PBL#17 Report: Sequence Detector

Name: Goutam Krapa
ID: 2215855

1. Introduction

In this design project PBL17, our goal is to implement a 101-sequence detector. The circuit detects the sequence 101 from the input sequence x, we will be working on Mealy and Moore finite state machine with the signals as described in Table 1. Figure 1 depicts how the output would look for both Mealy and Moore FSM for the given input sequence 1. We first draw the state graph for each machine and then derive a state table based on which we will be coding the states in our Verilog design code. Below figures show State graph, State and transition tables, expected timing diagrams for both Mealy and Moore FSM

This report provides a comprehensive explanation of the design process, design code, test bench code, and key comments.

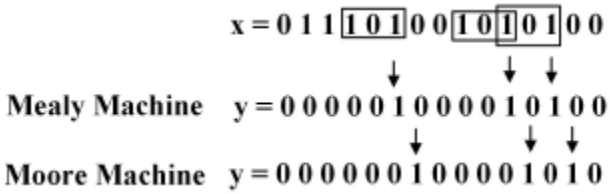

Figure 1: 101 Sequence Detection Utilizing Mealy and Moore FSM

Table 1: Sequence Detector IOs Description

Name	Direction	Bit Width	Description
clk	Input	1	Clock, rising edge trigger
rst	Input	1	Asynchronous reset, 0 valid
x	Input	1	Digit string input
y	Output	1	Output 1 when 101 string detected

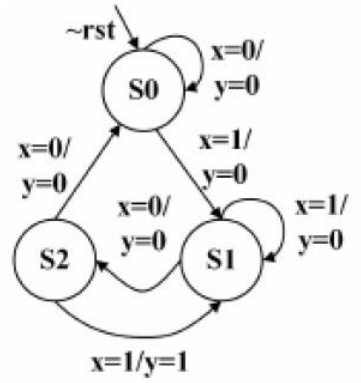

Figure 2: State Diagram for Mealy FSM

cur_state	nxt_state		y	
	x=0	x=1	x=0	x=1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S1	0	1

(a) State Table of the Mealy FSM

q1q0	q1+q0+		y	
	x=0	x=1	x=0	x=1
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

S0=2'b00, S1=2'b01, S2=2'b10

(b) Transition Table of the Mealy FSM

Figure 3: State and Transition Table of the Mealy FSM

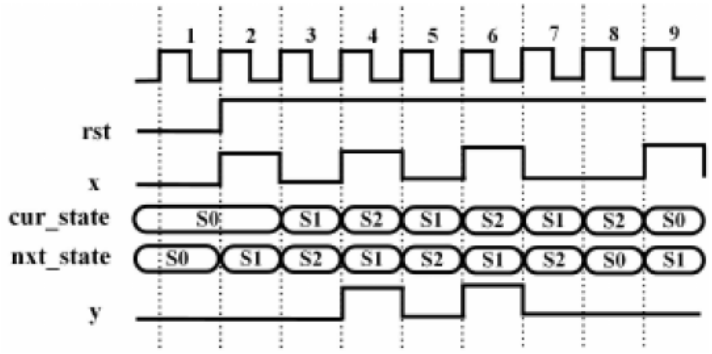

Figure 4: Timing Diagram of Mealy FSM

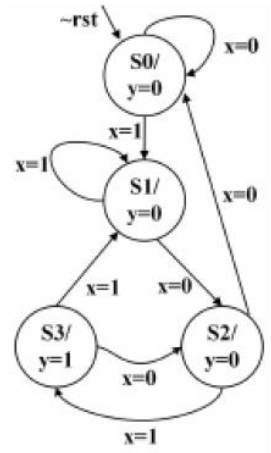

Figure 5: State Graph for Moore FSM

cur_state	nxt_state		y
	x=0	x=1	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S1	1

(a) State Table of the Moore FSM

q1q0	q1+q0+		y
	x=0	x=1	
00	00	01	0
01	10	01	0
10	00	11	0
11	10	01	1

S0=2'b00, S1=2'b01,
S2=2'b10, S3=2'b11

(b) Transition Table of the Moore FSM

Figure 6: State and Transition Tables for Moore FSM.

