ISTANBUL TECHNICAL UNIVERSITY

Digital System Design Applications

Experiment VII
FINITE STATE MACHINES

2024

Objectives

- Design and implementation of finite state machines.
- Understanding the difference between Mealy and Moore Type machines.
- Learning state reduction and state encoding.

Requirements

Students are expected to be able to

- The basic concepts of sequential circuits
- Structural and Behavioral Design Methods with Verilog
- Usage of Vivado: synthesize, implementation, design reports

Experiment Report Checklist

1. **CIRCUIT THAT DETECTS FOUR CONSECUTIVE 1 OR 0**
   - Explanation of the **Finite State Machine Encoding** methods: Binary, Gray and One-Hot Encoding.
   - State Encoding for the state diagram.
   - Reduction results of the output functions.
   - Verilog code, Testbench code and Simulation results.
   - Faulty outputs and explanation of their causes.
   - How to add DFF to the z output (changing machine type).
   - Simulation results after changing the machine type.
   - Post-Implementation Timing Simulation.
   - Utilization and Timing Summaries.
   - Stucking in Arbitrary States or not.
   - Comparison of the Behavioral model with your design.
   - Utilization and Timing Summaries of Behavioral Model.
   - Comparison in terms of resource usage and path delays.
2. **DESIGN WITH DIVIDED STATE DIAGRAMS**
   - State Encoding for the state diagram.
   - Reduction results of the output functions.
   - Verilog code, Testbench code and Simulation results.
   - Post-Implementation Timing Simulation.
   - Utilization and Timing Summaries.
   - Verilog code, Testbench code and Simulation results of the Behavioral Model.
   - Post-Implementation Timing Simulation of the Behavioral Model.
   - Utilization and Timing Summaries of the Behavioral Model.
   - Comparison in terms of resource usage and path delays.

- **Projects and reports are to be done INDIVIDUALLY. High amounts of points will be deducted from similar works.**
- **Reports must be written in a proper manner. Divide your text to sections and sub-sections if needed, label your figures and connect your sections with proper explanations of your works. Reports filled with imprecisely placed tables and figures, with no verbal explanations in workflow, will not fare well.**
- **Check homeworks section in Ninova for submission dates.**

CIRCUIT THAT DETECTS FOUR CONSECUTIVE 1 OR 0

Definition of the Problem

You will design a circuit that has 1-bit x input and 1-bit z output. The circuit will sample the x inputs at the rising edge of the clock signal (clk). If four consecutive 1s or 0s come from x input, the output z will be 1. In all other cases, z output will be 0. The state machine will be a Mealy Type Machine. Therefore, the output of the circuit will be 1 simultaneously when four consecutive 1s or 0s come from the input. Remember that if the machine was Moore, the output will be 1 after one clock cycle. In Fig. 1, state diagram of this problem is given. Use the state table without reduction given in Fig. 2.

State Encoding & Logic Minimization

1. Do research about the **Finite State Machine Encoding** methods. Explain the advantages and disadvantages of Binary, Gray and One-Hot encoding.
2. Perform a state encoding for the state diagram. You can use the same encodings given in Fig. 3.
3. Perform reduction for the output functions of each DFFs (Q2, Q1, Q0) and z by Karnaugh Map method.

RTL Description & Implementation

1. Create a new project and write down a module which is named **FSM1.v**. This module should have 1-bit inputs **clk**, **rst**, **x** and 1-bit output **z**. **x** refers to input bit sequence and **z** is the output bit sequence.
2. Realize the combinational parts of the circuit with **assign** keywords. Implement the state transitions (DFF values) with an **always** block.
3. Create a **testbench** to simulate your design. The input sequence from left to right: **0100110001110000111110000001111110000000111111** should be used.
4. Determine if your circuit is producing faulty 0 or faulty 1 outputs. If any, explain the causes of these faulty outputs.
5. Add a DFF in front of the **z** output in order to change the machine type from **Mealy to Moore**. **z** output should be triggered at the rising edge of the clock signal.
6. Simulate your design again with the same testbench. Are faulty outputs still exist? Explain.
7. Make your pin configurations. Assign **clk** and **rst** inputs to the buttons, **x** input to a switch and **z** output to a LED.
8. **Synthesize** and **Implement** your design.
9. Perform **Post-Implementation Timing Simulation** and verify your design.
10. Obtain **Utilization** and **Timing** Summaries.
11. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

