Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Advanced Digital Design -ENCS3310

Course Project -Report

MOOORE FSM for A sequence Detector

Instructor:
Abdellatif Abu-Issa

Prepared By:
Abd Khuffash-1200970

Section:
2

Date:
29-8-2023

Abstract/Objectives

The objective of this project is to create a Moore Finite State Machine (FSM) for sequence detection. The particular sequence that is being targeted for is "1011." Using T-Flip Flops and combination logic, the project entails building a structural circuit that can precisely detect instances of the required sequence. A comprehensive testbench will be built to assess the performance of the design and spot any potential errors in order to guarantee its dependability and usefulness. To guarantee consistency, the constructed circuit will be compared to the behavioral description as part of the validation process. Asynchronous reset input will also be included in the circuit to make it easier to reset the system. An effective and precise sequence detection system will be created through the successful completion of this project, showcasing the efficient use of Moore FSM and digital logic design concepts.

Table of Contents

Contents

Table of Figures

Table Of tables:

Introduction

Finite State Machine- FSM

Finite-state machines (FSM) is a mathematical model of computation commonly used in computer programs. It is conceptualized as an abstract machine which can only reside in one out of a number of pre-defined states, between which transitions are performed to enter a different state. Once in that state, some computation or action is performed before proceeding again to the next state.

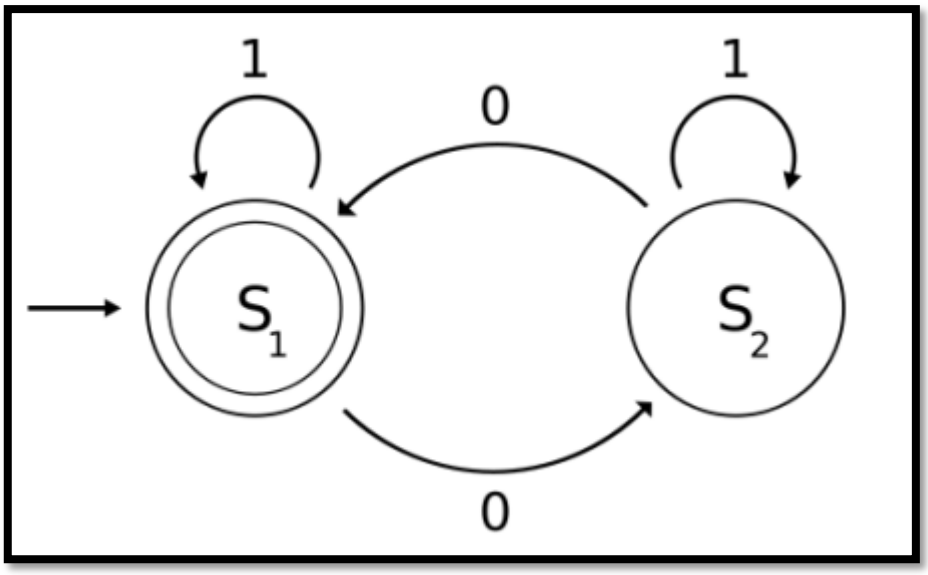

Figure 1.Example finite-state machine

FSMs are a common way to solve the high-level control problem in robotics and AI. However, there are a number of challenges and disadvantages which may arise when attempting to implement a control system using FSMs.

FSMs are known to become unmanageable for large complex systems, sometimes called the state and transition explosion. To have a fully reactive system, every state needs to have a transition to every other, making it a fully connected graph (O(n 2)). This makes maintenance and modification quite labor intensive and prone to bugs. I.e., removing a particular state requires correctly modifying those other states able to transition to the old one.

This lack of modularity also complicates delegation of work for independent components of a behavioral control system, due to state transitions being interconnected to such a high degree. FSMs simply become very hard to collaborate on as they grow.[1]

MOORE FSM Sequence Detector

Mealy and Moore machines are two different types of Finite State Machines (FSMs) that are commonly used for modeling and designing digital systems. Both types of machines define the behavior and state transitions of a system, but they differ in how they handle outputs.

Moore Machine, In this state machine, the output is determined only by the present active state of the state machine and not by any input events. No output during state transition.

