Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Advanced Digital System Design ENCS3310

Project Report

Prepared by:

Baraa Nasar 1210880

Instructor: Dr. Abdallatif Abuissa

Section: 1

Date: 28-8-2023

ABSTRACT

This report outlines the design and implementation of a Moore Finite State Machine (FSM) for a sequence detector. The detector is programmed to identify the specific sequence "1011". The project involves constructing a structured circuit that combines T-Flip Flops and combination logic to accurately detect the given sequence. An additional asynchronous reset input is incorporated to enable the resetting of the circuit.

The primary objective of this project is to develop a functional sequence detector that efficiently recognizes the pattern "1011" within input data streams. The detector's operation is based on a Moore FSM architecture, where the state transitions are governed by the input signals and the current state. The implemented design is subjected to rigorous testing using a comprehensive testbench. This testbench is used to validate the functionality of the detector, ensuring that it reliably identifies the sequence "1011" and produces the desired outputs.

The report underscores the significance of reliable sequence detection in various applications such as data analysis, error checking, and communication protocols. The successful realization and testing of the Moore FSM sequence detector demonstrate its effectiveness in accurately identifying the specified pattern. By comparing the circuit's behavior with its behavioral description, the report provides conclusive evidence of the design's accuracy and reliability.

In conclusion, the developed Moore FSM sequence detector offers a valuable solution for detecting the sequence "1011" in data streams. Its ability to function accurately, coupled with its flexibility for integration into various systems, highlights its potential applicability in a multitude of domains.

Table of Contents

List of Figures

List of Tables

Brief introduction and background

The aim of this project is to design a Moore Finite State Machine (FSM) for a sequence detector capable of recognizing the pattern "1011". A sequence detector is a fundamental component in digital circuit design and finds applications in various fields, including telecommunications, data processing, and control systems.

The task involves creating a structural circuit using T-Flip Flops and combinational logic to accurately detect the "1011" pattern. Detecting such sequences is crucial in larger digital systems, where the circuit can appropriately respond when specific data patterns are found.

In addition to circuit design, a comprehensive testing station will be developed to rigorously verify the design's functionality. This station will execute a series of test cases, comparing the actual circuit outputs to the expected results, identifying any discrepancies or errors, and addressing them.

Furthermore, to ensure the circuit's robustness and reliability, an asynchronous reset input will be included. This input allows for an immediate circuit reset when needed, preventing unexpected behavior.

In conclusion, this project aims to design a Moore FSM sequence detector for the "1011" pattern using T-Flip Flops and combinational logic. The subsequent verification and testing phases will ensure the circuit operates accurately and reliably, paving the way for its integration into larger and more complex digital systems.

Design philosophy

Step 1: Develop the state diagram

First, in step1, I'm designing a "1011" overlapping sequence detector, using Moore Model in Verilog. The Moore FSM keeps detecting a binary sequence from a digital input and the output of the FSM goes high only when a "1011" sequence is detected. The state diagram of the Moore FSM for the sequence detector is as shown below:

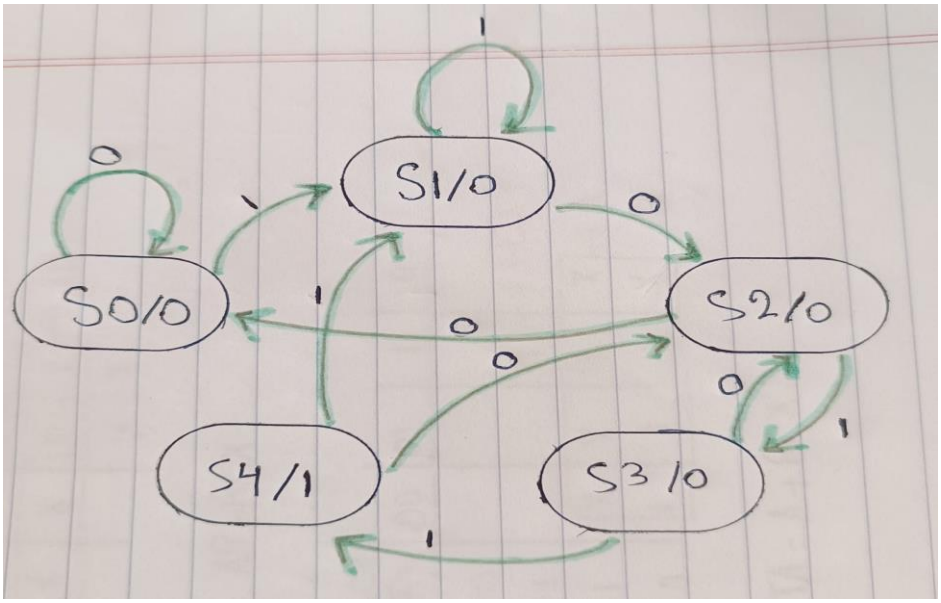

Figure 1: The state diagram of the Moore FSM for the "1011"sequence detector

The state diagram is a graphical representation of the logical circuit designed using T-Flip Flops to detect the sequence "1011." This state diagram illustrates how states transition between different states based on the input signals and represents the sequence detection process.

Let's explain the state diagram for this circuit:

1. **State 0 (S0)**: This is the initial state of the circuit. When the circuit starts or when a reset signal is applied, it enters this state. In this state, the **sequence_detected** output is disabled, and its value is 0.
2. **State 1 (S1)**: If the current state is S0 and the **data_in** signal has a value of 1, the circuit transitions to state S1. In this state, **sequence_detected** is disabled, and its value is 0.
3. **State 2 (S2)**: If the current state is S1 and the **data_in** signal has a value of 0, the circuit transitions to state S2. In this state, **sequence_detected** is disabled, and its value is 0.
4. **State 3 (S3)**: If the current state is S2 and the **data_in** signal has a value of 1, the circuit transitions to state S3. In this state, **sequence_detected** is disabled, and its value is 0.

5. **State 4 (S4)**: If the current state is S3 and the **data_in** signal has a value of 1, the circuit transitions to state S4. In this state, **sequence_detected** is activated, and its value becomes 1, indicating the detection of the "1011" sequence.

This state diagram illustrates how the circuit is configured and how states transition between T-Flip Flop units based on the input signals, which represent the received data.

Step 2: Make Present State/Next State table

We'll use T-Flip Flops for design purposes.

Present State			Input	Next State			Output	Flip-Flop Excitations		
A	B	C	X	A	B	C	Y	TA	TB	TC
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	1
0	0	1	0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0
0	1	0	1	0	1	1	0	0	0	1
0	1	1	0	0	1	0	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1
1	0	0	0	0	1	0	1	1	1	0
1	0	0	1	0	0	1	1	1	0	1

Table 1: State table of the Moore FSM for the "1011"sequence detector

Step 3: Draw K-maps for TA, TB, TC and output (Y) :

