CHITTAGONG UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Department of Electronics and Telecommunication Engineering

VLSI Technology Sessional

ETE 404

Experiment No:11

Modeling Sequential Systems and Finite State Machine Using Verilog HDL

Submitted by:

Shamanza Chowdhury

ID: 1908008

Submitted to:

Arif Istiaque Rupom

Lecturer

Dept. of ETE,CUET

November 07, 2024

1 Objectives:

- To verify the function of sequential circuits using Verilog Testbench.
- To model finite state machine and verify its function using Verilog Testbench.

2 Apparatus:

- **Software:**ModelSim-Altera

3 T-Flip Flop Design:

3.1 T-Flip Flop:

The Verilog HDL code of a positive edge-triggered T flip-flop with reset is demonstrated:

Listing 1: T Flip Flop

```
module T_FF(T,clk,reset,Q);
input T,clk,reset;
output reg Q;
always@(posedge clk)
begin
if(reset==0)
begin
if (T)
Q<=~Q;
else
Q<=Q;
end
else
Q<=0;
end
endmodule
```

3.2 T Flip Flop Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the T flip-flop.

Listing 2: T Flip Flop Testbench

```
1
2    `timescale 1ns/100ps
3  module T_FF_TB;
4  reg T,clk,reset;
5  wire Q;
6  T_FF dut(T,clk,reset,Q);
7  initial
8  begin
9  T=0; clk=0; reset=1;
10 end
11 always
12 #2 clk=~clk;
13 initial
14 begin
15 #10 reset=0; T=1;
16 #10 reset=1; T=1;
17 #10 reset=1; T=0;
18 #10 reset=0; T=0;
19 #4 $finish;
20 end
21 endmodule
```

3.3 Behavior Analysis of T Flip Flop:

Simulating the design using ModelSim and analyzing the output waveform:

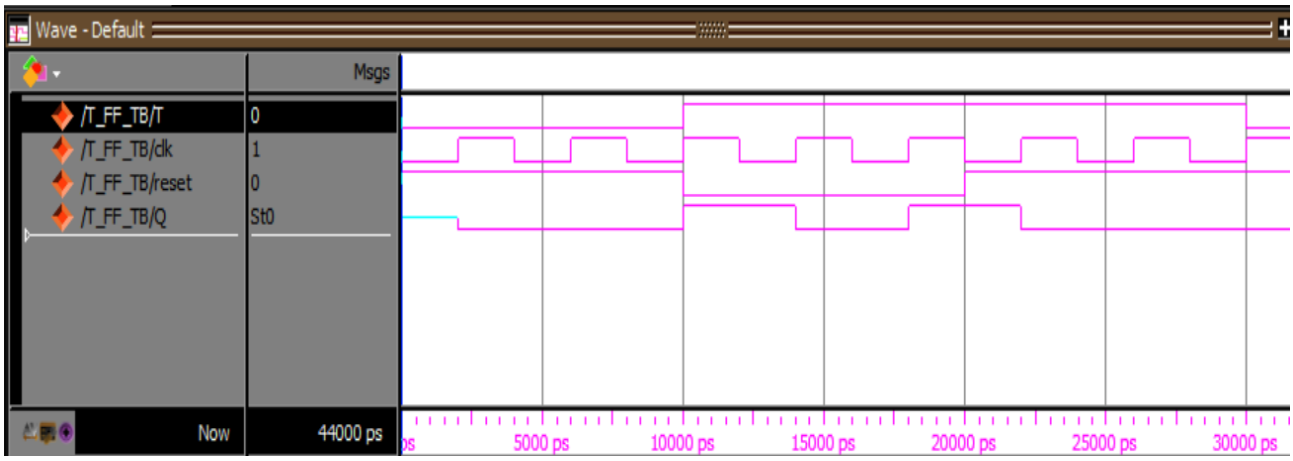

Figure 01:Output waveform of T Flip Flop.

It can be seen from the waveform that the output Q toggles its state on the rising edge of the clock (clk) when T is set to 1. When T is 0, Q retains its previous state. The reset signal properly initializes the output to 0, confirming the correctness of the T flip-flop's operation according to the expected behavior.