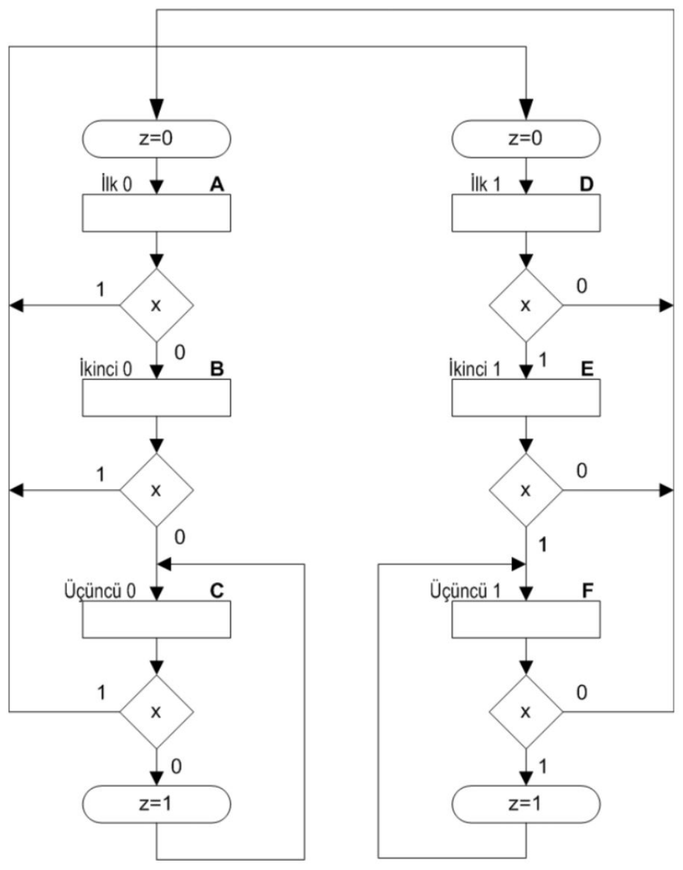


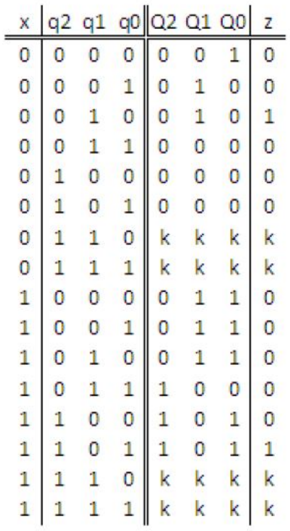
Figure 1: Algorithmic State Diagram

State	next state, output	
	x = 0	x = 1
A	B,0	D,0
B	C,0	D,0
C	C,1	D,0
D	A,0	E,0
E	A,0	F,0
F	A,0	F,1

Figure 2: State Table without any reduction

state	binary code
A	000
B	001
C	010
D	011
E	100
F	101

Figure 3: State Encoding Example



x	q2	q1	q0	Q2	Q1	Q0	z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	k	k	k	k
0	1	1	1	k	k	k	k
1	0	0	0	0	1	1	0
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	k	k	k	k
1	1	1	1	k	k	k	k

Figure 4: State Table after Encoding

Stucking in Undesirable States

1. Initialize the circuit from arbitrary states (i.e "110" and "111"). Use the same sequence to simulate the design.
2. Is the circuit switch its state to used states or is it stuck in arbitrary states? Explain.

Behavioral Model

1. Look at the Behavioral Model of the same problem which is uploaded to the Ninova **FSM1_behav.v** . Compare it with your previous design in terms of coding and design difficulty.
2. Create a new project and add the related files provided in Ninova.
3. **Synthesize** and **Implement** the design.
4. Perform **Post-Implementation Timing Simulation** and verify the design.
5. Obtain **Utilization** and **Timing** Summaries.
6. Make a comparison between the behavioral model and your previous design in terms of resource usage and path delays.

