DIGITAL SYSTEM DESIGN APPLICATIONS

Project-2

SEQUENTIAL CIRCUITS

GROUP-2

Elif ÇATIKKAŞ 040190094

Berfin DUMAN 040190108

1000/0000 Detector

Since the penultimate digit of the student number is 0, we gave the value 0000 to A, and the value 1000 to B because the last digit is 8. Thus, the values we needed to detect within the scope of the project were determined.

A → 0000

B → 1000

Firstly, to better observe our states, we performed state encoding, preventing any potential confusion in the later stages of the project. During state encoding, we considered each updated step as a state to be detected in the numbers. Using a more compact encoding for states can reduce the number of flip-flops required to implement the FSM. For example, using binary encoding would require more flip-flops than using one-hot encoding for the same number of states.

1000 → 1-10-100-1000

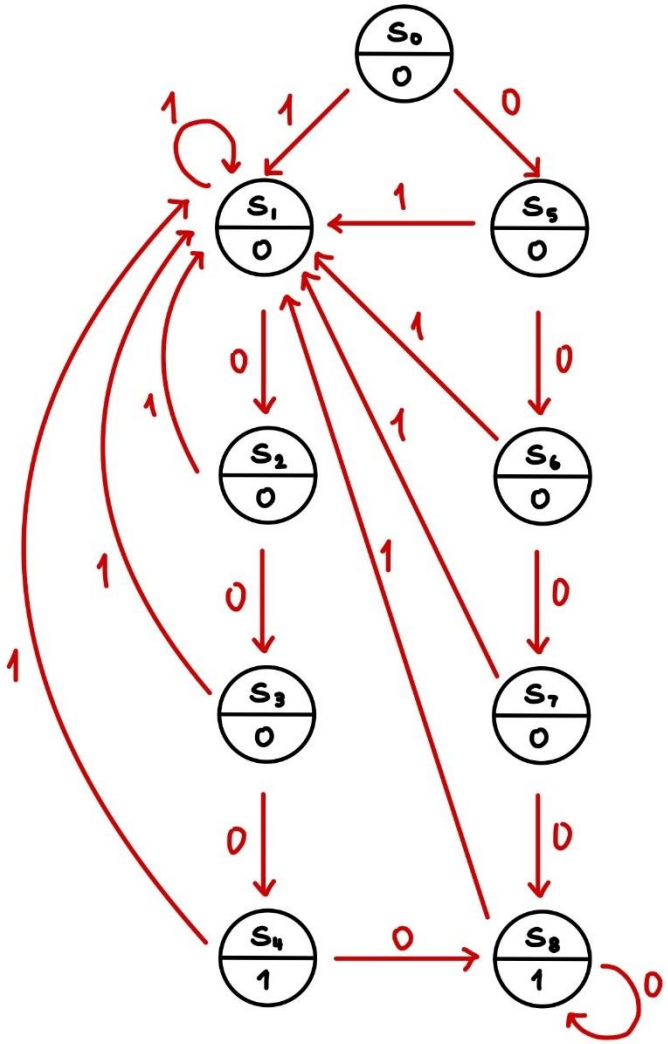
0000 → 0-00-000-0000

s0	0000
s1	0001
s2	0010
s3	0011
s4	0100
s5	0101
s6	0110
s7	0111
s8	1000

State	Next State , Output	
	x = 0	x = 1
s0	s5,0	s1,0
s1	s2	s1,0
s2	s3	s1,0
s3	s4,1	s1,0
s4	s8,1	s1,0
s5	s6,0	s1,0
s6	s7,0	s1,0
s7	s8,1	s1,0
s8	s8,1	s1,0

After this state encoding process, we created a State Table without any reduction. The State Table without any reduction in a project serves as a comprehensive reference that outlines all possible states and their corresponding transitions, without applying simplifications or optimizations. This unaltered representation provides a detailed and exhaustive overview of the system's behavior, allowing for a clear understanding of the interactions between states and transitions. It is particularly useful during the development and analysis phases, offering a complete and unambiguous depiction of the system's state space, which can aid in debugging, testing, and ensuring the correctness of the implemented logic.

State Diagram



Finite State Machine (FSM) and its visual representation, known as state diagrams, are utilized to model specific states of a system or program and the transitions between these states. Designing a state diagram serves various purposes, including understanding, analyzing, and communicating system design, debugging, facilitating communication, tracking and managing states, as well as expediting code generation. This visual representation simplifies the comprehension of complex systems, fostering collaboration within project teams and elucidating the system's behavior. State diagrams offer a clear visual summary of transitions between states in a system, thereby easing the overall system development process. For these reasons, we have designed a state diagram for FSM to illustrate the states and transitions that lead to the formation of patterns such as 1000 and 0000.

