Lab EE 371: Display Interface

Autumn 2018

Lab 3

Shawnna Cabanday

Professor Rania Hussein

Teaching Assistant: Shuowei Li

Lab Section: Wednesday from 12:30 - 2:30 PM

Date submitted: Saturday, October 27, 2018

Abstract

The purpose of this lab was to understand how to successfully display images on a monitor using the DE1-SoC's video-out port through the VGA terminal. The primary tasks of this lab was to implement a simple line-drawing algorithm and be able to animate those lines on a computer monitor.

Introduction

The DE1_SoC has a video-out port with a VGA controller that can be connected to a standard VGA monitor. Images that are displayed by the VGA controller are derived from the pixel buffer, which holds the color for each pixel displayed by the controller. The pixel buffer has a '(x,y) coordinate' system of 320x240 pixels where images can be written by color values into specified pixel addresses. In this lab, students must understand how to implement a simple line-drawing algorithm known as Bresenham's algorithm. Then, the algorithm must be used to draw a line on the monitor and animate it to move around the screen.

Materials

- ModelSim and Quartus 17.0 Software
- Power cable
- DE1_SoC FPGA board
- VGA Adapter for Monitor
- Spare Monitor

Procedures

Part 1: Implement Bresenham's Line Algorithm in System Verilog

The task of this lab procedure was to translate the psuedo-code for Bresenham's Line algorithm (created in high-level language) into System Verilog code. The primary difference from VHDL and high-level language is the order in which each the lines of code are executed. For VHDL, every line of code runs at the same time, whereas, high level language runs each line one after another.

The first module that was created was the *abs* module, which took in a variable, determined the sign of the value based on the first bit (whether it was 1 or 0), and then outputted the absolute value of the input value. The following logic variables were then created in the top level *line_drawer* module:

- deltax: equal to the difference between the rightmost x-value (x1) and the leftmost x-value (x0)
- deltay: equal to the difference in the leftmost and rightmost y values (y1-y0)
- abs_deltax, abs_deltay: the absolute values of deltax and deltay
- tempx0, tempx1, tempy0, tempy1: temporary storage registers used to re-write and write the most current values of the x0, x1, y0, and y1 without re-assigning the input logic x0, x1, y0, y1.
- ystep: used to increment current most value written to the final plotted y
- error: adjusts ystep
- write_x, write_y: final values of x and y that need to be plotted, plotted by assignment to final output logic of x and y
- state: indicates the states in the state machine
- finished_drawing: indicates when the verilog program has reached the last x-value (x1)

- delay: received from the DE1_SoC module to determine if the counter is incrementing and causing a delay, high signal causes a “reset” and stalls the *line_drawer* module

To mimic the behavior of high-level language, a functional state machine must be used to go from one step of System Verilog code to another. There are four states indicated by decimal values 1, 2, 3, and 4, and are represented by the parameter names: assign_temps, swap_values, prepare_output, and write_to_output, respectively. The following states represent the following:

