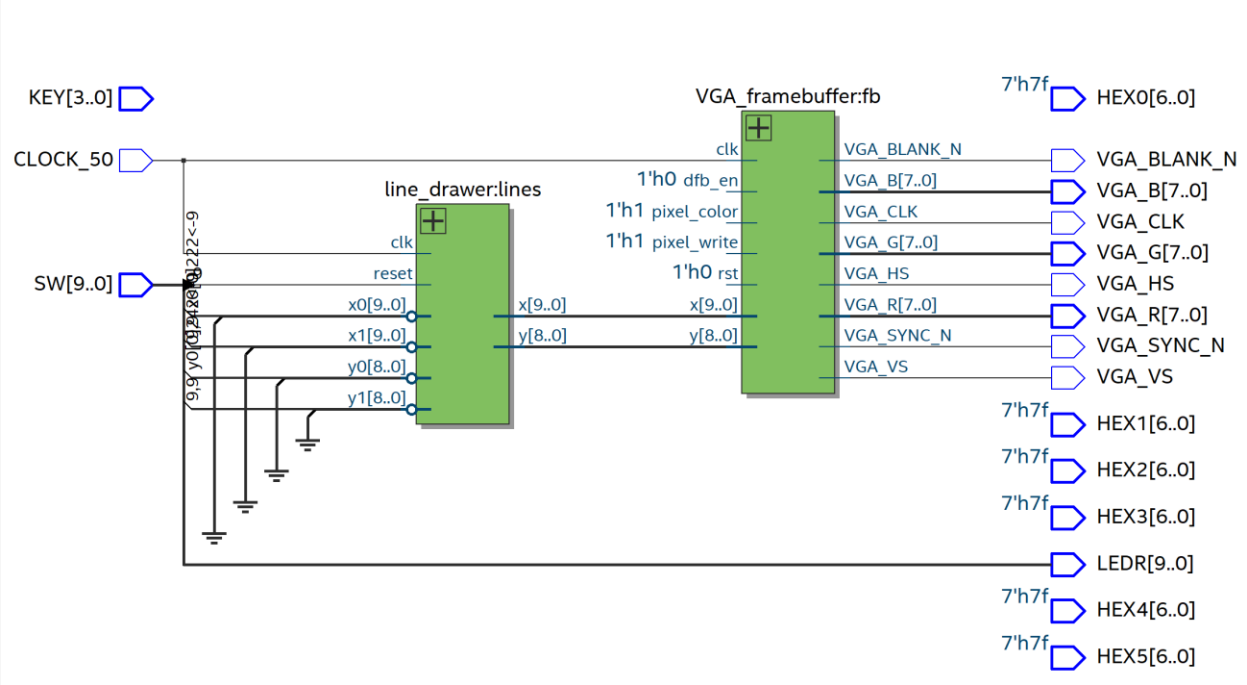
Alan Li
EE 371
February 12th, 2022
Lab 3 Report

