CHITTAGONG UNIVERSITY OF ENGINEERING & TECHNOLOGY

Department of Electronics And Telecommunication Engineering

Lab Report

Experiment Name: Modeling Sequential Systems and Finite State Machine Using Verilog HDL

Experiment No.: 11

Course Title: VLSI technology Sessional

Course No.: ETE 404

Date of Experiment: 20-10-2024

Date of Submission: 7-11-2024

Submitted By	Submitted To
Name: M.I. Yasir Arafat **ID:** 1908031 **Level:** 4 **Term:** I	Arif Istiaque Lecturer Dept. of ETE,CUET

Objective:

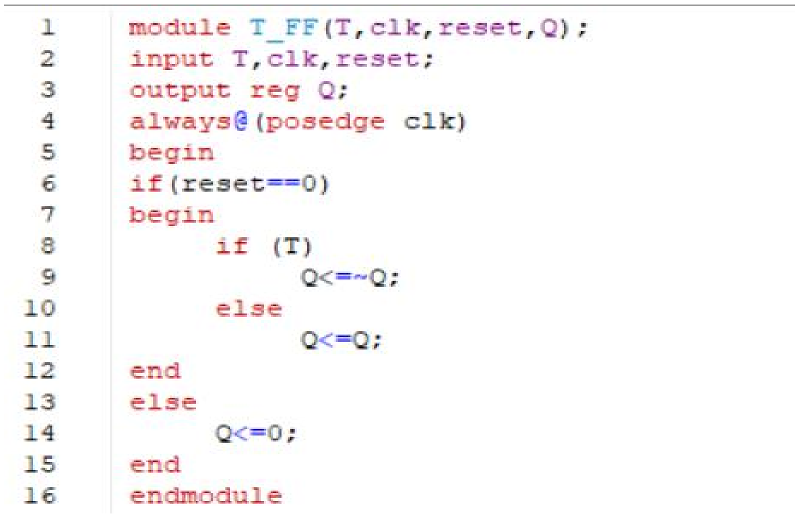
1. Functional verification of sequential circuits using Verilog Testbench.
2. Modeling a finite state machine and its functional verification using Verilog Testbench.

Required Component:

Model Sim ALTERA

Example 01: T-flip flop

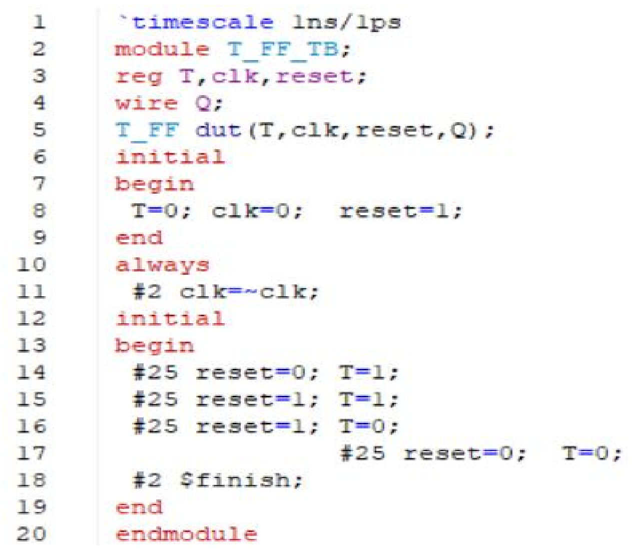
Verilog Code:



```
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
```

Figure 01: Verilog code for T-FF

Test Bench code:



```
`timescale 1ns/1ps
module T_FF_TB;
reg T,clk,reset;
wire Q;
T_FF dut(T,clk,reset,Q);
initial
begin
 T=0; clk=0;  reset=1;
end
always
 #2 clk=~clk;
initial
begin
 #25 reset=0; T=1;
 #25 reset=1; T=1;
 #25 reset=1; T=0;
              #25 reset=0;  T=0;
 #2 $finish;
end
endmodule
```

Figure 02: Testbench code for T -FF

Output of T-FF :

