Ishan Dane NetID: idane
Ismail Memon NETID: ishmemon

Lab 5

Design Procedure

In lab 5, we explored and programmed the VGA terminal of the DE1_SoC. We learned how to use and generate signals from the DE1_SoC to create graphical displays and animations onto a VGA board. We learned how to manipulate the pixel buffer, which allowed us to change what was displayed on the screen. In Task 1, we downloaded and set up the starter files for the VGA interface, and made sure the initial default settings worked on Labsland. In Task 2, we analyzed Bresenham's line drawing algorithm and understood how it works. We then implemented it into hardware and connected it to the VGA, so we could draw lines on the VGA by defining the start and end points. In Task 3, we took the line drawing algorithm a step further and used it to create an animation. We created multiple lines and displayed them one at a time to make an animation. We also worked to create a reset function which, if activated, would turn the screen fully black.

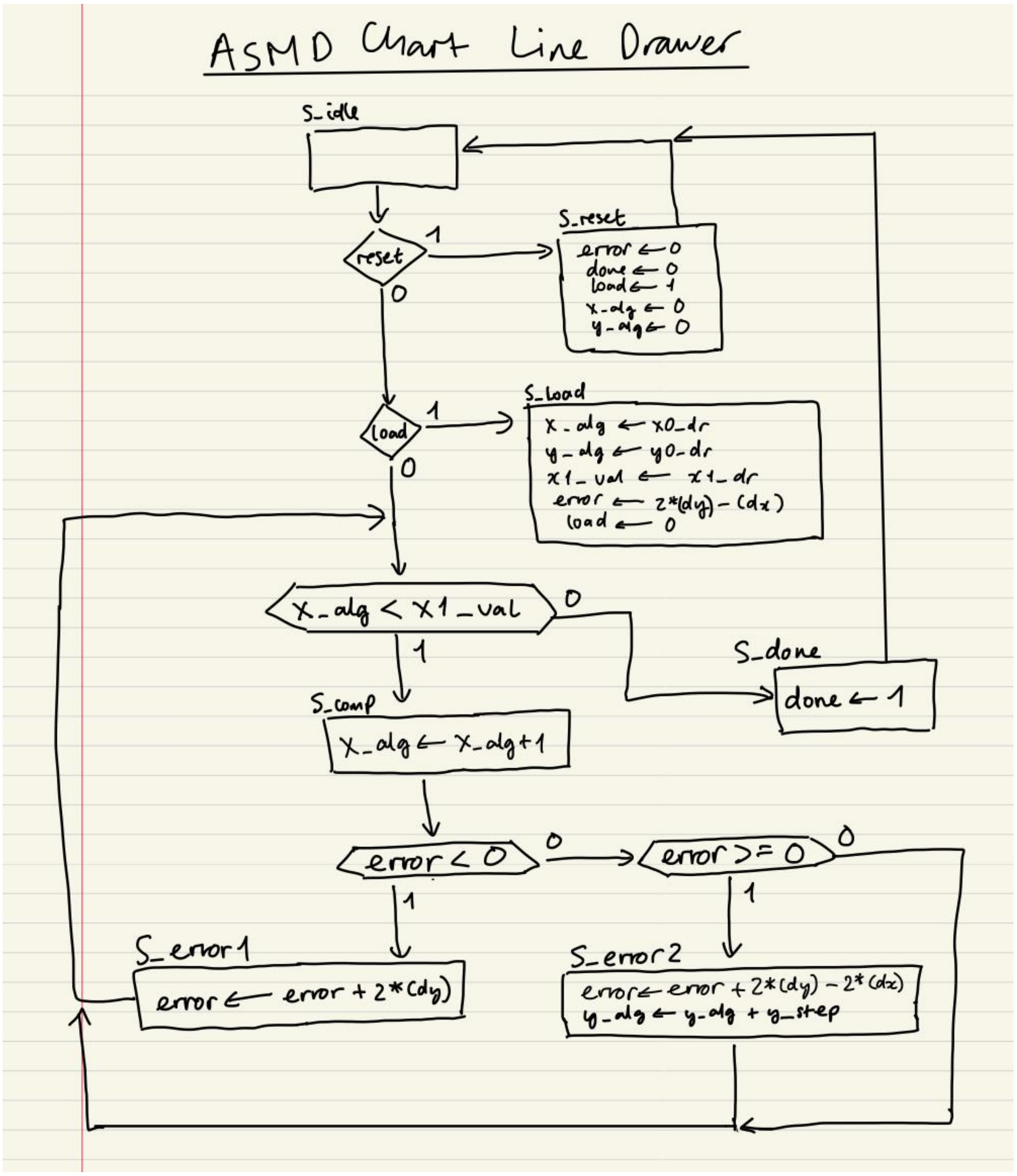
Task 1

In Task 1, we just downloaded the starter files and made sure they worked on the Labsland. There were no deliverables for this part.

