ECE 272 Lab 6
Fall 2018

Video Graphics Array (VGA)

Phi Luu

November 30th, 2018
Grading TA: Edgar Perez
Lab Partner: Benjamin Geyer

1 Introduction

Video graphics array (*VGA*) is a graphics standard for video display controller first introduced by IBM in 1987. VGA can be used to display a wide range of resolutions and refresh rates. VGA draws on the monitor from left to right and top to bottom. Each pixel is controlled by three analog pins for red, green, and blue (RGB) channels. Although VGA can control a wide range of resolutions and refresh rates, the 640x480 60Hz configuration is commonly used.

For the DE10-Lite, each of the RGB channels is a 4-bit port. The red, green, and blue channels connect to pins 1, 2, and 3 of the VGA connector, respectively. The horizontal sync (hsync) and vertical sync (vsync) signals of the FPGA connect to pins 13 and 14 of the VGA connector, respectively. Hsync and vsync are active-low. When hsync goes to 0, the next line starts to draw; when vsync goes to 0, the drawer goes back to the top of the screen again, starting a new frame. Figure 1.1.

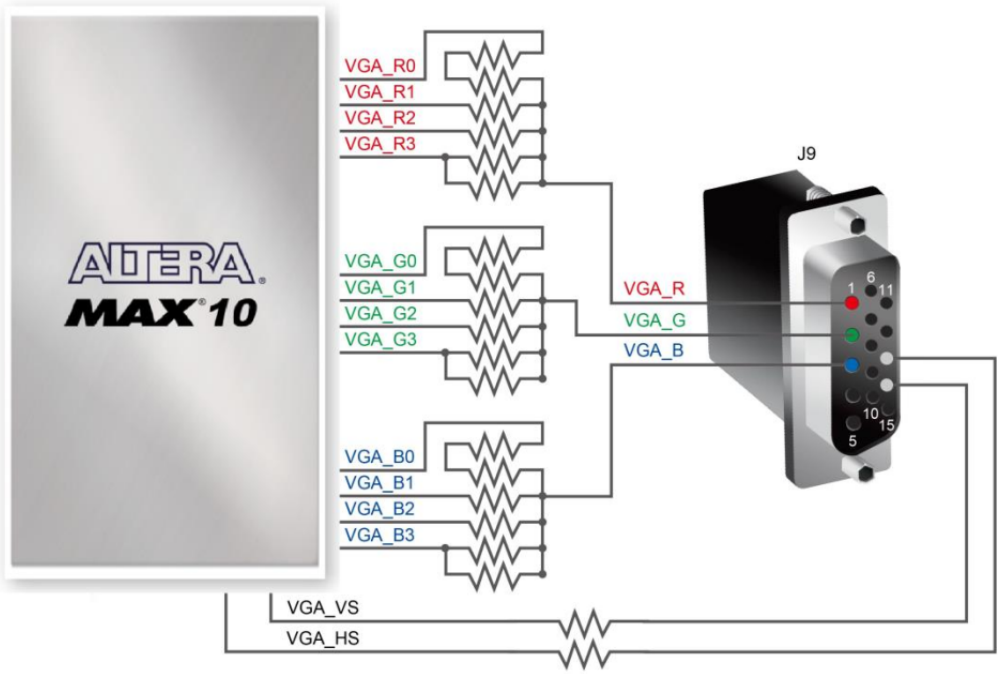

Figure 1.1: Connections between the VGA and the MAX 10 FPGA

A more detailed pinout of a VGA connector is illustrated by Figure 1.2.

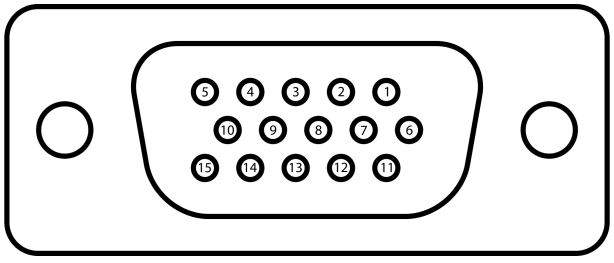

Figure 1.2: All 15 pins of a VGA connector. See Table 1.

Pin	Signal	Description
1	RED	Red video
2	GREEN	Green video
3	BLUE	Blue video
4	ID2/RES	formerly Monitor ID bit 2 Reserved since E-DDC
5	GND	Ground (H-Sync)
6	RED_RTN	Red return
7	GREEN_RTN	Green return
8	BLUE_RTN	Blue return
9	KEY/PWR	formerly key now +5V DC
10	GND	Ground (V-Sync, DDC)
11	ID0/RES	formerly Monitor ID bit 0 reserved since E-DDC
12	ID1/SDA	formerly Monitor ID bit 1 PC data since DDC2
13	HSync	Horizontal sync
14	VSync	Vertical sync
15	ID3/SCL	formerly Monitor ID bit 3 I2C clock since DDC2

Table 1: VGA pinout

2 Design

The timing sequence of an VGA controller is as follows: the pixels are displayed from left to right and top to bottom. When all the pixels of the active resolution are displayed, there is a blank period called the *front porch* which signaling the start of a *sync pulse*. When the front porch period finishes, the sync pulse itself is toggled from HIGH to LOW (the sync pulses are active-low). This sync pulse signals the "reset" of the xy counter: the hsync pulse resets the xy counter to the left most of the next line, whereas the vsync pulse resets the xy counter to the top left of the screen. After sync pulse period finishes, the sync pulse is toggled back to HIGH for a period of time called the *back porch*.

Figure 2.1 shows the VGA timing sequence for the 640x480 resolution with 60 Hz refresh rate.