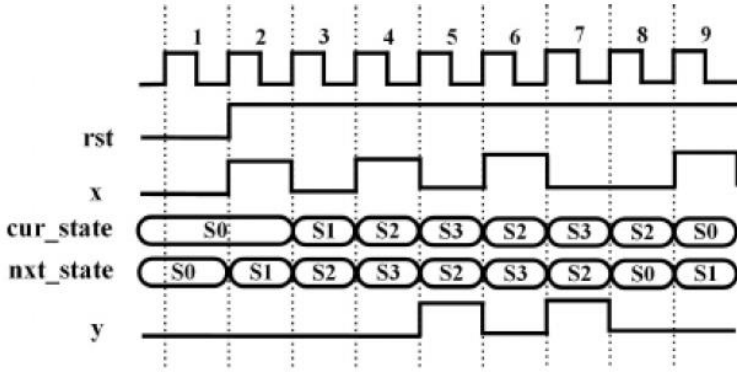

Figure 7:Timing Diagram for Moore FSM

2. Verilog HDL Design Code

2.1. Seq_det_101_mealy.v

```
module seq_det_101_mealy  (
   input      rst,     // Asynchronous reset
   input      clk,     // Clock signal
   input      x,       // Input signal
   output reg y        // Output signal
);

parameter SIZE = 2;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;

reg [SIZE-1:0] cur_state; // Current state register
reg [SIZE-1:0] nxt_state; // Next state register

//------------------State Register -----------------
always @ (posedge clk , negedge rst) begin
   if (~rst)
      cur_state <= S0;
   else
      cur_state <= nxt_state;
end

//-------Next State Combinational Circuit-----------
always @ (cur_state , x , rst) begin
   if (~rst) begin
      nxt_state <= S0;
   end else begin
      case(cur_state)
         S0: if(x) nxt_state <= S1; else nxt_state <= S0;
         S1: if(~x) nxt_state <= S2; else nxt_state <= S1;
         S2: if(x) nxt_state <= S1; else nxt_state <= S0;
         default: nxt_state <= S0;
      endcase
   end
end

//----------Output Combinational Circuit--------------
```

```
assign y = (cur_state==S2) & x;

//or

/* always @ (cur_state or x or rst) begin
   if (~rst) begin
      y <= 0;
   end else begin
      case(cur_state)
         S2: y <= x; // Output 1 when transitioning from S2 to S1 with x=1 (101 detected)
         default: y <= 1'b0;
      endcase
   end
end */

//or

/* always @ (cur_state , x , rst) begin
   if (~rst) begin
      y <= 0;
   end else begin
      case(cur_state)
                           S0: if(x) y <= 1'b0; else y<= 1'b0;
                           S1: if(x) y <= 1'b0; else y<= 1'b0;
         S2: if(x) y <= 1'b1; else y<= 1'b0;
         default: y <= 1'b0;
      endcase
   end
end */

endmodule
```

2.2. seq_det_101_moore.v

```
module seq_det_101_moore (
   input      rst,     // Asynchronous reset
   input      clk,     // Clock signal
   input      x,       // Input signal
   output reg y        // Output signal
);
```

```
parameter SIZE = 2;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

reg [SIZE-1:0] cur_state; // Current state register
reg [SIZE-1:0] nxt_state; // Next state register

//------------------State Register -----------------
always @ (posedge clk , negedge rst) begin
   if (~rst)
      cur_state <= S0;
   else
      cur_state <= nxt_state;
end

//-------Next State Combinational Circuit-----------
always @ (cur_state , x , rst) begin
   if (~rst) begin
      nxt_state <= S0;
   end else begin
      case(cur_state)
         S0: if(x) nxt_state <= S1;
         S1: if(~x) nxt_state <= S2;
         S2: if(x) nxt_state <= S3; else nxt_state <= S0;
         S3: if(x) nxt_state <= S1; else nxt_state <= S2;
         default: nxt_state <= S0;
      endcase
   end
end

/* always @ (cur_state , x , rst) begin
   if (~rst) begin
      nxt_state <= S0;
   end else begin
      case(cur_state)
         S0: if(x) nxt_state <= S1; else nxt_state <= S0;
         S1: if(~x) nxt_state <= S2; else nxt_state <= S1;
         S2: if(x) nxt_state <= S3; else nxt_state <= S0;
         S3: if(x) nxt_state <= S1; else nxt_state <= S2;
         default: nxt_state <= S0;
      endcase
```

```
   end
end */

//----------Output Combinational Circuit--------------

assign y = (cur_state==S3);

endmodule
```

