Khoa Tran
EE 371
April 23, 2021
Lab 2 Report

Procedure

Task #1

Approaching this problem, I first drew up the block diagram to figure out the necessary inputs of the line_drawer through passing in two coordinates as starting and ending points and figure out the outputs of x and y for the VGA_framebuffer to receive a x and y address as well as a color to output the given line. In order to do this, I looked through the pseudocode of the Bresenham's line algorithm in order to understand the conditions of combinational logic as it manipulates previous values and updates on the clock edge. As the algorithm is based on if the line is steep or not, I was able to decipher the code and understand the conditions of setting initial x and y values in order to progress and output the line with minimal errors.

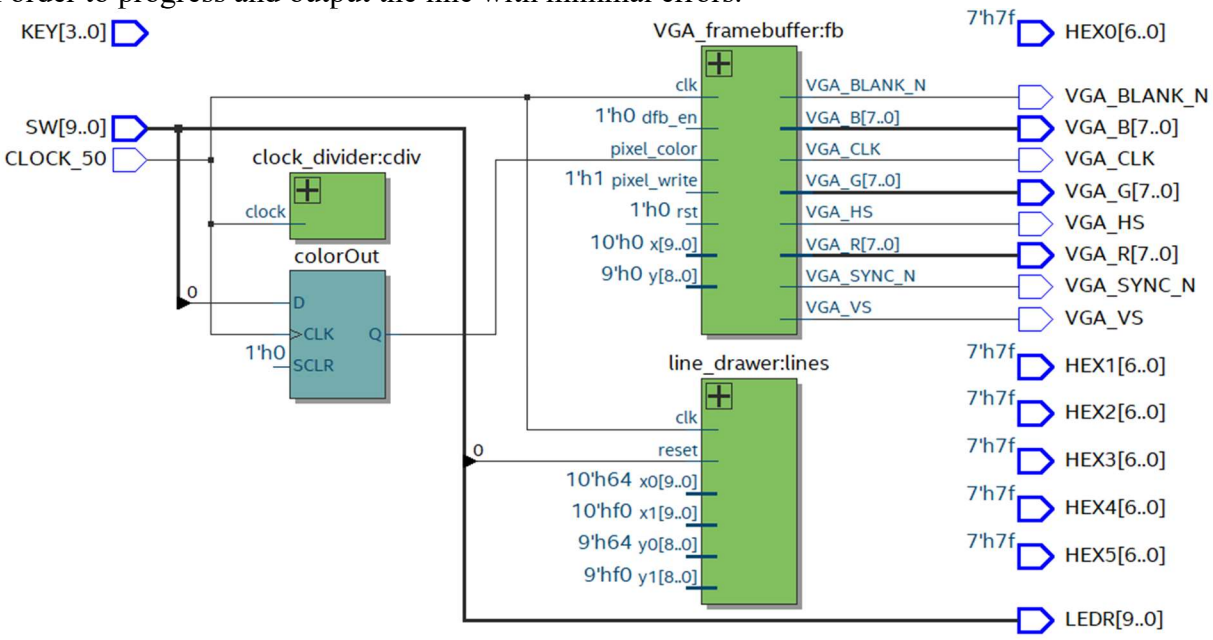

Figure 1: Block diagram for task 1

Task #2

Approaching this problem, I first drew up finite state machine that controls the inputs of the line_drawer based on conditions of a counter and if the line is done printing given from the line_drawer module. The finite state machine that I developed starts out with initial points for the line to be drawn. Then the line is cleared by giving the VGA an output of the color black and the line_drawer the same coordinate inputs. Afterwards, the fsm goes through a series of states that increments the values of either x or y coordinates in order to animate the line in a circle around the VGA board. After every line display, the line is then cleared. Also, there is a switch that clears the entire display of the board.