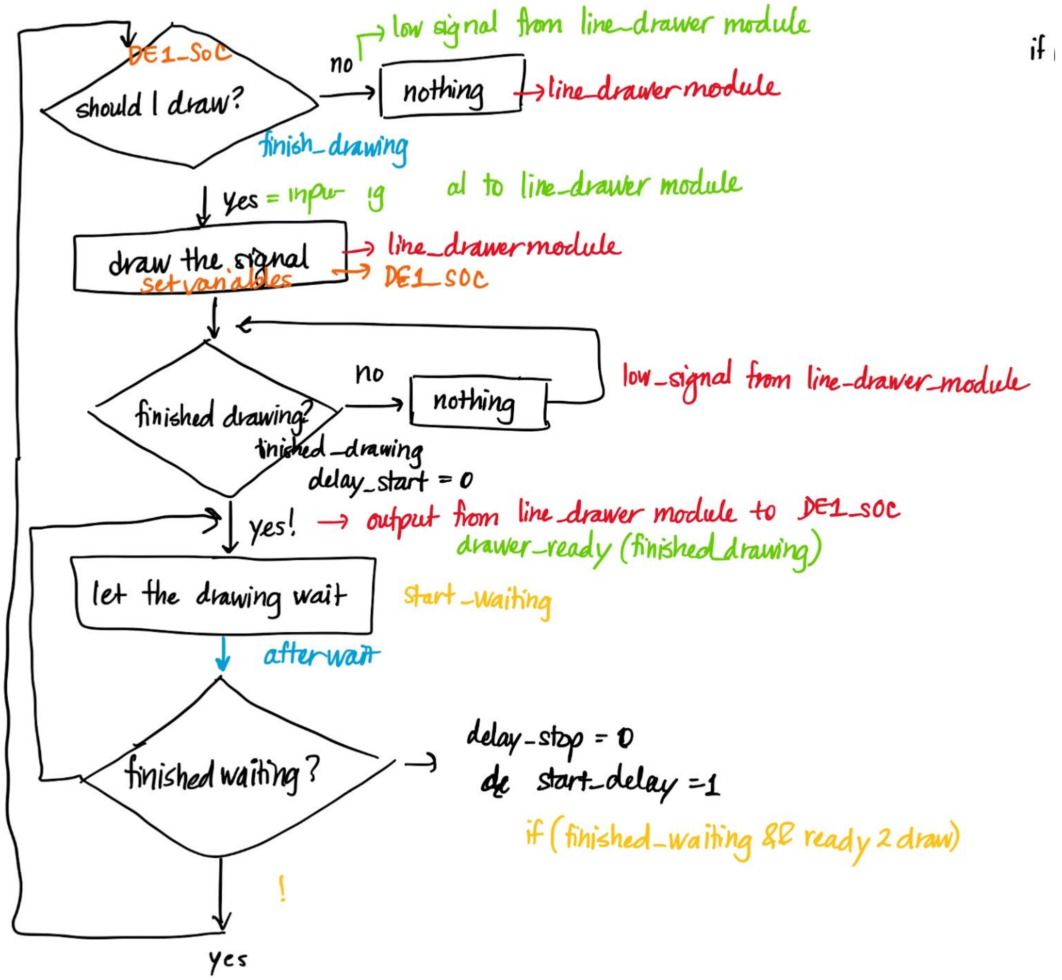
- assign_temps: reset finished drawer signal, calculate deltax and deltay, assign the is_steep boolean statement, and store all input values of x0, x1, y0, and y1 into their corresponding temporary variables.
- swap_values: swap values according to the pseudo-code provided. I also added additional logic to prevent errors in the case the x0 > x1.
- prepare_output: this stage prepares the output to be written by determining error, calculating ystep, assign write_x to the initial x0 and write_x to the initial y0.
- write_to_output: begins drawing the lines according to the algorithm and assigning x and y to the current most value of write_x or write_y or vice versa depending on whether or not is_steep is true or false. write_x is then incremented until the last value of x1 is reached. Afterwards, the state is sent back to the initial assign_temps state.

  There are two flagged signals being passed to and from the top level module *DE1_SoC* and the *line_drawer* module. They include: finished_drawing and start_drawing and indicate when the line_drawer has completed the algorithm for drawing a line and when the DE1_SoC is ready to start drawing with a new (x,y) input.

Part 2: Modifying the DE1_SoC Module to Animate a Line and Allow Reset State

The task of this lab procedure is to revise the *DE1_SoC* module to animate a line created by the *line_drawer* module.

