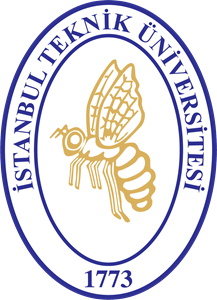
**DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF EXPERIMENT – 7**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

1. **Circuit That Detects Four Consecutive 1 or 0**

**Finite State Machine (FSM) Encoding Methodes:**

* **Binary Encoding:** In state machines, there could be a variety of states. In HDL coding these states are represented by an encoding number. Binary encoding method requires assigning these numbers one by one to minimize the length of the state vector, which is good for FPGA and PLA designs.

A table of numbers and values

Description automatically generated

Binary Encoded State Values Table1

* **Gray Encoding:** In gray encoding, only one-bit changes when moving between various states. As a result of this behavior, gray encoding consumes less power than binary coding. To explain furthermore, between each state jump, there are transistors on and off to represent 1s and 0s. This change in transistors causes the parasitic capacitors of the transistors to charge or discharge which requires power. With gray encoding, all the state jumps done by changing only one-bit that means the unchanged bits have not consumed any energy. Also, it can reduce glitches in the implementation.

A table of values

Description automatically generated

Gray Encoded State Values Table1

* **One-Hot Encoding:** One-Hot Encoding simplifies the circuits above via using one bit for each state. This simplifies the circuit and reduces propagation delays which results in making the circuit compatible with higher frequency clocks. However, the trade-off is that one-hot encoding increases the number of flip-flops used to store the state of the system.

A table of numbers and symbols

Description automatically generated

One-Hot Encoded State Values Table1

A table with numbers and symbols

Description automatically generated with medium confidence

Binary Encoding

A screenshot of a computer

Description automatically generated

Karnaugh Map of Q2

A screenshot of a computer

Description automatically generated

Karnaugh Map of Q1

A screenshot of a computer

Description automatically generated

Karnaugh Map of Q0

A screenshot of a computer

Description automatically generated

Karnaugh Map of Output z

**REDUCED EQUATIONS**

FSM Verilog Code

`timescale 1ns **/** 1ps

**module** FSM1**(**

**input** clk**,**

**input** rst**,**

**input** x**,**

**output** z

**);**

**reg** q0**,** q1**,** q2**;**

**assign** z **=** **(~**x **&** q1 **&** **~**q0**)** **|** **(**x **&** q2 **&** q0**);**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)** **begin**

q0 **<=** 1'b0**;**

q1 **<=** 1'b0**;**

q2 **<=** 1'b0**;**

**end** **else** **begin**

q2 **<=** **(**x **&** q1 **&** q0**)** **|** **(**x **&** q2**);**

q1 **<=** **(~**q2 **&** **~**q1 **&** q0**)** **|** **(**q1 **&** **~**q0**)** **|** **(**x **&** **~**q2 **&** **~**q1**);**

q0 **<=** **(~**q2 **&** **~**q1 **&** **~**q0**)** **|** **(**x **&** **~**q1**)** **|** **(**x **&** **~**q0**);**

**end**

**end**

**endmodule**

**!!! This is an addition page to report!!!**

After making FSM2, I fixed my code for FSM1 but after correcting my code there are not any faulty one or zero occurred in the simulation. Because of that I don’t know what to do so, I put the fixed code with simulation in this addition page. (I use the same testbench)

FSM1 Verilog Code (works correctly)

**module** FSM1**(**

**input** clk**,**

**input** rst**,**

**input** x**,**

**output** z

**);**

**reg** q0**,** q1**,** q2**;**

**wire** Q0**,** Q1**,** Q2**;**

**assign** z **=** **(~**x **&** q1 **&** **~**q0**)** **|** **(**x **&** q2 **&** q0**);**

**assign** Q2 **=** **(**x **&** q1 **&** q0**)** **|** **(**x **&** q2**);**

**assign** Q1 **=** **(~**q2 **&** **~**q1 **&** q0**)** **|** **(**q1 **&** **~**q0**)** **|** **(**x **&** **~**q2 **&** **~**q1**);**

**assign** Q0 **=** **(~**q2 **&** **~**q1 **&** **~**q0**)** **|** **(**x **&** **~**q1**)** **|** **(**x **&** **~**q0**);**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)** **begin**

q0 **<=** 1'b0**;**

q1 **<=** 1'b0**;**

q2 **<=** 1'b0**;**

**end** **else** **begin**

q2 **<=** Q2**;**

q1 **<=** Q1**;**

q0 **<=** Q0**;**

**end**

**end**

**endmodule**

A green and black background

Description automatically generated

FSM1 Behavioral Simulation (no faulty)

From now on the report continues as wanted order.