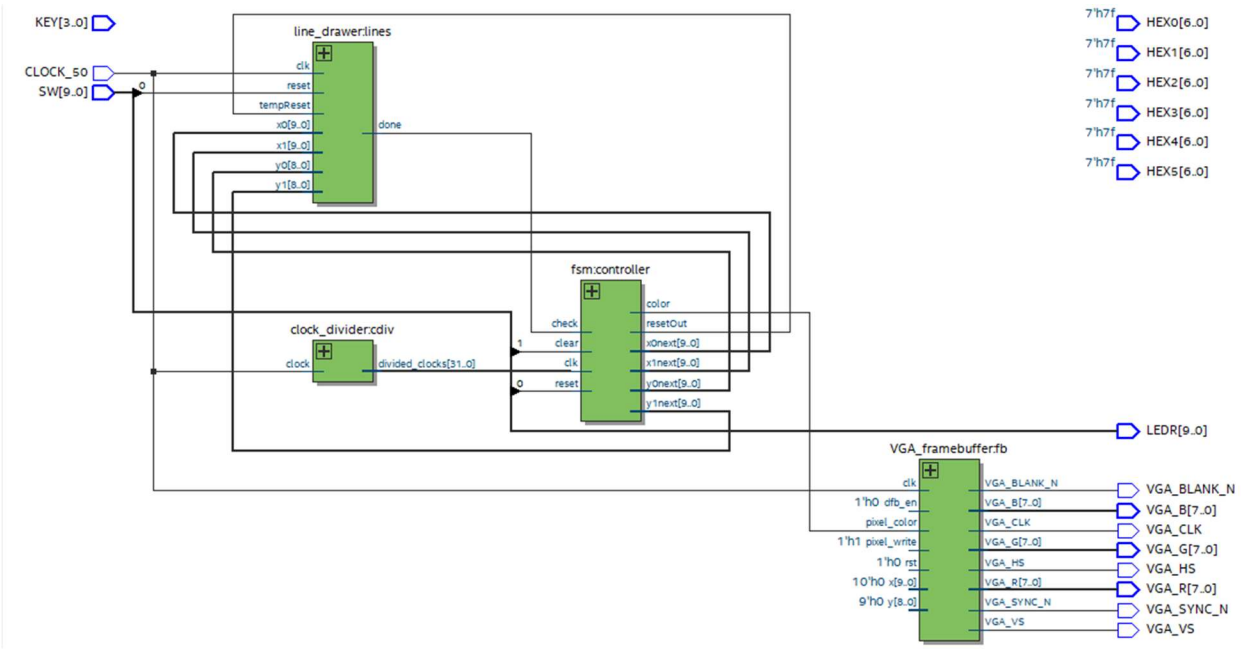

Figure 2: Block diagram for task 2

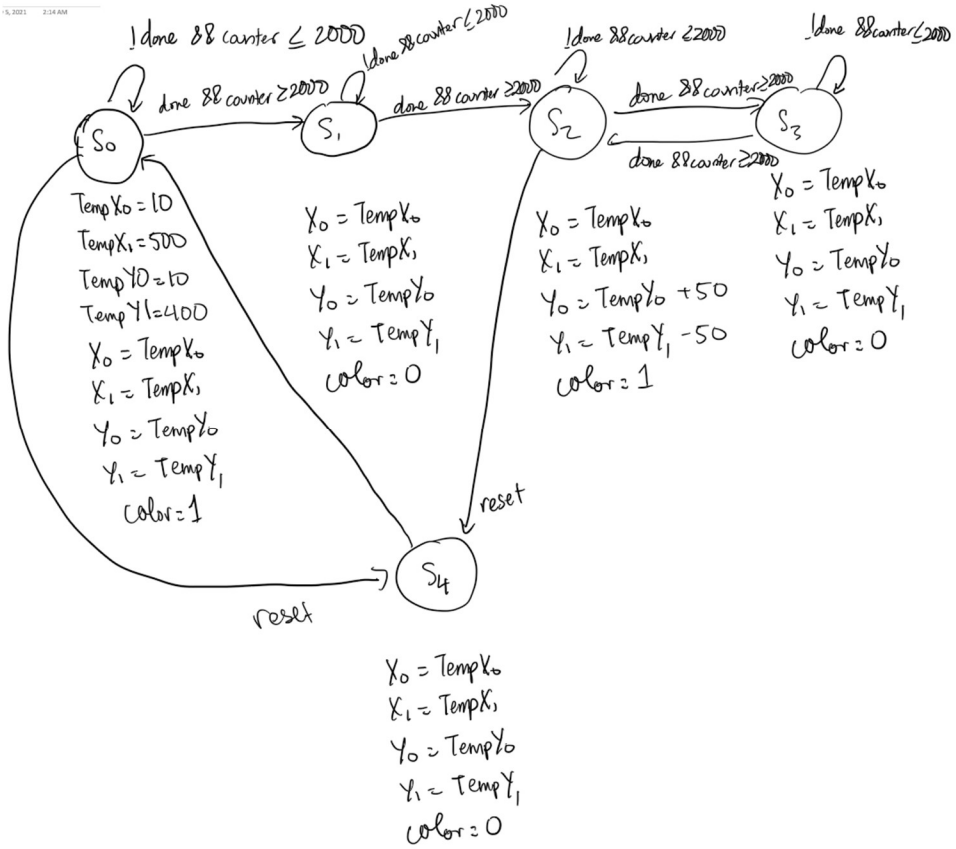

Figure 3: FSM for task 2

Results

Task 1:

For the first part, I tested if the line_drawer outputs the correct x and y values with the given starting and ending coordinates of (x0, y0) and (x1, y1), as you can see in figure 3, the x and y continuously outputs addresses for the VGA_framebuffer to write to and as the coordinates change, the x and y values change as well

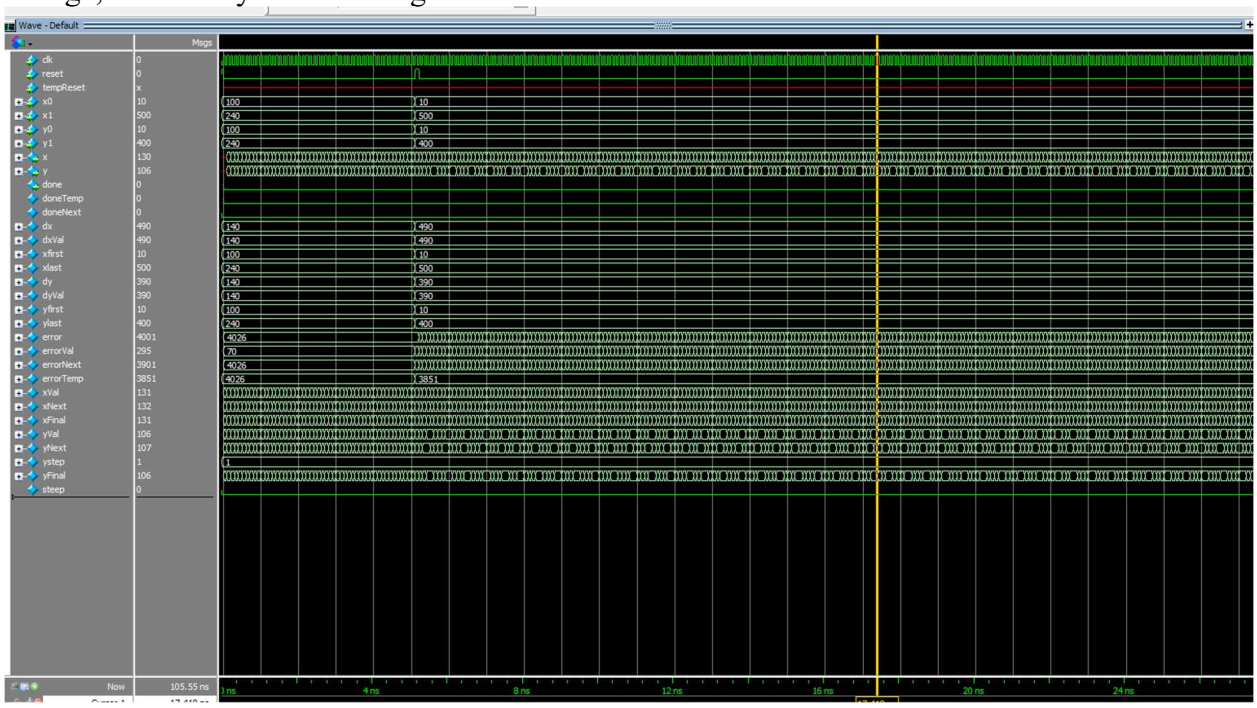

Figure 4: Waveform simulation for line_drawer for task 1

I also tested the simulation of the DE1_SoC module. However, the results were very different as when I ran the code on the labsland, it worked according to the line_drawer as the DE1_SoC only sends the output of line_drawer into the VGA_framebuffer.