Task1

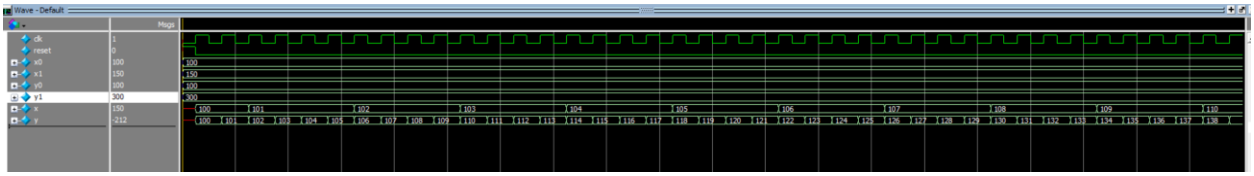
Section 1: Procedure



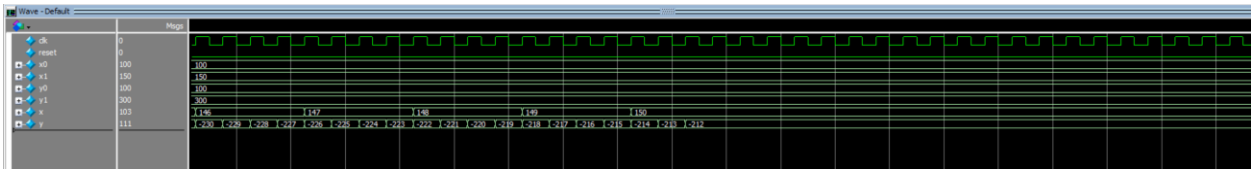
My approach to draw a line for task 1 is to display all pixels on the line one by one at each clock edge. There are 3 modules for task1, DE1_SoC is top-level module, VGA_framebuffer has already been designed. I implement a line_drawer module to finish the task. The algorithms for developing line_drawer can be found here: http://members.chello.at/easyfilter/bresenham.html. I did some modifications since the algorithm is originally written in C which is a programming language. There is some differences between programing languages and VHDL. I put the lines inside for loop of c code in a always_ff block, in that way the registers will get updated at each rising clock edge. I put everything else inside an always_comb block.

Section 2: Results

Line_drawer testbench



For line_drawer testbench, I set x0, y0, x1, y1 as 100,100,150,300. After reset, both x and y are increasing at different rates because the slope is not 45 degrees. After hundreds of clock cycles, xPointer and yPointer has reached the destination coordinates and will stop there like displayed below.



Section 3: Appendix

```
// Alan Li
// 02/12/2022
// EE 371
// Lab #3

// line_drawer takes 10-bit x0, x1, 9-bit y0, y1 and clk, reset as input
// and return 10-bit x and 9-bit y as output

module line_drawer(
    input logic clk, reset,

    // x and y coordinates for the start and end points of the line
    input logic [9:0] x0, x1,
    input logic [8:0] y0, y1,

    // outputs cooresponding to the coordinate pair (x, y)
    output logic [9:0] x,
    output logic [8:0] y
    );

    // These registers are created following Bresenham's Algorithm
    logic signed [11:0] error;
    logic signed [23:0] e2;
    logic signed [10:0] deltaX;
    logic signed [9:0] deltaY;
    logic sx, sy;

    // xPointer and yPointer act as local registers and interate all the pixels on the line
    logic [10:0] xPointer;
    logic [9:0] yPointer;

    // Set deltaX, deltaY, sx and sy as constant parameters, they are used inside always_ff
    always_comb begin
        // absolute value for deltaX and deltaY
        if (x1 > x0) deltaX = (x1 - x0);
        else deltaX = -(x1 - x0);

        if (y1 > y0) deltaY = -(y1 - y0);
        else deltaY = (y1 - y0);

        if (x0 < x1) sx = 1;
        else sx = -1;

        if (y0 < y1) sy = 1;
        else sy = -1;
    end

    // in c code, there is a for loop to iterate all the pixels on the lines,
    // in VHDL, I used a flip-flop to iterate all the points
    always_ff @(posedge clk) begin
        // when reset, set all registers to initial value
        if (reset) begin
            error <= deltaX + deltaY;
            e2 <= 0;
            xPointer <= x0;
            yPointer <= y0;
        end
        // when xPointer and yPointer has not reached the destination coordinates
        // excute this else block
        else if (~((xPointer == x1) & (yPointer == y1))) begin
            e2 = 2 * error;
```

```
            xPointer <= x0;
            yPointer <= y0;
        end
        // when xPointer and yPointer has not reached the destination coordinates
        // excute this else block
        else if (~((xPointer == x1) & (yPointer == y1))) begin
            e2 = 2 * error;
            if (e2 >= deltaY) begin
                error <= error + deltaY;
                xPointer <= xPointer + sx;
            end

            if (e2 <= deltaX) begin
                error <= error + deltaX;
                yPointer <= yPointer + sy;
            end

            // both condition is true
            // without this if block, if e2 >= deltaY and e2 <= deltaX, only the first if block
            // will be executed and the second if block will be ignored
            if ((e2 >= deltaY) && (e2 <= deltaX)) begin
                error <= error + deltaX + deltaY;
                yPointer <= yPointer + sy;
                xPointer <= xPointer + sx;
            end


        end
    end
    assign x = xPointer;
    assign y = yPointer;
endmodule

// 1,1 12,5 works
// 100,100 150,300 works
// 2,2 2,100 works
// line_drawer_testbench tests all expected behavior that will encounter
module line_drawer_testbench();
    logic clk, reset;
    logic [9:0] x0, x1;
    logic [8:0] y0, y1;
    logic [9:0] x;
    logic [8:0] y;

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

    line_drawer dut(.clk, .reset, .x0, .x1, .y0, .y1, .x, .y);

    initial begin
        reset <= 1; x0 <= 100; y0 <= 100; x1 <= 150; y1 <= 300; @(posedge clk);
        reset <= 0; @(posedge clk);
        repeat(500) @(posedge clk);

        $stop;
    end
endmodule
```

