ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY
HOMEWORK 2

HOMEWORK NO	:	2
HOMEWORK DATE	:	7.04.2023
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G8

GROUP MEMBERS:

150200919 : Abdullah Jafar Mansour Shamout

150220762 : Muhammed Yusuf Mermer

SPRING 2023

Contents

1 INTRODUCTION

In this experiment we implemented SR-Latch, D flip-flop, JK flip-flop using only NAND gates. We used 4 JK flip-flop to make synchronous and asynchronous up counters with 4-bits. In synchronous, we connected all of them to clock signal. However for the asynchronous, we connected one's input to another. In the last part we implemented 16 bit circular left shift register with parallel load.

2 PRELIMINARY

2.1 Question 1)

flip-flops are sequential logic circuits that are characterized with being bistable. Main reason behind the use of flip-flops is they can store information. To store information, they have loops inside. They will keep their values until the next edge occurs, so it will prevent unexpected asynchronous changes. And you can change inputs for the next cycles easily. One can use the output in the other circuits without giving same signals always.

2.2 Question 2)

The difference between Latch and Flip-Flop is about time of the change sensed in the output. In latches output changes whenever the one of the input changes. That is why they are called level triggered. However, it is not simple to say it for Flip-Flops. Beside of the input changes, to see the effect in output of the flip-flops, there should be edges in clock signal (high to low or low to high). With this feature, output of the flip-flops will change only in the next clock cycles. It will prevent instant series of effects in the outputs. So we can say flipflops are edge triggered while latches are level triggered.

2.3 Question 3)

The SR latch is a bistable memory element that has two inputs set(S) and reset(R). The structure of the SR latch can be made multiple ways, the way we did it was by taking two NAND gates and connecting each ones output to the others input as a feedback system. That means that one NAND gate will have SET as input and the other NAND gate's output as input, while the other will have RESET as an input and the other NAND gate's output as input. The NAND based SR latch is an active low, that means when we give a 0 to set that means we are setting Q to 1, likewise, when we give 0 to reset we are setting Q to 1. This happens due to the properties of the NAND gate, whenever there is a 0 input the output is forced to be 1 regardless of the other input. This is also the

reason why we can't give input as 0 0, because this results in Q and ∼Q having the same value which is invalid. However if we give 1 1, that makes the output fully dependent on the other input which makes a no change in the output, thus the same value is retained.

2.4 Question 4)

S	R	Q	∼Q
0	0	1/invalid	1/invalid
0	1	1	0
1	0	0	1
1	1	no change	no change

0 0 input is invalid because Q and ∼Q are the same when they should be opposite.

2.5 Question 5)

enable	S	R	Q	∼Q
0	X	X	no change	no change
1	0	0	no change	no change
1	0	1	0	1
1	1	0	1	0
1	1	1	1/invalid	1/invalid

1 1 input is invalid because Q and ∼Q are the same when they should be opposite.

2.6 Question 6)

clock	D	Q	∼Q
-	X	no change	no change
↑	0	0	1
↑	1	1	0

2.7 Question 7)

clock	J	K	Q
–	X	X	Q0 (no change)
↑	0	0	Q0 (no change)
↑	0	1	0
↑	1	0	1
↑	1	1	∼Q0 (toggle)

3 EXPERIMENT

3.1 Part 1

we can make a Kmap to find the characteristic equation of a SR-latch. As it can be seen in the truth table of the SR-latch in the preliminary section, we can see that if we give 0 0 we get an invalid state for our Q regardless of our previous Q value so we dont care about the output of an invalid input. for 0 1 we get Q as 1 regardless of its previous state, and for 1 0 we get Q as 0 regardless of its previous state, and for 1 1 we get Q meaning the same value as it was. if we plot those values on a kmap we get:

