Khoa Tran

EE 371

April 23, 2021

Lab 2 Report

**Procedure**

**Task #1**

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

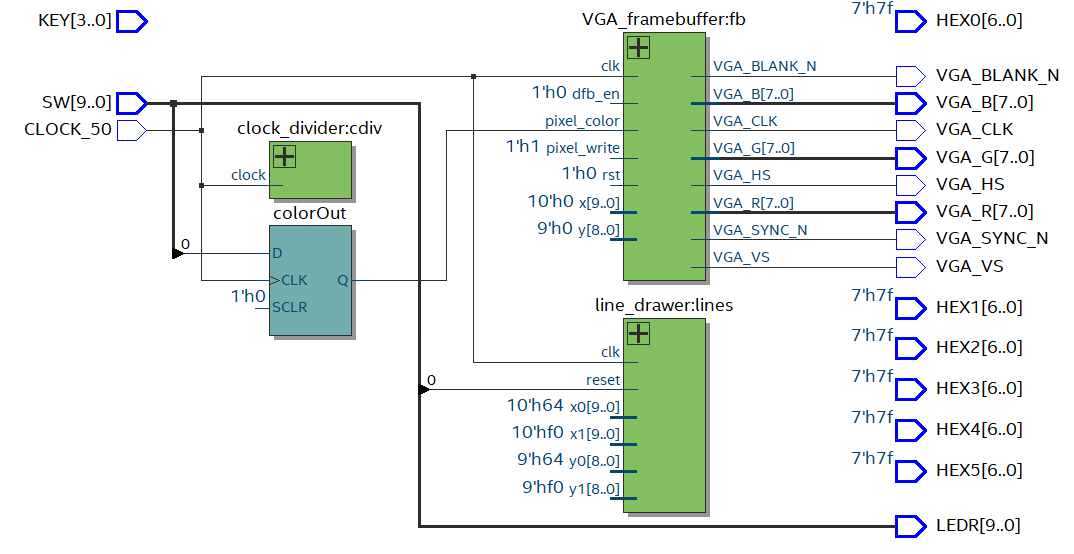
****

Figure 1: Block diagram for task 1

**Task #2**

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

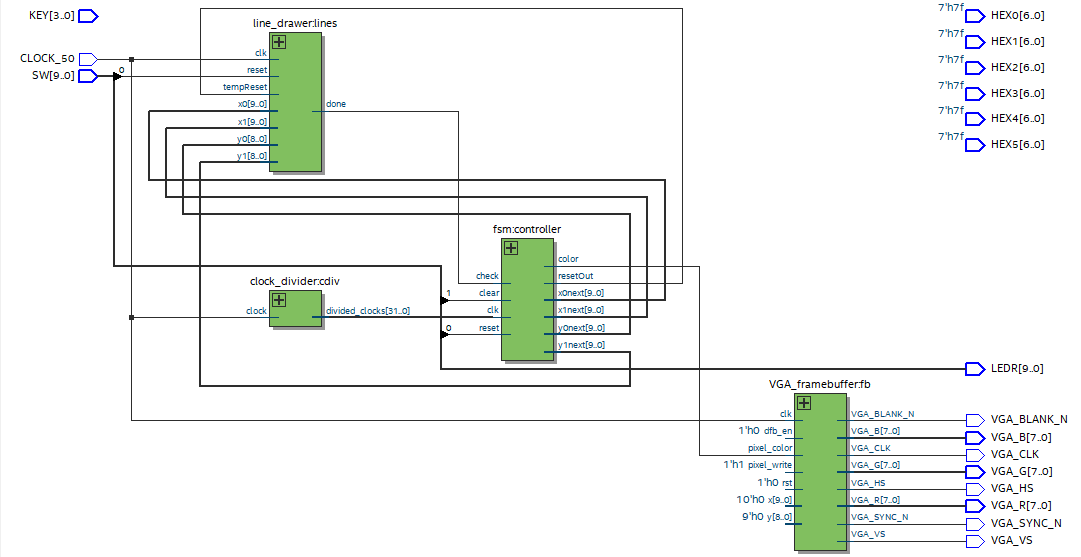


Figure 2: Block diagram for task 2

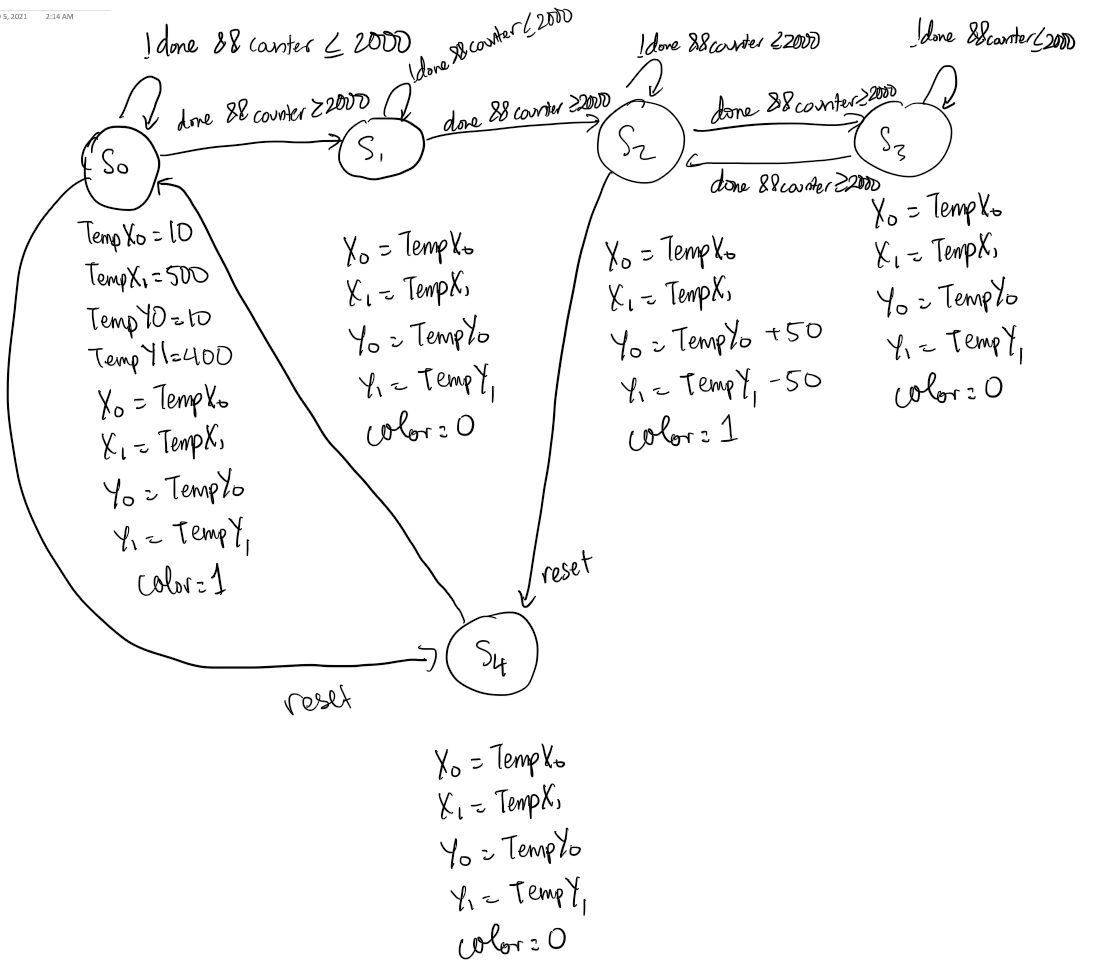


Figure 3: FSM for task 2

**Results**

**Task 1:**

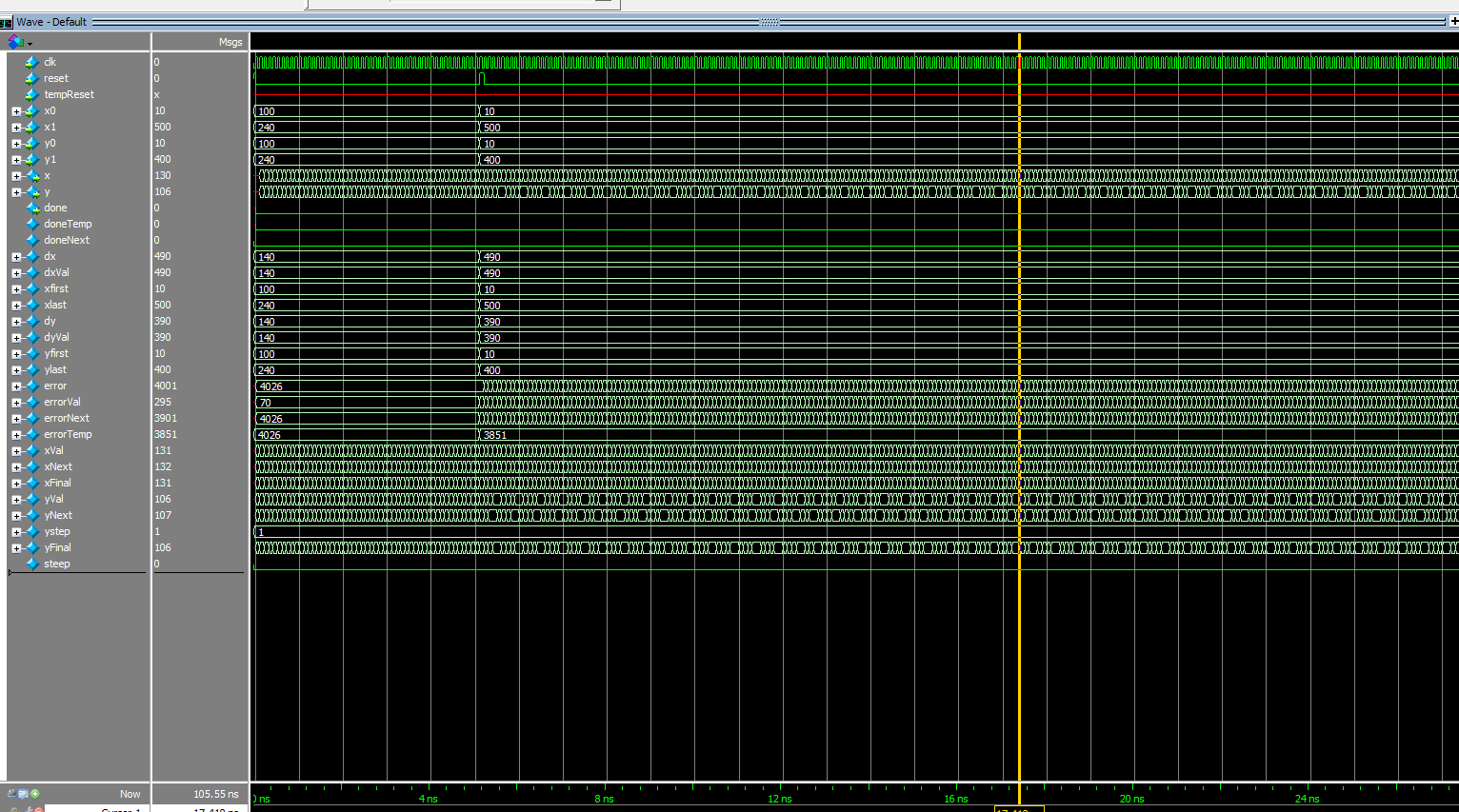
For the first part, I tested if the line\_drawer outputs the correct x and y values with the given starting and ending coordinates of (x0, y0) and (x1, y1), as you can see in figure 3, the x and y continuously outputs addresses for the VGA\_framebuffer to write to and as the coordinates change, the x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

Figure 4: Waveform simulation for line\_drawer for task 1

I also tested the simulation of the DE1\_SoC module. However, the results were very different as when I ran the code on the labsland, it worked according to the line\_drawer as the DE1\_SoC only sends the output of line\_drawer into the VGA\_framebuffer.