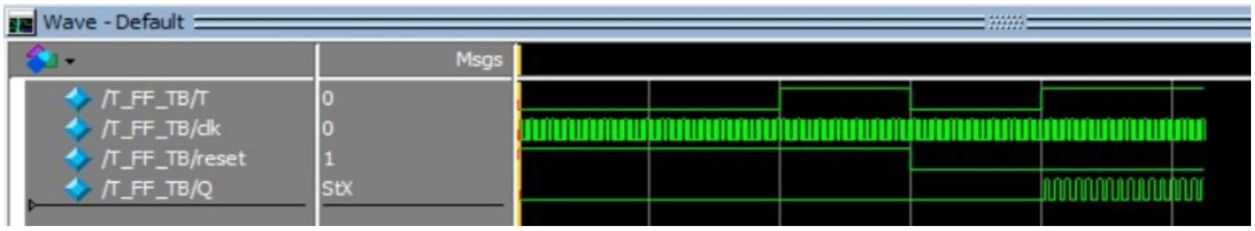
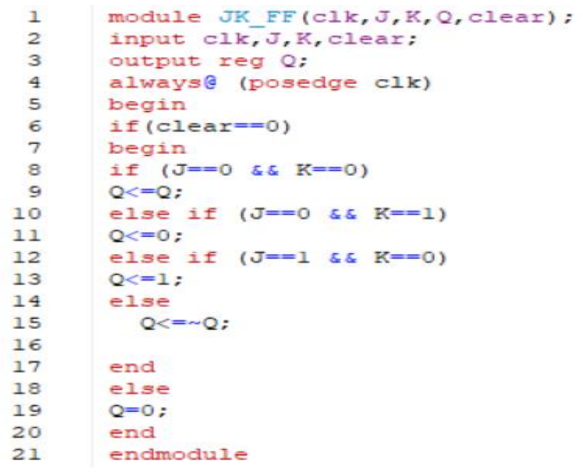


Figure 03: Output of T -FF

Example 02: JK- Flip Flop

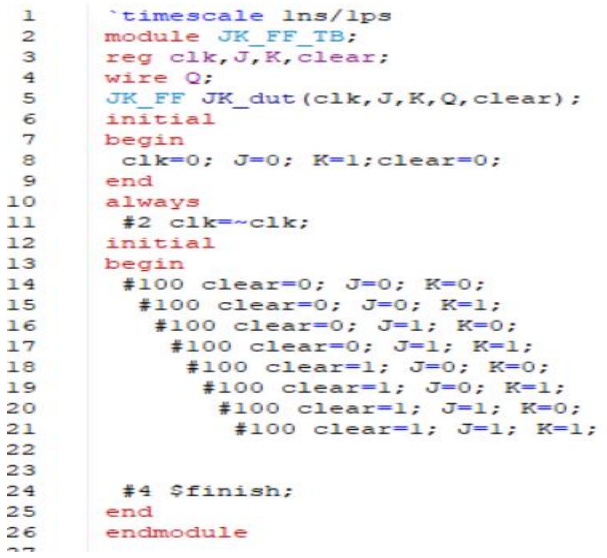
Verilog code:



```
module JK_FF(clk,J,K,Q,clear);
input clk,J,K,clear;
output reg Q;
always@ (posedge clk)
begin
if(clear==0)
begin
if (J==0 && K==0)
Q<=Q;
else if (J==0 && K==1)
Q<=0;
else if (J==1 && K==0)
Q<=1;
else
  Q<=~Q;

end
else
Q=0;
end
endmodule
```

Figure 04: Verilog code for Jk_flip-flop

Testbench code:



```
`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
 #100 clear=0; J=0; K=0;
  #100 clear=0; J=0; K=1;
   #100 clear=0; J=1; K=0;
    #100 clear=0; J=1; K=1;
     #100 clear=1; J=0; K=0;
      #100 clear=1; J=0; K=1;
       #100 clear=1; J=1; K=0;
        #100 clear=1; J=1; K=1;


 #4 $finish;
end
endmodule
```

Figure 05: Verilog code of Jk_flip-flop

Output :