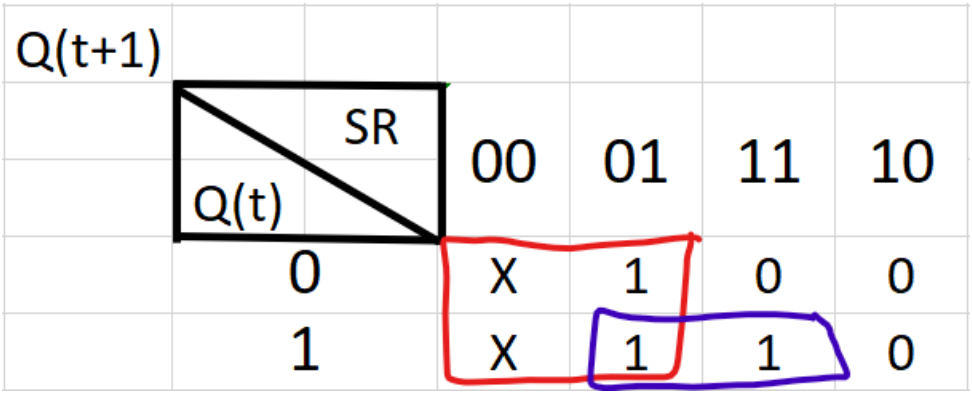

Figure 1: SR-latch kmap

from the kmap we realize the characteristic equation will be Q(t+1) = S' + Q(t)R. This is for the NAND gate latch which is an active low, so if we were to give inputs as S' and R' to the latch instead of S and R. we would get the NOR gate characteristic equation which is Q(t+1) = S + Q(t)R'

In the results section we see that when we give 1 1 without any previous inputs we get X X. which is a result of not changing the value of the latch, but since the latch has no presaved value we get this nondeterministic behaviour. And for input 0 0 even though it shows 1 1. this is a forbidden input that should be avoided.

3.2 Part 2

we can make a Kmap to find the characteristic equation of the enabled SR-latch. As it can be seen in the truth table of the enabled SR-latch in the preliminary section, we can see that if we give 0 to enabled and any SR or 0 0 to S and R and 1 to enabled we get an no change for our Q regardless of our previous Q value so the output is whatever Q was. For 0 1 to S and R we get Q as 0 regardless of its previous state so we reset it, and for 1 0 we get Q as 1 regardless of its previous state so we set it, and for 1 1 we get an invalid output for Q meaning that we dont care about its output. if we plot those values on a kmap we get:

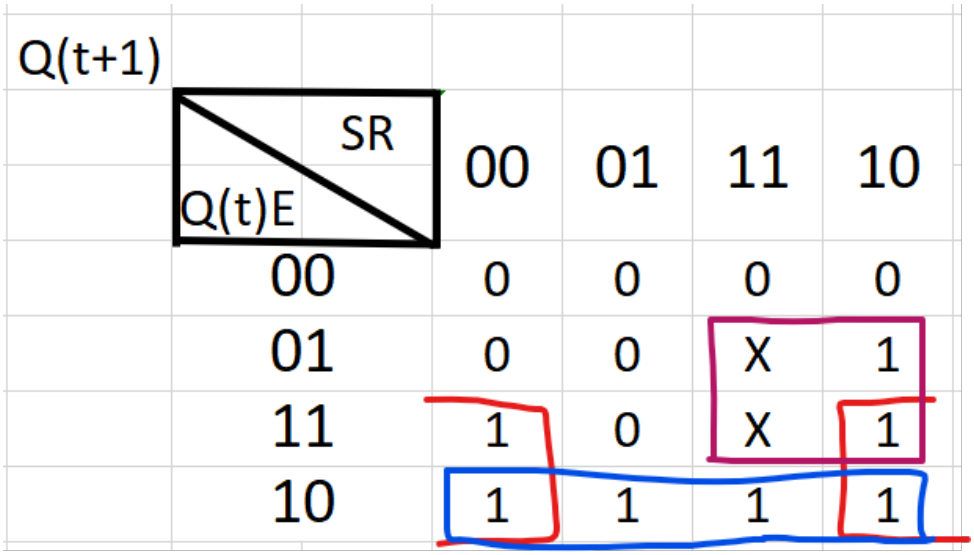

Figure 2: enabled SR-latch kmap

from the kmap we realize the characteristic equation will be:

Q(t+1) = Q(t)R' + Q(t)E' + SE.

we can see that this characteristic equation is different than the characteristic equation of a normal NAND SR-latch but same with NOR SR-latch when enable is set to 1 and when enable is set to 0 we get Q(t).

