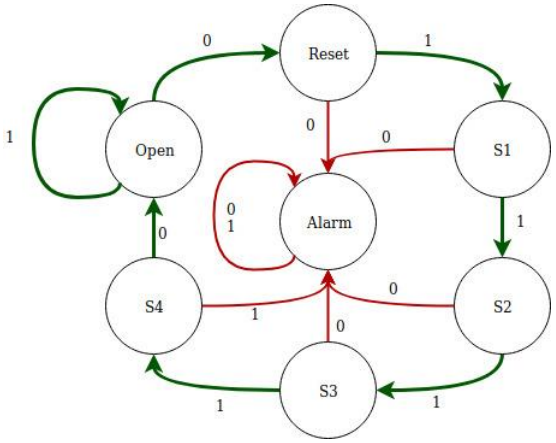
Coursework 2 – Sequential Circuit Design using Quartus

Ghusharib Chohan – CID 01500954

My short-code is MGC18 and hence I was given the number 0029, and so (1+29 = 30) which in binary is 11110. Hence my input code would be 11110. This created the following state transition diagram:

[1]



From this I created the truth table:

DATA	This State	Q2	Q1	Q0	Next State	D2	D1	D0
0	Reset	0	0	0	Alarm	1	1	1
0	S1	0	0	1	Alarm	1	1	1
0	S2	0	1	0	Alarm	1	1	1
0	S3	0	1	1	Alarm	1	1	1
0	S4	1	0	0	Open	1	1	0
0		1	0	1	X	X	X	X
0	Open	1	1	0	Reset	0	0	0
0	Alarm	1	1	1	Alarm	1	1	1
1	Reset	0	0	0	S1	0	0	1
1	S1	0	0	1	S2	0	1	0
1	S2	0	1	0	S3	0	1	1
1	S3	0	1	1	S4	1	0	0
1	S4	1	0	0	Alarm	1	1	1
1		1	0	1	X	X	X	X
1	Open	1	1	0	Open	1	1	0
1	Alarm	1	1	1	Alarm	1	1	1

[1] Created using tools from draw.io

I then created Karnaugh Maps for the above truth table which gave me the following:

D2

Data Q2 \ Q1 Q0	00	01	11	10
00	1	1	1	1
01	1	X	1	0
11	1	X	1	1
10	0	0	1	0

D1

Data Q2 \ Q1 Q0	00	01	11	10
00	1	1	1	1
01	1	X	1	0
11	1	X	1	1
10	0	1	0	1

D0

Data Q2 \ Q1 Q0	00	01	11	10
00	1	1	1	1
01	0	X	1	0
11	1	X	1	0
10	1	0	0	1

The Boolean equations generated from this

D2 = DATA'.Q2' + Q2.Q1' + Q1.Q0 + DATA.Q2
= DATA'.Q2' + Q2.(DATA+Q1') + Q1.Q0

D1 = DATA'.Q2' + Q2.Q1' + Q2.Q0 + Q1'.Q0 + DATA.Q1.Q0'
= DATA'.Q2' + Q2.(Q1'+Q0) + Q1'.Q0 + DATA.(Q1.Q0')

D0 = DATA'.Q2' + Q2.Q0 + DATA.Q1'.Q0' + Q2'.Q1.Q0'
= DATA'.Q2' + Q2.Q0 + Q0'.((DATA.Q1')+(Q2'.Q1))

The good thing to notice from my boolean equations is that the highlighted red equation is common in all three of my equations, so it will only need to be implemented once in hardware. Any other simplifications to the circuit will be done at a later point

Output Logic

LOCK – The state for open is 110 so in any other state, the lock should be on. This gives the boolean equation (Q2.Q1.Q0')'

ALARM - The state for alarm being on is 111 so this gives the boolean equation Q2.Q1.Q0