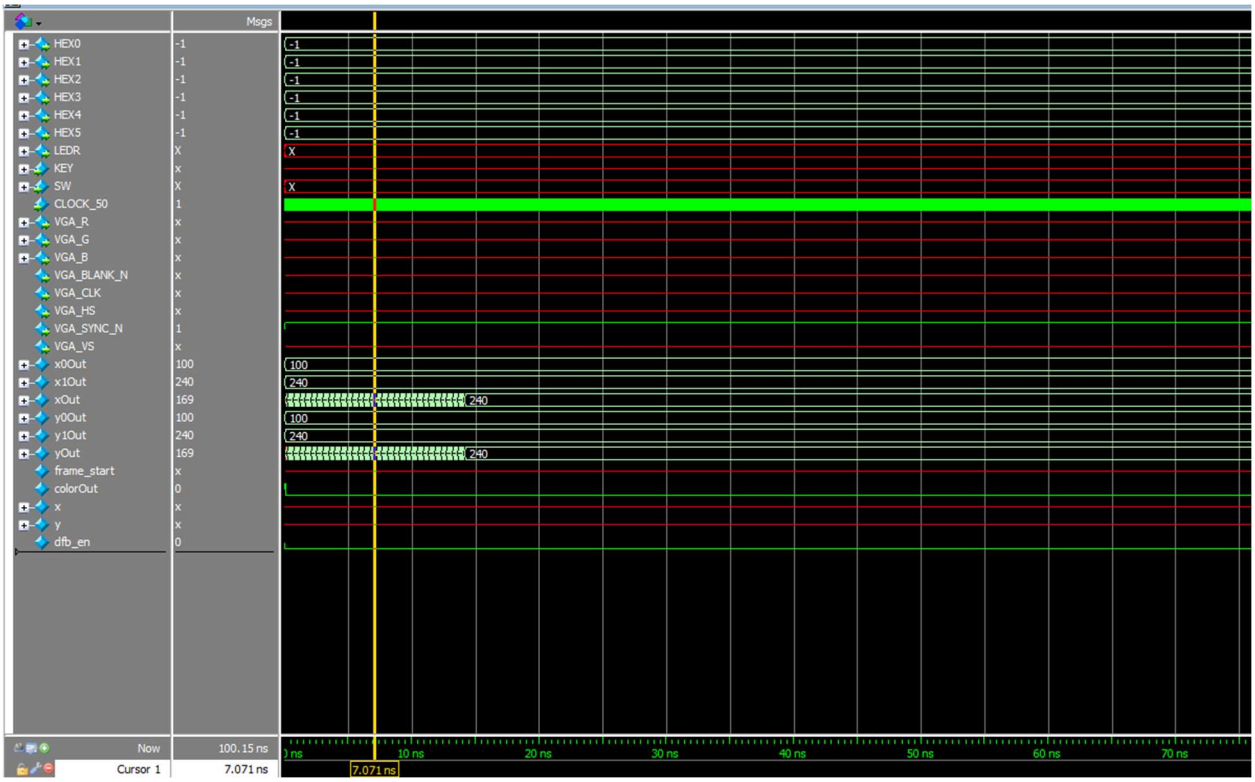

Figure 5: Waveform for DE1_SoC of task 1

Task 2:

This task replicated the same line_drawer module but with the addition of a temporary reset as an input, this is indicated as the line has been erased, forcing to go to the starting coordinates. This module also outputs if the line is done as in the current value is at the last coordinate.

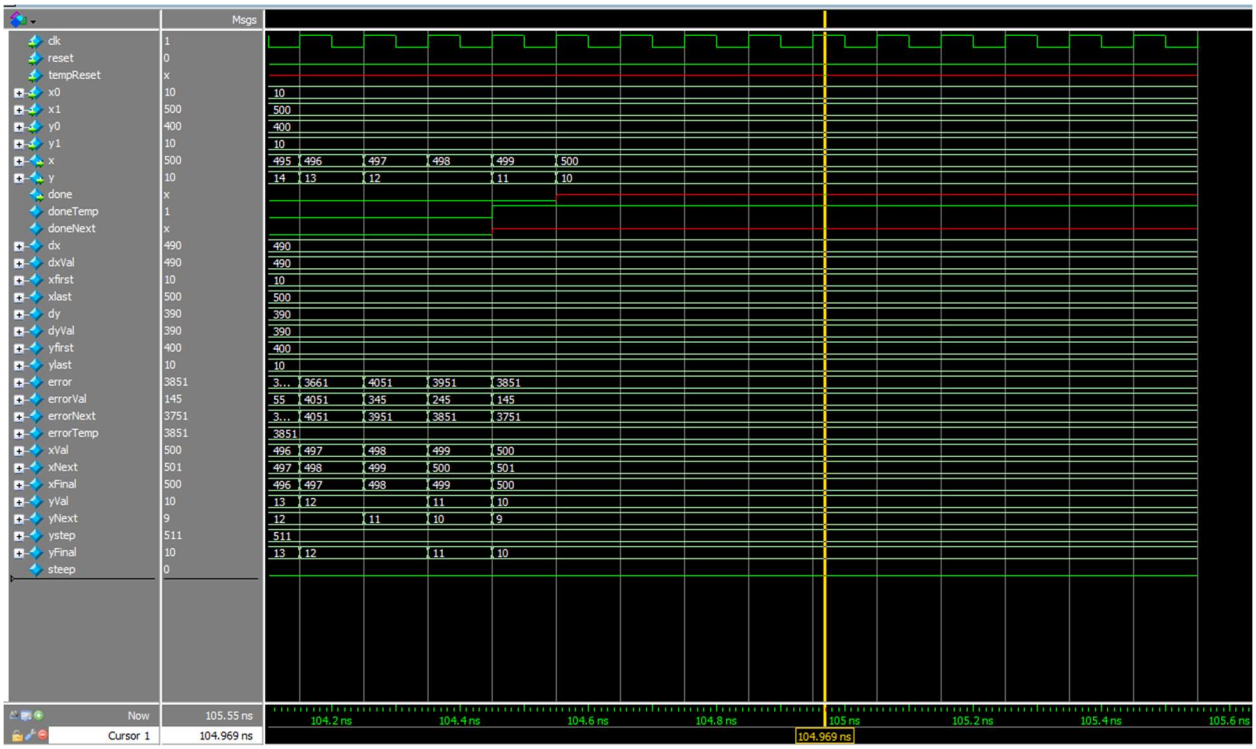

Figure 6: The waveform simulation generated by line_drawer for task 2

I also created an fsm and tested the output of it as it was able to transition between different states, incrementing values based on the location on the board.