In the results section we see that when we give 0 for enabled or 0 0 to S and R without any previous inputs we get X X. which is a result of not changing the value of the latch, but since the latch has no presaved value we get this nondeterministic behaviour. And for input 1 1 to S and R even though it shows 1 1. this is a forbidden input that should be avoided. The enabled SR-latch with NAND gate implementation is an active high circuit.

3.3 Part 3

We implemented a negative edge triggered D-flipflop from enabled D-latches in this part. To implement it we used a master and slave configuration with an inverted clock to the slave, thus the output would come at NOT clock. The schematics and the simulation van be seen in the results section

3.4 Part 4

for the design of a positive edge triggered JK flipflop Using a NAND configured SR flip flop is not possible in structural verilog and would result in a XX state that is why the only other way is to either use behavioral verilog, which we were instructed not to use or use a D flipflop which is what we did. For the implementation of the JK flipflop using a D flipflop we utilized its truthtable and its excitation table to create a conversion table between it and the JK, then we drew its kmap and found this expression D = K'Q + JQnot, then using nand modules we implemented that expression and got our JK-flipflop

3.5 Part 5

In the design of asynchronous 4-bit up counter, except the last one, we connected all of the flip-flops to the clock of the next one.

Implementation of this circuit is little bit trickier as to activate all of the flip-flops, we need to initialize them to either to 0 or 1. To do this we need to give J=0 and K=1 or J=1 and K=0. However, giving just insert constant J and K does not help to activate all flip flops as we need waves in outputs to active other flip-flops. For this reason, we gave different, fluctuating values to all the JK flip-flops. We created waves in the outputs to activate consequent flip flops. In here also clock frequency is little bit higher because we wanted to activate all flip-flops very fast.

Clock of the First Flip-Flop	Initial Binary	Initial Decimal	Final Binary	Final Decimal
↑	0000	0	0001	1
↑	0001	1	0010	2
↑	0010	2	0011	3
↑	0011	3	0100	4
↑	0100	4	0101	5
↑	0101	5	0110	6
↑	0110	6	0111	7
↑	0111	7	1000	8
↑	1000	8	1001	9
↑	1001	9	1010	10
↑	1010	10	1011	11
↑	1011	11	1100	12
↑	1100	12	1101	13
↑	1101	13	1110	14
↑	1110	14	1111	15
↑	1111	15	0000	0

Table 1: Truth Table of Asynchronous Up Counter

3.6 Part 6

Synchronous up counter is relatively easier compered to asynchronous up counter. We directly connected all clock's inputs to the same pulse, therefore we can activate all flip-flops with giving same JK values. But JK values also depend on the outputs of the former flip-flops. So in order to combine them, we used or gate. We used or gate because we want to assert K values to be 1 at the initial state to activate all of the flip flop with 0 outputs. Then we gave 0 for J and K values except the LSB to not interfere the process. In LSB flip-flop as inputs we gave J=1 and K=1 we need a toggling in the first flip flop. If output of all former flip-flops are 1 then current flip-flop's value will be toggled. With this implementation we successfully made synchronous up counter.

Clock (same for all)	Initial Binary	Initial Decimal	Final Binary	Final Decimal
↑	0000	0	0001	1
↑	0001	1	0010	2
↑	0010	2	0011	3
↑	0011	3	0100	4
↑	0100	4	0101	5
↑	0101	5	0110	6
↑	0110	6	0111	7
↑	0111	7	1000	8
↑	1000	8	1001	9
↑	1001	9	1010	10
↑	1010	10	1011	11
↑	1011	11	1100	12
↑	1100	12	1101	13
↑	1101	13	1110	14
↑	1110	14	1111	15
↑	1111	15	0000	0

Table 2: Truth Table of Synchronous Up Counter

3.7 Part 7

In positive edge triggering pulse generator, for each bit we used 2 AND gates 1 OR gates. We used AND gates because if load value is given shifting will not work but loading will work. We ORed result of the both and gates. Then when load flag turns back to 0, our design starts to do shifting. In here direct on shift operations are to the left, so that output of the circuit will be the MSB which is turned back to the LSB, later.

To obtain variable pulse frequencies and duration, we load new values to the system at different test benches.