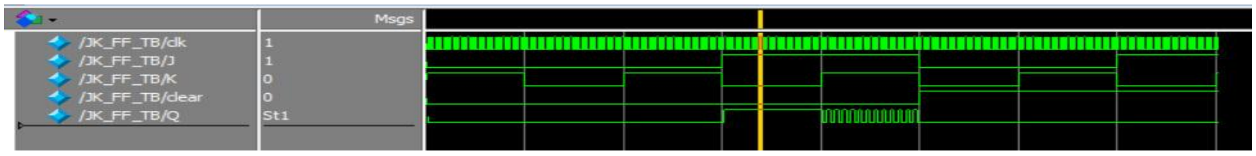
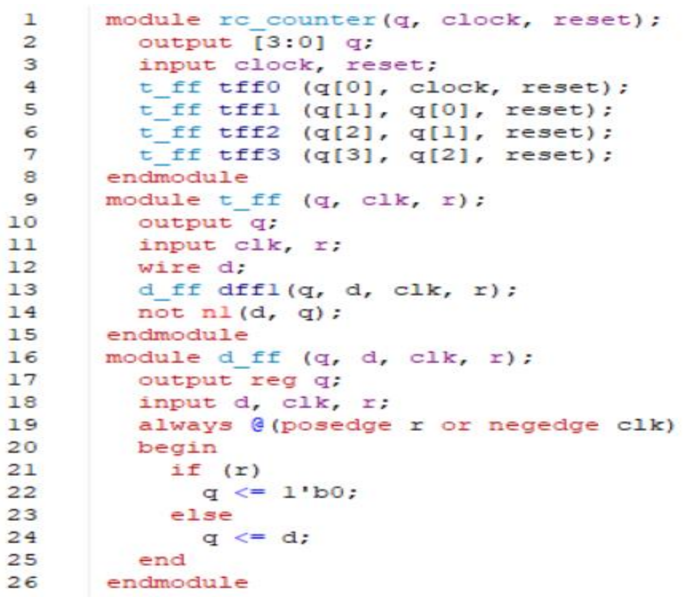


Figure 06: Output of Jk_flip-flop

Example 03: 4-bit ripple carry counter

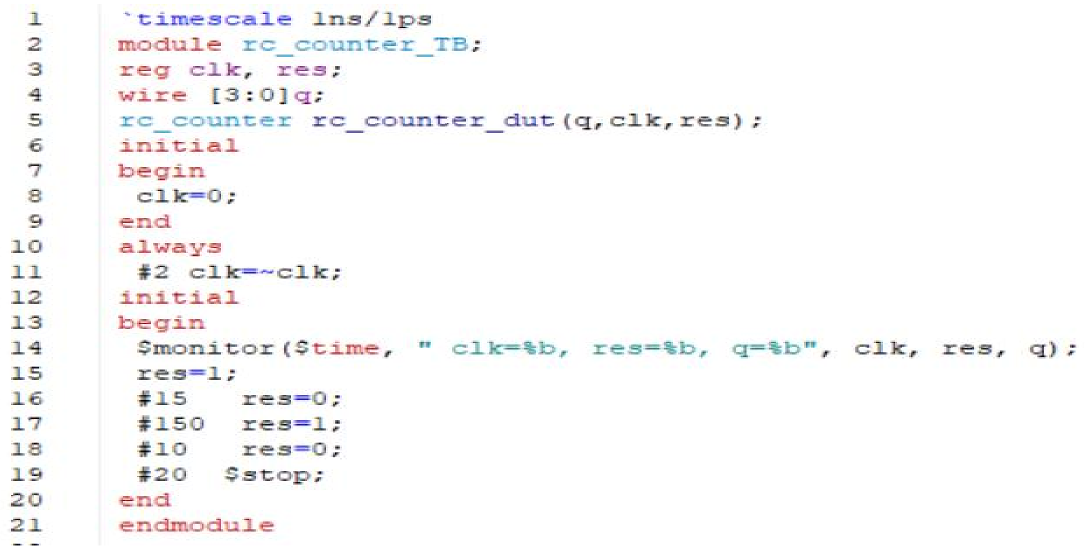
Verilog Code:



```
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);
  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);
  output reg q;
  input d, clk, r;
  always @(posedge r or negedge clk)
  begin
    if (r)
      q <= 1'b0;
    else
      q <= d;
  end
endmodule
```