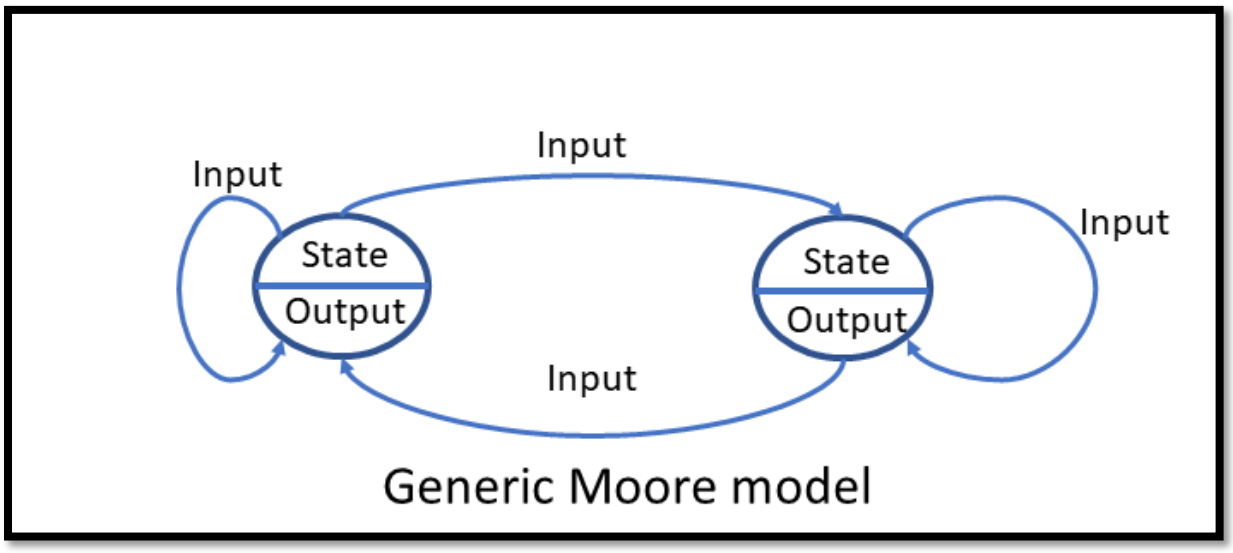

Figure 2.Generic Moore Model

The Output is represented inside the state, also called 'Action'. Output depends only on the current state, changes synchronously at the beginning of each state. The output is associated with the states.

A Moore machine can also be represented by a state diagram or a state table. The state diagram consists of circles (representing states) and arrows (representing transitions), where the outputs are typically written inside the circles representing the states.[2]

A sequence detector is a sequential state machine. In a Moore machine, output depends only on the present state and not dependent on the input (x). Hence in the diagram, the output is written with the states.

Design Philosophy

Theoretical Design

Moore FSM Sequence Detector 1011, to design a sequence detector for 4bits, we need 5 states, S0 S1 S2 S3 S4, the initial state is S0, when the input (x) is 0, this does not satisfy the sequence we need (The first bit, which is 1), so it will stay in S0, output is 0. If the input (x) is 1, the first bit is achieved, so it will move to the next state, S1 with output 0.

When it reaches the second state S1, if the input is 1, it will stay in the same state, S1. If the input (x) in zero, the next state will be S2, the sequence achieved till S2 is (10), but the output will stay (0).

When it reaches third state, S2, if the input is 0, The sequence is broken (100) and it will go back to S0, if the input is 1, the sequence in a good path, it will move to the next state S3, with output still 0.

When it reaches the fourth state, S3, if the input is 0, it will go back to state 2, sequence is back to (10), if the input is 1, the sequence in a good path (1011) and will move to the next state S4, the output will stay 0 for state 3.

When it reaches the last state (S4), the output will be 1, because the sequence is accomplished (1011). If the input is 0, it will move the second state, S2, the sequence will be (10). If the input is 1, it will move back to S1, the sequence will be (1), starting from the beginning. And so on.