For instance to get 1/2 frequency of clock we need to make sequence as 101010... because at every positive edge (which is one cycle of the clock), our pulse generator will return from one to zero or zero to one, which is half cycle because we don't return to the initial state. Clock frequency's 1/4 can be made by doubling consistency of a state like 110011, so this means that two clock cycle needed to convert 1 to 0 and 0 to 1.

Pulse gap duration rate is little bit different. Sequence of time spent for 0's and 1's are not the same. For 1/7 pulse gap we will spent 1 time for value 1 and 7 for 0. 10000000.....

4 RESULTS

4.1 Schematics and Simulations

4.1.1 PART 1

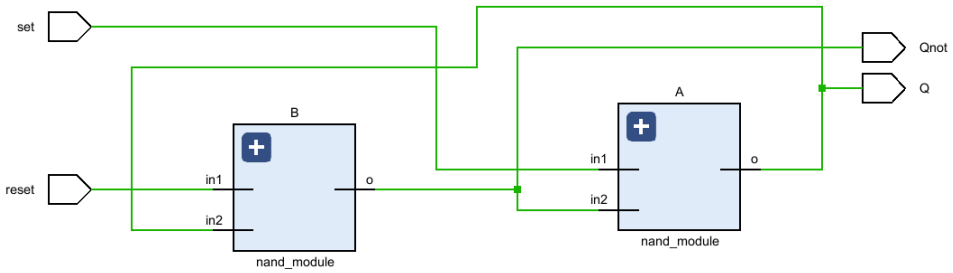

Figure 3: SR-latch schematic

