Report from Lab 2

J-K flip-flop

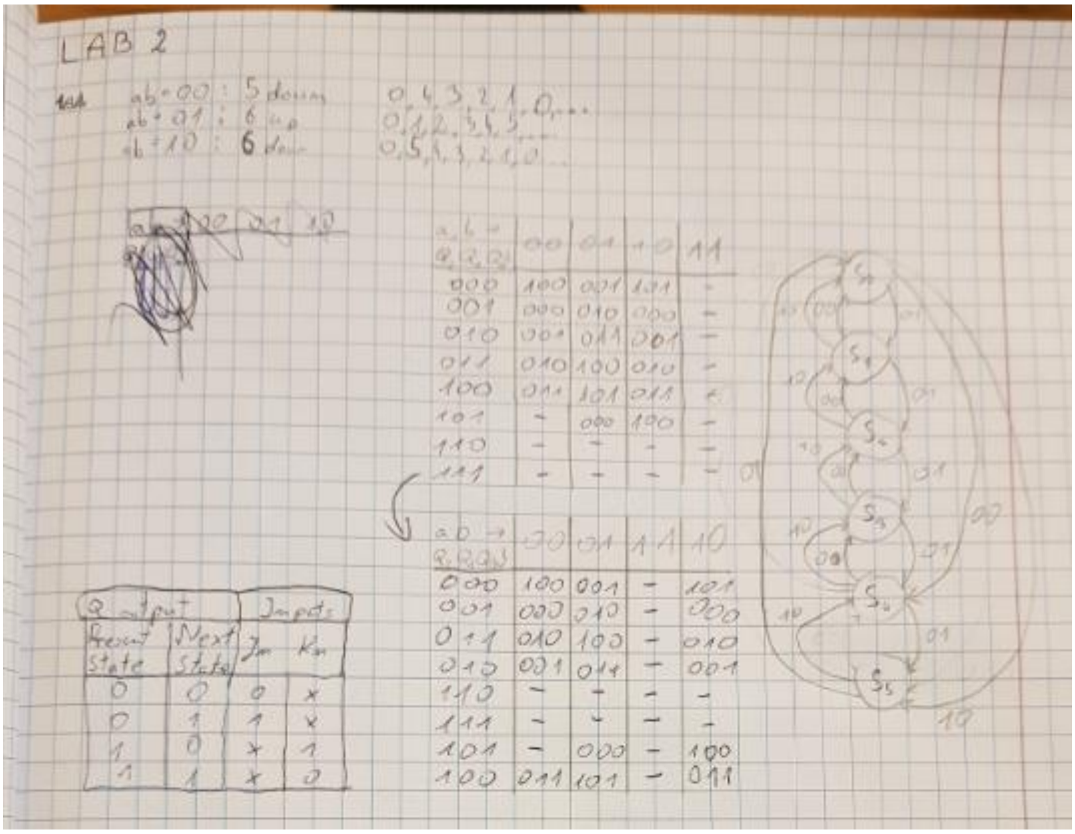
I declare that this piece of work, which is the basis for recognition of achieving learning outcomes in the Digital Circuits course, was completed on my own.

First and last name: Michał Łezka
Student record book number (Student ID number): 303873
Date: 23.04.2021

Theoretical design on paper

I have started from designing a graph based on the provided data, which was:
For ab = 00 : modulo 5, DOWN
For ab = 01 : modulo 6, UP
For XY = 10 : modulo 6, DOWN

First, I have created the graph shown on the photo below on the right. Then, based on that, I have created table on the top of the photo, which shows what state happens after which state and for what a and b input. After that, I have rewritten this table, but change the order to fit the Karnaugh table.



LAB 2

ab=00 : 5 down
ab=01 : 6 up
ab=10 : 6 down

0,4,3,2,1,0,...
0,1,2,3,4,5,0,...
0,5,4,3,2,1,0...