Moore machines, having no signal changes during transitions from one state to another, are less prone to input/output (I/O) errors, which can enhance system stability. For these reasons, Moore machines are preferred, especially when the emphasis is on the

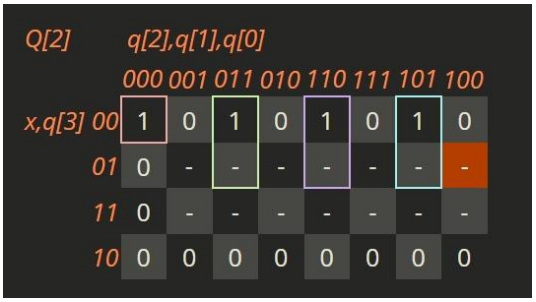
comprehensibility and reliability of system design. For this reason, as mentioned in the project, **we used a moore machine to avoid hazardous problems**.

We created the State Table after Encoding, taking into account the don't care conditions without neglecting them, in a manner suitable for the state diagram. This allowed us to derive functions for the appropriate states that the flip-flops will hold, using the Q4 - Q3 - Q2 - Q1 - y minterms.

mt	x	q4	q3	q2	q1	Q4	Q3	Q2	Q1	y
0	0	0	0	0	0	0	1	0	1	0
1	0	0	0	0	1	0	0	1	0	0
2	0	0	0	1	0	0	0	1	1	0
3	0	0	0	1	1	0	1	0	0	1
4	0	0	1	0	0	1	0	0	0	1
5	0	0	1	0	1	0	1	1	0	0
6	0	0	1	1	0	0	1	1	1	0
7	0	0	1	1	1	1	0	0	0	1
8	0	1	0	0	0	1	0	0	0	1
9	0	1	0	0	1	x	x	x	x	x
10	0	1	0	1	0	x	x	x	x	x
11	0	1	0	1	1	x	x	x	x	x
12	0	1	1	0	0	x	x	x	x	x
13	0	1	1	0	1	x	x	x	x	x
14	0	1	1	1	0	x	x	x	x	x
15	0	1	1	1	1	x	x	x	x	x
16	1	0	0	0	0	0	0	0	1	0
17	1	0	0	0	1	0	0	0	1	0
18	1	0	0	1	0	0	0	0	1	0
19	1	0	0	1	1	0	0	0	1	0
20	1	0	1	0	0	0	0	0	1	0
21	1	0	1	0	1	0	0	0	1	0
22	1	0	1	1	0	0	0	0	1	0
23	1	0	1	1	1	0	0	0	1	0
24	1	1	0	0	0	0	0	0	1	0
25	1	1	0	0	1	x	x	x	x	x
26	1	1	0	1	0	x	x	x	x	x
27	1	1	0	1	1	x	x	x	x	x
28	1	1	1	0	0	x	x	x	x	x
29	1	1	1	0	1	x	x	x	x	x
30	1	1	1	1	0	x	x	x	x	x
31	1	1	1	1	1	x	x	x	x	x

Karnaugh Maps

Q[3] = x'q[2]q[1]'q[0]' + x'q[2]q[1]q[0] + x'q[3]



Q[2] = x'q[3]'q[2]'q[1]'q[0]' + x'q[2]'q[1]q[0] + x'q[2]q[1]'q[0] + x'q[2]q[1]q[0]'

Q[1]= x'q[1]'q[0] + x'q[1]q[0]'

Q[0] = q[3]'q[2]'q[0]' + q[1]q[0]' + x

Y = x'q[1]q[0] + x'q[2]q[1]'q[0]' + x'q[3]

