Lebanese American University

LAU
School of Engineering

Digital Systems Lab

Section 33

Laboratory manual 3

Nour Kachmar: 201902597

Youssef Kourani: 202004265

13/10/2023

Table of Contents

List of Figures

List of Tables

Introduction

In this lab, we will first create a 4-bit counter using flip-flops and computational logic with the help of state diagrams, state tables, Kmaps and simulate it on Quartus as well as try it on the FPGA. After that, we will create a Digital Lock in the second experiment using the same steps as experiment one.

Experiment 1

In this experiment, we are going to design a 4-bit counter using flip-flops and computational logic. When the enable is 1, the counter should count up and when the enable is 0, the counter should hold its value.

A reset option is available for the user; if it is set to 1, the counter restarts from the first state. If not, the counter checks the enable pin.

State Diagram

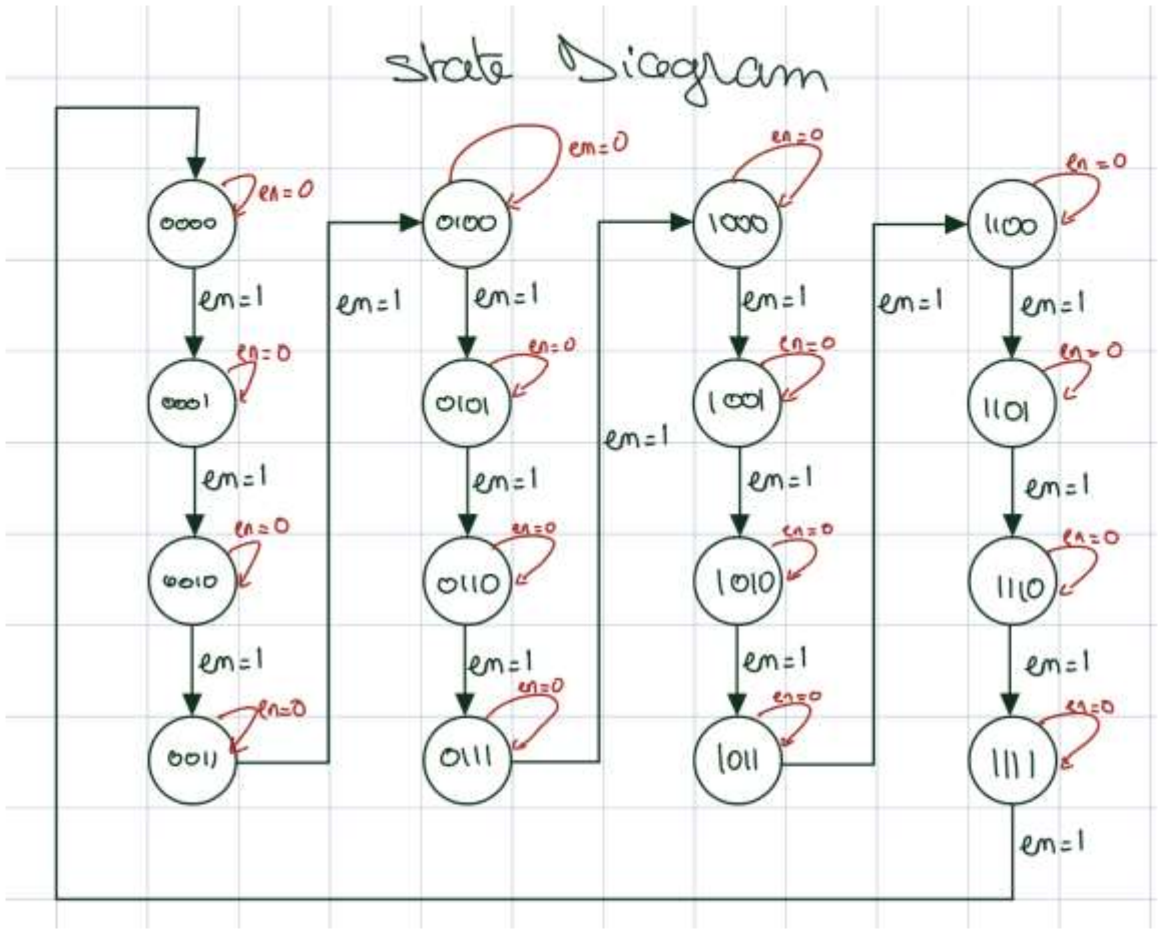


Figure 1- State Diagram of the 4-bit counter

Figure 1 represents the state diagram of a 4 bit-counter that will be built using D flip flops and logic gates. Since it is a 4-bit counter, we will need 2^4=16 states to do the counter. Thus, we will need 4 registers to represent the counter's values from 0 to 15 as shown in the state diagram.

Note that when the reset is 1, all of the states return back to the first one. However, it was not included in the state diagram for scalability purposes but will be included in the state table and so on.

Note that all these events only occur when the clock is on a rising edge.

State table

Table 1- State table for 4-bit counter

External Inputs		Present State				Next State				Output
Reset	Enable	Q_4	Q_3	Q_2	Q_1	D_4	D_3	D_2	D_1	Overflow Q_{flow}
1	X	X	X	X	X	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	1	0	0	1	1	0
0	0	0	1	0	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0	1	0
0	0	0	1	1	0	0	1	1	0	0
0	0	0	1	1	1	0	1	1	1	0
0	0	1	0	0	0	1	0	0	0	0
0	0	1	0	0	1	1	0	0	1	0
0	0	1	0	1	0	1	0	1	0	0
0	0	1	0	1	1	1	0	1	1	0
0	0	1	1	0	0	1	1	0	0	0
0	0	1	1	0	1	1	1	0	1	0
0	0	1	1	1	0	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0	1	1	0
0	1	0	0	1	1	0	1	0	0	0
0	1	0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	1	1	1	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	0	0	0	1	0	0	1	0
0	1	1	0	0	1	1	0	1	0	0
0	1	1	0	1	0	1	0	1	1	0
0	1	1	0	1	1	1	1	0	0	0
0	1	1	1	0	0	1	1	0	1	0
0	1	1	1	0	1	1	1	1	0	0
0	1	1	1	1	0	1	1	1	1	0
0	1	1	1	1	1	0	0	0	0	1

In this truth table, we presented the present states as well as the next states by going over all possible situations according to the enable's value and the reset.

K-map / State equations

Note that we did the first K-map by hand and all the remaining K-maps were built using online K-map simulators.

