UW Student: Brian Chen

EE 371

February 7, 2025

Lab3 Report

I. Procedure

Task 1:

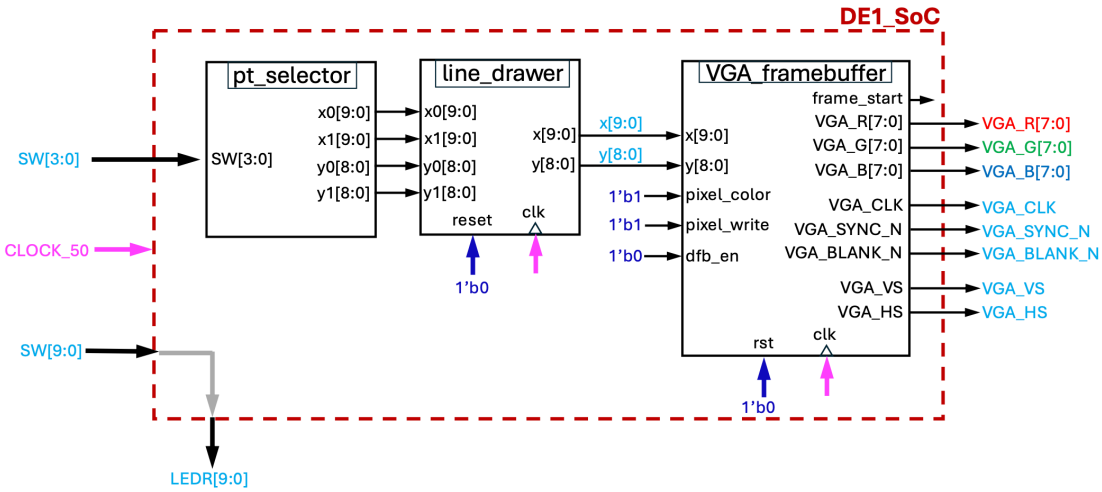

Figure 1: Block diagram of the top module of Task 1.

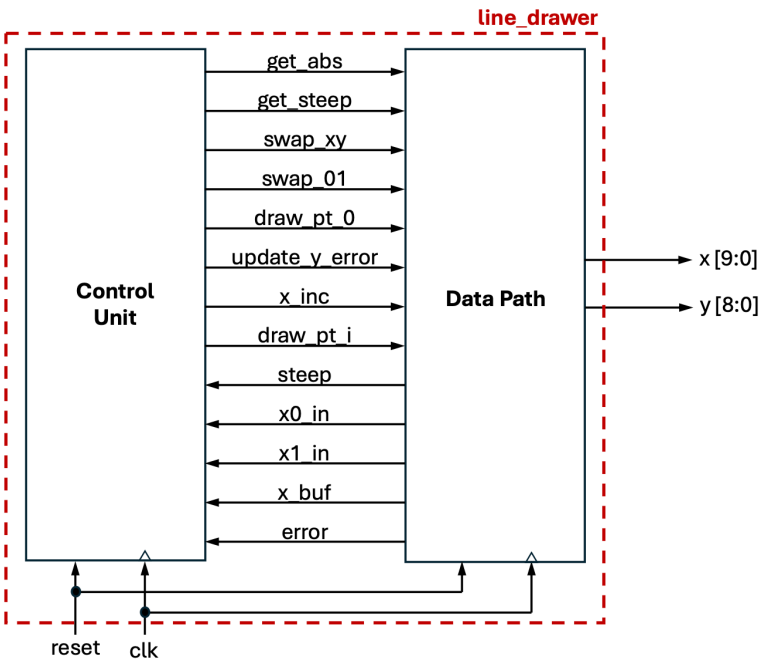

Figure 2: Block diagram of "line_drawer" module of Task 1.

Figure 1 shows the block diagram of the design in Task 1. This diagram contains two main modules which are “line_drawer” module and “VGA_framebuffer” module. I have to design “line_drawer” module which can generate the coordinate (x, y) of the points on an arbitrary straight line specified by a starting point (x_0, y_0) and an end point (x_1, y_1) no matter what the slope of this straight line is. I partitioned “line_drawer” module into two portions which are the “Control Unit” and the “Data Path” as shown in Figure 2.

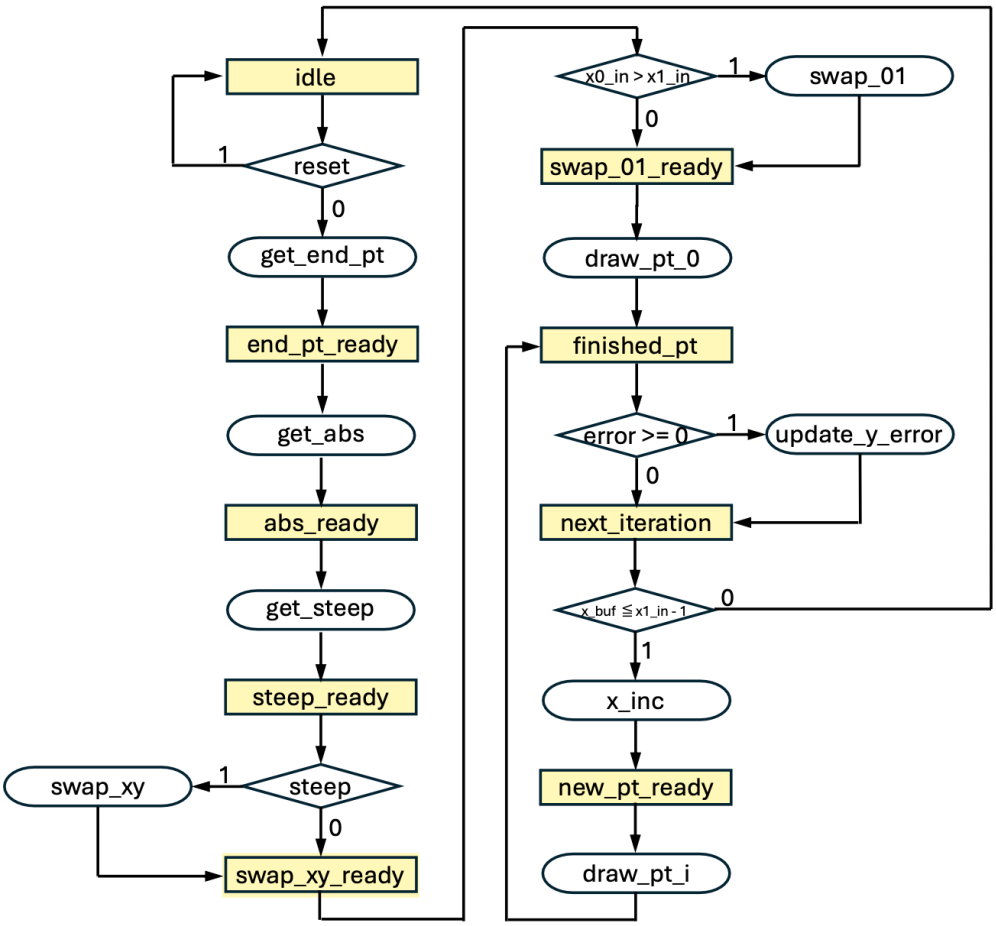

Figure 3: ASM chart of the “line_drawer” module.

In order to design the “line_drawer” module, I studied the pseudo-code for the line-drawing algorithm developed by Bersenham. I analyzed the timing relationships between the statements in the pseudo-code, and I derived the ASM chart of the control unit according to the pseudo-code. As shown in Figure 3, the ASM has 9 states, 5 decision-making nodes, and 9 outputs. These 9 output signals generated from the control unit will be sent to the data-path unit for guiding how the data be transferred between the registers such that the expected function of the algorithm described in the

pseudo-code will perform well. Then, I described the ASM chart and the related data path in SystemVerilog for running the simulation on Modelsim-Altera and verifying the functions on LabsLand.

Task 2:

The block diagram of the design in Task 2 is the same as that of Task 1. In order to animate the lines (or pixels) on the screen, I created a procedure assignment containing a "case" statement describing several straight lines by specifying the coordinates of the starting point $(x_0, y_0)$and the end point (x_1, y_1) for each straight line. Each straight line is specified by two "case items": one case item is for drawing each pixel with white color by setting the parameter "pixel_color" to "1'b1", and the other case item is for drawing each pixel with black color by setting the parameter "pixel_color" to "1'b0". The parameter "pixel_color" was hard-coded and sent to the "VGA_framebuffer" module. In additional, I created a counter which generates the signals for screening the case items (straight lines) in the case statement sequentially and cyclically. In this design, I assigned the switches (SW[3:0]) on DE1-SoC board to send a 4-bit signal for meeting the case expression such that the corresponding case item is activated.

II. Result

Task 1:

Simulation Result with the Testbench for "DE1_SoC.sv":

I wrote the testbench to test the top module in Task 1. In this testbench, there is a clock generator. Because there are several hard-coded straight lines in the device-under-test (dut), I applied the input signals to the device-under-test via the switches (SW[3:0]) with 4-bit signals. In order to verify the robustness of this design, I tried the straight lines with various directions from the starting points to the end points. I also tried the straight lines with different slopes including positive, negative, steep, and graduate. The term "steep straight line" means the absolute value of the slope is greater than 1, and the term "graduate straight line" means the absolute value of the slope is less than 1. Figures 4 to 13 are the simulation results of the straight lines specified by various starting points and end points, as indicated in the figures. The simulation results showed

this design performs well.

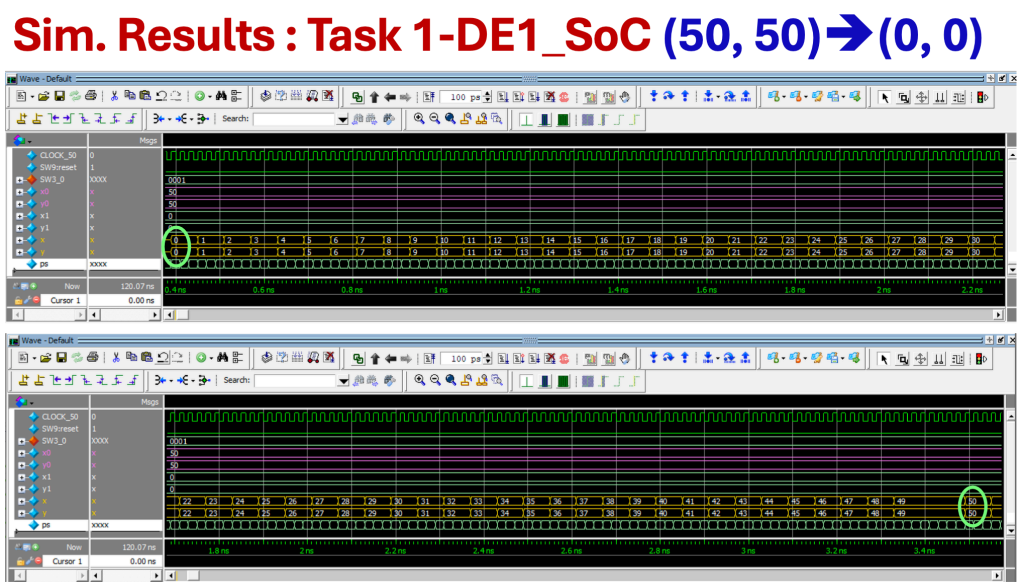

Figure 4: Simulation results of the design “DE1_SoC” in Task 1. (slope = 1)

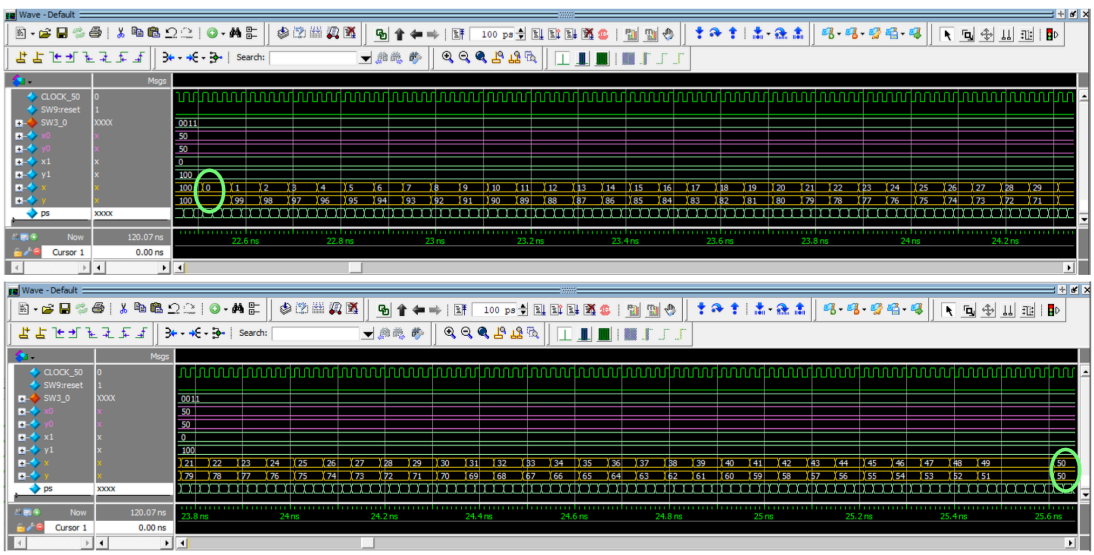

Figure 5: Simulation results of the design “DE1_SoC” in Task 1. (slope = 1)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(100, 0)