Task 2

In Task 2, we were asked to draw a line on the VGA display of the FPGA board by implementing Bresenham's Line algorithm in SystemVerilog. The basis of this algorithm is an error variable which would decide whether or not to increment the y value provided at each clock edge. How much to increment the y value would depend upon the relationship between y0 and y1: If y0 < y1, the incremental value for y, " y_step" would be set to 1. If y0 > y1, y_step would be set to -1 so that y decreases. If y0 = y1, y_step = 0 so that the y value stays the same. The steep line case and left/right case was applied to the coordinates. In our design, for the steep line case, if the absolute value of dy was greater than the absolute value of dx, we would swap x coordinates with y coordinates and dx and dy. The absolute values of dx and dy were assigned by using the larger x/ y coordinate and subtracting the smaller x/y coordinate. For our left/right case, if x0 was greater than x1, we would swap x0 with x1 and y0 with y1 and draw as normal. The final x0 , y0, x1, y1 coordinates inputted into the algorithm were given the name x0_dr , y0_dr, x1_dr and y1_dr respectively. To ensure there was no confusion with output x and y variables, x0 and y0 post case conditions were stored into x_alg and y_alg and used as such in the algorithm. When reset is true, the algorithm sets error, done, x_alg and y_alg to 0 and load to 1 to start the load process. When load is 1, the algorithm would then load the values error, x_alg and y_alg and set load to 0 which would start the drawing process. In our design of the drawing process, we used the pseudocode as a foundation. We used an if statement where if x0_alg was less than x1_dr, the drawing process would continue. Conditions for error were created that would alter the error value or at the same time increment the y_alg value by y_step. Furthermore, x_alg would increment each time the if statement was accessed. At every clock edge, the x_alg and y_alg values were stored into output values x and y. However, for the steep line cases, y_alg was stored into output x and x_alg into output y of the system. The system would output the signal “done” once the if statement condition was no longer met and the line was finished drawing.

The ASMD diagram is shown below:



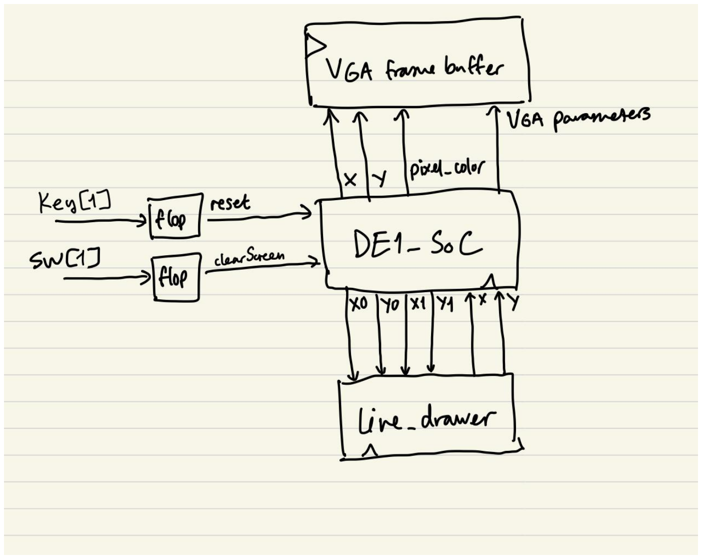
Task 3

In Task 3, what the objective was to make different lines appear at different points in time. We still had only one line drawer, so what we did was change the coordinates of the points that defined the line, so that new lines could be made. To allow for the animation and changing of lines to be actually seen by the user, this part used a slower clock so that the transitions could actually be seen. We also had to implement a reset, and we did so by drawing over what we already drew in the opposite color (black), in effect erasing what we made before.