Table 2- Two Kmaps for output D1

When En=0

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

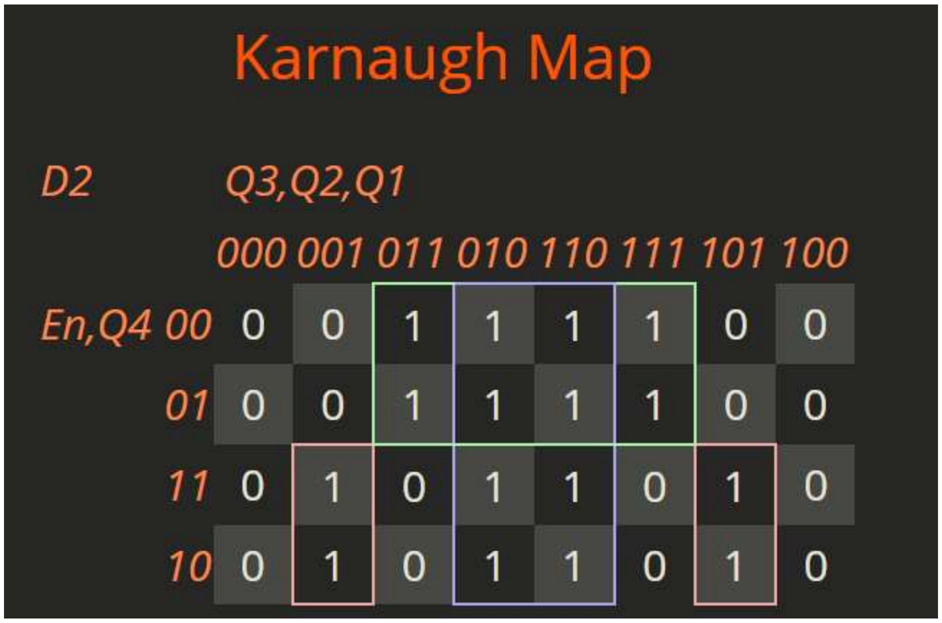
When En=1

Q_4Q_3 \ Q_2Q_1	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

From the K-maps above we get:

$$D1 = En'Q1 + EnQ1' = En \oplus Q1$$

Table 3- Kmap of output D2



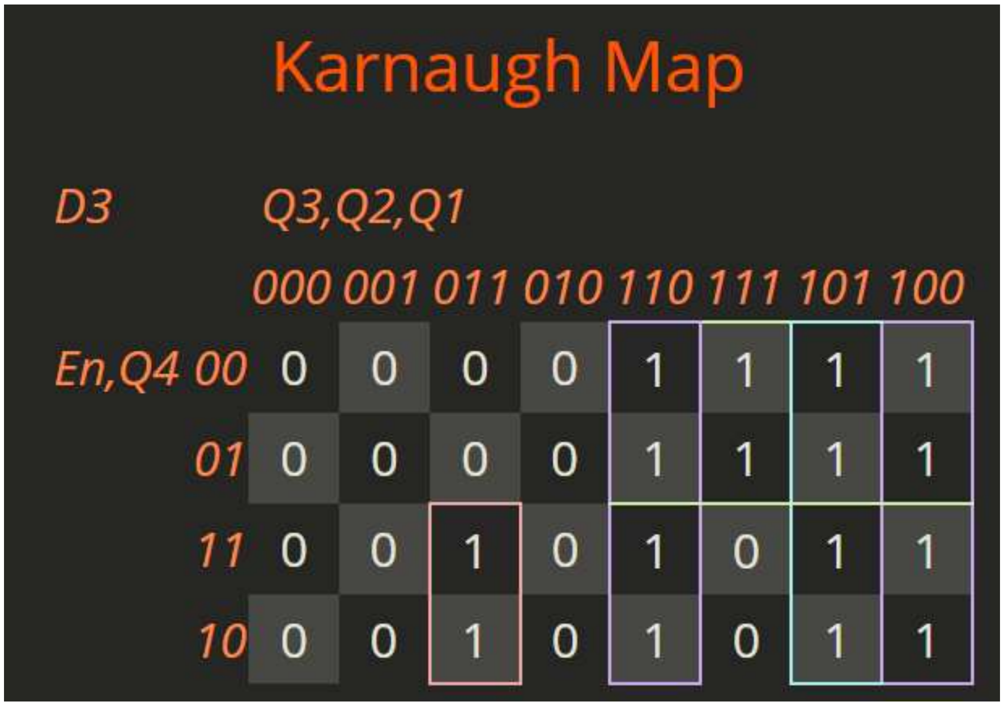
Karnaugh Map

D2	Q3,Q2,Q1							
	000	001	011	010	110	111	101	100
En,Q4 00	0	0	1	1	1	1	0	0
01	0	0	1	1	1	1	0	0
11	0	1	0	1	1	0	1	0
10	0	1	0	1	1	0	1	0

From Table 3 we get:

$$D2 = En'Q2 + EnQ2'Q1 + Q2Q1'$$

Table 4- Kmap of output D3


Karnaugh Map

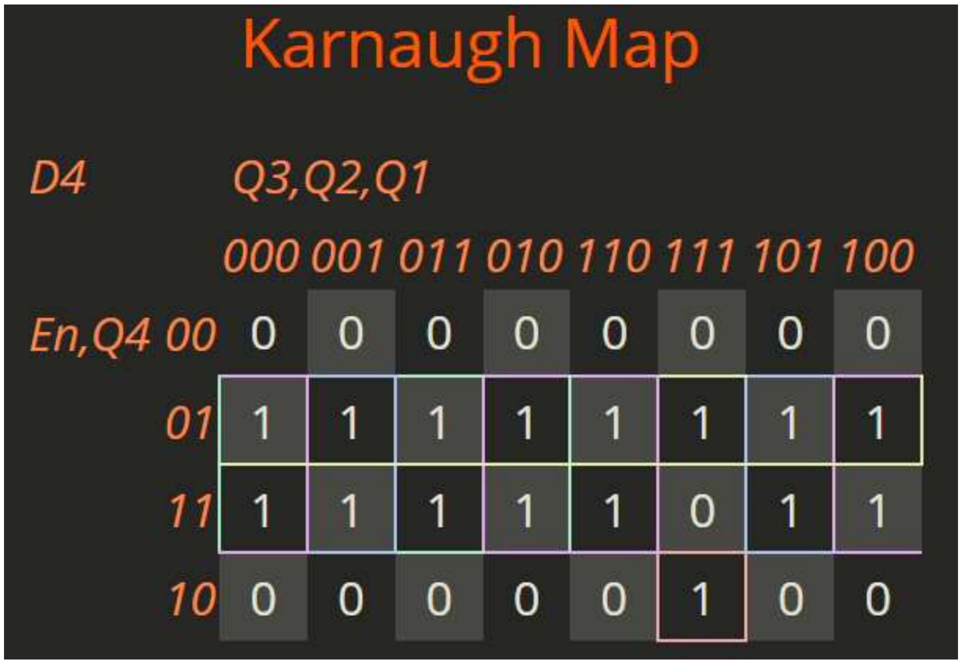
D3 Q3,Q2,Q1

En,Q4	000	001	011	010	110	111	101	100
00	0	0	0	0	1	1	1	1
01	0	0	0	0	1	1	1	1
11	0	0	1	0	1	0	1	1
10	0	0	1	0	1	0	1	1