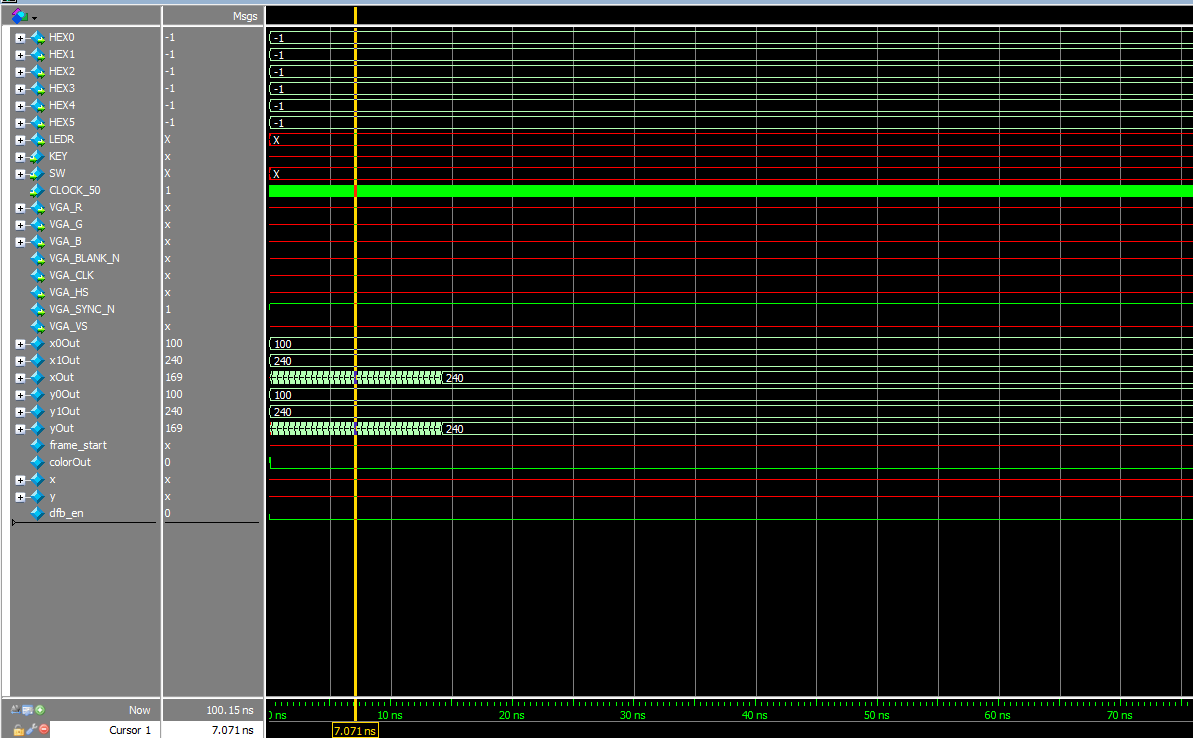


Figure 5: Waveform for DE1\_SoC of task 1

**Task 2:**

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

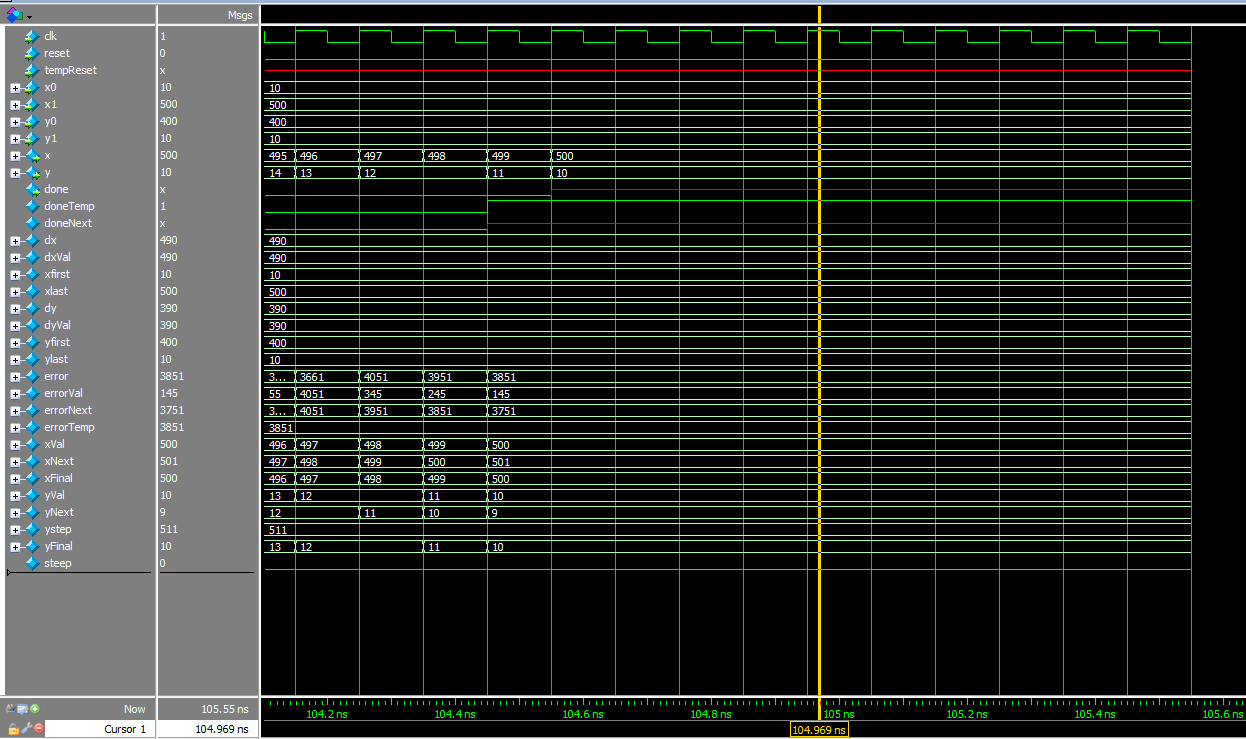


Figure 6: The waveform simulation generated by line\_drawer for task 2

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

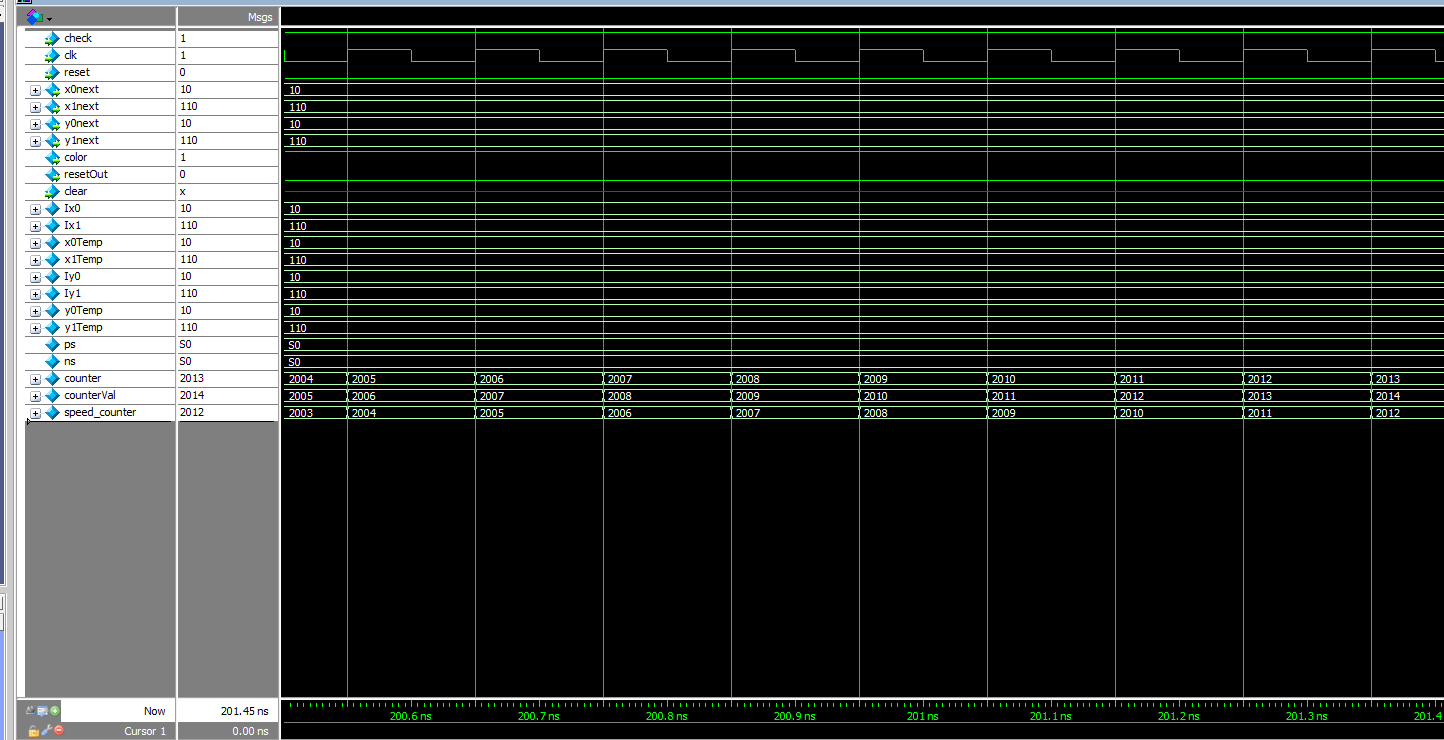


Figure 7: Waveform simulation generated by fsm for task 2

Once again, the DE1\_SoC simulation was incorrect as seen in figure 7 and x and y aren’t able to output anything. However, I have tested the fsm and the line\_drawer and couldn’t figure out why it wasn’t loading the output of line\_drawer into the VGA.