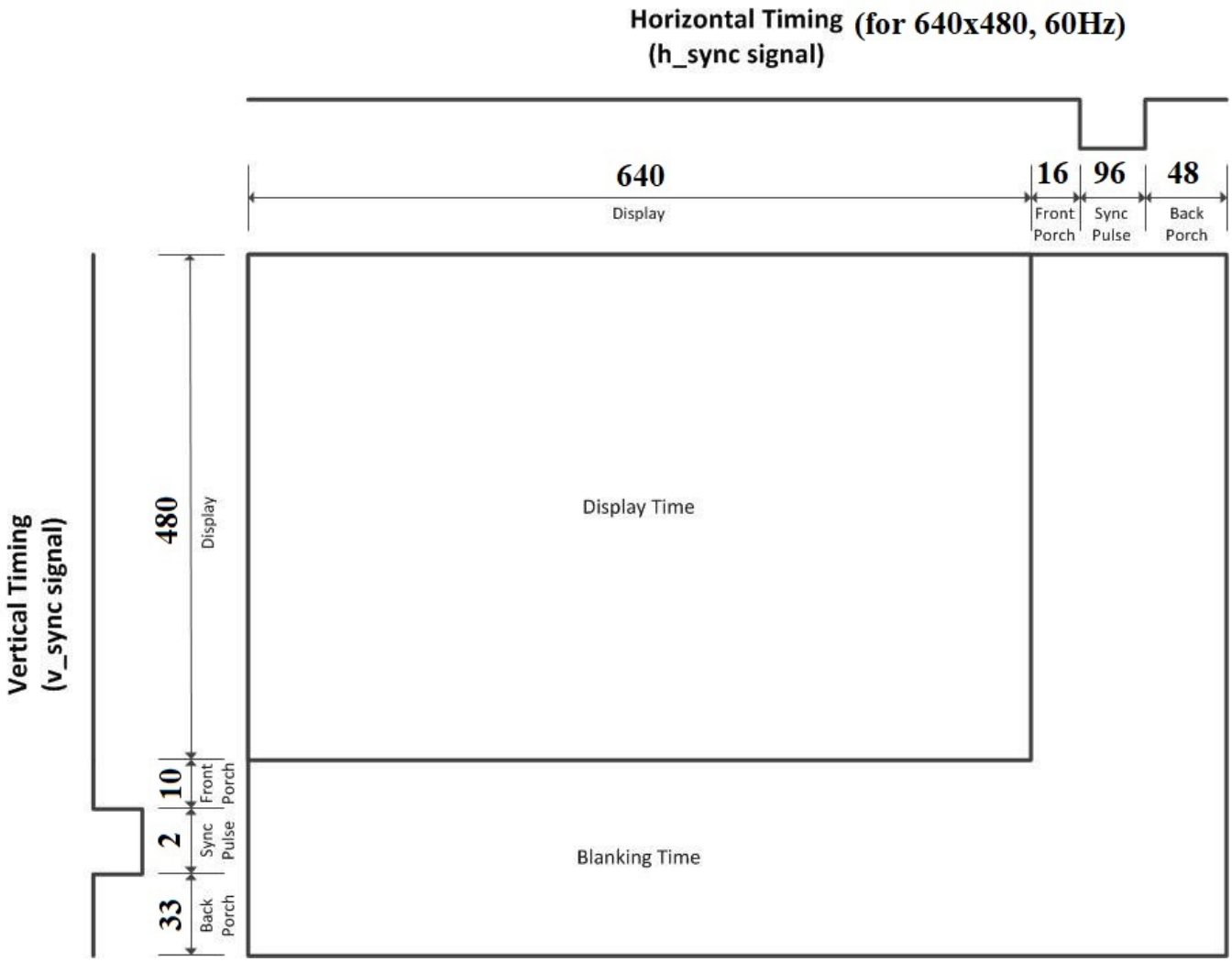

Figure 2.1: The display, front porch, sync pulse, and back porch signals of the horizontal timing and vertical timing of 640x480 60Hz monitor

Think of the timing sequence like this: the total resolution is $640 + 16 + 96 + 48 = 800$ by $480 + 33 + 2 + 10 = 525$, but the active zone is 640x480 on the top left corner of the frame.

The block diagram of the design is illustrated in Figure 2.2 below:

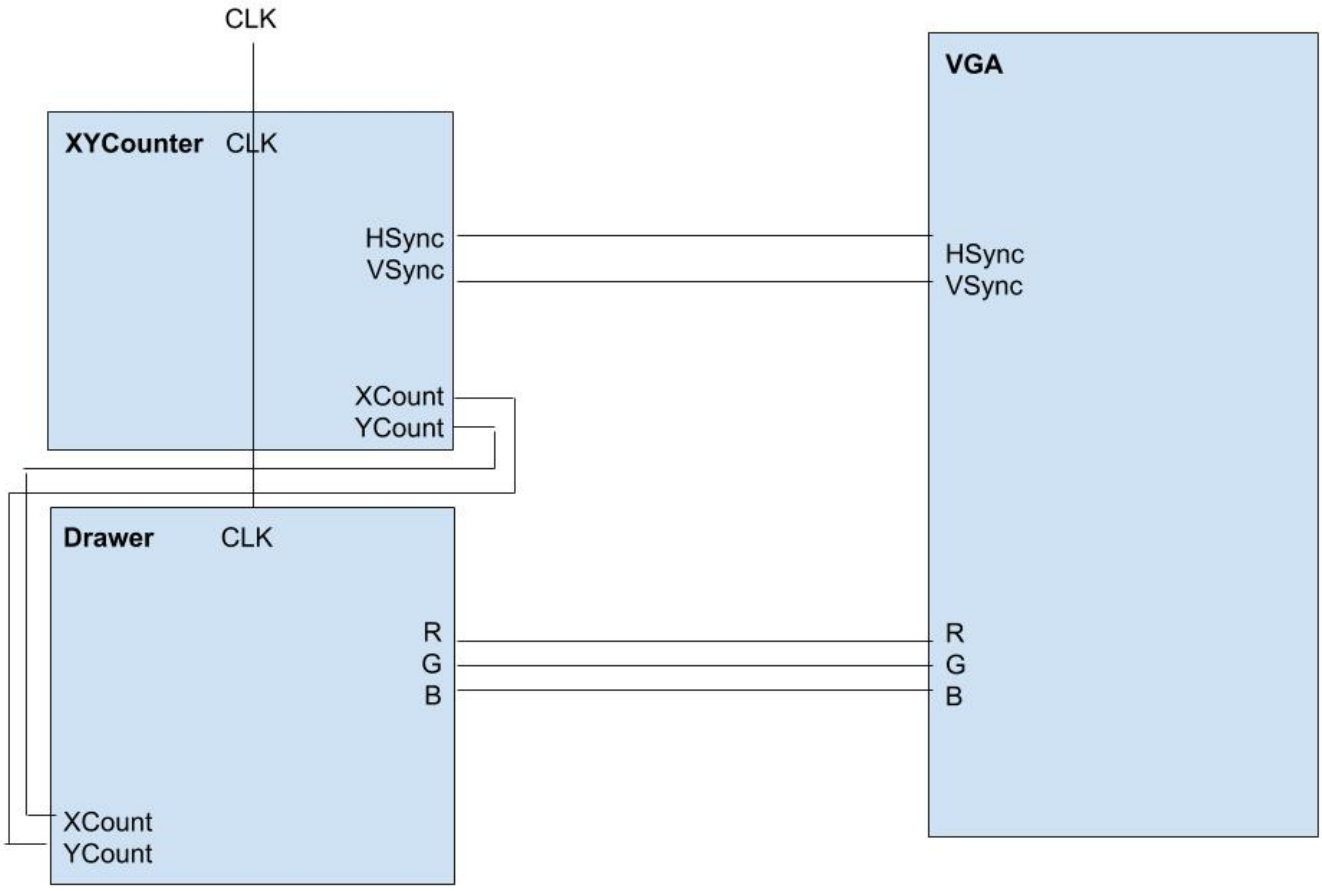

Figure 2.2: The block diagram

The SystemVerilog code which implements these modules is located in the Appendix section.

Putting the code through ModelSim simulation, the following results are yielded. The caption of each figure will explain the significance of the event the image captures.

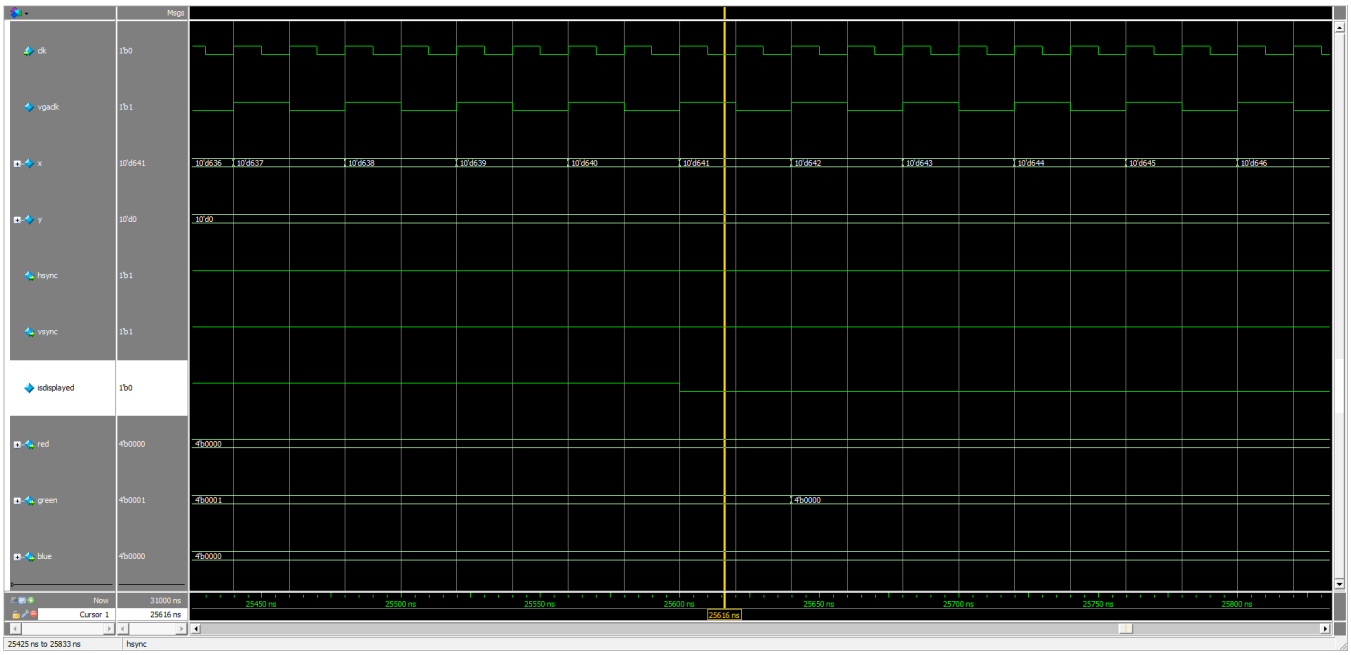

Figure 2.3: When x (the horizontal counter) reaches 640, the counter goes past the active resolution, and so this is where the **horizontal front porch starts**. High-resolution image