(-) means don't care

Verilog Code

```
module Detect_Machine(
input x,
input clk,
input rst,
output reg y);
reg [3:0] q;
wire [3:0] Q;
wire Y;
reg [1:0] lock_a, lock_b;
initial
begin
    q= 4'b0; lock_a = 0; lock_b= 0;
end
assign Q[3] = (~x & q[2] & ~q[1] & ~q[0]) | (~x & q[2] & q[1] & q[0]) | (~x
& q[3]);
assign Q[2] = (~x & ~q[3] & ~q[2] & ~q[1] & ~q[0]) | (~x & ~q[2] & q[1] &
q[0]) | (~x & q[2] & ~q[1] & q[0]) | (~x & q[2] & q[1] & ~q[0]);
assign Q[1] = (~x & ~q[1] & q[0]) | (~x & q[1] & ~q[0]);
assign Q[0] = (~q[3] & ~q[2] & ~q[0]) | (q[1] & ~q[0]) | (x);
assign Y = (~x & q[1] & q[0]) | (~x & q[2] & ~q[1] & ~q[0]) | (~x & q[3]);

always @(posedge clk or posedge rst)
 begin
    if (rst)
        begin
        lock_a=0;
        lock_b=0;
        q= 4'b0;
        end
    else
    begin
        if (lock_a ==2 || lock_b ==2)
            begin
                y=1'b1;
            end
        else
            begin
             q[3] <= Q[3];
             q[2] <= Q[2];
             q[1] <= Q[1];
             q[0] <= Q[0];
             y <= Y;
```

Detect_Machine module -Part 1

```
            if (Q==4'b0011 && x==0)
             begin
                lock_a=lock_a+1;
             end
            if (Q==4'b0100 && x==0)
             begin
                lock_b=lock_b+1;
             end
            if (Q==4'b0111 && x==0)
             begin
                lock_b=lock_b+1;
             end
            if (Q==4'b1000 && x==0)
             begin
                lock_b=lock_b+1;
             end
         end
     end
 end
endmodule
```

Detect_Machine module - Part 2

The main objective of the project is to detect the patterns 0000 and 1000 on the input signal x and subsequently create a lock state if the same pattern is detected for the second time. To achieve this, we defined the functions of flip-flops and the output value by calculating these values under certain conditions, and we updated the Finite State Machine (FSM) states.

To detect the patterns 0000 and 1000, we calculated the Q and Y signals for specific states and input values (x). These calculations involve setting Q and Y to specific values when the input signal conforms to a certain pattern.

For the lock state, two counters were used: lock_a and lock_b. When the pattern A (4'b0000) is detected, lock_a increments by one. Similarly, when pattern B (4'b1000) is detected, lock_b increments by one. If either lock_a or lock_b reaches 2, the system becomes locked, and the output (y) is always set to "1". When a reset operation (rst) occurs while the system is in the locked state, the system returns to the initial state, and all values are reset.

Thus, when the desired patterns are detected and the same pattern is detected for the second time, the system becomes locked, and the output takes the value "1". The reset operation while the system is in the locked state returns the system to the initial state, restarting the system and continuing the detection process.

Testbench Code

```
`timescale 1ns / 1ps
module detect_machine_tb();
 wire z;
 reg x; reg clk, rst;

reg [15:0] test = 16'b0100011000100001; reg flag= 1; //2A 8
//reg [15:0] test = 16'b1000001110000101; reg flag= 1; //2B 0
//reg [15:0] test = 16'b0010001000011001; reg flag= 1; //overlap
//reg [15:0] test = 16'b0010100000110100; reg flag= 0; //iki tane
//reg [15:0] test1= 16'b0101000100001010;//3 kitleme

detect_machine uut(x, clk,rst, z);
initial
    begin
        clk = 1;
        rst=1 ; #10 rst=0;
        x=test[15];
        #5;
 end
always #10 clk = !clk;

always @(posedge clk)
 begin


 test = test << 1;
x = test[15];

 if (test == 15'b0 ) begin
    if (!flag)
    begin
    rst= 1; #1
    rst=0;
    flag=1;
    test=test1;
    end
    else
    $finish;

 end
 end
always @(negedge clk)
    begin
        $display("%b,%b",x,z);
    end
endmodule
```

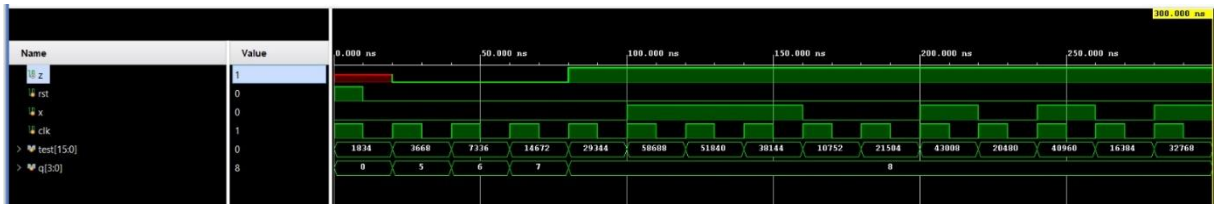
Behavioral Simulation

We created a simulation for each different situation to perform a detailed test. These situations:

Case 1: The situation where two B values come and then the circuit is locked.

