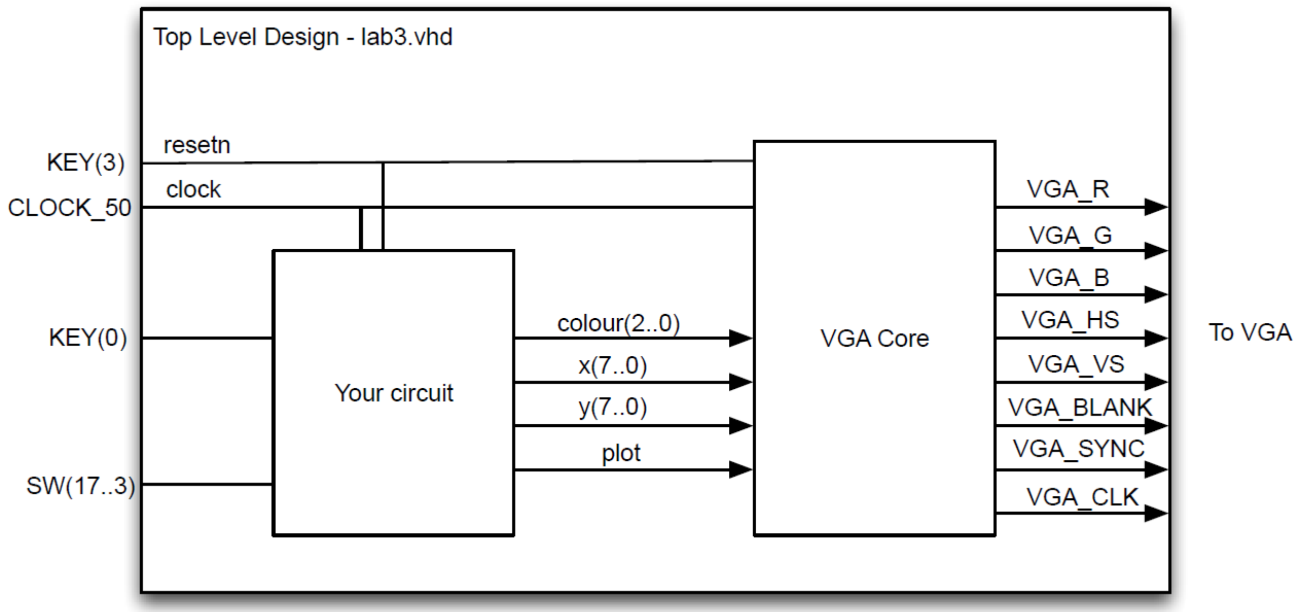
ENSC350: DIGITAL SYSTEMS DESIGN

SPRING 2024 - 1241

LAB 3: DATAPATHS EXTENDED

DESCRIPTION

In this lab, you will get more experience creating Datapath and State Machines. You will also learn how to use an embedded core provided to 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.