FSM Testbench Code

`timescale 1ns **/** 1ps

**module** FSM1\_tb**();**

**reg** clk **=** 0**;**

**reg** rst **=** 0**;**

**reg** x **=** 0**;**

**wire** z**;**

FSM1 dut**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**x**(**x**),**

**.**z**(**z**));**

**always** **#**5 clk **=** **~**clk**;**

**reg** **[**41**:**0**]** test **=** 42'b01\_0011\_0001\_1100\_0011\_1100\_0001\_1111\_0000\_0011\_1111**;**

**integer** i**;**

**initial** **begin**

**repeat(**80**)** **@(posedge** clk**);**

rst **=** 1**;**

**repeat(**20**)** **@(posedge** clk**);**

rst **=** 0**;**

**for(**i**=**41**;** i**>=**0**;** i**=**i**-**1**)** **begin**

x **=** test**[**i**];**

**@(posedge** clk**);**

**end**

**end**

**endmodule**

A screenshot of a video game

Description automatically generated

FSM Behavioral Simulation

This design works as a Mealy Machine which resulted in the output to go high within three clock cycles rather than four. This is because of the way Mealy Machine gives output. Whenever the state registered the output changes without waiting the clock which means the device works as a “Three consecutive 1s or 0s”. This is rather a design choice.

To convert a Mealy Machine into Moore Machine, the output must be synchronized with the clock. With the code above, the output z can be synchronized with the clock using “always” block. In the RTL Schematic, the DFF connected to the output z can be seen.

Moore FSM Verilog Code

`timescale 1ns **/** 1ps

**module** FSM1**(**

**input** clk**,**

**input** rst**,**

**input** x**,**

**output** **reg** z

**);**

**reg** q0**,** q1**,** q2**;**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)** **begin**

q0 **<=** 1'b0**;**

q1 **<=** 1'b0**;**

q2 **<=** 1'b0**;**

z **<=** 1'b0**;**

**end** **else** **begin**

q2 **<=** **(**x **&** q1 **&** q0**)** **|** **(**x **&** q2**);**

q1 **<=** **(~**q2 **&** **~**q1 **&** q0**)** **|** **(**q1 **&** **~**q0**)** **|** **(**x **&** **~**q2 **&** **~**q1**);**

q0 **<=** **(~**q2 **&** **~**q1 **&** **~**q0**)** **|** **(**x **&** **~**q1**)** **|** **(**x **&** **~**q0**);**

z **<=** **(~**x **&** q1 **&** **~**q0**)** **|** **(**x **&** q2 **&** q0**);**

**end**

**end**

**endmodule**

A diagram of a circuit

Description automatically generated

RTL Schematic of The Output Z (Moore)

A green squares on a black background

Description automatically generated

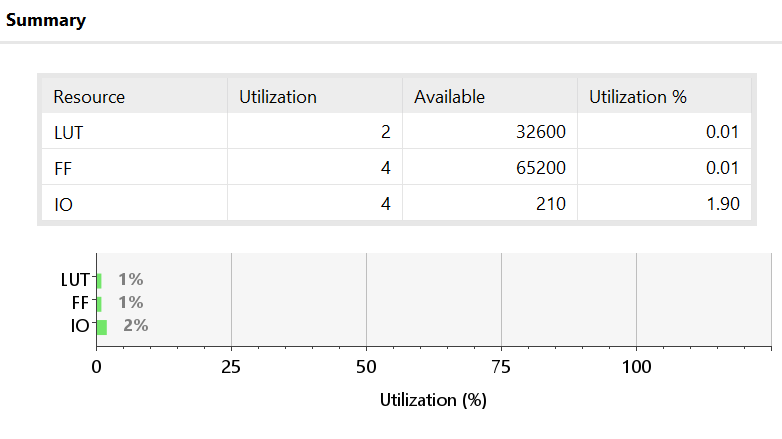
FSM Moore Behavioral Simulation

By making the design Moore, the output value synchronized with the clock which resulted an increasement in clock cycle of detecting the 1s and 0s. This is caused by the one clock cycle time requirement on the output due to the DFF. Instead of giving the output immediately after the 3rd clock cycle, while waiting for the clock for the output it receives the 4th state value correctly. As it can be seen on the simulation figure above, z only gives “1” when the input is either zero or one for four clock cycles instead of the three in the Mealy Machine. This time the design works as intended.

A screenshot of a video game

Description automatically generated

FSM Moore Post-Implementation Timing Simulation



FSM Moore Utilization Summary

A screenshot of a computer

Description automatically generated

FSM Moore Design Timing Summary

A screenshot of a computer

Description automatically generated

FSM Moore Path Delays - Setup

**STUCKING IN UNDESIREBLE STATES:**

A screenshot of a video game

Description automatically generated

Behavioral Simulation for "111" state

A screenshot of a video game

Description automatically generated

Behavioral Simulation for "110" state

As it can be seen in the simulations initiating from the states “111” and “110” do not force device to be stuck in arbitrary states. However, the first z output value of the “110” state initiation is wrong.