2.3. tb_seq_det_101.v

```
module tb_seq_det_101();

  reg clk;
  reg rst;
  reg x;
  wire y;

        //dut instantiation
        `ifdef MEALY
         seq_det_101_mealy u_seq_det_101_mealy
        `elsif MOORE
         seq_det_101_moore u_seq_det_101_moore
        `endif
                                                                    (.rst(rst),
                                                                     .clk(clk),
                                                                     .x(x),
                                                                     .y(y)      );

        //bus function
        reg [14:0] digit_seq;
        integer i;
        // Clock generation (50 MHz)
        always #5 clk = ~clk;

  initial begin
        rst=1'b0; clk=1'b0; x=1'b0;
        digit_seq = 15'b011_1010_0101_0100;
        #100 rst=1'b1;
```

```
        @(posedge clk);
        for (i=0; i<20;i=i+1) begin
                @(posedge clk);
                x=digit_seq[14];
                digit_seq=digit_seq<<1;
        end
 end

  // Monitoring statement to display signal values
 initial begin
   // Monitor relevant signals: clk, rst, x, cur_state, nxt_state, and y
   `ifdef MEALY
     $monitor("Time: %0t | x: %b | Mealy_y: %b | cur_state: %b | nxt_state: %b | ",
           $time, x, y, u_seq_det_101_mealy.cur_state, u_seq_det_101_mealy.nxt_state);
   `elsif MOORE
     $monitor("Time: %0t | x: %b | Moore_y: %b | cur_state: %b | nxt_state: %b | ",
           $time, x, y, u_seq_det_101_moore.cur_state, u_seq_det_101_moore.nxt_state);
   `endif
 end
endmodule
```

2.4. FileList: filelist_mealy.f

```
../dut/seq_det_101_mealy.v
../tb/tb_seq_det_101.v
```

2.5. FileList: filelist_moore.f

```
../dut/seq_det_101_moore.v
../tb/tb_seq_det_101.v
```

2.6. TCL Script: run_mealy

```
#!/bin/csh -f
##############################################################################
##
# simulation top script by Goutam Krapa 2215855
# change modelsim in this file dir and run "do this_file.do"
# project dir ---+---> hdl source dir
#               +---> sim script dir
#
```

```
##############################################################################
##

# check if current dir has modelsim config file
set has_config  [file exists modelsim.do]
# config modelsim
if {$has_config==1} { do modelsim.do  ; }

echo "+====================================="
echo "| Creat Lib Work soc                  "
echo "+====================================="
vlib work
vmap work work

echo "+====================================="
echo "| Complile RTL Code of soc            "
echo "+====================================="

vlog +define+MEALY\
        -f ../filelist/filelist_mealy.f
# Alternative manual compilation if filelist not available:
#vlog +define+mealy\ -v ../dut/seq_det_101_mealy_b.v  ../tb/tb_seq_det_101.v

echo "+====================================="
echo "| Compiler Pass                       "
echo "| Being to Run Simulation             "
echo "+====================================="
vsim work.tb_seq_det_101 -t 1ns

################## ################### mus
#####################################
add wave -noupdate  tb_seq_det_101/clk
add wave -noupdate  tb_seq_det_101/rst
add wave -noupdate  tb_seq_det_101/x
add wave -noupdate  tb_seq_det_101/y
add wave -noupdate -color blue tb_seq_det_101/u_seq_det_101_mealy/cur_state
add wave -noupdate -color blue tb_seq_det_101/u_seq_det_101_mealy/nxt_state

run 265ns
```

