Charles Tung Fang

EE 371

2019/10/30

Professor Rania Hussein

Lab 3 Display Interface

Procedure

I approached the problem by understanding the pixel buffer and Bresenham's algorithm at the very top level. After reading the specs, I examined the sudo code to make sure I know what is going on in the algorithm and the relationship among x0, y0, x1, y1, and error. The major components in the algorithm is using error to determine if one should move down or up on the y axis while x is being iterated. Once I grasp the main takeaway of the algorithm by reading a University of Hawaii's explanation, the rest had become easier since they were just translating java code to SystemVerilog.

Task 1 is simply hardware installment and was very straightforward. Task 2 is asking us to modify the line_drawer.sv file in order to implement Bresenham's line algorithm. There are eight cases in total to handle: The first four are drawing a flat line, drawing a steep line, drawing a flat line from the end point, and drawing a steep line from the end point. The other four cases are the similar four cases but the difference is that the lines are inverted. I developed my code similar to the University of Hawaii's demonstration. In my line_drawer.sv file, I first attained the dx and dy representing a line's width and height. Then I determine if I need to draw from the start point or the end point. Later I use logics x_sign and y_sign to see if the direction my line going is forward or backward. I created a for_loop module and pass in all the information to carry out the iteration part.

In my for_loop module, I have an alwayss_ff block to iterate depending on the line_drawer's logics I passed in. The most important detail of this module is change of error because it determines if x or y axis should increment or decrement by one to get closer to the destination coordinate. Once the iteration is over, I then choose to output the right x and y depending on my dx and dy, which gives me some successful results.

Task 3 is implementing a reset function that clear the screen and create an animation to move the lines. I first implement a reset function by drawing every pixel black at reset (when switch 9 is activated). I then use a clock divider and my line algorithm to create a simple animation. I change input x0, y0, x1, y1 coordinates in order to attain four different lines. The result successfully output a trapezoid.

Results

My result was successfully shown on both ModelSim and the FPGA board. In line_drawer testbench (Figure 1), one can easily see how the data is being set and pass in to the for_loop module. I used this testbench to make sure that for_loop module is receiving the right calculation as well as the inputs that I expected.

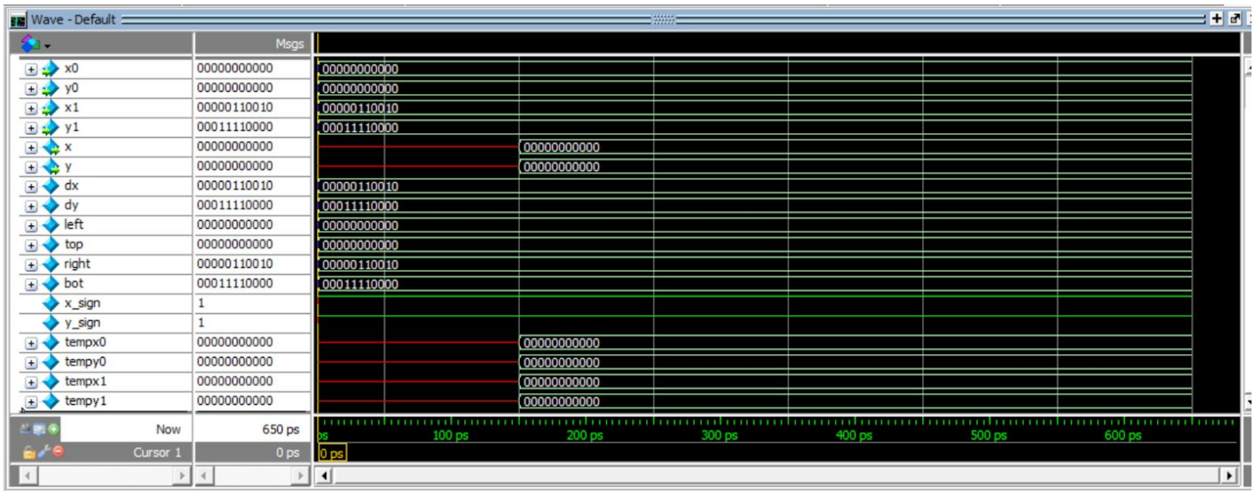


Figure 1. ModelSim for line_drawer module

The more advanced functions are tested in the for_loop ModelSim. We can see how the system iterate through x values until the destination and update y value by adding or subtracting the direction logic (1 or -1) if and only if the error has gone out of range. The error I designed also update the value once y value has been added one(Figure 2).