4 JK-Flip Flop Design:

4.1 JK-Flip Flop :

The Verilog HDL code of a positive edge-triggered JK flip-flop with clear is demonstrated:

Listing 3: JK Flip Flop

```
1
2  module JK_FF(clk,J,K,Q,clear);
3  input clk,J,K,clear;
4  output reg Q;
5  always@ (posedge clk)
6  begin
7  if(clear==0)
8  begin
9  if (J==0 && K==0)
10 Q<=Q;
11 else if (J==0 && K==1)
12 Q<=0;
13 else if (J==1 && K==0)
14 Q<=1;
```

```
else
Q<=~Q;
end
else
Q=0;
end
endmodule
```

4.2 JK Flip Flop Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the JK flip-flop.

Listing 4: JK Flip Flop Testbench

```
timescale 1ns/1ps
module JK_FF_TB;
reg clk,J,K,clear;
wire Q;
JK_FF JK_dut(clk,J,K,Q,clear);
initial
begin
clk=0; J=0; K=1;clear=0;
end
always
#2 clk=~clk;
initial
begin
#10 clear=0; J=0; K=0;
#10 clear=0; J=0; K=1;
#10 clear=0; J=1; K=0;
#10 clear=0; J=1; K=1;
#10 clear=1; J=0; K=0;
#10 clear=1; J=0; K=1;
#10 clear=1; J=1; K=0;
#10 clear=1; J=1; K=1;
#4 $finish;
end
endmodule
```

4.3 Behavior Analysis of JK Flip Flop:

Simulating the design using ModelSim and analyzing the output waveform:

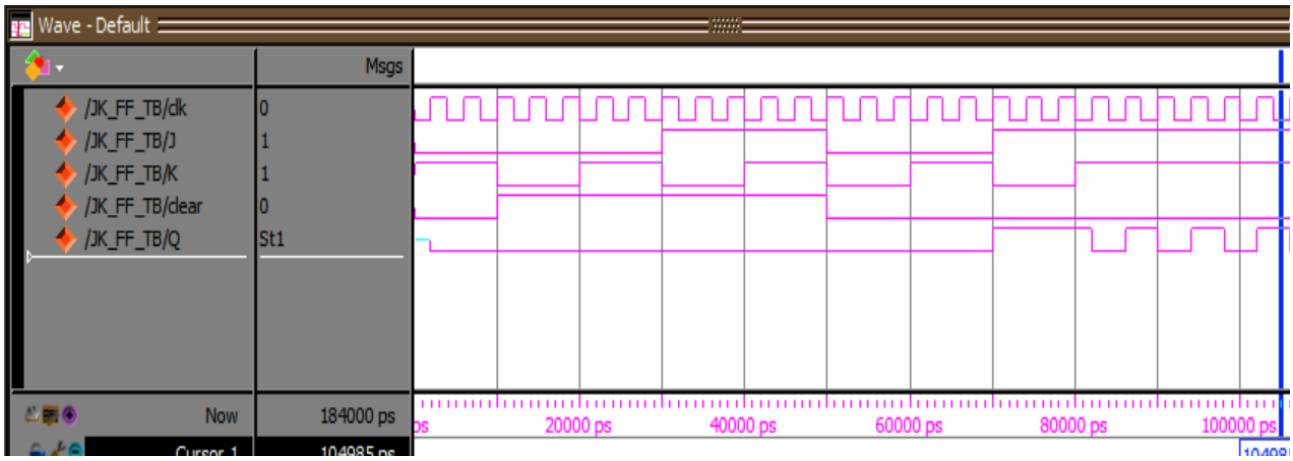

Figure 02:Output waveform of JK Flip Flop.

It can be observed from the waveform that the output Q behaves as expected for a JK flip-flop. When J = 0 and K = 0, Q retains its previous state. When J = 0 and K = 1, Q is reset to 0. When J = 1 and K = 0, Q is set to 1. Finally, when J = 1 and K = 1, Q toggles its state on the rising edge of the clock (clk). The clear signal successfully initializes the output to 0 when active, confirming the proper functionality of the JK flip-flop.

5 4-bit Ripple-Carry Counter Design:

5.1 4-bit Ripple-Carry Counter :

The Verilog HDL code of a 4-bit asynchronous ripple-carry counter is demonstrated:

Listing 5: 4-bit Ripple-Carry Counter

```
module rc_counter(q,clock,reset);
output [3:0] q;
input clock,reset;
t_ff tff0 (q[0], clock, reset);
t_ff tff1 (q[1], q[0], reset);
t_ff tff2 (q[2], q[1], reset);
t_ff tff3 (q[3], q[2], reset);
endmodule
module t_ff (q,clk,r);//T-Flip-Flop
output q;
input clk,r;
wire d;
d_ff dff1(q,d,clk,r);
```

```
not n1(d,q);
endmodule
module d_ff (q,d,clk,r);//D-Flip-Flop
output reg q;
input d,clk,r;
always @(posedge r or negedge clk)
begin
if (r)
q<=1'b0;
else
q<=d;
end
endmodule
```

5.2 4-bit Ripple-Carry Counter Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the 4-bit asynchronous Ripple-Carry Counter .