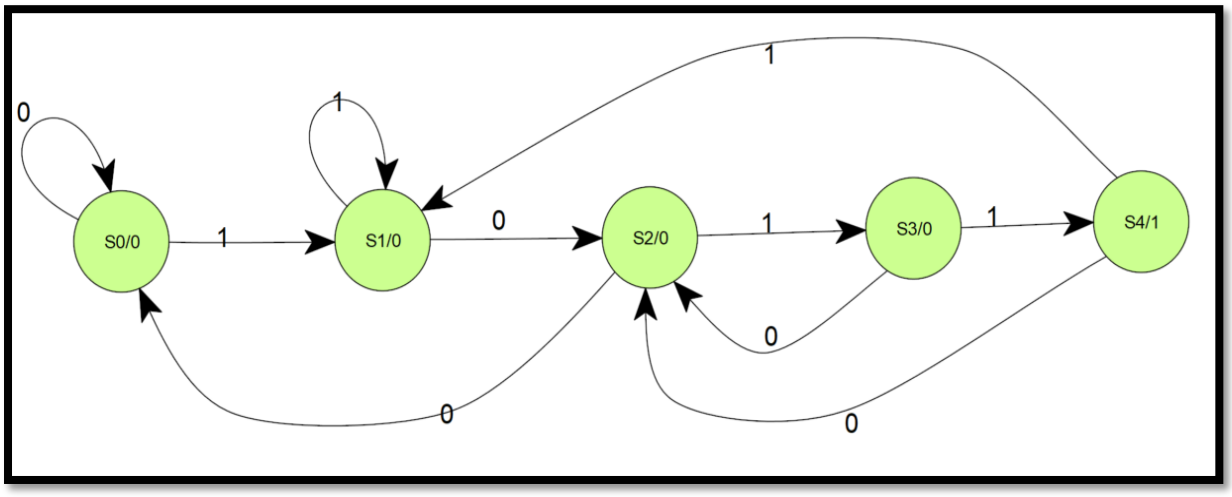

Figure 3.Sequence 1011 detector State diagram

The table below summarize what has been said above:

Table 1.Sequence 1011 detector State Table

Present State	Input(x)	Next State	Output(y)
S0	0	S0	0
S0	1	S1	
S1	0	S2	0
S1	1	S1	
S2	0	S0	0
S2	1	S3	
S3	0	S2	0
S3	1	S4	
S4	0	S2	1
S4	1	S1	

Since there are 5 states to detect a 4bit sequence, there will be 3 flip flops to be used in the making of this detector, at first:

- S0: 0 0 0, which is the initial state.
- S1: 0 0 1
- S2: 0 1 0
- S3: 0 1 1
- S4: 1 0 0

3 flip flops will be used, (Q2 Q1 Q0). Which will represent the present state, and (Q2` Q1` Q0`) will represent the next state. The table below emphasize on how the Detector will work with T flip flops:

Table 2.Sequence 1011 detector State Table with T flipflops

	Present State			**Input**	**Next State**			**Output**	**T-Flipflops**		
	Q2	**Q1**	**Q0**	**X**	**Q2`**	**Q1`**	**Q0`**	**Y**	**T2**	**T1**	**T0**
S0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	1	0	0	0	1
S1	0	0	1	0	0	1	0	0	0	1	1
	0	0	1	1	0	0	1	0	0	0	0
S2	0	1	0	0	0	0	0	0	0	1	0
	0	1	0	1	0	1	1	0	0	0	1
S3	0	1	1	0	0	1	0	0	0	0	1
	0	1	1	1	1	0	0	0	1	1	0
S4	1	0	0	0	0	1	0	1	1	1	0
	1	0	0	1	0	0	1	1	1	0	1

After that, we will be able to extract each of the T flip flop inputs equations (T2, T1, T0), a Karnaugh map, is constructed for each T-Flipflop.

Starting with T2:

Q2Q1 \ Q0X	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	1	1	X	X

Table 3. T2 K-Map

The equation For T2= Q2 + Q1Q0X

For the next T-Flipflop, T1:

Q2Q1 \ Q0X	00	01	11	10
00	0	0	0	1
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

Table 4.T1 K-Map

The equation for T1 =Q2X` + Q1Q0X + Q1Q0`X`+ Q1`Q0X`

For the last T-Flipflop, T0:

Q0X

Q2Q1	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	x	x	x	x
10	0	1	x	x

Table 5.T0 K map

The equation for T0=Q0X` + Q0`X

For the Output (Y):

Q1Q0

Q2	00	01	11	10
0	0	0	0	0
1	1	x	x	x

Table 6. Output Y K-Map

The equation for the output Y= Q2

The Block Diagram for the Circuit:

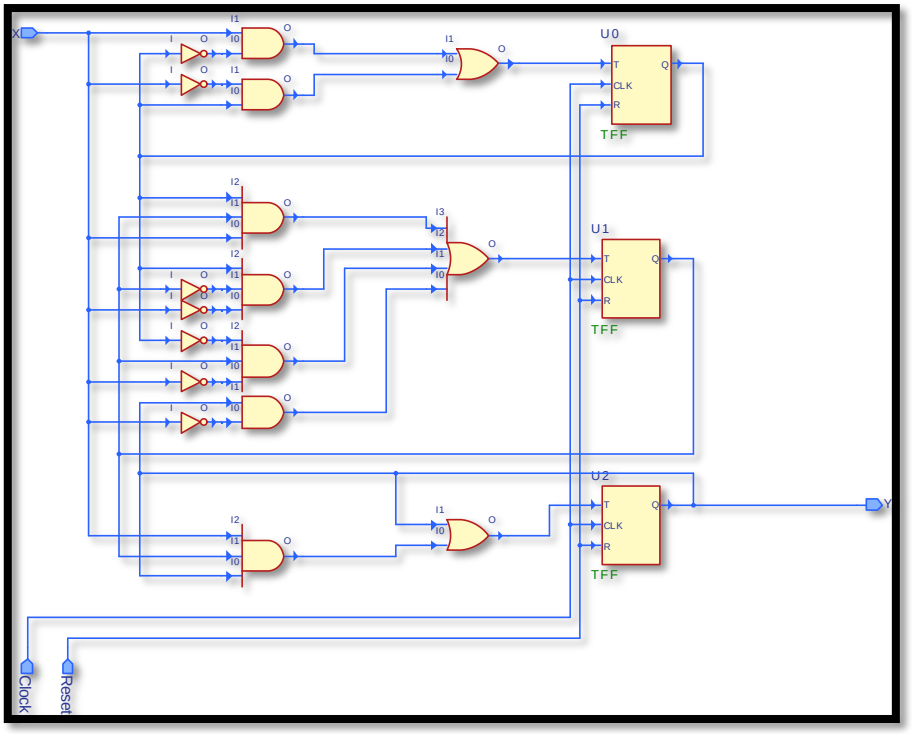

Figure 4.Block Diagram for the Circuit

3 Or gates, 7 and Gates, 7 invertors.

Inputs: X, Clock and Reset.

Output: Y.

Verilog Design

T-Flipflop Code & Test Bench

The figure below shows A T flipflop code:

```
module TFF(T, CLK, R, Q);
    // Define a module named "TFF" with the following ports:
    // - T: Input signal for toggling the flip-flop state.
    // - CLK: Clock input signal.
    // - R: Reset input signal.
    // - Q: Output signal representing the state of the flip-flop and assing value 0 to it.
    input T,CLK,R;
    output reg Q=0;

    always @(posedge CLK, negedge R)
        // Always block triggered by the rising edge of clock(CLK) or the falling edge of reset(R).

        if (~R)
            // If the reset signal R is low (active low reset), ~R=1.
            Q <= 0; // Set the output Q to logic 0 (reset state).

        else
            // If the reset signal R is high (not in reset condition).
            Q <= T ^ Q;
            // Perform an XOR (^) operation between the input T and the current value of Q.
            // The result is assigned to the output Q. This effectively toggles the Q and gets its negate value, if T is 1.
endmodule
```

Figure 5.T-Flipflop Code

The figure below is a test bench for T-Flipflop:

```
module TFF_tb;
    // Inputs
    reg T;
    reg CLK;
    reg R;

    // Outputs
    wire Q;

    // Instantiate the TFF module
    TFF uut (.T(T),.CLK(CLK),.R(R),.Q(Q));

    // Clock generation
    always #5 CLK = ~CLK;

    // Initial values
    initial begin
        CLK = 0;
        T = 0;
        R = 1;


        // Test sequence
        #30 T = 0;
        #40 T = 1;
        #50 T = 0;
        #60 T = 1;

        #100 $finish;
    end

    // Display output
    always @(posedge CLK) begin
        $display("Time = %0t | T = %b | R = %b | Q = %b", $time, T, R, Q);
    end

endmodule
```