2.7. TCL Script: run_moore

```
#!/bin/csh -f
##############################################################################
##
# simulation top script by Goutam Krapa 2215855
# change modelsim in this file dir and run "do this_file.do"
# project dir ---+---> hdl source dir
#                +---> sim script dir
#
##############################################################################
##

# check if current dir has modelsim config file
set has_config  [file exists modelsim.do]
# config modelsim
if {$has_config==1} { do modelsim.do  ; }

echo "+====================================="
echo "| Creat Lib Work soc                  "
echo "+====================================="
vlib work
vmap work work

echo "+====================================="
echo "| Complile RTL Code of soc            "
echo "+====================================="

vlog +define+MOORE\
          -f ../filelist/filelist_moore.f
#vlog -f ../filelist/filelist.f
# Alternative manual compilation if filelist not available:
##vlog -v ../dut/seq_det_101_moore_b.v  ../tb/tb_seq_det_101.v

echo "+====================================="
echo "| Compiler Pass                       "
echo "| Being to Run Simulation             "
echo "+====================================="
vsim work.tb_seq_det_101 -t 1ns
```

```
################## ################### mus
####################################
add wave -noupdate tb_seq_det_101/clk
add wave -noupdate tb_seq_det_101/rst
add wave -noupdate tb_seq_det_101/x
add wave -noupdate tb_seq_det_101/y
add wave -noupdate -color blue tb_seq_det_101/u_seq_det_101_moore/cur_state
add wave -noupdate -color blue tb_seq_det_101/u_seq_det_101_moore/nxt_state

run 265ns
```

2.8. Project code

Please find the zip file of the code in the attachment.

PBL-Report-GoutamKrapa-2215855-3.zip

3. Simulation Waveform

The below following simulation waveforms are the resultant simulations generated from model sim software a tool used for simulation. We can validate that the output as follows:

- It is working with 100MHz of clock cycles.
- We initialize the clock, reset and input x to 0.
- It is working with asynchronous reset as the sequence is detected when the rst=1.
- We pass the input sequence with x signal that is 011_1010_0101_0100.
- Based on the Mealy and Moore FSM the current state and next state of the machine is decided.
- The next state and current states are defined based on the state graph derived in the pre-design stage of the project. We first theoretical come up with the state graph, state table, and transition table then based on which we designed our file.
- When the required sequence 101 is detected, the output y is 1 for Mealy FSM and whereas for Moore FSM the output y is 1 after the sequence is detected which aligned with the behavior of Moore FSM of output depending on the state.
- The states are coded as S0=00, S1=01, S2=10.

  .