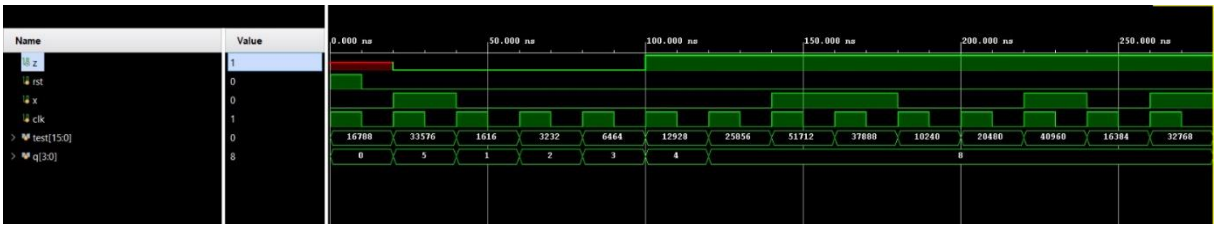
Our input value is 0100011000100001. As seen here, two B values appear. Therefore, after these two situations, the system is expected to crash. As we expected, crashes were observed.

Case 2: The situation where two A values come and then the circuit is locked.



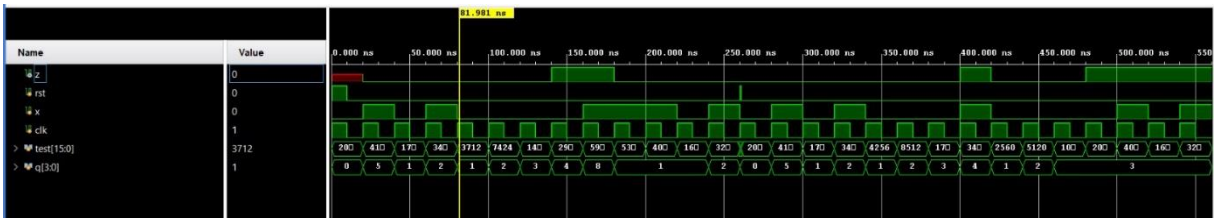
Our input value is 1000001110000101. As seen here, two B values appear. Therefore, after these two situations, the system is expected to crash. As we expected, crashes were observed.

Case 3: The situation where A and B values overlap.



Our input value is 0010001000011001. As can be seen here, the first B value is observed and then the A value is observed after the successive 0 and it creates an overlap as we expected. Since there are no situations where B is repeated in A due to the values, the opposite of the current overlap cannot be observed.

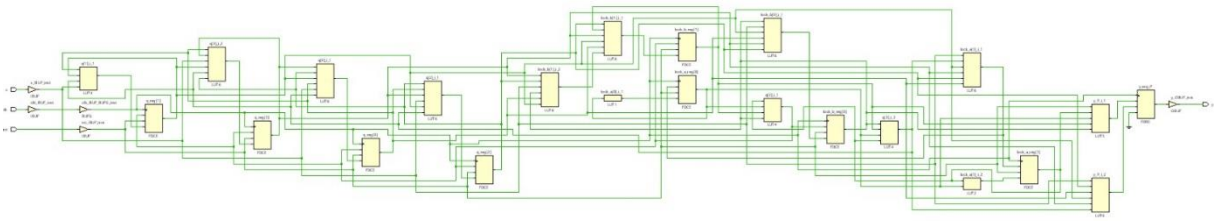
Case 4: The state where the lock state is reset after rst and the circuit is locked after two A or B values.



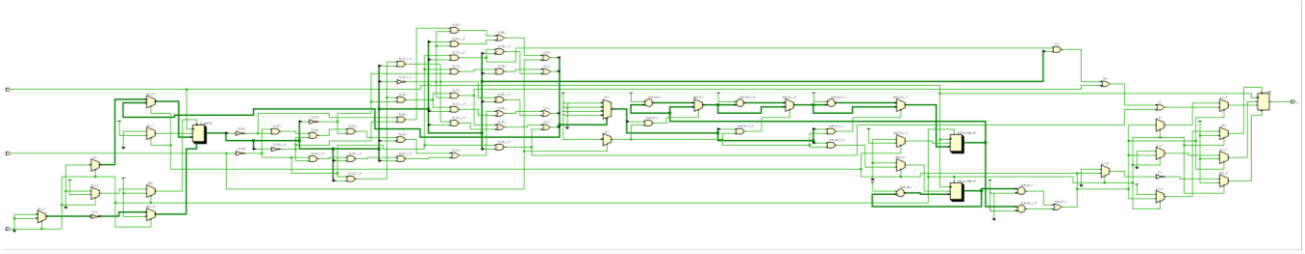
0010100000110100 This entry contains both patterns (A and B), in this case, if A or B is observed again, the system must be locked, but we applied a reset process in accordance with the scenario we wanted to observe and tried to create a scenario where we set the lock status to