Figure 07: Verilog code of 4-bit ripple carry counter

Testbench code:



```
`timescale 1ns/1ps
module rc_counter_TB;
reg clk, res;
wire [3:0]q;
rc_counter rc_counter_dut(q,clk,res);
initial
begin
 clk=0;
end
always
 #2 clk=~clk;
initial
begin
 $monitor($time, " clk=%b, res=%b, q=%b", clk, res, q);
 res=1;
 #15   res=0;
 #150  res=1;
 #10   res=0;
 #20  $stop;
end
endmodule
```

Figure 08: Testbench code of 4-bit ripple carry counter

Output:

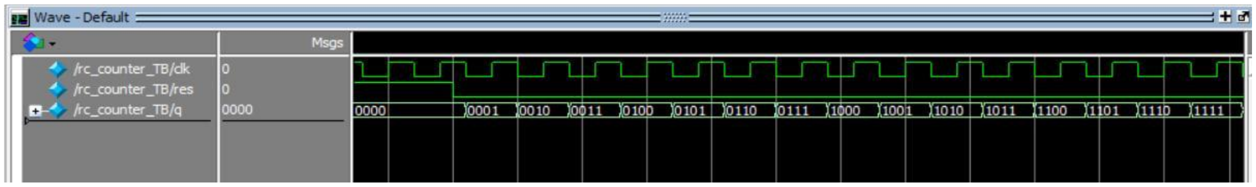
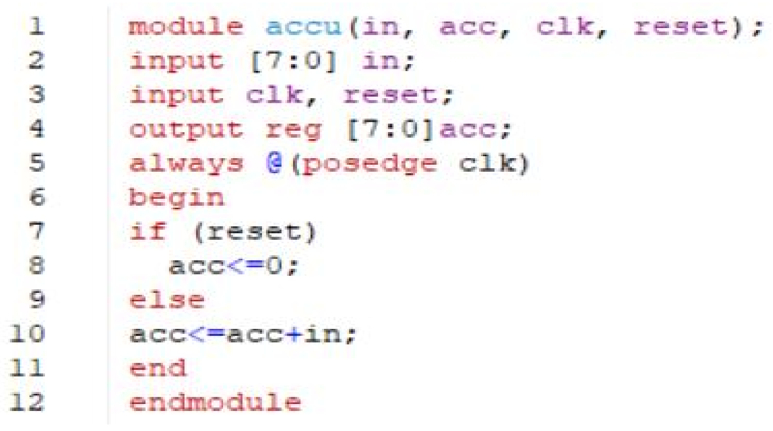


Figure 09: Output of 4-bit ripple carry counter

Example 04: Accumulator

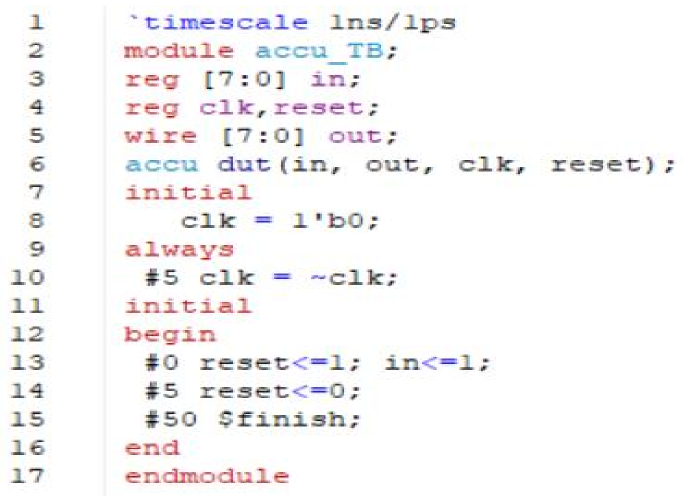
Verilog code:



```
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
```