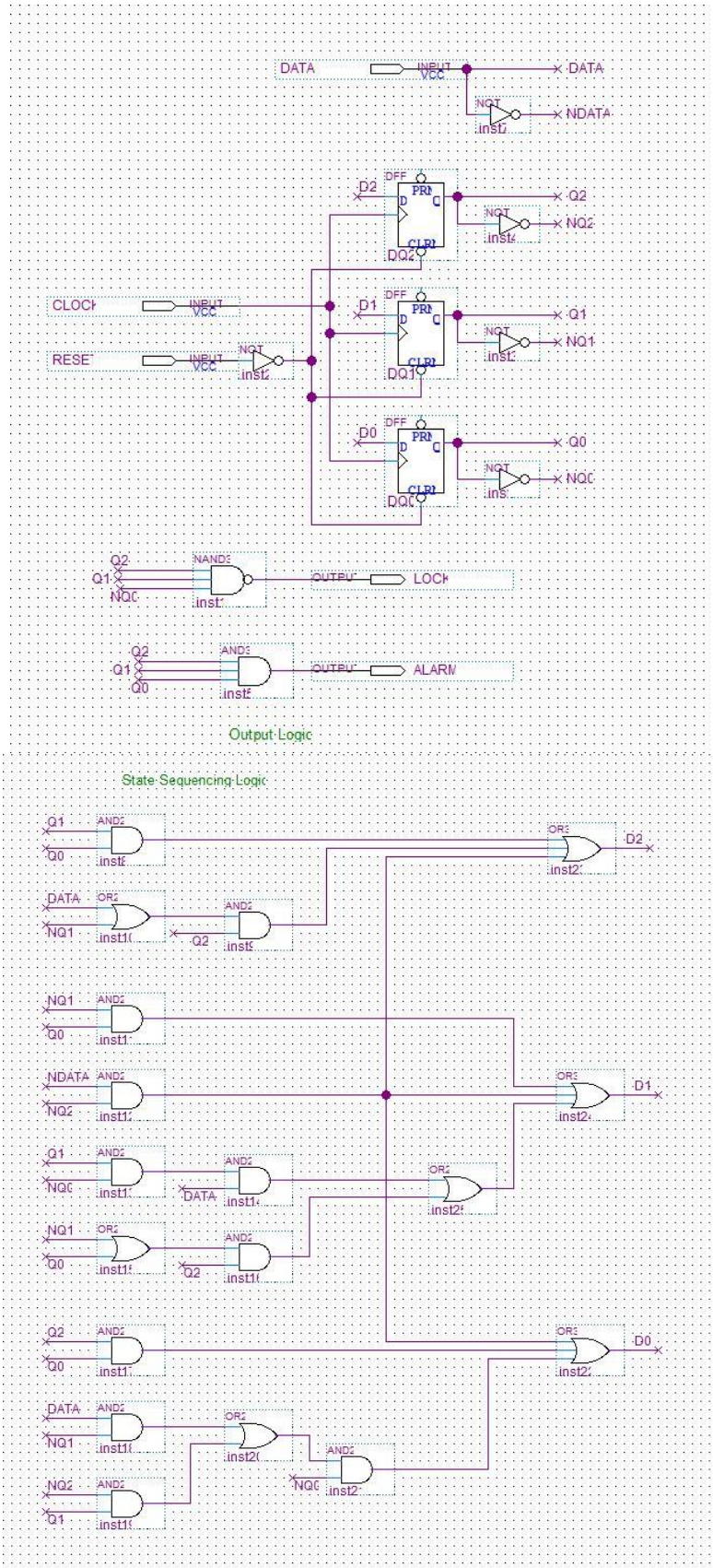
Checking the Don't Cares

DATA	This State	Q2	Q1	Q0	Next State	D2	D1	D0
0	Reset	0	0	0	Alarm	1	1	1
0	S1	0	0	1	Alarm	1	1	1
0	S2	0	1	0	Alarm	1	1	1
0	S3	0	1	1	Alarm	1	1	1
0	S4	1	0	0	Open	1	1	0
0		1	0	1	Alarm	1	1	1
0	Open	1	1	0	Reset	0	0	0
0	Alarm	1	1	1	Alarm	1	1	1
1	Reset	0	0	0	S1	0	0	1
1	S1	0	0	1	S2	0	1	0
1	S2	0	1	0	S3	0	1	1
1	S3	0	1	1	S4	1	0	0
1	S4	1	0	0	Alarm	1	1	1
1		1	0	1	Alarm	1	1	1
1	Open	1	1	0	Open	1	1	0
1	Alarm	1	1	1	Alarm	1	1	1