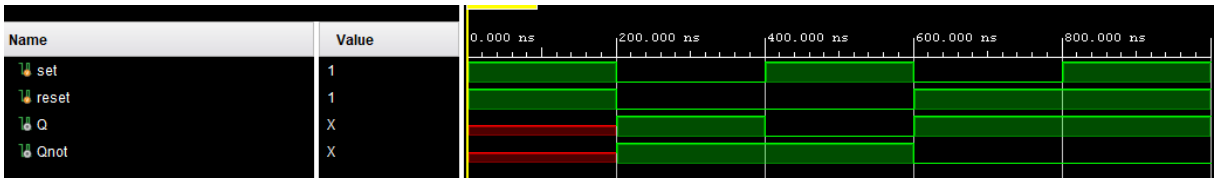

Figure 4: SR-latch simulation

4.1.2 PART 2

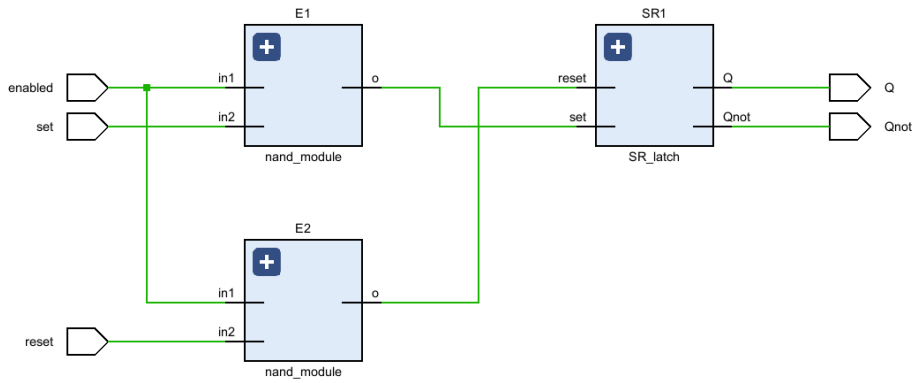

Figure 5: enabled SR-latch schematic

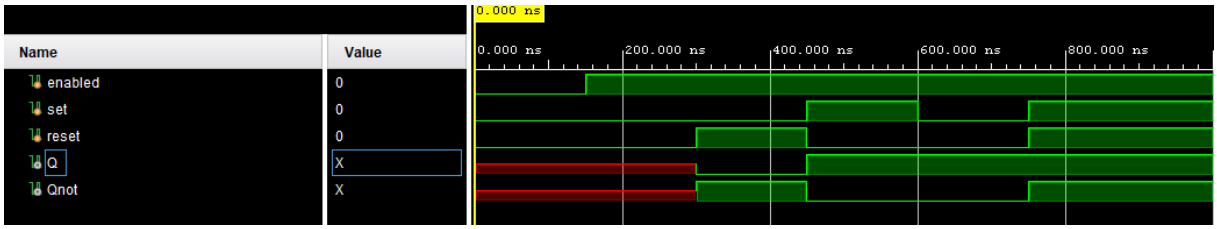

Figure 6: enabled SR-latch simulation

4.1.3 PART 3

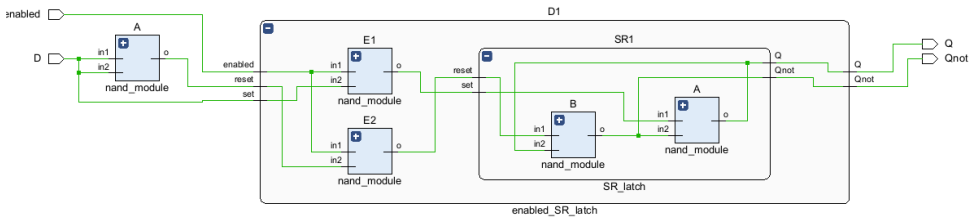

Figure 7: enabled D-latch schematic

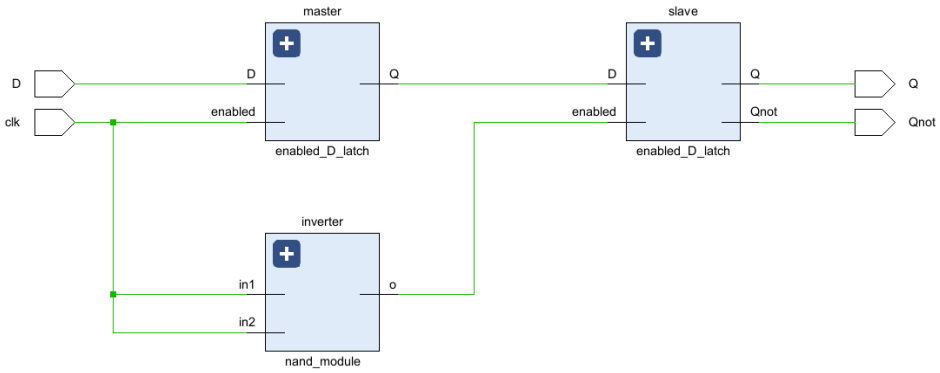

Figure 8: D-flip-flop schematic

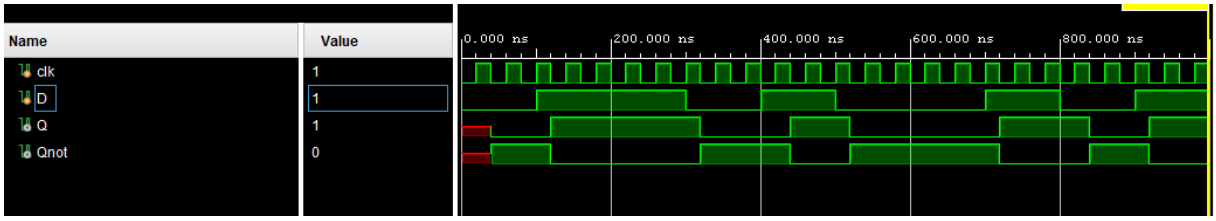

Figure 9: D-flip-flop simulation

4.1.4 PART 4

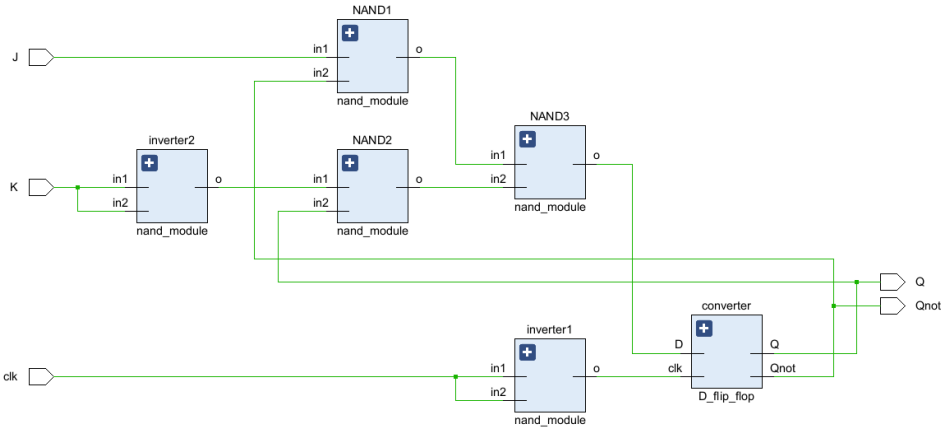

Figure 10: JK-flip-flop schematic

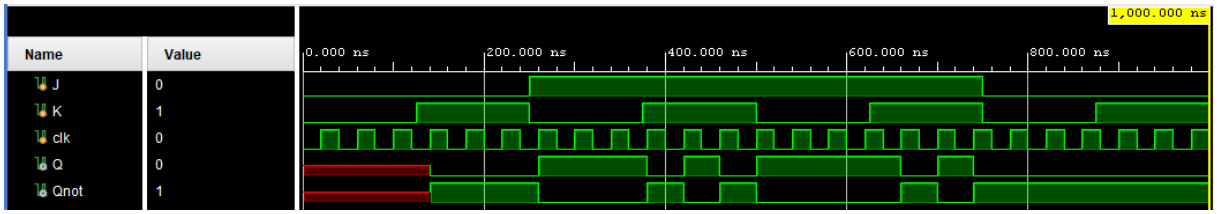

Figure 11: JK-flip-flop simulation

4.1.5 PART 5

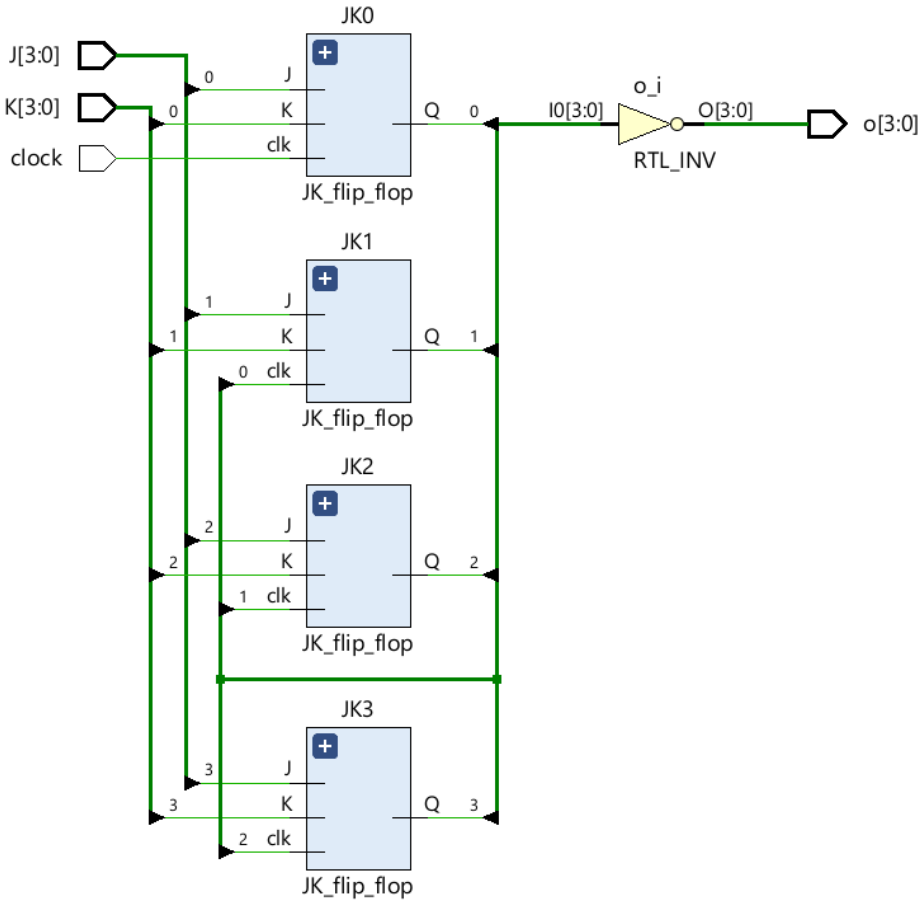

Figure 12: Asynchronous Up Counter

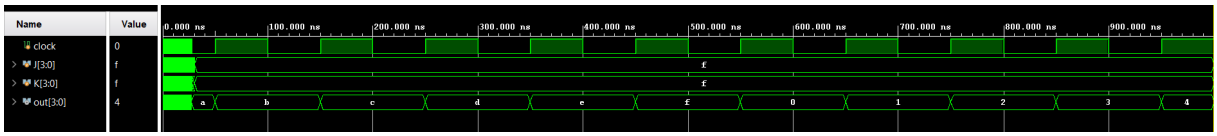