Figure 6. T-Flipflop Test Bench Code

System Code & Test Bench

The figure below demonstrates the code for a structural build for a circuit that detects a sequence “1011”.

```
module Sys(X,CLK,RES, Y);

    input X,CLK,RES;
    output  Y=0;

    wire T0,T1,T2;
    wire Q0,Q1,Q2;
    wire a,b,c,e,f,g,h,i;


    TFF TFF0( .T(T0), .CLK(CLK), .R(RES) , .Q(Q0));
    TFF TFF1( .T(T1), .CLK(CLK), .R(RES) , .Q(Q1));
    TFF TFF2( .T(T2), .CLK(CLK), .R(RES) , .Q(Q2));

    //T0=q0x`+q0`x
    assign T0=((Q0 & ~X) | (~Q0 & X));
    //and a1(a,X,~Q0);
    //and a2(b,~X,Q0);
    //or a3(T0,a,b);

    //T1=Q2x`+q1q0x+q1q0`x`+q1`q0x`
    assign T1=((Q2 & ~X) | (Q1 & Q0 & X) | (Q1 & ~Q0 & ~X) | (~Q1 & Q0 & ~X));
    //and a4(c,Q2,~X);
    //and a5(e,Q1,Q2,X);
    //and a6(f,Q1,~Q0,~X);
    //and a7(g,~Q1,Q0,~X);
    //or a8(T2,c,e,f,g);

    //T2=q2 + q1q0x
    assign T2=(Q2 | (Q1 & Q0 & X));
    //and a9(h,Q1,Q0,X);
    //or a10(T2,h,Q2);
    //y
    assign Y=Q2;
endmodule
```

Figure 7.Structural build For the System

The figure below shows the code for a test bench of the system:

```
module test2;
    reg x, clk, r;
    wire y;
    //Instanse of the system
    Sys c(.X(x), .CLK(clk), .RES(r), .Y(y));

    initial begin
        // Initialize Inputs
        x = 0;
        r = 0;
        clk = 0;
        #30ns;
        r = 1;
        #10ns;
        x = 0;
        #10ns;
        x = 1;
        #10ns;
        x = 0;
        #10ns;
        x = 1;
        #10ns;
        // Generate the clock aligned with the sequence
        // Assuming the sequence is detected in the first rising edge of the clock after detection
        #10ns;
        clk = 1;
        #10ns;
        clk = 0;
        #10ns;
        clk = 1;

        #200 $finish;
    end
    always #5ns clk = ~clk;
    // Display output
    always @(posedge clk) begin
        $monitor("Time = %0t | X = %b | Y = %b", $time, x, y);
    end
endmodule
```

Figure 8.System Test Bench Code

This Test bench will be used to generate a wave form for the system in structural build, also it will be used in the Behavioral build, to compare results.

The sequence to be detected is “1011”, at first the input signal is set to 0, reset is set to 0, and clock is also set to zero, after a delay of 30ns the reset will be 1(will not reset the values since the system is negative edge reset triggered, Active low reset), a further delay of 10ns, x=0, another delay of 10ns x=1, this satisfies the first bit of the sequence wanted in “1011” and so on.

To align the clock, a delay of 10ns clock =1, another 10ns the clock =0. An always block with change every 5ns.

For the Behavioral section, at first, the figure below represents a behavioral way of writing a code for a sequence detector circuit:

```
module SysBeh(x,clk,r,y);
    input x,clk,r;
    output reg y;
    parameter s0=3'b000,s1=3'b001,s2=3'b010,s3=3'b011,s4=3'b100;
    reg [2:0] ps,ns;

    always@(posedge clk , negedge r) // works on positive edge for the clock , and negative edge of the reset ,active low reset.
        begin
            if(~r)
                ps<=s0;
            else
                ps<=ns;
        end

    always @(ps,x)
        begin
            case(ps) //checks the posiblites of states, and assigns the ns , y for each case.
                s0:
                begin
                    y=0;
                    if(x==1)
                        ns=s1;
                    else
                        ns=s0;
                end

                s1:
                begin
                    y=0;
                    if(x==1)
                        ns=s1;
                    else
                        ns=s2;
                end

                s2:
                begin
                    y=0;
                    if(x==1)
                        ns=s3;
                    else
                        ns=s0;
                end

                s3:
                begin
                    y=0;
                    if(x==1)
                        ns=s4;
                    else
                        ns=s2;
                end

                s4:
                begin
                    y=1;
                    if(x==1)
                        ns=s1;
                    else
                        ns=s2;
                end
            default:ns=s0;
            endcase
        end
endmodule
```

Figure 9. Behavioral Build for the System.

For the test bench it's the same in figure 8, and it will be used to for both structural build and behavioral build.