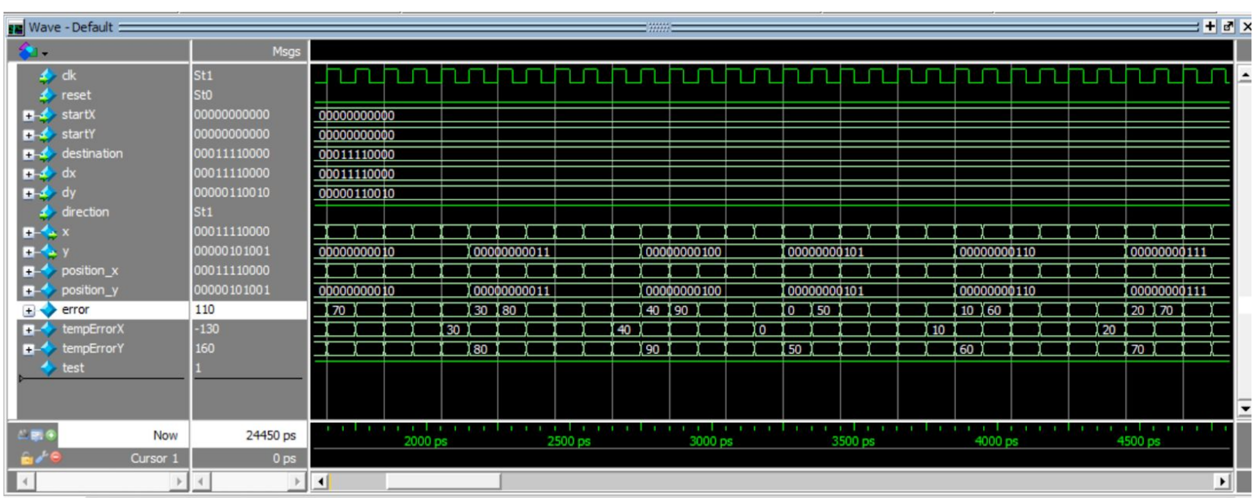


Figure 2. ModelSim for for_loop

The actual design of task 2 and 3 can be seen in Figure 3~5. Figure 3 shows the most basic situation of drawing a flat line from (0,0) to (240, 50). Figure 4 shows the other case of drawing a steep line from (0,0) to (240, 50).