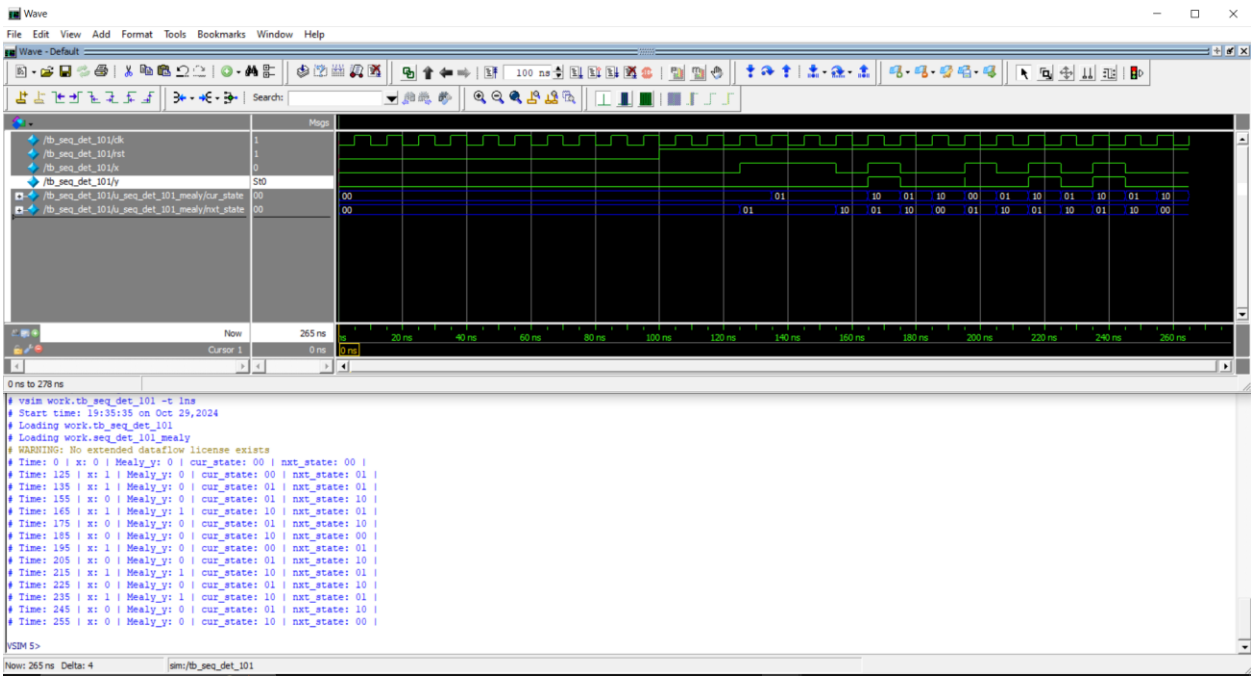

Figure 8: Simulation waveform for the seq_det_101_mealy.v with tb_ seq_det_101.v and run_mealy

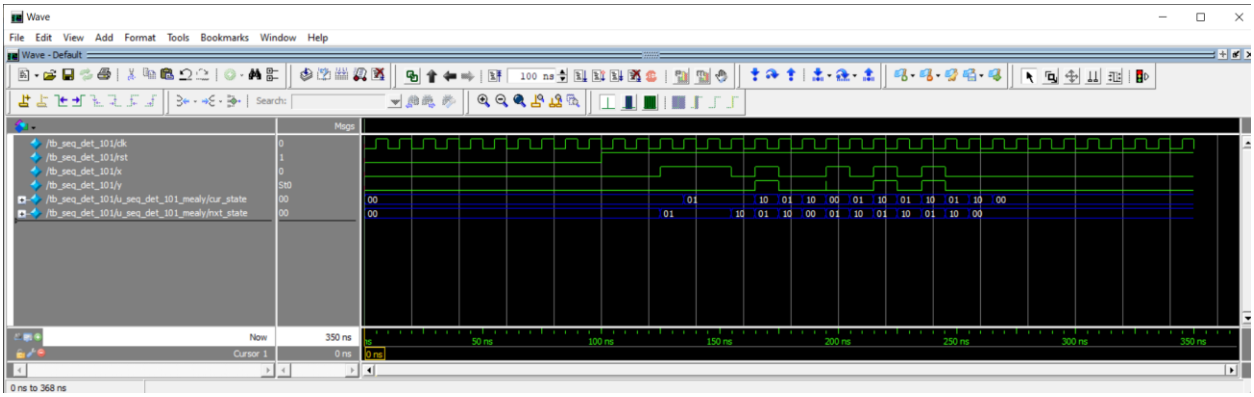

Figure 9: Simulation waveform for the seq_det_101_mealy.v with tb_ seq_det_101.v and run_mealy