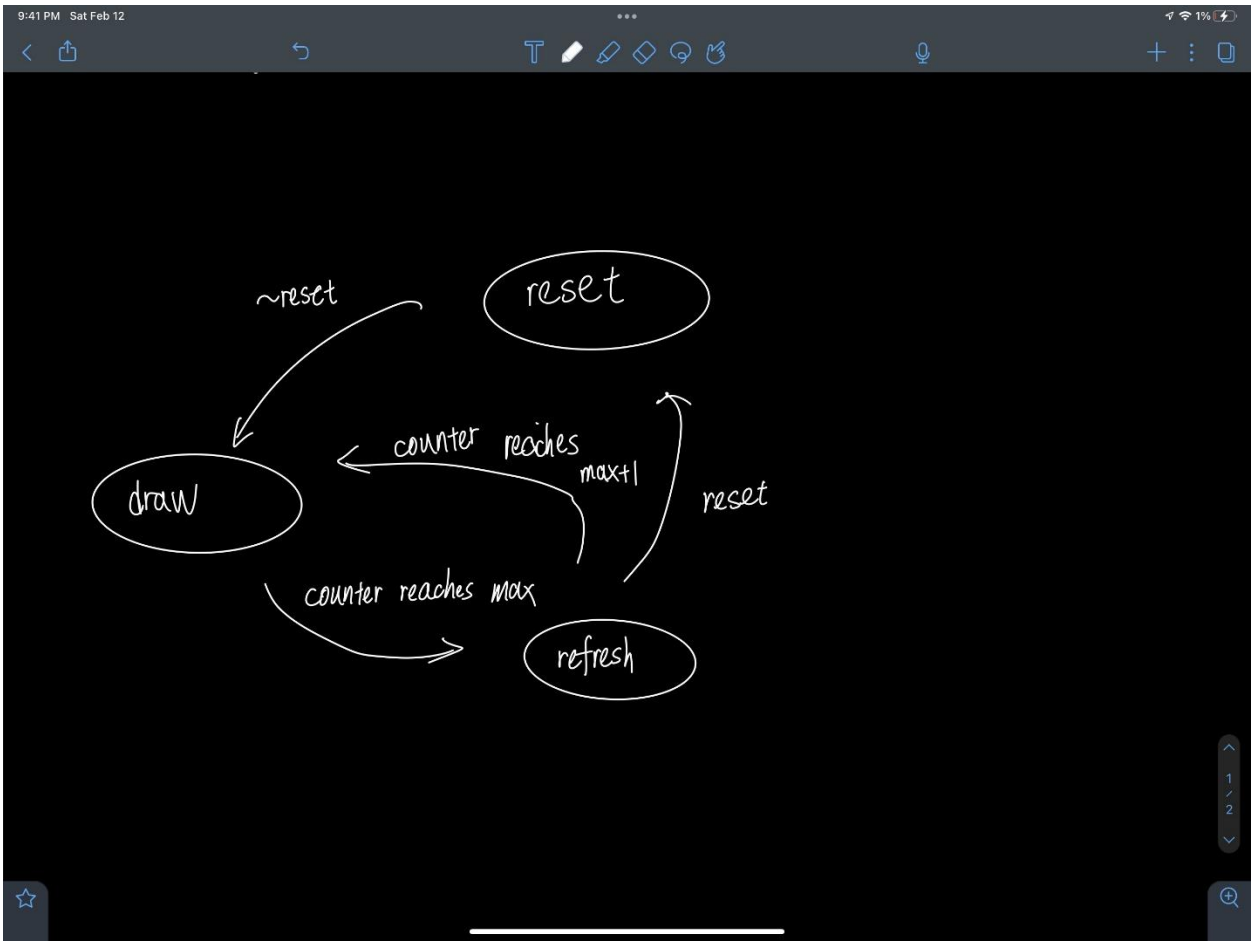
```
1  // Alan Li
2  // 02/12/2022
3  // EE 371
4  // Lab #3
5
6  // DE1_SoC takes 4-bit KEY, 10-bit SW as input and return 7-bit HEX, 10-bit LEDR,
7  // 8-bit VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS as output
8  // This serves as top-level module.
9  module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
10     VGA_R, VGA_G, VGA_B, 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 [9:0] x;
36     logic [9:0] x0, x1;
37     logic [8:0] y;
38     logic [8:0] y0, y1;
39     logic frame_start;
40     logic pixel_color;
41
42
43     //////// DOUBLE_FRAME_BUFFER ////////
44     logic dfb_en;
45     assign dfb_en = 1'b0;
46     ////////////////////////////////////
47
48     VGA_framebuffer fb(.clk(CLOCK_50), .rst(1'b0), .x, .y,
49              .pixel_color, .pixel_write(1'b1), .dfb_en, .frame_start,
50              .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .VGA_VS,
51              .VGA_BLANK_N, .VGA_SYNC_N);
52
53     // draw lines between (x0, y0) and (x1, y1)
54     // line drawers takes CLOCK_50, SW[9], x0, y0, x1, y1 as input parameter clk, reset, x0, y0, x1, y1, x, y
55     // abd return x and y respectively
56     line_drawer lines (.clk(CLOCK_50), .reset(SW[9]),
57              .x0, .y0, .x1, .y1, .x, .y);
58
59     // draw an arbitrary line
60
61     //1,1 12,5 works
```

```
    //2,2 2,100 works(vertical)
    //2,2 100,2 works(horizontal)
    //0,0 640,480
    //2,2 24,10 does not work
    //5,5 60,25 does not work
    //10,10 120,50 does not work
    //100,100 150,300 does not work
    //0,0 640,480 does not work(no lines at all)
    //0,0 600,400

    always_comb begin
        if (SW[9]) begin
            x0 = 0;
            y0 = 0;
            x1 = 0;
            y1 = 0;
            pixel_color = 1'b1;
        end else begin
            x0 = 100;
            y0 = 100;
            x1 = 150;
            y1 = 300;
            pixel_color = 1'b1;
        end
    end
endmodule

// DE1_SoC_testbench tests all expected behavior that will encounter
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 clk;
    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;

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

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

    initial begin
        SW[9] <= 1; repeat(10)  @(posedge clk);
        SW[9] <= 0;             @(posedge clk);
                    repeat(500) @(posedge clk);
        $stop;
    end
endmodule
```