From Table 4 we get:

$$D3 = En'Q3 + EnQ3'Q2Q1 + Q3Q2' + Q3Q1'$$

Table 5- Kmap of output D4


Karnaugh Map

D4 Q3,Q2,Q1

En,Q4	000	001	011	010	110	111	101	100
00	0	0	0	0	0	0	0	0
01	1	1	1	1	1	1	1	1
11	1	1	1	1	1	0	1	1
10	0	0	0	0	0	1	0	0

From Table 4 we get:

$$D4 = En'Q4 + EnQ4'Q3Q2Q1 + Q4Q3' + Q4Q2' + Q4Q1'$$

Finally, from truth table inspection, we get

$$Q_{overflow} = EnQ4Q3Q2Q1'$$

Since we got simplified equations from K-maps, we can now add the reset variable to these equations. In fact, we will consider these equations when the reset is null so that we can keep counting.

Our final equations will be the following:

$$D1 = (En \oplus Q1).Rst'$$

$$D2 = (En'Q2 + EnQ2'Q1 + Q2Q1').Rst'$$

$$D3 = (En'Q3 + EnQ3'Q2Q1 + Q3Q2' + Q3Q1').Rst'$$

$$D4 = (En'Q4 + EnQ4'Q3Q2Q1 + Q4Q3' + Q4Q2' + Q4Q1').Rst'$$

$$Q_{overflow} = (EnQ4Q3Q2Q1).Rst'$$

We will know implement the Quartus circuitry to get the required design.

For the next parts, consider o_0, o_1, o_2 and o_3 as the outputs of the D flip-flops and O_{flow} as the overflow.

Moreover, cnt_rst is the reset, en_sig is the enable signal and sys_clk is the clock.

Block diagram

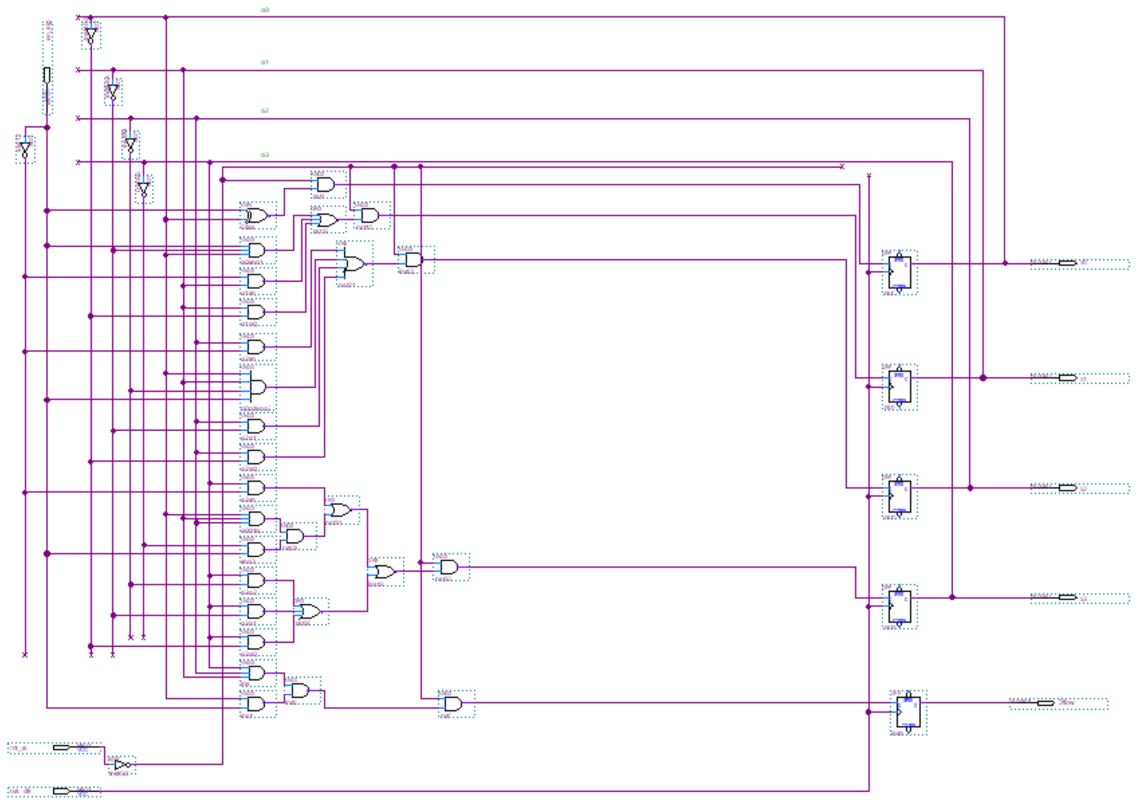


Figure 2- Block Diagram of the counter circuit

Figure 2 represents the block diagram of the counter circuit. We can see all the inputs with the flip flops and logical gates that were related using the state equations obtained in the previous part. If not visible, please refer to the block diagram named "counter" in the files submitted with this report.

Note that we added a register at O_{flow} so that the green LED turns on only when the previous state is 1111 or F and the next state is 0. This way, it will turn on when a reset occurs.

