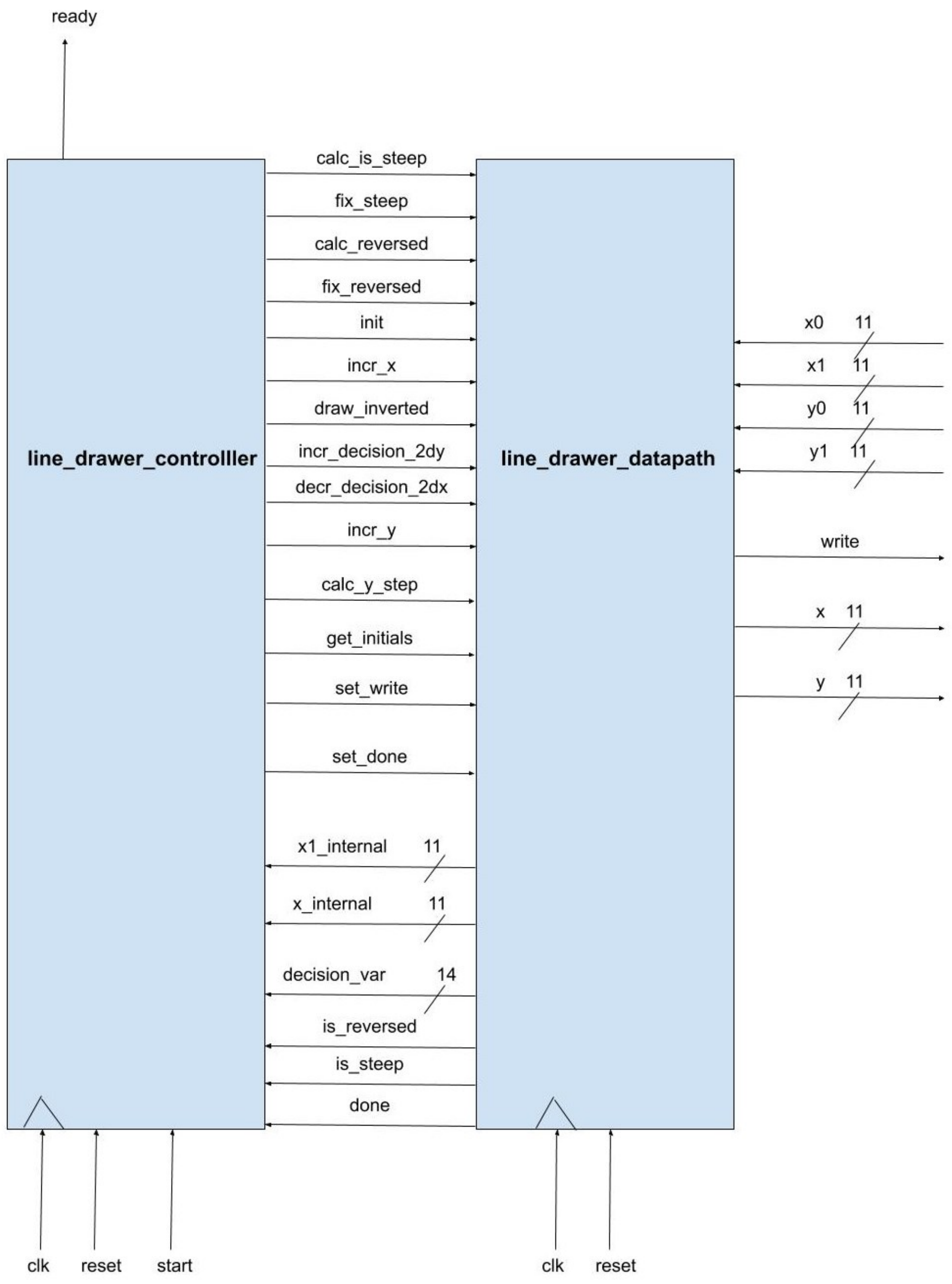
Student Name: Rafael Laya
Student ID Number: 1075974

EE/CSE 371: Laboratory #3, Display Interface

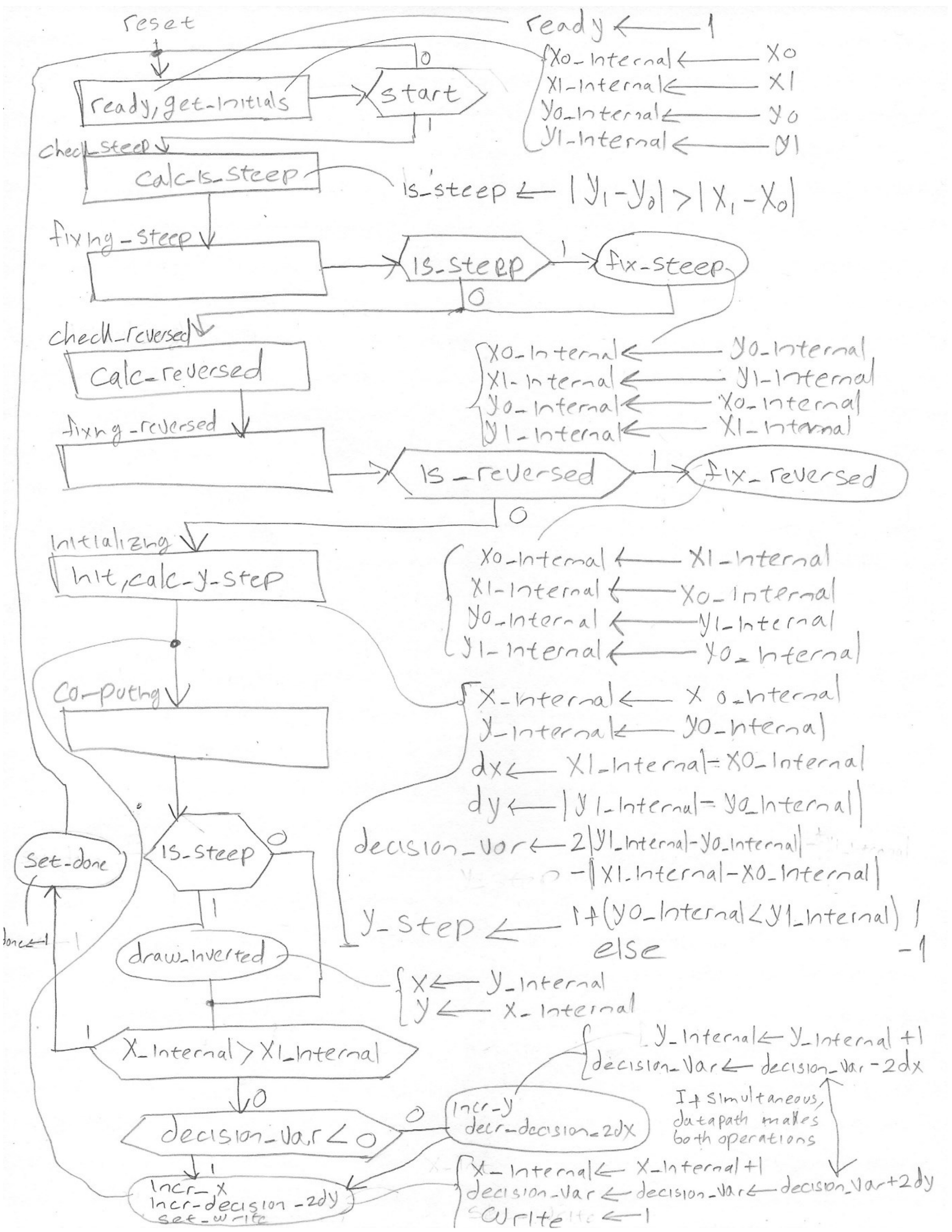
Design Procedure

Task #2

The following image shows a block diagram of the controller and the Datapath developed for the *line_drawer* module.