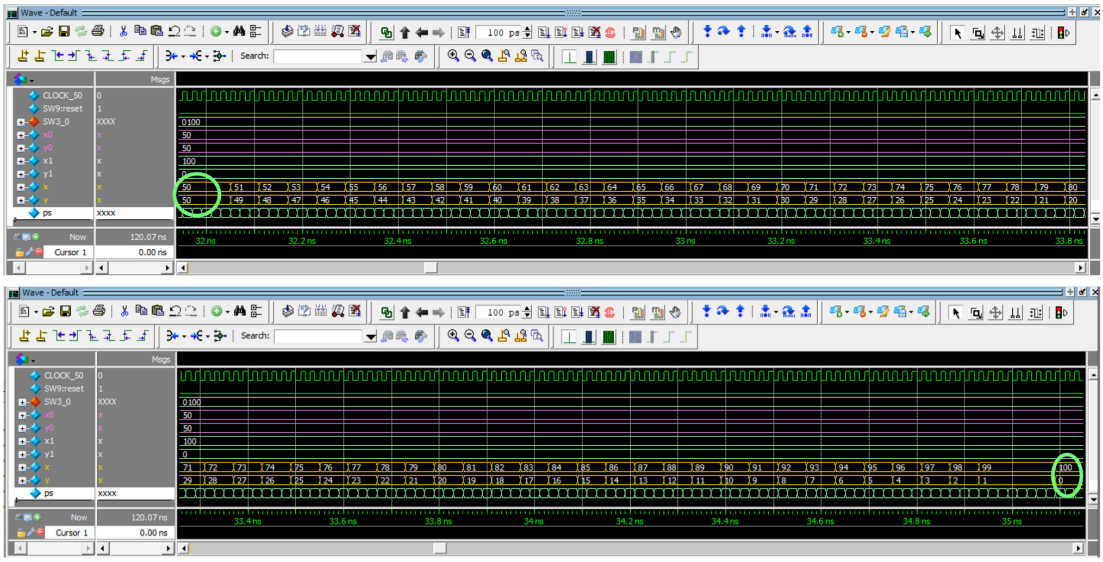

Figure 6: Simulation results of the design “DE1_SoC” in Task 1. (slope $= -1$)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(70, 0)

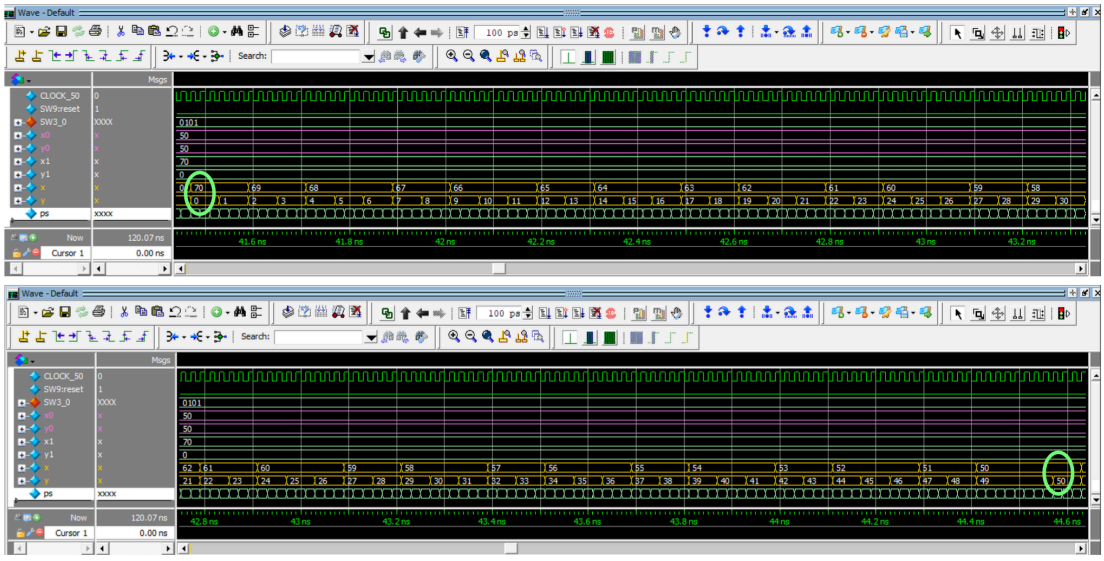

Figure 7: Simulation results of the design “DE1_SoC” in Task 1. (slope $= -2.5$)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(30, 100)

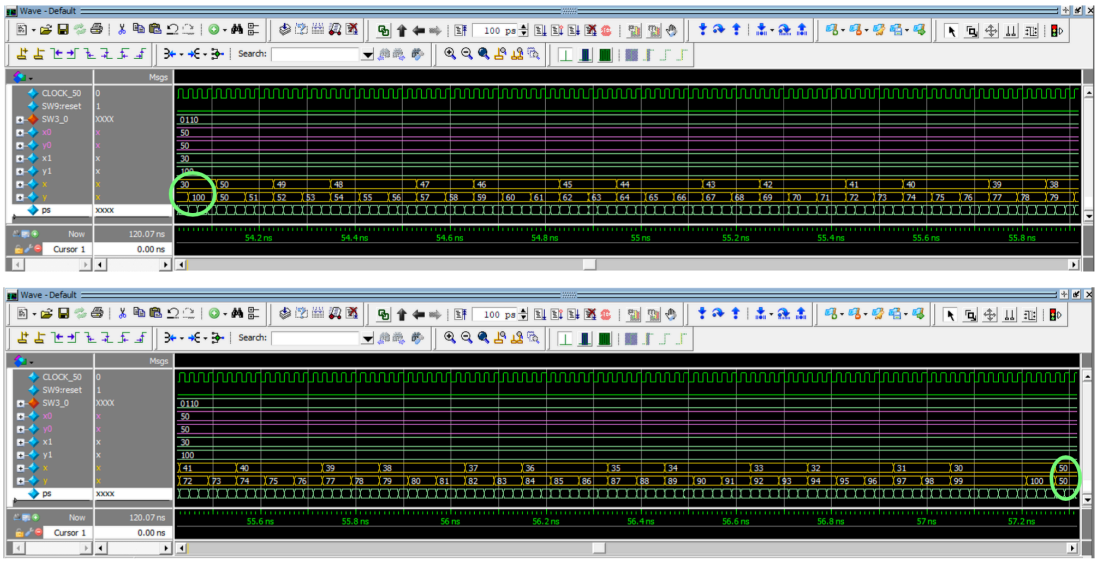

Figure 8: Simulation results of the design “DE1_SoC” in Task 1. (slope = −2.5)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(30, 0)

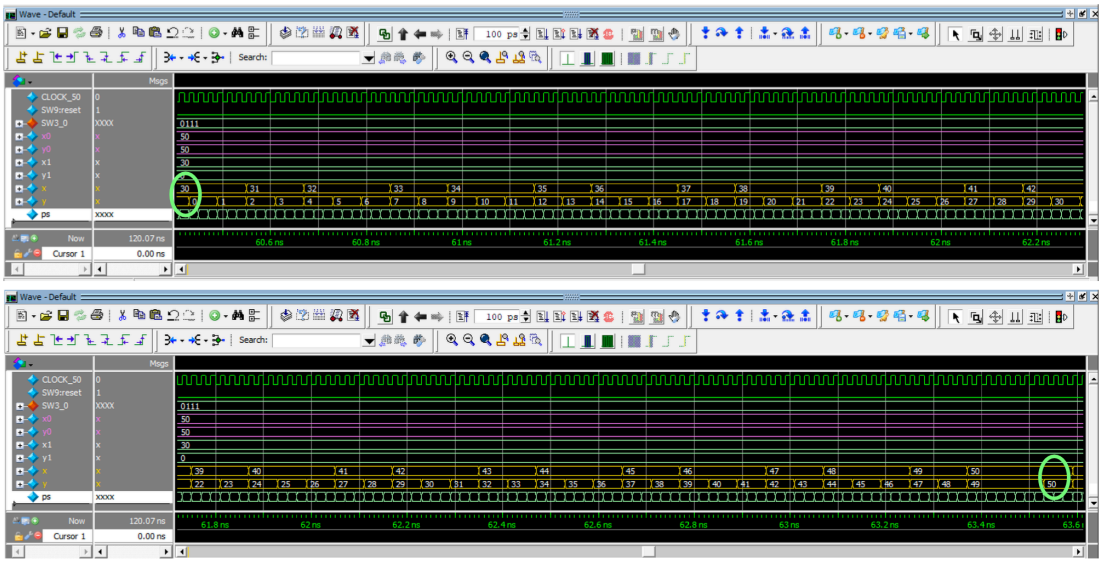

Figure 9: Simulation results of the design “DE1_SoC” in Task 1. (slope = 2.5)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(0, 35)

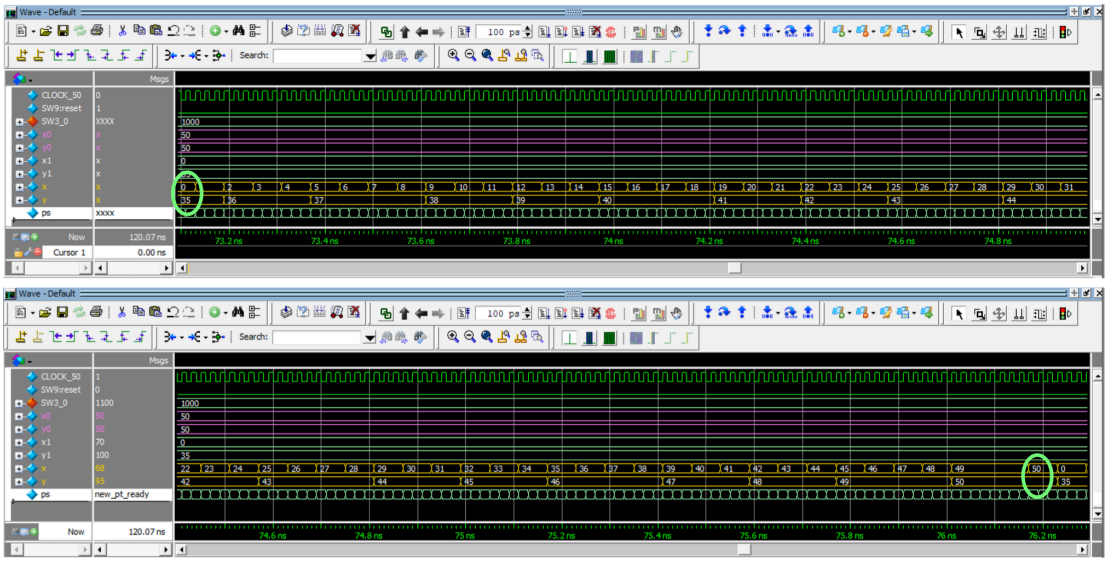

Figure 10: Simulation results of the design “DE1_SoC” in Task 1. (slope = 0.3)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(0, 70)

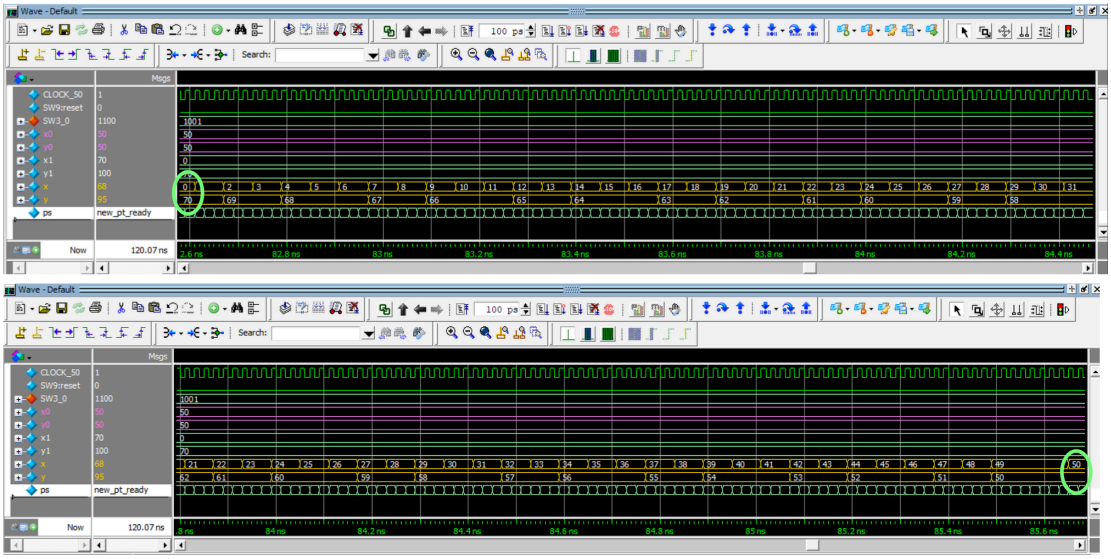

Figure 11: Simulation results of the design “DE1_SoC” in Task 1. (slope = 0.4)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(100, 70)