Figure 13: Asynchronous Up Counter simulation

4.1.6 PART 6

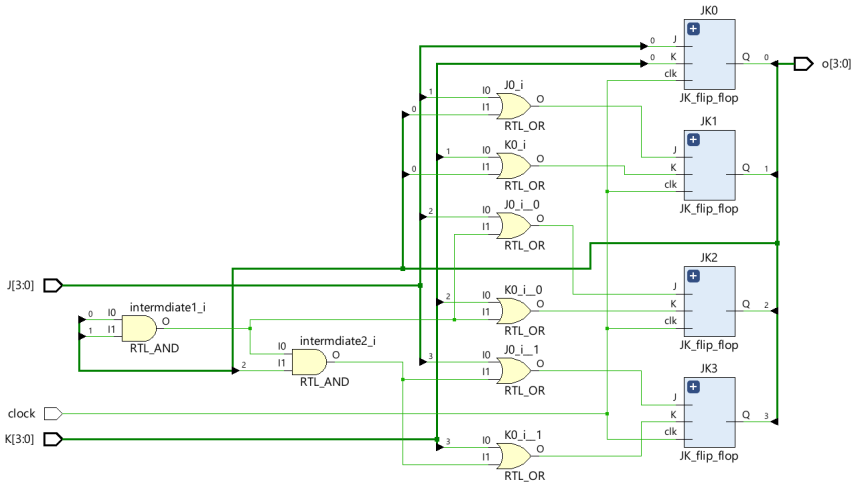

Figure 14: Synchronous Up Counter

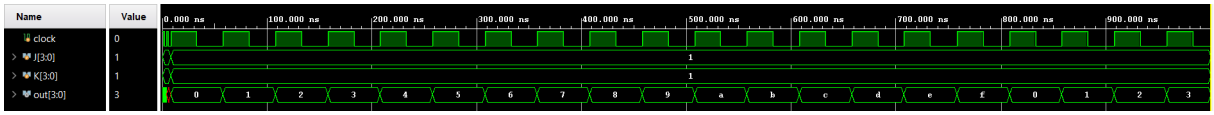

Figure 15: Synchronous Up Counter simulation

4.1.7 PART 7

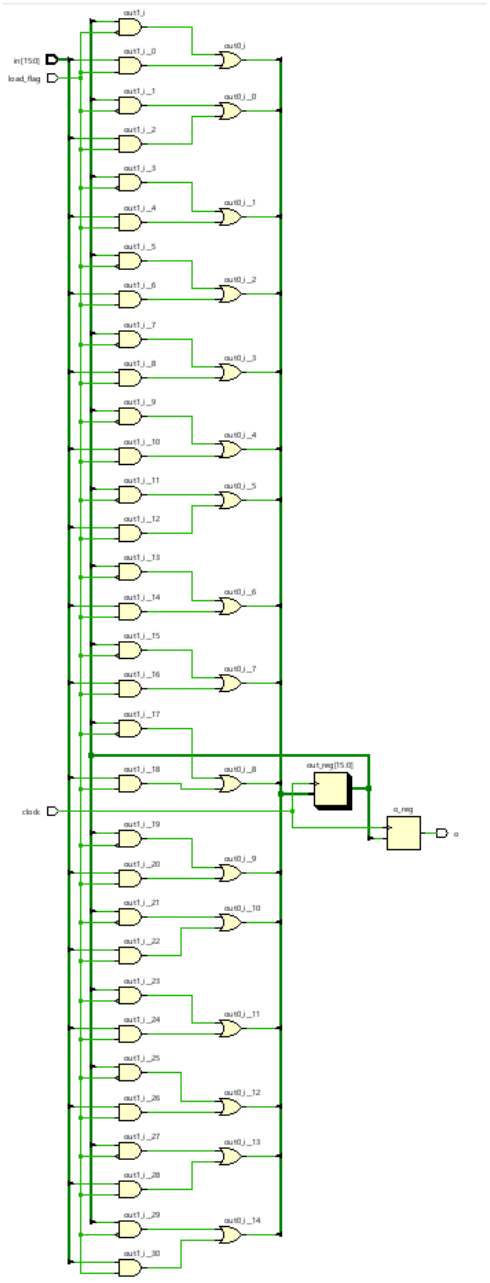

Figure 16: Positive Edge Triggered Pulse Generator

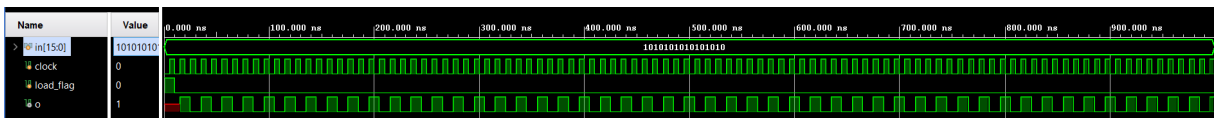

Figure 17: Positive Edge Triggered Pulse Generator with 1/2 frequency of clock

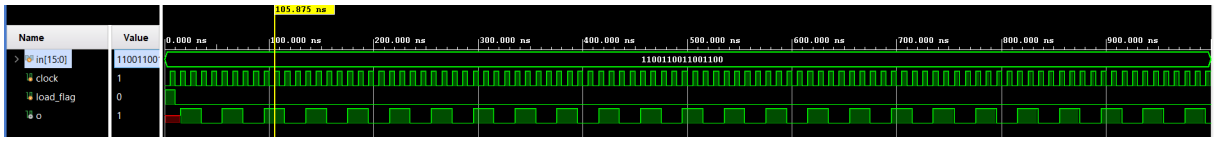

Figure 18: Positive Edge Triggered Pulse Generator with 1/4 frequency of clock

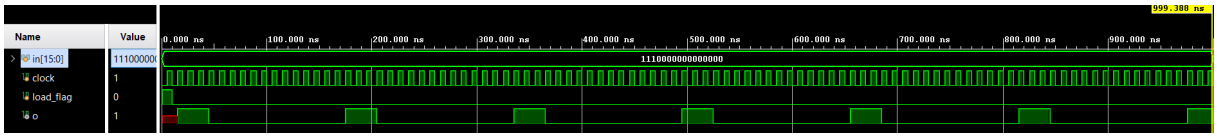

Figure 19: Positive Edge Triggered Pulse Generator with 3/13 pulse gap duration rate

5 CONCLUSION

In this homework, we learned to implement basic latches and flip-flops using NAND gates. Such as SR latch, SR latch with enabled, enabled D-latch, D-flip-flop, JK flip-flop. We had struggle in at first SR flip-flop's as S=1 and R=1 condition have to be avoided. Then especially in JK flip-flop we tried lots and lots of combinations of NAND gates to obtain correct results. However also by discussion with other groups, we come to a result that JK flip-flops cannot be implemented by SR flip-flop made from NAND gates. Therefore, we designed it with D flip-flop instead. Truth table that obtained after simulations are satisfied our expectation.

Then we designed asynchronous and synchronous up counters. For asynchronous, in the test case we gave initial values intentionally to activate all flip-flops properly. In the synchronous we give initial inputs to J's and K's of flip flop's. At the end we got all result like we want from the design.

At the very end question, we designed pulse generator. In here as we can use always block in our module, implementation of the last question was easier. However we had struggle at test bench as we did not get how we can obtain frequencies and gap durations. We thought that it is about differentiation in the clock of pulse generator. Then we found that it is about changes in input load, then we changed our test bench and gave loadFlag=1 just for a small period of time. Then we saw that the pulse generator gives sequential and correct pulses in the simulation. We made our clarifications and saw that our results were correct.