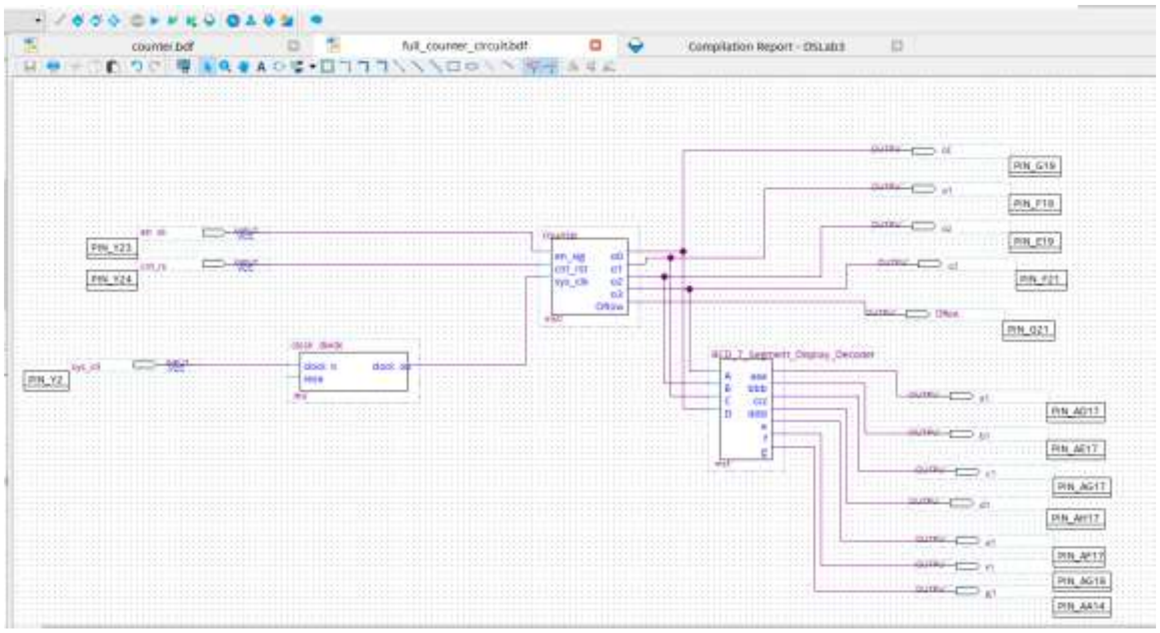


Figure 3- Block Diagram of the full counter circuit with BCD display

Figure 3 shows the block diagram named "full_counter_circuit" which is the representation of the final design required in this exercise. We have three input pins being respectively the enable "en_sig", the reset "cnt_rst" and the clock "sys_clk". The clock is fed to a block symbol called clock divider which takes as input the 50 MHz clock provided by our board and returns a 2Hz through Verilog code which will be the clock of our counter clock so that we can see the LED's blink each 0.5 seconds being visible to the human eye. The binary value of the counter is connected to Red LED's which display from most significant to least significant the value of the counter through $o_3o_2o_1o_o$. Also, the same value is being passed to a BCD-7-segment display Decoder which displays the value in hexadecimal on the board. Finally, when the value of the counter reaches F in hexadecimal or 1111 in binary, there will be an overflow in the next increment. To indicate this increment, O_{flow} was connected to a Green Light which blinks when the counter reaches 1111 or F.

Simulation

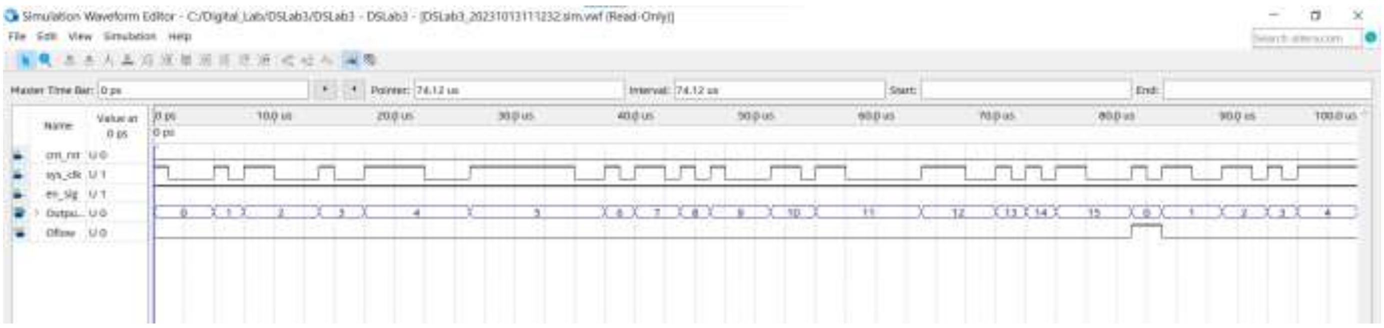


Figure 4- Waveform simulation of the counter

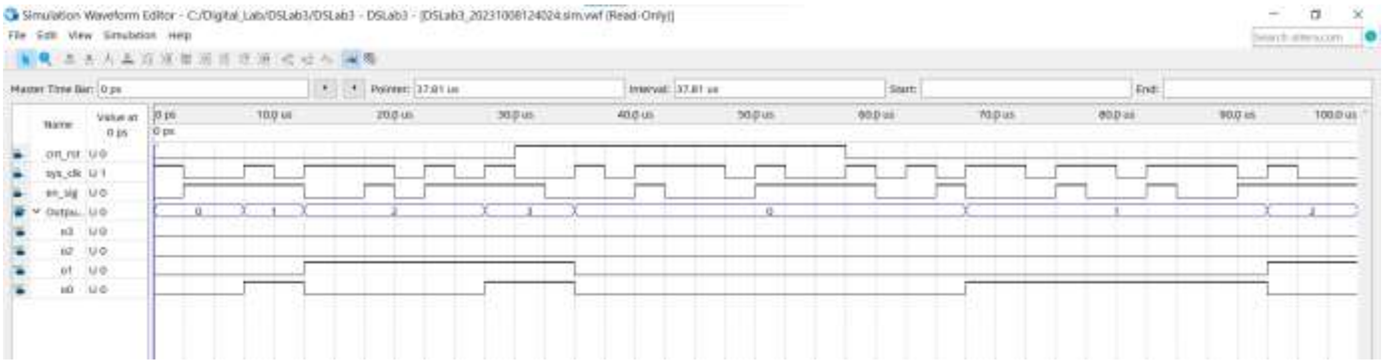


Figure 5- Waveform simulation of the counter-2

Figure 4 and 5 show different scenarios in our counter.