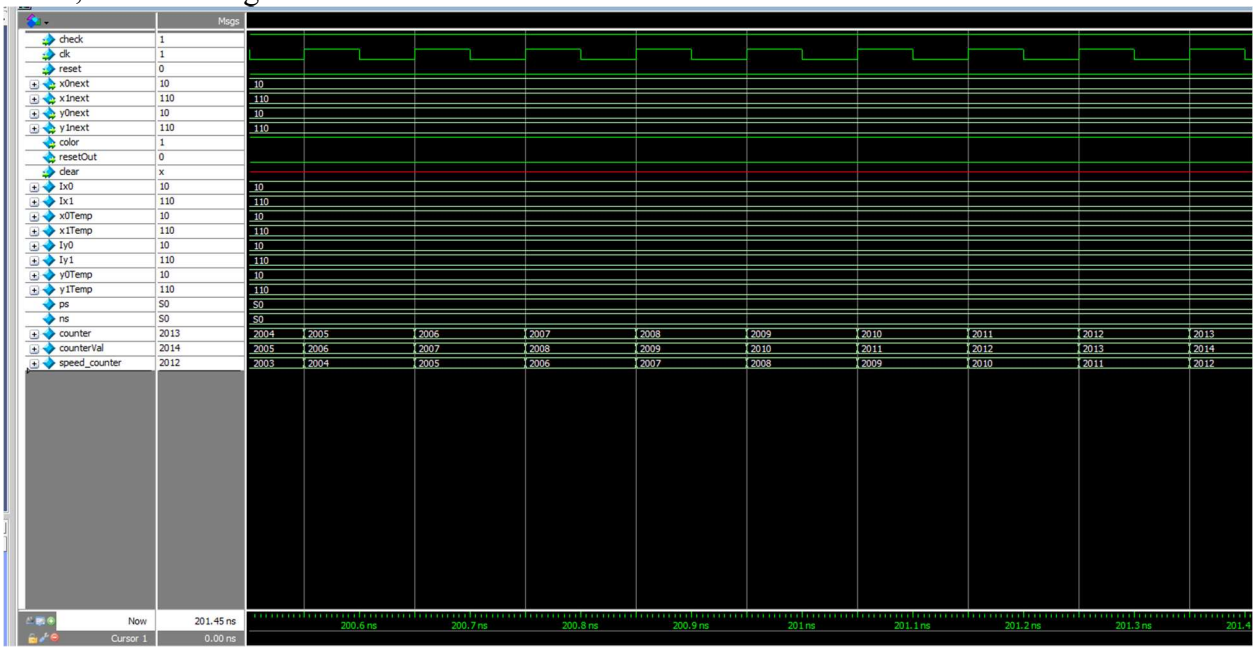

Figure 7: Waveform simulation generated by fsm for task 2

Once again, the DE1_SoC simulation was incorrect as seen in figure 7 and x and y aren't able to output anything. However, I have tested the fsm and the line_drawer and couldn't figure out why it wasn't loading the output of line_drawer into the VGA.

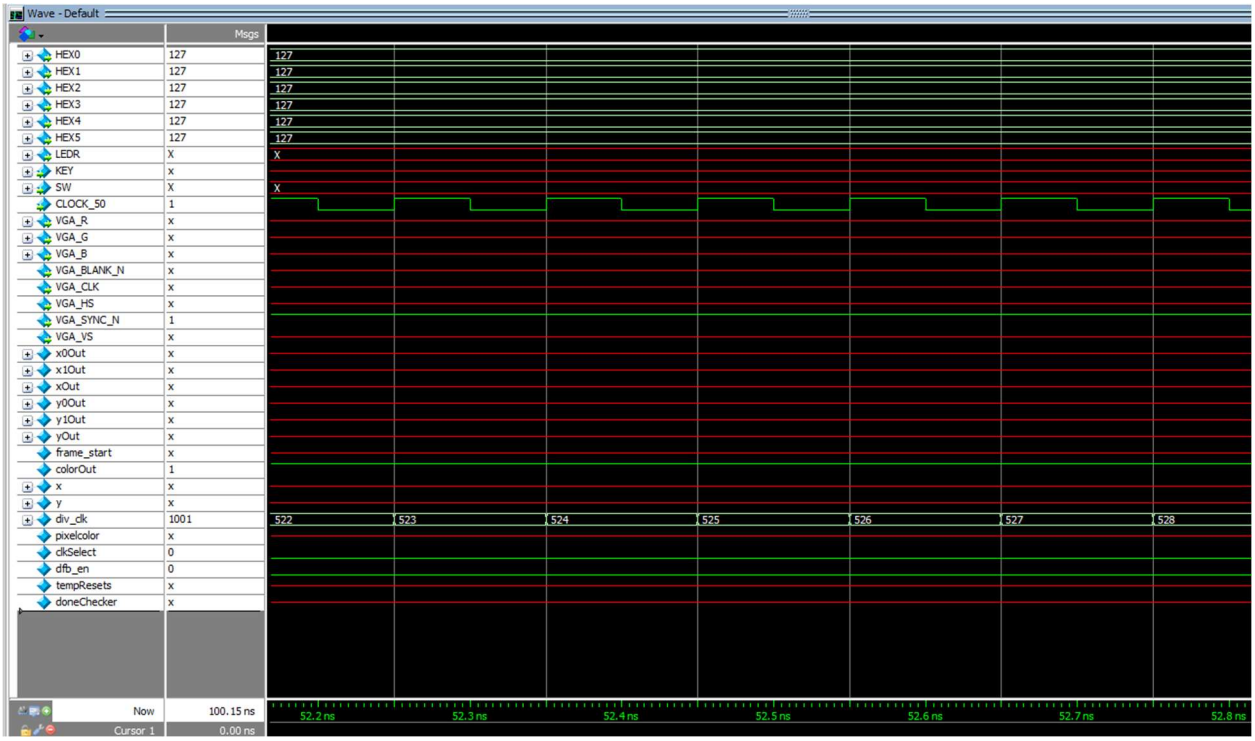

Figure 8: The waveform simulation generated by DE1_SoC for task 2

Final Product

The overarching goal of this project was to design a module that outputs a line given a starting and ending coordinate using Bresenham's algorithm and as well as animate the line by outputting the line and clearing it and incrementing starting and ending coordinates to continue the process and make it seem like it is animating through the screen. I wasn't able to accomplish task 2 as my DE1_SoC wouldn't transfer the outputs of line_drawer into the VGA_framebuffer. However, I was able to implement line_drawer and an fsm that controls the color that the VGA_framebuffer writes as I had the right idea but couldn't execute properly. Overall, I was able to learn a tremendous amount from this lab based on implementing C code in SystemVerilog and even though I wasn't able to animate the line, I am pleased with being able to implement line_drawer.