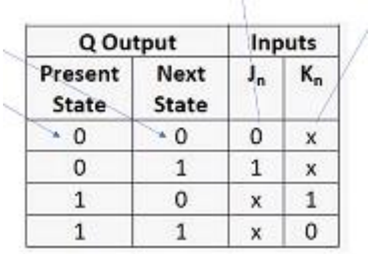
a, b → Q2Q1Q0	00	01	10	11
000	100	001	101	–
001	000	010	000	–
010	001	011	001	–
011	010	100	010	–
100	011	101	011	–
101	–	000	100	–
110	–	–	–	–
111	–	–	–	–

ab → Q2Q1Q0	00	01	11	10
000	100	001	–	101
001	000	010	–	000
011	010	100	–	010
010	001	011	–	001
110	–	–	–	–
111	–	–	–	–
101	–	000	–	100
100	011	101	–	011

Q output		Inputs	
Present State	Next State	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

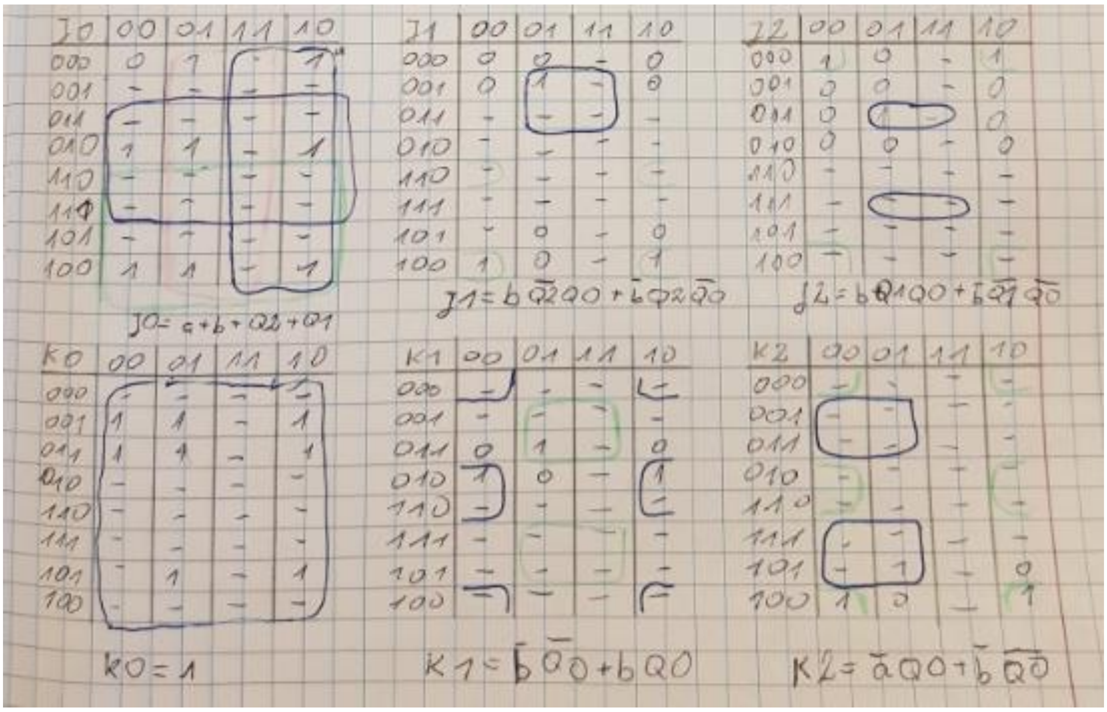
Then, I have transcoded the state transition table. I had to transcode in total 6 times, for J0, J1, J2, K0, K1, K2. For the J and K with 0, to transcode it I look at the least significant bit in the State category (Q0) and the least significant bit in the body of the table, that is in the same row. Then I looked at the J-K flip-flop table shown below and transcoded it.

The table for transcoding:



Q Output		Inputs	
Present State	Next State	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Those are the results for the transcoded tables:



J0	00	01	11	10
000	0	1	-	1
001	-	-	-	-
011	-	-	-	-
010	1	1	-	1
110	-	-	-	-
111	-	-	-	-
101	-	-	-	-
100	1	1	-	1

J0= a+b+Q2+Q1

J1	00	01	11	10
000	0	0	-	0
001	0	1	-	0
011	-	-	-	-
010	-	-	-	-
110	-	-	-	-
111	-	-	-	-
101	-	0	-	0
100	1	0	-	1

$J1 = b\overline{Q2}Q0 + \bar{b}Q2\overline{Q0}$

J2	00	01	11	10
000	1	0	-	1
001	0	0	-	0
011	0	1	-	0
010	0	0	-	0
110	-	-	-	-
111	-	-	-	-
101	-	-	-	-
100	-	-	-	-

$J2 = bQ1Q0 + \bar{b}\overline{Q1}\overline{Q0}$

K0	00	01	11	10
000	-	-	-	-
001	1	1	-	1
011	1	1	-	1
010	-	-	-	-
110	-	-	-	-
111	-	-	-	-
101	-	1	-	1
100	-	-	-	-

K0 = 1

K1	00	01	11	10
000	-	-	-	-
001	-	-	-	-
011	0	1	-	0
010	1	0	-	1
110	-	-	-	-
111	-	-	-	-
101	-	-	-	-
100	-	-	-	-

$K1 = \bar{b}\overline{Q0} + bQ0$

K2	00	01	11	10
000	-	-	-	-
001	-	-	-	-
011	-	-	-	-
010	-	-	-	-
110	-	-	-	-
111	-	-	-	-
101	-	1	-	0
100	1	0	-	1

$K2 = \bar{a}Q0 + \bar{b}\overline{Q0}$

After finding most optimised groups, I got the following results:

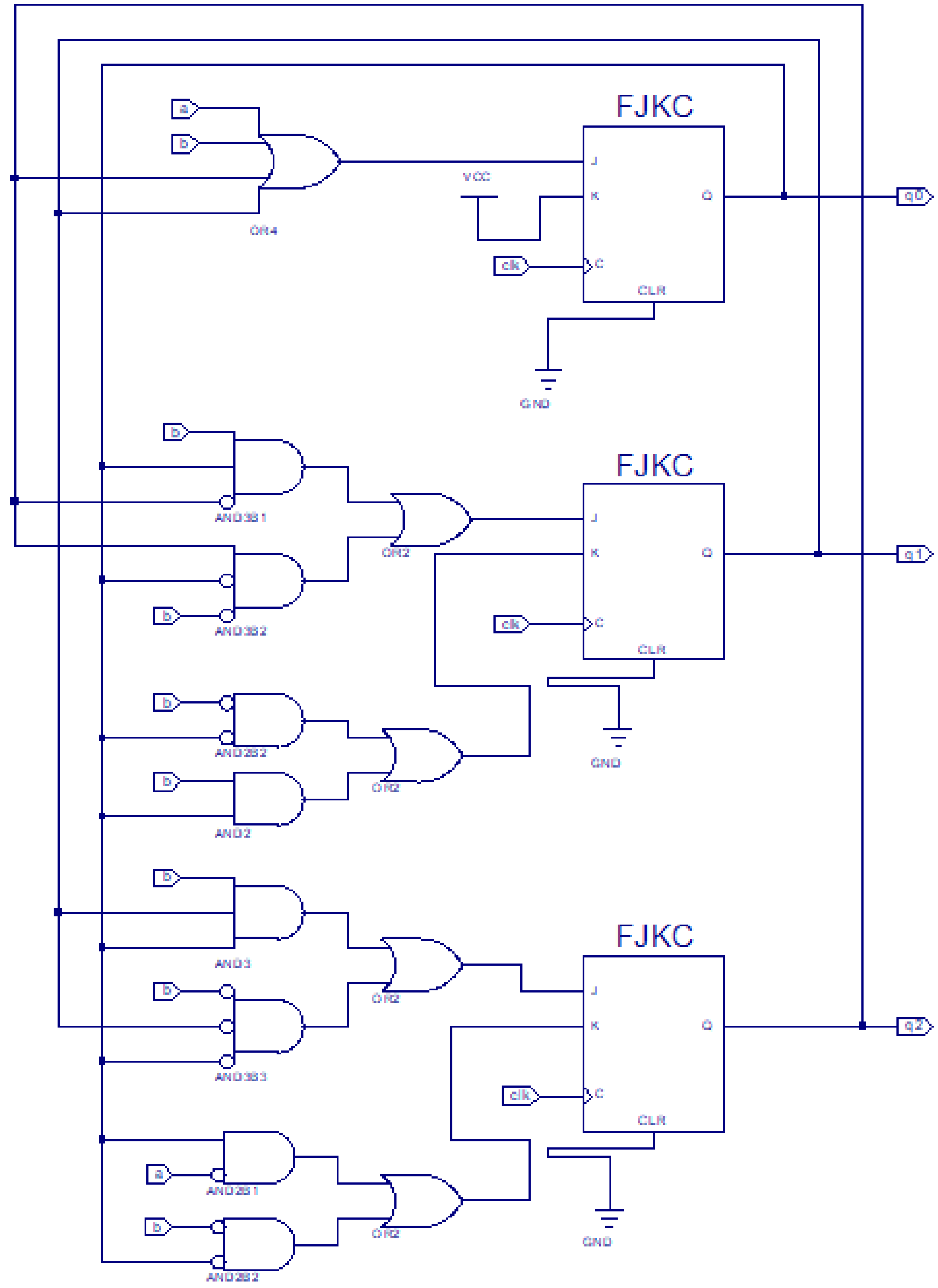
J0 = a + b + Q2 + Q1 J1 = b~Q2Q0 + ~bQ2~Q0 J2 = bQ1Q0 + ~b~Q1~Q0