In this waveform named "Waveform1", we were able to confirm that our counter is giving us correct values. In fact, when the clock is not on a rising edge, nothing happens. When the clock is on a rising edge, it checks the value of the enable. If it is 0, it holds its value. If its 1, it increments the counter. Moreover, if on a rising edge the reset is 1 such as in Figure 5 it resets the counter's value to 0. Finally, when the counter reaches 15, O_{flow} is ON the next state (0) with a value of 1 indicating that there an overflow happened as shown in Figure 4.

Experiment 2

In this experiment, we're going to design a system that takes one input W, a reset and a clock. When 4 consecutive 0's or 1's are detected, the output F is turned on which will be seen by a red LED.

State Diagram

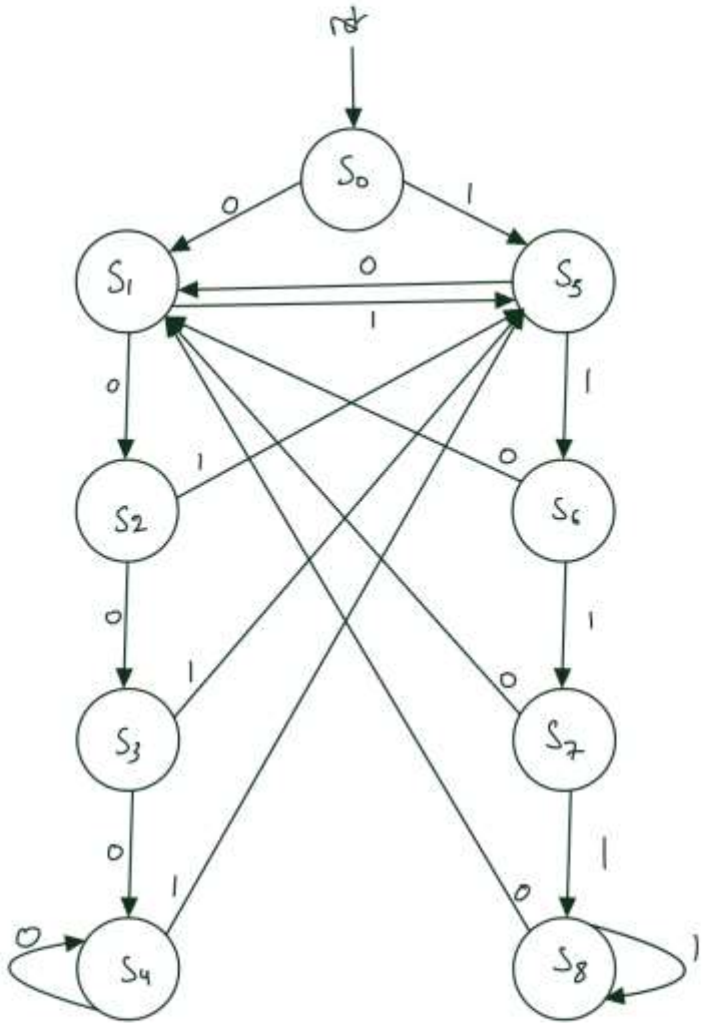


Figure 6- State Diagram of the Digital Lock

State Table

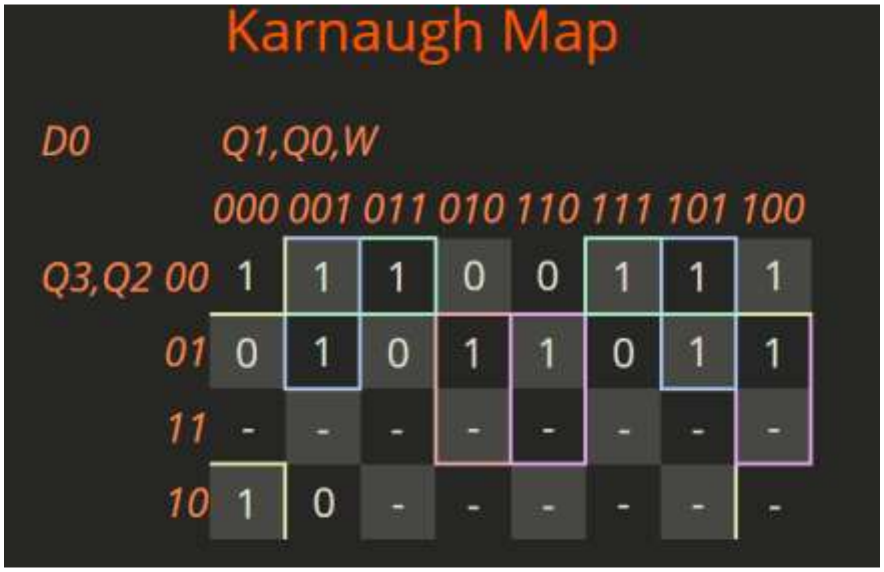
Table 6- State table for the Digital Lock

States	Present	State			Input	Next	State			Output
	Q3	Q2	Q1	Q0	W	D3	D2	D1	D0	F
S_0	0	0	0	0	0	0	0	0	1	0
					1	0	1	0	1	0
S_1	0	0	0	1	0	0	0	1	0	0
					1	0	1	0	1	0
S_2	0	0	1	0	0	0	0	1	1	0
					1	0	1	0	1	0
S_3	0	0	1	1	0	0	1	0	0	0
					1	0	1	0	1	0
S_4	0	1	0	0	0	0	1	0	0	1
					1	0	1	0	1	1
S_5	0	1	0	1	0	0	0	0	1	0
					1	0	1	1	0	0
S_6	0	1	1	0	0	0	0	0	1	0
					1	0	1	1	1	0
S_7	0	1	1	1	0	0	0	0	1	0
					1	1	0	0	0	0
S_8	1	0	0	0	0	0	0	0	1	1
					1	1	0	0	0	1

K-map / State equations

Note that all the K-maps were built using online K-map simulators since we never dealt with 5 input K-maps