Listing 6: 4-bit Ripple-Carry Counter Testbench

```
    `timescale 1ns/1ps
module rc_counter1_TB;
reg clk, res;
wire [3:0]q;
rc_counter rc_counter_dut(q,clk,res);
initial
begin
clk=0;
end
always
#5 clk=~clk;
initial
begin
$monitor($time, " clk=%b, res=%b, q=%b", clk, res, q);
res=1;
#10 res=0;
#160 res=1;
```

```
#10 res=0;
#160 $stop;
end
endmodule
```

5.3 Behavior Analysis of Ripple-Carry Counter:

Simulating the design using ModelSim and analyzing the output waveform:

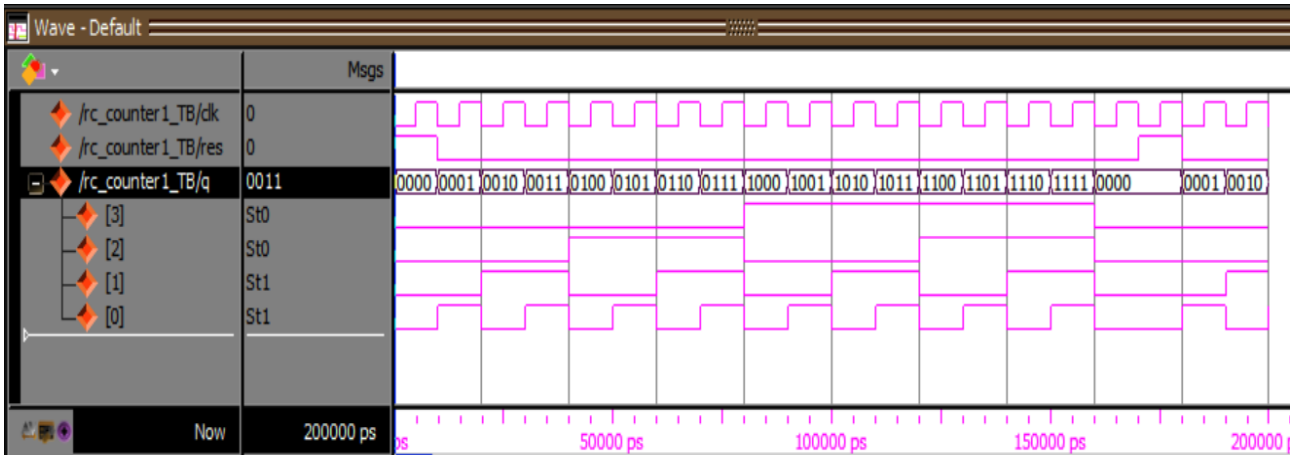

Figure 03:Output waveform of 4-bit Ripple-Carry Counter Flip Flop.

It can be observed from the waveform that each bit toggles on the falling edge of the preceding bit. The least significant bit (Q0) toggles with each clock pulse, while subsequent bits (Q1, Q2, Q3) toggle at half the frequency of their preceding bit, forming a binary counting sequence. The counter increments in binary, starting from 0000, and resets properly to 0000 after reaching 1111, confirming the correct operation of the ripple counter.

6 8-bit Accumulator Design:

6.1 8-bit Accumulator:

The Verilog HDL code of a 8-bit accumulator is demonstrated:

Listing 7: 8-bit Accumulator

```

module accu(in, acc, clk, reset);
input [7:0] in;
input clk, reset;
output reg [7:0]acc;
always @(posedge clk)
```

```
begin
if (reset)
acc<=0;
else
acc<=acc+in;
end
endmodule
```

6.2 8-bit Accumulator Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the 8-bit accumulator .