**BEHAVIORAL DESIGN:**

**A screenshot of a video game

Description automatically generated**

Behavioral FSM Behavioral Simulation

The simulations are the same for behavioral and Moore design. However, Moore design requires significantly more pre-computations. Behavioral design is better in terms of coding simplicity.

A diagram of a computer

Description automatically generated

Behavioral FSM RTL Schematic

In terms of RTL placement, in behavioral design, instead of using numerous gates and paths, all the cases are represented with MUXs and registers. However, this does not mean that the technology schematic would decrease similarly.

A screenshot of a graph

Description automatically generated

Behavioral FSM Utilization Summary

A screenshot of a computer

Description automatically generated

Behavioral FSM Path Delays - Setup

As it can be seen in the Utilization Summaries, the LUT usage is increase to 4 from 2 in the behavioral design. However, the longest path delay is decreased considerably. Beside the increase in “rst to z\_reg” path, all the other paths decreased a little.

A screenshot of a video game

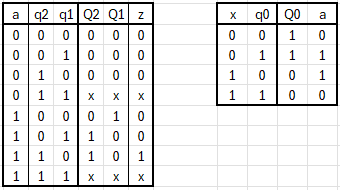
Description automatically generated

Behavioral FSM Post-Implementation Timing Simulation

1. **Design with Divided State Diagrams**

A blackboard with white text

Description automatically generated



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**REDUCED EQUATIONS**

`timescale 1ns **/** 1ps

**module** FSM2**(**

**input** clk**,**

**input** rst**,**

**input** x**,**

**output** **reg** z

**);**

**reg** q0**,** q1**,** q2**;**

**wire** a**,** Z**;**

**wire** Q0**,** Q1**,** Q2**;**

**assign** a **=** x **^** q0**;**

**assign** Q0 **=** **!**x**;**

**assign** Q1 **=** a **&** **(!**q1**)** **&** **(!**q2**);**

**assign** Q2 **=** **(**q1 **&** a**)** **|** **(**a **&** q2**);**

**assign** Z **=** a **&** q2**;**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if** **(**rst**)** **begin**

q2 **<=** 1'b0**;**

q1 **<=** 1'b0**;**

q0 **<=** 1'b0**;**

z **<=** 1'b0**;**

**end** **else** **begin**

q2 **<=** Q2**;**

q1 **<=** Q1**;**

q0 **<=** Q0**;**

z **<=** Z**;**

**end**

**end**

**endmodule**

FSM2 Verilog Code

FSM2 Testbench Code

`timescale 1ns **/** 1ps

**module** FSM2\_tb**();**

**reg** clk **=** 0**;**

**reg** rst **=** 0**;**

**reg** x **=** 0**;**

**wire** z**;**

FSM2 dUT**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**x**(**x**),**

**.**z**(**z**)**

**);**

**always** **#**5 clk **=** **~**clk**;**

**reg** **[**41**:**0**]** test **=** 42'b0100\_1100\_0111\_0000\_1111\_0000\_0111\_1100\_0000\_1111\_11**;**

**integer** i**;**

**initial**

**begin**

**repeat(**80**)** **@(posedge** clk**);**

rst **=** 1**;**

**repeat(**20**)** **@(posedge** clk**);**

rst **=** 0**;**

**for(**i**=**41**;** i**>=**0**;** i**=**i**-**1**)**

**begin**

x **=** test**[**i**];**

**@(posedge** clk**);**

**end**

**end**

**endmodule**

A screen shot of a video game

Description automatically generated

FSM2 Behavioral Simulation

A screenshot of a graph

Description automatically generated

Utilization Summary

A screenshot of a computer

Description automatically generated

FSM2 Path Delays

A screenshot of a computer

Description automatically generated

Timing Summary

A screenshot of a video game

Description automatically generated

FSM2 Post-Implementatiın Timing Simulation

**FSM2 BEHAVIORAL:**

FSM2 Behavioral Verilog Code

`timescale 1ns **/** 1ps