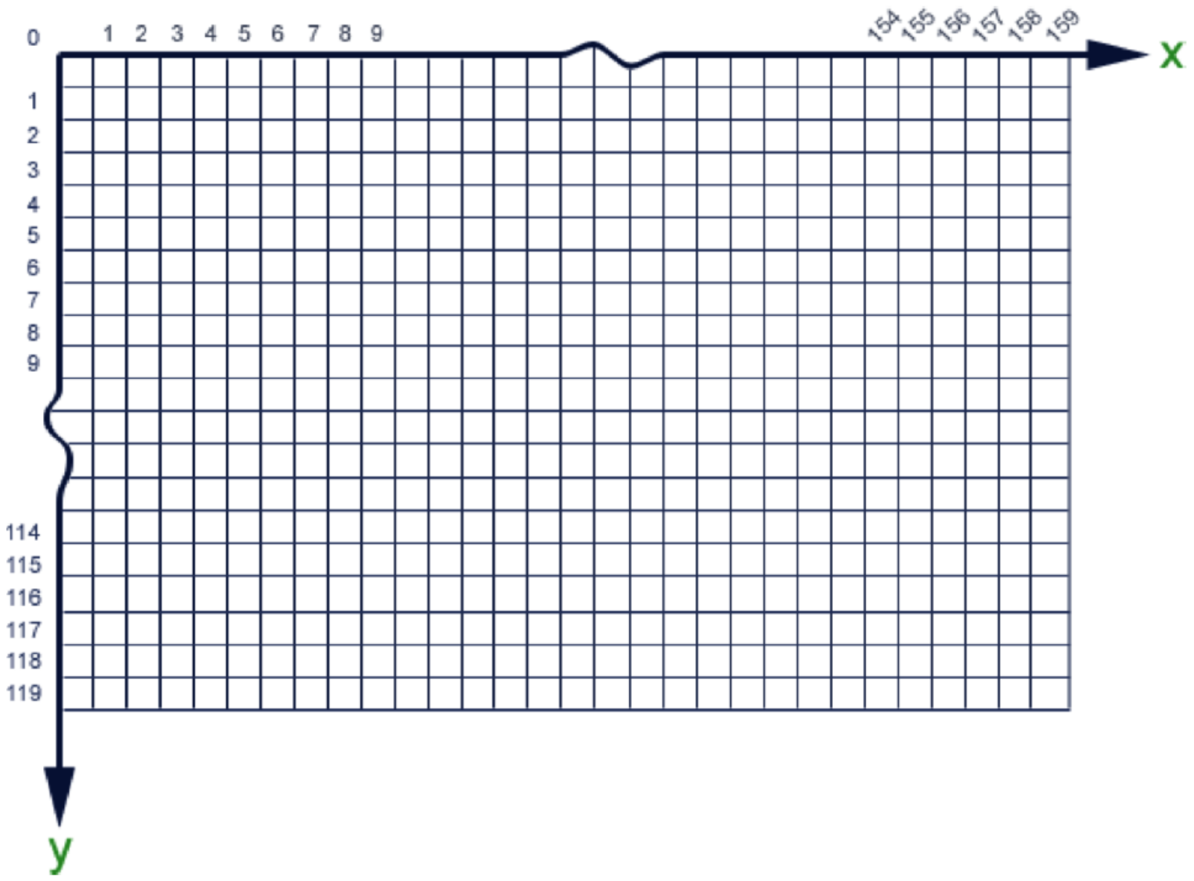
The VGA Core is the part given to you, so all the excitement will be in the block labeled “Your Circuit”. This handout first explains how to use the VGA Core, and then specifies what your circuit should do. Laboratory computer screens and cables could be used for testing, just treat all equipment with proper care and leave everything as you received. If any additional elements or materials are required, please let us know.

Like all labs in this course, this lab will span two weeks. The first week is intended to be a work-week, where you spend the lab working on the tasks. A TA will be available to help you if you run into problems. The second week is primarily for marking (you should not expect help from the TA during the second week). It is expected that you should do some work on your own (at home or in the lab outside of lab hours) as well. You most certainly cannot finish the lab if you don’t start until the marking week.

- Working week: February 26 – March 1, 2024
- Marking week: March 5 & 7, 2024

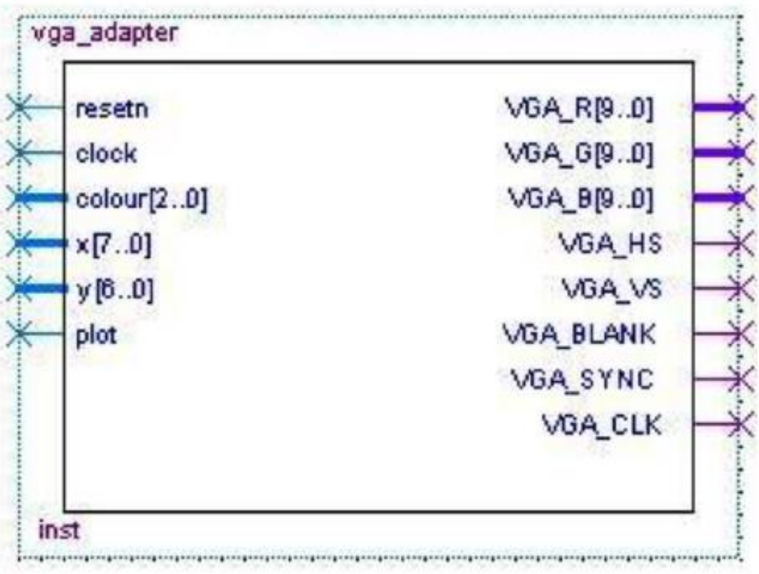
UNDERSTANDING THE VGA ADAPTER CORE

The VGA Adapter was created at the University of Toronto. All details can be found on University of Toronto’s web page: http://www.eecg.utoronto.ca/~jayar/ece241_07F/vga. Please read their documentation and understand the module. Some of the following figures have been taken from that website. In order to save on the limited memory on DE2 board, the VGA adapter has been setup to display a grid of 160x120 pixels, as indicated in the following figure:



The VGA Adapter module is defined as follows:

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



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

A description of each input/output is:

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 $0 \le x < 160$.
- **Y (6 downto 0)**: Y coordinate of pixel to be drawn (7 bits)–supported values $0 \le x < 120$.
- **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)**: Red component of display (10 bits).
- **VGA_G(9 downto 0)**: Green component of display (10 bits).
- **VGA_B(9 downto 0)**: Blue components of display (10 bits).
  These RGB signals are connected to the Digital-to-Analog Converter (DAC) on the DE2 board before transmitting to the monitor.