Listing 8: 8-bit Accumulator Testbench

```
`timescale 1ns/1ps
module accu_TB;
reg [7:0] in;
reg clk,reset;
wire [7:0] out;
accu dut(in, out, clk, reset);
initial
clk = 1'b0;
always
#5 clk = ~clk;
initial
begin
#0 reset<=1; in<=1;
#5 reset<=0;
#500 $finish;
end
endmodule
```

6.3 Behavior Analysis of 8-bit Accumulator:

Simulating the design using ModelSim and analyzing the output waveform:

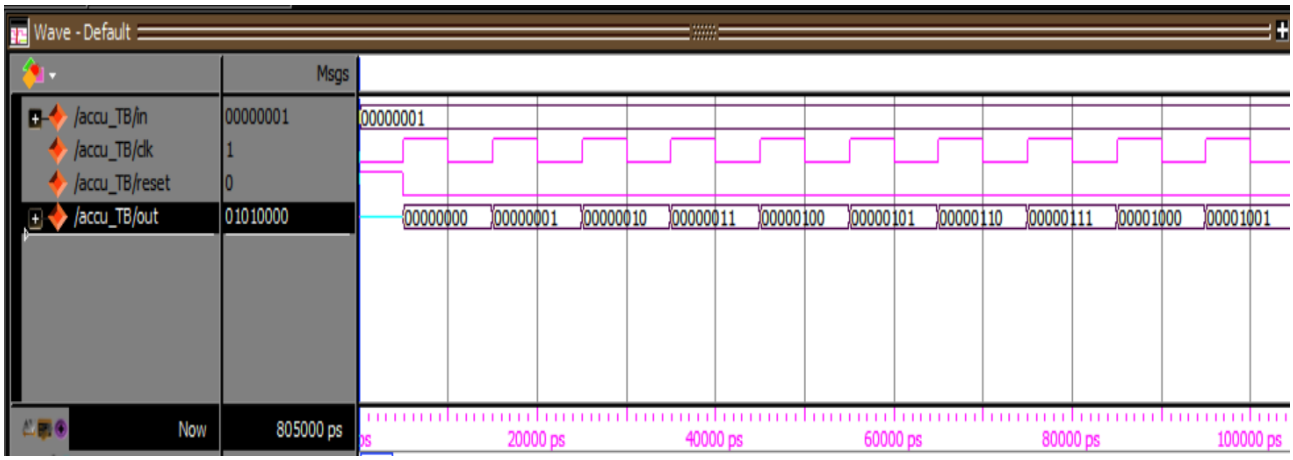

Figure 04:Output waveform of 8-bit accumulator.

It can be observed from the waveform that When the signal is reset(0), the output (out) increments by adding the value of the input (in) on each rising edge of the clock (clk). The pattern shows consistent additions, verifying that the accumulator correctly sums the input with the current output on each clock pulse, demonstrating accurate functionality..

7 4-bit ALU Design:

7.1 4-bit ALU:

The Verilog HDL code of a 4-bit ALU is demonstrated:

Listing 9: 4-bit ALU

```
module alu_4bit(A,B,Y,clk,Opcode);
input [3:0]A,B;
input [1:0]Opcode;
input clk;
output [7:0]Y;
wire [7:0]y1,y2,y3,y4;
arithmetic_unit sm1(A,B,y1,y2);
logical_unit sm2(A,B,y3,y4);
control_unit sm3(y1,y2,y3,y4,clk,Opcode,Y);
endmodule
module arithmetic_unit(x,y,y1,y2);
input [3:0]x,y;
```

```
output reg[7:0]y1,y2;
always@(x,y)
begin
y1<=x+y;
y2<=x-y;
end
endmodule
module logical_unit(x,y,y3,y4);
input [3:0]x,y;
output [7:0]y3,y4;
assign y3=x&y;
assign y4=x^y;
endmodule
module control_unit(y1,y2,y3,y4,clk,Opcode,Y);
input [7:0]y1,y2,y3,y4;
input [1:0]Opcode;
input clk;
output reg[7:0]Y;
always@(posedge clk)
begin
if(Opcode ==2'b00)
Y<=y1;
else if(Opcode ==2'b01)
Y<=y2;
else if(Opcode ==2'b10)
Y<=y3;
else if(Opcode ==2'b11)
Y<=y4;
else
Y<=0;

end
endmodule
```

7.2 4-bit ALU Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the 4-bit ALU .