To execute this task, a functional state machine was created with 8 different states: line1, eraseline1, line2, eraseline2, line3, eraseline3, line4, and eraseline4. The following sketch illustrates the progression from one state to another:



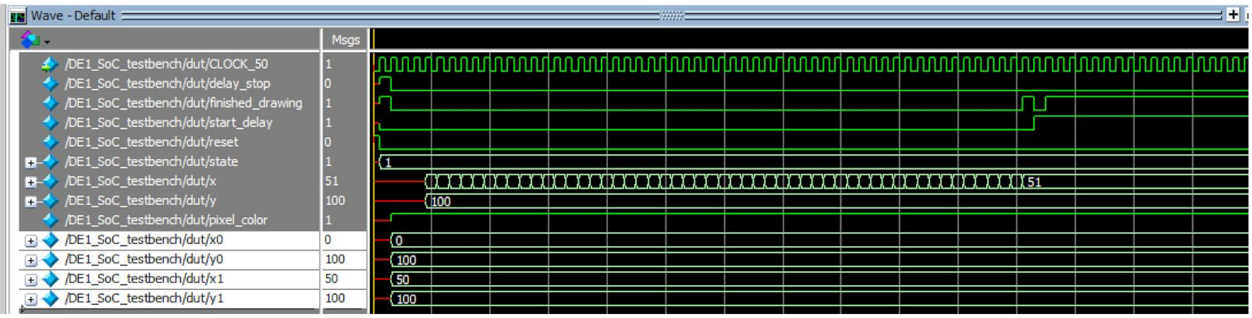
If the *line_drawer* sends a signal to the *DE1_SoC* module that it is *finished_drawing*, and when the *start_delay* signal is false, the DE1_SoC will allow the *line_drawer* to begin creating the first line. After that line is created, the *finished_drawing* signal becomes high again and the system checks if a delay has not been introduced. If this is true, then the delay will be initialized to keep the currentmost line on the screen. Once the delay is finished, the counter module outputs a signal *delay_stop*, stops the *start_delay* signal and continues to the next state. This following pattern continues for the rest of the states, drawing a line in white, erasing the previous one by drawing a black version of it, until the last one, which returns to the initial line1 drawing. 4 white lines are drawn, and 4 black lines are drawn to cover each one before the next one is shown on the screen. This will make it appear as 4 horizontal lines are traveling from left to right.

Results

This lab took me quite a lot of time to finish, and the most difficult task was converting the psuedo-code to VHDL. The difficulty with doing that is because every line of code is executed at the same time, without delay, which is unlike all high-level languages.

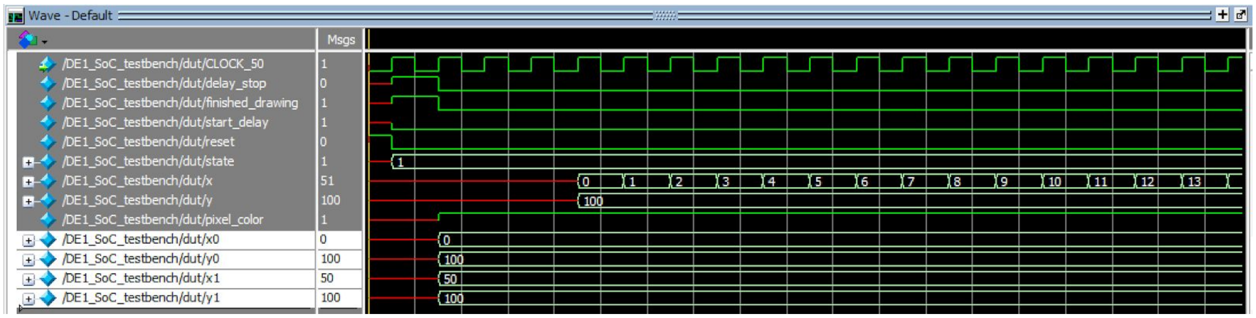
Both the *line_drawer* and the *DE1_SoC* behaved according to expectations after immense trial and error. The line drawer was capable of graphing diagonal, horizontal, and vertical lines, and the lab demo for that particular task was successful. The *DE1_SoC* demo, however, did not work during the lab demo. I have revised the code since, and, based on the waveforms, I am determining that the system is capable of working; I just simply can't test it. Thus, this remains an unsolved error.