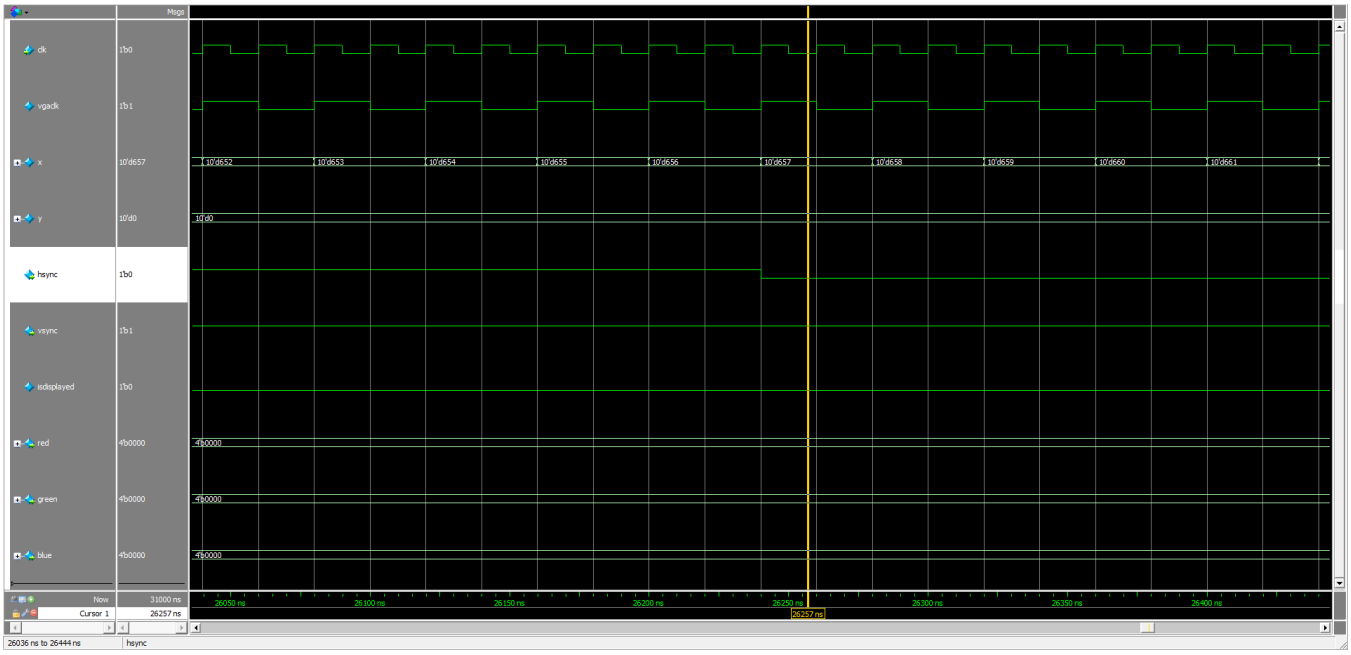

Figure 2.4: When x reaches 656, the counter has gone an extra 16—which is the width of the horizontal front porch, and so this is where the **horizontal sync pulse starts going LOW**. High-resolution image

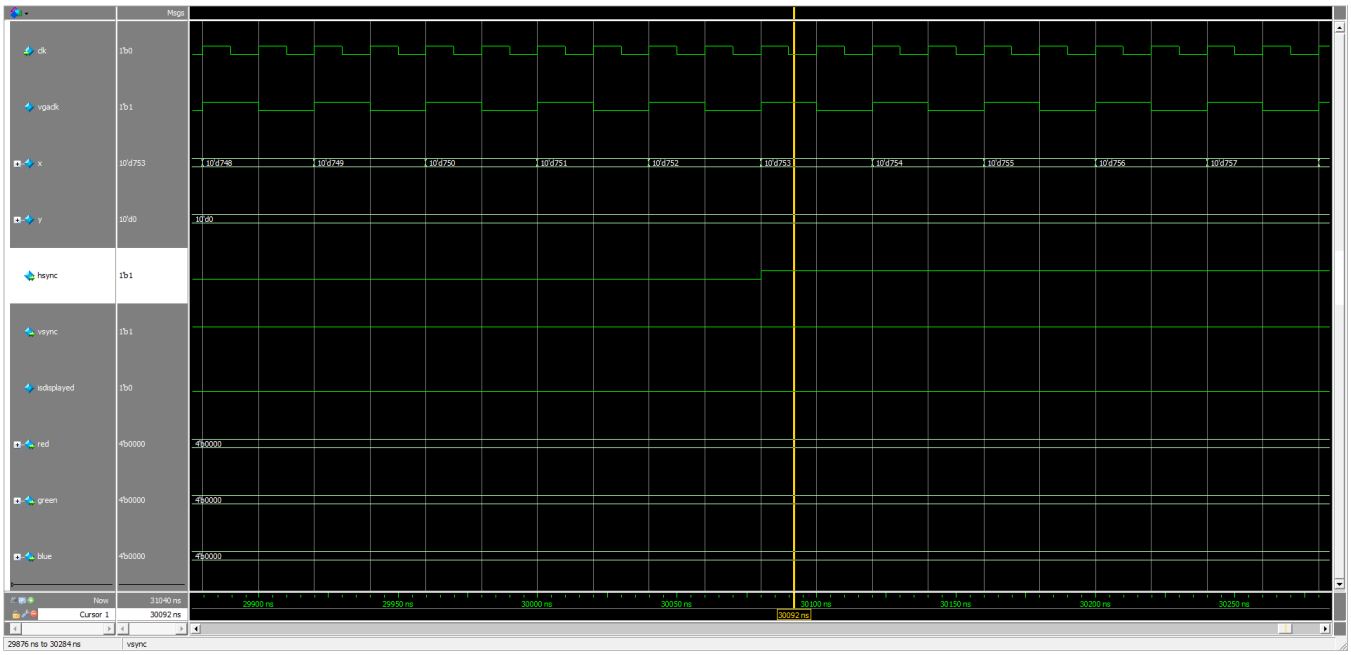

Figure 2.5: When x reaches 752, the counter has gone an extra 96—which is the width of the horizontal sync pulse, and so this is where the **horizontal back porch starts**. High-resolution image