0. To observe this, after resetting, we showed that the locking process occurred after two A values were observed as a result of the entry here 0101000100001010. So, in this case, it gave the result we wanted.

Technology Schematic



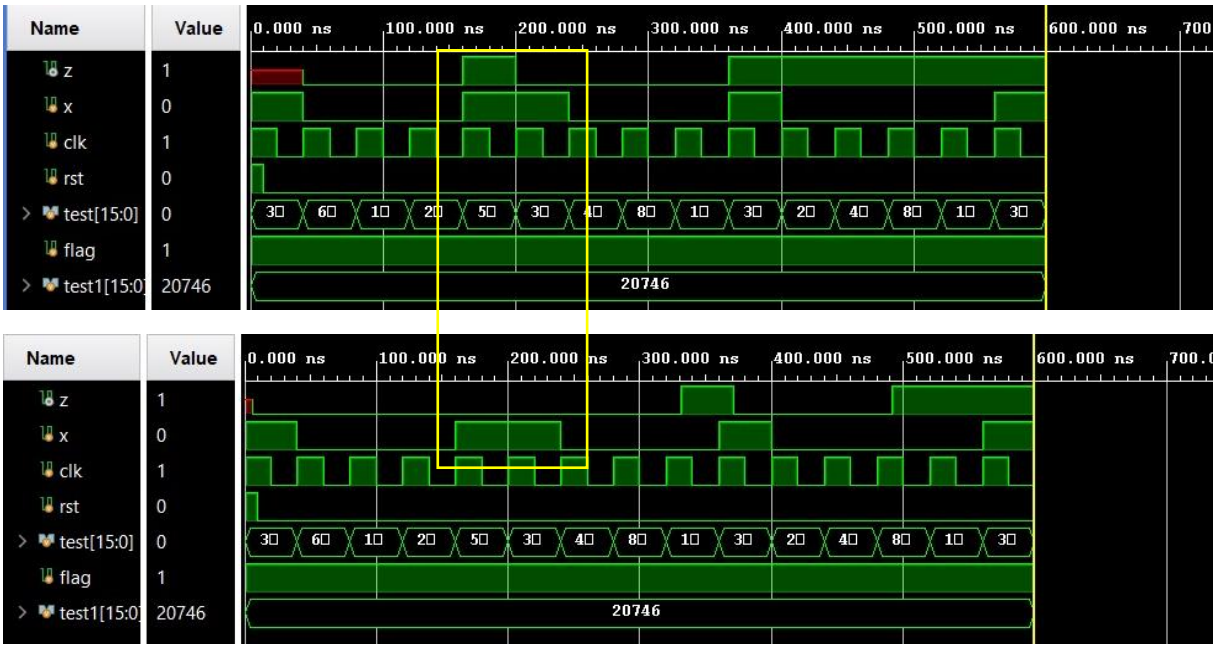
RTL Schematic



Implementation

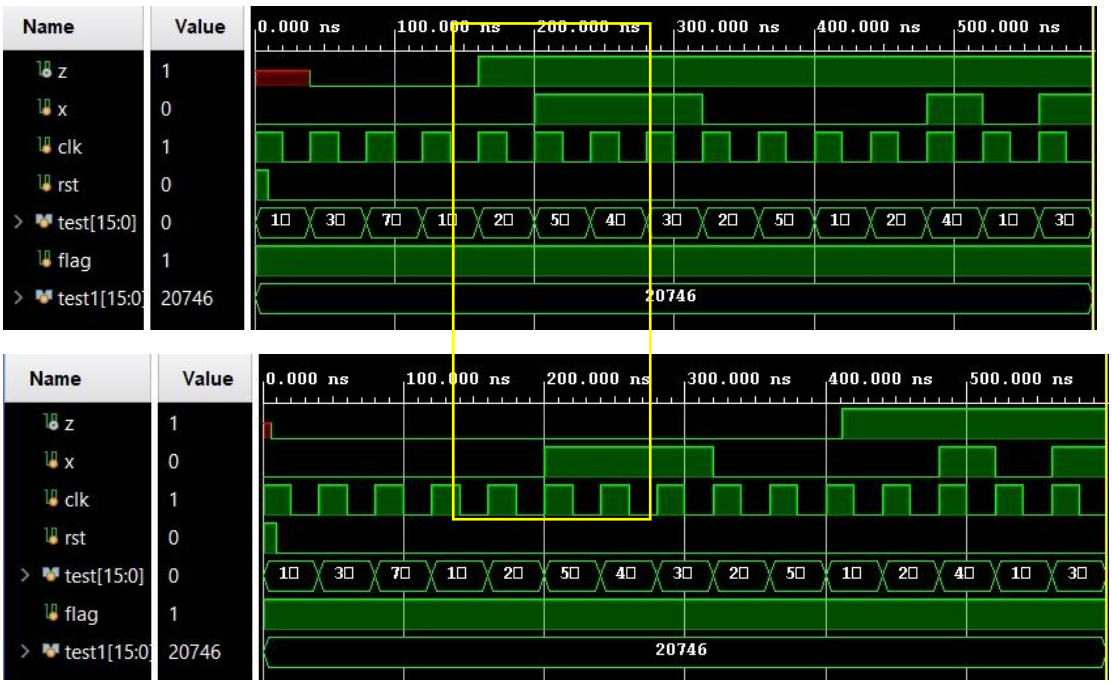
Post- Implementation Simulation