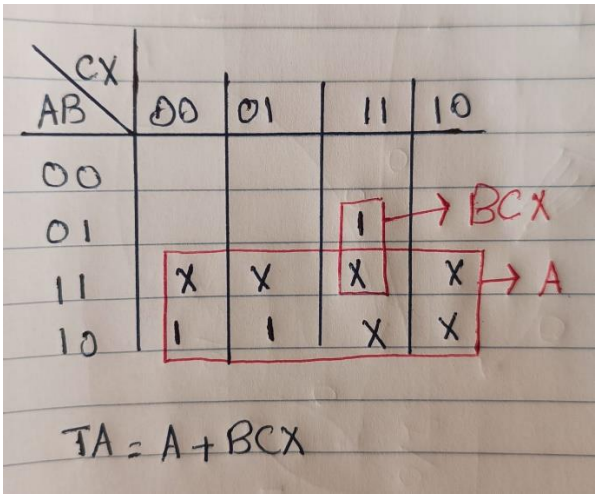

Table 2: The K-maps for TA

>> TA=A+B&CX

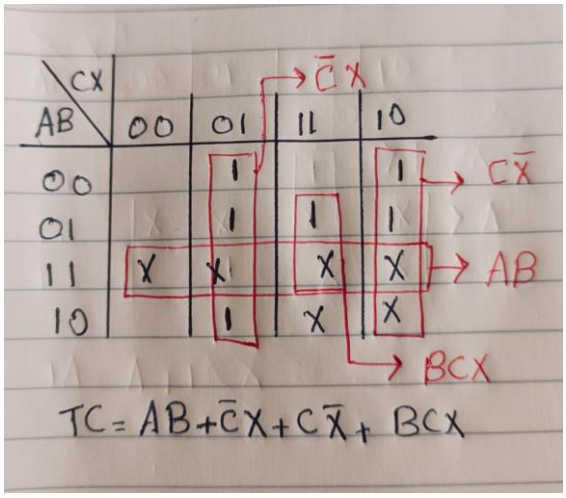

Table 3: The K-maps for TC

>> TC=AB+C^X+BCX

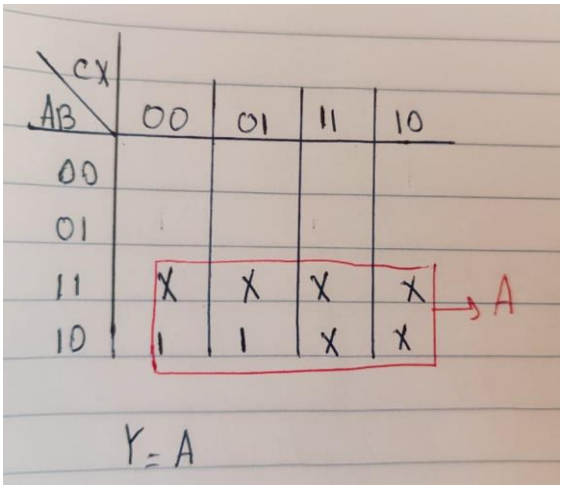

Table 4: The K-maps for output Y

>>Y=A

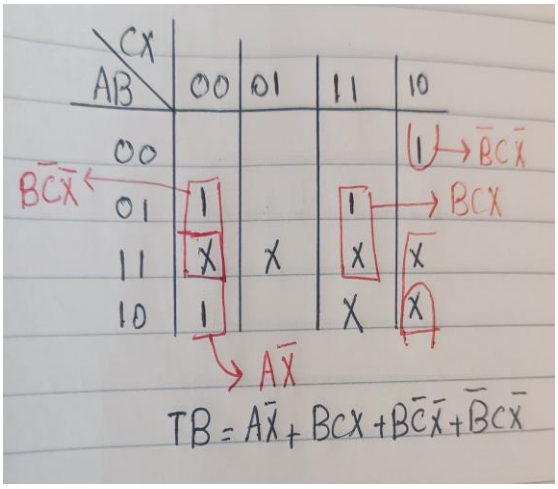

Table 5: The K-maps for TB

>>TB=A&~X+B&C&X+B&~C&~X+~B&C&~X

Step 4:T-flip flop module

A T-Flip Flop (Toggle Flip Flop) is a type of flip-flop where the output toggles (changes its state) based on the value of the T (toggle) input.

4.1: Block Diagram of T-flip flop

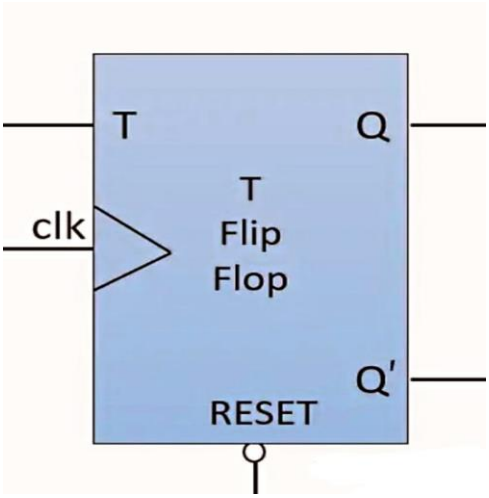

Figure 2: Block Diagram of T-flip flop

4.2: Truth Table of T-flip flop

Clock	INPUT		OUTPUT	
	RESET	T	Q	Q'
X	LOW	X	0	1
HIGH	HIGH	0	No Change	
HIGH	HIGH	1	Toggle	
LOW	HIGH	X	No Change	

Table 6: Truth Table of T-flip flop

The Q and Q' represents the output states of the flip-flop. According to the table, based on the input the output changes its state. But, the important thing to consider is all these can occur only in the presence of the clock signal.

RESET: The RESET pin has to be active HIGH. All the pins will become inactive upon LOW at RESET pin. Hence, this pin always pulled up and can be pulled down only when needed.