Appendix: SystemVerilog Code

1) Line_drawer.sv (task 1)

```
1   //Khoa Tran
2   //05/07/2021
3   //Lab 3, Task 1
4   //This module implements a controller for the VGA_framebuffer that draws a line from inputs
5   //of x0, y0, x1, y1 as two coordinates and outputs a series of x and y addresses of the
    VGA_framebuffer
6   //and output of done indicating the line is done printing. Taking inputs of clk and reset
    for
7   //making sure that values are given to the output at the posedge clk as well as having a
    functionality
8   //to reset or tempReset input from the fsm as an indication that the line has been cleared.
9   module line_drawer(clk, reset, x0, x1, y0, y1, x, y); //tempReset, x0, x1, y0, y1, x, y,
    done);
10     input logic clk, reset;
11     //input logic tempReset;
12
13     input logic [9:0] x0, x1;
14     input logic [8:0] y0, y1;
15
16     output logic [9:0] x;
17     output logic [8:0] y;
18     //output logic done;
19     //logic doneTemp, doneNext;
20
21
22     logic [9:0] dx, dxVal, xfirst, xlast;
23     logic [8:0] dy, dyVal, yfirst, ylast;
24     logic signed [11:0] error, errorVal, errorNext, errorTemp;
25     logic [9:0] xVal, xNext, xFinal;
26     logic [8:0] yVal, yNext, ystep, yFinal;
27     logic steep;
28
29     //combinational logic for setting temporary variables for outputs
30     always_comb begin
31        //calc of absolute value of delta x and delta y
32        dxVal = (x1 > x0) ? (x1 - x0) : (x0 - x1);
33        dyVal = (y1 > y0) ? (y1 - y0) : (y0 - y1);
34        //logic stating if line is steep or not
35        steep = (dyVal > dxVal);
36
37        //logic assigning xfirst, xlast, yfirst, ylast to x0,x1,y0,y1 based on conditions of
    steep and which value is bigger
38        xfirst = (~steep & (x0>x1)) ? x1 : (~steep & ~(x0>x1)) ? x0 : (steep & (y0>y1)) ? y1:
    y0;
39        xlast = (~steep & (x0>x1)) ? x0: (~steep & ~(x0>x1)) ? x1: (steep & (y0>y1)) ? y0: y1;
40        yfirst = (~steep & (x0>x1)) ? y1: (~steep & ~(x0>x1)) ? y0: (steep & (y0>y1)) ? x1: x0;
41        ylast = (~steep & (x0>x1)) ? y0: (~steep & ~(x0>x1)) ? y1: (steep & (y0>y1)) ? x0: x1;
42
43        //setting dx and dy based on steep or not
44        dx = steep ? dyVal : dxVal;
45        dy = steep ? dxVal : dyVal;
46
47        //initial error value
48        errorTemp = -(dx/2);
49
50        //logic of ystep based on if ylast is bigger than yfirst
51        ystep = (yfirst < ylast) ? 1: -1;
52
53        //logic of for loop, incrementing x values and y values based on error
54        xNext = xVal + 1'b1;
55        errorVal = error + dy;
56        yNext = (errorVal >= 0) ? yVal + ystep: yVal;
57        errorNext = (errorVal >= 0) ? errorVal - dx: errorVal;
58
59        //output yFinal and xFinal based on steep or not
60        yFinal = steep ? xVal : yVal;
61        xFinal = steep ? yVal : xVal;
62
63        //logic for outputting done based on conditions of being at the endpoint
64        //doneTemp = (xlast == xVal && ylast == yVal) ? 1 : 0;
65        //doneNext = (reset || tempReset) ? 0 : doneTemp;
66
67     end
68
69     //sequential logic setting next x and y values to the variables in combinational logic
    for changes and updates
```

```
70      always_ff @(posedge clk)
71         begin
72            if (reset) // || tempReset)
73               begin
74                  //on reset, set current values to the starting point based on combinational
    logic
75                  xVal <= xfirst;
76                  yVal <= yfirst;
77                  error <= errorTemp;
78                  //done <= doneNext;
79               end
80            else
81            begin
82                  //stopping condition
83                  if (xlast == xVal && ylast == yVal)
84                     begin
85                        xVal <= xVal;
86                        yVal <= yVal;
87                        error <= error;
88                        //done <= doneNext;
89                     end
90                  else
91                  //set next values to current
92                     begin
93                        xVal <= xNext;
94                        yVal <= yNext;
95                        error <= errorNext;
96                        //done <= doneNext;
97                     end
98               end
99            //setting output of x and y
100           x <= xFinal;
101           y <= yFinal;
102        end
103
104  endmodule
105
106  //Module testbench for line_drawer testing if outputs on x and y are correct
107  //along the posedge clk of inputs x0, y0, x1, y1 by passing a series of inputs
108  //and seeing if the output is correct
109  module line_drawer_testbench();
110     logic clk, reset;
111     //logic tempReset;
112
113     logic [9:0] x0, x1;
114     logic [8:0] y0, y1;
115
116     logic [9:0] x;
117     logic [8:0] y;
118     //logic done;
119
120     //line_drawer device under test
121     line_drawer dut(.clk, .reset, .x0, .x1, .y0, .y1, .x, .y);
122
123     parameter clock_period = 100;
124
125     //initial
126     initial begin
127        clk <= 0;
128        forever #(clock_period /2) clk <= ~clk;
129     end
130
131     initial begin
132        x0 <= 10'd100; x1 <= 10'd240;
133        y0 <= 9'd100; y1 <= 9'd240;
134        reset <= 1;                                    @(posedge clk);
135        reset <= 0;                                    @(posedge clk);
136
137        repeat(50)                                      @(posedge clk);
138
139        x0 <= 10'd10; x1 <= 10'd500;
140        y0 <= 9'd10; y1 <= 9'd400;
141        reset <= 1;                                    @(posedge clk);
142        reset <= 0;                                    @(posedge clk);
```



```
143
144        repeat(500)                                     @(posedge clk);
145
146        x0 <= 10'd10; x1 <= 10'd500;
147        y0 <= 9'd400; y1 <= 9'd10;
148        reset <= 1;                                    @(posedge clk);
149        reset <= 0;                                    @(posedge clk);
150
151        repeat(500)                                     @(posedge clk);
152
153        $stop;
154     end
155
156  endmodule
157
```