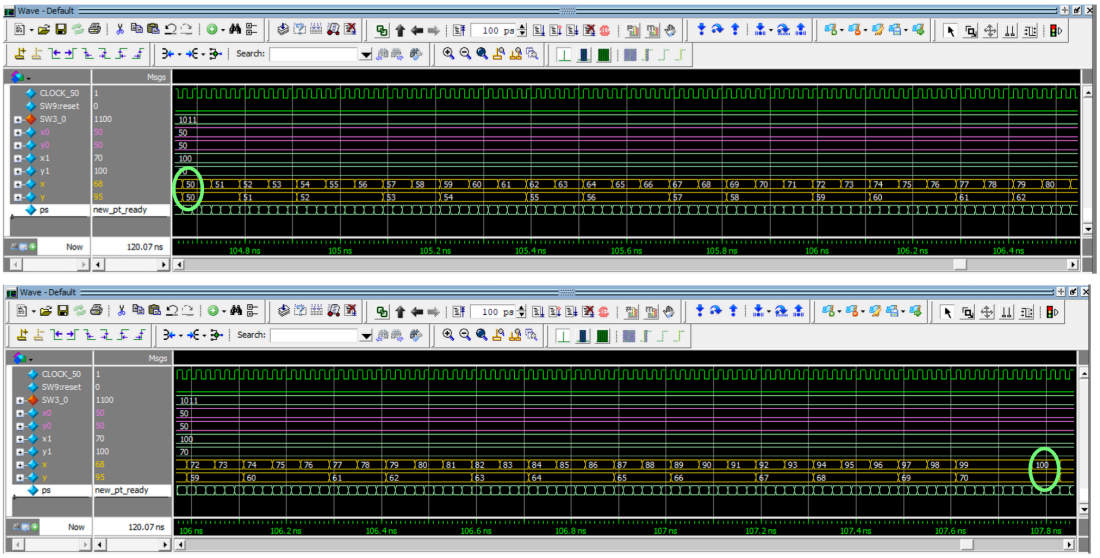

Figure 12: Simulation results of the design “DE1_SoC” in Task 1. (slope = 0.4)

Sim. Results : Task 1-DE1_SoC (50, 50)➔(70, 100)

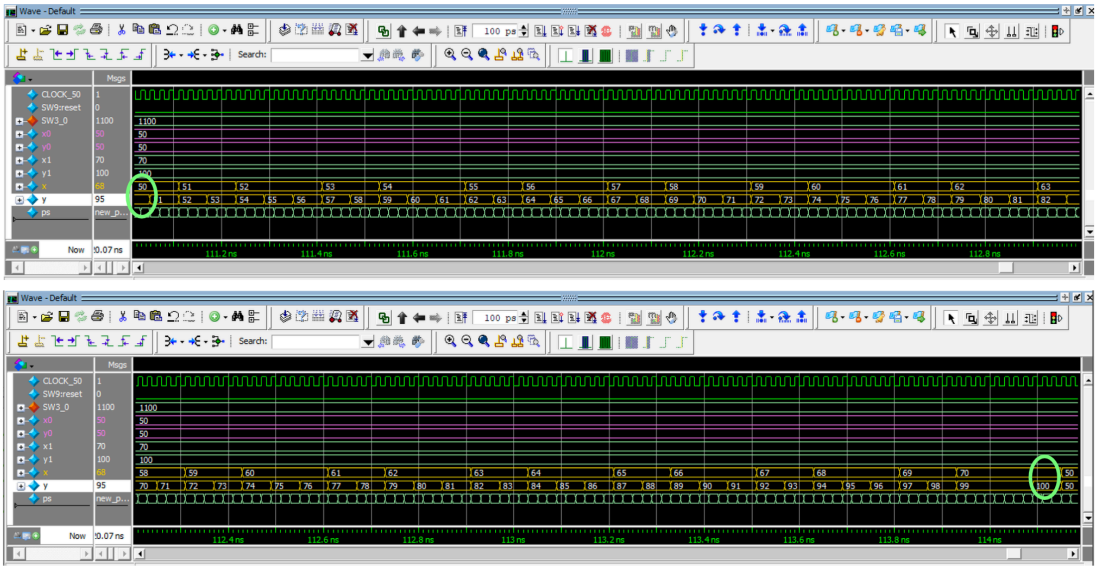

Figure 13: Simulation results of the design “DE1_SoC” in Task 1. (slope = 2.5)

Simulation Result with the Testbench for "line_drawer.sv":

In fact, I wrote the testbench to test the "line_drawer" module in Task 1 before testing the DE1-SoC module. In this testbench, there is a clock generator. I applied the input signals to the device-under-test by assigning the coordinate information of the starting point and the end point with the 4-bit signals in the testbench. In order to verify the robustness of this design, I tried the straight lines with various directions from the starting points to the end points. I also tried the straight lines with different slopes including positive, negative, steep, and graduate. Figures 14 to 18 are the simulation results of the straight lines specified by various starting points and end points, as indicated in the figures. The simulation results showed this design performs well.

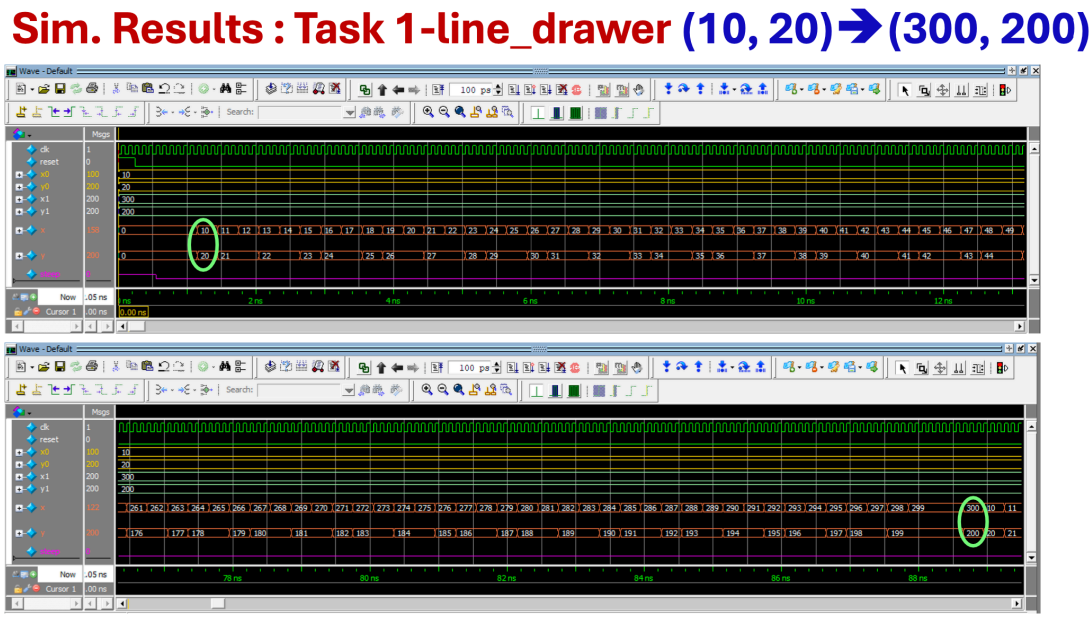

Figure 14: Simulation results of the design "line_drawer" in Task 1. (slope = 0.62)

Sim. Results : Task 1-line_drawer (10, 20)➔(100, 200)

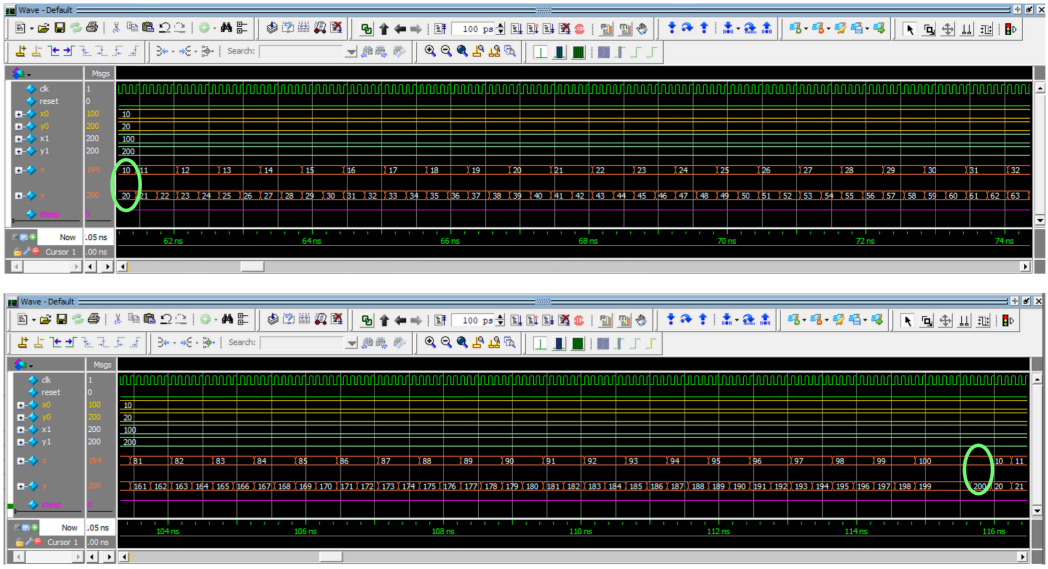

Figure 15: Simulation results of the design “line_drawer” in Task 1. (slope = 2)

Sim. Results : Task 1-line_drawer (10, 20)➔(10, 200)

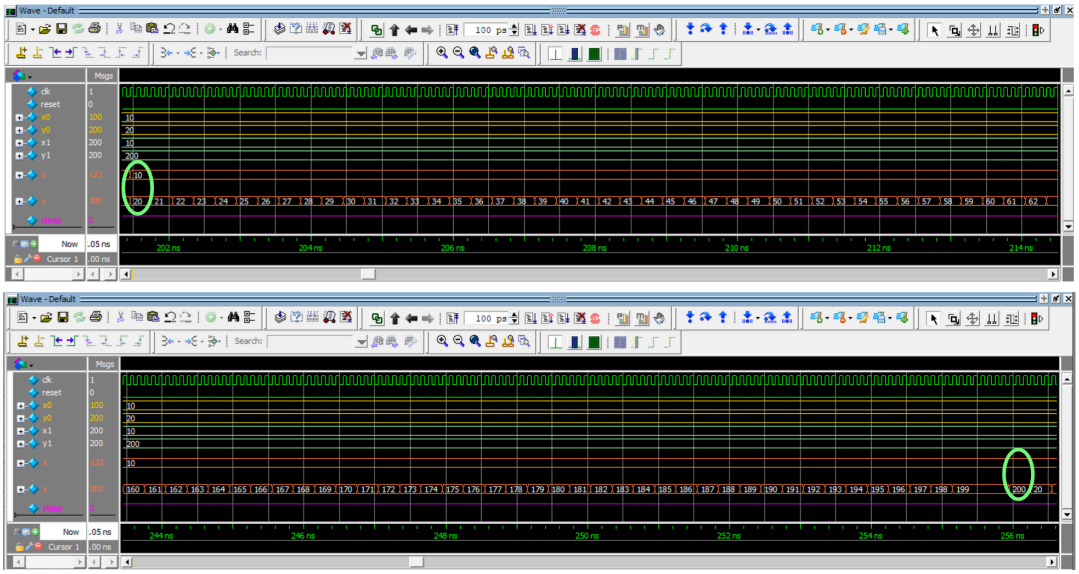

Figure 16: Simulation results of the design “line_drawer” in Task 1. (slope = ∞)

Sim. Results : Task 1-line_drawer (300, 20)➔(10, 200)

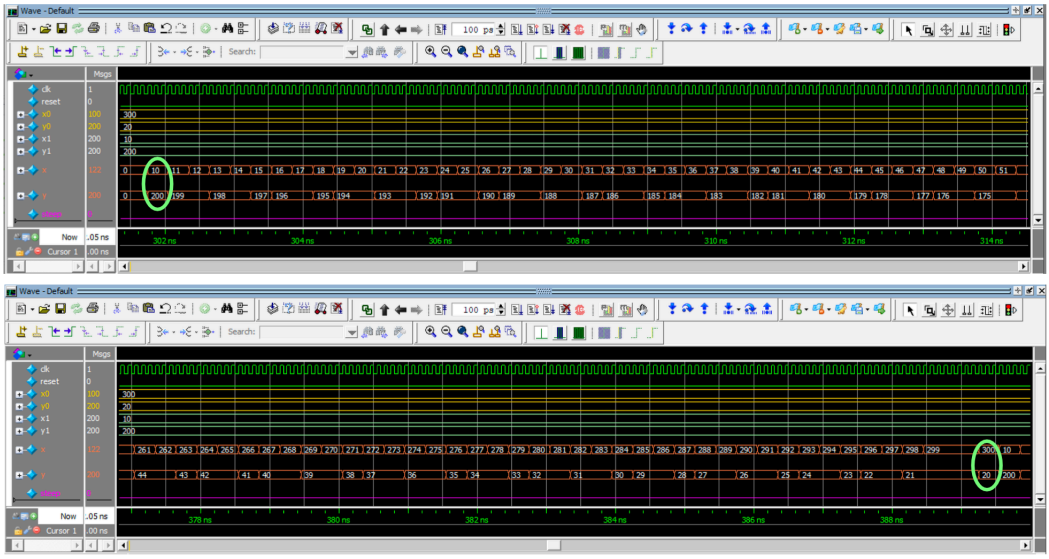

Figure 17: Simulation results of the design “line_drawer” in Task 1. (slope = −0.62)