Case 1: The situation where two B values come and then the circuit is locked. (cannot detect the first pattern)



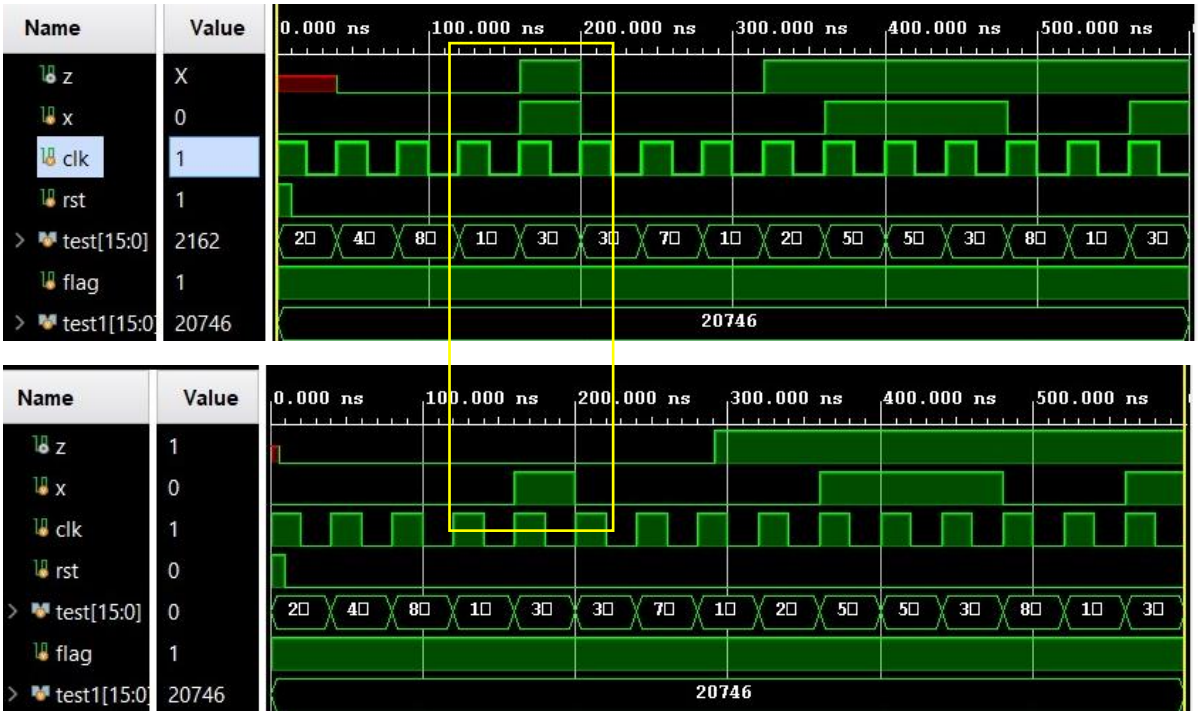
Our input value is 0100011000100001. As seen here, three B values appear. As we observed in the behavioral simulation, the system should have locked when 2 patterns were detected, but in the post-implementation period, the first pattern cannot be detected, so we observe a deadlock in the third pattern.