2) DE1_SoC.sv (task 1)

```
1   //Khoa Tran
2   //05/07/2021
3   //Lab 3, Task 2
4   //Module DE1_SoC that instantiates line_drawer and fsm for VGA_framebuffer
5   //as line_drawer is a module that takes inputs from fsm in order to output
6   //x and y values to the VGA_framebuffer. The fsm controller outputs color
7   //as well as resetOut for line drawer to start at initial value and VGA_framebuffer
8   //to set the color. The fsm module goes through a series of states to
9   //manipulate the line_drawer to animate a line going through the display in a circle
10  module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
11     VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);
12
13     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
14     output logic [9:0] LEDR;
15     input logic [3:0] KEY;
16     input logic [9:0] SW;
17
18     input CLOCK_50;
19     output [7:0] VGA_R;
20     output [7:0] VGA_G;
21     output [7:0] VGA_B;
22     output VGA_BLANK_N;
23     output VGA_CLK;
24     output VGA_HS;
25     output VGA_SYNC_N;
26     output VGA_VS;
27
28     assign HEX0 = '1;
29     assign HEX1 = '1;
30     assign HEX2 = '1;
31     assign HEX3 = '1;
32     assign HEX4 = '1;
33     assign HEX5 = '1;
34     assign LEDR = SW;
35
36     logic [9:0] x0Out, x1Out, xOut;
37     logic [9:0] y0Out, y1Out, yOut;
38     logic frame_start;
39     logic colorOut;
40     logic [9:0] x;
41     logic [8:0] y;
42
43    // logic [31:0] div_clk;
44     //logic pixelcolor;
45  // parameter whichClock = 10;
46  // clock_divider cdiv (.clock(CLOCK_50 ), .divided_clocks (div_clk ));
47  // logic clkSelect;
48  // assign clkSelect = div_clk[whichClock];
49
50     //////// DOUBLE_FRAME_BUFFER ////////
51     logic dfb_en;
52     assign dfb_en = 1'b0;
53     ////////////////////////////////////
54
55  // logic tempResets;
56  // logic doneChecker;
57
58     //instatiation of VGA_framebuffer colorOut
59     VGA_framebuffer fb(.clk(CLOCK_50), .rst(1'b0), .x(xOut), .y(yOut), .pixel_color(colorOut
    ), .pixel_write(1'b1), .dfb_en, .frame_start, .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .
    VGA_VS, .VGA_BLANK_N, .VGA_SYNC_N);
60
61     //instatiation of fsm for setting values of line_drawer in order to animate a line and
    send a color to VGA_framebuffer and coordinates for the line_drawer
62     //fsm controller (.check(doneChecker), .clear(SW[1]), .clk(clkSelect), .reset(SW[0]),
    .x0next(x0Out), .x1next(x1Out), .y0next(y0Out), .y1next(y1Out), .color(colorOut),
    .resetOut(tempResets));
63
64     // instantiation of line_drawer passing in x0, x1, y0, y1, and outputting a series of x
    and y outputs for VGA_framebuffer and done for fsm
65     line_drawer lines (.clk(CLOCK_50), .reset(SW[0]), .x0(x0Out), .x1(x1Out), .y0(y0Out), .y1
    (y1Out), .x(xOut), .y(yOut)); //.tempReset(tempResets), .x0(x0Out), .x1(x1Out), .y0(y0Out),
    .y1(y1Out), .x, .y, .done(doneChecker));
66
67
68     //uncomment below for task 1
```

```
69      // draw an arbitrary line
70      assign x0Out = 10'd100;
71      assign y0Out = 9'd100;
72      assign x1Out = 10'd240;
73      assign y1Out = 9'd240;
74
75      //initial colorOut = 1'b0;
76      always_ff @(posedge CLOCK_50)
77         begin
78            if (SW[0]) colorOut <= 1'b1;
79            else colorOut <= 1'b0;
80         end
81
82   endmodule
83
84   //Module test the output of the DE1_SoC module by running a sequence of inputs
85   //on the switches that controls the reset of fsm and line_drawer and seeing if
86   //the x and y output is correct along the posedge CLOCK_50 based on different states
87   //that the fsm outputs.
88   module DE1_SoC_testbench();
89      logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
90      logic [9:0] LEDR;
91      logic [3:0] KEY;
92      logic [9:0] SW;
93
94      logic CLOCK_50;
95      logic [7:0] VGA_R;
96      logic [7:0] VGA_G;
97      logic [7:0] VGA_B;
98      logic VGA_BLANK_N;
99      logic VGA_CLK;
100     logic VGA_HS;
101     logic VGA_SYNC_N;
102     logic VGA_VS;
103
104     //device under test of DE1_SoC
105     DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW, .CLOCK_50,
106        .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N, .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
107
108
109     parameter clock_period = 100;
110
111        initial begin
112           CLOCK_50 <= 0;
113           forever #(clock_period /2) CLOCK_50 <= ~CLOCK_50;
114
115        end
116
117     //initial
118     initial begin
119         SW[0] <= 1; @(posedge CLOCK_50);
120         SW[0] <= 0; @(posedge CLOCK_50);
121         repeat(1000)   @(posedge CLOCK_50);
122
123     $stop;
124  end
125
126  endmodule
```

3) Fsm.sv (task 2)

```
1   //Khoa Tran
2   //05/07/2021
3   //Lab 3, Task 2
4   //This module implements a fsm that manages outputs for the line_drawer as well as color
5   //for the VGA_framebuffer in order to animate a line going in a circle around the VGA
6   //display. This controller moves through a series of states setting outputs of color and
7   //coordinates in order to display a line, clear it, and move the line and continue the
    process.
8   //On posedge clk, next state is loaded into present state and output values as well.
9   //This also output a resetOut that is a temporary reset for the line_drawer to start off on
    the
10  //given coordinates
11  module fsm (check, clear, clk, reset, x0next, x1next, y0next, y1next, color, resetOut);
12     input logic check;
13     input logic clk, reset;
14     output logic [9:0] x0next, x1next;
15     output logic [9:0] y0next, y1next;
16     output logic color;
17     output logic resetOut;
18     input logic clear;
19
20     logic [9:0] Ix0, Ix1, x0Temp, x1Temp;
21     logic [9:0] Iy0, Iy1, y0Temp, y1Temp;
22
23     //states of S0, S1, S2, S3, and S4
24     enum {S0, S1, S2, S3, S4} ps, ns;
25
26     //assigning output of x0, y0, x1, y1 next values
27
28
29     //counter logic
30     logic [15:0] counter, counterVal, speed_counter;
31
32     //combinational logic for going through states
33     always_comb begin
34        x0Temp = Ix0;
35        x1Temp = Ix1;
36        y0Temp = Iy0;
37        y1Temp = Iy1;
38        case (ps)
39           S0:
40              //initial state, setting initial values
41              begin
42                 x0Temp = 10'd10;
43                 x1Temp = 10'd500;
44                 y0Temp = 10'd10;
45                 y1Temp = 10'd400;
46                 color = 1;
47                 //conditions for moving onto the next state based on
48                 //if the line is done and counter, holding the state
49                 if (check && speed_counter >= 2000)
50                    ns = S1;
51                 else if (clear)
52                     ns = S4;
53                 else
54                    ns = S0;
55              end
56           S1:
57              begin
58                 //second state, clearing the previous line
59                 x0Temp = Ix0;
60                 x1Temp = Ix1;
61                 y0Temp = Iy0;
62                 y1Temp = Iy1;
63                 color = 0;
64                 //conditions for moving onto the next state based on
65                 //if the line is done and counter, holding the state
66                 if (check && speed_counter >= 2000)
67                    ns = S2;
68                 else if (clear)
69                    ns = S4;
70                 else
71                    ns = S1;
72              end
73           S2:
74              begin
```

```
75                  //conditions for incrementing x0Temp, x1Temp, y0Temp, y1Temp
76                  //based on current value and outputting color based on condition
77                  if (Iy0 < 409 && Iy1 > 11 && Ix0 == 10 && Ix1 == 510) begin
78                     x0Temp = Ix0;
79                     x1Temp = Ix1;
80                     y0Temp = Iy0 + 50;
81                     y1Temp = Iy1 - 50;
82                     color = 1;
83                  end
84                  else if (Ix0 < 509 && Ix1 > 11 && Iy0 > 409 && Iy1 < 11) begin //(x0Temp <
      509 && x1Temp > 11 && y0Temp > 409 && y1Temp < 11) begin
85                     x0Temp = Ix0 + 50;
86                     x1Temp = Ix1 - 50;
87                     y0Temp = Iy0;
88                     y1Temp = Iy1;
89                     color = 1;
90                  end
91                  else begin
92                     x0Temp = Ix0;
93                     x1Temp = Ix1;
94                     y0Temp = Iy0;
95                     y1Temp = Iy1;
96                     color = 0;
97                  end
98                  //conditions for moving onto the next state based on
99                  //if the line is done and counter, holding the state
100                 if (check && speed_counter >= 2000)
101                    ns = S3;
102                 else if (clear)
103                    ns = S4;
104                 else
105                    ns = S2;
106              end
107           S3:
108              begin
109                 //clearing previous line
110                 x0Temp = Ix0;
111                 x1Temp = Ix1;
112                 y0Temp = Iy0;
113                 y1Temp = Iy1;
114                 color = 0;
115                 //conditions for moving onto the next state based on
116                 //if the line is done and counter, holding the state
117                 if (check && speed_counter >= 2000)
118                    ns = S2;
119                 else
120                    ns = S3;
121              end
122           S4:
123               begin
124                 //state of clearing the board
125                 x0Temp = Ix0;
126                 x1Temp = Ix1;
127                 y0Temp = Iy0;
128                 y1Temp = Iy1;
129                 color = 0;
130                 //conditions for moving onto the next state based on
131                 //if the line is done and counter, holding the state
132                 if (clear)
133                    ns = S4;
134                 else
135                    ns = S0;
136              end
137        endcase
138        x0next = Ix0;
139        x1next = Ix1;
140        y0next = Iy0;
141        y1next = Iy1;
142        //outputs of resetOut for line_drawer reset
143        resetOut = (counter == 0) ? 1: 0;
144        //counter value based on if present state is equal to next state
145        counterVal = (ns == ps) ? counter + 1: 0;
146
147     end
148
149     //sequential logic for incrementing speed_counter
```

```
150       always_ff @(posedge clk) begin
151          if (reset || resetOut) speed_counter <= 0;
152          else speed_counter <= speed_counter + 1;
153       end
154
155       //sequential logic for setting present state based on reset or other
156       always_ff @(posedge clk) begin
157          if (reset) begin
158             ps <= S0;
159             counter <= 0;
160             end
161          else
162             begin
163                ps <= ns;
164                Ix0 <= x0Temp;
165                Ix1 <= x1Temp;
166                Iy0 <= y0Temp;
167                Iy1 <= y1Temp;
168                counter <= counterVal;
169             end
170          end
171
172    endmodule
173
174    //module testing the outputs of fsm and state transition by setting
175    //a series of inputs on reset and check, seeing if color, x0next, x1next,
176    //y0next, y1next, and resetOut is correct
177    module fsm_testbench();
178       logic clk, reset;
179       logic check;
180       logic [9:0] x0next, x1next;
181       logic [9:0] y0next, y1next;
182       logic color;
183       logic resetOut;
184
185       //device under test
186       fsm dut(.check, .clk, .reset, .x0next, .x1next, .y0next, .y1next, .color, .resetOut);
187
188       parameter clock_period = 100;
189
190       initial begin
191          clk <= 0;
192          forever #(clock_period /2) clk <= ~clk;
193       end
194
195       //initial
196       initial begin
197          reset = 1; @(posedge clk);
198          reset = 0; check = 0; @(posedge clk);
199
200          repeat(10)       @(posedge clk);
201
202          check = 1; @(posedge clk);
203          check = 0; @(posedge clk);
204          check = 1; @(posedge clk);
205          repeat(2000)       @(posedge clk);
206
207
208          $stop;
209       end
210
211    endmodule
212
```