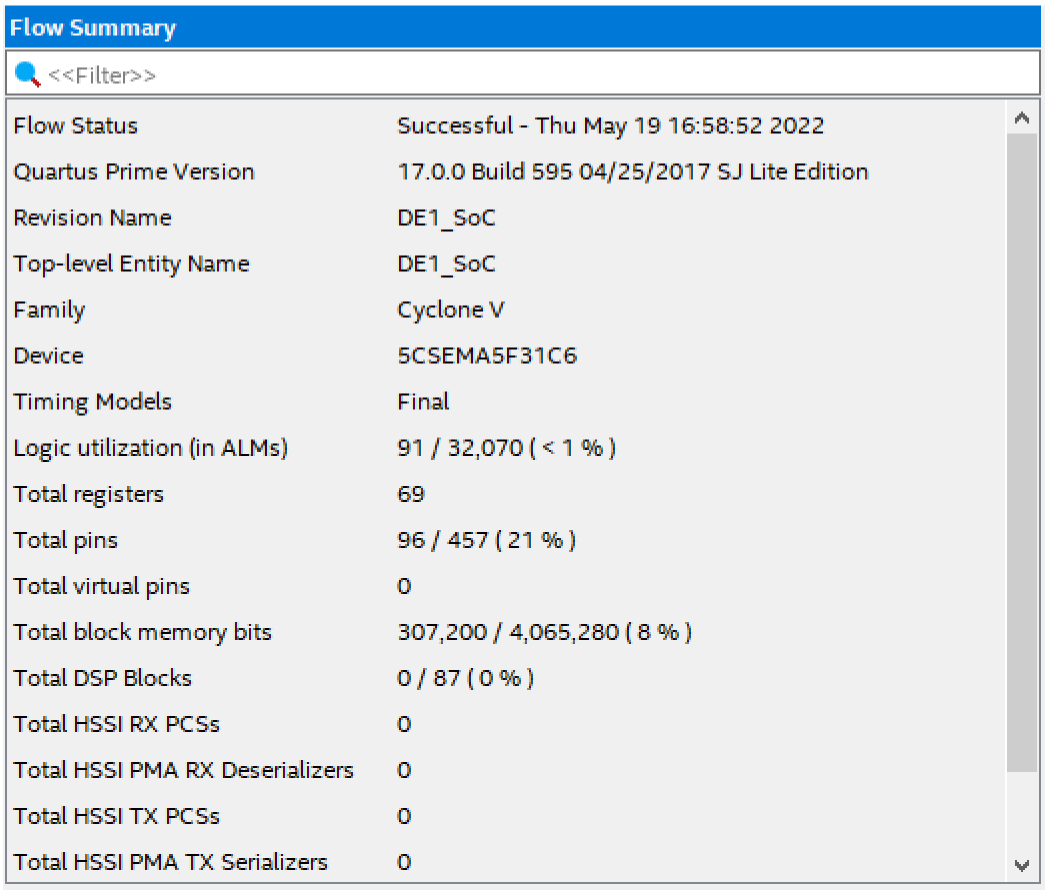
Overall System (Top level):

Our top level module is DE1_SoC.sv. In this module we made the animation of the lines that we displayed on the VGA. We first defined reset and screen clear to be keys 1 and switch 1, respectively. We chose these arbitrarily, since the key number and the switch number were the same we wouldn't have to move our mouse far apart to complete the two tasks. We also assigned the color to be white if the screen is not to be cleared (so we can draw) and black if it is to be cleared (so we can erase). Then we called the line_drawer module with a set x0, y0 and changing x1, y1 that we will be modifying later. Then we used the clock divider code from EE 271 to create a slower clock that could actually be seen by us when we do the Labs Land simulation. That is the clock the line_drawer is operating on. After that we create an always_ff block to implement the conditions. If reset is not pressed, we can increment our second point of the line (the first one stays the same) so that the lines move along and create the pattern we want. If reset is pressed, then we set the second point to (20, 20) and do not increment it (the point (20, 20) was arbitrary). Using these values for x1 and y1, the line drawer can make the line. We also instantiate the VGA_framebuffer which allows the whole animation to be completed and displayed on the VGA.

Block Diagram for our System



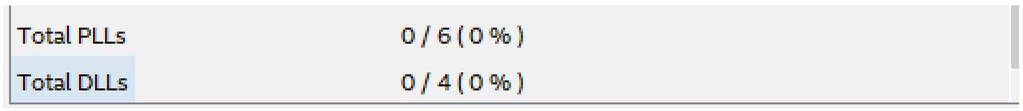
Flow Summary



Flow Summary

<<Filter>>

Flow Status	Successful - Thu May 19 16:58:52 2022
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	91 / 32,070 (< 1 %)
Total registers	69
Total pins	96 / 457 (21 %)
Total virtual pins	0
Total block memory bits	307,200 / 4,065,280 (8 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0