The following waveforms illustrate the success of the *line drawer* and the revised behavior of the *DE1_SoC* module. In Figure 1.1, the DE1_SoC successfully draws a horizontal line from (0,100) to (50, 100), and once the last x-value is reached, the start_delay signal is triggered and the *counter* module is then called to delay the device from going to the next state and erasing the newly created line. The delay is about 1 second. Figure 1.2 illustrates the zoomed in version of the waveforms to depict the incrementing x-coordinates. Figure 1.3 demonstrates the creation of a horizontal and diagonal line. The diagonal line created has beginning points (1,1) and endpoints of (12,5), whereas, the horizontal line has beginning points (1,1,) and endpoints of (4,1). Note the y-values incrementing based on the algorithm. Figure 1.4 depicts the implementation of the delay signal which stalls the creation of future (x,y) coordinates and lines after the first line from (1,1) to (12,5) is created.



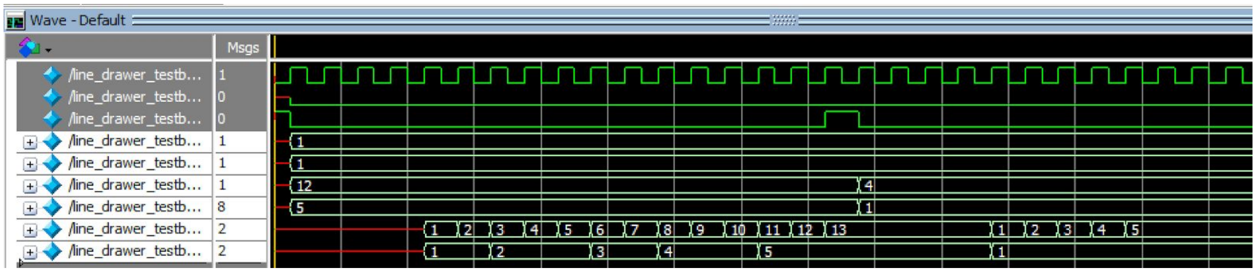
	Msgs
/DE1_SoC_testbench/dut/CLOCK_50	1
/DE1_SoC_testbench/dut/delay_stop	0
/DE1_SoC_testbench/dut/finished_drawing	1
/DE1_SoC_testbench/dut/start_delay	1
/DE1_SoC_testbench/dut/reset	0
/DE1_SoC_testbench/dut/state	1
/DE1_SoC_testbench/dut/x	51
/DE1_SoC_testbench/dut/y	100
/DE1_SoC_testbench/dut/pixel_color	1
/DE1_SoC_testbench/dut/x0	0
/DE1_SoC_testbench/dut/y0	100
/DE1_SoC_testbench/dut/x1	50
/DE1_SoC_testbench/dut/y1	100

Figure 1.1: DE1_SoC Waveform Output (Includes Delay)



	Msgs
/DE1_SoC_testbench/dut/CLOCK_50	1
/DE1_SoC_testbench/dut/delay_stop	0
/DE1_SoC_testbench/dut/finished_drawing	1
/DE1_SoC_testbench/dut/start_delay	1
/DE1_SoC_testbench/dut/reset	0
/DE1_SoC_testbench/dut/state	1
/DE1_SoC_testbench/dut/x	51
/DE1_SoC_testbench/dut/y	100
/DE1_SoC_testbench/dut/pixel_color	1
/DE1_SoC_testbench/dut/x0	0
/DE1_SoC_testbench/dut/y0	100
/DE1_SoC_testbench/dut/x1	50
/DE1_SoC_testbench/dut/y1	100

Figure 1.2: DE1_SoC Waveform Output (No Delay)



	Msgs
/line_drawer_testb...	1
/line_drawer_testb...	0
/line_drawer_testb...	0
/line_drawer_testb...	1
/line_drawer_testb...	1
/line_drawer_testb...	1
/line_drawer_testb...	8
/line_drawer_testb...	2
/line_drawer_testb...	2

Figure 1.3: Line Drawer (Diagonal and Horizontal Example)