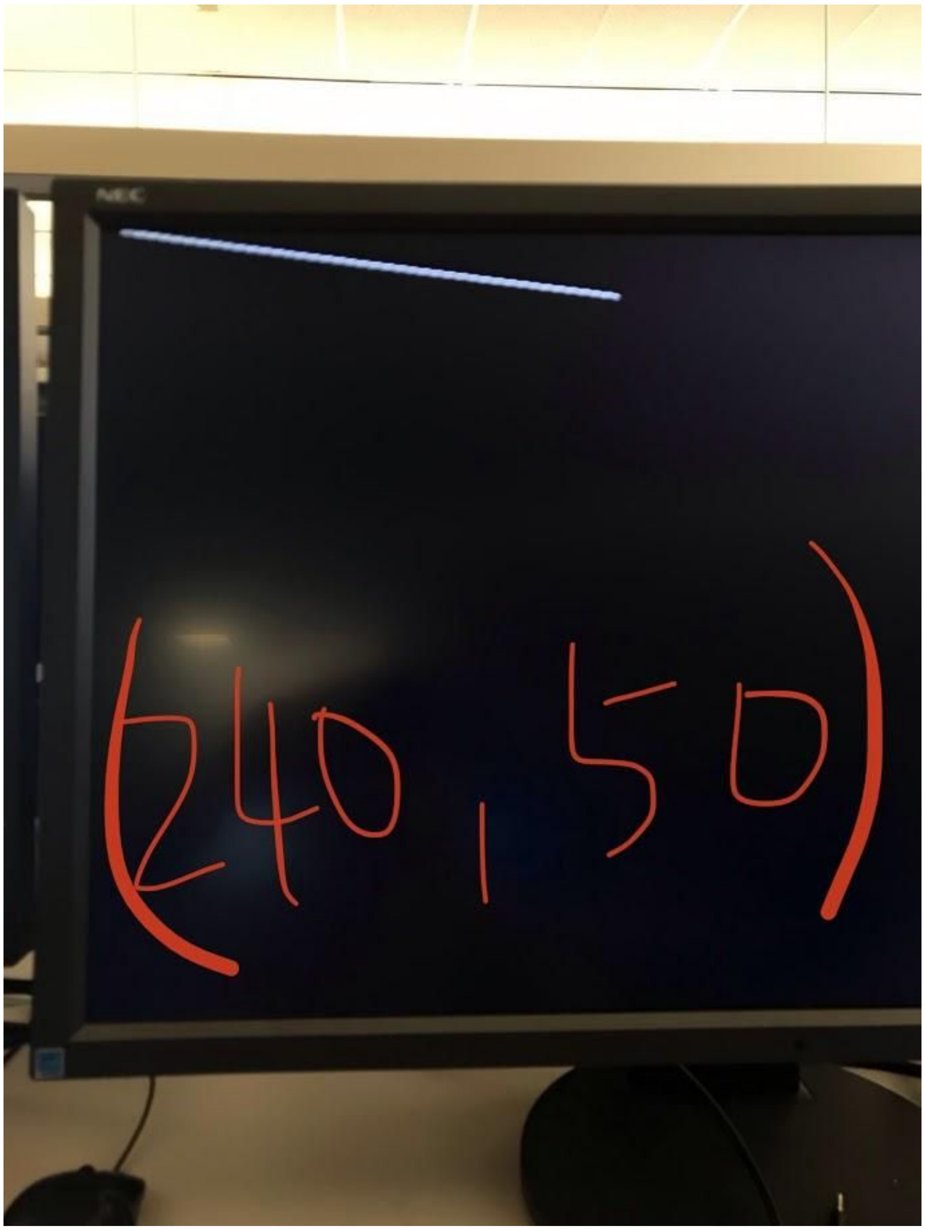


Figure 3. Drawing line from (0,0) to (240, 50)