Sim. Results : Task 1-line_drawer (100, 200)➔(200, 200)

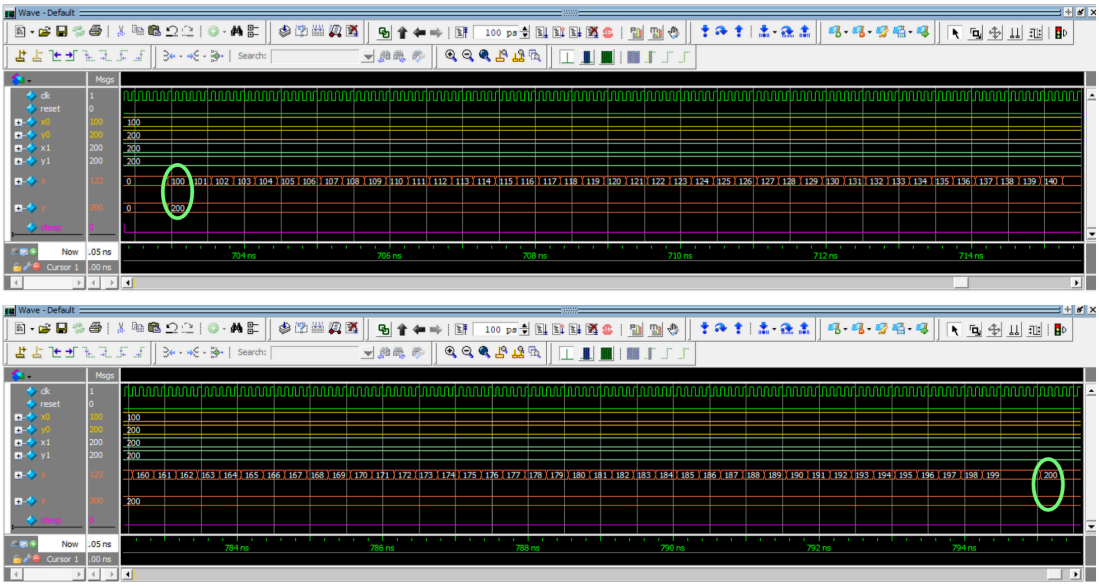

Figure 18: Simulation results of the design “line_drawer” in Task 1. (slope = 0)

Task 2:

Simulation Result with the Testbench for "DE1_SoC.sv":

I wrote the testbench to test the top module in Task 2. The simulation of the design in Task 2 includes two portions: drawing pixels in "white" and clearing pixels in "black". Figures 19 and 20 are the simulation results of the design "DE1-SoC" with the parameter "pixel_color" set to "1'b1". Figure 21 is the simulation results of the design "DE1-SoC" with the parameter "pixel_color" set to "1'b0". These simulation results showed that this the design in Task 2 performs well.

Sim. Results : Task 2-DE1_SoC (0, 200)➔(50, 0), pixel_color = 1

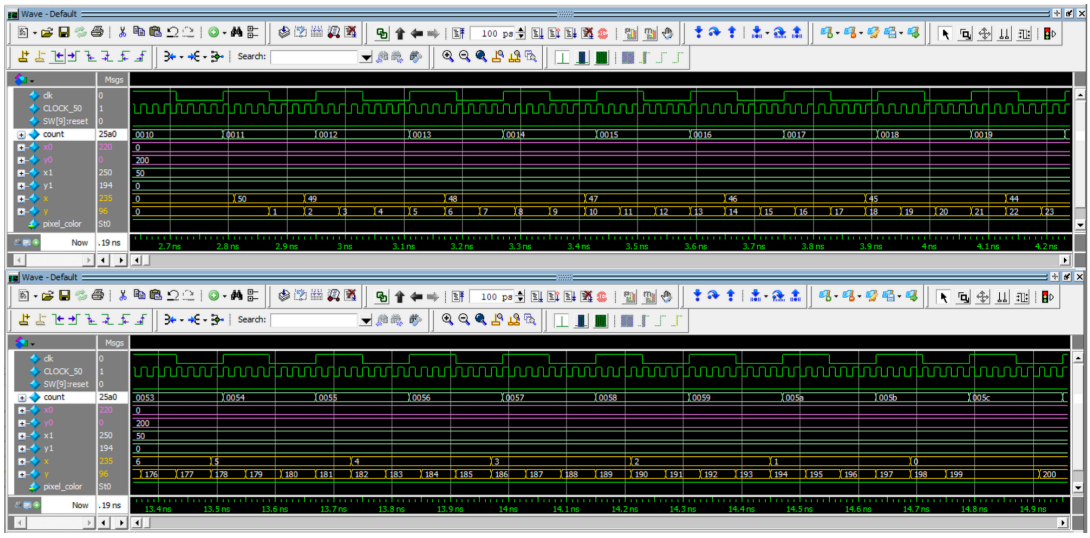

Figure 19: Simulation results of "DE1_SoC" in Task 2 with the parameter "pixel_color" set to "1'b1". (slope $= -0.25$)

Sim. Results : Task 2-DE1_SoC (50, 0)➔(120, 194), pixel_color = 1

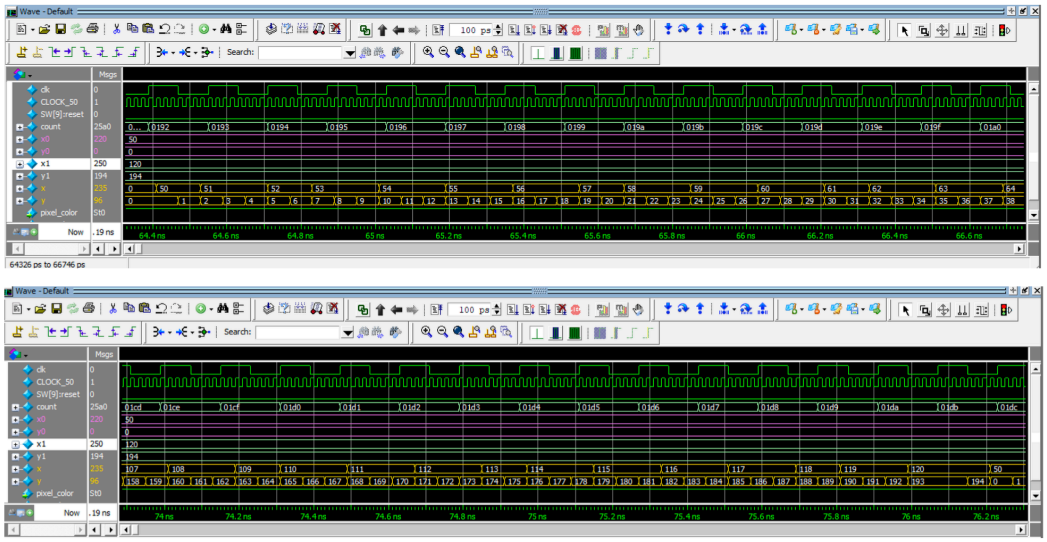

Figure 20: Simulation results of “DE1_SoC” in Task 2 with the parameter “pixel_color” set to “1’b1”. (slope = 2.8)

Sim. Results : Task 2-DE1_SoC (50, 0)➔(120, 194), pixel_color = 0

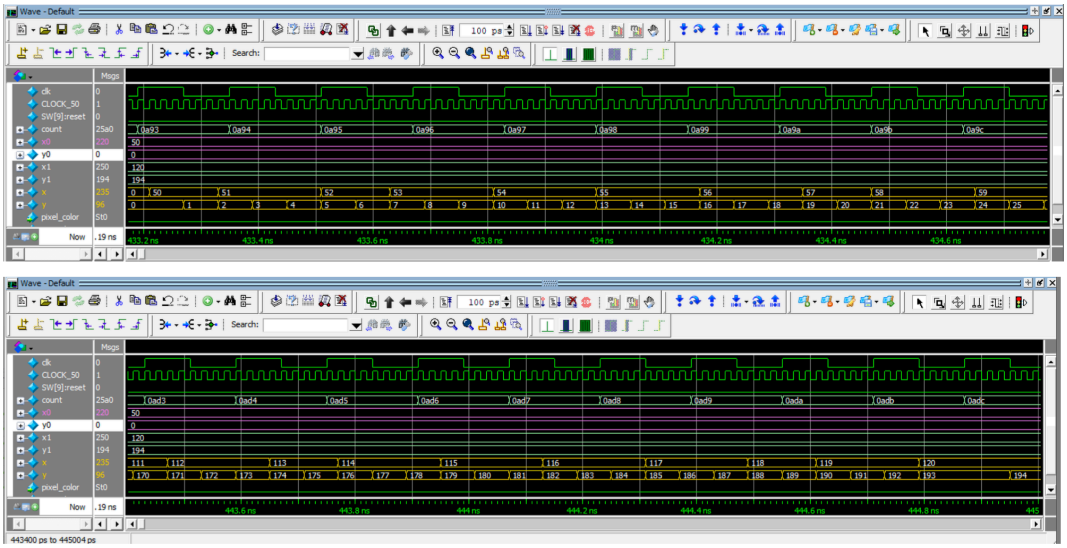

Figure 21: Simulation results of “DE1_SoC” in Task 2 with the parameter “pixel_color” set to “1’b0”. (slope = 2.8)

Simulation Result with the Testbench for "clock_divider.sv":

I wrote the testbench to test the "clock_divider" module in Task 2. Figure 22 is the simulation results.

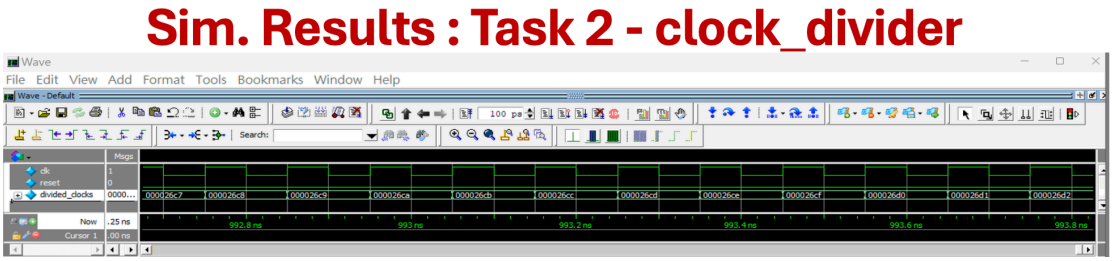

Figure 22: Simulation results of "clock_divider" in Task 2.

Simulation Result with the Testbench for "line_drawer.sv":

The simulation results of "line_drawer" module is Task 2 are the same as that in Task 1. So, I skip the descriptions in this paragraph.

III. Final Product

I finished the design specified in Task 1 and Task 2. I organized each design as shown in the corresponding block diagrams, simulated on the Modelsim with testbenches, and verified the functions with DE1-SoC on the LabsLand. In Task 2, I tried to draw a piecewise-linear function by connecting several line segments.

IV. Appendix

1. Video for Task 1, https://youtu.be/rusGmbzhAY8
2. Video for Task 2, https://youtu.be/THaTnibDRmw
3. SystemVerilog codes of Task 1, please see the following pages.
4. SystemVerilog codes of Task 2, please see the following pages.