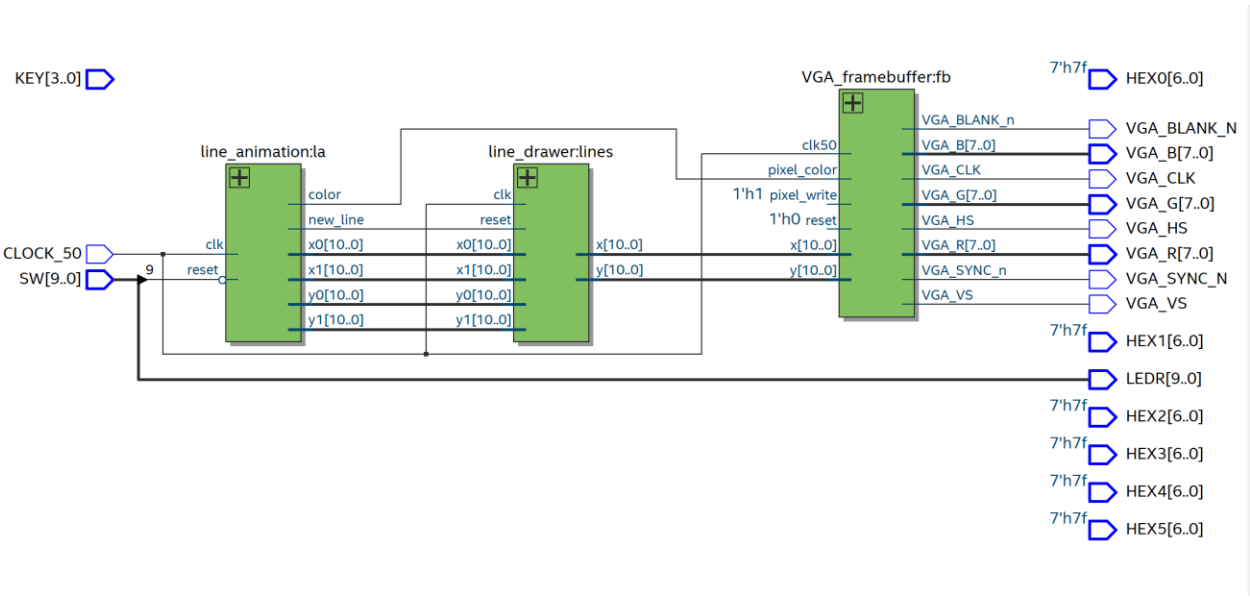
Task2

Section 1: Procedure

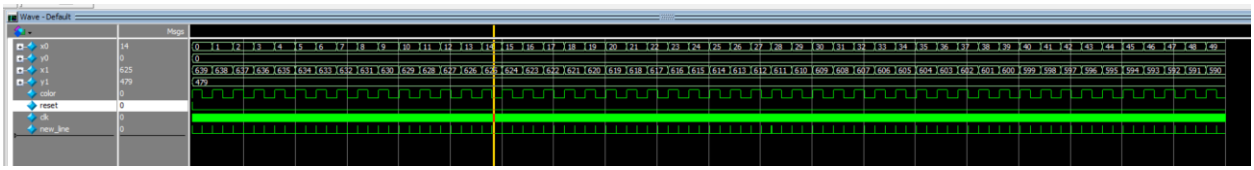
I'm using the same lilne_drawer module as task1, for task2 I added a line_animation module that make the line move around the screen. The module periodically updates the start and end point. I implemented a module of a three-state FSM. The state diagram is given below.



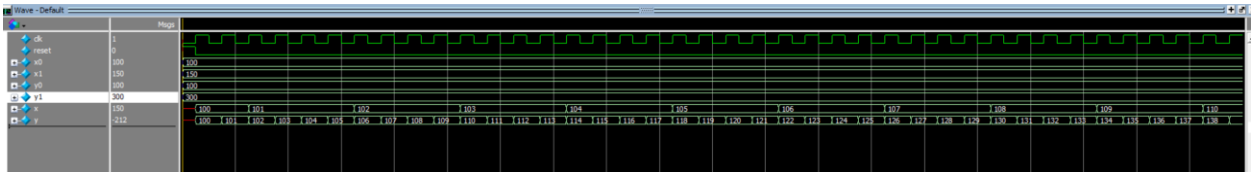
After the counter reaches the maximum, the drawing would be finished, the system goes to the state of refresh and redraws the same path but with color black. Before moving to the next drawing state, the system updates the new coordinate and makes sure that the coordinate stays within the bound. If the reset switch is enabled, the module goes immediately from drawing to refreshing. After clearing the screen, the system stays at reset until the reset switch is off.



Section 2: Results



From the simulation for line animation, you can see that the x0, y0 is getting updated. At the same time, the color is toggling between 1 and 0.