Table 7- K-map of output D0



Karnaugh Map

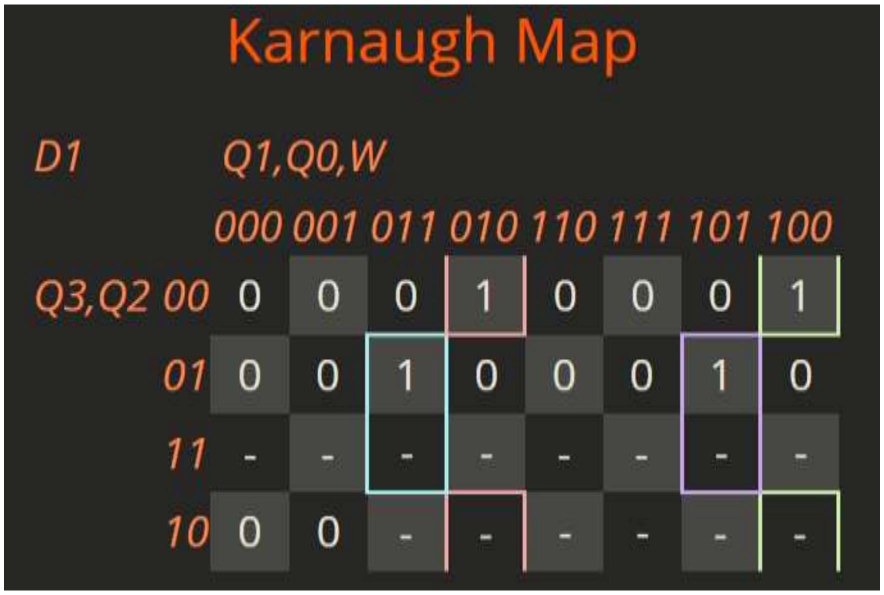
D0 Q1,Q0,W

Q3,Q2	000	001	011	010	110	111	101	100
00	1	1	1	0	0	1	1	1
01	0	1	0	1	1	0	1	1
11	-	-	-	-	-	-	-	-
10	1	0	-	-	-	-	-	-

According to the online simulator, we get:

$$D0 = Q2Q0W' + Q2'Q0'W' + Q3'Q2'W + Q3'Q0'W + Q2Q1W'$$

Table 8- K-map of output D1



Karnaugh Map

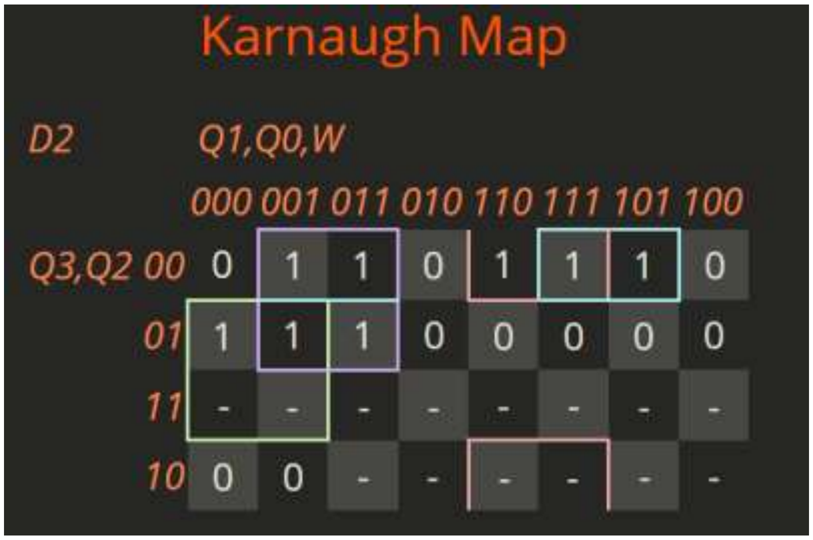
D1 Q1,Q0,W

Q3,Q2	000	001	011	010	110	111	101	100
00	0	0	0	1	0	0	0	1
01	0	0	1	0	0	0	1	0
11	-	-	-	-	-	-	-	-
10	0	0	-	-	-	-	-	-

According to the online simulator, we get:

$$\boldsymbol{D1 = Q2'Q1'Q0W' + Q2'Q1Q0'W' + Q2Q1'Q0W + Q2Q1Q0'W}$$

Table 9- K-map of output D2



Karnaugh Map

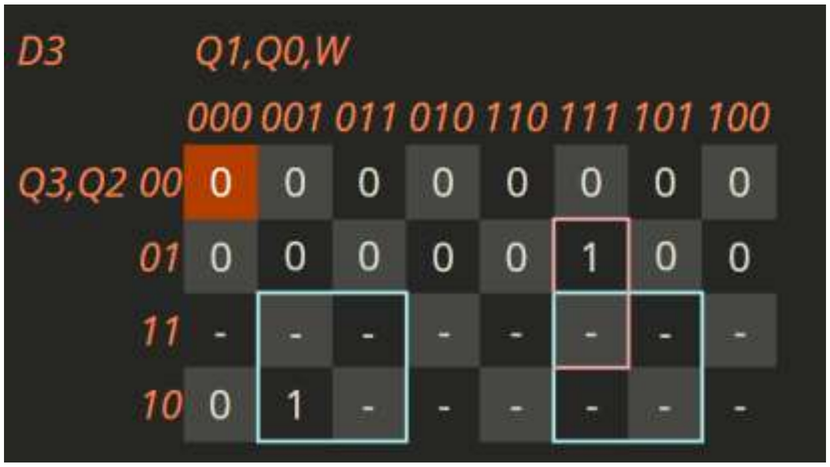
Q1,Q0,W

D2	000	001	011	010	110	111	101	100
Q3,Q2 00	0	1	1	0	1	1	1	0
01	1	1	1	0	0	0	0	0
11	-	-	-	-	-	-	-	-
10	0	0	-	-	-	-	-	-

According to the online simulator, we get:

$$D2 = Q2'Q1Q0 + Q2Q1'Q0' + Q3'Q1'W + Q3'Q0'W$$

Table 10- K-map of output D3



Q1,Q0,W

D3	000	001	011	010	110	111	101	100
Q3,Q2 00	0	0	0	0	0	0	0	0
01	0	0	0	0	0	1	0	0
11	-	-	-	-	-	-	-	-
10	0	1	-	-	-	-	-	-

$$D3 = Q2Q1Q0W + Q3W$$

Note that we need to AND all the outputs with rst’ to take into consideration the reset as done in experiment 1.