I. LAB3-Task1-DE1_SoC.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task 1)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
9                   KEY, LEDR, SW, CLOCK_50, VGA_R, VGA_G, VGA_B,
10                  VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);
11
12      output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
13      output logic [9:0] LEDR;
14      input logic [3:0] KEY;
15      input logic [9:0] SW;
16
17      input CLOCK_50;
18      output [7:0] VGA_R;
19      output [7:0] VGA_G;
20      output [7:0] VGA_B;
21      output VGA_BLANK_N;
22      output VGA_CLK;
23      output VGA_HS;
24      output VGA_SYNC_N;
25      output VGA_VS;
26
27      assign HEX0 = '1;
28      assign HEX1 = '1;
29      assign HEX2 = '1;
30      assign HEX3 = '1;
31      assign HEX4 = '1;
32      assign HEX5 = '1;
33      assign LEDR = SW;
34
35      logic reset;  // delcare reset
36
37      assign reset = SW[9]; // assign SW[9] for readibility
38
39      logic [9:0] x0, x1, x;
40      logic [8:0] y0, y1, y;
41      logic frame_start;
42      logic pixel_color;
43
44
45      //////// DOUBLE_FRAME_BUFFER ////////
46      logic dfb_en;
47      assign dfb_en = 1'b0; //*****************
48      ////////////////////////////////////
49
50      VGA_framebuffer fb (.clk(CLOCK_50), .rst(1'b0), .x, .y,
51                          .pixel_color, .pixel_write(1'b1), .dfb_en, .frame_start,
52                          .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .VGA_VS,
53                          .VGA_BLANK_N, .VGA_SYNC_N);
54
55      // draw lines between (x0, y0) and (x1, y1)
56      line_drawer lines (.clk(CLOCK_50), .reset(1'b0),
57                           .x0, .y0, .x1, .y1, .x, .y);
58
59      // specify 12 lines in hard code
60      always_latch begin
61          case (SW[3:0])
62
63              // draw 1st line
64              1: begin
65                     x0 = 50;
66                     y0 = 50;
67                     x1 = 0;
68                     y1 = 0;
69                     pixel_color = 1'b1;
70                 end
71
72              // draw 2nd line
73              2: begin
74                     x0 = 50;
75                     y0 = 50;
76                     x1 = 100;
77                     y1 = 100;
78                     pixel_color = 1'b1;
```

```
79                  end
80
81                // draw 3rd line
82                3: begin
83                      x0 = 50;
84                      y0 = 50;
85                      x1 = 0;
86                      y1 = 100;
87                      pixel_color = 1'b1;
88                  end
89
90                // draw 4th line
91                4: begin
92                      x0 = 50;
93                      y0 = 50;
94                      x1 = 100;
95                      y1 = 0;
96                      pixel_color = 1'b1;
97                  end
98
99                // draw 5th line
100               5: begin
101                     x0 = 50;
102                     y0 = 50;
103                     x1 = 70;
104                     y1 = 0;
105                     pixel_color = 1'b1;
106                 end
107
108               // draw 6th line
109               6: begin
110                     x0 = 50;
111                     y0 = 50;
112                     x1 = 30;
113                     y1 = 100;
114                     pixel_color = 1'b1;
115                 end
116
117               // draw 7th line
118               7: begin
119                     x0 = 50;
120                     y0 = 50;
121                     x1 = 30;
122                     y1 = 0;
123                     pixel_color = 1'b1;
124                 end
125
126               // draw 8th line
127               8: begin
128                     x0 = 50;
129                     y0 = 50;
130                     x1 = 0;
131                     y1 = 35;
132                     pixel_color = 1'b1;
133                 end
134
135               // draw 9th line
136               9: begin
137                     x0 = 50;
138                     y0 = 50;
139                     x1 = 0;
140                     y1 = 70;
141                     pixel_color = 1'b1;
142                 end
143
144               // draw 10th line
145               10: begin
146                     x0 = 50;
147                     y0 = 50;
148                     x1 = 100;
149                     y1 = 35;
150                     pixel_color = 1'b1;
151                 end
152
153               // draw 11th line
154               11: begin
155                     x0 = 50;
156                     y0 = 50;
```

```
157                     x1 = 100;
158                     y1 = 70;
159                     pixel_color = 1'b1;
160                 end
161
162             // draw 12th line
163             12: begin
164                     x0 = 50;
165                     y0 = 50;
166                     x1 = 70;
167                     y1 = 100;
168                     pixel_color = 1'b1;
169                 end
170
171              default:  ;
172         endcase
173     end
174
175 endmodule
176
177 /*=============================================================================*/
178 //  Testbench------------------
    DE1_SoC_testbench

179 /*=============================================================================*/
180 module DE1_SoC_testbench();
181
182     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
183     logic [9:0] LEDR;
184     logic [3:0] KEY;
185     logic [9:0] SW;
186
187     logic CLOCK_50;
188     logic [7:0] VGA_R;
189     logic [7:0] VGA_G;
190     logic [7:0] VGA_B;
191     logic VGA_BLANK_N;
192     logic VGA_CLK;
193     logic VGA_HS;
194     logic VGA_SYNC_N;
195     logic VGA_VS;
196
197     integer i;
198
199
200      DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5,
201                   .KEY, .LEDR, .SW, .CLOCK_50, .VGA_R, .VGA_G, .VGA_B,
202                   .VGA_BLANK_N, .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
203
204     // Set up a simulated clock: 50 MHz
205     parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
206
207     initial begin
208         CLOCK_50 <= 0;
209         forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
210     end
211
212     // Test the design.
213     initial begin
214         SW[9] <= 1;           // reset = 1
215         repeat(3)
216         @(posedge CLOCK_50);
217         SW[9] <= 0;           // reset = 0
218         @(posedge CLOCK_50);
219
220         for (i = 1; i < 13; i++) begin
221             SW[3:0] <= i;
222             repeat(500)
223             @(posedge CLOCK_50);
224         end
225
226         $stop; // End the simulation.
227     end
228
229 endmodule
230
```

```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task 1 & Task2)
5   //  Device under Test (dut) --- VGA_framebuffer
6   //  File Name: VGA_framebuffer.sv
7   /*=======================================================================*/
8
9   // VGA driver: provides I/O timing and double-buffering for the VGA port.
10
11  module VGA_framebuffer(
12     input logic clk, rst,
13     input logic [9:0] x, // The x coordinate to write to the buffer.
14     input logic [8:0] y, // The y coordinate to write to the buffer.
15     input logic pixel_color, pixel_write, // The data to write (color) and write-enable.
16
17     input logic dfb_en, // Double-Frame Buffer Enable
18
19     output logic frame_start,   // Pulse is fired at the start of a frame.
20
21     // Outputs to the VGA port.
22     output logic [7:0] VGA_R, VGA_G, VGA_B,
23     output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N
24  );
25
26     /*
27     *
28     * HCOUNT 1599 0             1279       1599 0
29     *                _____________               ________
30     * ___________|    Video      |_____________|  Video
31     *
32     *
33     * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
34     *       _____________________      _____________
35     * |____|       VGA_HS          |____|
36     *
37     */
38
39     // Constants for VGA timing.
40     localparam HPX = 11'd640*2, HFP = 11'd16*2, HSP = 11'd96*2, HBP = 11'd48*2;
41     localparam VLN = 11'd480,   VFP = 10'd11,   VSP = 10'd2,    VBP = 10'd31;
42     localparam HTOTAL = HPX + HFP + HSP + HBP; // 800*2=1600
43     localparam VTOTAL = VLN + VFP + VSP + VBP; // 524
44
45     // Horizontal counter.
46     logic [10:0] h_count;
47     logic end_of_line;
48
49     assign end_of_line = h_count == HTOTAL - 1;
50
51     always_ff @(posedge clk)
52        if (rst) h_count <= 0;
53        else if (end_of_line) h_count <= 0;
54        else h_count <= h_count + 11'd1;
55
56     // Vertical counter & buffer swapping.
57     logic [9:0] v_count;
58     logic end_of_field;
59     logic front_odd; // whether odd address is the front buffer.
60
61     assign end_of_field = v_count == VTOTAL - 1;
62     assign frame_start = !h_count && !v_count;
63
64     always_ff @(posedge clk)
65        if (rst) begin
66           v_count <= 0;
67           front_odd <= 0;
68        end else if (end_of_line)
69           if (end_of_field) begin
70              v_count <= 0;
71              front_odd <= !front_odd;
72           end else
73              v_count <= v_count + 10'd1;
74
75     // Sync signals.
76     assign VGA_CLK = h_count[0]; // 25 MHz clock: pixel latched on rising edge.
77     assign VGA_HS = !(h_count - (HPX + HFP) < HSP);
78     assign VGA_VS = !(v_count - (VLN + VFP) < VSP);
```

```
79      assign VGA_SYNC_N = 1; // Unused by VGA
80
81      // Blank area signal.
82      logic blank;
83      assign blank = h_count >= HPX || v_count >= VLN;
84
85      // Double-buffering.
86      logic buffer[640*480*2-1:0];
87      logic [19:0] wr_addr, rd_addr;
88      logic rd_data;
89
90      assign wr_addr = {y * 19'd640 + x, (!front_odd & dfb_en)};
91      assign rd_addr = {v_count * 19'd640 + (h_count / 19'd2), (front_odd & dfb_en)};
92
93      always_ff @(posedge clk) begin
94         if (pixel_write) buffer[wr_addr] <= pixel_color;
95         if (VGA_CLK) begin
96            rd_data <= buffer[rd_addr];
97            VGA_BLANK_N <= ~blank;
98         end
99      end
100
101     // Color output.
102     assign {VGA_R, VGA_G, VGA_B} = rd_data ? 24'hFFFFFF : 24'h000000;
103  endmodule
104
```

```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task 1 & Task 2)
5   //  Device under Test (dut) --- line_drawer
6   //  File Name: line_drawer.sv
7   /*=======================================================================*/
8   module line_drawer(
9      input logic clk, reset,
10
11     // x and y coordinates for the start and end points of the line
12     input logic [9:0] x0, x1,
13     input logic [8:0] y0, y1,
14
15     //outputs cooresponding to the coordinate pair (x, y)
16     output logic [9:0] x,
17     output logic [8:0] y
18     );
19
20     /*
21      * You'll need to create some registers to keep track of things
22      * such as error and direction
23      * Example: */
24      logic signed [9:0] x0_in, x1_in, x_buf;  // buffer for I/O points
25      logic signed [8:0] y0_in, y1_in, y_buf;  // buffer for I/O points
26      logic signed [9:0] error, error_initial; // reg. for error etc.
27      logic signed [9:0] diff_x;               // reg. for difference of x
28      logic signed [8:0] diff_y, diff_y_new;   // reg. for difference of y
29      logic signed [9:0] abs_diff_x;    // reg. for absolute of diff_x
30      logic signed [8:0] abs_diff_y;    // reg. for absolute of diff_y
31      logic signed [9:0] delta_x;       // reg. for delta_x
32      logic signed [8:0] delta_y;       // reg. for delta_y
33      logic signed [8:0] y_step;        // reg. for y_step
34      logic signed [9:0] x_new;         // reg. for x in new iteration
35      logic signed [8:0] y_new;         // reg. for y in new iteration
36      logic signed steep;               // reg. for steep
37
38      // delcare output signals of ASM (control unit)
39      logic get_end_pt, get_abs, get_steep, swap_xy, swap_01;
40      logic draw_pt_0, update_y_error, x_inc, draw_pt_i;
41
42  /*    Define_Variables_Here     */
43  // State variables
44      typedef enum logic [3:0] {
45              idle,
46              end_pt_ready,   // end-points get ready
47              abs_ready,      // absolute value get ready
48              steep_ready,    // steep value get ready
49              swap_xy_ready,  // swap(x,y) get ready
50              swap_01_ready,  // swap(x0, x1), swap(y0,y1) get ready
51              finished_pt,    // finished drawing a point
52              next_iteration, // get ready to enter next iteration
53              new_pt_ready    // (x, y) get ready
54              } state_t;
55      state_t ps, ns;
56
57  /*    Combinational_Logic_Here   */
58  // Next State logic
59      always_latch begin
60          ns = idle; // Default to avoid latches
61          case (ps)
62              idle:
63                  if (!reset) begin
64                      ns = end_pt_ready;
65                      get_end_pt     = 1'b1; // get end points
66                      get_abs        = 1'b0;
67                      get_steep      = 1'b0;
68                      swap_xy        = 1'b0;
69                      swap_01        = 1'b0;
70                      draw_pt_0      = 1'b0;
71                      update_y_error = 1'b0;
72                      x_inc          = 1'b0;
73                      draw_pt_i      = 1'b0;
74                  end
75                  else
76                      ns = idle;
77
78              end_pt_ready:
```

```
79                  begin
80                      ns = abs_ready;
81                      get_end_pt     = 1'b0;
82                      get_abs        = 1'b1; // calculte absolute value
83                      get_steep      = 1'b0;
84                      swap_xy        = 1'b0;
85                      swap_01        = 1'b0;
86                      draw_pt_0      = 1'b0;
87                      update_y_error = 1'b0;
88                      x_inc          = 1'b0;
89                      draw_pt_i      = 1'b0;
90                  end
91
92              abs_ready:
93                  begin
94                      ns = steep_ready;
95                      get_end_pt     = 1'b0;
96                      get_abs        = 1'b0;
97                      get_steep      = 1'b1; // get steep value
98                      swap_xy        = 1'b0;
99                      swap_01        = 1'b0;
100                     draw_pt_0      = 1'b0;
101                     update_y_error = 1'b0;
102                     x_inc          = 1'b0;
103                     draw_pt_i      = 1'b0;
104                 end
105
106             steep_ready:
107                 if (steep) begin
108                     ns = swap_xy_ready;
109                     get_end_pt     = 1'b0;
110                     get_abs        = 1'b0;
111                     get_steep      = 1'b0;
112                     swap_xy        = 1'b1;  // do swap(x, y)
113                     swap_01        = 1'b0;
114                     draw_pt_0      = 1'b0;
115                     update_y_error = 1'b0;
116                     x_inc          = 1'b0;
117                     draw_pt_i      = 1'b0;
118                 end
119                 else begin
120                     ns = swap_xy_ready;
121                     get_end_pt     = 1'b0;
122                     get_abs        = 1'b0;
123                     get_steep      = 1'b0;
124                     swap_xy        = 1'b0;
125                     swap_01        = 1'b0;
126                     draw_pt_0      = 1'b0;
127                     update_y_error = 1'b0;
128                     x_inc          = 1'b0;
129                     draw_pt_i      = 1'b0;
130                 end
131
132             swap_xy_ready:
133                 if (x0_in > x1_in) begin
134                     ns = swap_01_ready;
135                     get_end_pt     = 1'b0;
136                     get_abs        = 1'b0;
137                     get_steep      = 1'b0;
138                     swap_xy        = 1'b0;
139                     swap_01        = 1'b1;  //do swap(x0,x1), swap(y0,y1)
140                     draw_pt_0      = 1'b0;
141                     update_y_error = 1'b0;
142                     x_inc          = 1'b0;
143                     draw_pt_i      = 1'b0;
144                 end
145                 else begin
146                     ns = swap_01_ready;
147                     get_end_pt     = 1'b0;
148                     get_abs        = 1'b0;
149                     get_steep      = 1'b0;
150                     swap_xy        = 1'b0;
151                     swap_01        = 1'b0;
152                     draw_pt_0      = 1'b0;
153                     update_y_error = 1'b0;
154                     x_inc          = 1'b0;
155                     draw_pt_i      = 1'b0;
156                 end
```