Case 2: The situation where two A values come and then the circuit is locked. (cannot detect the first pattern)

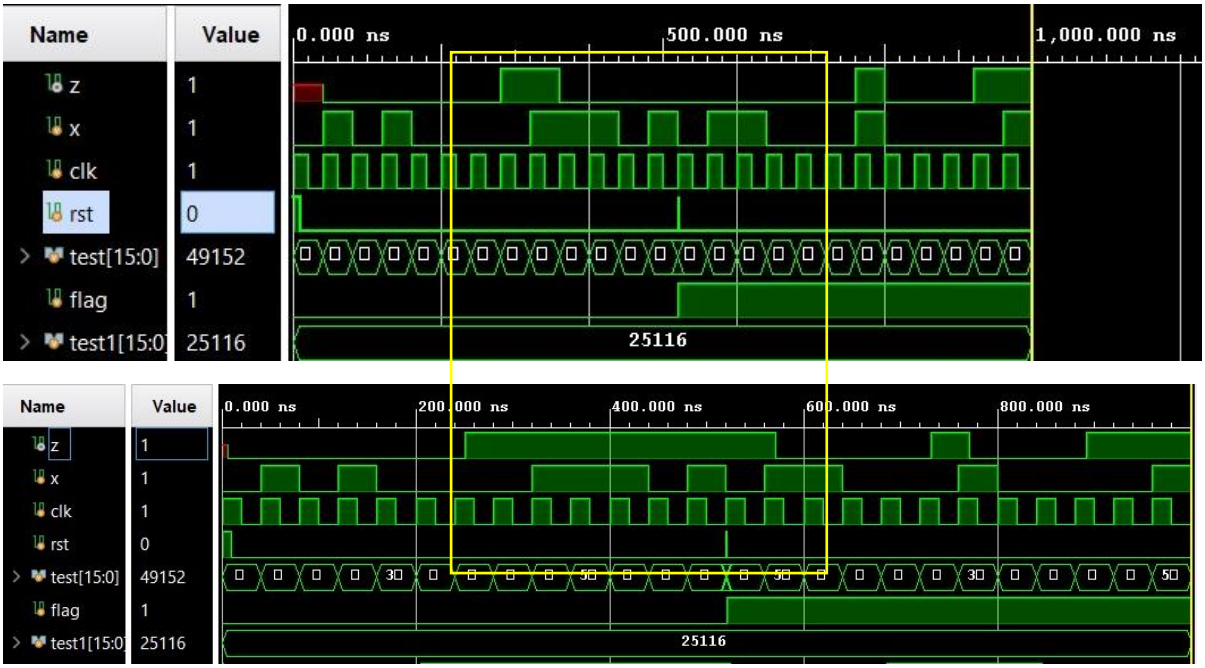


Our input value is 1000001110000101. As seen here, three B values appear. As we observed in the behavioral simulation, the system should have locked when 2 patterns were detected, but in the post-implementation period, the first pattern cannot be detected, so we observe a deadlock in the third pattern.

Case 3: The situation where A and B values overlap. (cannot detect the first pattern)



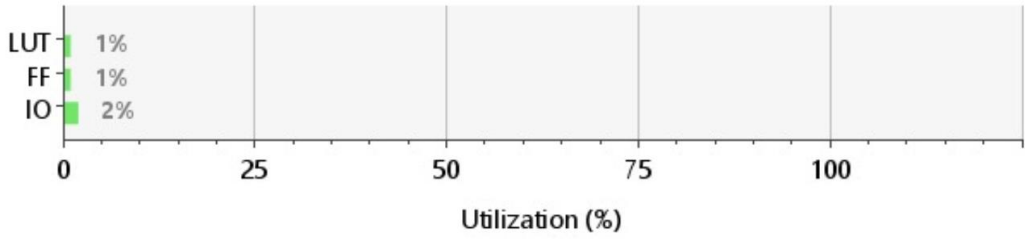
Case 4: The state where the lock state is reset after rst and the circuit is locked after two A or B values. (cannot detect the first pattern)



Utilization Report

Summary

Resource	Utilization	Available	Utilization %
LUT	12	63400	0.02
FF	9	126800	0.01
IO	4	210	1.90