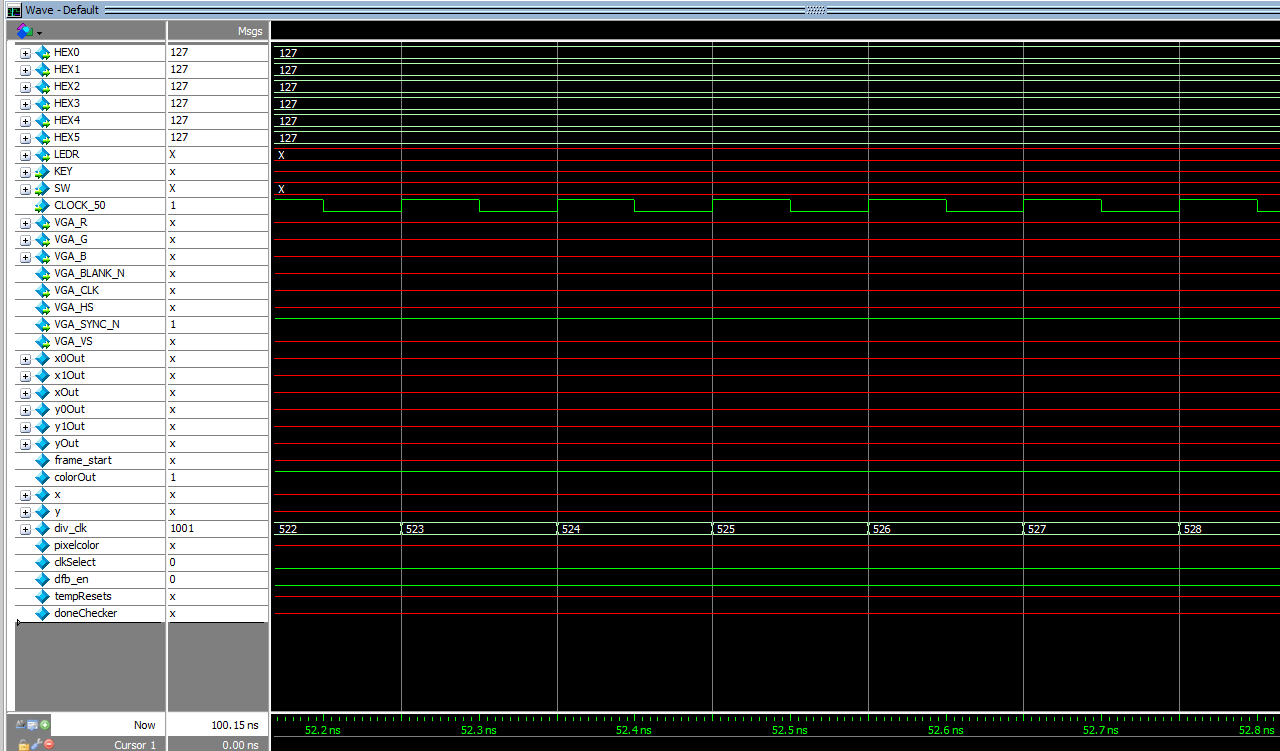
****

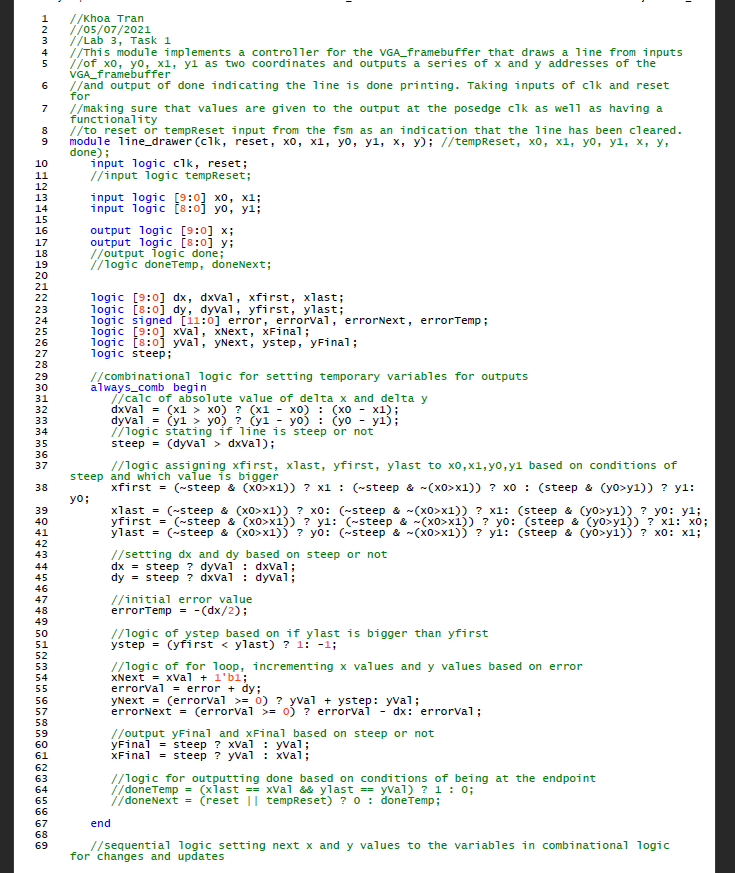
Figure 8: The waveform simulation generated by DE1\_SoC for task 2

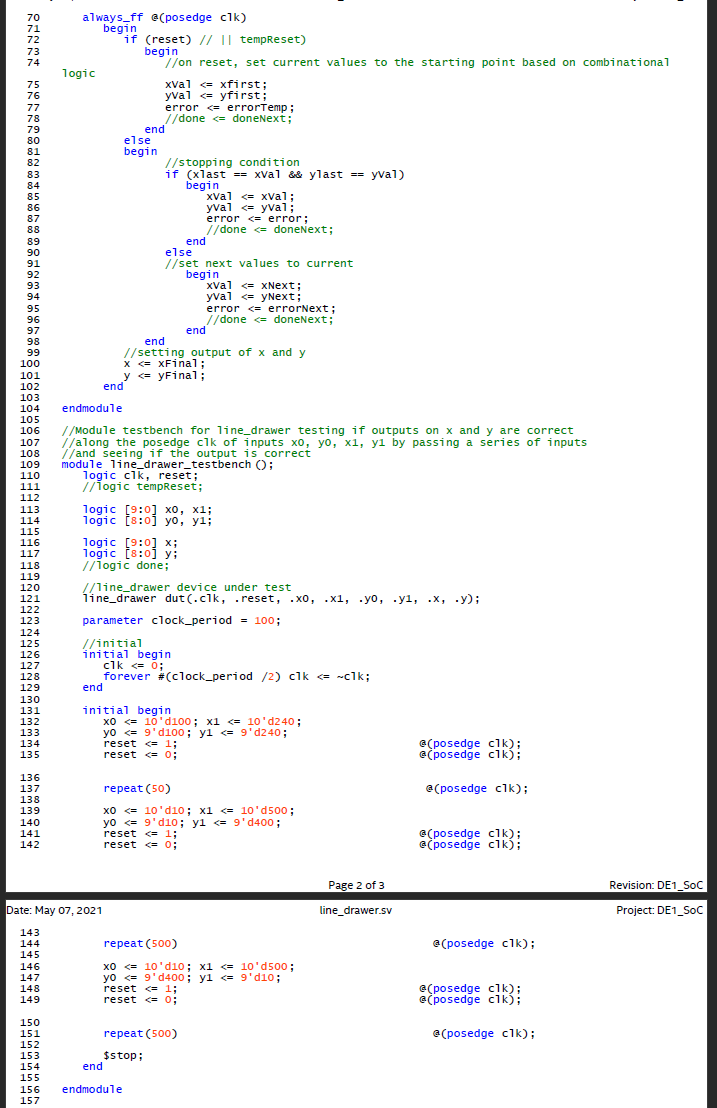
**Final Product**

The overarching goal of this project was to design a module that outputs a line given a starting and ending coordinate using Bresenham’s algorithm and as well as animate the line by outputting the line and clearing it and incrementing starting and ending coordinates to continue the process and make it seem like it is animating through the screen. I wasn’t able to accomplish task 2 as my DE1\_SoC wouldn’t transfer the outputs of line\_drawer into the VGA\_framebuffer. However, I was able to implement line\_drawer and an fsm that controls the color that the VGA\_framebuffer writes as I had the right idea but couldn’t execute properly. Overall, I was able to learn a tremendous amount from this lab based on implementing C code in SystemVerilog and even though I wasn’t able to animate the line, I am pleased with being able to implement line\_drawer.