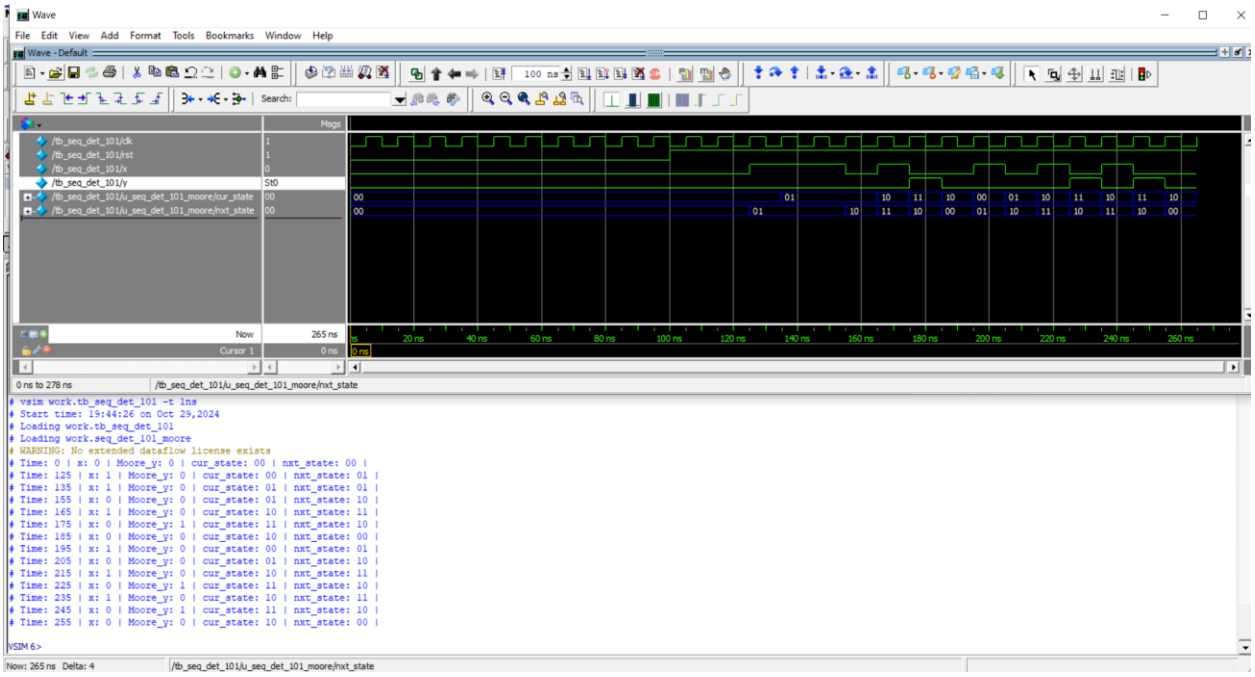

Figure 10: Simulation waveform for the seq_det_101_moore.v with tb_ seq_det_101.v and run_moore

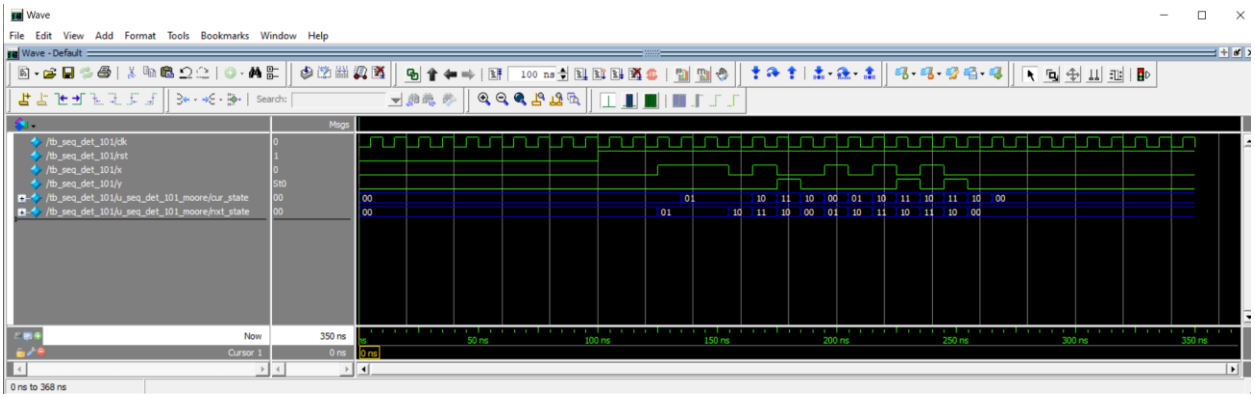

Figure 11: Simulation waveform for the seq_det_101_moore.v with tb_ seq_det_101.v and run_moore