The *line_drawer* module is attached as *line_drawer.sv*, its controller is attached as *line_drawer_controller.sv*, and the datapath is attached as *line_drawer_datapath.sv*. Next is the Algorithmic State Machine Chart with Datapath (ASMD) chart:


reset
ready ← 1
x0_internal ← x0
x1_internal ← x1
y0_internal ← y0
y1_internal ← y1
ready, get_initials
start
0
1
check_steep
calc_is_steep
is_steep ← |y1 - y0| > |x1 - x0|
fixing_steep
is_steep
1
fix_steep
0
x0_internal ← y0_internal
x1_internal ← y1_internal
y0_internal ← x0_internal
y1_internal ← x1_internal
check_reversed
calc_reversed
fixing_reversed
is_reversed
1
fix_reversed
0
x0_internal ← x1_internal
x1_internal ← x0_internal
y0_internal ← y1_internal
y1_internal ← y0_internal
initializing
init, calc_y_step
x_internal ← x0_internal
y_internal ← y0_internal
dx ← x1_internal - x0_internal
dy ← |y1_internal - y0_internal|
decision_var ← 2|y1_internal - y0_internal| - |x1_internal - x0_internal|
y_step ← 1 if (y0_internal < y1_internal) 1 else -1
computing
set_done
done ← 1
is_steep
0
1
draw_inverted
x ← y_internal
y ← x_internal
1
x_internal > x1_internal
0
y_internal ← y_internal + 1
decision_var ← decision_var - 2dx
decision_var < 0
0
incr_y
decr_decision_2dx
If simultaneous, datapath makes both operations
1
incr_x
incr_decision_2dy
set_write
x_internal ← x_internal + 1
decision_var ← decision_var ← decision_var + 2dy
write ← 1

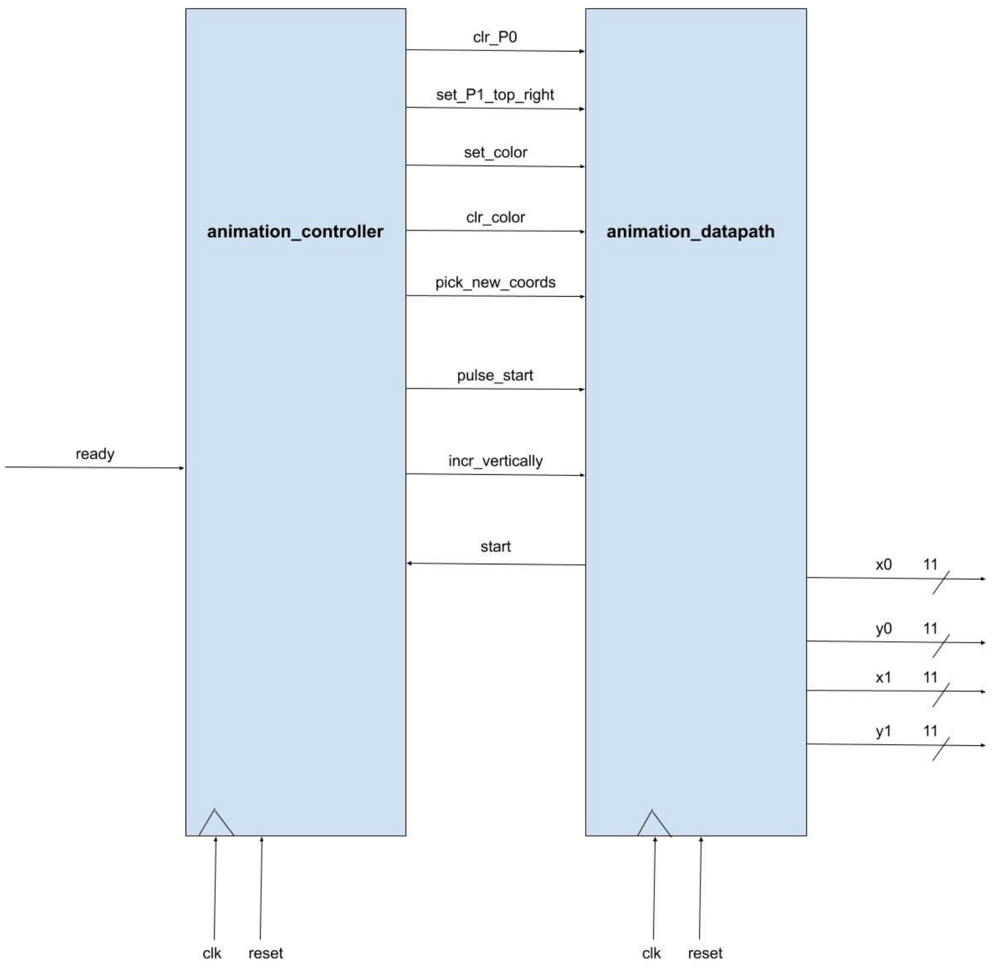
Important design decisions in the development of this module are summarized:

- Produces one point *(x_0,y_0)* on the line each clock cycle.
- Internal variables allow the module to capture a snapshot of the external world before starting the algorithm. This way, if a user wants to use this module, they don't need to hold the initial values of *x_0, y_0, x_1, and y_1* after the machine has started. The module will capture a snapshot and use them.
- The module produces a *write* signal which other modules can use since it is asserted TRUE when the machine is producing points on the line.
- Each control signal performs a small operation, which makes future modifications easier.
- Setting *write* and *done* is done in pulses. The Datapath knows to clear *done* and *write* when it is not receiving a *set_done* or *set_write* signal, respectively.

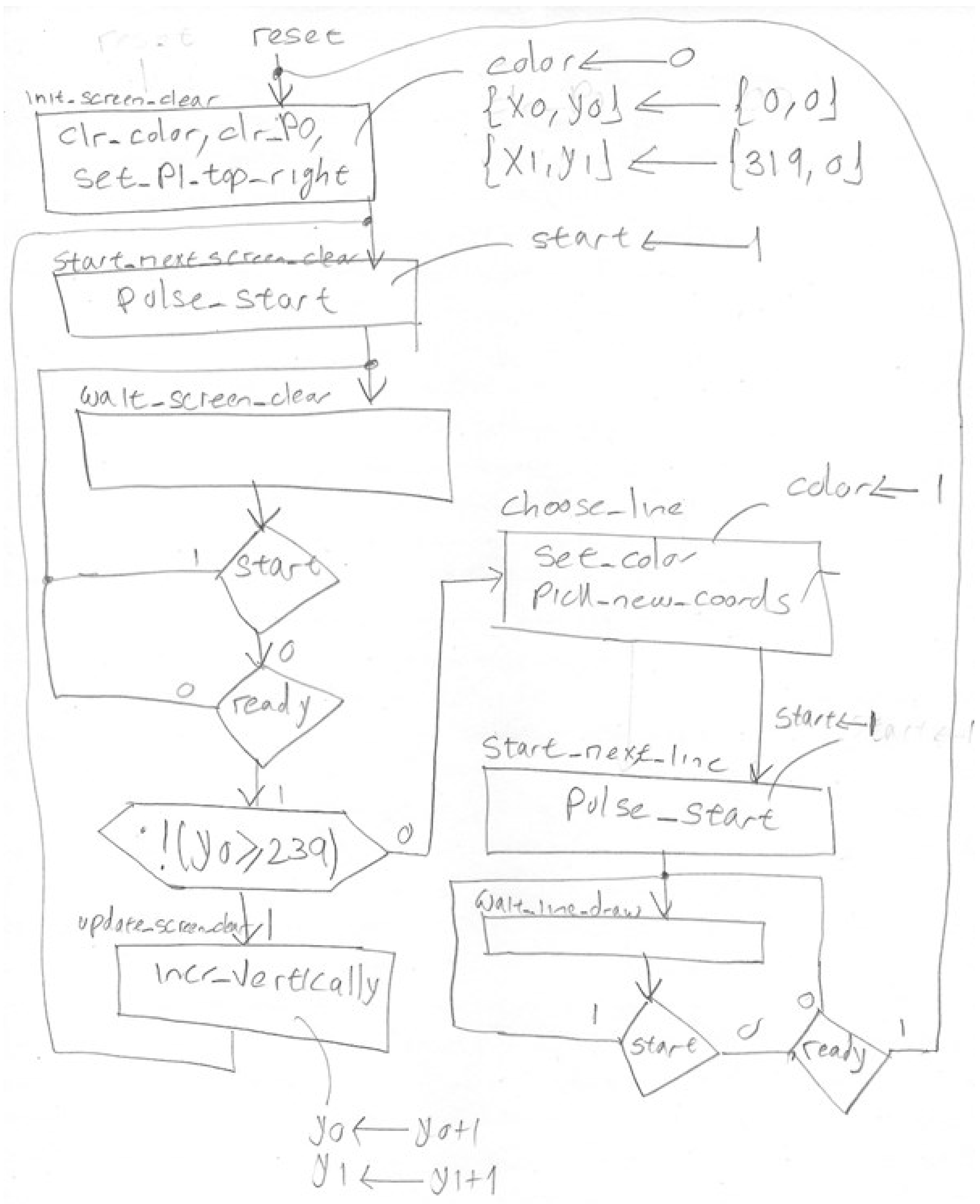
Finally, *DE1_SoC_task2.sv* simply uses line drawer to draw a sample line as required.