Block diagram

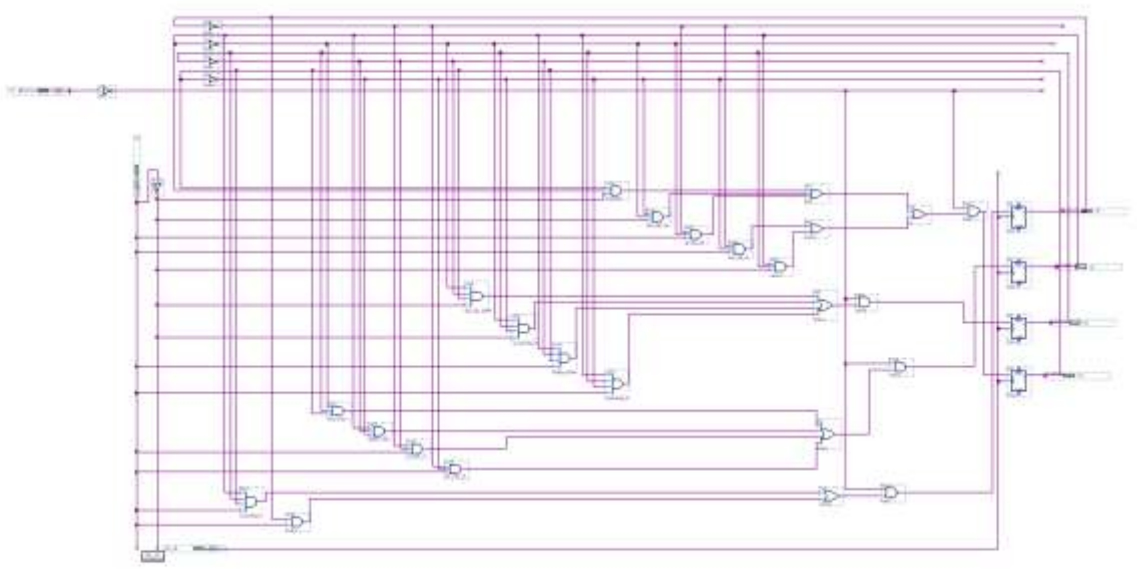


Figure 7-Block Diagram of the Digital Lock

Figure 7 represents the block diagram of the digital lock circuit. We can see all the inputs with the flip flops and logical gates that were related using the state equations obtained in the previous part. If not visible, please refer to the block diagram named “digitallock” in the files submitted with this report.

We will include a simulation in the simulation section to test our digital lock outputs.

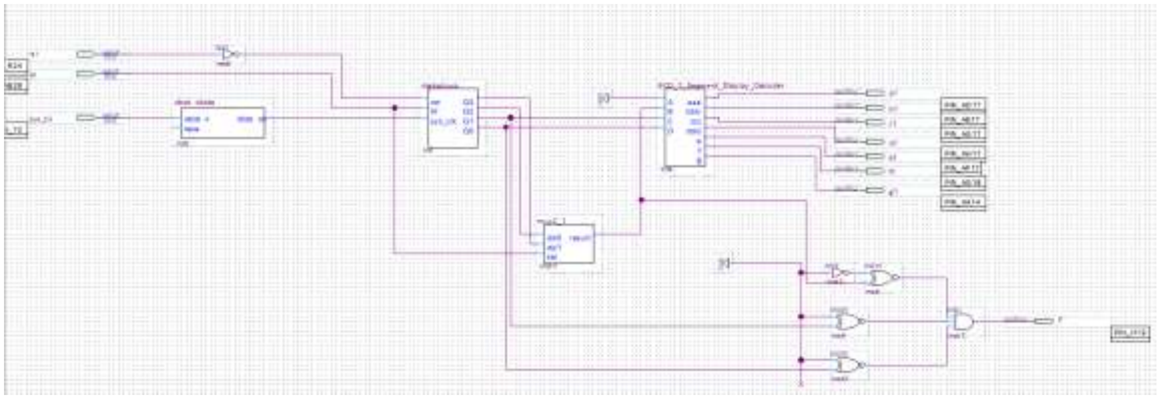


Figure 8- Block Diagram of the full digital lock circuit

Figure 8 shows the block diagram of the full digital circuit. We used a multiplexer which passes either Q_3 if W=1 or Q_2 if W=0. This way we can make sure that state 5 to 8 are represented as 1,2,3 and 4 respectively on the BCD. Moreover, to check for F, we XNORed Q_3/Q_2 Q_1Q_0 with100 which represents the last state either when counting 0’s or counting 1’s. Finally, we had to implement our own MUX2_1 as shown above.

Note that we had to use an inverter on the reset rst since the push buttons work on active low.

Simulation

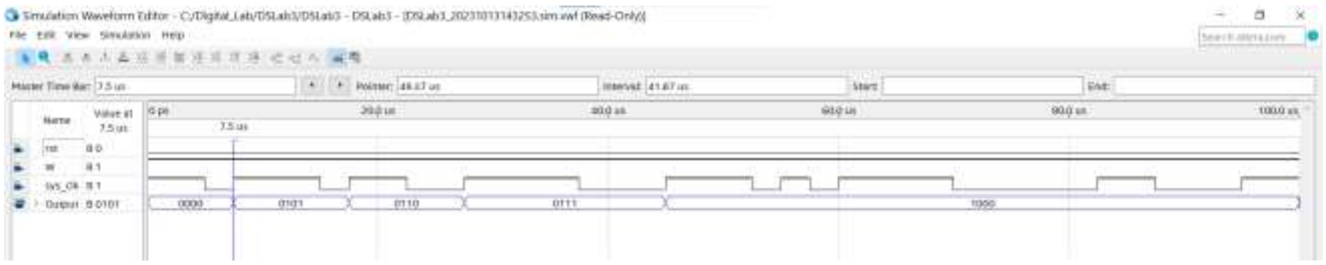


Figure 9- Simulation of the Digital Lock -1

Figure 9 shows when the reset is 0, the counter counts the consecutive (1) given at the input. It's starting at state 5 that represents the first (1) and so on till state 8. This numbering will be changed onwards so that the BCD displays the values from 1 to 4 and not 5 to 8. At each rising edge, the counter increments its value only when there are consecutive (1), so here since we have consecutive (1) the counter will keep on counting till it gets to 1000 which should be the 4th state, if there are more than 4 consecutive (1), the output will hold its state at 4 represented by 1000.