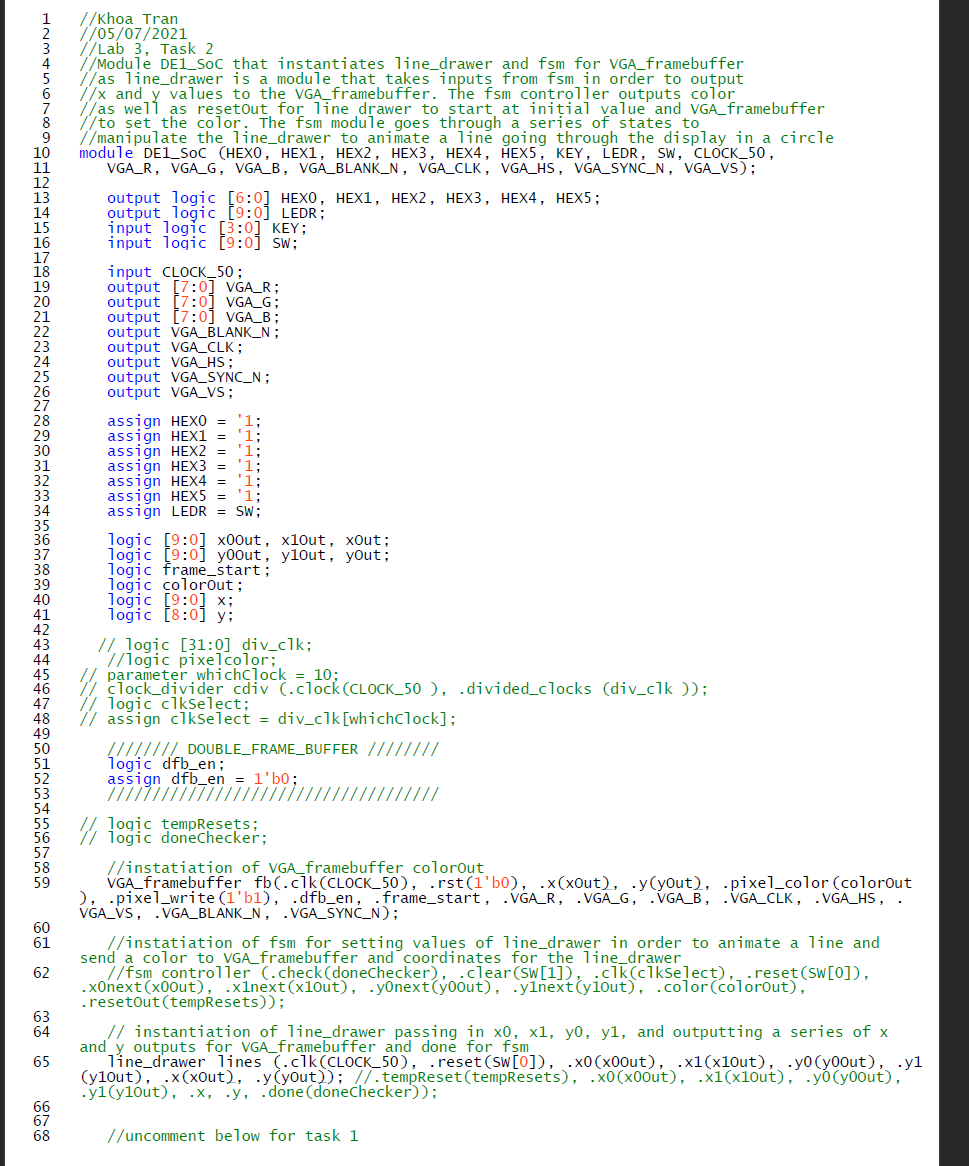
**Appendix: SystemVerilog Code**

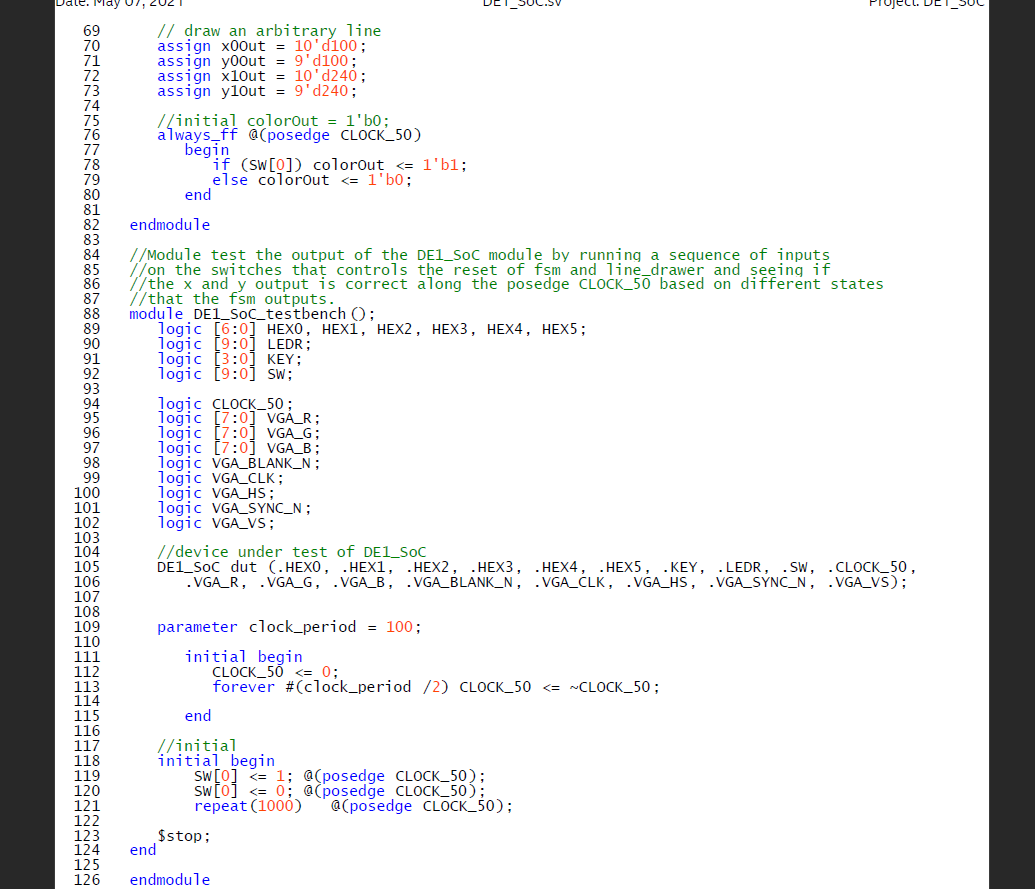