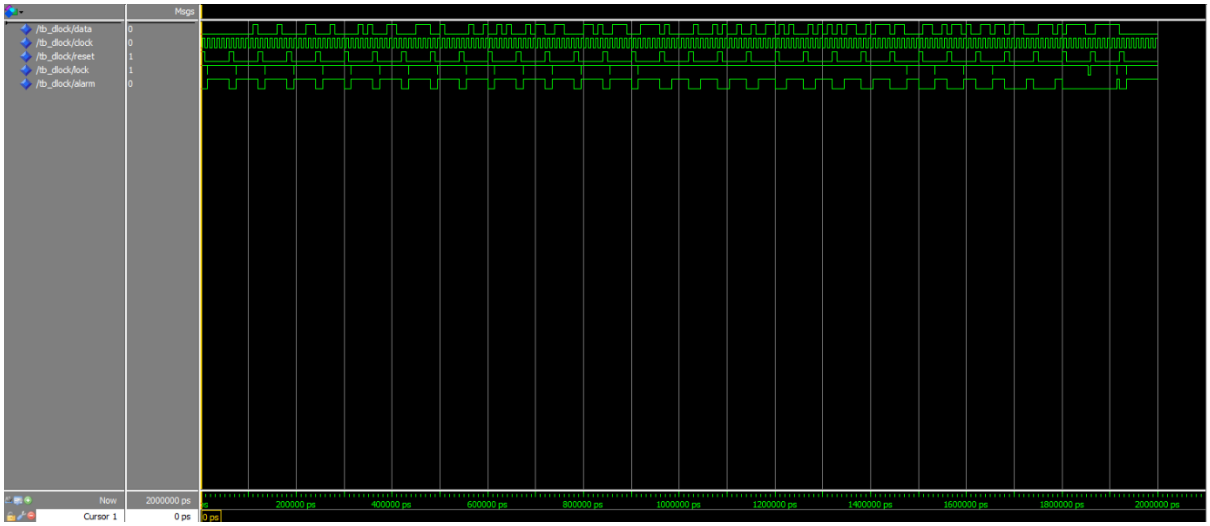
So, upon start-up it could go straight to the alarm state. However, this is not a concern because in the actual circuit we are given a reset pin, which when it is set to 1, the lock is put into the reset state. So, upon start-up we could simply set the reset pin to 1 and it would go from Alarm State into reset state. Also, it is good it does not go to the open state on startup.

Implementing the initial design into hardware

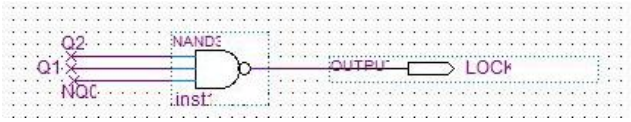
Note this is not the final circuit, as it requires a lot more simplification, but is purely for testing purposes. I implemented the following circuit into hardware:



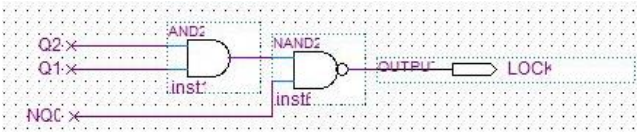
This gave the following output on ModelSim:



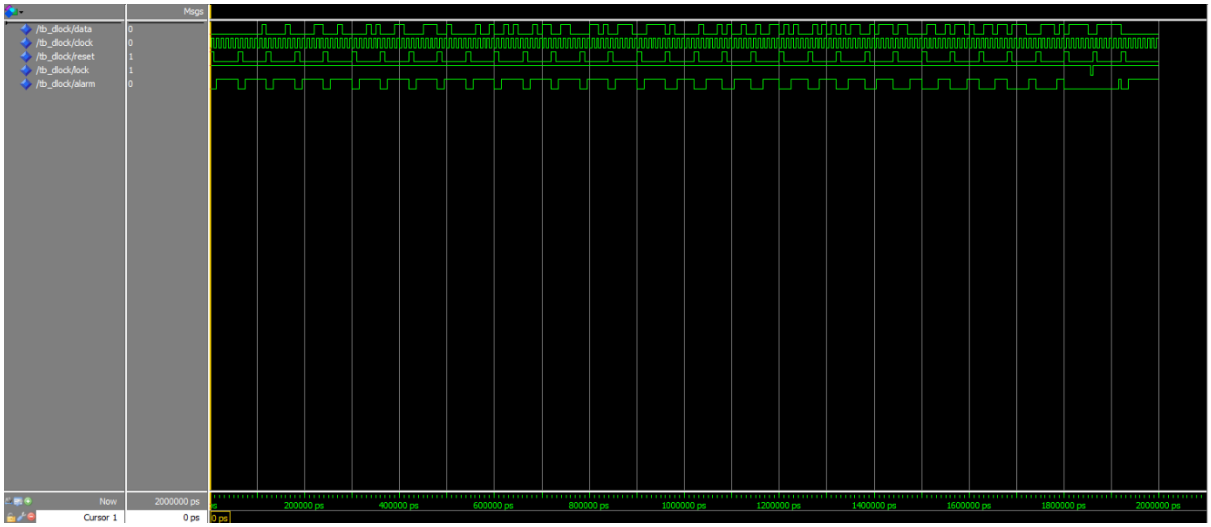
It can be seen that the circuit does unlock for the correct input (11110) but unfortunately, we have many spikes in the circuit for the lock. Though these occur for mere picoseconds, they are still a cause for concern and it is better if there were no spikes at all. The problem lies in the output logic:



Q2 and Q1 both feed directly into the NAND3 gate, but Q0 goes through a NOT gate before being fed into the NAND3 gate (hence why it is NQ0). This causes a timing issue and is the cause of the spike. I fixed this by changing the logic to the following:



This results in all three inputs having to go through some gate before being fed into the NAND2 gate. This increases the area (which is a trade-off) but gets rid of the spikes as can be seen on the next ModelSim diagram.

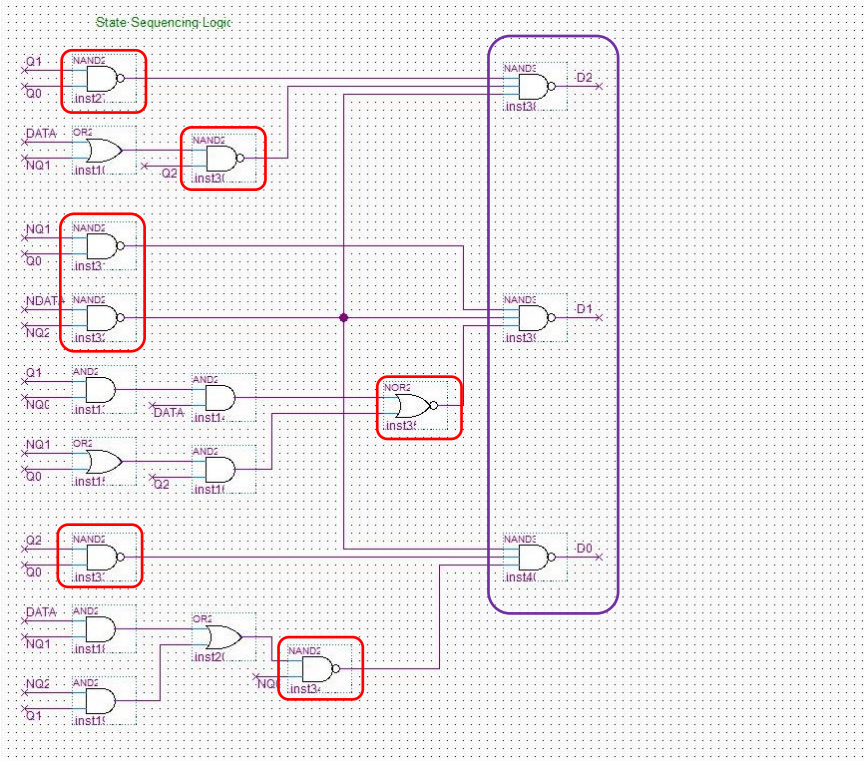


Simplifying the circuit

The above circuit is not at all ideal. It uses up far too much area. Currently, the total area calculation for the circuit is as follows:

Gate Name	Amount	Total Area
DFF	3	300
INPUT	3	0
OUTPUT	2	0
NOT	5	75
AND2	12	324
AND3	1	34
NAND2	1	20
OR2	4	112
OR3	3	102
	TOTAL	967

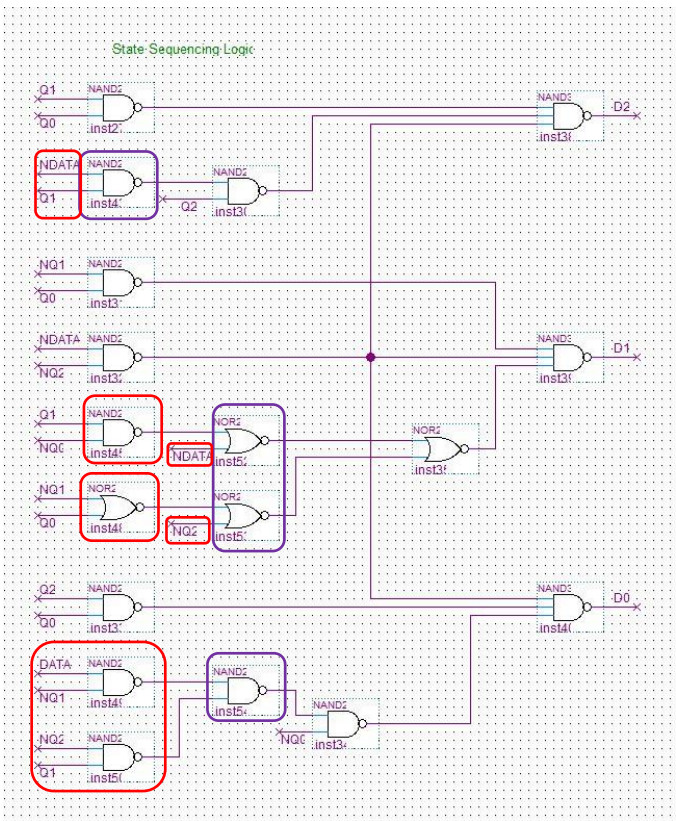
I took the following steps to simplify my circuit



Firstly, using De Morgan's law, I can change all three NOR3 gates to NAND3 gates given that I not their inputs.

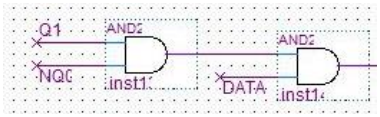
This results in the creation of 6 NAND gates and a NOR gate.