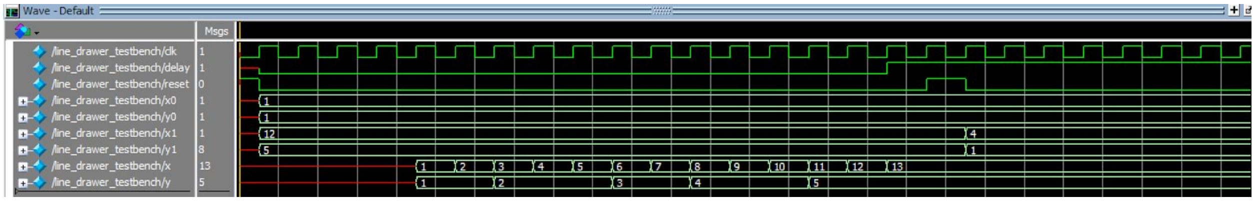
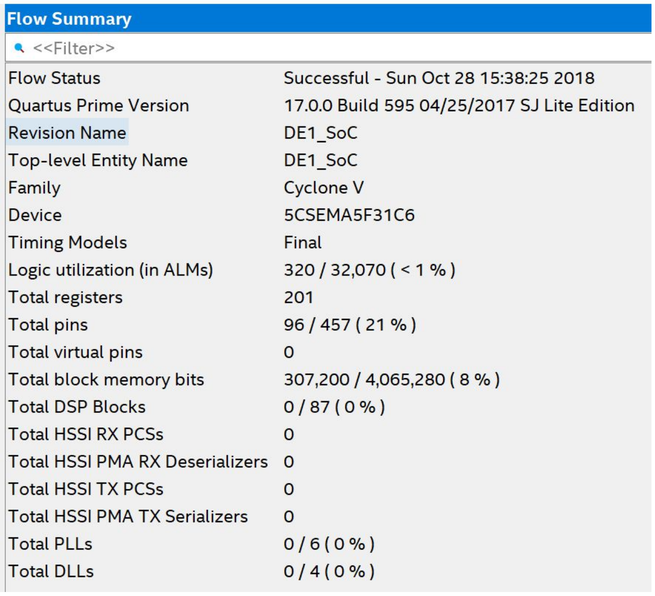

Wave - Default
Msgs
/line_drawer_testbench/clk 1
/line_drawer_testbench/delay 1
/line_drawer_testbench/reset 0
/line_drawer_testbench/x0 1
/line_drawer_testbench/y0 1
/line_drawer_testbench/x1 1
/line_drawer_testbench/y1 8
/line_drawer_testbench/x 13
/line_drawer_testbench/y 5
1
1
12
4
5
1
1 2 3 4 5 6 7 8 9 10 11 12 13
1 2 3 4 5

Figure 1.4: Line Drawer (Delay Included)

Flow Summary



Flow Summary	
<<Filter>>	
Flow Status	Successful - Sun Oct 28 15:38:25 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	320 / 32,070 (< 1 %)
Total registers	201
Total pins	96 / 457 (21 %)
Total virtual pins	0
Total block memory bits	307,200 / 4,065,280 (8 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

The flow summary indicates that I use an extreme amount of registers! For future reference, I should limit the number of temporary registers I use to store values, in the case that I want to improve execution time.

Conclusion

Now, the DE1-SoC's video-out port can be used with a VGA controller to connect and display lines on a VGA monitor. This lab helped to illustrate the major differences between coding in a VHDL program and a traditional high-level language like C or Java. The conveniences of coding in high-level language are especially noticed. There were significant hurdles that I had to jump through for this lab just in terms of syntax and creating delay, and it proved to be extremely time consuming for me. For future reference, I need to make sure to start these labs earlier than initially planned.