1. **Line\_drawer.sv (task 1)**



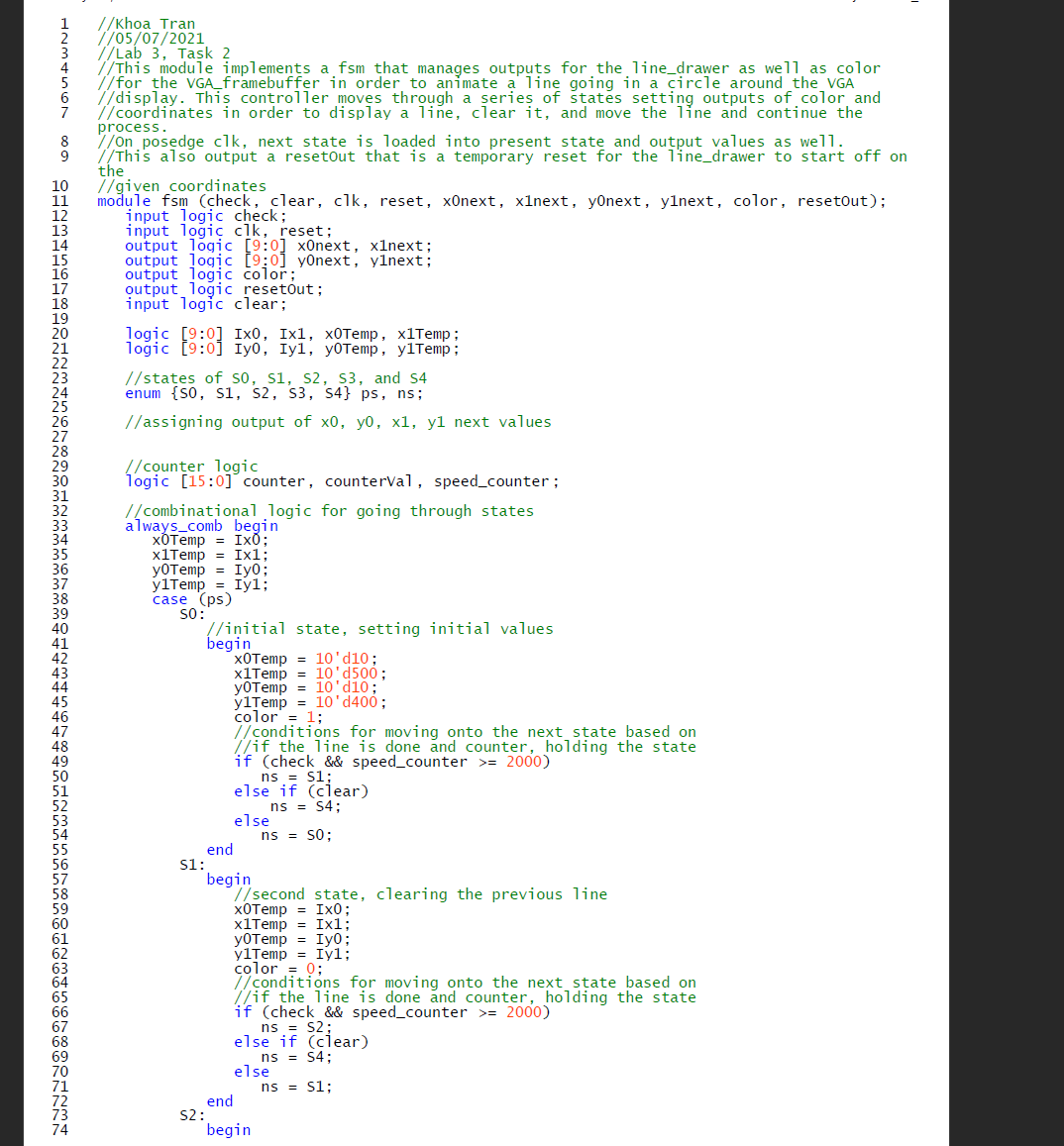
****

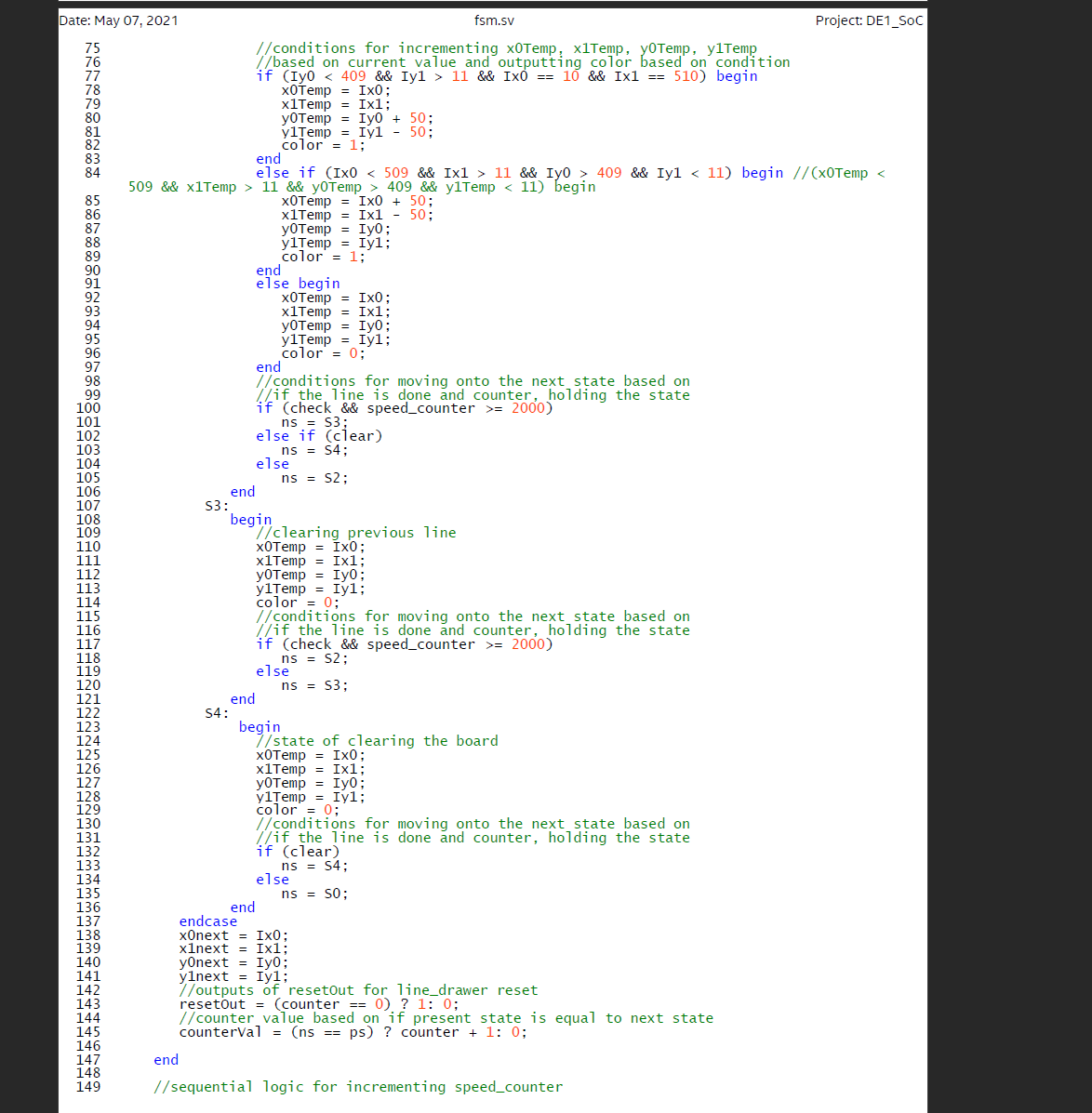
1. **DE1\_SoC.sv (task 1)**