4) DE1_SoC.sv (task 2)

Date: May 07, 2021 DE1_SoC.sv Project: DE1_SoC

```
1   //Khoa Tran
2   //05/07/2021
3   //Lab 3, Task 2
4   //Module DE1_SoC that instantiates line_drawer and fsm for VGA_framebuffer
5   //as line_drawer is a module that takes inputs from fsm in order to output
6   //x and y values to the VGA_framebuffer. The fsm controller outputs color
7   //as well as resetOut for line drawer to start at initial value and VGA_framebuffer
8   //to set the color. The fsm module goes through a series of states to
9   //manipulate the line_drawer to animate a line going through the display in a circle
10  module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
11     VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);
12
13     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
14     output logic [9:0] LEDR;
15     input logic [3:0] KEY;
16     input logic [9:0] SW;
17
18     input CLOCK_50;
19     output [7:0] VGA_R;
20     output [7:0] VGA_G;
21     output [7:0] VGA_B;
22     output VGA_BLANK_N;
23     output VGA_CLK;
24     output VGA_HS;
25     output VGA_SYNC_N;
26     output VGA_VS;
27
28     assign HEX0 = '1;
29     assign HEX1 = '1;
30     assign HEX2 = '1;
31     assign HEX3 = '1;
32     assign HEX4 = '1;
33     assign HEX5 = '1;
34     assign LEDR = SW;
35
36     logic [9:0] x0Out, x1Out, xOut;
37     logic [9:0] y0Out, y1Out, yOut;
38     logic frame_start;
39     logic colorOut;
40     logic [9:0] x;
41     logic [8:0] y;
42
43     logic [31:0] div_clk;
44     logic pixelcolor;
45     parameter whichClock = 10;
46     clock_divider cdiv (.clock(CLOCK_50 ), .divided_clocks (div_clk ));
47     logic clkSelect;
48     assign clkSelect = div_clk[whichClock];
49
50     //////// DOUBLE_FRAME_BUFFER ////////
51     logic dfb_en;
52     assign dfb_en = 1'b0;
53     ////////////////////////////////////
54
55     logic tempResets;
56     logic doneChecker;
57
58     //instatiation of VGA_framebuffer colorOut
59     VGA_framebuffer fb(.clk(CLOCK_50), .rst(1'b0), .x(xOut), .y(yOut), .pixel_color(colorOut
   ), .pixel_write(1'b1), .dfb_en, .frame_start, .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .
   VGA_VS, .VGA_BLANK_N, .VGA_SYNC_N);
60
61     //instatiation of fsm for setting values of line_drawer in order to animate a line and
   send a color to VGA_framebuffer and coordinates for the line_drawer
62     fsm controller (.check(doneChecker), .clear(SW[1]), .clk(clkSelect), .reset(SW[0]), .
   x0next(x0Out), .x1next(x1Out), .y0next(y0Out), .y1next(y1Out), .color(colorOut), .resetOut(
   tempResets));
63
64     // instantiation of line_drawer passing in x0, x1, y0, y1, and outputting a series of x
   and y outputs for VGA_framebuffer and done for fsm
65     line_drawer lines (.clk(CLOCK_50), .reset(SW[0]), .tempReset(tempResets), .x0(x0Out), .x1
   (x1Out), .y0(y0Out), .y1(y1Out), .x, .y, .done(doneChecker));
66
67
```

```
80  //    end
81
82  endmodule
83
84  //Module test the output of the DE1_SoC module by running a sequence of inputs
85  //on the switches that controls the reset of fsm and line_drawer and seeing if
86  //the x and y output is correct along the posedge CLOCK_50 based on different states
87  //that the fsm outputs.
88  module DE1_SoC_testbench();
89     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
90     logic [9:0] LEDR;
91     logic [3:0] KEY;
92     logic [9:0] SW;
93
94     logic CLOCK_50;
95     logic [7:0] VGA_R;
96     logic [7:0] VGA_G;
97     logic [7:0] VGA_B;
98     logic VGA_BLANK_N;
99     logic VGA_CLK;
100    logic VGA_HS;
101    logic VGA_SYNC_N;
102    logic VGA_VS;
103
104    //device under test of DE1_SoC
105    DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .LEDR, .SW, .CLOCK_50,
106       .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N, .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
107
108
109    parameter clock_period = 100;
110
111       initial begin
112          CLOCK_50 <= 0;
113          forever #(clock_period /2) CLOCK_50 <= ~CLOCK_50;
114
115       end
116
117    //initial
118    initial begin
119        SW[0] <= 1; @(posedge CLOCK_50);
120        SW[0] <= 0; @(posedge CLOCK_50);
121        repeat(1000)   @(posedge CLOCK_50);
122
123    $stop;
124 end
125
126 endmodule
127
128 //clock_divider module has inputs of the clock, reset, and the
129 //32 bit divided_clock which allows to sequence through and
130 //output whenever the postive edge has been reached on the clock
131 // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ...
132 module clock_divider (clock, divided_clocks );
133    input logic clock;
134    output logic [31:0] divided_clocks = 0;
135
136    always_ff @(posedge clock) begin
137       divided_clocks <= divided_clocks + 1;
138    end
139 endmodule
140
```

