EE20021 Coursework – Activity 4
VGA Controller

Uriel Martinez-Hernandez

Introduction

In this activity you will develop and implement a VGA controller using the file *vgaController.sv*, that has been provided in Moodle, to display objects in the screen of a computer. Download the zip file Files for *Files for CW4-VgaController.zip* for this tutorial from Moodle, where you will find the following:

- *vgaController.sv:* template file where you will add your code for the implementation of the VGA controller. This file already includes the required input and output signals for this module.
- *vgaTb.sv:* testbench that you can use to test your vgaController in Questa. You don't need to modify this file.
- *vgaTopFPGA.sv:* top file to connect your module to the monitor using the FPGA and a VGA cable. If your program is correct then it should display a pattern of colours in the monitor (see example below).

How VGA signals work

The VGA system on the DE1-SoC has three synchronisation signals, and a blanking signal. These control signals are used to position the visible red, green and blue pixel values on the screen. Pixel values are output starting at the top left of the screen going all the way along each row before moving down to the beginning of the next row. At the end of each row there are some blank pixels and at the end of the frame there are some blank lines. Roughly in the middle of the blanking region is a synchronisation pulse. The blank time before the sync pulse is called the *front porch* and the blank time after the sync pulse is called the *back porch*.

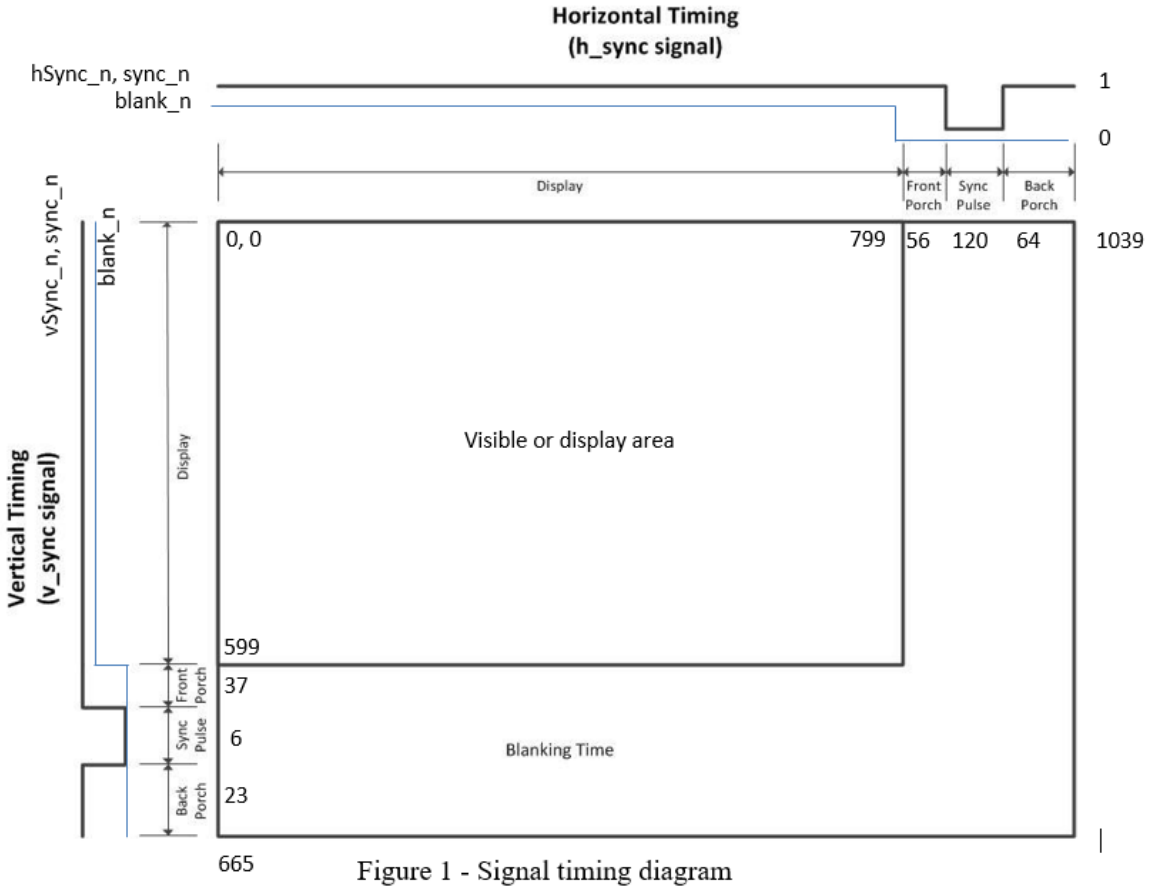

Figure 1 - Signal timing diagram

Activity 1 – VGA controller and simulation

In this activity you will develop a VGA controller in Quartus Prime required to display objects on the screen of a computer. Once your VGA controller is ready, then simulate it using the *vgaTb.sv* provided for this activity and check that all the signals change according to the specification in Figure 1.

The module should have the following I/O:

```
module VgaController
(
      input Clock,
      input Reset,
      output logic  blank_n,
      output logic  sync_n,
      output logic  hSync_n,
      output logic  vSync_n,
      output logic [10:0] nextX,
      output logic [ 9:0] nextY
);

      // use this signal as counter for the horizontal axis