Total Estimated Hours: 20 hours

Acknowledgements: Annika Lawrence, Truong Phu Nguyen

Appendix

DE1_SoC Module

```
module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
        VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);

        output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
        output logic [9:0] LEDR;
        input logic [3:0] KEY;
        input logic [9:0] SW;

        input CLOCK_50;
        output [7:0] VGA_R;
        output [7:0] VGA_G;
        output [7:0] VGA_B;
        output VGA_BLANK_N;
        output VGA_CLK;
        output VGA_HS;
        output VGA_SYNC_N;
        output VGA_VS;

        assign HEX0 = '1;
        assign HEX1 = '1;
        assign HEX2 = '1;
        assign HEX3 = '1;
        assign HEX4 = '1;
        assign HEX5 = '1;
        assign LEDR = SW;

        logic [10:0] x0, y0, x1, y1, x, y;
        logic finished_drawing, reset, start_delay, delay_stop, pixel_color, delay;
        logic [2:0] state;

        assign reset = ~KEY[0];

        parameter      black = 1'b0,
                                      white = 1'b1;

        parameter [2:0]       line1 = 1,
                                                    eraseline1 = 2,
                                                    line2 = 3,
                                                    eraseline2 = 4,
                                                    line3 = 5,
                                                    eraseline3 = 6,
                                                    line4 = 7,
                                                    eraseline4 = 8;

        VGA_framebuffer fb(.clk50(CLOCK_50), .reset, .x, .y,
                              .pixel_color, .pixel_write(1'b1),
                              .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .VGA_VS,
                              .VGA_BLANK_n(VGA_BLANK_N), .VGA_SYNC_n(VGA_SYNC_N));

        line_drawer lines (.clk(CLOCK_50), .reset, .delay(start_delay),
                              .x0, .y0, .x1, .y1, .x, .y, .finished_drawing);

        counter delay_time (.clk(CLOCK_50), .start_delay, .reset, .delay_stop);
```

```
       //Task 2 DEMO
//     assign x0 = 0;
//     assign y0 = 0;
//     assign x1 = 240;
//     assign y1 = 240;

       always @(posedge CLOCK_50) begin
               if(reset) begin
                      state = line1;
                      start_delay = 0;
               end
               case(state)
                      line1: if(finished_drawing && delay_stop && !start_delay) begin
                                                           pixel_color = white;
                                                           x0 = 0; y0 = 100; x1 = 50; y1 = 100;
                                            end
                                            else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                            else if(delay_stop && start_delay) begin
start_delay = 0; state = eraseline1; end
                      eraseline1: if(finished_drawing && delay_stop && !start_delay) begin
                                                           pixel_color = black;
                                                           x0 = 0; y0 = 100; x1 = 50; y1 = 100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = line2; end
                      line2: if(finished_drawing && delay_stop && !start_delay) begin
                                                           pixel_color = white;
                                                           x0 = 70; y0 = 100; x1 = 120; y1 =
100;
                                            end
                                            else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                            else if(delay_stop && start_delay) begin
start_delay = 0; state = eraseline2; end
                      eraseline2: if(finished_drawing && delay_stop && !start_delay) begin
                                                           pixel_color = black;
                                                           x0 = 70; y0 = 100; x1 = 120; y1 =
100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = line3; end

                      line3: if(finished_drawing && delay_stop && !start_delay) begin
                                                           pixel_color = white;
                                                           x0 = 140; y0 = 100; x1 = 190; y1 =
100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = eraseline3; end
```

```
                      eraseline3: if(finished_drawing && delay_stop && !start_delay) begin
                                                          pixel_color = black;
                                                          x0 = 140; y0 = 100; x1 = 190; y1 =
100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = line4; end

                      line4   : if(finished_drawing && delay_stop && !start_delay) begin
                                                          pixel_color = white;
                                                          x0 = 210; y0 = 100; x1 = 260; y1 =
100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = eraseline4; end

                      eraseline4     : if(finished_drawing && delay_stop && !start_delay) begin
                                                          pixel_color = black;
                                                          x0 = 210; y0 = 100; x1 = 260; y1 =
100;
                                                   end
                                                   else if(finished_drawing && !start_delay &&
!delay_stop) start_delay = 1;
                                                   else if(delay_stop && start_delay) begin
start_delay = 0; state = line1; end


               endcase
       end


endmodule

module DE1_SoC_testbench();
       logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
       logic [9:0] LEDR;
       logic [3:0] KEY;
       logic [9:0] SW;

       logic CLOCK_50;
       logic [7:0] VGA_R;
       logic [7:0] VGA_G;
       logic [7:0] VGA_B;
       logic VGA_BLANK_N;
       logic VGA_CLK;
       logic VGA_HS;
       logic VGA_SYNC_N;
       logic VGA_VS;
       logic clk;

       DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW,
.CLOCK_50(clk),
       .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N, .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
```

```
        parameter CLOCK_PERIOD = 100;
        initial begin
                clk <= 0;
                forever #(CLOCK_PERIOD/2) clk <= ~clk;

        end


        initial begin
                KEY <= 0;      @(posedge clk);
                KEY <= 1;      @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
                @(posedge clk);
        end
endmodule
```