Marks

No implementations done in this section; therefore, no marks. However, the information provided must be understood to proceed to the next sections for a correct implementation of the system.

THE DIRECT OPERATION

TASK 1: you will work over the skeleton file provided, lab3.vhd. Design the "Your Circuit" block implement a functionality were coordinates and color are manually input by the switches on your board.

The entity part of the block showing the inputs and outputs is as follows (a skeleton file is provided in Canvas as part of this lab folder):

```
entity lab3 is
  port(CLOCK_50             : in  std_logic;
       KEY                  : in  std_logic_vector(3 downto 0);
       SW                   : in  std_logic_vector(17 downto 0);
       VGA_R, VGA_G, VGA_B  : out std_logic_vector(9 downto 0);  -- The outs go to VGA controller
       VGA_HS               : out std_logic;
       VGA_VS               : out std_logic;
       VGA_BLANK            : out std_logic;
       VGA_SYNC             : out std_logic;
       VGA_CLK              : out std_logic)
end lab3;
```

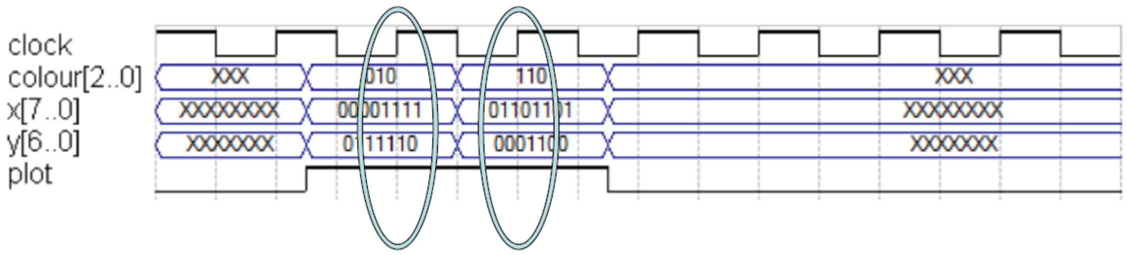
Notice that there are two inputs: the KEY variable assigned to the push buttons, and the SW variable assigned to the dip-switches. The next points describe their functionality.

- **Key (3)** will be used to reset the whole system. It will affect both blocks "Your Circuit" and "VGS Core", as indicated in the schematic figure presented before. You must determine if this button could "blank" the screen.
- **SW (2 - 0)** will independently indicate which color will be considered for pixel color combination. A total of 8 possible color combinations could be generated.
- **SW (9 - 3)** will provide the y coordinate of the pixel.
- **SW (17 - 10)** will provide the x coordinate of the pixel.

There are eight outputs of the design, all of them correspond to the pins of the VGA terminal of the board.

The design at this stage will have a straight forward operation, the coordinates and color set at the input, will activate the pixel in the VGA monitor. You can use any of the additional push buttons, like **KEY (0),** to be used as an "enter" command to display the updated pixel activation (Consider debouncing if needed).

TASK 2: you will test your implementation using Modelsim. You will have to check that the signals provided to your "VGA Core" block are built and accepted, and outputs are changed. You could use a signal display, like:



The previous figure shows the activation of 2 pixels. Also, you are allowed to use a text-based debugging, or a combination or both.