      logic [10:0] hCount;

      // use this signal as counter for the vertical axis
      logic [ 9:0] vCount;


      // add here your code for the VGA controller

endmodule
```

The controller should implement **sync** and **blank** signals for an 800x600 VGA display and output the X and Y pixel position to be displayed in the next clock cycle. If the **blank_n** signal is active low the **nextX** and **nextY** signals will be set to 0. The clock input is 50MHz and the positive edge of the clock should be used for synchronisation. The **sync_n** and **blank_n** signals should be active whenever either the horizontal or vertical signals are active.

Timings for the display signals are (start the count in 0):

		Display Section			
Direction	**Units**	Visible	Front Porch	Sync Pulse	Back Porch
Horizontal	Clock Cycles	800	856	976	1040
Vertical	Lines	600	637	643	666

Activity 2 – VGA controller and FPGA

Use the **vgaTopFPGA** module as the top module to implement your VGA controller in the FPGA. Once your VGA controller has been programmed in the FPGA, connect the board to the PC monitor with a VGA cable. If your module is working correctly, then you will see in the monitor the pattern of colours shown in Figure 2. The Red, Green, Blue (RGB) colours are defined by the 8 bit signals VGA_R, VGA_G, VGA_B, which allow to define a colour using values from 0 to 255.

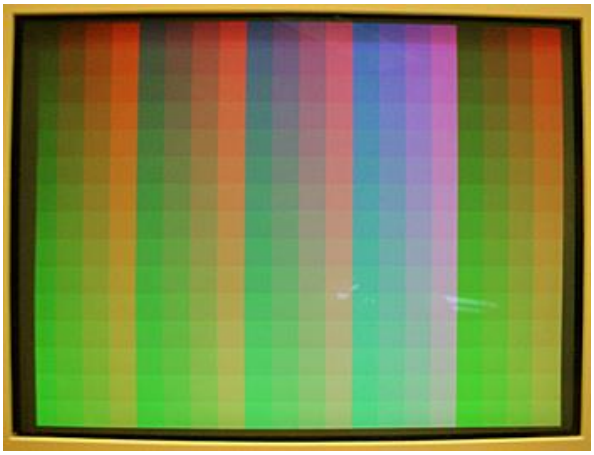

Figure 2 – Output from the VGA controller

Activity 3 – Experimenting with VGA_R, G, B

You can assign other values to VGA_R, VGA_G and VGA_B signals in the **vgaTopFPGA.sv** module and see the changes in the monitor of the PC.

If your VGA controller does not work, then use the testbench **vgaTb.sv** to check that all signals are working correctly.

IMPORTANT

You have the option to work in a group of ***2 students maximum***. Add your name (and the name of your teammate if you worked in a group) at the top of all the SystemVerilog files developed in this coursework activity.

Add clear comments and detailed description of all the components of your module, and do not forget to use meaningful names for the variables, ports, etcetera. Also remember that it is important to indent your code to make it easier to read by others. All these aspects will be considered for the mark of this module.