Task #3

The module *animation_controller* attached as *animation_controller.sv* and *animation_datapath* attached as *animation_datapath.sv*, use *line_drawer* to produce an animation on the screen. The animation consists of a line that is rotating clockwise around a fixed point. The fixed point has coordinates (159, 150), and the other endpoint is at the border of the screen. A block diagram of this controller-datapath pair is shown next:



Now, having the block diagram, it is easier to draw the ASMD chart for this controller-datapath pair. From the ASMD chart in the next page, observe that the module clears the whole screen after drawing a line. Initially, the machine would draw a linear, clear it, and draw the next one. However, it was found to be too fast and not aesthetically pleasing. After some testing, I found it was more aesthetically pleasing and with a better speed by clearing the screen after each line-draw.

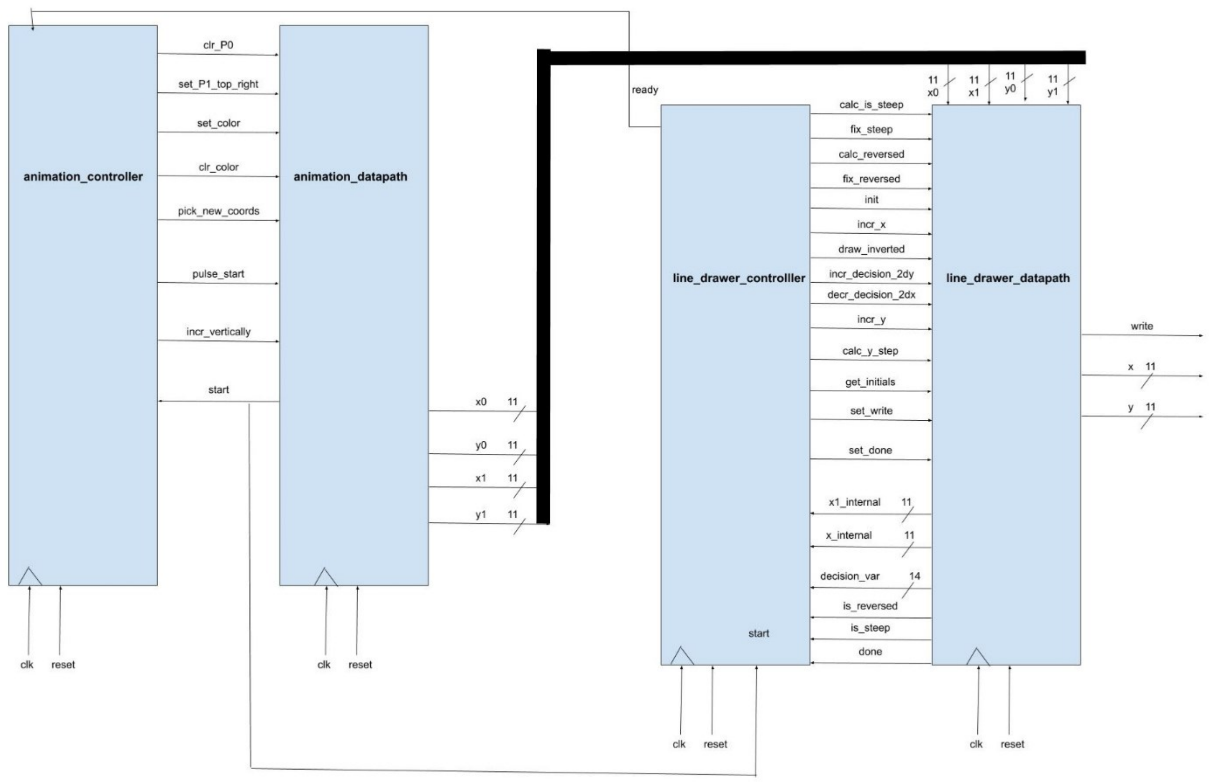


The RTL operation produced by *pick_new_coords* is the following:

x0 ← if (cntr <= 319): cntr
else if (cntr <= 558): 319;
else if (cntr <= 877): 318 – (cntr – 559);
else if (cntr <= 1115): 0;
	y0 ← if (cntr <= 319): 0;
	else if (cntr <= 558): cntr - 319;
	else if (cntr <= 877): 239;
	else if (cntr <= 1115): 239 – (cntr – 877);

x1 ← 159	y1 ← 150

The *animation_controller* and *animation_datapath* are interconnected with *line_drawer* from the previous task to produce the animation. The connections are done in *DE1_SoC_task3.sv* and are shown in the image:



Results

Overview

The module *line_drawer* produces a point (x, y) on the line with provided endpoints (x_0,y_0) and (x_1, y_1) upon sense of an active-high *start* signal. Each point is produced during a clock cycle, and the module produces a *ready* signal when it is idle and ready to draw another line. The module also provides a *write* signal that can be used by a drawing module for convenience, although the *ready* and *done* signals are enough. The module works to specification and behaves as designed.

The modules *animation_controller* and *animation_datapath* make use of *line_drawer* in order to draw a line that rotates clockwise around a fixed point on the screen which was chosen experimentally in terms of visual preference and which is (159, 150). The animation is smooth and meets performs as designed. One can clearly see the rotating line through a screen.