4.3:T-Flip Flop Verilog Code

This code represents a Verilog module for a T flip-flop. The flip-flop changes its output state based on the input signal T when a positive edge occurs in the clock signal. If the reset signal is low, the flip-flop's output is set to zero regardless of the T signal.

In summary, this module implements the fundamental behavior of a T flip-flop in Verilog.

```
1  //Baraa Nasar
2  //1210880
3  //Sec:1
4  //T flip-flop module
5  module  T_flipflop(Q,T,clock,reset);
6    output Q;
7    input T,clock,reset;
8    reg Q;
9
10   always@(posedge clock or negedge reset)
11     if(~reset)
12         Q=1'b0;
13     else
14         Q=Q^T;
15 endmodule
16
```

Figure 3: T-Flip Flop Verilog Code

4.4: T-Flip Flop Test bench

The TFF Testbench provides a comprehensive testing environment to validate the T-Flip Flop module's operation under various conditions. It enables designers to observe how the module responds to clock signals, input changes, and resets, ensuring that it functions correctly according to its design specifications. This simulation is a crucial step in the digital design verification process before moving to hardware implementation

```
17  //Baraa Nasar
18  //1210880
19  //Sec:1
20  //T flip-flop Testbench  module
21  module TFF_Testbench;
22    reg clk, reset, T;
23    wire Q;
24
25    // Instantiate the T flip-flop module
26    T_flipflop dff(Q, T, clk, reset);
27
28    // Clock generation
29    always begin
30      #5 clk = ~clk;
31    end
32
33    // Initial block
34    initial begin
35      // Initialize signals
36      clk = 0;
37      reset = 0;
38      T = 0;
39
40      // Apply reset
41      reset = 1;
42      #10 reset = 0;
43
44      // Test 1: T = 0
45      #5 T = 0;
46
47      // Test 2: T = 1
48      #5 T = 1;
49
50      // End simulation
51      $finish;
52    end
53  endmodule
54
```

Figure 4: T-Flip Flop Testbench code

Step5: Build Structural circuit using T-Flip and combination logic

I built a structural circuit using T-Flip Flops and combination logic, that detect the sequence 1011 by calling full adder module 3 times, and the input(reset, clock,input_x) and output Y1 . I built it as shown below:

```
57  //Baraa Nasar
58  //1210880
59  //Sec:1
60  //structural circuit using T-Flip Flops
61  module SequenceDetector (
62      input  clock,
63      input  reset,
64      input  in_x,
65      output reg Y1
66  );
67       wire A,B,C,TA,TB,TC;
68
69
70       or(TA,A,(B&C&in_x)); //TA=A+BCX
71
72       or(TB,(A&~in_x),(B&~C&~in_x),(B&C&in_x),(~B&C&~in_x));
73
74       or(TC,(A&B),(C^in_x),(B&C&in_x));//TC=AB+C^X+BCX
75
76
77       T_flipflop G1( A,TA,clock,reset);
78       T_flipflop G2( B,TB,clock,reset);
79       T_flipflop G3( C,TC,clock,reset);
80       assign Y1=A;
81
82  endmodule
```

Figure 5: structural circuit using T-Flip Flops and combination logic code

Step6: Build behavioural description of the circuit.