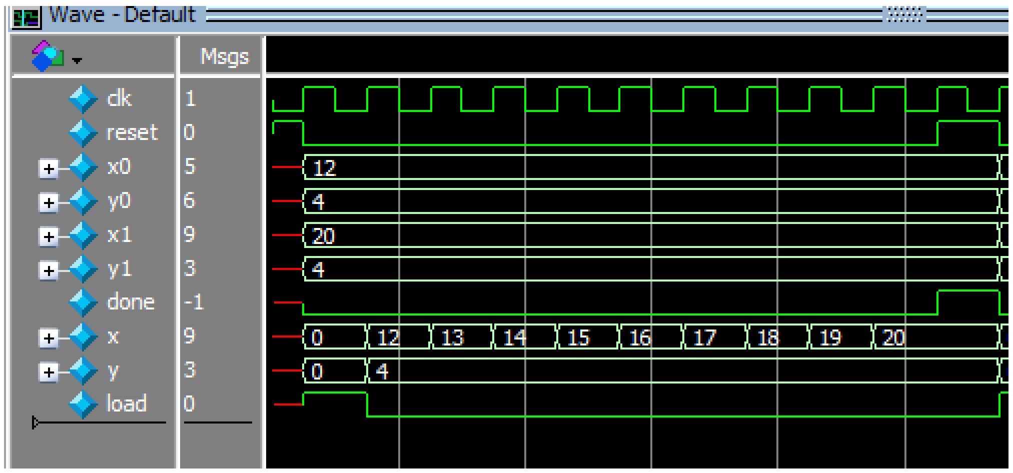
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Results

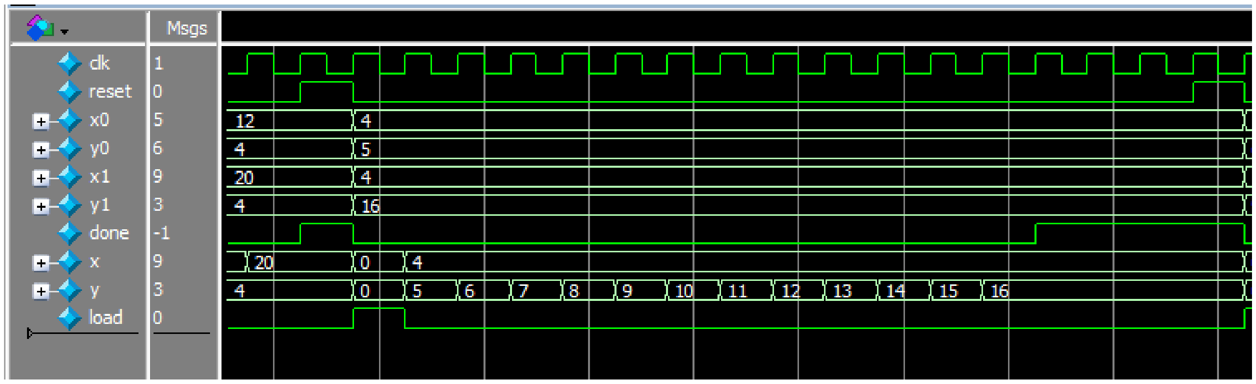
Task 2 Modelsim:

To test Task 2 , we made a testbench that would simulate every line case provided to the module through x0, y0, x1 and y1. We tested for the horizontal line case, vertical line case, gradual diagonal [Left up , Right up, Left down, Right down] cases and steep diagonal [Left up , Right up, Left down, Right down] cases. This was done to ensure that all cases of coordinates provided to the module would draw a line. It was important for us to test for the left cases where x0 was less than x1 as this would test that the algorithm was able to switch the x0 and x1 values and increment correspondingly to draw the same line. For the steep/ gradual cases, it was important for us to test that the steepLine assignments worked correctly. X and y values would switch if a steep line was inputted (absolute dy > absolute dx) and these values would then be switched back when assigned to the x and y output of the system. We tested that the switched values would increment correctly to draw the correct steep line. Reset was set to 1 to reset the values and error, then Reset was set to 0 to start the line drawing. The results are shown below.

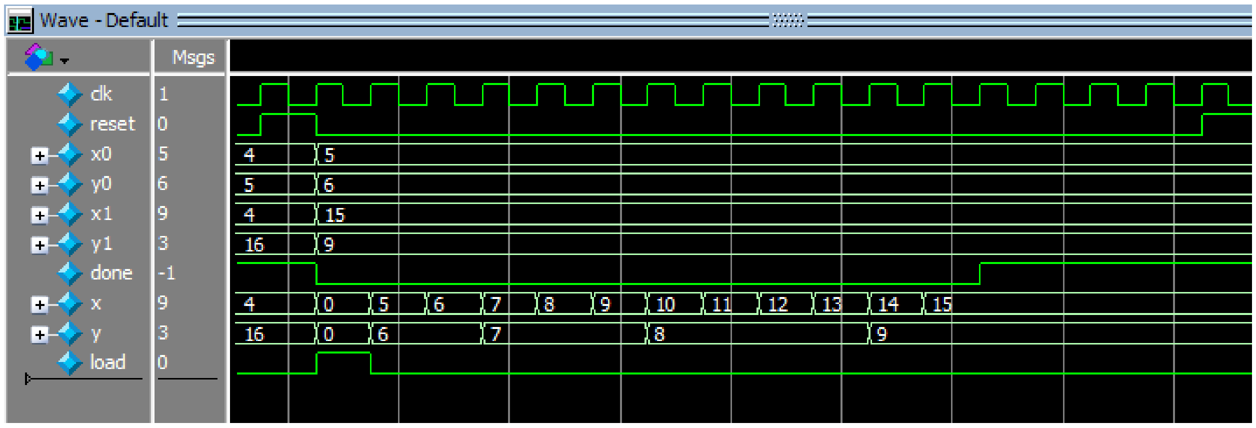
Horizontal Line Test:



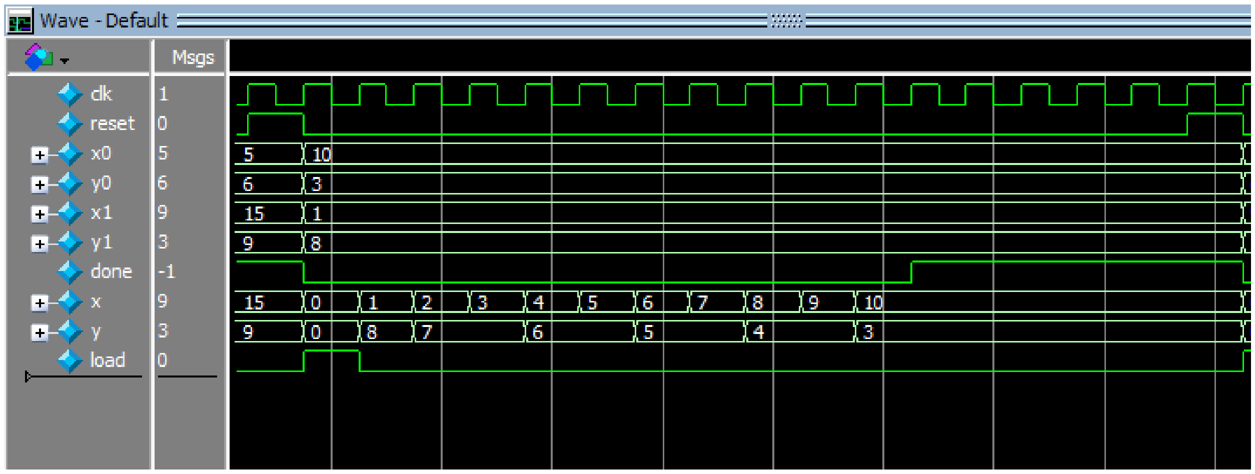
Vertical Line Test:



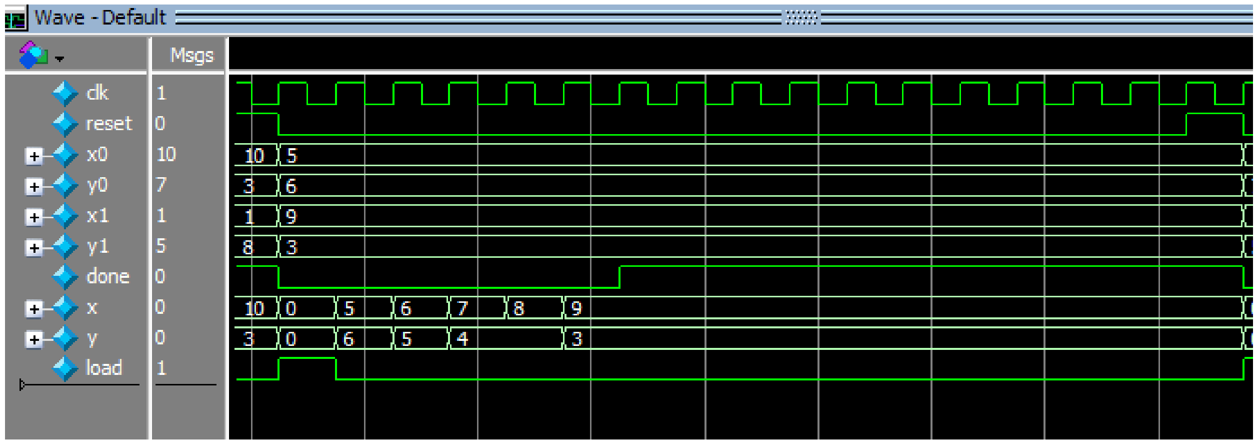
Diagonal Line GRADUAL RIGHT DOWN Test:



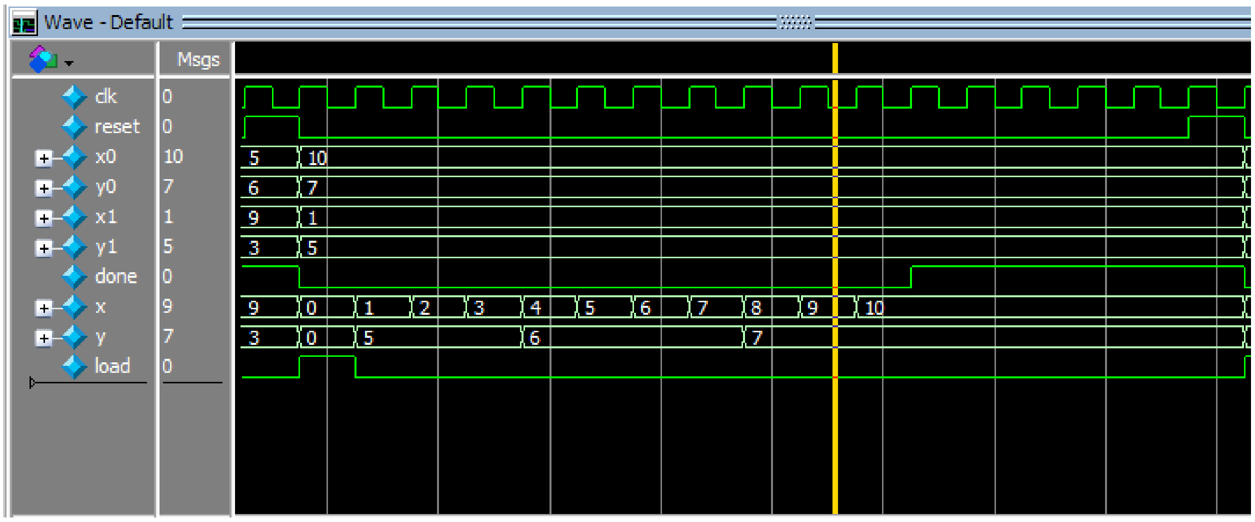
Diagonal Line GRADUAL LEFT DOWN Test:



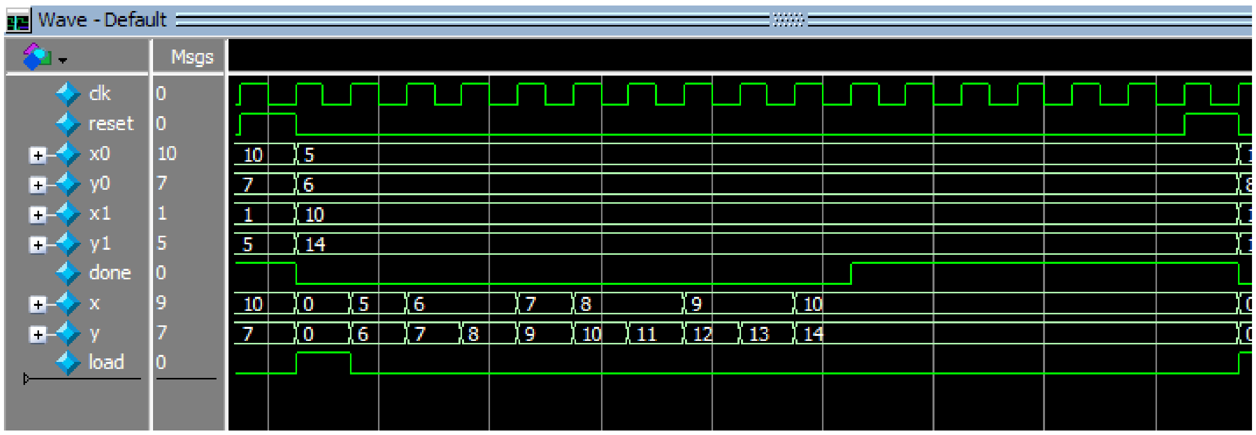
Diagonal Line GRADUAL RIGHT UP Test:



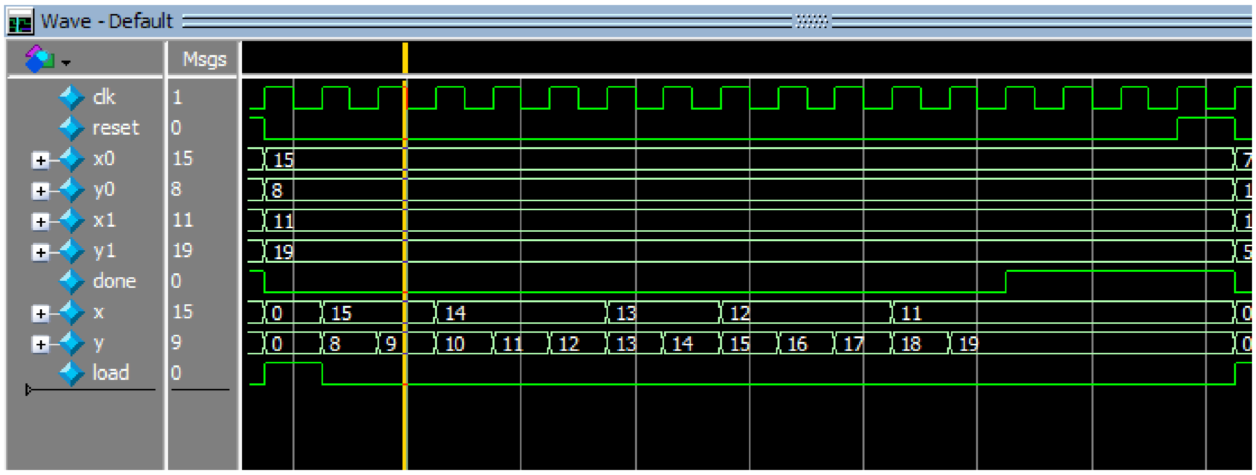
Diagonal Line GRADUAL LEFT UP Test:



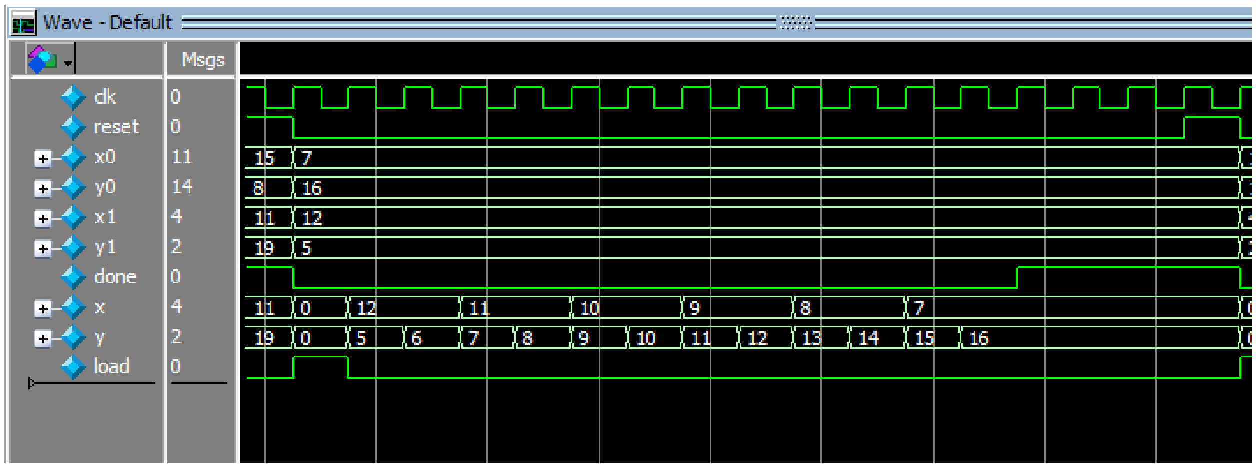
Diagonal Line STEEP RIGHT DOWN Test:



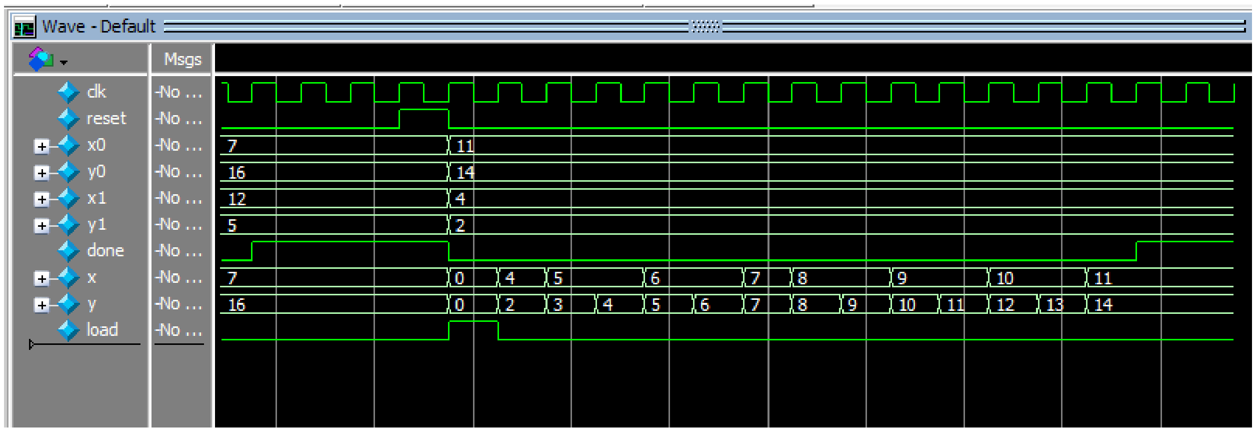
Diagonal Line STEEP LEFT DOWN Test:



Diagonal Line STEEP RIGHT UP Test:

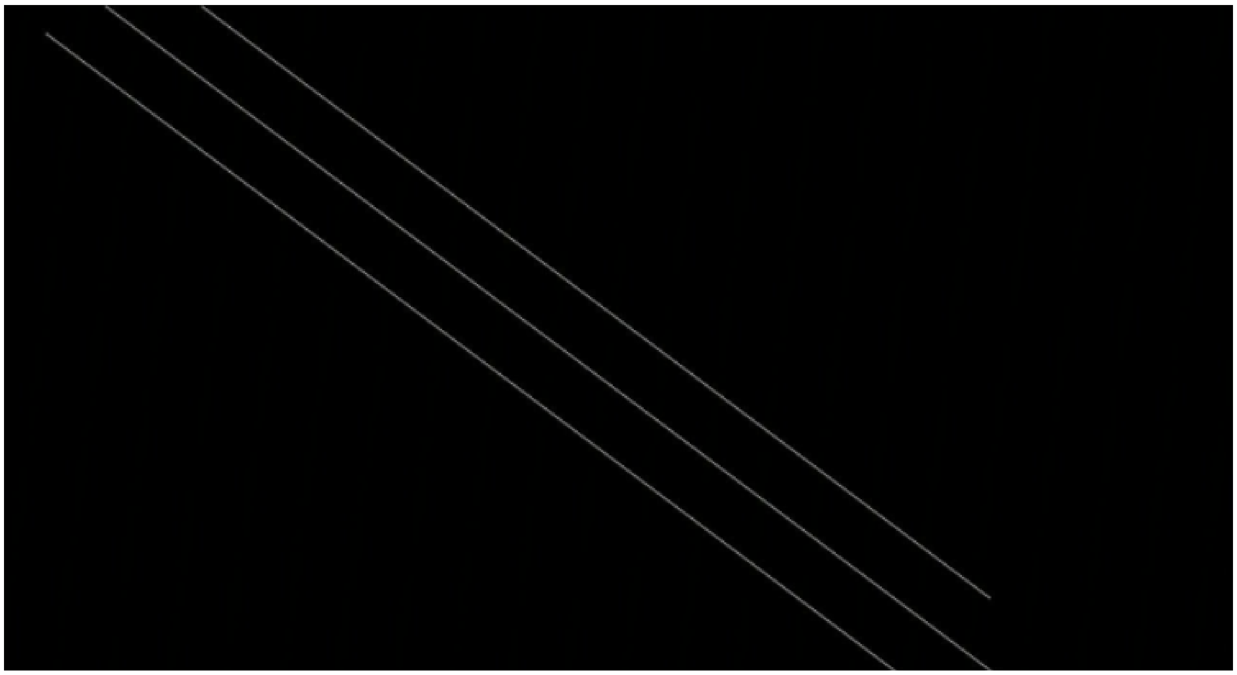


Diagonal Line STEEP LEFT UP Test:



<u>Task 3 Animation:</u>

Here is a picture of what our animation looks like. The lines are drawn slowly to show the whole thing being created. Along with that, we chose this pattern to represent a meteorite shower of 3 meteorites going across the night sky.



Experience

We found this lab to be the hardest lab so far. For task 2 implementing the algorithm into code from the provided pseudo code was relatively simple. Creating the cases for steep lines and Left/Right were also simple. However what took the longest time for task 2 was the debugging stage. We reached a point where all the cases were working except for the steep diagonal line cases. It was really hard to debug this as we weren't sure where to look for the error. The algorithm was clearly working for the other cases to be drawn correctly and so we thought it was to do with our assignments for steepline cases. We changed and swapped variables here and there but our steep line case was still not working. We reached a point where the y value was incrementing correctly but x was incrementing past the x1 provided. To fix this, we condensed the code for all the cases with help from a TA during office hours so that we can clearly see where something was going wrong with the if statements (it was hard to see with so many if statements). We then noticed that we forgot to create an if steepLine statement to switch dx and dy for steep line cases as advised. Once we did this, the steep line cases started drawing correctly. The debugging for task 3 required a lot of testing on ModelSim and office hours visits as our task 2 line drawer was not correctly drawing the line as we wanted it to. However, after alot of confusion and debugging, we found the cause to be because we had placed the y_step cases in an always_comb block and so we removed the always_comb block and assigned y_step as normal. We then also slowed the clock down, using clock[14] which finally created the lines we wanted to create.

This lab took us approximately 37 hours, broken down as follows:
Reading – 1 hours
Planning – 0.5 hours
Design – 1.5 hour
Coding – 13 hours
Testing – 1 hours
Debugging – 20 hours