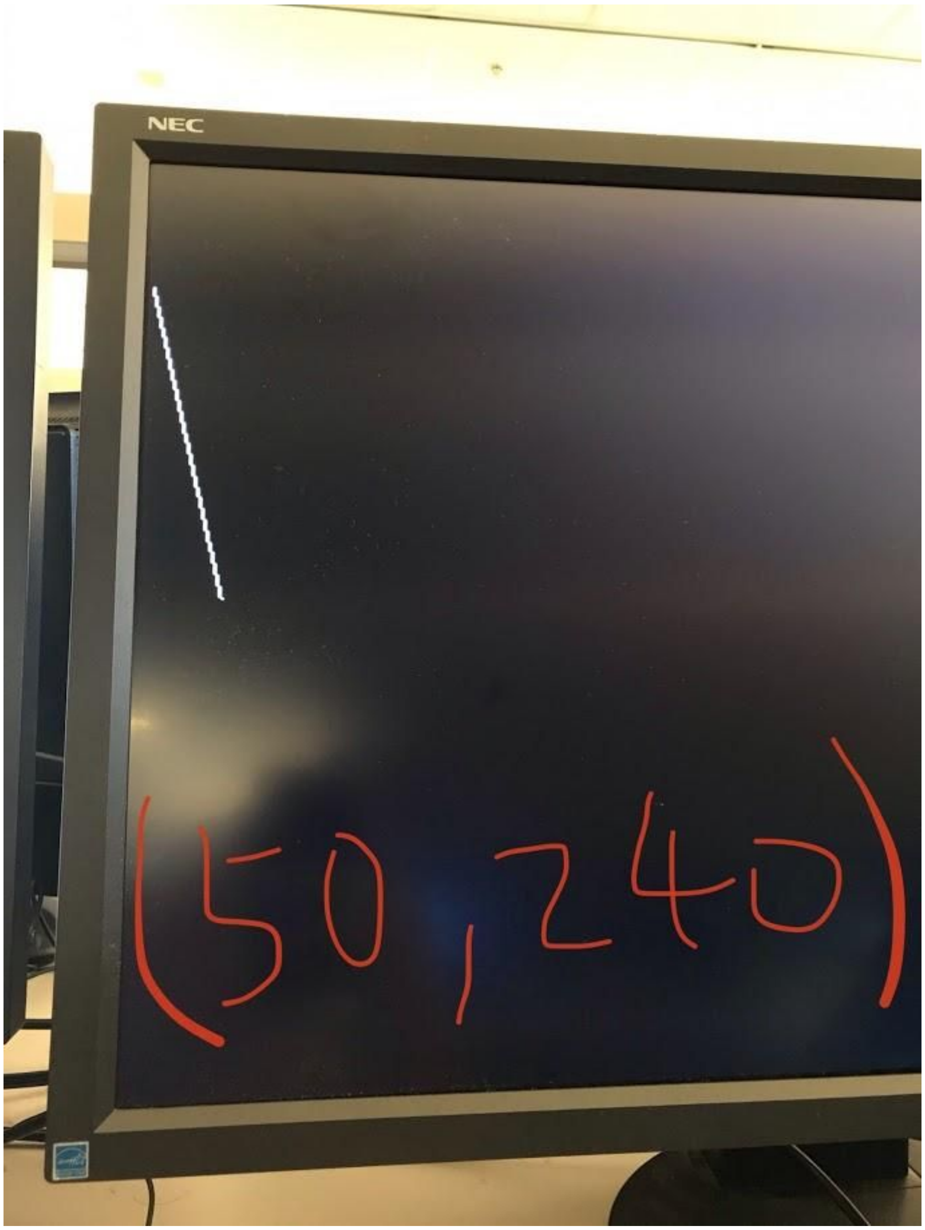


Figure 4. Drawing line from (0,0) to (50, 240)

By activating the animation (KEY[0]) and change the input data(x0, y0, x1, y1), I am able to change the resulting trapezoid animation. (Figure 5 & Figure 6)

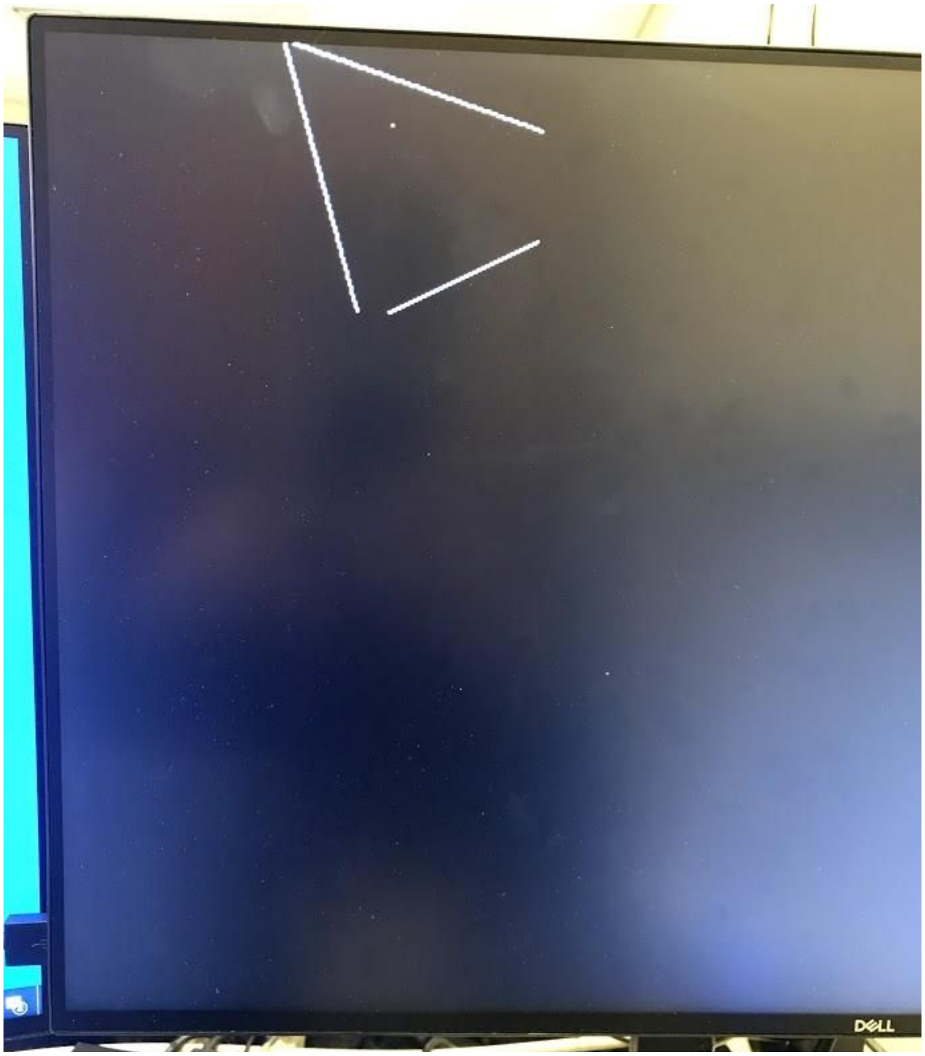
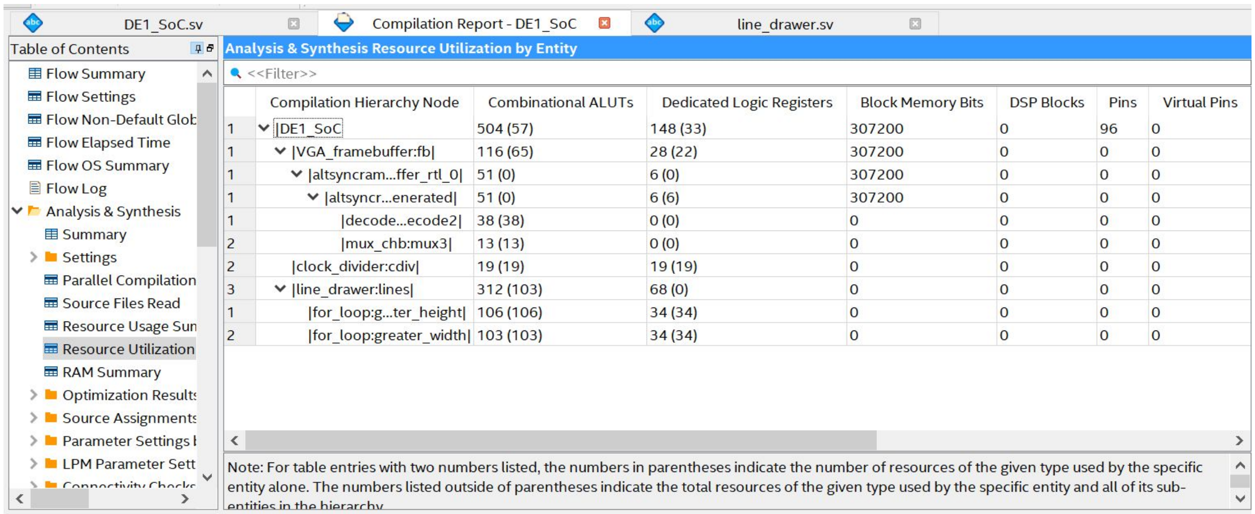