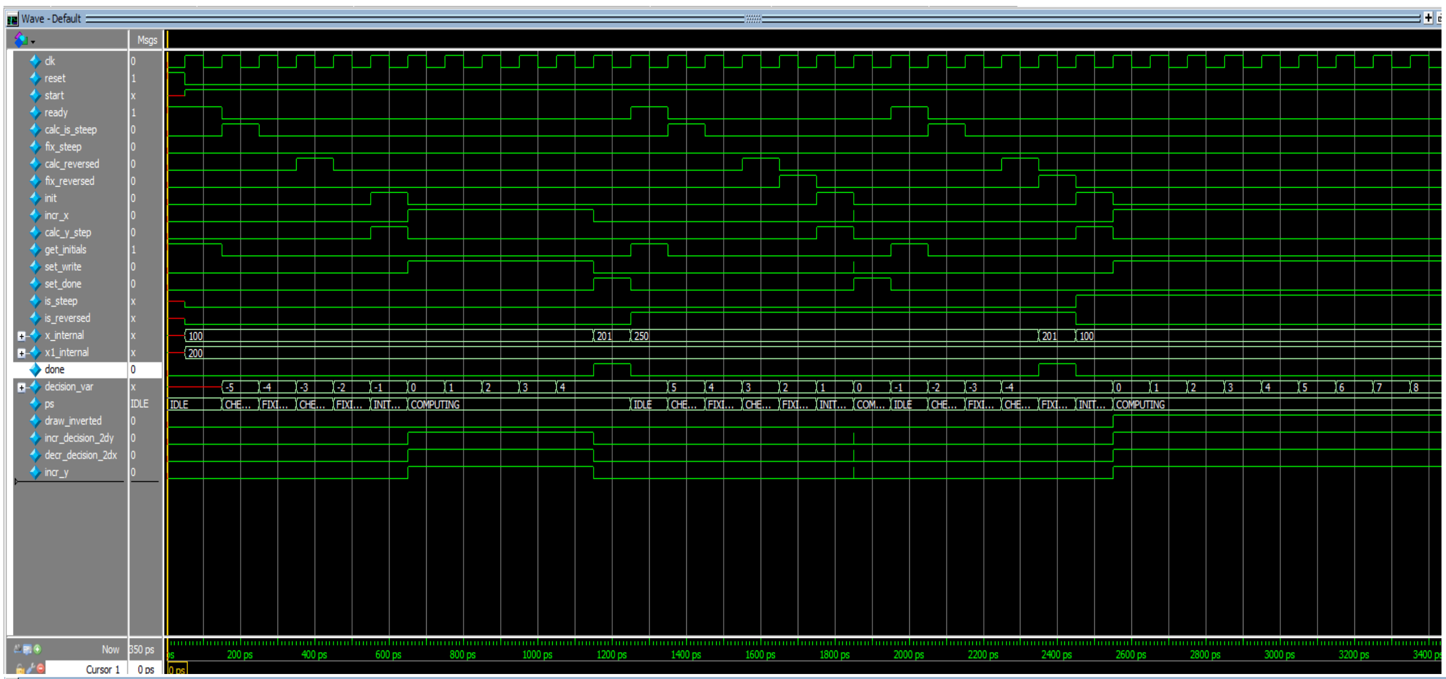
Task #2

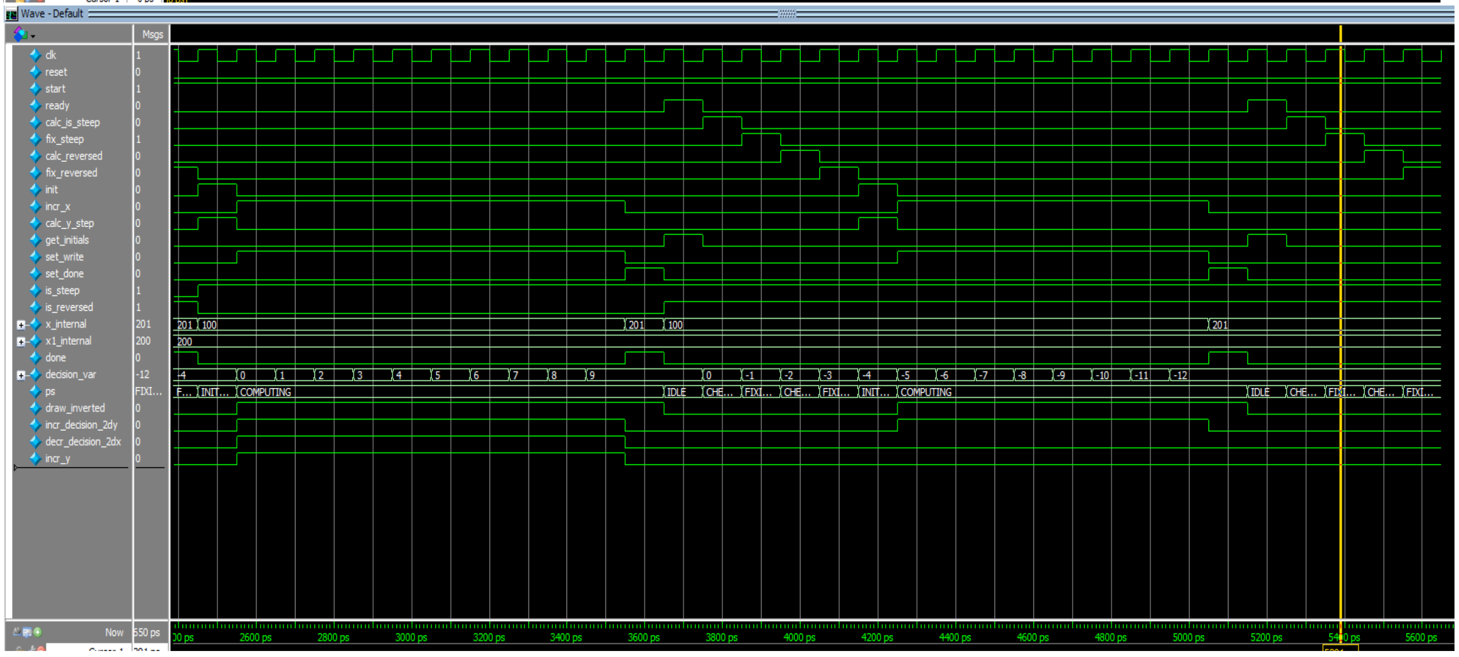
Line Drawer Controller

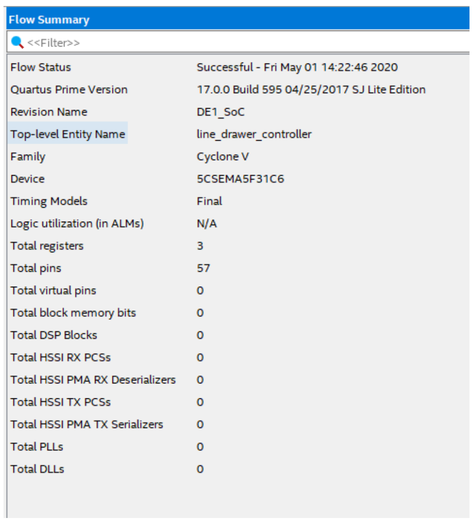
The simulation is shown next.

- The testbench goes through various types of lines. Each control signal behaves as expected from the ASMD chart.
- The module follows through the state sequence of IDLE→CHECKING_STEEP→FIXING_STEEP→INITIALIZING→COMPUTING→IDLE, and so on.
- Each control signal is triggered at the correct moment. For example, *fix_steep* is asserted for a clock cycle during FIXING_STEEP if *is_steep* is TRUE. Analogous is the case for *fix_reversed* during FIXING_REVERSED due to *is_reversed*. All signals follow the behaviour from the ASMD.
- The *decision_var* value is simulated since it is computed by the datapath. The controller however is indeed providing *incr_decision_2dy* each time during a computation, and *decr_decision_2dx* only

when the decision variable is negative. Furthermore, *incr_x* always gets asserted during a computation, but *incr_y* only when the decision variable is negative.