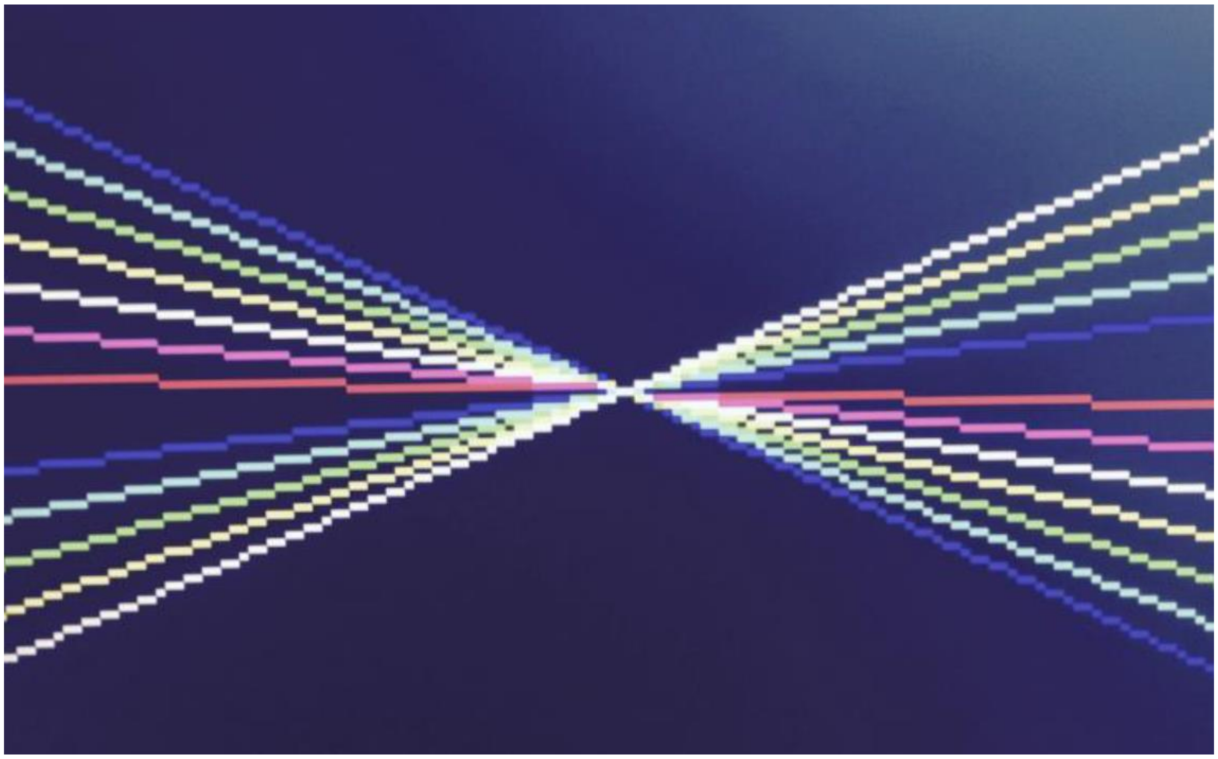
Marks

This section will have a value of 6 marks.

AUTOMATIC OPERATION

TASK 3: You are required to adapt your system to produce a line linking the center of the display grid to a specified coordinate. Users can designate this coordinate point through dip switches and also choose the line's color. All existing input ports can be retained for this modification from the previous task. It is recommended to use an additional switch, such as **KEY (1)**, to serve as the "enter" command for this operational mode.

The expected result after some lines generated could be seen in the following figure:

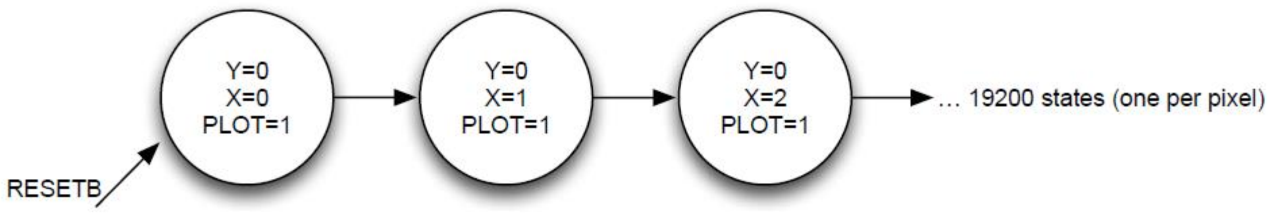


Remember that to calculate the points to generate the lines, you can use the basic equation of a line, considering its slope only since all lines crosses the origin. Lines must start and stop at the indicated coordinates as possible; you do not need to extend the line to the border of the display as shown in the previous example figure.

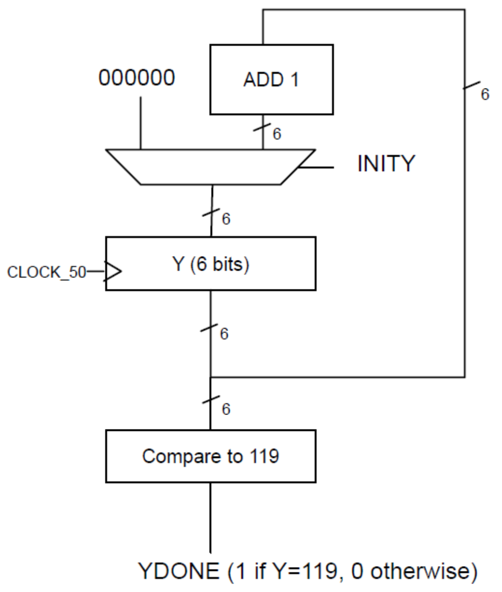
You must compensate for the origin coordinates displacement to the center of the display. Similarly, for the input coordinate point.