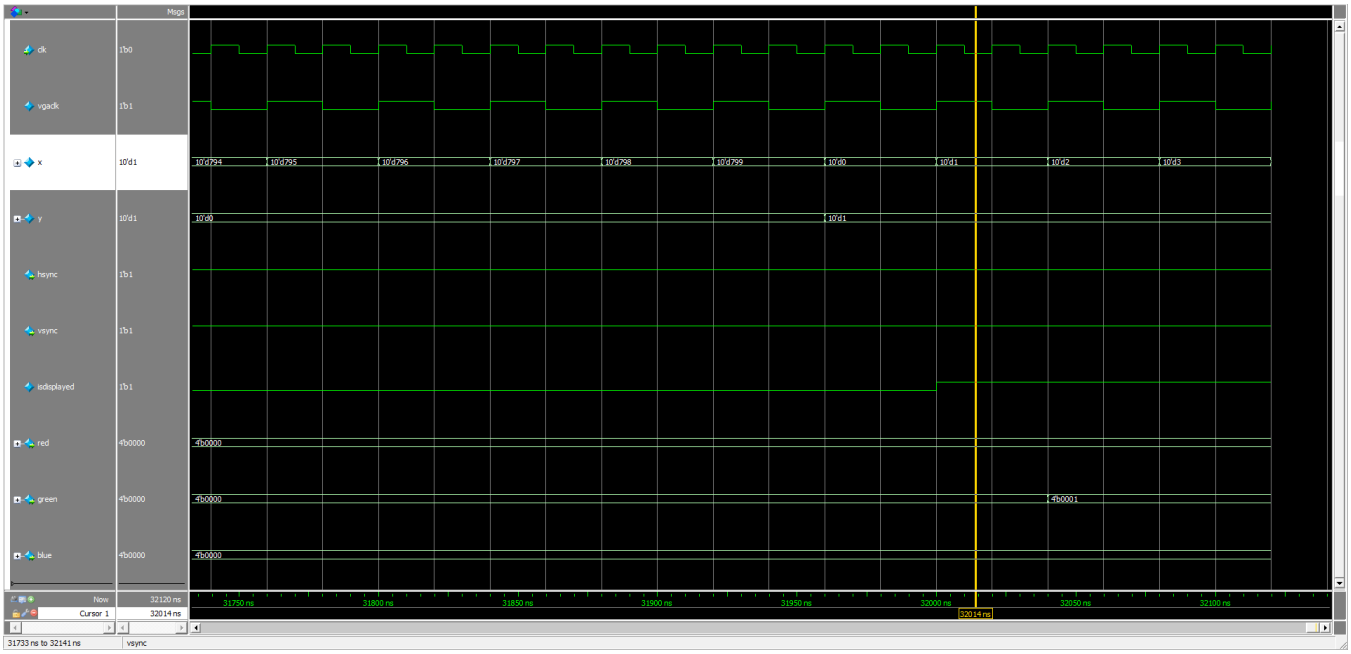

Figure 2.6: When x reaches 800, it will reset back to 0, and y (the vertical counter) will increased by 1. The counters will keep counting horizontally and then vertically until the vertical counter reaches its limit. High-resolution image

Similarly, the figures below show and explain how the vertical front porch, sync pulse, and back porch work.

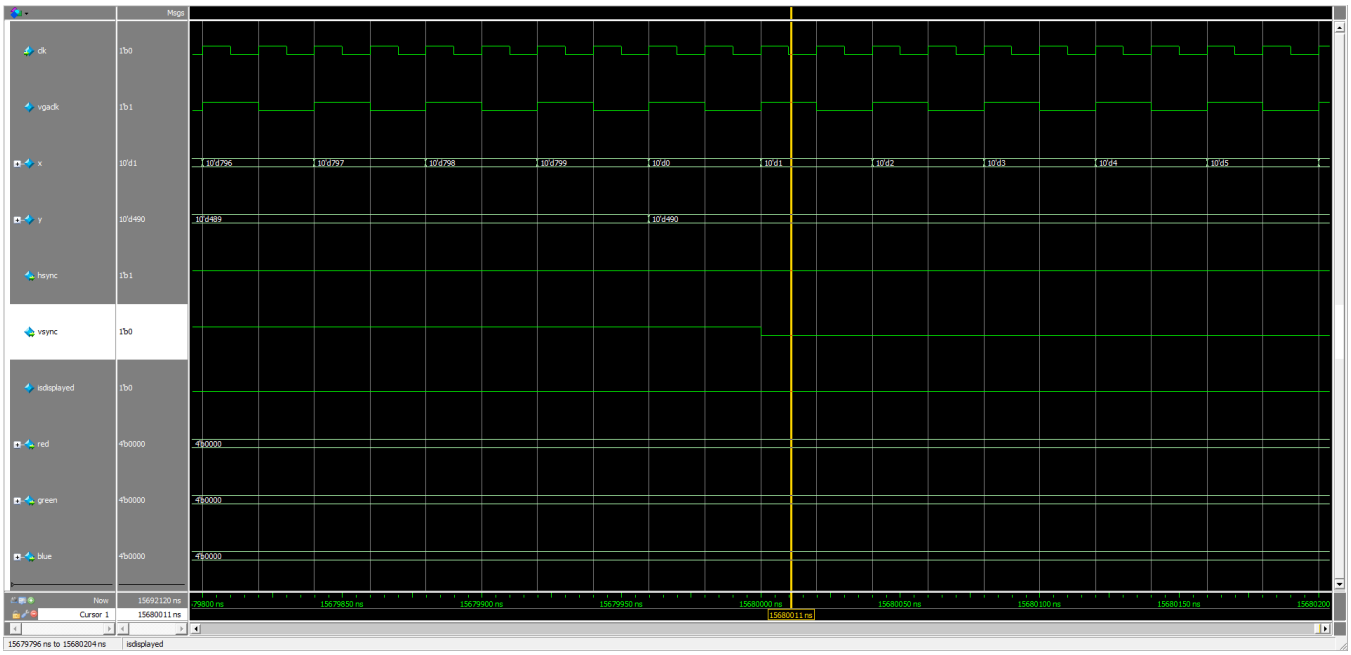

Figure 2.7: When x reaches 640 and y reaches 480, the counter has finished one whole active zone of the screen, and so this is where the **vertical front porch starts**. High-resolution image