Listing 10: 4-bit ALU Testbench

```
`timescale 1ns/1ps
module alu_4bit_TB;
reg [3:0]A,B;
reg [1:0]Opcode;
reg clk;
wire [7:0]Y;
alu_4bit dut(A,B,Y,clk,Opcode);
initial
begin
clk = 1'b0; Opcode=2'b00; A=4'b0100; B=4'b1100;
end
always
#2.5 clk = ~clk;
initial
begin
#5 Opcode<=2'b00; A=4'b1001; B=4'b1011;
#5 Opcode<=2'b01; A=4'b0111; B=4'b1110;
#5 Opcode<=2'b10; A=4'b1011; B=4'b1011;
#5 Opcode<=2'b11; A=4'b0101; B=4'b1111;
#5 Opcode<=2'b00; A=4'b1111; B=4'b0110;
#5 Opcode<=2'b01; A=4'b1101; B=4'b1001;
#5 Opcode<=2'b10; A=4'b1011; B=4'b0010;
#5 Opcode<=2'b11; A=4'b1101; B=4'b1100;
#5 $finish;
end
endmodule
```

7.3 Behavior Analysis of 4-bit ALU:

Simulating the design using ModelSim and analyzing the output waveform:

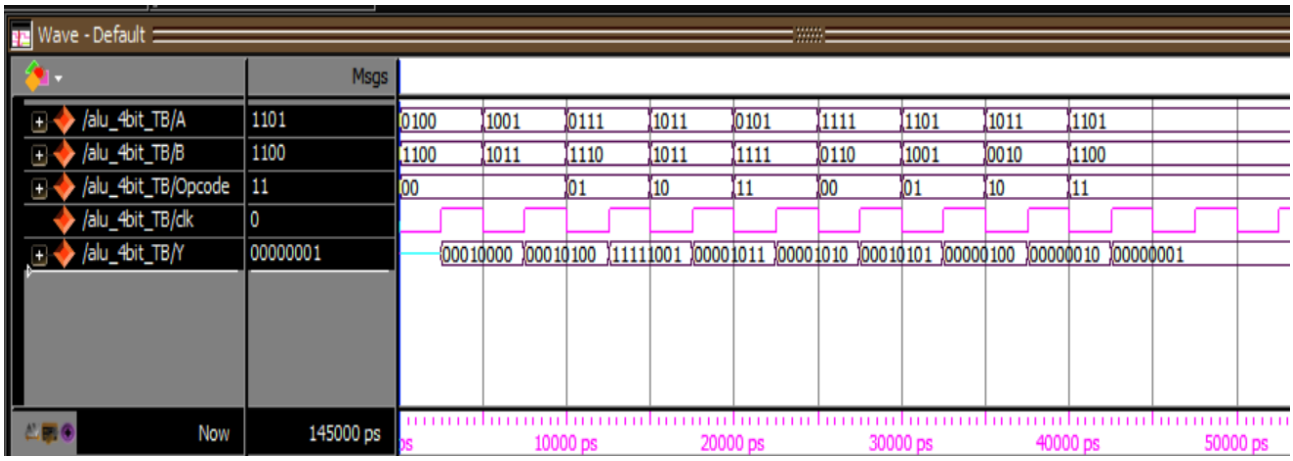

Figure 05:Output waveform of 4-bit ALU.

It can be observed from the waveform that the output (Y) changes according to the opcode provided and the inputs (A and B). Each opcode determines the arithmetic or logical operation performed. For example, when Opcode is 00, the ALU may perform addition, resulting in Y reflecting the sum of A and B. When the Opcode changes to other values such as 01, 10, or 11, Y updates accordingly to match operations like subtraction, bitwise AND, or bitwise OR. The waveform confirms that the ALU responds correctly on the rising edge of the clock (clk), producing the expected output based on the given inputs and opcode.

8 Sequence Detector:

8.1 Sequence Detector:

The Verilog HDL code of a Sequence Detector is demonstrated:

Listing 11: Sequence Detector

```
1
2  module seq_101(i,clk,out);
3  input i,clk;
4  output reg out;
5  localparam S0=2'b00, S1=2'b01, S2=2'b10;
6  reg [1:0]state;
7  always@ (posedge clk)
8  begin
9  case (state)
10 S0: begin
```

```
out<=i?0:0;
state<=i?S1:S0;
end
S1: begin
out<=i?0:0;
state<=i?S1:S2;
end
S2: begin
out<=i?1:0;
state<=i?S0:S0;
end
default:
begin
out<=0;
state<=S0;
end
endcase
end
endmodule
```

8.2 Sequence Detector Testbench :

The following Verilog HDL code demonstrates the Testbench Module of the Sequence Detector .