All of these are circled

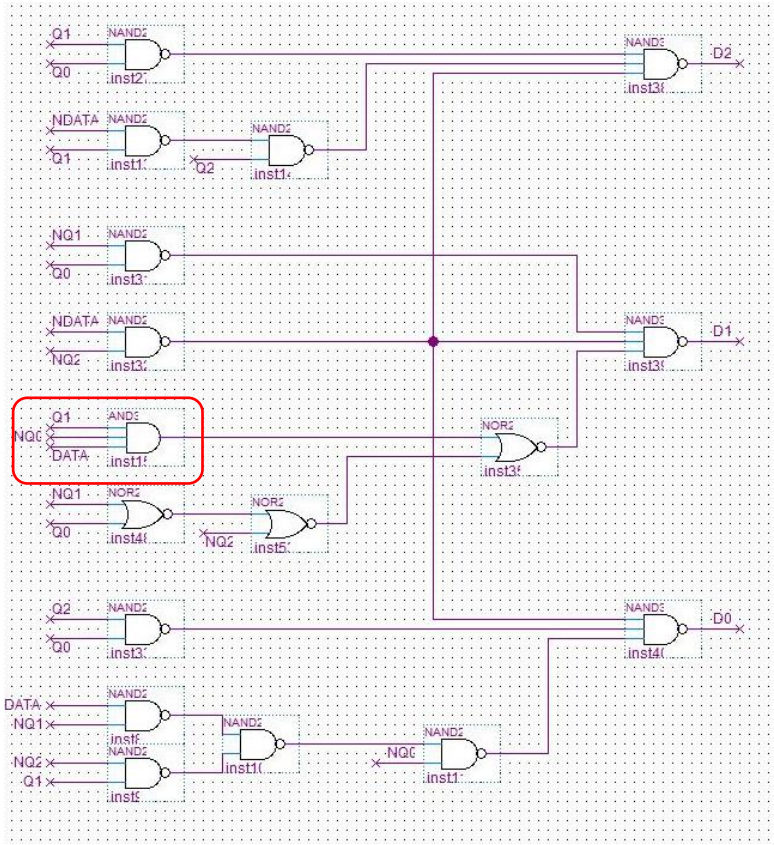


I then used the exact same principles to convert AND gates into NOR gates and as a result got many more NAND and NOR gates from my circuit. The purple represents the conversion from AND to NOR or OR to NAND, and the red represents the negation required to carry this out.

The final simplification comes from a realistation that AND3 gates are normally far better than two NAND2 gates or similar. At the start it can be noted that I had a AND2 gate's output feeding into another AND2 gate:



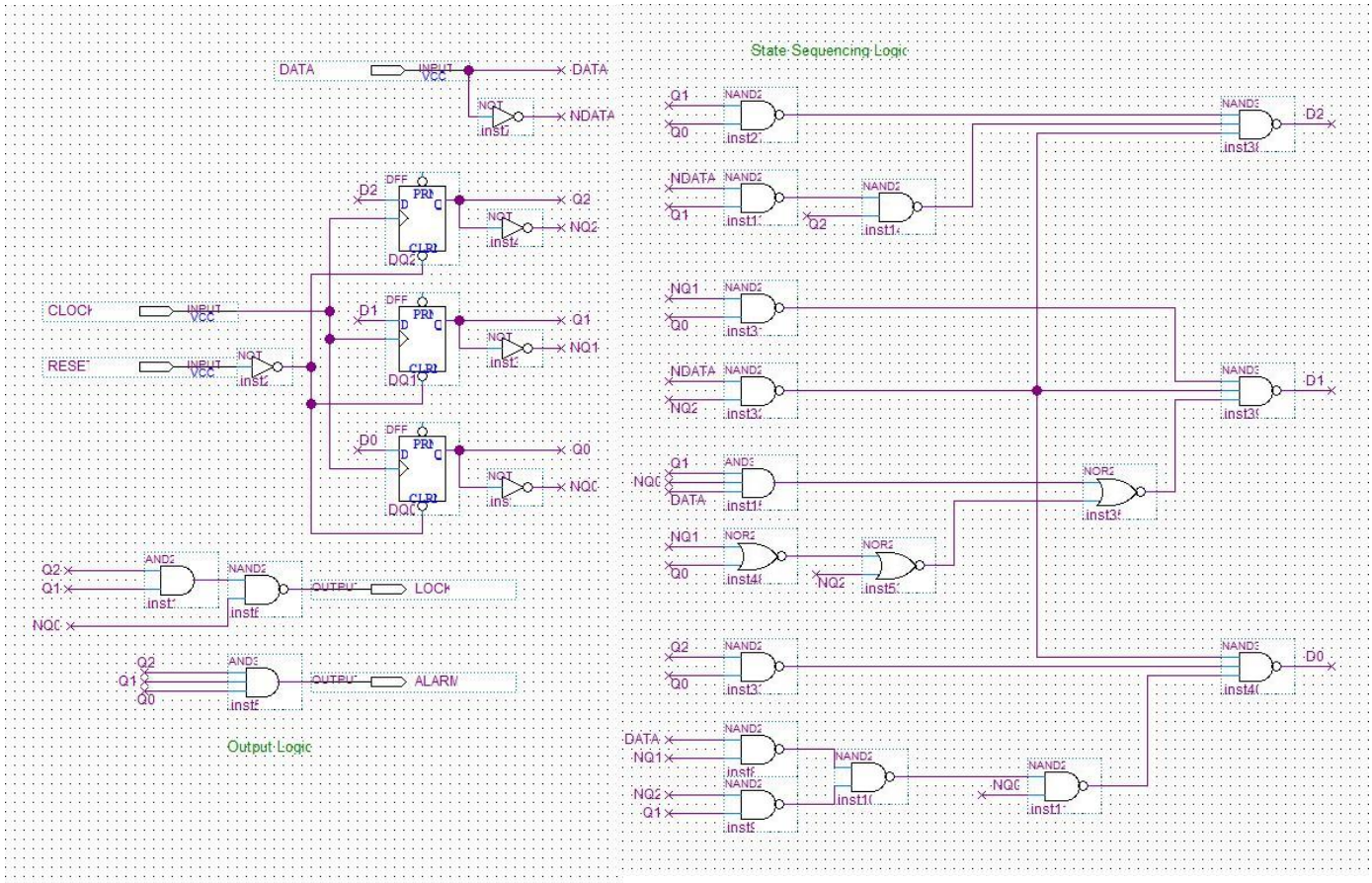
This should instead have been an AND3 gate and as a result, we should backtrack to step 2. There, I convert the AND2 gate into a NOR2 gate but if already had an AND3 gate there, that would save a lot more space. Hence, the circuit is changed to become this (see next page):