```
157
158                 swap_01_ready:
159                     begin
160                         ns = finished_pt;
161                         get_end_pt     = 1'b0;
162                         get_abs        = 1'b0;
163                         get_steep      = 1'b0;
164                         swap_xy        = 1'b0;
165                         swap_01        = 1'b0;
166                         draw_pt_0      = 1'b1; //draw the 1st point
167                         update_y_error = 1'b0;
168                         x_inc          = 1'b0;
169                         draw_pt_i      = 1'b0;
170                     end
171
172                 finished_pt:
173                     if (error >= 0) begin
174                         ns = next_iteration;
175                         get_end_pt     = 1'b0;
176                         get_abs        = 1'b0;
177                         get_steep      = 1'b0;
178                         swap_xy        = 1'b0;
179                         swap_01        = 1'b0;
180                         draw_pt_0      = 1'b0;
181                         update_y_error = 1'b1; // update y and error
182                         x_inc          = 1'b0;
183                         draw_pt_i      = 1'b0;
184                     end
185                     else begin
186                         ns = next_iteration;
187                         get_end_pt     = 1'b0;
188                         get_abs        = 1'b0;
189                         get_steep      = 1'b0;
190                         swap_xy        = 1'b0;
191                         swap_01        = 1'b0;
192                         draw_pt_0      = 1'b0;
193                         update_y_error = 1'b0;
194                         x_inc          = 1'b0;
195                         draw_pt_i      = 1'b0;
196                     end
197
198                 next_iteration:
199                     if (x_buf <= x1_in - 1) begin
200                         ns = new_pt_ready;
201                         get_end_pt     = 1'b0;
202                         get_abs        = 1'b0;
203                         get_steep      = 1'b0;
204                         swap_xy        = 1'b0;
205                         swap_01        = 1'b0;
206                         draw_pt_0      = 1'b0;
207                         update_y_error = 1'b0;
208                         x_inc          = 1'b1; // increase x_new by 1
209                         draw_pt_i      = 1'b0;
210                     end
211                     else begin
212                         ns = idle;
213                         get_end_pt     = 1'b0;
214                         get_abs        = 1'b0;
215                         get_steep      = 1'b0;
216                         swap_xy        = 1'b0;
217                         swap_01        = 1'b0;
218                         draw_pt_0      = 1'b0;
219                         update_y_error = 1'b0;
220                         x_inc          = 1'b0;
221                         draw_pt_i      = 1'b0;
222                     end
223
224                 new_pt_ready:
225                     begin
226                         ns = finished_pt;
227                         get_end_pt     = 1'b0;
228                         get_abs        = 1'b0;
229                         get_steep      = 1'b0;
230                         swap_xy        = 1'b0;
231                         swap_01        = 1'b0;
232                         draw_pt_0      = 1'b0;
233                         update_y_error = 1'b0;
234                         x_inc          = 1'b0;
```

```
235                     draw_pt_i      = 1'b1; // draw the i-th point
236                 end
237
238          default:
239                 begin
240                     ns = finished_pt;
241                     get_end_pt     = 1'b0;
242                     get_abs        = 1'b0;
243                     get_steep      = 1'b0;
244                     swap_xy        = 1'b0;
245                     swap_01        = 1'b0;
246                     draw_pt_0      = 1'b0;
247                     update_y_error = 1'b0;
248                     x_inc          = 1'b0;
249                     draw_pt_i      = 1'b0;
250                 end
251        endcase
252    end
253
254  /*    State Transition at posedge clk  */
255    always_ff @(posedge clk or posedge reset) begin
256         if (reset)
257             ps <= idle;
258         else
259             ps <= ns;
260    end
261
262  /*    Data Path   */
263    always_ff @(posedge clk or posedge reset) begin
264        if (reset) begin
265           // clear regiters related to x, y
266            x <= 0;
267            y <= 0;
268            x_new <= 0;
269            y_new <= 0;
270            x_buf <= 0;
271            y_buf <= 0;
272        end
273        else begin
274
275             if (get_end_pt) begin
276                 // put end points into buffers
277                 // calculte the difference
278                 x0_in <= x0;
279                 y0_in <= y0;
280                 x1_in <= x1;
281                 y1_in <= y1;
282                 diff_x <= x1 - x0;
283                 diff_y <= y1 - y0;
284             end
285
286
287             if (get_abs) begin
288                 // calculte the absolute values
289                 abs_diff_x = (diff_x[9] == 1) ?
290                              (~diff_x + 1'b1) : diff_x;
291                 abs_diff_y = (diff_y[8] == 1) ?
292                              (~diff_y + 1'b1) : diff_y;
293             end
294
295
296             if (get_steep)
297                 // check if it is a steep line
298                 steep = (abs_diff_y > abs_diff_x) ? 1 : 0;
299
300
301             if (swap_xy) begin
302                 // do swap
303                 x0_in <= y0_in;
304                 y0_in <= x0_in;
305                 x1_in <= y1_in;
306                 y1_in <= x1_in;
307             end
308             else if (swap_01) begin
309                 // do swap
310                 x0_in <= x1_in;
311                 x1_in <= x0_in;
312                 y0_in <= y1_in;
```

```
313                 y1_in <= y0_in;
314             end
315             else begin
316                 ;
317             end
318
319
320             if (draw_pt_0) begin  // set and caculate prarmeters
321                 delta_x = x1_in - x0_in;
322                 diff_y_new = y1_in - y0_in;
323                 delta_y = (diff_y_new[8] == 1) ?
324                           (~diff_y_new + 1'b1) : diff_y_new;
325                 error_initial = -(delta_x >> 1);
326                 y_step = (y0_in < y1_in) ? 1 : -1;
327
328                 x_new <= x0_in;
329                 y_new <= y0_in;
330                 x_buf <= x0_in;
331                 y_buf <= y0_in;
332
333                 if (!steep) begin
334                     x = x_buf;
335                     y = y_buf;
336                 end
337                 else begin
338                     y = x_buf;
339                     x = y_buf;
340                 end
341                 error <= error_initial + delta_y;
342             end
343
344
345             if (ps == finished_pt)  // update error & y_new
346                 if (update_y_error) begin
347                     y_new <= y_buf + y_step;
348                     error <= error - delta_x;
349                 end
350                 else
351                     y_new <= y_buf;
352
353
354             if (x_inc) begin
355                 x_new <= x_buf + 1;
356             end
357
358
359             if (draw_pt_i) begin // prepare (x,y) to be drawn
360                 x_buf <= x_new;
361                 y_buf <= y_new;
362                 if (!steep) begin
363                     x = x_buf;
364                     y = y_buf;
365                 end
366                 else begin
367                     x = y_buf;
368                     y = x_buf;
369                 end
370                 error <= error + delta_y;
371             end
372
373
374         end
375     end
376
377 endmodule
378
379 /*=============================================================================*/
380 //   Testbench
381 /*=============================================================================*/
382 module line_drawer_testbench();
383
384 logic clk, reset;
385 logic [9:0] x0, x1, x;
386 logic [8:0] y0, y1, y;
387
388 line_drawer dut (.clk, .reset, .x0, .x1, .y0, .y1, .x, .y);
389
390 parameter CLK_Period = 100;  // set clock period
```

```
391
392     // generate clock signal
393     initial begin
394         clk <= 1'b0;
395         forever #(CLK_Period/2) clk <= ~clk;
396     end
397
398     // reset this module
399     initial begin
400         reset <= 1;
401
402         repeat(3)
403         @(posedge clk);
404         reset <= 0;
405     end
406
407
408     // set values of end points of each line
409     initial begin
410         x0 <= 10'd10;
411         y0 <=  9'd20;
412         x1 <= 10'd300;
413         y1 <=  9'd200;
414         repeat(1000)
415         @(posedge clk);
416
417         reset <= 1;
418         repeat(3)
419         @(posedge clk);
420         reset <= 0;
421
422         x0 <= 10'd10;
423         y0 <=  9'd20;
424         x1 <= 10'd100;
425         y1 <=  9'd200;
426         repeat(1000)
427         @(posedge clk);
428
429         reset <= 1;
430         repeat(3)
431         @(posedge clk);
432         reset <= 0;
433
434         x0 <= 10'd10;
435         y0 <=  9'd20;
436         x1 <= 10'd10;
437         y1 <=  9'd200;
438         repeat(1000)
439         @(posedge clk);
440
441         reset <= 1;
442         repeat(3)
443         @(posedge clk);
444         reset <= 0;
445
446         x0 <= 10'd300;
447         y0 <=  9'd20;
448         x1 <= 10'd10;
449         y1 <=  9'd200;
450         repeat(1000)
451         @(posedge clk);
452
453         reset <= 1;
454         repeat(3)
455         @(posedge clk);
456         reset <= 0;;
457
458         x0 <= 10'd300;
459         y0 <=  9'd20;
460         x1 <= 10'd100;
461         y1 <=  9'd200;
462         repeat(1000)
463         @(posedge clk);
464
465         reset <= 1;
466         repeat(3)
467         @(posedge clk);
468         reset <= 0;
```