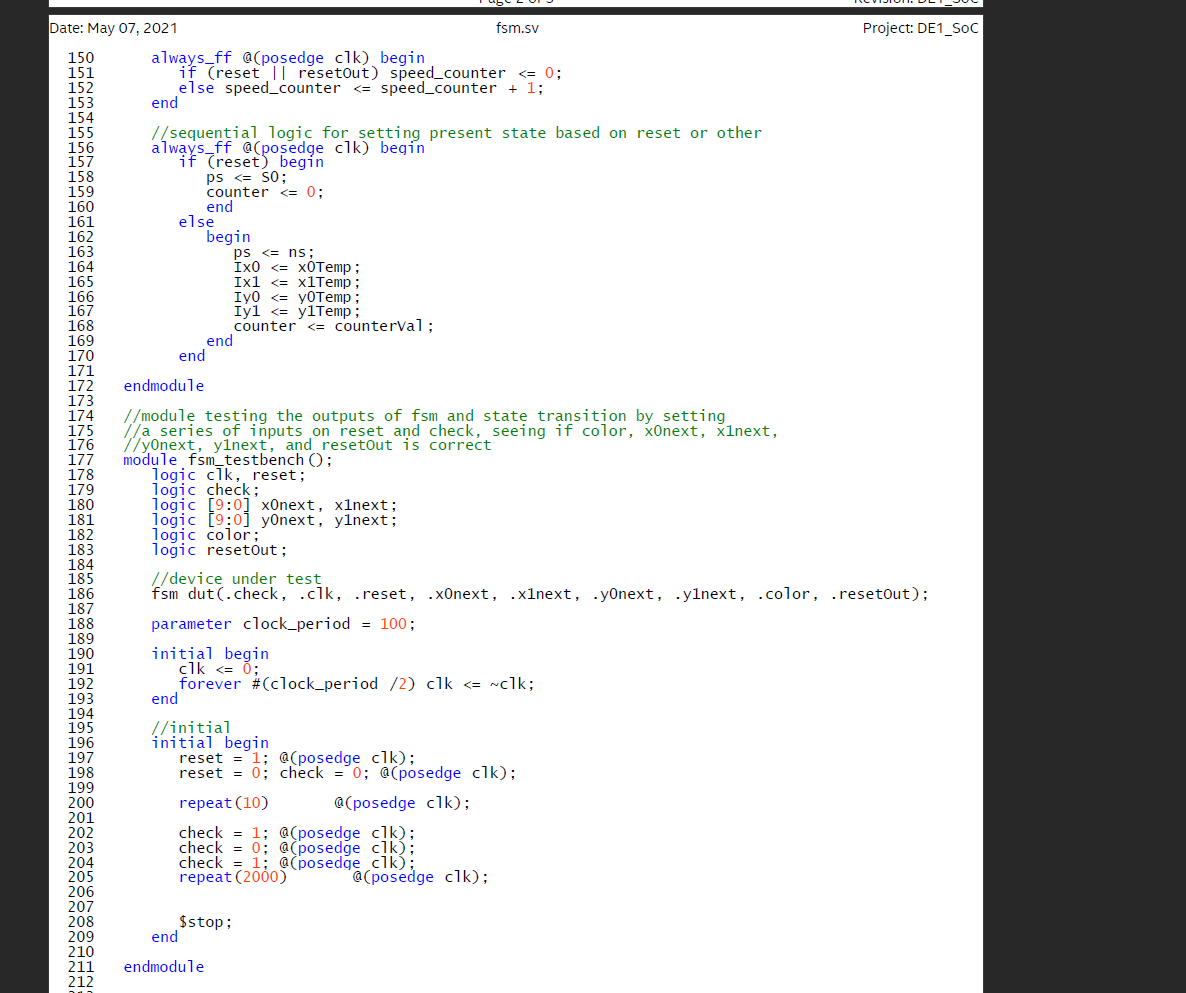


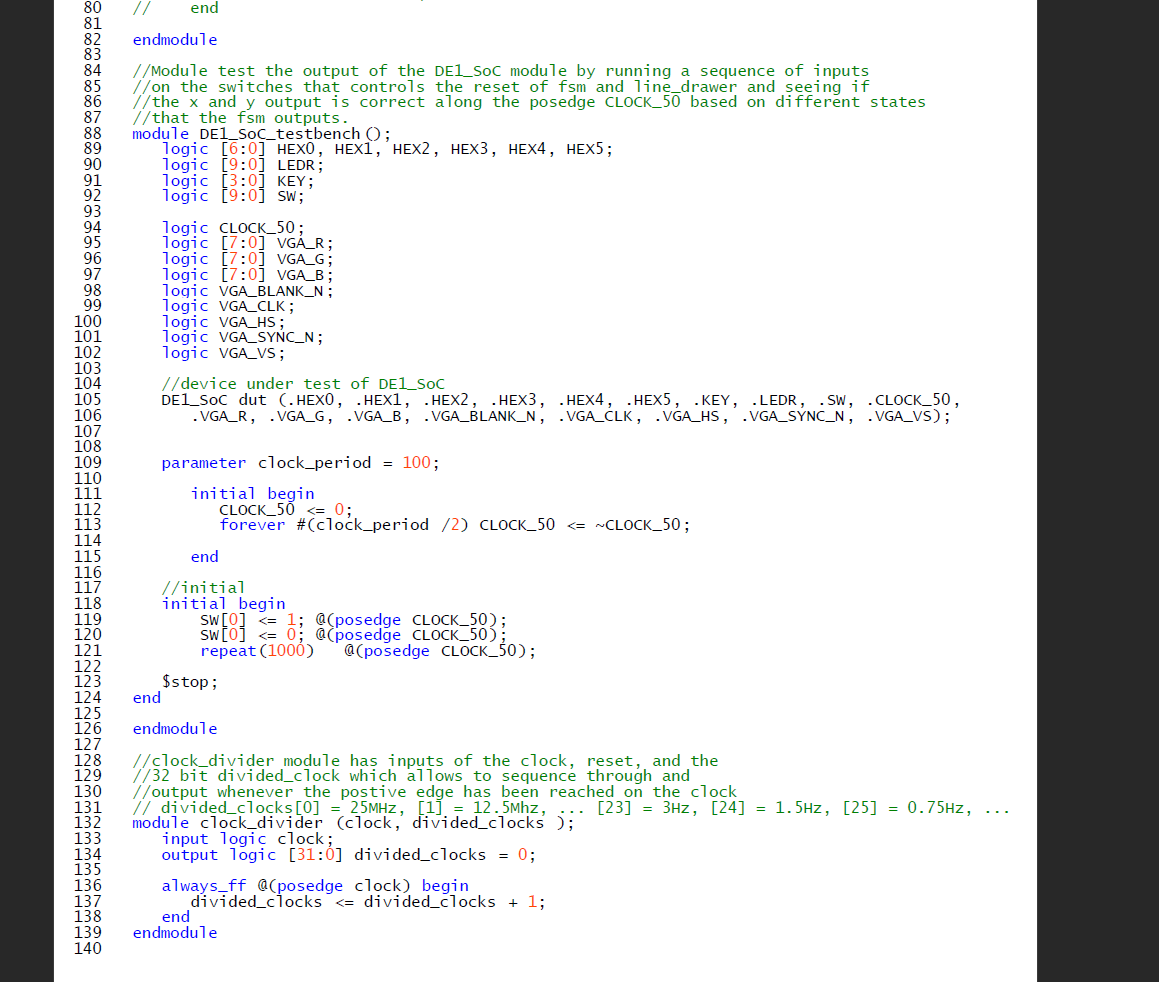
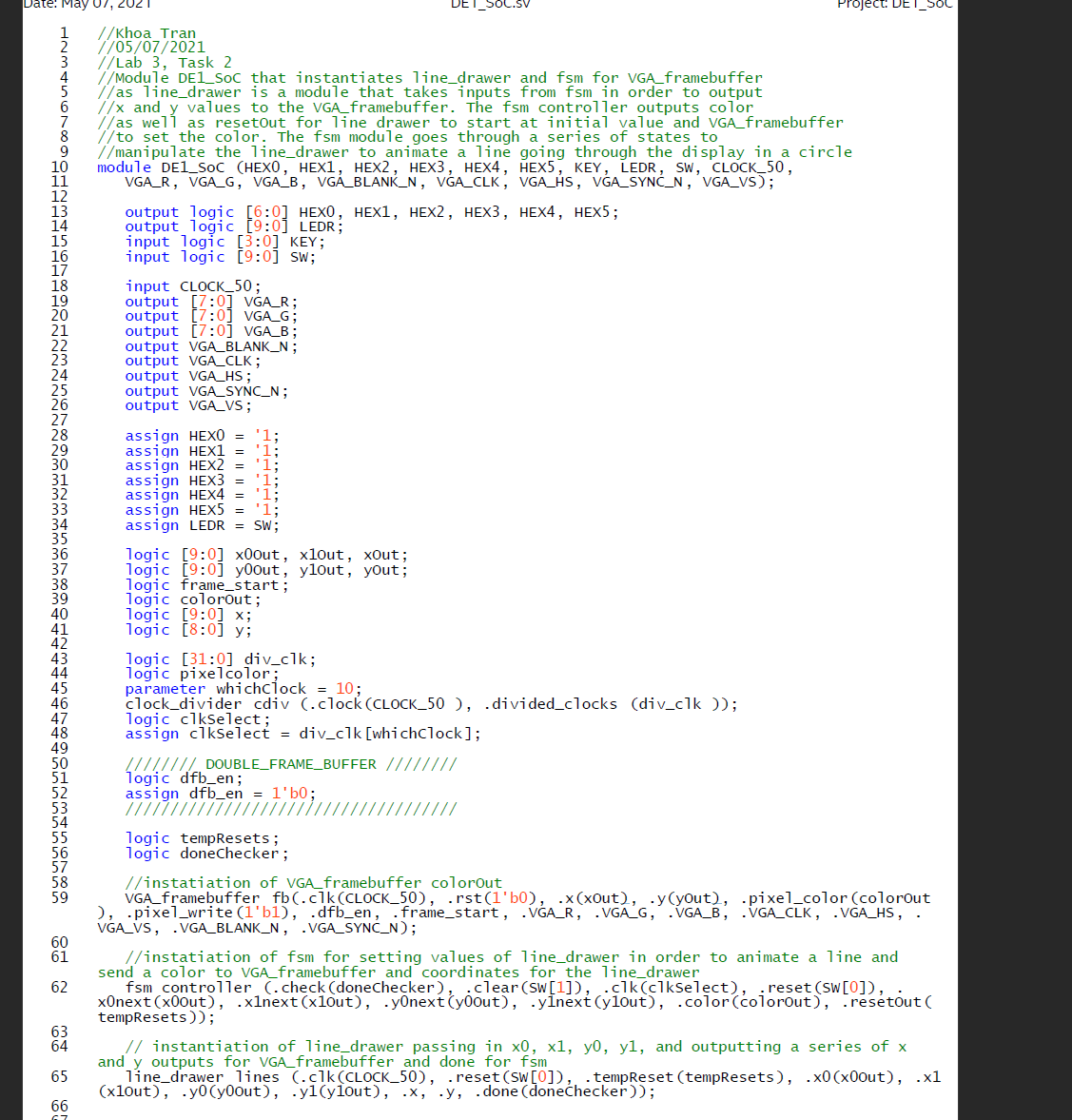