DESIGN WITH DIVIDED STATE DIAGRAMS

The divided state diagram in Fig. 5 has the same functionality with the state diagram in Fig. 1. It also detects four consecutive 1s or 0s with a different approach. You are expected to repeat the same steps as in the previous design **FSM1**.

1. Perform **State Encoding** & **Logic Minimization**.
2. Create a new project and write down a module which is named **FSM2.v**. This module should also have 1-bit inputs **clk**, **rst**, **x** and 1-bit output **z**. **x** refers to input bit sequence and **z** is the output bit sequence.

3. Realize the combinational parts of the circuit with **assign** keywords. Implement the state transitions (DFF values) with an **always** block. Use **Moore Type** machine. **z** output should be triggered at the rising edge of the clock signal.

4. Create a **testbench** to simulate your design. Use the same input sequence from left to right: **0100110001110000111100000111111000000111111**.

5. Make your pin configurations. Assign **clk** and **rst** inputs to the buttons, **x** input to a switch and **z** output to a LED.

6. **Synthesize** and **Implement** your design.

7. Perform **Post-Implementation Timing Simulation** and verify your design.

8. Obtain **Utilization** and **Timing** Summaries.

9. Generate **.BIT file** and program your FPGA. Show that your circuit works as expected.

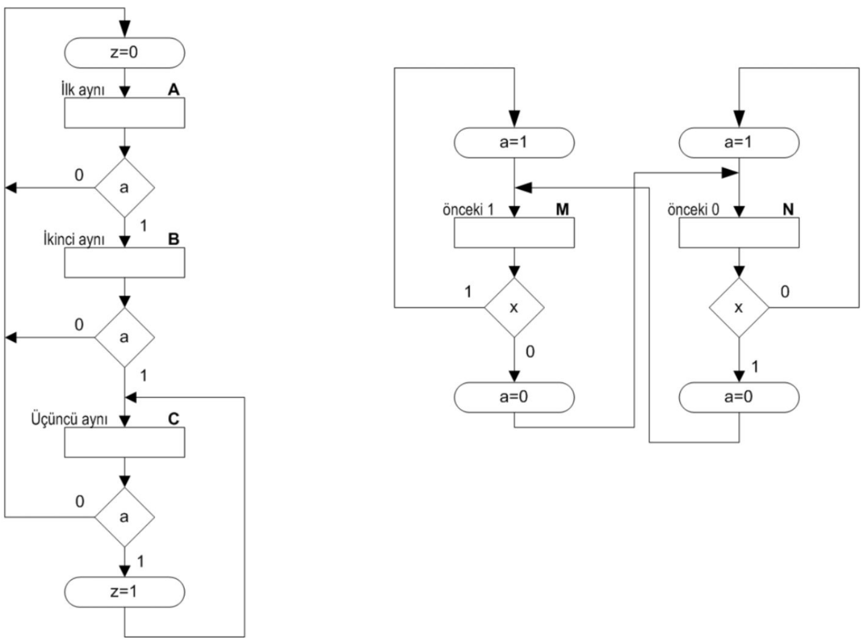


Figure 5: Divided Algorithmic State Table

Behavioral Model

1. Create a new project and write down a module which is named **FSM2_behav.v** by taken **FSM1_behav.v** as a reference.

2. Create a testbench to simulate your design. Use the same input sequence from left to right: **0100110001110000111100000111111000000111111**.

3. **Synthesize** and **Implement** your design.

4. Perform **Post-Implementation Timing Simulation** and verify your design.

5. Obtain **Utilization** and **Timing** Summaries.

6. Make a comparison between the behavioral model and your previous design in terms of resource usage and path delays.

References:

1. Nexys4DDR Reference Manual
2. Artix-7 Libraries Guide for HDL Designs
3. Constraints Guide
4. S. Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3rd ed.