Results

T-Flipflop Test Bench Results

Referring to figure 6, the figure below is Console Results for a TFF test bench:

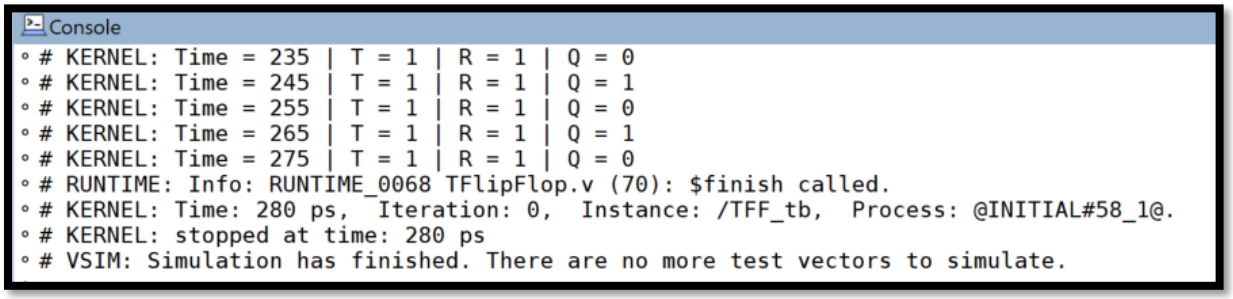

Figure 10.Console Results for TFF tb

The figure below shows the wave form for a TFF test bench:

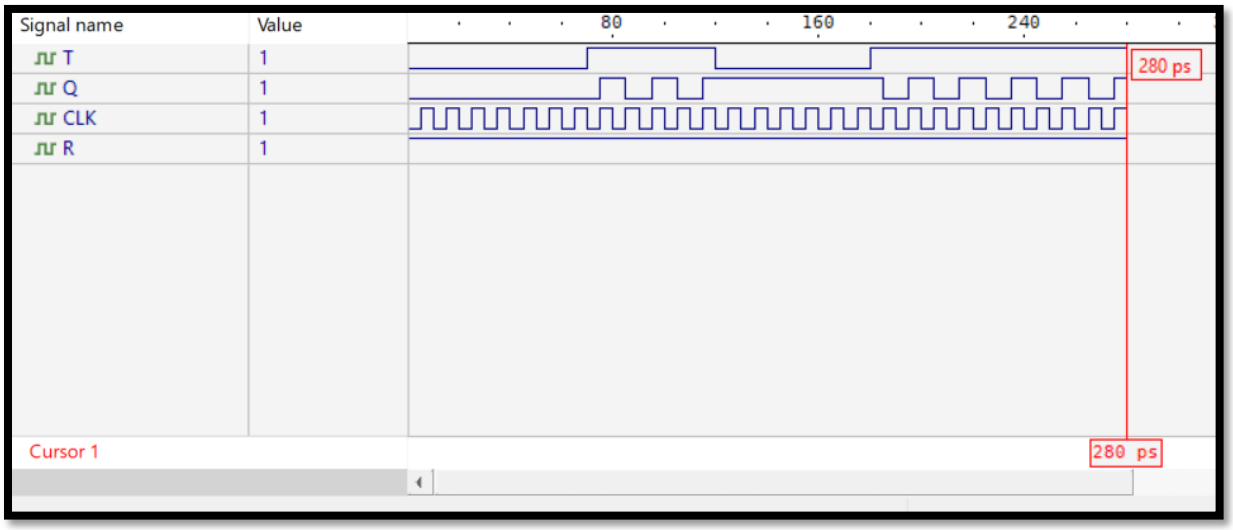

Figure 11. Wave Form Results for TFF tb

System Test Bench Results

For the Structural Build, the figure below demonstrates, how the wave form of the system will be:

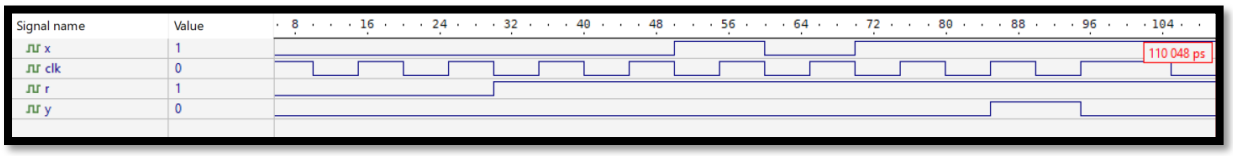

Figure 12. Structural build wave form

At time 50ns the input signal changes from 0 to 1, the first bit is accomplished "1011", then at 55ns a clock rises, so the input will be considered and changed to zero at 60ns, which means the flipflops contains "10", which does not satisfy the sequence 1011, output will stay 0. Then at 65ns a clock occurs again, so the value of x will change or remain the same depends on it. In this case the value of x change from 0 to 1, so the sequence will be "101". Then a clock signal triggered again at 75ns, the value of x will remain the same 1.