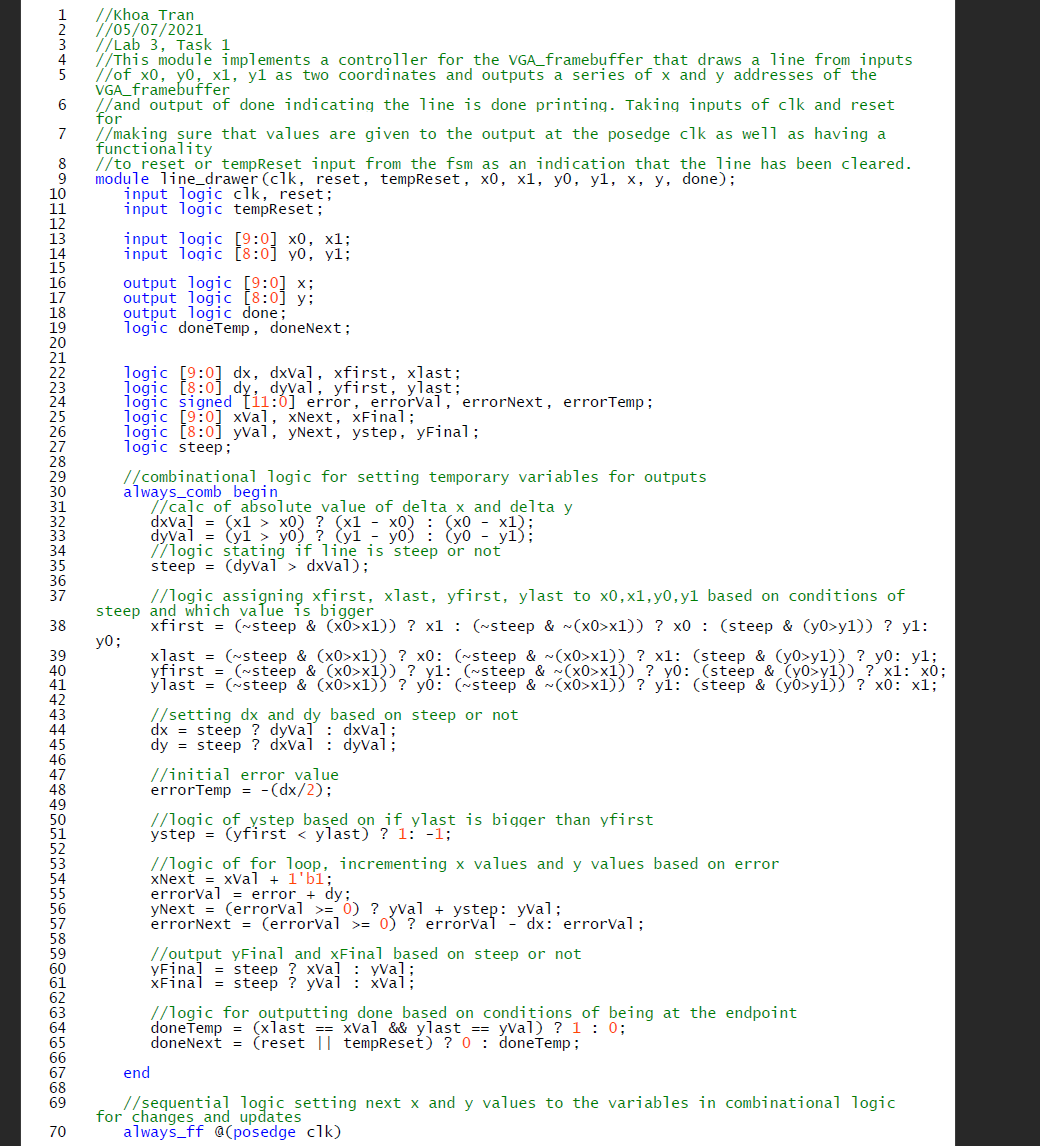
1. **Fsm.sv (task 2)**

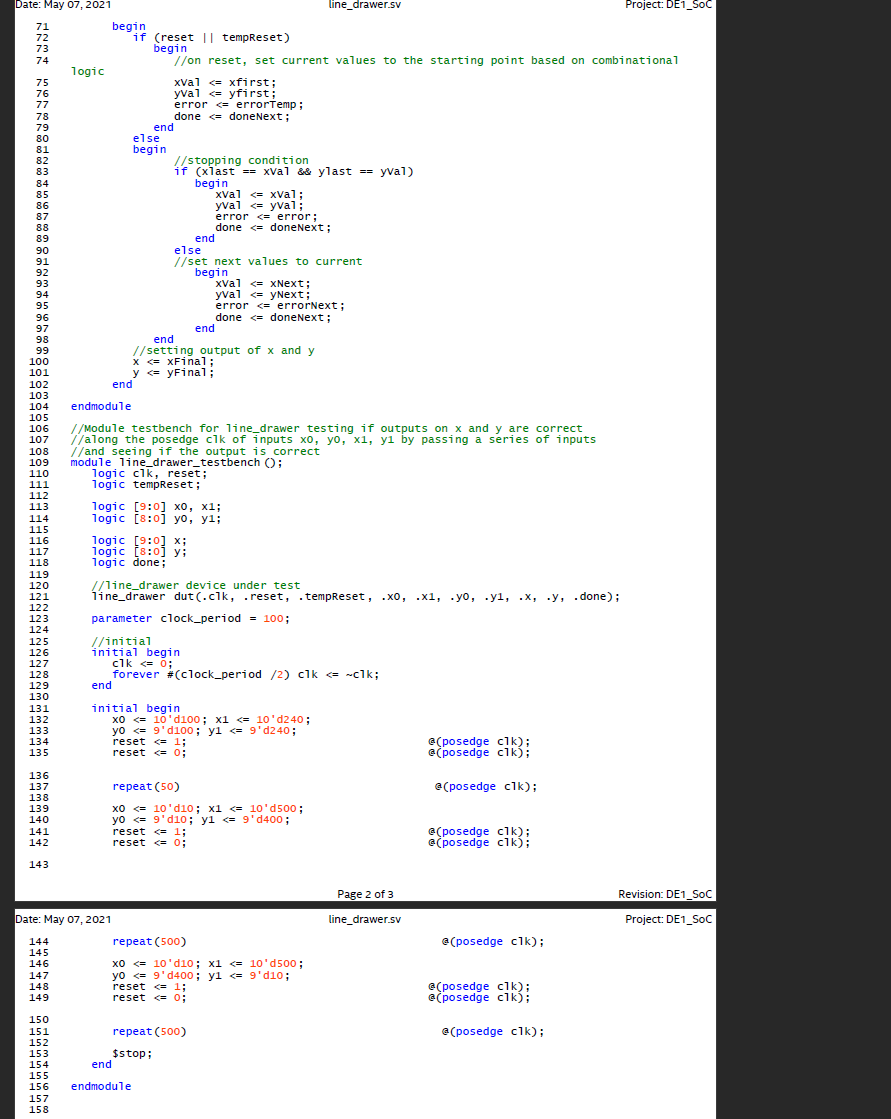
****





1. **DE1\_SoC.sv (task 2)**
2. **Line\_drawer.sv (task 2)**

****

****