Flow Summary

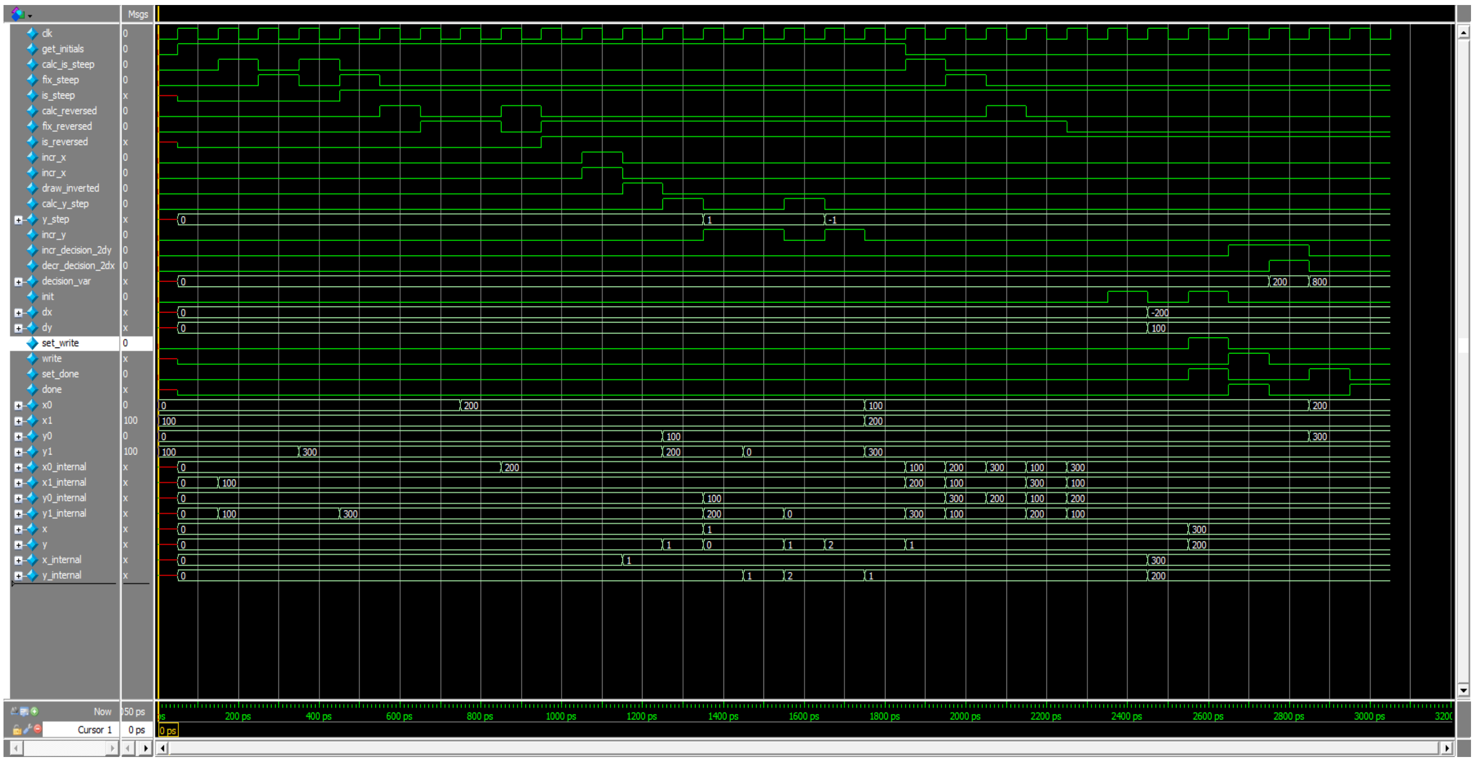
<<Filter>>

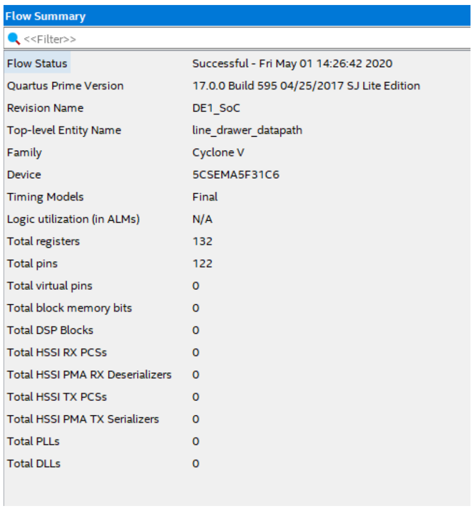
Flow Status	Successful - Fri May 01 14:22:46 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	line_drawer_controller
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	3
Total pins	57
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

Line Drawer Datapath

The easy way to read the waveforms is to first check what signal gets asserted in the top part, and then see the effect at the bottom in the next clock cycle. For instance:

- At 300ps *calc_is_steep* is asserted and causes *is_steep* to be false since the line has slope less than 1. Then at 350ps a new line is simulated with slope bigger than 1 and the module asserts *is_steep*. The same occurs with *is_reversed* between 400ps and 800ps.
- At 1100ps the signals *incr_x* causes *x_internal to increase*. Similarly, around 1800ps *incr_y* causes *y_internal* to increase with the appropriate *y_step* of 1. Then *y_step* is recalculated and becomes -1 which causes an increase of -1 in *y_internal* as expected.
- The decision variable gets calculated according to *init* and gets initialized to 0 since 2*(100)-200 is 0 (around ps 2400ps). Then at around 2900ps it becomes 200 because of *incr_decision_2dy* and then it becomes 800 due to increasing by 2*dy which is 200 and decreasing by 2*dx which turns out to be -200*2=400. Which is what we expected in the simulation.
- The signals done and write are asserted after a *set_done* or a *set_write*, respectively and for one cycle.

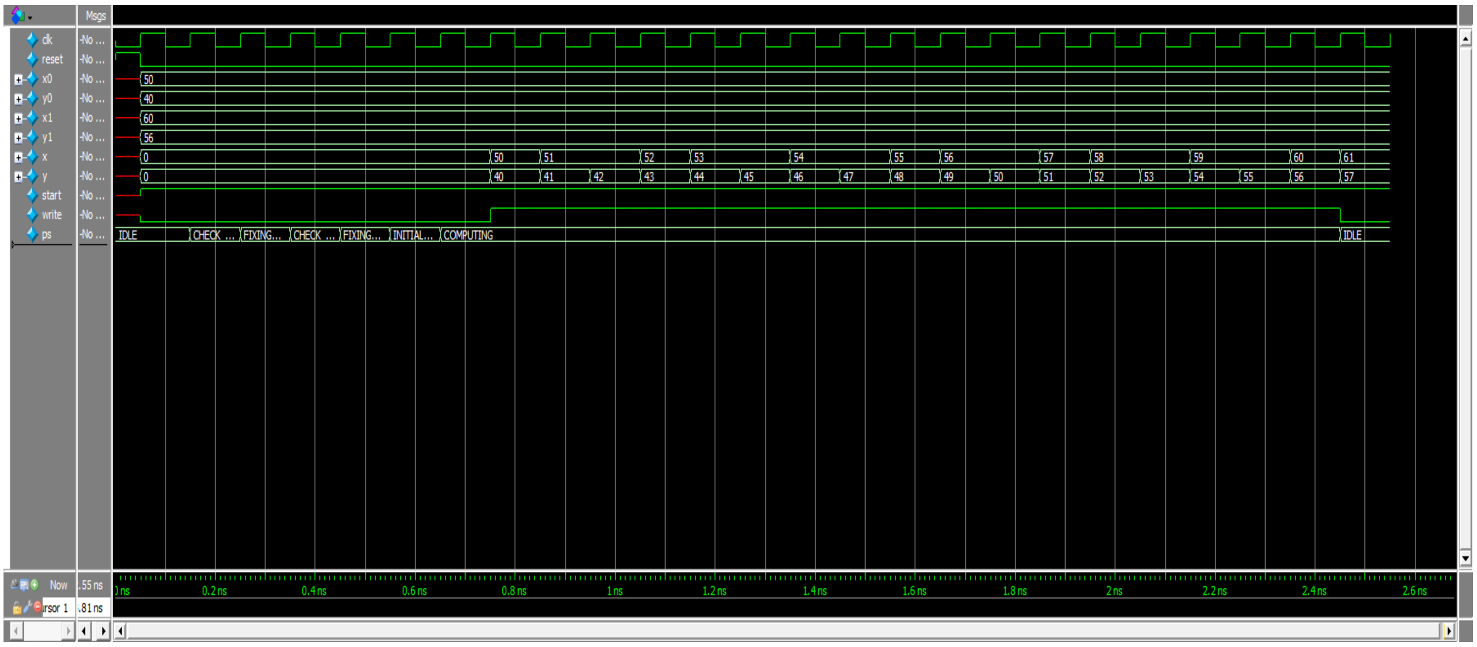


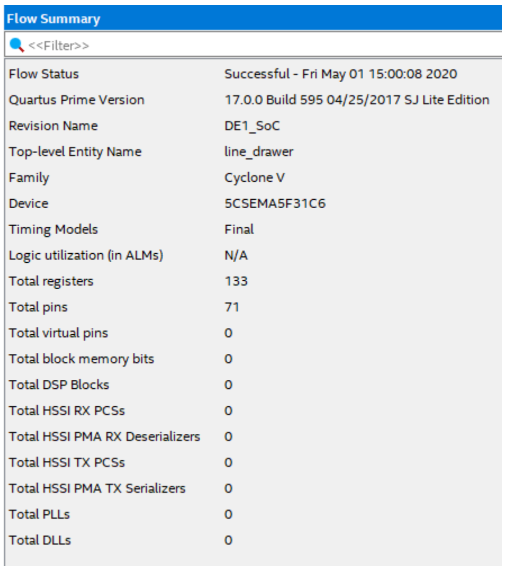


Flow Summary	
Flow Status	Successful - Fri May 01 14:26:42 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	line_drawer_datapath
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	132
Total pins	122
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

Line Drawer

This one is much easier to read since it attempts (and succeeds) to draw a line. The line starts at (50, 40) and ends at (60, 56), and it draws all the points in the middle which is the expected behavior.