I built behavioural description of the circuit. I built it as shown below:

```
85  //Baraa Nasar
86  //1210880
87  //Sec:1
88  //behavioural description of the circuit
89  module sqd_moore( output reg in_x, output reg clock, output reg reset, output reg Y);
90
91      initial begin
92         in_x =0;
93         clock =0;
94          #10ns reset =1;
95          #10ns in_x=1;
96          #20ns in_x=0;
97          #30ns in_x=1;
98          #40ns in_x=1;
99      end
100 initial begin
101 repeat (11)
102         #20ns clock=~clock;
103 end
104
105     parameter S0=3'b000,
106               S1=3'b001,
107               S2=3'b011,
108               S3=3'b010,
109               S4=3'b110;
110
111     reg[1:0] Prsent_state,Next_state;
```

```
113     always @(posedge clock or negedge reset)
114        begin
115         if(reset==0)
116           Prsent_state <= S0;
117         else
118          Prsent_state<=Next_state;
119        end
120        begin
121
122       always @(Prsent_state ,in_x)
123         begin
124          case (Prsent_state)
125             S0:begin
126                if(in_x==1)
127                  Next_state <= S1;
128                else
129                  Next_state <= S0;
130                end
131             S1:begin
132                if(in_x==0)
133                  Next_state <=S2;
134                else
135                  Next_state <=S1;
136                end
```

```
137             S2:begin
138                 if(in_x==0)
139                   Next_state  <=S0;
140                 else
141                   Next_state  <=S3;
142                 end
143             S3:begin
144                 if(in_x==0)
145                   Next_state <=S2;
146                 else
147                   Next_state <=S4;
148                 end
149             S4:begin
150                 if(in_x==0)
151                   Next_state <=S2;
152                 else
153                   Next_state <=S1;
154                 end
155         endcase
156             end
```

```
157     always @(Prsent_state)
158     begin
159     case (Prsent_state)
160     S0: Y<=0;
161     S1: Y<=0;
162     S2: Y<=0;
163     S3: Y<=0;
164     S4: Y<=1;
165     endcase
166     end
167     end
168
169 endmodule
```

Figure 6: behavioural description of the circuit code

Module Overview:

The `sqd_moore` module is designed to implement a Moore Finite State Machine (FSM) for sequence detection. It aims to detect the specific sequence "1011" within a stream of binary input data. This module utilizes a clock signal, a reset signal, and an input signal (`in_x`) to perform the sequence detection.

Initialization:

The module begins with an initial block that sets the initial values for `in_x`, `clock`, and `reset`. It also generates a clock signal with a specified clock period, simulating the passage of time within the module.

State Definitions:

The module defines five distinct states, namely `S0`, `S1`, `S2`, `S3`, and `S4`, to represent different phases or conditions of the sequence detection process.

State Transition:

The module employs an `always` block sensitive to the rising edge of the clock signal or the falling edge of the reset signal to manage state transitions. When the reset signal is low, the FSM resets to the initial state `S0`. Otherwise, it transitions to the next state (`Next_state`) based on the current state (`Present_state`) and the value of the input signal `in_x`.

State Transition Logic:

Within the `always` block, a `case` statement is used to determine the `Next_state` based on the current state and the value of `in_x`. For example, if the current state is `S0` and `in_x` is equal to 1, the `Next_state` is set to `S1`. This logic defines the sequence detection criteria.

Output Generation:

The `Y` output represents the result of the sequence detection. It is set based on the current state. When the FSM is in state `S4`, `Y` is set to 1, indicating that the target sequence "1011" has been detected. In all other states, `Y` is set to 0.

In summary, the `sqd_moore` module is designed as a Moore FSM for sequence detection. It tracks the state of the detection process, transitions between states based on inputs and predefined criteria, and provides an output signal `Y` to indicate whether the target sequence is detected or not.

Step7: Build testanlyser module

```
172
173   //Baraa Nasar
174   //1210880
175   //Sec:1
176   module testanlyser (clock, Y1 , Y);
177       input clock,Y1 , Y  ;
178
179       always @(posedge clock) begin
180           if (Y1==Y)
181               $display("True");
182           else
183               $display("false");
184       end
185
186   endmodule
```

Figure 7: testanlyser module code

The "testanlyser" module is designed to verify the performance of the circuit by comparing the signals Y1 and Y along with the clock signal.

1. Verification Purpose: The main purpose of this module is to ensure that the output signal Y1, which is generated by the "SequenceDetector" module, matches the expected output Y based on the behavioral description of the circuit.

2. Signal Comparison: Inside this module, there is an "always" block that is sensitive to the rising edge (posedge) of the clock signal. This means that whenever the clock signal transitions from low to high, the block is activated.

3. Comparison Logic: Within the "always" block, there is logic that checks if the signals Y1 and Y match. If they match, it means that the circuit is behaving as expected, and the module displays "True" as the output.

4. Verification Process: During simulation, as the clock signal toggles, this module continuously compares Y1 and Y. If they are consistent, it confirms that the circuit's behavior matches the expected behavior, and it outputs "True." If there is a mismatch between Y1 and Y, it would output "False," indicating that there might be an issue in the circuit.