5) Line_drawer.sv (task 2)

```
1   //Khoa Tran
2   //05/07/2021
3   //Lab 3, Task 1
4   //This module implements a controller for the VGA_framebuffer that draws a line from inputs
5   //of x0, y0, x1, y1 as two coordinates and outputs a series of x and y addresses of the
    VGA_framebuffer
6   //and output of done indicating the line is done printing. Taking inputs of clk and reset
    for
7   //making sure that values are given to the output at the posedge clk as well as having a
    functionality
8   //to reset or tempReset input from the fsm as an indication that the line has been cleared.
9   module line_drawer(clk, reset, tempReset, x0, x1, y0, y1, x, y, done);
10     input logic clk, reset;
11     input logic tempReset;
12
13     input logic [9:0] x0, x1;
14     input logic [8:0] y0, y1;
15
16     output logic [9:0] x;
17     output logic [8:0] y;
18     output logic done;
19     logic doneTemp, doneNext;
20
21
22     logic [9:0] dx, dxVal, xfirst, xlast;
23     logic [8:0] dy, dyVal, yfirst, ylast;
24     logic signed [11:0] error, errorVal, errorNext, errorTemp;
25     logic [9:0] xVal, xNext, xFinal;
26     logic [8:0] yVal, yNext, ystep, yFinal;
27     logic steep;
28
29     //combinational logic for setting temporary variables for outputs
30     always_comb begin
31        //calc of absolute value of delta x and delta y
32        dxVal = (x1 > x0) ? (x1 - x0) : (x0 - x1);
33        dyVal = (y1 > y0) ? (y1 - y0) : (y0 - y1);
34        //logic stating if line is steep or not
35        steep = (dyVal > dxVal);
36
37        //logic assigning xfirst, xlast, yfirst, ylast to x0,x1,y0,y1 based on conditions of
    steep and which value is bigger
38        xfirst = (~steep & (x0>x1)) ? x1 : (~steep & ~(x0>x1)) ? x0 : (steep & (y0>y1)) ? y1:
    y0;
39        xlast = (~steep & (x0>x1)) ? x0: (~steep & ~(x0>x1)) ? x1: (steep & (y0>y1)) ? y0: y1;
40        yfirst = (~steep & (x0>x1)) ? y1: (~steep & ~(x0>x1)) ? y0: (steep & (y0>y1)) ? x1: x0;
41        ylast = (~steep & (x0>x1)) ? y0: (~steep & ~(x0>x1)) ? y1: (steep & (y0>y1)) ? x0: x1;
42
43        //setting dx and dy based on steep or not
44        dx = steep ? dyVal : dxVal;
45        dy = steep ? dxVal : dyVal;
46
47        //initial error value
48        errorTemp = -(dx/2);
49
50        //logic of ystep based on if ylast is bigger than yfirst
51        ystep = (yfirst < ylast) ? 1: -1;
52
53        //logic of for loop, incrementing x values and y values based on error
54        xNext = xVal + 1'b1;
55        errorVal = error + dy;
56        yNext = (errorVal >= 0) ? yVal + ystep: yVal;
57        errorNext = (errorVal >= 0) ? errorVal - dx: errorVal;
58
59        //output yFinal and xFinal based on steep or not
60        yFinal = steep ? xVal : yVal;
61        xFinal = steep ? yVal : xVal;
62
63        //logic for outputting done based on conditions of being at the endpoint
64        doneTemp = (xlast == xVal && ylast == yVal) ? 1 : 0;
65        doneNext = (reset || tempReset) ? 0 : doneTemp;
66
67     end
68
69     //sequential logic setting next x and y values to the variables in combinational logic
    for changes and updates
70     always_ff @(posedge clk)
```

```
71          begin
72             if (reset || tempReset)
73                begin
74                   //on reset, set current values to the starting point based on combinational
       logic
75                   xVal <= xfirst;
76                   yVal <= yfirst;
77                   error <= errorTemp;
78                   done <= doneNext;
79                end
80             else
81             begin
82                   //stopping condition
83                   if (xlast == xVal && ylast == yVal)
84                      begin
85                         xVal <= xVal;
86                         yVal <= yVal;
87                         error <= error;
88                         done <= doneNext;
89                      end
90                   else
91                   //set next values to current
92                      begin
93                         xVal <= xNext;
94                         yVal <= yNext;
95                         error <= errorNext;
96                         done <= doneNext;
97                      end
98                end
99             //setting output of x and y
100            x <= xFinal;
101            y <= yFinal;
102         end
103
104   endmodule
105
106   //Module testbench for line_drawer testing if outputs on x and y are correct
107   //along the posedge clk of inputs x0, y0, x1, y1 by passing a series of inputs
108   //and seeing if the output is correct
109   module line_drawer_testbench();
110      logic clk, reset;
111      logic tempReset;
112
113      logic [9:0] x0, x1;
114      logic [8:0] y0, y1;
115
116      logic [9:0] x;
117      logic [8:0] y;
118      logic done;
119
120      //line_drawer device under test
121      line_drawer dut(.clk, .reset, .tempReset, .x0, .x1, .y0, .y1, .x, .y, .done);
122
123      parameter clock_period = 100;
124
125      //initial
126      initial begin
127         clk <= 0;
128         forever #(clock_period /2) clk <= ~clk;
129      end
130
131      initial begin
132         x0 <= 10'd100; x1 <= 10'd240;
133         y0 <= 9'd100; y1 <= 9'd240;
134         reset <= 1;                                   @(posedge clk);
135         reset <= 0;                                   @(posedge clk);
136
137         repeat(50)                                     @(posedge clk);
138
139         x0 <= 10'd10; x1 <= 10'd500;
140         y0 <= 9'd10; y1 <= 9'd400;
141         reset <= 1;                                   @(posedge clk);
142         reset <= 0;                                   @(posedge clk);
143
```



```
144         repeat(500)                                    @(posedge clk);
145
146         x0 <= 10'd10; x1 <= 10'd500;
147         y0 <= 9'd400; y1 <= 9'd10;
148         reset <= 1;                                   @(posedge clk);
149         reset <= 0;                                   @(posedge clk);
150
151         repeat(500)                                    @(posedge clk);
152
153         $stop;
154      end
155
156   endmodule
157
158
```