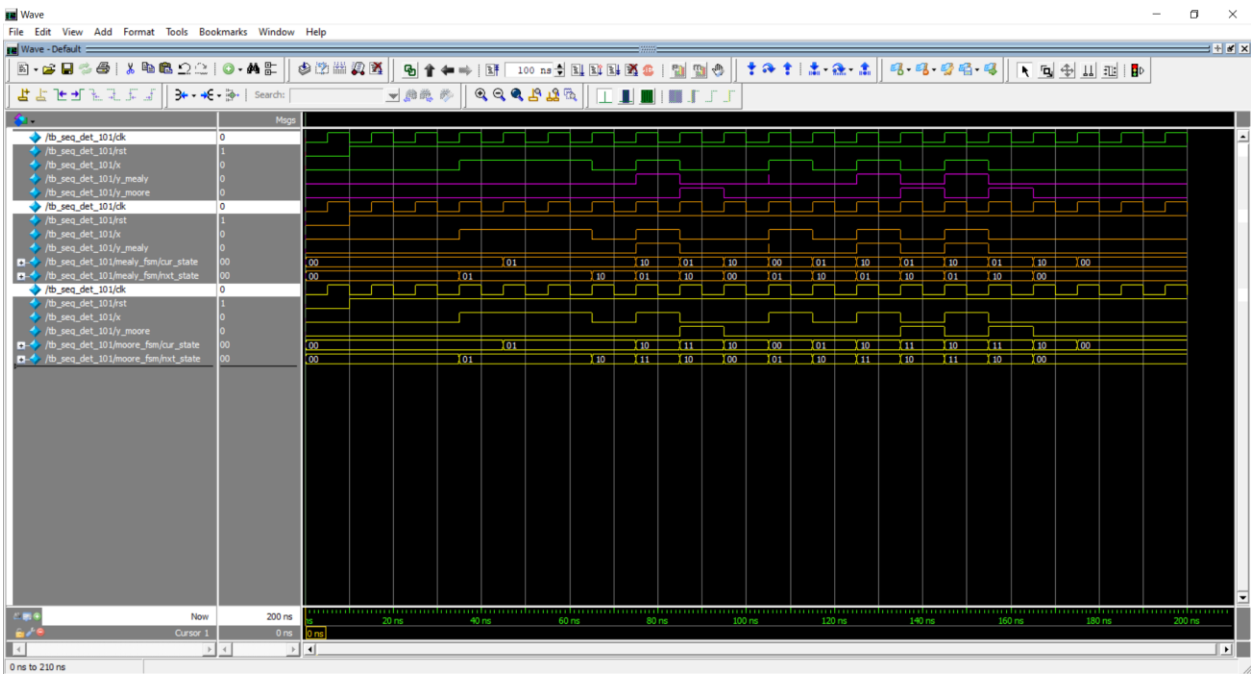

Figure 12: Simulation waveform for the mealy and moore FSM

4. Comments and Conclusion

4.1. Design project

Our project is about sequence detection with help of mealy FSM and Moore FSM. Our Design code Test Bench code and TCL scripts are used to implement Mealy and Moore machine separately. Each FSM is divided into component locks for better design and function, the following blocks are state register to store reset and initiate the states, next state combination circuit to define which state to move to when an input x is given, and output state combination circuit to make the machine output y=1 when the sequence is detected.

4.2. Design Challenges

- **Designing States and transition:** Designing of the FSM such that the state transition also considers overlapping detection was the main challenge. The FSM needed to initiate to an intermediate state after detecting the required sequence "101" rather than returning to the initial state S0.

4.3. Constraint

- **State Encoding:** One of the main constraints was the need of a compact and efficient state encoding so as the usage of available bits (2bits for Mealy, 2bits for Moore based on the number of states required) to represent each state without using additional resources.

4.4. Conclusion

On cross validating the generated simulation waveform we can conclude that with the help of Verilog HDL code we have translated the design specification into logical functional code and through simulation we displayed the simulated waveform. The sequence detector either Mealy or

Moore FSM designed to detect the required sequence 101 and successfully took overlapping sequences also into account and detected them as well. The simulation waveforms show accurate detection and validated the Mealy and Moore finite state machine. For a Mealy FSM the output is asserted immediately after detecting 101 and Moore FSM asserts output y after the required 101 sequence is detected which aligns with the behavior of Mealy and Moore FSM. Thus, we can conclude that the counter is working as expected from the waveform simulation.