K0 = 1 K1 = ~b~Q0 + bQ0 K2 = ~aQ0 + ~b~Q0

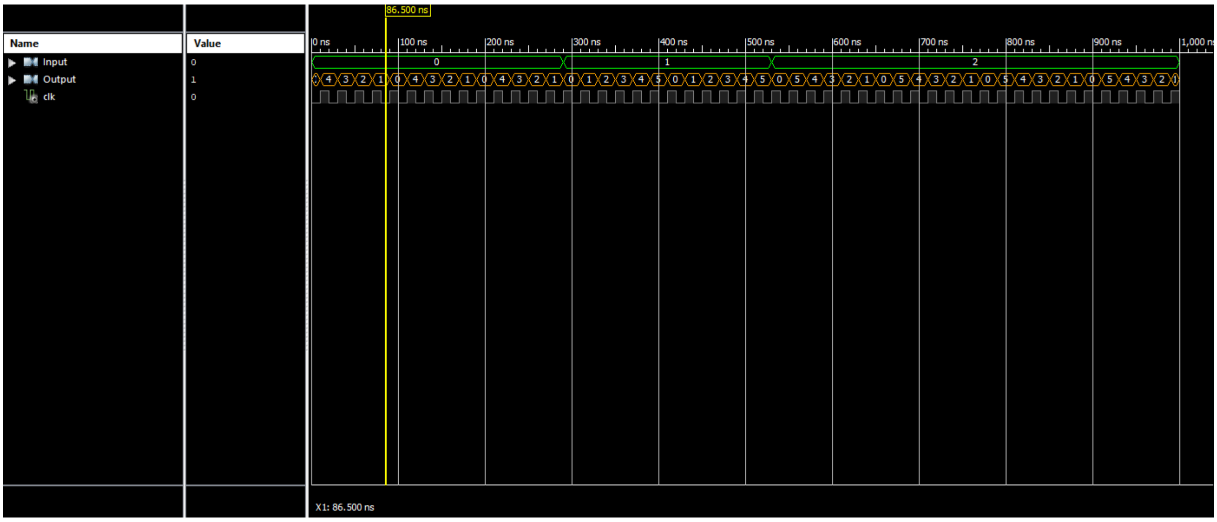
Schematics and test results

Having the Karnaugh tables solved, it was simple to implement it in the Xilinix as a few "and" and "or" gates. Because you can't add output as an input, if I needed Q0, Q1 or Q2 as input I drew a wire from the proper flip-flop and connected it to the input where needed. Clock was set up a little different than it was during presentation, as clock was set to 10ns instead of 50 because the results weren't really visible with the slower clock.

Below is presented how the schematic looked like:



And this is how the testbench looked like:



The received answer was according to the expectations and it shows perfectly how the clock works. To achieve that, I have created 2 buses, one with inputs a and b, and one with outputs Q2, Q1 and Q0.

Conclusions

Overall, the experiment succeeded. Results were as expected, and I had no issues performing the task. Thanks to J-K flip-flops there was a lot of don’t cares, which significantly reduces the complexity of the device.