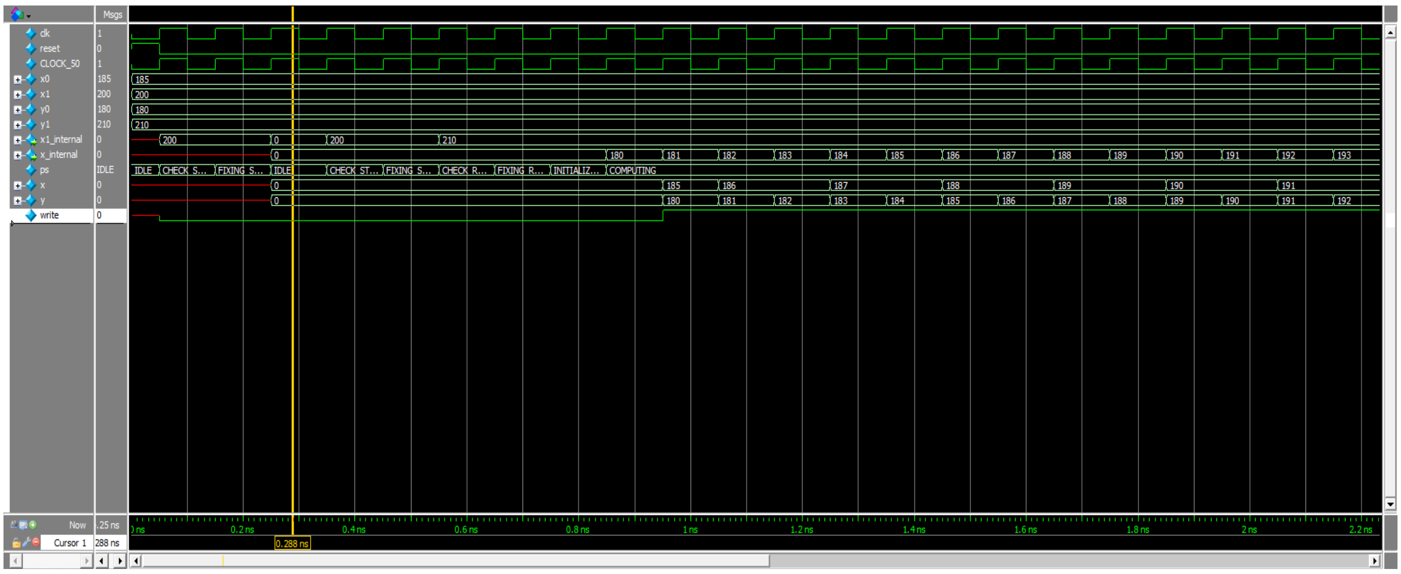
Flow Summary

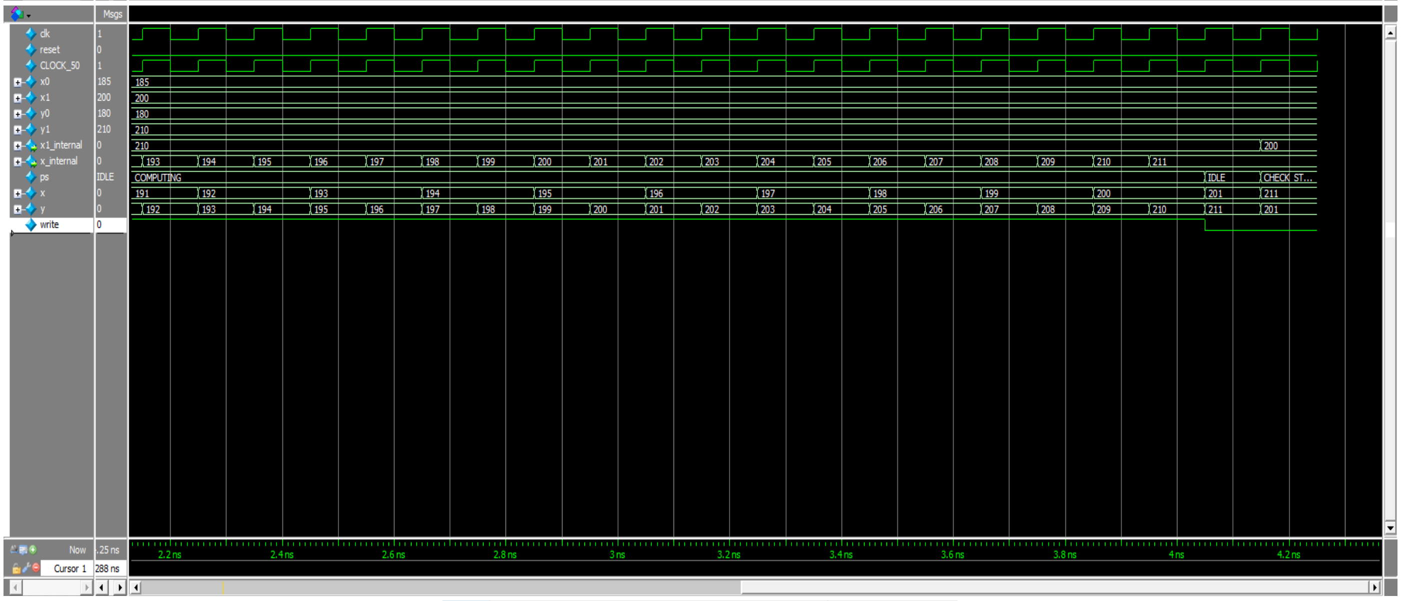
<<Filter>>

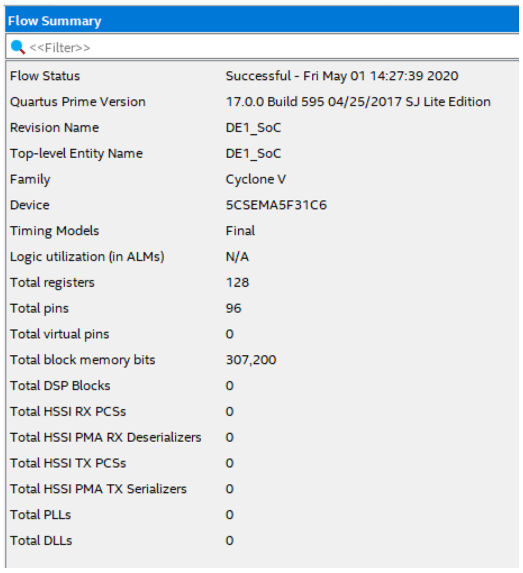
Flow Status	Successful - Fri May 01 15:00:08 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	line_drawer
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	133
Total pins	71
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

DE1_SoC

The testbench draws a line from (185, 180) to (200, 210), which is done correctly. See also that the *write* signal is asserted during the whole computation process and only there.






Flow Summary

<<Filter>>

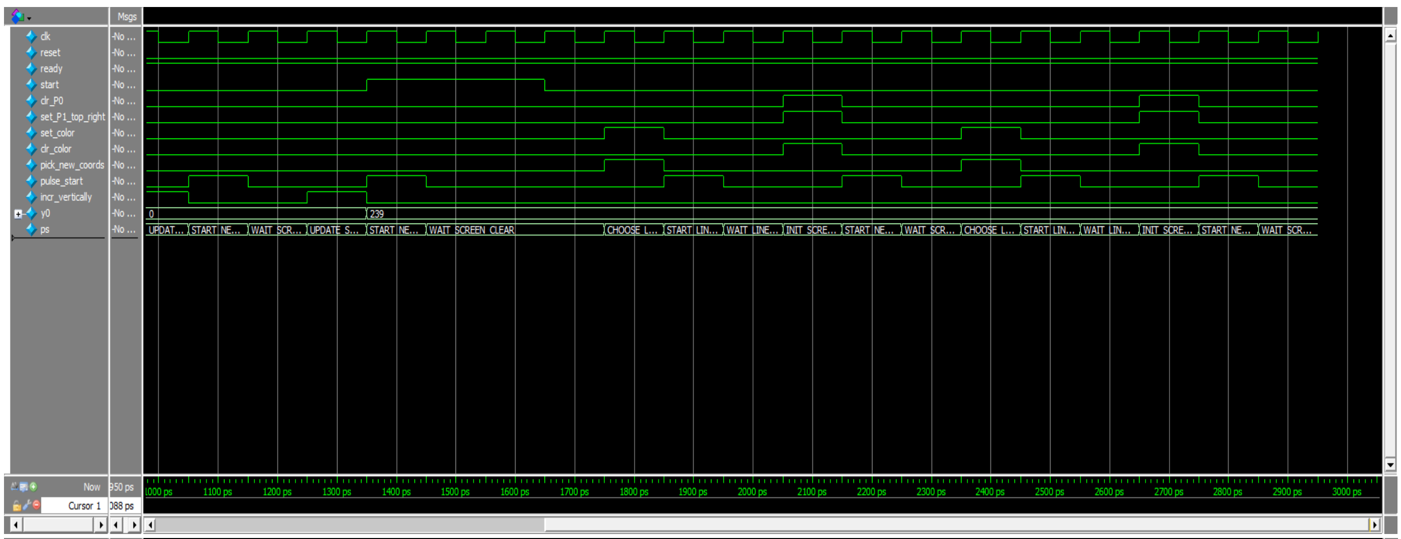
Flow Status	Successful - Fri May 01 14:27:39 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	128
Total pins	96
Total virtual pins	0
Total block memory bits	307,200
Total DSP Blocks	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
Total DLLs	0