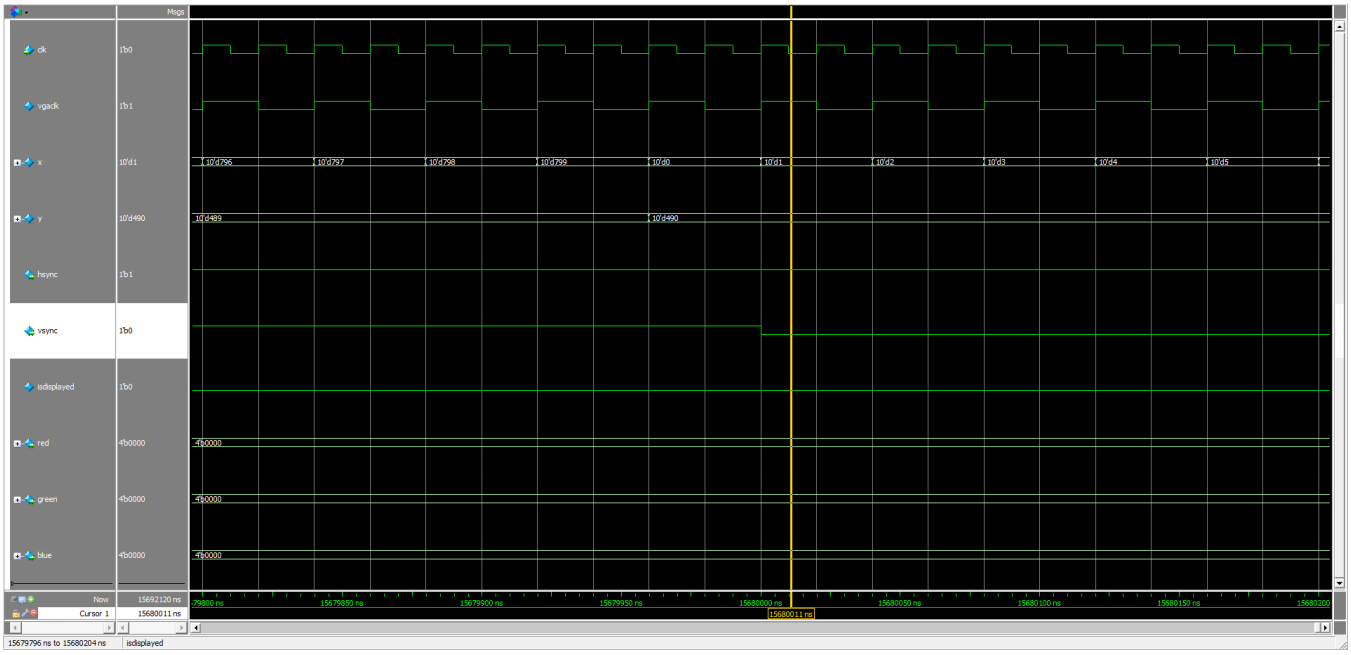

Figure 2.8: When y reaches 490, the counter has gone an additional 10—which is the width of the vertical front porch, and so this is where the **vertical sync pulse starts going LOW**. High-resolution image

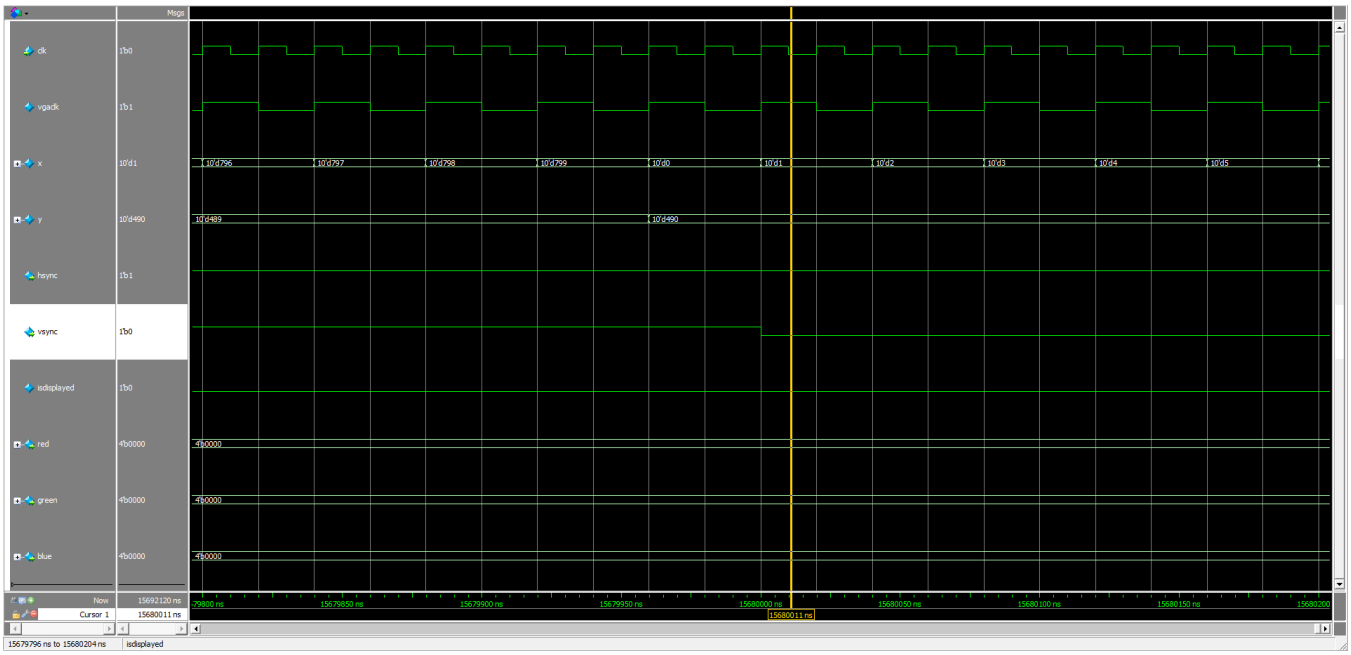

Figure 2.9: When y reaches 492, the counter has gone an additional 2—which is the width of the vertical sync pulse, and so this is where the **vertical back porch starts**. High-resolution image