In summary, the "testanlyser" module serves as a verification tool to confirm that the circuit, particularly the "SequenceDetector" module, is functioning correctly based on the expected behavioral description. It does this by continuously comparing the actual output Y1 with the expected output Y whenever the clock signal rises. If they match, it signals that the circuit is working as intended.

Step8: Build testbench module:

```
187
188 //Baraa Nasar
189 //1210880
190 //Sec:1
191 module testbench ;
192
193     wire in_x,clock,reset;
194     reg Y1 , Y ;
195
196     sqd_moore t1(in_x,clock,reset,Y);
197
198     SequenceDetector t3(clock,reset,in_x,Y1);
199
200     testanlyser t2(clock, Y1 , Y);
201
202 endmodule
```

Figure 8: testanlyser module

Module Overview:

1. module testbench : This module represents the main environment for testing the circuit. It is used to set up the testing scenario and connect with other modules.

2. wire in_x, clock, reset: These wires are used to create input signals. In this case, they are used to create signals in_x, clock, and reset.

3. reg Y1, Y: These variables represent the signals that will be used to monitor and analyze the results. `Y1` and `Y` are the variables used to verify the correctness of signals generated by the circuit.

4. sqd_moore t1(in_x, clock, reset, Y): This line creates the sqd_moore module and connects it to the required input signals and timing signals.

5. SequenceDetector t3(clock, reset, in_x, Y1): Here, the SequenceDetector module is created and connected to the same signals.

6. testanlyser t2(clock, Y1, Y): This module is created to monitor signals and verify the performance of the circuit. It will compare the signal `Y1` with `Y` and print "True" if they match correctly.

In summary, the testbench module sets up the necessary environment for circuit testing, generates input signals, monitors the signals generated by the circuit, and verifies that it is functioning correctly.

Results

The test benches were designed and executed to verify the accuracy of the implemented special counter circuit. The results obtained from the test benches matched the expected outcomes.

>> T flip flop testbench :

The simulation results demonstrate the functionality of the TFF module. The Q output toggles and the T input is asserted. When the T input is deserted, the Q output retains its current value. The reset signal is used to initialize the Q output to 0 at the start of the simulation.

Overall, the TFF module successfully exhibits the behavior of a toggle flip-flop circuit, as observed in the simulation results.

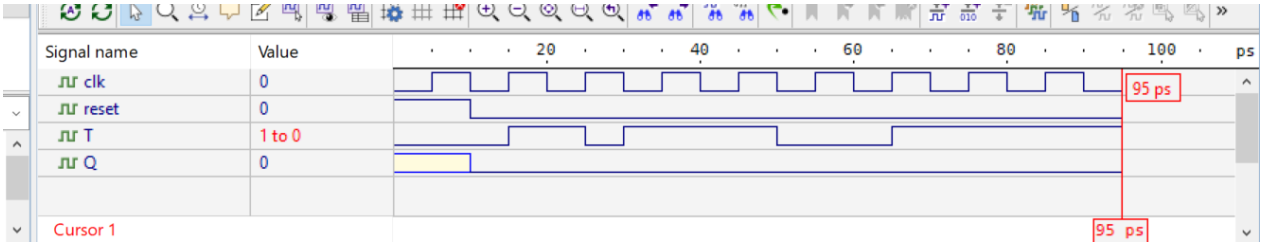

Figure 9: T-flip flop testbench simulation

```
run 100 ns
# KERNEL: At time                    0, Q = x, reset = 1, T = 0
# KERNEL: At time                   10, Q =x, reset = 0, T= 0
# KERNEL: At time                   10, Q =0, reset = 0, T= 0
# KERNEL: At time                   15, Q =0, reset = 0, T= 1
# KERNEL: At time                   25, Q =0, reset = 0, T= 0
# KERNEL: At time                   30, Q =0, reset = 0, T= 1
# KERNEL: At time                   50, Q =0, reset = 0, T= 0
# KERNEL: At time                   65, Q =0, reset = 0, T= 1
# RUNTIME: Info: RUNTIME 0068 Proj1210880.v (52): $finish called.
```

Figure 10: T-flip flop testbench result

>>testbench module:

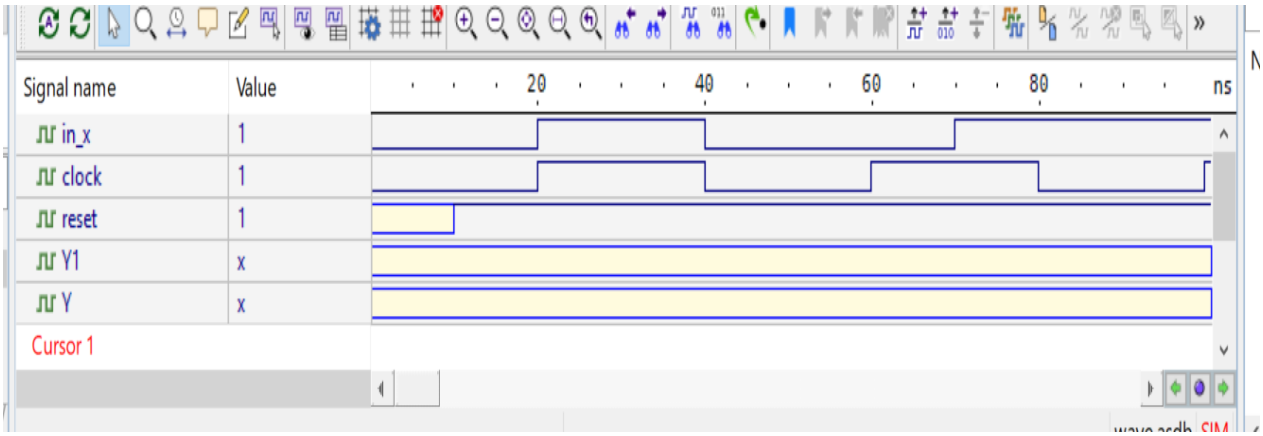

Figure 11: testbench module simulation

```
◦ # Waveform file 'untitled.awc' connected to 'c:/My_Designs/project1210880/proj1210880/src/wave.asdb'.
◦ run 100 ns
◦ # KERNEL: false
◦ # KERNEL: false
◦ # KERNEL: false
◦ # KERNEL: stopped at time: 100 ns
◦ run 100 ns
◦ # KERNEL: false
◦ # KERNEL: false
◦ # KERNEL: stopped at time: 200 ns
◦ run 100 ns
◦ # KERNEL: false
◦ # KERNEL: stopped at time: 300 ns
◦ # KERNEL: Simulation has finished. There are no more test vectors to simulate.
◦ run 100 ns
◦ # KERNEL: stopped at time: 400 ns
◦ # KERNEL: Simulation has finished. There are no more test vectors to simulate.
◦ run 100 ns
>
```

Figure 12: testbench module result

Conclusion and Future work

In conclusion, this project successfully devised and executed a Moore Finite State Machine (FSM) for the precise detection of the "1011" sequence. This accomplishment was achieved through the strategic integration of T-Flip Flops and combinational logic, resulting in a highly efficient sequence detector. The effectiveness of the circuit was rigorously validated using a comprehensive testbench that meticulously compared the actual output to the expected output, reaffirming the circuit's reliability and correctness.

Regarding future enhancements, several promising avenues can be explored to further elevate the capabilities of the sequence detector circuit. Firstly, expanding the bit width of the detector registers can significantly extend its sequence detection capacity, allowing it to recognize more intricate patterns beyond "1011." This expansion would necessitate corresponding adjustments in storage and logic design to accommodate the increased complexity.

Furthermore, the circuit's versatility can be enhanced by enabling it to detect a broader spectrum of sequences or mathematical patterns. This endeavor would entail modifying the combinational logic and state transitions to cater to diverse detection criteria, thereby making the circuit adaptable to a wider range of applications.

Efficiency optimization remains a crucial aspect of future work. Evaluating alternative flip-flop types, such as D flip-flops or JK flip-flops, can offer insights into potential enhancements in terms of power efficiency and speed. Implementing advanced techniques like clock gating can effectively minimize power consumption, making the circuit more energy-efficient.

In summary, the sequence detector circuit, while currently adept at detecting the "1011" sequence, holds substantial potential for further development. By increasing its sequence detection capacity, diversifying its detection capabilities, and optimizing its efficiency, this circuit can evolve into a versatile and powerful tool suitable for a wide array of practical applications.