**module** FSM2\_behav**(**

**input** clk**,**

**input** rst**,**

**input** x**,**

**output** **reg** z

**);**

**parameter** A **=** 2'b00**,** B **=** 2'b01**,** C **=** 2'b10**,**

M **=** 1'b0**,** N **=** 1'b1**;**

**reg** **[**1**:**0**]** state1**;**

**reg** state2**;**

**reg** a**;**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)** **begin**

state1 **<=** A**;**

z **<=** 1'b0**;**

**end** **else** **begin**

**case(**state1**)**

A**:** **begin**

**if(**a**)** **begin**

state1 **<=** B**;**

z **<=** 1'b0**;**

**end** **else** **begin**

state1 **<=** A**;**

z **<=** 1'b0**;**

**end**

**end**

B**:** **begin**

**if(**a**)** **begin**

state1 **<=** C**;**

z **<=** 1'b0**;**

**end** **else** **begin**

state1 **<=** A**;**

z **<=** 1'b0**;**

**end**

**end**

C**:** **begin**

**if(**a**)** **begin**

state1 **<=** C**;**

z **<=** 1'b1**;**

**end** **else** **begin**

state1 **<=** A**;**

z **<=** 1'b0**;**

**end**

**end**

**default:** **begin**

state1 **<=** A**;**

z **<=** 1'b0**;**

**end**

**endcase**

**end**

**end**

**always** **@(posedge** clk **or** **posedge** rst**)** **begin**

**if(**rst**)** **begin**

state2 **<=** 1'b0**;**

a **<=** 1'b0**;**

**end** **else** **begin**

**case(**state2**)**

M**:** **begin**

**if(**x**)** **begin**

state2 **<=** M**;**

a **<=** 1'b1**;**

**end** **else** **begin**

state2 **<=** N**;**

a **<=** 1'b0**;**

**end**

**end**

N**:** **begin**

**if(**x**)** **begin**

state2 **<=** M**;**

a **<=** 1'b0**;**

**end** **else** **begin**

state2 **<=** N**;**

a **<=** 1'b1**;**

**end**

**end**

**default:** **begin**

state2 **<=** M**;**

a **<=** 1'b0**;**

**end**

**endcase**

**end**

**end**

**endmodule**

FSM2 Behavioral Testbench Code

`timescale 1ns **/** 1ps

**module** FSM2\_behav\_tb**();**

**reg** clk **=** 0**;**

**reg** rst **=** 0**;**

**reg** x **=** 0**;**

**wire** z**;**

FSM2\_behav DUT**(**

**.**clk**(**clk**),**

**.**rst**(**rst**),**

**.**x**(**x**),**

**.**z**(**z**)**

**);**

**always** **#**5 clk **=** **~**clk**;**

**reg** **[**41**:**0**]** test **=** 42'b0100\_1100\_0111\_0000\_1111\_0000\_0111\_1100\_0000\_1111\_11**;**

**integer** i**;**

**initial**

**begin**

**repeat(**80**)** **@(posedge** clk**);**

rst **=** 1**;**

**repeat(**20**)** **@(posedge** clk**);**

rst **=** 0**;**

**for(**i**=**41**;** i**>=**0**;** i**=**i**-**1**)**

**begin**

x **=** test**[**i**];**

**@(posedge** clk**);**

**end**

**end**

**endmodule**

A green and black screen

Description automatically generated

FSM2 Behavioral - Behavioral Simulation

A screenshot of a graph

Description automatically generated

FSM2 Behavioral Utilization Summary

A screenshot of a computer

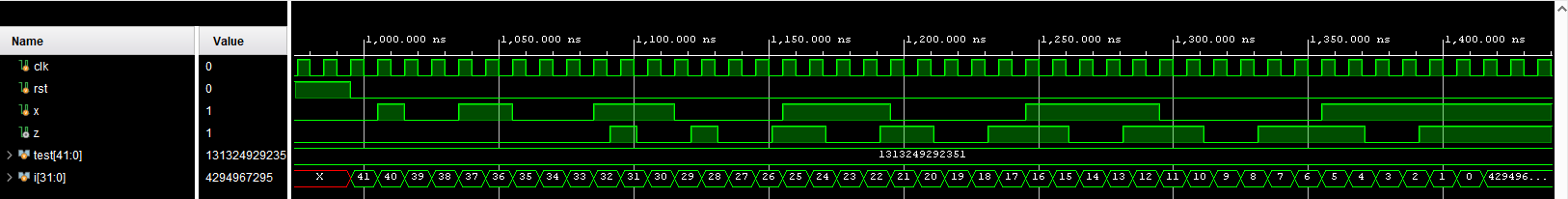
Description automatically generated

FSM2 Behavioral Timing Summary

A screenshot of a computer

Description automatically generated

FSM2 behavioral Path Delays - Setup



FSM2 Behavioral Post-Implementation Timing Simulation

To compare two different models of FSM2, it can be seen on the utilization summaries that both the LUT and FF usage is increased by 1 for behavioral model. These additions are caused by the registers that contain states. However, besides the addition of the two new paths, the existing path delays have not changed at all.

To conclude, in FSM2 the behavioral model’s coding and design is simpler but it takes more space than the first design. However, unlike FSM1, making the design behavioral does not affect the path delays of FSM2.

**References**

1. <https://www.allaboutcircuits.com/technical-articles/encoding-the-states-of-a-finite-state-machine-vhdl/>