Line_drawer Module Code

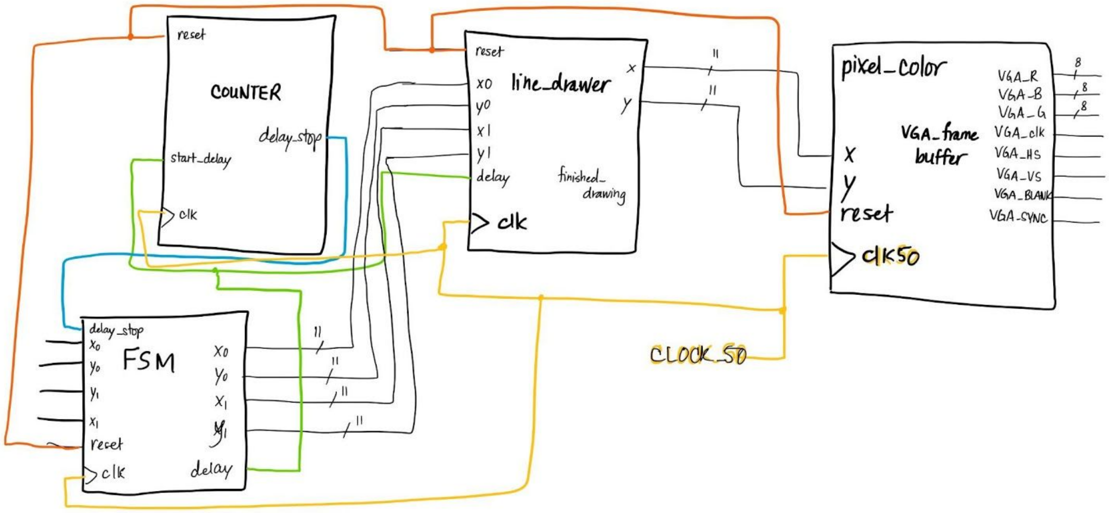
```
module line_drawer(
        input logic clk, reset,
        input logic delay,
        input logic [10:0]    x0, y0, x1, y1, //the end points of the line
        output logic [10:0]   x, y, //outputs corresponding to the pair (x, y)
        output logic finished_drawing
        );

        /*
         * You'll need to create some registers to keep track of things
         * such as error and direction
         * Example: */
        logic is_steep;
        logic signed [10:0] error;
        logic signed [10:0] deltax, deltay, abs_deltax, abs_deltay;
        logic signed [10:0] tempx0, tempx1, tempy0, tempy1, write_x, write_y;
        logic signed [1:0] ystep;
        logic [2:0] state;

        parameter [2:0] assign_temps = 1,
                              swap_values = 2,
                              prepare_output = 3,
                write_to_output = 4;

    abs absdeltax (.in(deltax), .out(abs_deltax));
    abs absdeltay (.in(deltay), .out(abs_deltay));

    always_ff @(posedge clk) begin
        if(reset||delay) begin
            deltax <= 0;
            deltay <= 0;
            error <= 0;
            ystep <= 1;
            is_steep <= 0;
            finished_drawing <= 1;
            state <= assign_temps;
        end
        else if(state == assign_temps) begin
            finished_drawing <= 0;
            deltax <= x1 - x0;
            deltay <= y1 - y0;
            is_steep <= abs_deltay > abs_deltax;
            tempx0 <= x0;
            tempx1 <= x1;
            tempy0 <= y0;
            tempy1 <= y1;
            state <= swap_values;
        end
        else if(state == swap_values) begin
            if(is_steep) begin
                if(tempy0 > tempy1) begin
                    tempx0 <= tempy1;
                    tempx1 <= tempy0;
                    tempy0 <= tempx1;
                    tempy1 <= tempx0;
                    deltax <= tempy0 - tempy1;
                    deltay <= (tempx1 > tempx0 ? tempx1-tempx0 : tempx0-x1);
                end
                else begin
                    tempx0 <= tempy0;
                    tempx1 <= tempy1;
                    tempy0 <= tempx0;
                    tempy1 <= tempx1;
                    deltax <= tempy1 - tempy0;
                    deltay <= (tempx1 > tempx0 ? tempx1-tempx0 :
tempx0-tempx1);
                end
            end
            else begin
                if(tempx0 > x1) begin
                    tempx0 <= tempx1;
                    tempx1 <= tempx0;
                    tempy0 <= tempy1;
                    tempy1 <= tempy0;
                    deltax <= tempx0-tempx1;
                    deltay <= (tempy1 > tempy0 ? tempy1-tempy0 :
tempy0-tempy1);
                end
                else begin
```

```
                                        deltax <= tempx1 - tempx0;
                                        deltay <= (tempy1 > tempy0 ? tempy1-tempy0 :
tempy0-tempy1);
                                end
                                state <= prepare_output;
                        end
                end
                else if (state == prepare_output) begin
                        error <= deltax/2;
                        ystep <= (y0 < y1 ? 1 : -1);
                        write_x <= x0;
                        write_y <= y0;
                        state <= write_to_output;
                end

                else if(state == write_to_output) begin
                        if(is_steep) begin
                                x <= write_y;
                                y <= write_x;
                        end
                        else begin
                                x <= write_x;
                                y <= write_y;
                        end

                        write_x <= write_x + 1;

                        if(error - deltay <0) begin
                                        error <= error - deltay + deltax;
                                        write_y <= write_y + ystep;
                        end
                        else begin
                                error <= error - deltay;
                        end

                        if(x == x1) begin
                                finished_drawing <= 1;
                                state <= assign_temps;
                        end
                end

        end     //end of always_ff


endmodule

module abs(in, out);
        input logic [10:0] in;
        output logic [10:0] out;

        assign out = in[10] ? -in : in;
endmodule
```