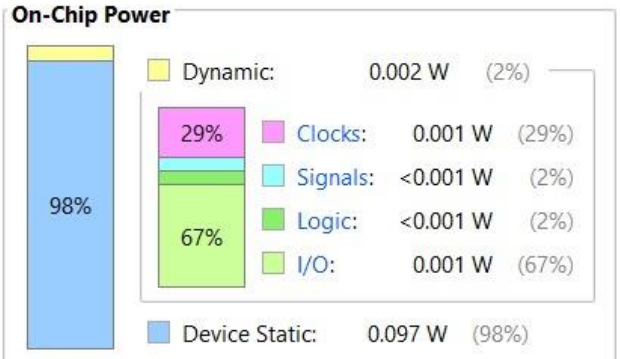
Power Consumption

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.099 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (12.9 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



Timing Report

Set-up Delay

Unconstrained Paths - NONE - sys_clk_pin - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 21	∞	2	2	9	rst	y_reg_P/CE	4.261	1.604	2.657	∞	input port clock	sys_clk_pin
Path 22	∞	1	1	9	rst	lock_a_reg[0]/CLR	3.346	1.480	1.867	∞	input port clock	sys_clk_pin
Path 23	∞	1	1	9	rst	lock_a_reg[1]/CLR	3.346	1.480	1.867	∞	input port clock	sys_clk_pin
Path 24	∞	1	1	9	rst	lock_b_reg[0]/CLR	3.342	1.480	1.862	∞	input port clock	sys_clk_pin
Path 25	∞	1	1	9	rst	lock_b_reg[1]/CLR	3.342	1.480	1.862	∞	input port clock	sys_clk_pin
Path 26	∞	1	1	9	rst	q_reg[0]/CLR	3.187	1.480	1.707	∞	input port clock	sys_clk_pin
Path 27	∞	1	1	9	rst	q_reg[1]/CLR	3.187	1.480	1.707	∞	input port clock	sys_clk_pin
Path 28	∞	1	1	9	rst	q_reg[2]/CLR	3.187	1.480	1.707	∞	input port clock	sys_clk_pin
Path 29	∞	1	1	9	rst	q_reg[3]/CLR	3.187	1.480	1.707	∞	input port clock	sys_clk_pin
Path 30	∞	3	2	8	x	lock_b_reg[1]/D	3.146	1.726	1.421	∞	input port clock	sys_clk_pin

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 41	∞	1	1	1	y_reg_P/C	y	6.080	3.976	2.103	∞	sys_clk_pin		

The max delay was observed in the path 21 and was 4.261 ns. max frequency is 1 / 4.261 ns = 234,686MHz

```
set_max_delay -from [get_ports {x rst clk}] -to [get_ports {y}] 4.000
```

No change in the situation was observed as a result of the constraint we applied to reduce the timing delay.

Hold Delay

Unconstrained Paths - NONE - sys_clk_pin - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 40	∞	1	1	9	rst	q_reg[0]/CLR	1.003	0.247	0.755	-∞	input port clock	sys_clk_pin
Path 39	∞	3	2	8	x	lock_b_reg[1]/D	0.883	0.335	0.548	-∞	input port clock	sys_clk_pin
Path 38	∞	2	1	8	x	q_reg[1]/D	0.877	0.290	0.587	-∞	input port clock	sys_clk_pin
Path 37	∞	2	1	8	x	q_reg[3]/D	0.876	0.290	0.586	-∞	input port clock	sys_clk_pin
Path 36	∞	2	1	8	x	y_reg_P/D	0.823	0.290	0.532	-∞	input port clock	sys_clk_pin
Path 34	∞	2	2	8	x	lock_a_reg[0]/CE	0.816	0.290	0.525	-∞	input port clock	sys_clk_pin
Path 35	∞	2	2	8	x	lock_a_reg[1]/CE	0.816	0.290	0.525	-∞	input port clock	sys_clk_pin
Path 33	∞	2	1	8	x	q_reg[0]/D	0.810	0.290	0.520	-∞	input port clock	sys_clk_pin
Path 32	∞	2	1	8	x	q_reg[2]/D	0.737	0.290	0.447	-∞	input port clock	sys_clk_pin
Path 31	∞	2	1	8	x	lock_b_reg[0]/D	0.650	0.290	0.360	-∞	input port clock	sys_clk_pin

Unconstrained Paths - sys_clk_pin - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 42	∞	1	1	1	y_reg_P/C	y	1.882	1.362	0.519	-∞	sys_clk_pin		

Work Package Table

	Berfin DUMAN	Elif ÇATIKKAŞ
Literature Review	x	x
Research	x	x
Preparation	x	x
Design/Verilog	x	x
Report	x	x