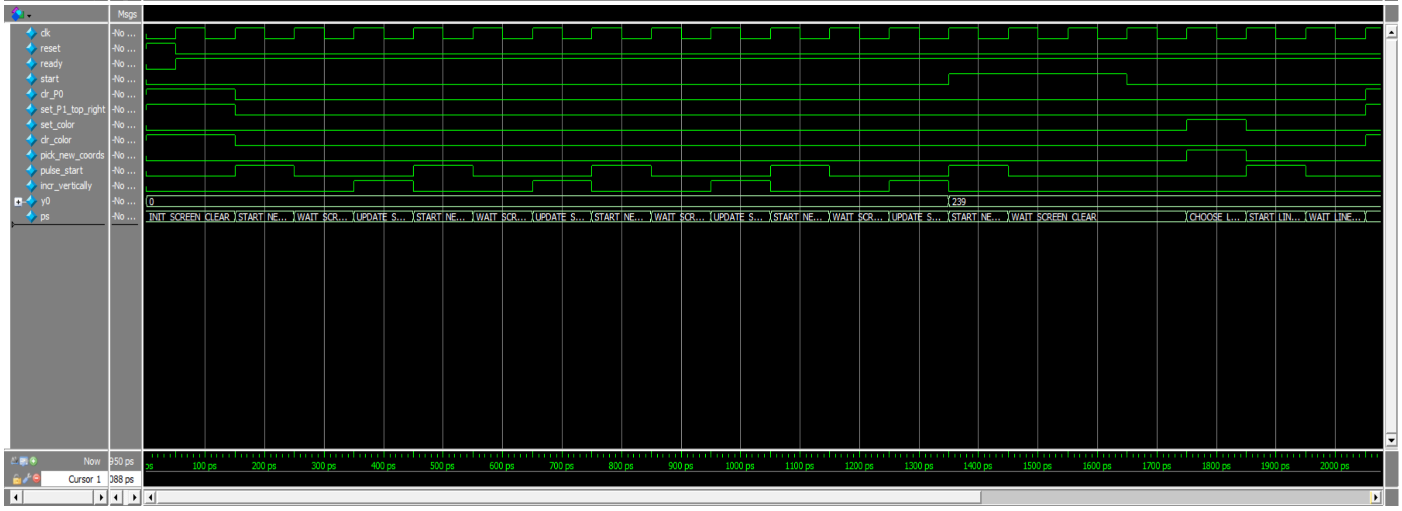
Task #3

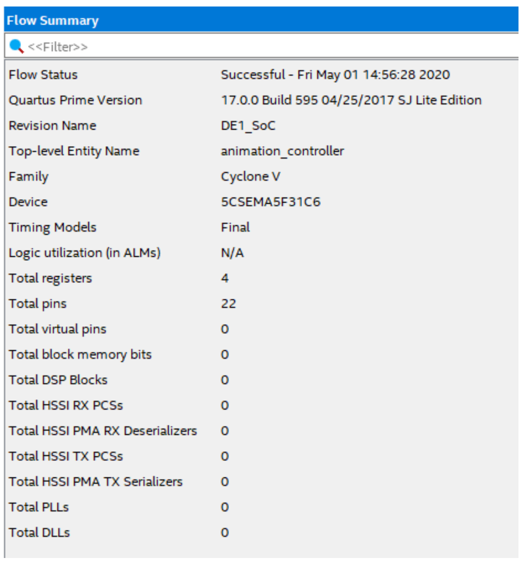
Animation Controller

- Since the behaviour is fixed, it will always do the same operations in the same order, which is what we defined in the ASMD. Therefore, verifying this machine simply means making sure the waveforms confirm the ASMD specifications from design.

- The simulation jumps from $y_0=0$ *to* $y_0=239$ since otherwise the simulation would be thousands of cycles long (see t=1400ps).
- We can see *incr_vertically* is asserted after each line is drawn during the clearing phase which makes the next line to be cleared (color=0) the next horizontal line on the screen (0 ps to 1400ps). The simulation asserts ready to simulate the line being drawn much faster to avoid an extremely long simulation.
- Then the machine starts telling the datapath to choose the coordinates for the animation (1700ps to the end). It also clears the screen after each line is drawn (ex: 2300ps and 2800ps). Naturally, color is set (1800ps and 2400ps) when drawing lines, and cleared when clearing the screen (2100, 2700).





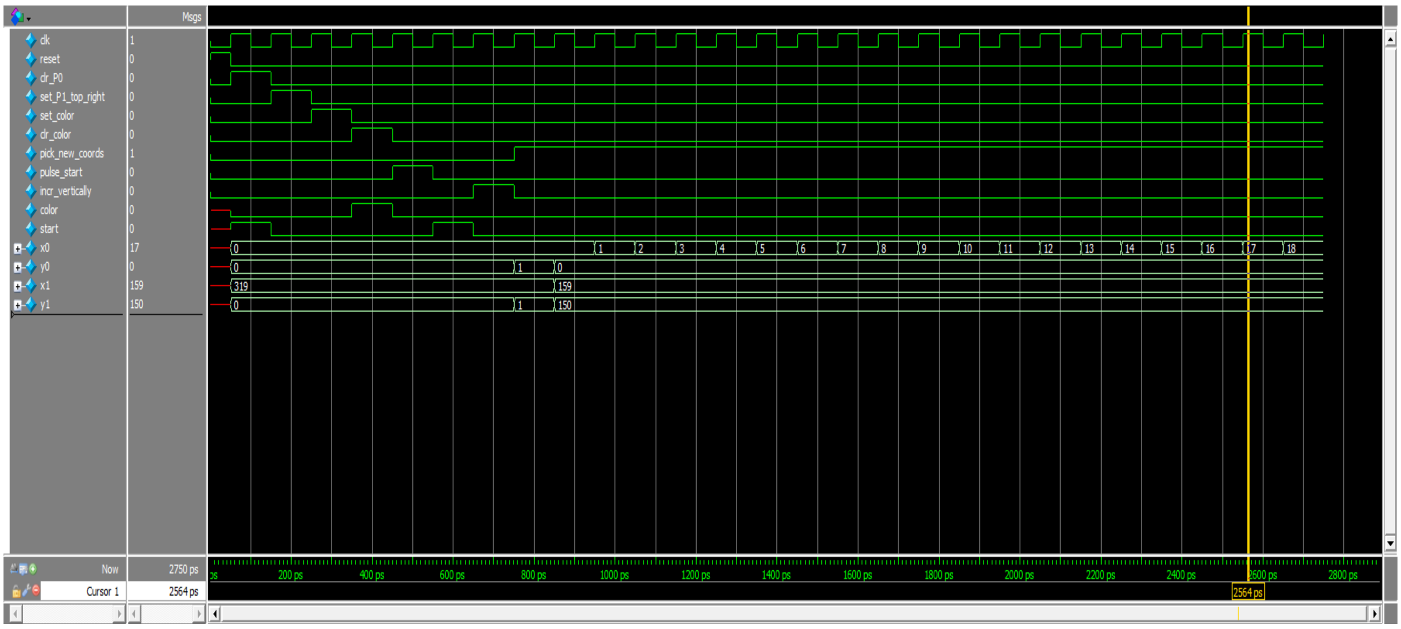

Flow Summary

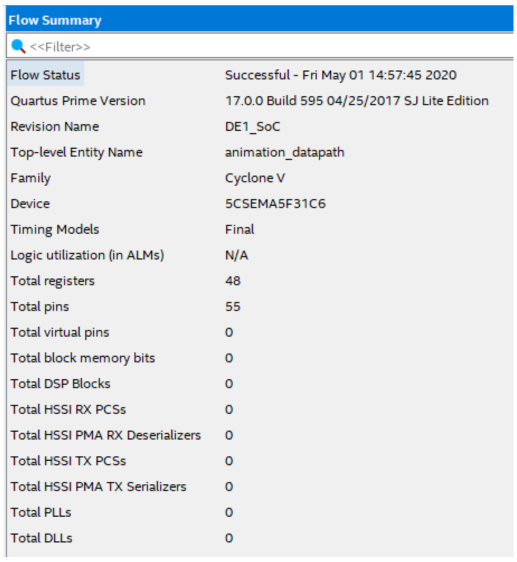
<<Filter>>

Flow Status	Successful - Fri May 01 14:56:28 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	animation_controller
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	4
Total pins	22
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

Animation Datapath

- First tests *clr_P0* clears *P0* by making x0 and y0 zero. Then *set_P1_top_right which makes x1 be 239 and y1 be 0 (top right-corner of the screen).*
- Then tests setting and clearing the color.
- Then pulse-start produces a start pulse.
- The signal *incr_vertically* causes *y0* and *y1* to increase by 1 (used for clearing the screen).
- The signal *pick_new_coords* makes (x0,y0) be (0,0) and then x0 starts increasing (this is going through all the points at the top of the screen. It also sets (x1,y1) to (159, 150) which is the fixed point for the rotations.