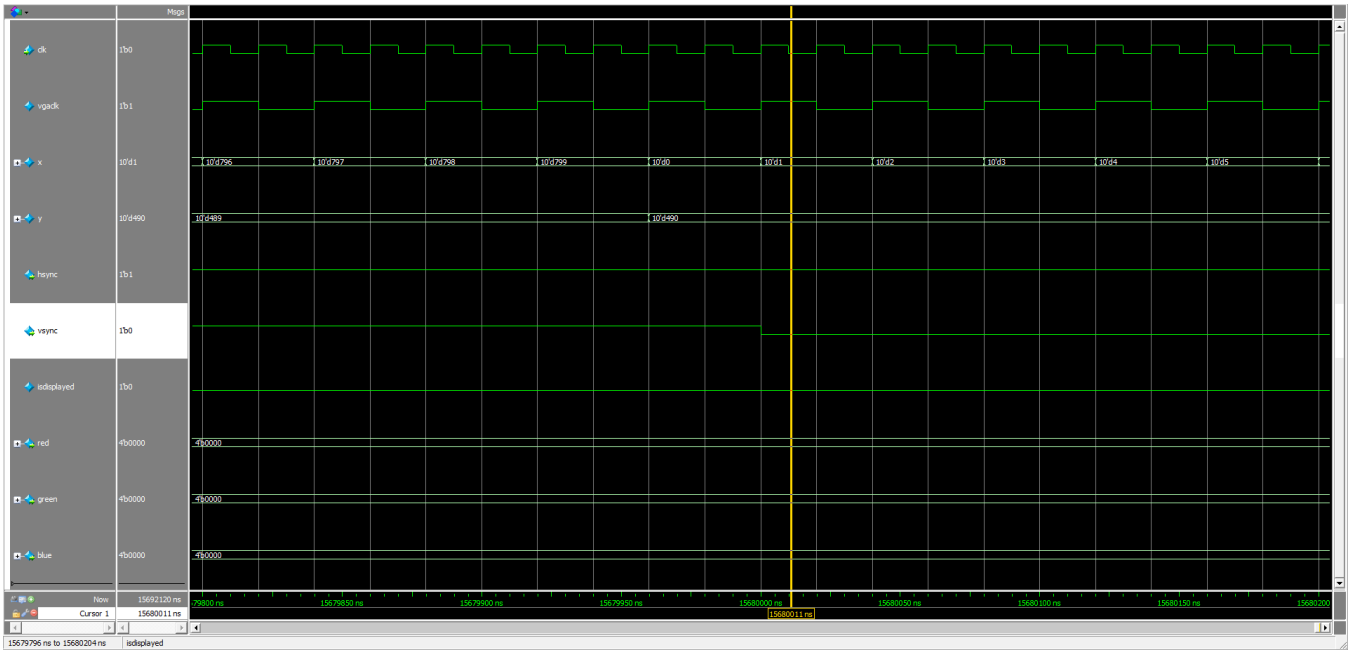

Figure 2.10: When x reaches 800 and y reaches 525, the counter has finished the entire screen. Since they have reached their limit, the counter will be wrapped around, and x and y will both reset to 0. High-resolution image

According to the figures above, the simulation yields the expected results. I uploaded the program to the FPGA and get the desired screen. Therefore, the VGA controller and drawer have been successfully implemented.

3 Experiment Notes

Reflection

This lab, particularly the timing sequence of the VGA protocol, was quite challenging. It took me more than 3 hours to finally get the timing right in SystemVerilog. The rest of the program was very easy, though. I was able to draw a red box on top of the blue background on the screen using this program.

Study Questions

1. What was the toughest aspect of ECE 272? What should be changed or added to the ECE 272 manual to make this course better?

   The toughest aspect of ECE 272 was the SystemVerilog tasks. It was difficult to do sequential logic and timing in SystemVerilog, especially the SPI and the VGA labs. A suggestion to make ECE 272 better is to add starter code into the lab manual, at least for the SPI and the VGA labs.

2. What would you like to explore further about Lattice Diamond or Digital Logic Design?

   I would like to learn more about building complex state machine using SystemVerilog. Also, I would like to explore different kinds of communication protocols in digital logic design, such as HDMI, DisplayPort, Bluetooth, etc.

3. What section of ECE 272 did you dislike the most? Why?

   I disliked this section (VGA) the most because the lab manual really helped me nothing. The vertical front porch and back porch timing of this lab's manual were inconsistent with those of the official DE10-Lite manual. The XY module design description in the Design section was also very vague.

4. What was your favorite section of ECE 272? Why?

   My favorite section of ECE 272 was the SPI lab because the objective was clear and easy to understand. The program did not have much timing into it, so it was clearly easier than the VGA lab. SPI was also one of the topics I'm familiar with as well.