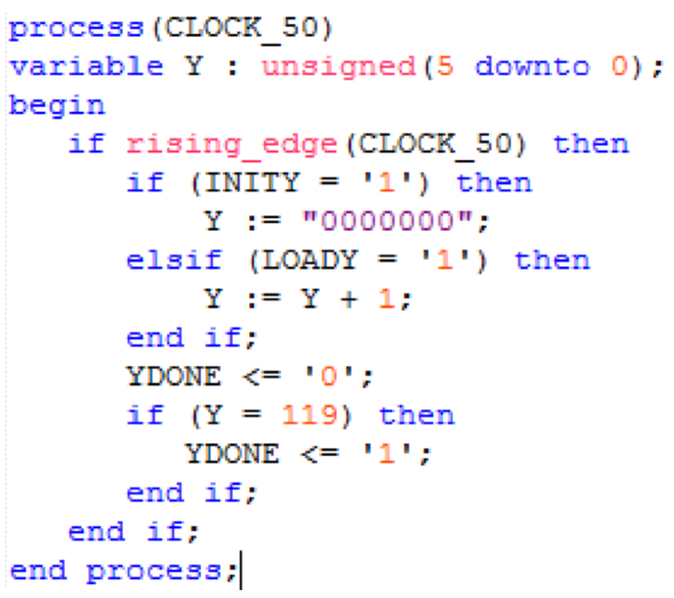
Hints:

If you need to generate a sequential activation of pixels, you can use the traditional method of activating one by one by using a state machine:



However, a more effective strategy would involve employing increments derived from arithmetic calculations. Look at the following implementation to represent a vertical line:





```
process(CLOCK_50)
variable Y : unsigned(5 downto 0);
begin
   if rising_edge(CLOCK_50) then
      if (INITY = '1') then
          Y := "0000000";
      elsif (LOADY = '1') then
          Y := Y + 1;
      end if;
      YDONE <= '0';
      if (Y = 119) then
         YDONE <= '1';
      end if;
   end if;
end process;
```

Y variable will be the signal to be connected to the "VGA Core" input at the output of its block.

TASK 4: Just like the earlier task, you will specify the coordinates for a point. These coordinates will serve as the basis for your adapted design to create a triangle. The triangle will originate from the center of the display (the origin), extending to the specified point, and then connecting to the x-axis.

- It is allowed to have a pre-existing "white" line representing the x-axis, which remains unchanged for each newly generated triangle
- We recommend you to use **Key (2)** as the "enter" command to draw the triangle.

Marks

This section will have a value of 8 marks.

CHALLENGE TASK

Arrange your code in a way that enables the simultaneous use of any operation mode during a single runtime. It is crucial to appropriately utilize the KEY inputs. The system should permit the user to activate a pixel, draw a line to a pixel, and generate a triangle using a pixel reference and the x-axis in any sequence, without requiring a program reload or a reset.

Marks

This section will have a value of 1 mark.

WHAT TO DEMO

Each section of this lab builds a component of the whole system. Because you have limited demo time with your TA, you don't need to demo each version individually. However, Simulations will have their own marking points assigned, so plan carefully how you will manage your time. Also, the block architecture must be shown, and explained.

To show your system is fully operational, bring ready some coordinates that you would like to try with the respective expected plot results (Try not to overlap your used pixels). The TA's will also have some examples ready for you to input into your system.

In all cases, for full marks, you must demo your working circuit on the FPGA board (simulation will get up to 2 marks).

Remember to always:

- Have support material to explain the operation of your code (State Machine diagrams, Logic blocks diagrams, commented code, previous ModelSim result graphs, etc.).
- Get a backed up fully operational code copy in case any corrupted file or unexpected situation happens to your original files.
- Be certain of how your device operates and how to explain its operation so that full marks could be achieved.
- Use additional hardware resources to make your implementation easier to explain (LEDs to show changes or events, input accepted, others), mostly for FMS.

If you got the entire design working (with or without the challenge task), demonstrate that to the TA and submit all VHDL files.

Your code must be submitted no later than 24 hours after your demo. You must ensure you submit exactly the code that you demoed; submitting code other than the code you demo is considered academic misconduct.

You should submit all of the .vhd files that you have written. Please do not zip up your entire working directory and submit it; if everyone does this, it takes too much space. Only submit the .vhd files. Do not submit temporary files, or .vhd files that you did not modify.

Finally, a reminder that **all team members must be present** to demo the operation of your design (***<u>if one person is not present, that person will not receive marks for this lab</u>***).