Flow Summary

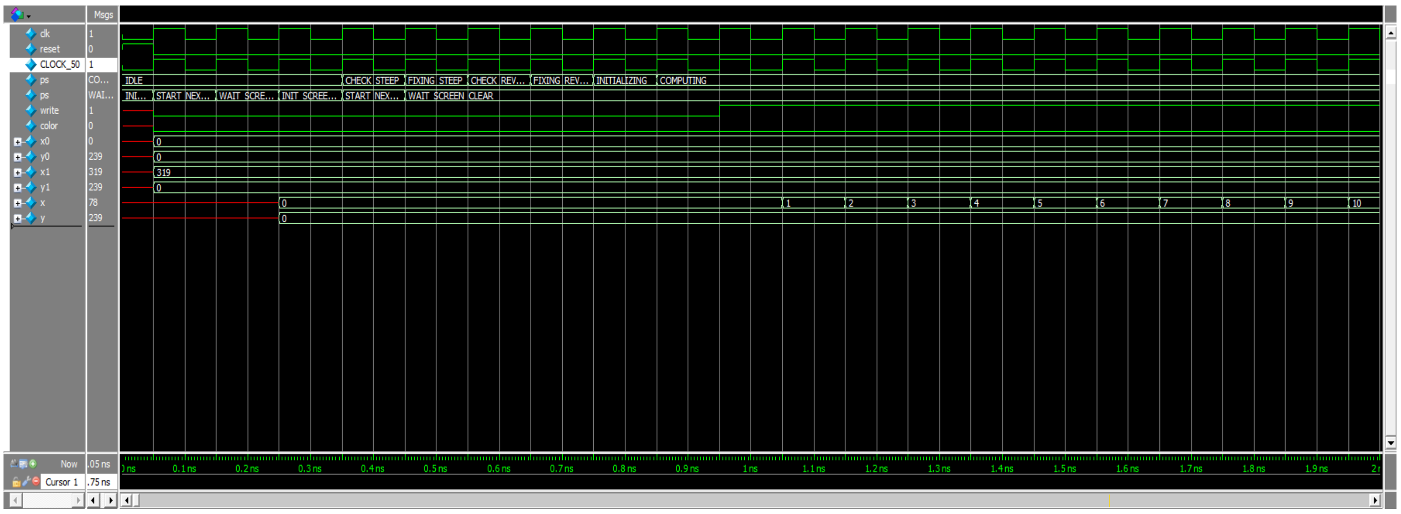
<<Filter>>

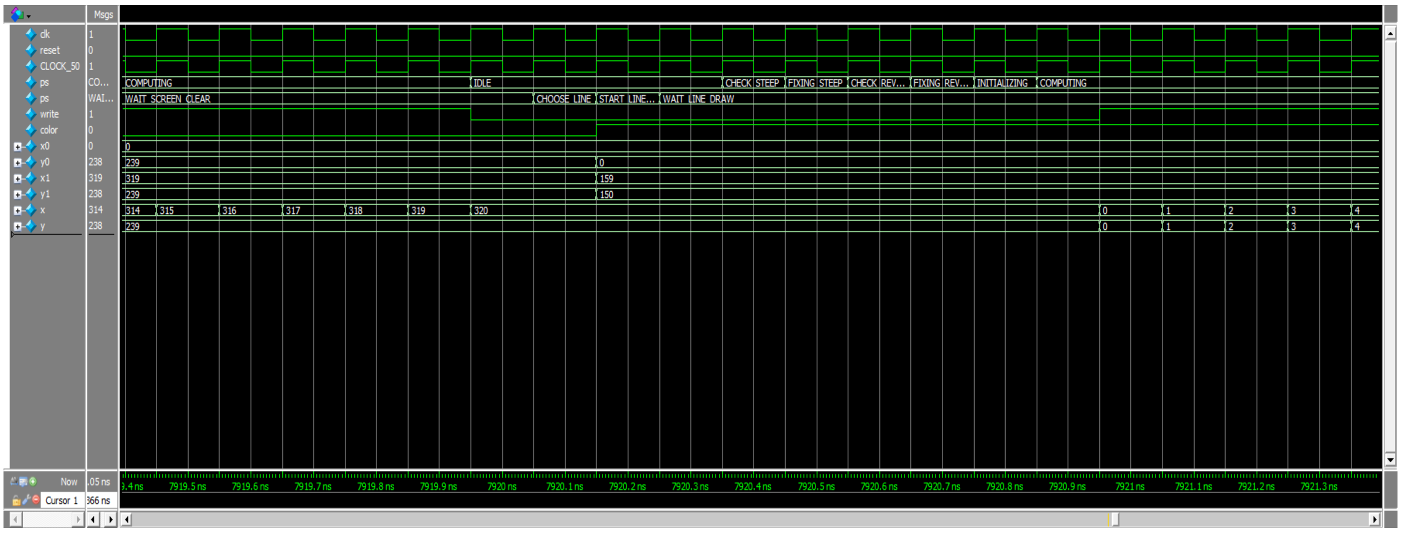
Flow Status	Successful - Fri May 01 14:57:45 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	animation_datapath
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	48
Total pins	55
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	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
Total DLLs	0

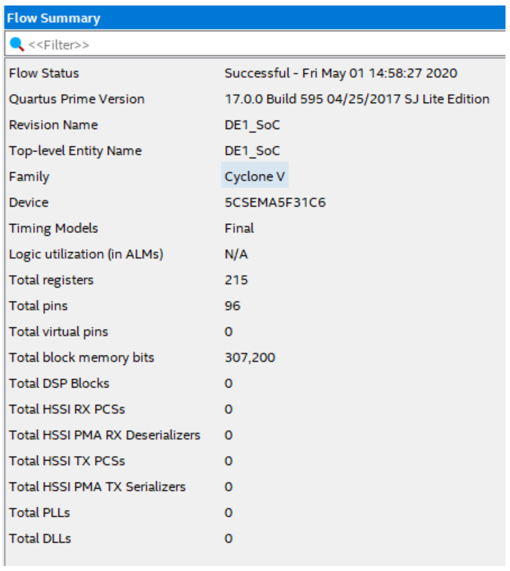
DE1_SoC

There are way too many cycles in an animation to show all of them, but the following shows confidence in the design:

- Starts by drawing the line (0,0) to (319, 0) to clear the first line of the screen at 1ns. It does so until it reaches the line with y0=y1=239 (7919.9ns).
- At 7920.9 it starts drawing the line from (0,0) at the top-left to (159, 150). Which is the first line in the animation. Then it keeps drawing each line one by one and clearing the screen each time.







Flow Summary

Flow Status	Successful - Fri May 01 14:58:27 2020
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)	N/A
Total registers	215
Total pins	96
Total virtual pins	0
Total block memory bits	307,200
Total DSP Blocks	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
Total DLLs	0

Experience Report

One challenge was to get an animation of the rotating line that wasn't too fast nor too slow. I spent some time playing around with the *clock_divider* module and I figured that I needed to draw the same line a couple of times in the same spot before going to the next one. However, I found another fix that I liked more which was more elegant since it simplified the machine that I had at the time. The fix was to clear the screen after each line draw. When I tested, I felt that it was aesthetically looking better than all other alternatives I had tried, thus I stuck with this option which was satisfactory.

Initially, I also had very complex logic to draw the lines on the screen which made the animation_controller/datapath ASMD chart very large. However, I noticed that I could abstract the logic of picking a new set of coordinates by simply having a signal that makes the datapath choose a new coordinate (*pick_new_coords)*. Therefore making the ASMD chart much simpler and easier to implement. Now the controller only provides this signal to pick a new set of coordinates, and the circuit in the datapath makes sure the next coordinate is what I decided it would be for the animation (a rotating line, so it will keep a fixed endpoint while the other one moves along the border of the screen).

Regarding tips for future laboratories, the hardest part was figuring out how to make sequential logic in hardware, which is a new concept for us. Perhaps doing more examples by hand before working on the laboratory assignment might have made development much easier and less time consuming. So, my tip is to work more on practice problems before going to the laboratory assignment will save time.

Feedback about the laboratory is overall positive since I felt that I learned a lot. However, I think it might be beneficial to make future students solve a few problems before getting into this laboratory. I would imagine having a homework assignment that is due before this laboratory is posted that deals with ASMDs and sequential algorithms in hardware would help.

Estimated total time to complete the laboratory: Around 30 hours