Figure 5. Process of animating Trazopid 1

Figure 6. Resulting Trapezoid 2

On the other hand, the Resource Utilization by Entity page (Figure 7) shows that the size of the program is 504+148 = 652



Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins
1	|DE1_SoC	504 (57)	148 (33)	307200	0	96	0
1	|VGA_framebuffer:fb|	116 (65)	28 (22)	307200	0	0	0
1	|altsyncram...ffer_rtl_0|	51 (0)	6 (0)	307200	0	0	0
1	|altsyncr...enerated|	51 (0)	6 (6)	307200	0	0	0
1	|decode...ecode2|	38 (38)	0 (0)	0	0	0	0
2	|mux_chb:mux3|	13 (13)	0 (0)	0	0	0	0
2	|clock_divider:cdiv|	19 (19)	19 (19)	0	0	0	0
3	|line_drawer:lines|	312 (103)	68 (0)	0	0	0	0
1	|for_loop:g...ter_height|	106 (106)	34 (34)	0	0	0	0
2	|for_loop:greater_width|	103 (103)	34 (34)	0	0	0	0

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy

Figure 7. Resource Utilization by Entity page

Problems Faced and Feedback

I think the most difficult part of this lab is understanding the line algorithm and implementing it with SystemVerilog. There is a huge difference between computer science code and VHDL so I really can't think the design in java. I wasn't so sure about doing the revert cases but I later then figure out I simply need to run multiple for loop and then pick the right output at the end. The animation part was a bit challenging. I spent a lot of time trying to do fancy animation but failed. Eventually I have a straightforward but fine animation that portray my design could draw all kinds of lines

I really don't like this lab because it was very much harder than the others. It was very hard to use modelsim because model sim can only give you the value of the logic. There were often error with the line when I actually upload my FPGA. Plus every compilation takes a long time so I feel very frustrated whenever I see the line is not what I expected. However, I feel rewarded when I resolve all the problems and pass the demo.