The simulation for line_drawer is the same as task1

Section 3: Appendix

```
// Alan Li
// 02/12/2022
// EE 371
// Lab #3

// line_drawer takes 10-bit x0, x1, 9-bit y0, y1 and clk, reset as input
// and return 10-bit x and 9-bit y as output

module line_drawer(
    input logic clk, reset,

    // x and y coordinates for the start and end points of the line
    input logic [9:0] x0, x1,
    input logic [8:0] y0, y1,

    // outputs cooresponding to the coordinate pair (x, y)
    output logic [9:0] x,
    output logic [8:0] y
    );

    // These registers are created following Bresenham's Algorithm
    logic signed [11:0] error;
    logic signed [23:0] e2;
    logic signed [10:0] deltaX;
    logic signed [9:0] deltaY;
    logic sx, sy;

    // xPointer and yPointer act as local registers and interate all the pixels on the line
    logic [10:0] xPointer;
    logic [9:0] yPointer;

    // Set deltaX, deltaY, sx and sy as constant parameters, they are used inside always_ff
    always_comb begin
        // absolute value for deltaX and deltaY
        if (x1 > x0) deltaX = (x1 - x0);
        else deltaX = -(x1 - x0);

        if (y1 > y0) deltaY = -(y1 - y0);
        else deltaY = (y1 - y0);

        if (x0 < x1) sx = 1;
        else sx = -1;

        if (y0 < y1) sy = 1;
        else sy = -1;
    end

    // in c code, there is a for loop to iterate all the pixels on the lines,
    // in VHDL, I used a flip-flop to iterate all the points
    always_ff @(posedge clk) begin
        // when reset, set all registers to initial value
        if (reset) begin
            error <= deltaX + deltaY;
            e2 <= 0;
            xPointer <= x0;
            yPointer <= y0;
        end
        // when xPointer and yPointer has not reached the destination coordinates
        // excute this else block
        else if (~((xPointer == x1) & (yPointer == y1))) begin
            e2 = 2 * error;
```

```
            xPointer <= x0;
            yPointer <= y0;
        end
        // when xPointer and yPointer has not reached the destination coordinates
        // excute this else block
        else if (~((xPointer == x1) & (yPointer == y1))) begin
            e2 = 2 * error;
            if (e2 >= deltaY) begin
                error <= error + deltaY;
                xPointer <= xPointer + sx;
            end

            if (e2 <= deltaX) begin
                error <= error + deltaX;
                yPointer <= yPointer + sy;
            end

            // both condition is true
            // without this if block, if e2 >= deltaY and e2 <= deltaX, only the first if block
            // will be executed and the second if block will be ignored
            if ((e2 >= deltaY) && (e2 <= deltaX)) begin
                error <= error + deltaX + deltaY;
                yPointer <= yPointer + sy;
                xPointer <= xPointer + sx;
            end


        end
    end
    assign x = xPointer;
    assign y = yPointer;
endmodule

// 1,1 12,5 works
// 100,100 150,300 works
// 2,2 2,100 works
// line_drawer_testbench tests all expected behavior that will encounter
module line_drawer_testbench();
    logic clk, reset;
    logic [9:0] x0, x1;
    logic [8:0] y0, y1;
    logic [9:0] x;
    logic [8:0] y;

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

    line_drawer dut(.clk, .reset, .x0, .x1, .y0, .y1, .x, .y);

    initial begin
        reset <= 1; x0 <= 100; y0 <= 100; x1 <= 150; y1 <= 300; @(posedge clk);
        reset <= 0; @(posedge clk);
        repeat(500) @(posedge clk);

        $stop;
    end
endmodule
```

```
// Alan Li
// 02/12/2022
// EE 371
// Lab #3

// This module enables the line to move around the screen
// If the reset is on, the system clears the screen and
// sets the line to the initial position

// line_animation take reset and clk as inputs and return x0, x1, y0, y1, color, new line as outputs
module line_animation (x0, y0, x1, y1, color, clk, reset, new_line);
   output logic [10:0] x0, y0, x1, y1;
   input logic reset, clk;
   output logic color, new_line;
   logic x1_inc;
   logic x0_inc;

   logic [9:0] count;  // counter used to add delay

   enum {res, refresh, draw} ps;

   // After the counter reaches the maximum, the drawing would be finished, the system goes to the state of refresh and redraws the same path but with color black.
   always_ff@(posedge clk) begin
      // draw the line
      if (ps == draw) begin
         color <= 1;