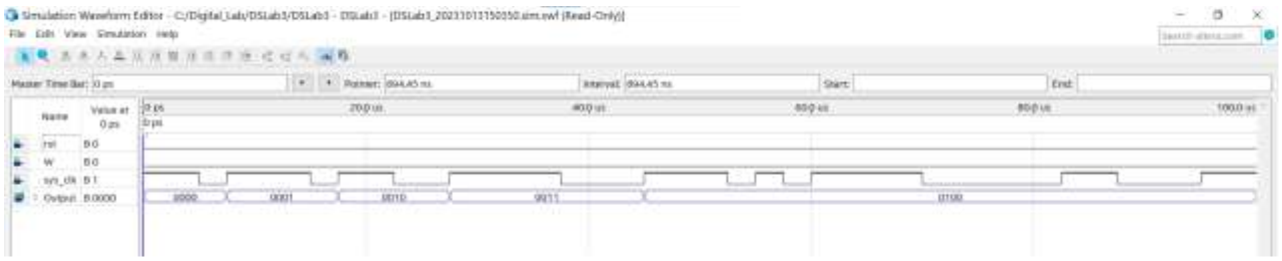


Figure 10- Simulation of the Digital Lock -2

Figure 10 shows when the reset is 0, the counter counts the consecutive (0) given at the input. At each rising edge, the counter increments its value only when there are consecutive (0), so here since we have consecutive (0) the counter will keep on counting till it gets to 0100 which should be the 4th state, if there are more than 4 consecutive (0), the output will hold its state at 4 represented by 0100.

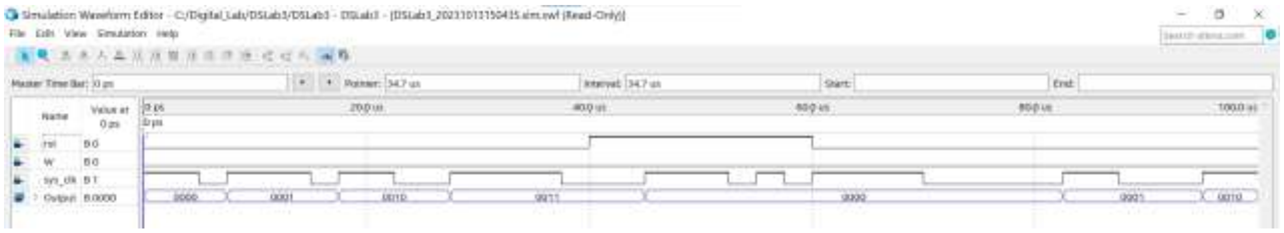


Figure 11- Simulation of the Digital Lock -3

Figure 11 shows that when the reset is 1, the counter will reset to zero no matter the previous value.

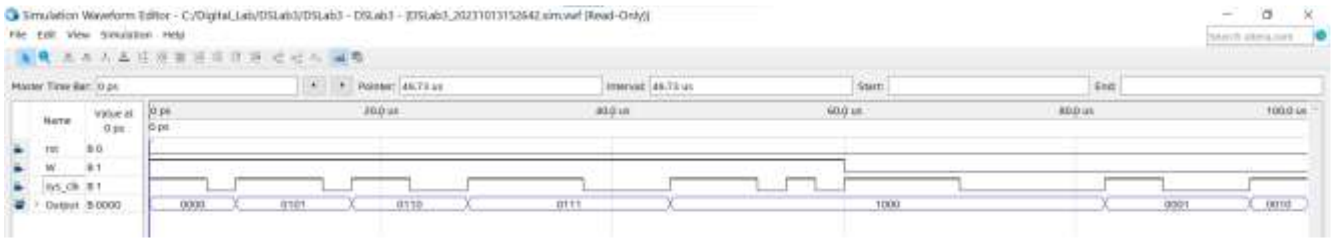


Figure 12- Simulation of the Digital Lock -4

Figure 12 shows that for any consecutive (1), when a (0) comes next as an input, the counter goes from the current state to State 1.

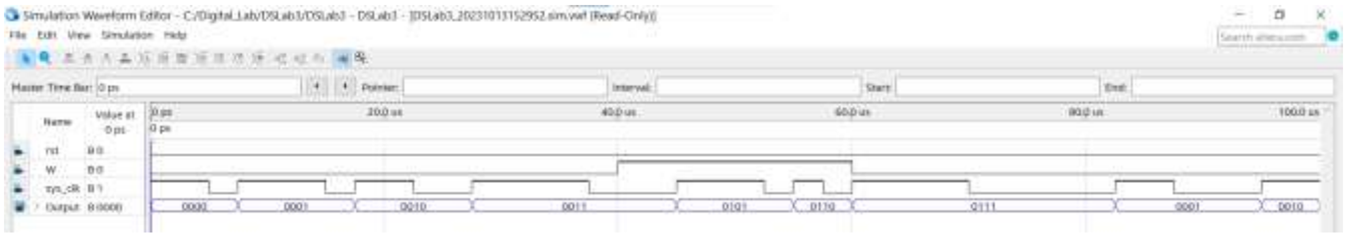


Figure 13- Simulation of the Digital Lock -5

Figure 13 shows that for any consecutive (0), when a (1) comes next as an input, the counter goes from the current state to State 5.

Note that Figures 11,12,13 and 14 are all the waveform “Waveform2” but simulated for different values to cover the test cases.

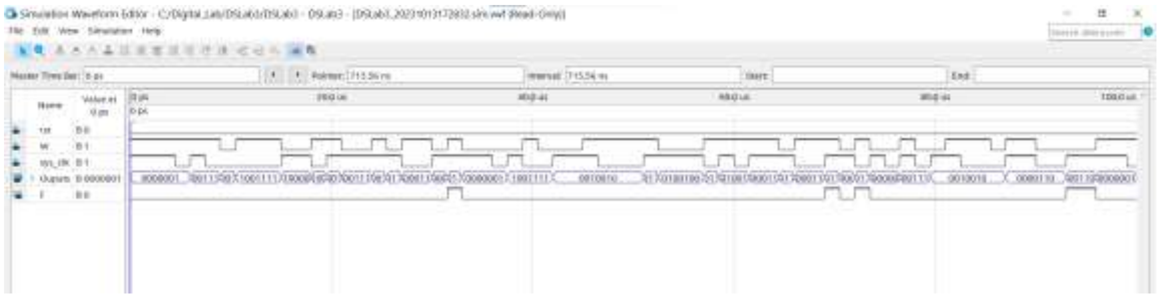


Figure 14- Simulation of Waveform 4

Figure 14 shows a bigger representation of the final circuit obtained for multiple values. Please refer to waveform 4 to see it in details since it is impossible for us to cover the cases.

Conclusion

We created a 4-bit counter using flip-flops and computational logic with the help of state diagram, state table, Kmaps and successfully simulated it on Quartus and tried it on the FPGA which gave us a correct view of our design. In the second part, we created a Digital Lock using the same steps and successfully tried it on the FPGA.

References

https://www.charlie-coleman.com/experiments/kmap/