```
469
470         x0 <= 10'd300;
471         y0 <=  9'd200;
472         x1 <= 10'd10;
473         y1 <=  9'd20;
474         repeat(1000)
475         @(posedge clk);
476
477         reset <= 1;
478         repeat(3)
479         @(posedge clk);
480         reset <= 0;
481
482
483         x0 <= 10'd100;
484         y0 <=  9'd200;
485         x1 <= 10'd10;
486         y1 <=  9'd20;
487         repeat(1000)
488         @(posedge clk);
489
490         reset <= 1;
491         repeat(3)
492         @(posedge clk);
493         reset <= 0;
494
495         x0 <= 10'd100;
496         y0 <=  9'd200;
497         x1 <= 10'd200;
498         y1 <=  9'd200;
499         repeat(1000)
500         @(posedge clk);
501
502         $stop;
503
504         end
505
506     endmodule
507
```

IV. LAB3-Task2-DE1_SoC.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task2)
5   //  Device under Test (dut) --- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=================================================================*/
8   module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
9      VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS);
10
11     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
12     output logic [9:0] LEDR;
13     input logic [3:0] KEY;
14     input logic [9:0] SW;
15
16     input CLOCK_50;
17     output [7:0] VGA_R;
18     output [7:0] VGA_G;
19     output [7:0] VGA_B;
20     output VGA_BLANK_N;
21     output VGA_CLK;
22     output VGA_HS;
23     output VGA_SYNC_N;
24     output VGA_VS;
25
26     assign HEX0 = '1;
27     assign HEX1 = '1;
28     assign HEX2 = '1;
29     assign HEX3 = '1;
30     assign HEX4 = '1;
31     assign HEX5 = '1;
32     assign LEDR = SW;
33
34     logic [9:0] x0, x1, x;
35     logic [8:0] y0, y1, y;
36     logic frame_start;
37     logic pixel_color;
38
39
40     logic [15:0] count;
41     logic reset;      ///<--------Task2, for counter
42     logic reset_line; ///<--------Task2, only for line_drawer
43
44     assign reset = SW[9]; // do reset by sliding SW[9]
45
46
47         // Generate clk off of CLOCK_50, whichClock picks rate    .
48     logic [31:0] div_clk;
49
50     parameter whichClock = 16; // 2 for simulation, 16 for DE1_SoC board
51  /*=================================================================*/
52  //     clock_divider (File Name: clock_divider.sv)
53  /*=================================================================*/
54      clock_divider cdiv (.clock(CLOCK_50), .reset(SW[9]), .divided_clocks(div_clk));
55  /*-----------------------------------------------------------------*/
56  //     Clock selection;
57  /*-----------------------------------------------------------------*/
58     logic clkSelect;
59  //*****************************************************************
60     assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
61  //*****************************************************************
62
63
64     /////// DOUBLE_FRAME_BUFFER ////////
65     logic dfb_en;
66     assign dfb_en = 1'b0; //****************
67     ////////////////////////////////////
68
69     VGA_framebuffer fb (.clk(CLOCK_50), .rst(1'b0), .x, .y,
70              .pixel_color, .pixel_write(1'b1), .dfb_en, .frame_start,
71              .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .VGA_VS,
72              .VGA_BLANK_N, .VGA_SYNC_N);
73
74     // draw lines between (x0, y0) and (x1, y1)
75     line_drawer lines (.clk(CLOCK_50),
76                        .reset(reset_line),  //////<---reset, only in Task 2
77                        .x0, .y0, .x1, .y1, .x, .y);
78
```

```
 79
 80      // counter for drawing lines cyclically
 81      always_ff @(posedge clkSelect or posedge reset) begin
 82          if (reset)
 83              count <= 16'b0;
 84          else
 85              count <= count + 1;
 86      end
 87
 88      // specify different lines
 89      always_latch begin
 90          case (count)
 91              // draw 1st line
 92              16'h0000: {reset_line, pixel_color} = 2'b10;  // reset line_drawer
 93              16'h0010: begin
 94                            reset_line = 0;
 95                            x0 = 0;
 96                            y0 = 200;
 97                            x1 = 50;
 98                            y1 = 0;
 99                            pixel_color = 1;  // white
100                        end
101
102              // draw 2nd line
103              16'h0190: {reset_line, pixel_color} = 2'b10;  // reset line_drawer
104              16'h0191: begin
105                            reset_line = 0;
106                            x0 = 50;
107                            y0 = 0;
108                            x1 = 120;
109                            y1 = 194;
110                            pixel_color = 1'b1; // white
111                        end
112
113
114              // draw 3rd line
115              16'h0310: {reset_line, pixel_color} = 2'b10;  // reset
    line_drawer
116              16'h0312: begin
117                            reset_line = 0;
118                            x0 = 120;
119                            y0 = 194;
120                            x1 = 170;
121                            y1 = 194;
122                            pixel_color = 1'b1;  // white
123                        end
124
125              // draw 4th line
126              16'h0490: {reset_line, pixel_color} = 2'b10;  // reset line_drawer
127              16'h0492: begin
128                            reset_line = 0;
129                            x0 = 170;
130                            y0 = 194;
131                            x1 = 220;
132                            y1 = 9;
133                            pixel_color = 1'b1;    // white
134                        end
135
136              // draw 5th line
137              16'h0610: {reset_line, pixel_color} = 2'b10;  // reset line_drawer
138              16'h0612: begin
139                            reset_line = 0;
140                            x0 = 220;
141                            y0 = 0;
142                            x1 = 250;
143                            y1 = 194;
144                            pixel_color = 1'b1;    // white
145                        end
146
147
148
149              // ******clear 3rd line
150              16'h0790: {reset_line, pixel_color} = 2'b10; // reset line_drawer
151              16'h0792: begin
152                            reset_line = 0;
153                            x0 = 120;
154                            y0 = 194;
155                            x1 = 170;
```

```
                            y1 = 194;
                            pixel_color = 1'b0;    // black
                        end

                // ****** clear 1st line
                16'h0910: {reset_line, pixel_color} = 2'b10;  // reset line_drawer
                16'h0912: begin
                            reset_line = 0;
                            x0 = 0;
                            y0 = 200;
                            x1 = 50;
                            y1 = 0;
                            pixel_color = 1'b0;     // black
                        end

                // ****** clear 2nd line
                16'h0A90: {reset_line, pixel_color} = 2'b10; // reset line_drawer
                16'h0A92: begin
                            reset_line = 0;
                            x0 = 50;
                            y0 = 0;
                            x1 = 120;
                            y1 = 194;
                            pixel_color = 1'b0;     // black
                        end

                // ****** clear 4th line
                16'h0C10: {reset_line, pixel_color} = 2'b10; // reset line_drawer
                16'h0C12: begin
                            reset_line = 0;
                            x0 = 170;
                            y0 = 194;
                            x1 = 220;
                            y1 = 9;
                            pixel_color = 1'b0;      // black
                        end

                // ****** clear 5th line
                16'h0D90: {reset_line, pixel_color} = 2'b10; // reset line_drawer
                16'h0D92: begin
                            reset_line = 0;
                            x0 = 220;
                            y0 = 0;
                            x1 = 250;
                            y1 = 194;
                            pixel_color = 1'b0;       // black
                        end

                default:  ;
            endcase
        end

    endmodule

    /*=================================================================*/
    //  Testbench-----------------
    DE1_SoC_testbench

    /*=================================================================*/
    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;

        // instant
         DE1_SoC dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5,
```

```
232                      .KEY, .LEDR, .SW, .CLOCK_50,
233                     .VGA_R, .VGA_G, .VGA_B, .VGA_BLANK_N,
234                      .VGA_CLK, .VGA_HS, .VGA_SYNC_N, .VGA_VS);
235
236   // Set up a simulated clock: 50 MHz
237   parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
238
239   initial begin
240       CLOCK_50 <= 0;
241       forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
242   end
243
244   // Test the design.
245   initial begin
246       SW[9] <= 1;  // reset <= 1;
247       repeat(10)
248       @(posedge CLOCK_50);
249       SW[9] <= 0;  // reset <= 0;
250
251       repeat(20000000)
252      @(posedge CLOCK_50);
253       $stop; // End the simulation.
254   end
255
256   endmodule
257
```

```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task 1 & Task2)
5   //  Device under Test (dut) --- VGA_framebuffer
6   //  File Name: VGA_framebuffer.sv
7   /*=======================================================================*/
8
9   // VGA driver: provides I/O timing and double-buffering for the VGA port.
10
11  module VGA_framebuffer(
12     input logic clk, rst,
13     input logic [9:0] x, // The x coordinate to write to the buffer.
14     input logic [8:0] y, // The y coordinate to write to the buffer.
15     input logic pixel_color, pixel_write, // The data to write (color) and write-enable.
16
17     input logic dfb_en, // Double-Frame Buffer Enable
18
19     output logic frame_start,   // Pulse is fired at the start of a frame.
20
21     // Outputs to the VGA port.
22     output logic [7:0] VGA_R, VGA_G, VGA_B,
23     output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N
24  );
25
26     /*
27     *
28     * HCOUNT 1599 0             1279       1599 0
29     *             _______________           ________
30     * ___________|    Video      |___________|  Video
31     *
32     *
33     * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
34     *       _____________________     _____________
35     * |____|       VGA_HS          |____|
36     *
37     */
38
39     // Constants for VGA timing.
40     localparam HPX = 11'd640*2, HFP = 11'd16*2, HSP = 11'd96*2, HBP = 11'd48*2;
41     localparam VLN = 11'd480,   VFP = 10'd11,   VSP = 10'd2,    VBP = 10'd31;
42     localparam HTOTAL = HPX + HFP + HSP + HBP; // 800*2=1600
43     localparam VTOTAL = VLN + VFP + VSP + VBP; // 524
44
45     // Horizontal counter.
46     logic [10:0] h_count;
47     logic end_of_line;
48
49     assign end_of_line = h_count == HTOTAL - 1;
50
51     always_ff @(posedge clk)
52        if (rst) h_count <= 0;
53        else if (end_of_line) h_count <= 0;
54        else h_count <= h_count + 11'd1;
55
56     // Vertical counter & buffer swapping.
57     logic [9:0] v_count;
58     logic end_of_field;
59     logic front_odd; // whether odd address is the front buffer.
60
61     assign end_of_field = v_count == VTOTAL - 1;
62     assign frame_start = !h_count && !v_count;
63
64     always_ff @(posedge clk)
65        if (rst) begin
66           v_count <= 0;
67           front_odd <= 0;
68        end else if (end_of_line)
69           if (end_of_field) begin
70              v_count <= 0;
71              front_odd <= !front_odd;
72           end else
73              v_count <= v_count + 10'd1;
74
75     // Sync signals.
76     assign VGA_CLK = h_count[0]; // 25 MHz clock: pixel latched on rising edge.
77     assign VGA_HS = !(h_count - (HPX + HFP) < HSP);
78     assign VGA_VS = !(v_count - (VLN + VFP) < VSP);
```

```
79      assign VGA_SYNC_N = 1; // Unused by VGA
80
81      // Blank area signal.
82      logic blank;
83      assign blank = h_count >= HPX || v_count >= VLN;
84
85      // Double-buffering.
86      logic buffer[640*480*2-1:0];
87      logic [19:0] wr_addr, rd_addr;
88      logic rd_data;
89
90      assign wr_addr = {y * 19'd640 + x, (!front_odd & dfb_en)};
91      assign rd_addr = {v_count * 19'd640 + (h_count / 19'd2), (front_odd & dfb_en)};
92
93      always_ff @(posedge clk) begin
94         if (pixel_write) buffer[wr_addr] <= pixel_color;
95         if (VGA_CLK) begin
96            rd_data <= buffer[rd_addr];
97            VGA_BLANK_N <= ~blank;
98         end
99      end
100
101     // Color output.
102     assign {VGA_R, VGA_G, VGA_B} = rd_data ? 24'hFFFFFF : 24'h000000;
103  endmodule
104
```

```
1   /*=======================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task 1 & Task 2)
5   //  Device under Test (dut) --- line_drawer
6   //  File Name: line_drawer.sv
7   /*=======================================================================*/
8   module line_drawer(
9      input logic clk, reset,
10
11     // x and y coordinates for the start and end points of the line
12     input logic [9:0] x0, x1,
13     input logic [8:0] y0, y1,
14
15     //outputs cooresponding to the coordinate pair (x, y)
16     output logic [9:0] x,
17     output logic [8:0] y
18     );
19
20     /*
21      * You'll need to create some registers to keep track of things
22      * such as error and direction
23      * Example: */
24      logic signed [9:0] x0_in, x1_in, x_buf;  // buffer for I/O points
25      logic signed [8:0] y0_in, y1_in, y_buf;  // buffer for I/O points
26      logic signed [9:0] error, error_initial; // reg. for error etc.
27      logic signed [9:0] diff_x;               // reg. for difference of x
28      logic signed [8:0] diff_y, diff_y_new;   // reg. for difference of y
29      logic signed [9:0] abs_diff_x;    // reg. for absolute of diff_x
30      logic signed [8:0] abs_diff_y;    // reg. for absolute of diff_y
31      logic signed [9:0] delta_x;       // reg. for delta_x
32      logic signed [8:0] delta_y;       // reg. for delta_y
33      logic signed [8:0] y_step;        // reg. for y_step
34      logic signed [9:0] x_new;         // reg. for x in new iteration
35      logic signed [8:0] y_new;         // reg. for y in new iteration
36      logic signed steep;               // reg. for steep
37
38      // delcare output signals of ASM (control unit)
39      logic get_end_pt, get_abs, get_steep, swap_xy, swap_01;
40      logic draw_pt_0, update_y_error, x_inc, draw_pt_i;
41
42  /*    Define_Variables_Here     */
43  // State variables
44      typedef enum logic [3:0] {
45              idle,
46              end_pt_ready,   // end-points get ready
47              abs_ready,      // absolute value get ready
48              steep_ready,    // steep value get ready
49              swap_xy_ready,  // swap(x,y) get ready
50              swap_01_ready,  // swap(x0, x1), swap(y0,y1) get ready
51              finished_pt,    // finished drawing a point
52              next_iteration, // get ready to enter next iteration
53              new_pt_ready    // (x, y) get ready
54              } state_t;
55      state_t ps, ns;
56
57  /*    Combinational_Logic_Here   */
58  // Next State logic
59      always_latch begin
60          ns = idle; // Default to avoid latches
61          case (ps)
62              idle:
63                  if (!reset) begin
64                      ns = end_pt_ready;
65                      get_end_pt     = 1'b1; // get end points
66                      get_abs        = 1'b0;
67                      get_steep      = 1'b0;
68                      swap_xy        = 1'b0;
69                      swap_01        = 1'b0;
70                      draw_pt_0      = 1'b0;
71                      update_y_error = 1'b0;
72                      x_inc          = 1'b0;
73                      draw_pt_i      = 1'b0;
74                  end
75                  else
76                      ns = idle;
77
78              end_pt_ready:
```

```
79                  begin
80                      ns = abs_ready;
81                      get_end_pt     = 1'b0;
82                      get_abs        = 1'b1; // calculte absolute value
83                      get_steep      = 1'b0;
84                      swap_xy        = 1'b0;
85                      swap_01        = 1'b0;
86                      draw_pt_0      = 1'b0;
87                      update_y_error = 1'b0;
88                      x_inc          = 1'b0;
89                      draw_pt_i      = 1'b0;
90                  end
91
92              abs_ready:
93                  begin
94                      ns = steep_ready;
95                      get_end_pt     = 1'b0;
96                      get_abs        = 1'b0;
97                      get_steep      = 1'b1; // get steep value
98                      swap_xy        = 1'b0;
99                      swap_01        = 1'b0;
100                     draw_pt_0      = 1'b0;
101                     update_y_error = 1'b0;
102                     x_inc          = 1'b0;
103                     draw_pt_i      = 1'b0;
104                 end
105
106             steep_ready:
107                 if (steep) begin
108                     ns = swap_xy_ready;
109                     get_end_pt     = 1'b0;
110                     get_abs        = 1'b0;
111                     get_steep      = 1'b0;
112                     swap_xy        = 1'b1;  // do swap(x, y)
113                     swap_01        = 1'b0;
114                     draw_pt_0      = 1'b0;
115                     update_y_error = 1'b0;
116                     x_inc          = 1'b0;
117                     draw_pt_i      = 1'b0;
118                 end
119                 else begin
120                     ns = swap_xy_ready;
121                     get_end_pt     = 1'b0;
122                     get_abs        = 1'b0;
123                     get_steep      = 1'b0;
124                     swap_xy        = 1'b0;
125                     swap_01        = 1'b0;
126                     draw_pt_0      = 1'b0;
127                     update_y_error = 1'b0;
128                     x_inc          = 1'b0;
129                     draw_pt_i      = 1'b0;
130                 end
131
132             swap_xy_ready:
133                 if (x0_in > x1_in) begin
134                     ns = swap_01_ready;
135                     get_end_pt     = 1'b0;
136                     get_abs        = 1'b0;
137                     get_steep      = 1'b0;
138                     swap_xy        = 1'b0;
139                     swap_01        = 1'b1;  //do swap(x0,x1), swap(y0,y1)
140                     draw_pt_0      = 1'b0;
141                     update_y_error = 1'b0;
142                     x_inc          = 1'b0;
143                     draw_pt_i      = 1'b0;
144                 end
145                 else begin
146                     ns = swap_01_ready;
147                     get_end_pt     = 1'b0;
148                     get_abs        = 1'b0;
149                     get_steep      = 1'b0;
150                     swap_xy        = 1'b0;
151                     swap_01        = 1'b0;
152                     draw_pt_0      = 1'b0;
153                     update_y_error = 1'b0;
154                     x_inc          = 1'b0;
155                     draw_pt_i      = 1'b0;
156                 end
```

```
157
158                 swap_01_ready:
159                     begin
160                         ns = finished_pt;
161                         get_end_pt     = 1'b0;
162                         get_abs        = 1'b0;
163                         get_steep      = 1'b0;
164                         swap_xy        = 1'b0;
165                         swap_01        = 1'b0;
166                         draw_pt_0      = 1'b1; //draw the 1st point
167                         update_y_error = 1'b0;
168                         x_inc          = 1'b0;
169                         draw_pt_i      = 1'b0;
170                     end
171
172                 finished_pt:
173                     if (error >= 0) begin
174                         ns = next_iteration;
175                         get_end_pt     = 1'b0;
176                         get_abs        = 1'b0;
177                         get_steep      = 1'b0;
178                         swap_xy        = 1'b0;
179                         swap_01        = 1'b0;
180                         draw_pt_0      = 1'b0;
181                         update_y_error = 1'b1; // update y and error
182                         x_inc          = 1'b0;
183                         draw_pt_i      = 1'b0;
184                     end
185                     else begin
186                         ns = next_iteration;
187                         get_end_pt     = 1'b0;
188                         get_abs        = 1'b0;
189                         get_steep      = 1'b0;
190                         swap_xy        = 1'b0;
191                         swap_01        = 1'b0;
192                         draw_pt_0      = 1'b0;
193                         update_y_error = 1'b0;
194                         x_inc          = 1'b0;
195                         draw_pt_i      = 1'b0;
196                     end
197
198                 next_iteration:
199                     if (x_buf <= x1_in - 1) begin
200                         ns = new_pt_ready;
201                         get_end_pt     = 1'b0;
202                         get_abs        = 1'b0;
203                         get_steep      = 1'b0;
204                         swap_xy        = 1'b0;
205                         swap_01        = 1'b0;
206                         draw_pt_0      = 1'b0;
207                         update_y_error = 1'b0;
208                         x_inc          = 1'b1; // increase x_new by 1
209                         draw_pt_i      = 1'b0;
210                     end
211                     else begin
212                         ns = idle;
213                         get_end_pt     = 1'b0;
214                         get_abs        = 1'b0;
215                         get_steep      = 1'b0;
216                         swap_xy        = 1'b0;
217                         swap_01        = 1'b0;
218                         draw_pt_0      = 1'b0;
219                         update_y_error = 1'b0;
220                         x_inc          = 1'b0;
221                         draw_pt_i      = 1'b0;
222                     end
223
224                 new_pt_ready:
225                     begin
226                         ns = finished_pt;
227                         get_end_pt     = 1'b0;
228                         get_abs        = 1'b0;
229                         get_steep      = 1'b0;
230                         swap_xy        = 1'b0;
231                         swap_01        = 1'b0;
232                         draw_pt_0      = 1'b0;
233                         update_y_error = 1'b0;
234                         x_inc          = 1'b0;
```

```
235                     draw_pt_i      = 1'b1; // draw the i-th point
236                 end
237
238          default:
239                 begin
240                     ns = finished_pt;
241                     get_end_pt     = 1'b0;
242                     get_abs        = 1'b0;
243                     get_steep      = 1'b0;
244                     swap_xy        = 1'b0;
245                     swap_01        = 1'b0;
246                     draw_pt_0      = 1'b0;
247                     update_y_error = 1'b0;
248                     x_inc          = 1'b0;
249                     draw_pt_i      = 1'b0;
250                 end
251        endcase
252    end
253
254  /*    State Transition at posedge clk  */
255    always_ff @(posedge clk or posedge reset) begin
256         if (reset)
257             ps <= idle;
258         else
259             ps <= ns;
260    end
261
262  /*    Data Path   */
263    always_ff @(posedge clk or posedge reset) begin
264        if (reset) begin
265           // clear regiters related to x, y
266            x <= 0;
267            y <= 0;
268            x_new <= 0;
269            y_new <= 0;
270            x_buf <= 0;
271            y_buf <= 0;
272        end
273        else begin
274
275             if (get_end_pt) begin
276                 // put end points into buffers
277                 // calculte the difference
278                 x0_in <= x0;
279                 y0_in <= y0;
280                 x1_in <= x1;
281                 y1_in <= y1;
282                 diff_x <= x1 - x0;
283                 diff_y <= y1 - y0;
284             end
285
286
287             if (get_abs) begin
288                 // calculte the absolute values
289                 abs_diff_x = (diff_x[9] == 1) ?
290                              (~diff_x + 1'b1) : diff_x;
291                 abs_diff_y = (diff_y[8] == 1) ?
292                              (~diff_y + 1'b1) : diff_y;
293             end
294
295
296             if (get_steep)
297                 // check if it is a steep line
298                 steep = (abs_diff_y > abs_diff_x) ? 1 : 0;
299
300
301             if (swap_xy) begin
302                 // do swap
303                 x0_in <= y0_in;
304                 y0_in <= x0_in;
305                 x1_in <= y1_in;
306                 y1_in <= x1_in;
307             end
308             else if (swap_01) begin
309                 // do swap
310                 x0_in <= x1_in;
311                 x1_in <= x0_in;
312                 y0_in <= y1_in;
```

```
313                 y1_in <= y0_in;
314             end
315             else begin
316                 ;
317             end
318
319
320             if (draw_pt_0) begin  // set and caculate prarmeters
321                 delta_x = x1_in - x0_in;
322                 diff_y_new = y1_in - y0_in;
323                 delta_y = (diff_y_new[8] == 1) ?
324                           (~diff_y_new + 1'b1) : diff_y_new;
325                 error_initial = -(delta_x >> 1);
326                 y_step = (y0_in < y1_in) ? 1 : -1;
327
328                 x_new <= x0_in;
329                 y_new <= y0_in;
330                 x_buf <= x0_in;
331                 y_buf <= y0_in;
332
333                 if (!steep) begin
334                     x = x_buf;
335                     y = y_buf;
336                 end
337                 else begin
338                     y = x_buf;
339                     x = y_buf;
340                 end
341                 error <= error_initial + delta_y;
342             end
343
344
345             if (ps == finished_pt)  // update error & y_new
346                 if (update_y_error) begin
347                     y_new <= y_buf + y_step;
348                     error <= error - delta_x;
349                 end
350                 else
351                     y_new <= y_buf;
352
353
354             if (x_inc) begin
355                 x_new <= x_buf + 1;
356             end
357
358
359             if (draw_pt_i) begin // prepare (x,y) to be drawn
360                 x_buf <= x_new;
361                 y_buf <= y_new;
362                 if (!steep) begin
363                     x = x_buf;
364                     y = y_buf;
365                 end
366                 else begin
367                     x = y_buf;
368                     y = x_buf;
369                 end
370                 error <= error + delta_y;
371             end
372
373
374         end
375     end
376
377 endmodule
378
379 /*=============================================================================*/
380 //   Testbench
381 /*=============================================================================*/
382 module line_drawer_testbench();
383
384 logic clk, reset;
385 logic [9:0] x0, x1, x;
386 logic [8:0] y0, y1, y;
387
388 line_drawer dut (.clk, .reset, .x0, .x1, .y0, .y1, .x, .y);
389
390 parameter CLK_Period = 100;  // set clock period
```

```
391
392     // generate clock signal
393     initial begin
394         clk <= 1'b0;
395         forever #(CLK_Period/2) clk <= ~clk;
396     end
397
398     // reset this module
399     initial begin
400         reset <= 1;
401
402         repeat(3)
403         @(posedge clk);
404         reset <= 0;
405     end
406
407
408     // set values of end points of each line
409     initial begin
410         x0 <= 10'd10;
411         y0 <=  9'd20;
412         x1 <= 10'd300;
413         y1 <=  9'd200;
414         repeat(1000)
415         @(posedge clk);
416
417         reset <= 1;
418         repeat(3)
419         @(posedge clk);
420         reset <= 0;
421
422         x0 <= 10'd10;
423         y0 <=  9'd20;
424         x1 <= 10'd100;
425         y1 <=  9'd200;
426         repeat(1000)
427         @(posedge clk);
428
429         reset <= 1;
430         repeat(3)
431         @(posedge clk);
432         reset <= 0;
433
434         x0 <= 10'd10;
435         y0 <=  9'd20;
436         x1 <= 10'd10;
437         y1 <=  9'd200;
438         repeat(1000)
439         @(posedge clk);
440
441         reset <= 1;
442         repeat(3)
443         @(posedge clk);
444         reset <= 0;
445
446         x0 <= 10'd300;
447         y0 <=  9'd20;
448         x1 <= 10'd10;
449         y1 <=  9'd200;
450         repeat(1000)
451         @(posedge clk);
452
453         reset <= 1;
454         repeat(3)
455         @(posedge clk);
456         reset <= 0;;
457
458         x0 <= 10'd300;
459         y0 <=  9'd20;
460         x1 <= 10'd100;
461         y1 <=  9'd200;
462         repeat(1000)
463         @(posedge clk);
464
465         reset <= 1;
466         repeat(3)
467         @(posedge clk);
468         reset <= 0;
```

```
469
470         x0 <= 10'd300;
471         y0 <=  9'd200;
472         x1 <= 10'd10;
473         y1 <=  9'd20;
474         repeat(1000)
475         @(posedge clk);
476
477         reset <= 1;
478         repeat(3)
479         @(posedge clk);
480         reset <= 0;
481
482
483         x0 <= 10'd100;
484         y0 <=  9'd200;
485         x1 <= 10'd10;
486         y1 <=  9'd20;
487         repeat(1000)
488         @(posedge clk);
489
490         reset <= 1;
491         repeat(3)
492         @(posedge clk);
493         reset <= 0;
494
495         x0 <= 10'd100;
496         y0 <=  9'd200;
497         x1 <= 10'd200;
498         y1 <=  9'd200;
499         repeat(1000)
500         @(posedge clk);
501
502         $stop;
503
504         end
505
506     endmodule
507
```