Counter Module

```
module counter(clk, start_delay, reset, delay_stop);
 input logic clk, reset, start_delay;
 output logic delay_stop;
 logic [31:0] delay;

 always @(posedge clk) begin
         if(reset) begin
          delay <= 0;
          delay_stop <= 1;
         end
         else if(start_delay) begin
          delay <= 0;
          delay_stop <= 0;
         end
         else if(delay == 32'h77359400) begin // if(delay = 2 seconds)
          delay_stop <= 1;
         end
         else begin
          delay <= delay + 1;
          delay_stop <= 0;
         end
 end

endmodule
module counter_testbench();
        logic clk, reset, start_delay, delay_stop;
        logic [31:0] delay;

        parameter CLOCK_PERIOD = 100;
        initial begin
         clk <= 0;
         forever #(CLOCK_PERIOD/2) clk <= ~clk;

        end

        counter dut (.clk, .start_delay, .reset, .delay_stop);

        initial begin
        start_delay <= 1; @(posedge clk);
        start_delay <= 0; @(posedge clk);
        End
endmodule
```

Block Diagram for DE1_SoC Top Level Module



reset
COUNTER
delay_stop
start_delay
clk
reset
x0
y0
x1
y1
line_drawer
x
y
delay
finished_drawing
clk
11
11
pixel_color
VGA_frame buffer
x
y
reset
clk50
VGA_R
VGA_B
VGA_G
8
8
8
VGA_clk
VGA_HS
VGA_VS
VGA_BLANK
VGA_SYNC
CLOCK_50
delay_stop
x0
y0
y1
x1
reset
clk
FSM
X0
Y0
X1
Y1
delay
11
11
11
11