Appendix

VgaTopLevel.qsf (Pin Assignment)

```
38
39  set_global_assignment -name FAMILY "MAX 10"
40  set_global_assignment -name DEVICE 10M50DAF484C7G
41  set_global_assignment -name TOP_LEVEL_ENTITY VgaTopLevel
42  set_global_assignment -name ORIGINAL_QUARTUS_VERSION 18.0.0
43  set_global_assignment -name PROJECT_CREATION_TIME_DATE "11:28:02  NOVEMBER 14, 2018"
44  set_global_assignment -name LAST_QUARTUS_VERSION "18.0.0 Lite Edition"
45  set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
46  set_global_assignment -name MIN_CORE_JUNCTION_TEMP 0
47  set_global_assignment -name MAX_CORE_JUNCTION_TEMP 85
48  set_global_assignment -name ERROR_CHECK_FREQUENCY_DIVISOR 256
49  set_global_assignment -name POWER_PRESET_COOLING_SOLUTION "23 MM HEAT SINK WITH 200 LFPM
    ↪  AIRFLOW"
50  set_global_assignment -name POWER_BOARD_THERMAL_MODEL "NONE (CONSERVATIVE)"
51  set_global_assignment -name SYSTEMVERILOG_FILE VgaDrawer.sv
52  set_global_assignment -name SYSTEMVERILOG_FILE VgaController.sv
53  set_global_assignment -name SYSTEMVERILOG_FILE HalfClock.sv
54  set_global_assignment -name SYSTEMVERILOG_FILE VgaTopLevel.sv
55  set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
56  set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL PLACEMENT_AND_ROUTING
    ↪  -section_id Top
57  set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
58
59  set_location_assignment PIN_P11 -to clk
60  set_location_assignment PIN_AA1 -to red[0]
61  set_location_assignment PIN_V1 -to red[1]
62  set_location_assignment PIN_Y2 -to red[2]
63  set_location_assignment PIN_Y1 -to red[3]
64  set_location_assignment PIN_W1 -to green[0]
65  set_location_assignment PIN_T2 -to green[1]
66  set_location_assignment PIN_R2 -to green[2]
67  set_location_assignment PIN_R1 -to green[3]
68  set_location_assignment PIN_P1 -to blue[0]
69  set_location_assignment PIN_T1 -to blue[1]
70  set_location_assignment PIN_P4 -to blue[2]
71  set_location_assignment PIN_N2 -to blue[3]
72  set_location_assignment PIN_N3 -to hsync
73  set_location_assignment PIN_N1 -to vsync
74
75
76  set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -section_id Top
```

VgaTopLevel.sv

```
module VgaTopLevel (
    input logic clk,
    output logic hsync, vsync,
    output logic [3:0] red, green, blue
);

logic vgaclk, isdisplayed;
logic [9:0] x, y;

HalfClock clock(
    .clk_in(clk),
    .clk_out(vgaclk)
);

VgaController vga(
    .vgaclk(vgaclk),
    .hsync(hsync),
    .vsync(vsync),
    .isdisplayed(isdisplayed),
    .x(x),
    .y(y)
);

VgaDrawer drawer(
    .clk(vgaclk),
    .isdisplayed(isdisplayed),
    .x(x),
    .y(y),
    .r(red),
    .g(green),
    .b(blue)
);

endmodule
```

HalfClock.sv

```
module HalfClock(
    input logic clk_in,
    output logic clk_out
);

logic out = 0;
```

```
7
8   always_ff @(posedge clk_in)
9       out <= ~out;
10
11  assign clk_out = out;
12
13  endmodule
```

VgaController.sv

```
1   module VgaController #(parameter
2       HACTIVE = 10'd640,                  // horizontal active resolution
3       HFP = 10'd16,                       // horizontal front porch
4       HSYN = 10'd96,                      // horizontal sync
5       HBP = 10'd48,                       // horizontal back porch
6       HMAX = HACTIVE + HFP + HSYN + HBP,  // horizontal max resolution
7
8       VACTIVE = 10'd480,                  // verical active resolution
9       VFP = 10'd10,                       // vertical front porch
10      VSYN = 10'd2,                       // vertical sync
11      VBP = 10'd33,                       // vertical back porch
12      VMAX = VACTIVE + VFP + VSYN + VBP   // verical max resolution
13  )(
14      input logic vgaclk,
15      output logic hsync, vsync, isdisplayed,
16      output logic [9:0] x, y  // 10 bits to cover all 640x480
17  );
18
19  logic hsync_hi = 0, vsync_hi = 0, displayed = 0;
20  logic [9:0] xcnt = 0, ycnt = 0;
21
22  // count the horizontal and vertical positions, wrap around max resolutions
23  always_ff @(posedge vgaclk) begin
24      // displayed  changes from HIGH to LOW at x == HACTIVE && y == VACTIVE
25      // and changes back from LOW to HIGH at x == 0 && y == 0
26      displayed <= (xcnt < HACTIVE) && (ycnt < VACTIVE);
27
28      // compute sync signals (change to active-low later)
29      // and ends right before the beginning of horizontal back porch
30      hsync_hi <= (HACTIVE + HFP <= xcnt) && (xcnt < HACTIVE + HFP + HSYN);
31
32      // vsync pulse starts right at the end of vertical front porch
33      // and ends right before the beginning of horizontal back porch
34      vsync_hi <= (VACTIVE + VFP <= ycnt) && (ycnt < VACTIVE + VFP + VSYN);
35
```

```
    // move to the next pixel. Wrap x around HMAX and y around VMAX
    if (xcnt == HMAX - 1) ycnt <= (ycnt + 1) % VMAX;
    xcnt <= (xcnt + 1) % HMAX;
end

// produce the outputs
assign {hsync, vsync, isdisplayed} = {~hsync_hi, ~vsync_hi, displayed};
assign {x, y} = {xcnt, ycnt};

endmodule
```

VgaDrawer.sv

```
module VgaDrawer#(parameter
    HACTIVE = 10'd640,                  // horizontal active resolution
    VACTIVE = 10'd480,                  // verical active resolution
    BOXW = 10'd320,                     // box width
    BOXH = 10'd240                      // box height
)(
    input logic clk,
    input logic isdisplayed,
    input logic [9:0] x, y,
    output logic [3:0] r, g, b
);

logic [3:0] r_out = 0, g_out = 0, b_out = 0;

always_ff @(posedge clk)
    if (isdisplayed)
        if ((HACTIVE - BOXW) / 2 <= x && x < (HACTIVE - BOXW) / 2 + BOXW
            && (VACTIVE - BOXH) / 2 <= y && y < (VACTIVE - BOXH) / 2 + BOXH)
            {r_out, g_out, b_out} <= {4'b1111, 4'b0, 4'b0};
        else
            {r_out, g_out, b_out} <= {4'b0, 4'b0, 4'b1111};
    else
        {r_out, g_out, b_out} <= {4'b0, 4'b0, 4'b0};

assign {r, g, b} = {r_out, g_out, b_out};

endmodule
```