APPENDIX CODE :

```
 //1210880
//Sec:1
//T flip-flop module
module  T_flipflop(Q,T,clock,reset);
  output Q;
  input T,clock,reset;
  reg Q;

  always@(posedge clock or negedge reset)
    if(~reset)
Q=1'b0;
    else
Q=Q^T;
endmodule

//Baraa Nasar
//1210880
//Sec:1
//T flip-flop Testbench  module
module TFF_Testbench;
  reg clk, reset, T;
  wire Q;

  // Instantiate the T flip-flop module
  T_flipflop dff(Q, T, clk, reset);

  // Clock generation
  always begin
    #5 clk = ~clk;
  end

  // Initial block
  initial begin
    // Initialize signals
    clk = 0;
    reset = 0;
    T = 0;

    // Apply reset
    reset = 1;
```

```
    #10 reset = 0;

    #5 T = 1;
    #10 T = 0;
#5 T = 1;
    #20 T = 0;
#15 T = 1;
    #30 T = 0;

    // End simulation
    $finish;
  end

  // Monitor block to display output
  always @(Q,T,reset) begin

if (reset==1)
   $display ("At time %t, Q = x, reset = %b, T = %b", $time, reset, T);
else

   $display("At time %t, Q =%b, reset = %b, T= %b", $time, Q, reset, T);
   end
endmodule



//Baraa Nasar
//1210880
//Sec:1
//structural circuit using T-Flip Flops
module SequenceDetector (
    input  clock,
    input  reset,
    input  in_x,
    output reg Y1
);
 wire A,B,C,TA,TB,TC;


 or(TA,A,(B&C&in_x)); //TA=A+BCX

 or(TB,(A&~in_x),(B&~C&~in_x),(B&C&in_x),(~B&C&~in_x));
```

```
 or(TC,(A&B),(C^in_x),(B&C&in_x));//TC=AB+C^X+BCX


 T_flipflop G1( A,TA,clock,reset);
 T_flipflop G2( B,TB,clock,reset);
 T_flipflop G3( C,TC,clock,reset);
    assign Y1=A;

endmodule


//Baraa Nasar
//1210880
//Sec:1
//behavioural description of the circuit
module sqd_moore( output reg in_x, output reg clock, output reg reset, output reg Y);

initial begin
  in_x =0;
  clock =0;
   #10ns reset =1;
#10ns in_x=1;
#20ns in_x=0;
#30ns in_x=1;
#40ns in_x=1;
end
initial begin
repeat (11)
#20ns clock=~clock;
end

   parameter S0=3'b000,
             S1=3'b001,
             S2=3'b011,
             S3=3'b010,
             S4=3'b110;

   reg[1:0] Prsent_state,Next_state;

   always @(posedge clock or negedge reset)
     begin
      if(reset==0)
        Prsent_state <= S0;
```

```
   else
 Prsent_state<=Next_state;
      end
      begin

 always @(Prsent_state ,in_x)
   begin
       case (Prsent_state)
          S0:begin
            if(in_x==1)
              Next_state <= S1;
            else
              Next_state <= S0;
            end
          S1:begin
            if(in_x==0)
              Next_state <=S2;
            else
              Next_state <=S1;
            end
          S2:begin
            if(in_x==0)
              Next_state  <=S0;
            else
              Next_state  <=S3;
            end
          S3:begin
            if(in_x==0)
              Next_state <=S2;
            else
              Next_state <=S4;
            end
          S4:begin
            if(in_x==0)
              Next_state <=S2;
            else
              Next_state <=S1;
            end
       endcase
       end
always @(Prsent_state)
begin
case (Prsent_state)
```

```
S0: Y<=0;
S1: Y<=0;
S2: Y<=0;
S3: Y<=0;
S4: Y<=1;
endcase
   end
end

endmodule



//Baraa Nasar
//1210880
//Sec:1
module testanlyser (clock, Y1 , Y);
input clock,Y1 , Y  ;

always @(posedge clock) begin
if (Y1==Y)
        $display("True");
else
        $display("false");
end

endmodule

//Baraa Nasar
//1210880
//Sec:1
module testbench ;

wire in_x,clock,reset;
reg Y1 , Y ;

sqd_moore t1(in_x,clock,reset,Y);

   SequenceDetector t3(clock,reset,in_x,Y1);

testanlyser t2(clock, Y1 , Y);

endmodule
```