The sequence is 1011, so the output y will be 1, since the wanted sequence is accomplished.

The figure below shows the output in the console, generated by monitor line in the test bench:

```
# KERNEL: Time = 5000 | X = 0 | Y = 0
# KERNEL: Time = 50000 | X = 1 | Y = 0
# KERNEL: Time = 60000 | X = 0 | Y = 0
# KERNEL: Time = 70000 | X = 1 | Y = 0
# KERNEL: Time = 85000 | X = 1 | Y = 1
# KERNEL: Time = 95000 | X = 1 | Y = 0
```

Figure 13.Structural System Console results

The second line, input x is 1, next 0, next 1, next 1, so the output is 1. Then input is x=1, sequence broken, output y=0.

For the Behavioral Build, the figure below demonstrates the wave form of the System:

Figure 14.Behavioral build wave form

The wave form is the same in structural system save form.

The figure below is a console results generated by ($monitor):

```
# KERNEL: Time = 5000 | X = 0 | Y = 0
# KERNEL: Time = 50000 | X = 1 | Y = 0
# KERNEL: Time = 60000 | X = 0 | Y = 0
# KERNEL: Time = 70000 | X = 1 | Y = 0
# KERNEL: Time = 85000 | X = 1 | Y = 1
# KERNEL: Time = 95000 | X = 1 | Y = 0
# RUNTIME: Info: RUNTIME_0068 SysBeh.v (127): $finish called.
```

Figure 15.Behavioral System Console results

Conclusion

the design and implementation of a Moore Finite State Machine (FSM) sequence detector for the pattern "1011" have been successfully achieved. This sequence detector utilizes five states (S0, S1, S2, S3, and S4) and three flip-flops (Q2, Q1, and Q0) to track the current state and transition to the next state based on the input bit (X). The state transitions are controlled by the T flip-flops (T2, T1, and T0) equations, which were derived from Karnaugh maps.

The state transition equations for the T flip-flops are as follows:

T2 = Q2 + Q1Q0X

T1 = Q2X` + Q1Q0X + Q1Q0`X` + Q1`Q0X`

T0 = Q0X` + Q0`X

The output Y is determined by the current state Q2, as given by the equation Y = Q2. This output signal is high (1) when the sequence "1011" is detected in the input stream.

The structural and behavioral designs, despite having different underlying implementations, exhibit identical output waveforms, thereby demonstrating their functional equivalence. This remarkable similarity in output behavior underscores the correctness and reliability of both design approaches. The structural design relies on a hierarchical arrangement of components and connections, specifying the exact hardware elements and their interconnections. In contrast, the behavioral design abstracts the system's operation, emphasizing the logical relations and operations between components without detailing the hardware. The fact that both designs yield identical output waveforms reaffirms that they are both valid representations of the same logical functionality. This consistency in results not only reinforces their equivalence but also highlights the flexibility and versatility of different design methodologies in achieving the same desired outcome.

In the Structural Build, using ONLY gates, I encountered an issue where the output Y in my Verilog Moore finite state machine consistently showed an "x" or an unknown value during simulation, so Instead I used “assign”.

To address this problem, I initially ensured that I correctly assigned the output Y to the state variable Q2 at the end of my “Sys” module (Figure7), as intended. I also verified that the inputs X, CLK, and RES were properly initialized and behaved as expected in my testbench. Additionally, I reviewed my T flip-flop (TFF) implementations (TFF0, TFF1, and TFF2) to ensure they were functioning correctly and producing valid outputs. Despite these efforts, the issue persisted, suggesting that there may be subtle errors or misconfigurations in my simulation setup.

Another issue I encountered was that, in my Moore finite state machine, when the "1011" sequence occurred twice, only the first occurrence caused the output to transition to a "1," while subsequent occurrences were not detected and did not affect the output (Stays 0).

References

[1] Behavior Trees for decision-making in Autonomous Driving, by MAGNUS OLSSON (3:12 #29-8-23) Available: https://www.diva-portal.org/smash/get/diva2:907048/FULLTEXT01.pdf

[2] Mealy and Moore machine. (3:20 #29-8-23) Available: https://fastbitlab.com/mealy-and-moore-machine/