         // counter reaches max or reset, refresh screen
         // counter back to 0
         if (count == 1000 || reset) begin
            count <= 0;
            ps <= refresh;
            new_line <= 1;
         end
         // increases counter, keepdrawing
         else begin
            count <= count + 1;
            new_line <= 0;
         end
      end

      // refreshes the screen
      if (ps == refresh) begin
         color <= 0;
         // counter reaches maximum, next stage
         // counter back to 0
         if (count == 1001) begin
            count <= 0;
            new_line <= 1;
            if (reset)
               ps <= res;
            else
               ps <= draw;
            // updates the end points of the line
            if (x1_inc && x1 < 639)
               x1 <= x1 + 1;
            else if (x1 > 0)
               x1 <= x1 - 1;
            if (x0_inc && x0 < 639)
               x0 <= x0 + 1;
```

```
            x0_inc <= 1;
         if (x0 == 639)
            x0_inc <= 0;
      end
      else begin
         count <= count + 1;
         new_line <= 0;
      end
   end
   // reset the end points of the line
   // set counter to 0
   // prepare to draw
   if (ps == res) begin
      color <= 0;
      x0 <= 0;
      y0 <= 0;
      x1 <= 639;
      y1 <= 479;
      x0_inc <= 1;
      x1_inc <= 0;
      new_line <= 1;
      count <= 0;
      if (~reset)
         ps <= draw;
   end
end
endmodule

// tests the previous module
module line_animation_testbech ();
   logic [10:0] x0, y0, x1, y1;
   logic color, reset, clk, new_line;

   line_animation la (.*);

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

   initial begin
      reset <= 1; @(posedge clk);
      reset <= 0; @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  @(posedge clk);
                  repeat(100000)@(posedge clk);
      $stop;
   end
endmodule
```

```
1  // This module draws a line moving around the screen.
2  // If switch 9 is truned off, the system stops drawing,
3  // clears the screen and reset the line to the starting position
4
5
6  // DE1_SoC takes 4-bit KEY, 10-bit SW as input and return 7-bit HEX, 10-bit LEDR,
7  // 8-bit VGA_R, VGA_G, VGA_B, VGA_BLANK_N, VGA_CLK, VGA_HS, VGA_SYNC_N, VGA_VS as output
8  // This serves as top-level module.
9  module DE1_SoC (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, CLOCK_50,
10    VGA_R, VGA_G, VGA_B, 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 [10:0] x0, y0, x1, y1, x, y;
36    logic color, new_line;
37
38    // draws on the VGA buffer
39    VGA_framebuffer fb(.clk50(CLOCK_50), .reset(1'b0), .x, .y,
40              .pixel_color(color), .pixel_write(1'b1),
41              .VGA_R, .VGA_G, .VGA_B, .VGA_CLK, .VGA_HS, .VGA_VS,
42              .VGA_BLANK_n(VGA_BLANK_N), .VGA_SYNC_n(VGA_SYNC_N));
43
44    // walks through the points to be drawn
45    line_drawer lines (.clk(CLOCK_50), .reset(new_line),
46              .x0, .y0, .x1, .y1, .x, .y);
47
48    // move the end points around the screen
49    line_animation la (.x0, .y0, .x1, .y1, .color, .clk(CLOCK_50), .reset(~SW[9]), .new_line);
50
51 endmodule
52
53 // tests the previous module
54 module DE1_SoC_testbench ();
55    logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
56    logic [9:0] LEDR;
57    logic [3:0] KEY;
58    logic [9:0] SW;
59
```

```
    // walks through the points to be drawn
    line_drawer lines (.clk(CLOCK_50), .reset(new_line),
              .x0, .y0, .x1, .y1, .x, .y);

    // move the end points around the screen
    line_animation la (.x0, .y0, .x1, .y1, .color, .clk(CLOCK_50), .reset(~SW[9]), .new_line);

endmodule

// tests the previous module
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 clk;
    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;

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

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

    initial begin
        SW[9] <= 0;             @(posedge clk);
        SW[9] <= 1;             @(posedge clk);
                    repeat(100000) @(posedge clk);

    end
endmodule

/*
// This module chooses at what frequency the clock is running
module clock_divider (clock, divided_clocks);
    input logic clock;
    output logic [31:0] divided_clocks;
    initial begin
      divided_clocks <= 0;
    end
    always_ff @(posedge clock) begin
      divided_clocks <= divided_clocks + 1;
    end
endmodule
*/
```