Listing 12: Sequence Detector Testbench

```
`timescale 1ns/1ps
module seq_101_TB;
  reg i, clk;
  wire out;
  seq_101 dut(i, clk, out);

  initial clk = 0;

  always #2 clk = ~clk;
  initial begin
    $monitor($time, " clk=%b, i=%b, out=%b", clk, i, out);

```

```
    // Initialize input
    #0 i = 0;

    // Sequence to test transitions and detect 101
    #4 i = 1; // Move to S1
    #4 i = 0; // Move to S2 (input '10')
    #4 i = 1; // Detects '101' and should output 1, moves back to S0
    #4 i = 0; // Remain in S0
    #4 i = 1; // Move to S1 again
    #4 i = 0; // Move to S2
    #4 i = 1; // Detects '101' again, output should be 1, back to S0
    #4 i = 1; // Stay in S1
    #4 i = 1; // Remain in S1 with repeated 1
    #4 i = 0; // Move to S2 (input '10')
    #4 i = 1; // Detects '101' for a third time, output 1
    #4 i = 0; // Back to S0
    // Finish simulation
    #4 $finish;
  end
endmodule
```

8.3 Behavior Analysis of Sequence Detector:

Simulating the design using ModelSim and analyzing the output waveform:

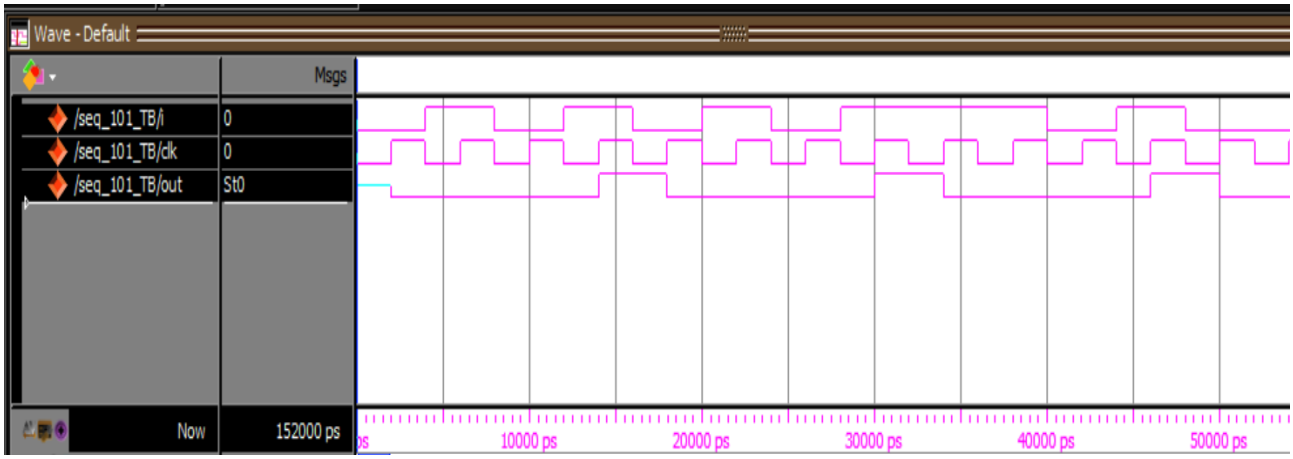

Figure 06:Output waveform of Sequence Detector.

It can be observed from the waveform that the Mealy machine generates an output of '1' when the sequence '101' is detected in the input bitstream. The output transitions to '1' at the same clock edge when the last bit of

the sequence is detected, confirming the correct implementation of the Mealy machine for detecting the '101' sequence.

9 Discussion:

- The experiment analyzed and implemented digital logic design concepts using Verilog.
- Sequential components like flip-flops, counters, and arithmetic units were simulated to verify functionality.
- T-flip flops and JK-flip flops were implemented to store and toggle binary states.
- A 4-bit ripple carry counter demonstrated sequential counting but revealed timing delays due to ripple effects.
- An 8-bit accumulator and a 4-bit ALU performed arithmetic operations. The accumulator handled repeated additions, and the ALU executed addition, subtraction, and logic operations.
- A sequence detector was tested to identify specific bit patterns(101) using a Mealy state machine. .
- The designs were effective and accurately implemented.
- The experiments verified theoretical concepts and practical applications in signal processing, communication, and computation.