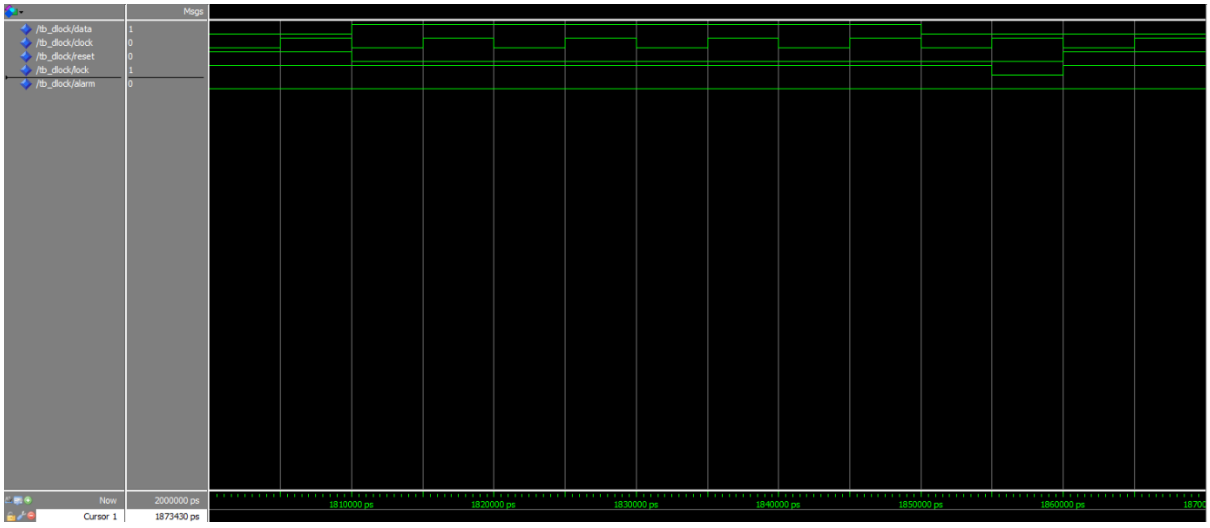
And so, the circuit is now in its simplest form. The calculation for area can now be made:

Gate Name	Amount	Total Area
DFF	3	300
INPUT	3	0
OUTPUT	2	0
NOT	5	75
AND2	1	27
AND3	2	68
NAND2	11	220
NAND3	3	84
NOR2	3	57
	TOTAL	831

Final Circuit (Images also included in the .tar.gz file under the Images folder)



ModelSim Zoomed into 11110



ModelSim full