VII. LAB3-Task2-clock_divider.sv


```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 02-07-2025
4   //  EE/CSE371 LAB3-- Display Interface (Task2)
5   //  Device under Test (dut) --- clock_divider
6   //  File Name: clock_divider.sv
7   /*=================================================================*/
8   // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz,
    ...
9   module clock_divider (clock, reset, divided_clocks);
10
11  input logic reset, clock;
12  output logic [31:0] divided_clocks = 0;
13
14  always_ff @(posedge clock) begin
15      divided_clocks <= divided_clocks + 1;
16  end
17
18  endmodule
19
20  /*=================================================================*/
21  //   Testbench
22  /*=================================================================*/
23  module clock_divider_testbench();
24
25  logic clk, reset;
26  logic [31:0] divided_clocks;
27
28  // instance
29  clock_divider dut (.clock(clk), .reset, .divided_clocks);
30
31  parameter CLK_Period = 100;  // set clock period
32
33  // generate clock signal
34  initial begin
35      clk <= 1'b0;
36      forever #(CLK_Period/2) clk <= ~clk;
37  end
38
39  // reset and test this module
40  initial begin
41      reset <= 1;
42
43      repeat(3)
44      @(posedge clk);
45      reset <= 0;
46
47      repeat(10000)
48      @(posedge clk);
49
50      $stop;
51
52  end
53
54  endmodule
55
56
```