Figure 09: 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;
 #50 $finish;
end
endmodule
```

Figure 10: Testbench of 8-bit Accumulator

Output:

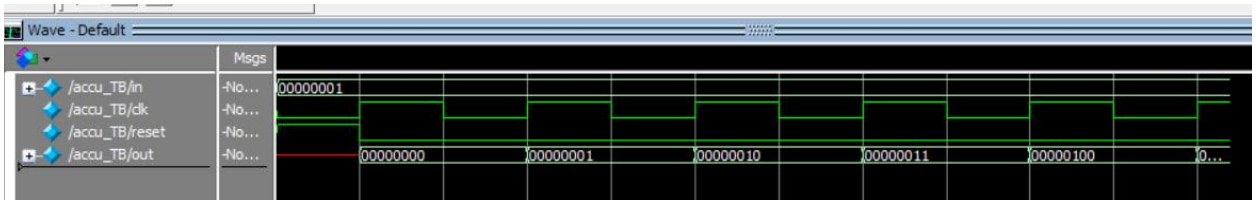
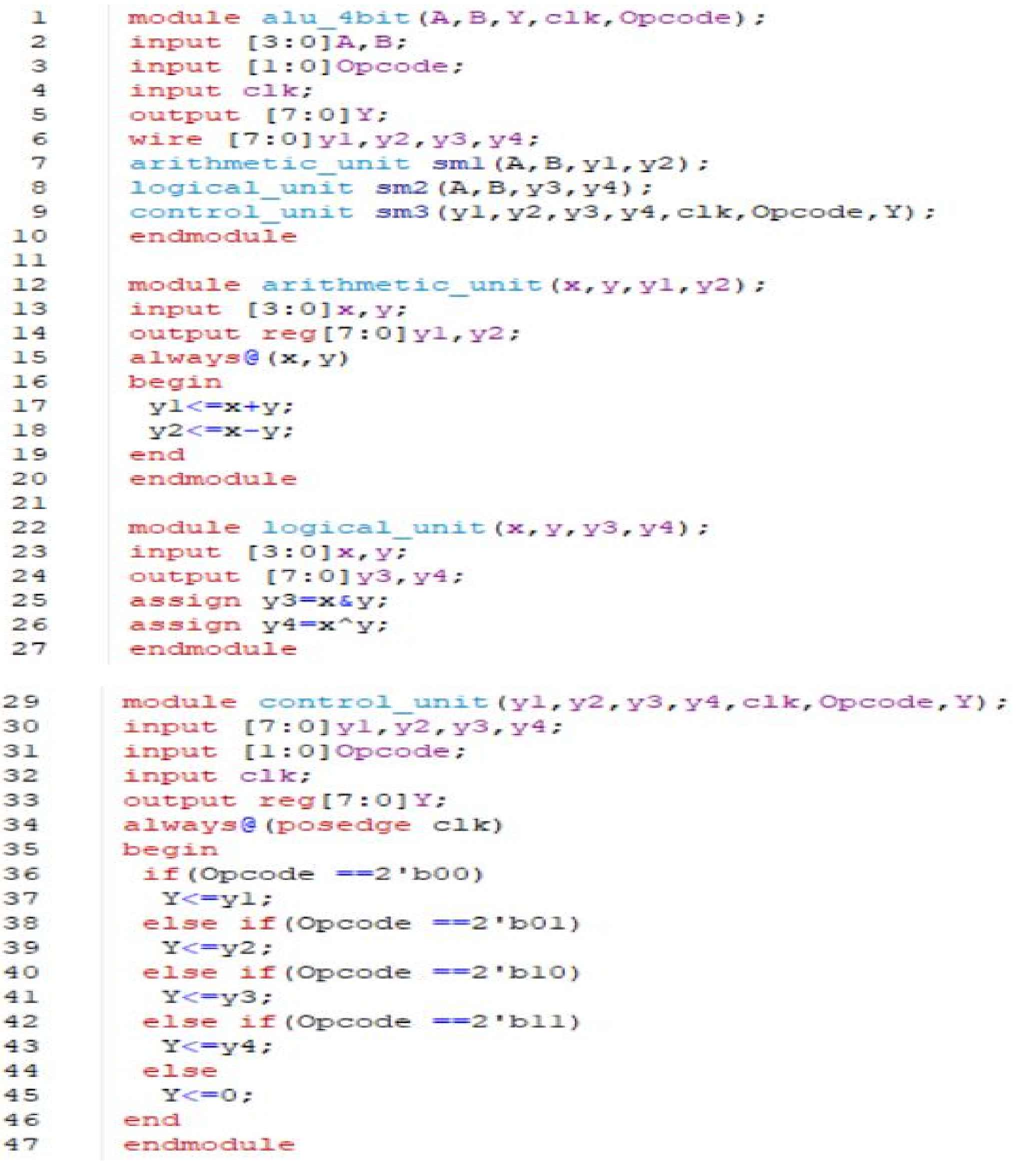


Figure 10: Output of 8-bit Accumulator

Example 05: 4- Bit ALU

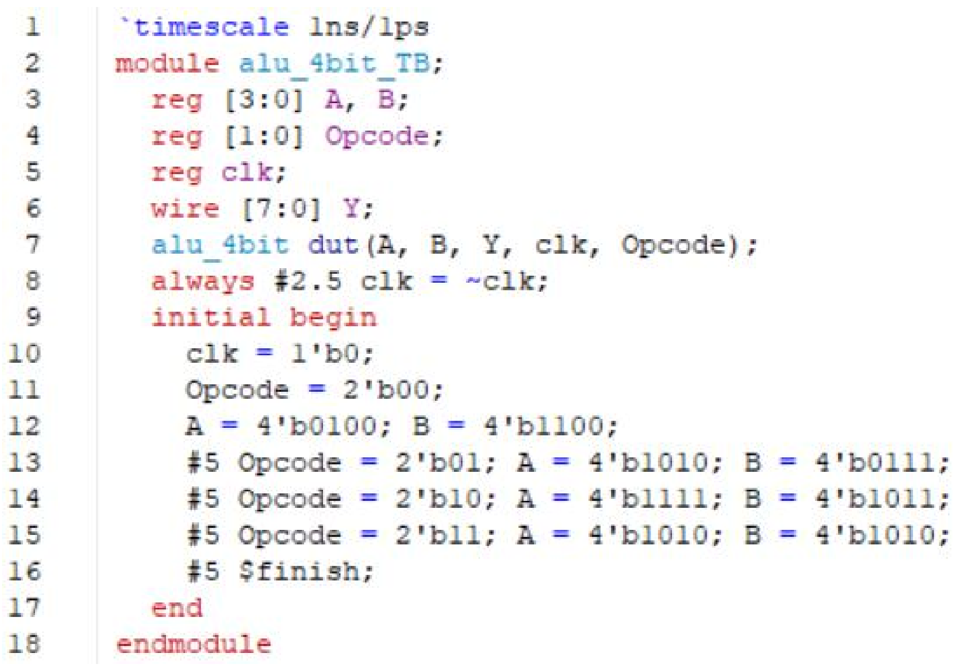
Verilog code:



```
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
```

Figure 11: Verilog code for 4-bit ALU

Testbench code:



```
`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);
  always #2.5 clk = ~clk;
  initial begin
    clk = 1'b0;
    Opcode = 2'b00;
    A = 4'b0100; B = 4'b1100;
    #5 Opcode = 2'b01; A = 4'b1010; B = 4'b0111;
    #5 Opcode = 2'b10; A = 4'b1111; B = 4'b1011;
    #5 Opcode = 2'b11; A = 4'b1010; B = 4'b1010;
    #5 $finish;
  end
endmodule
```

Figure 12: Testbench code for 4-bit ALU

Output:

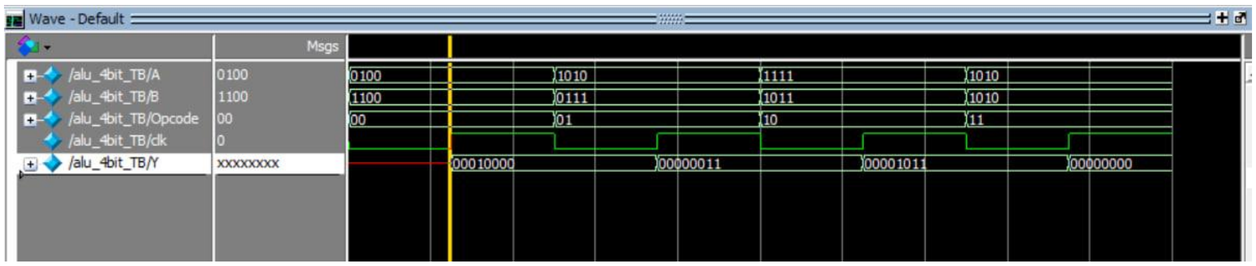
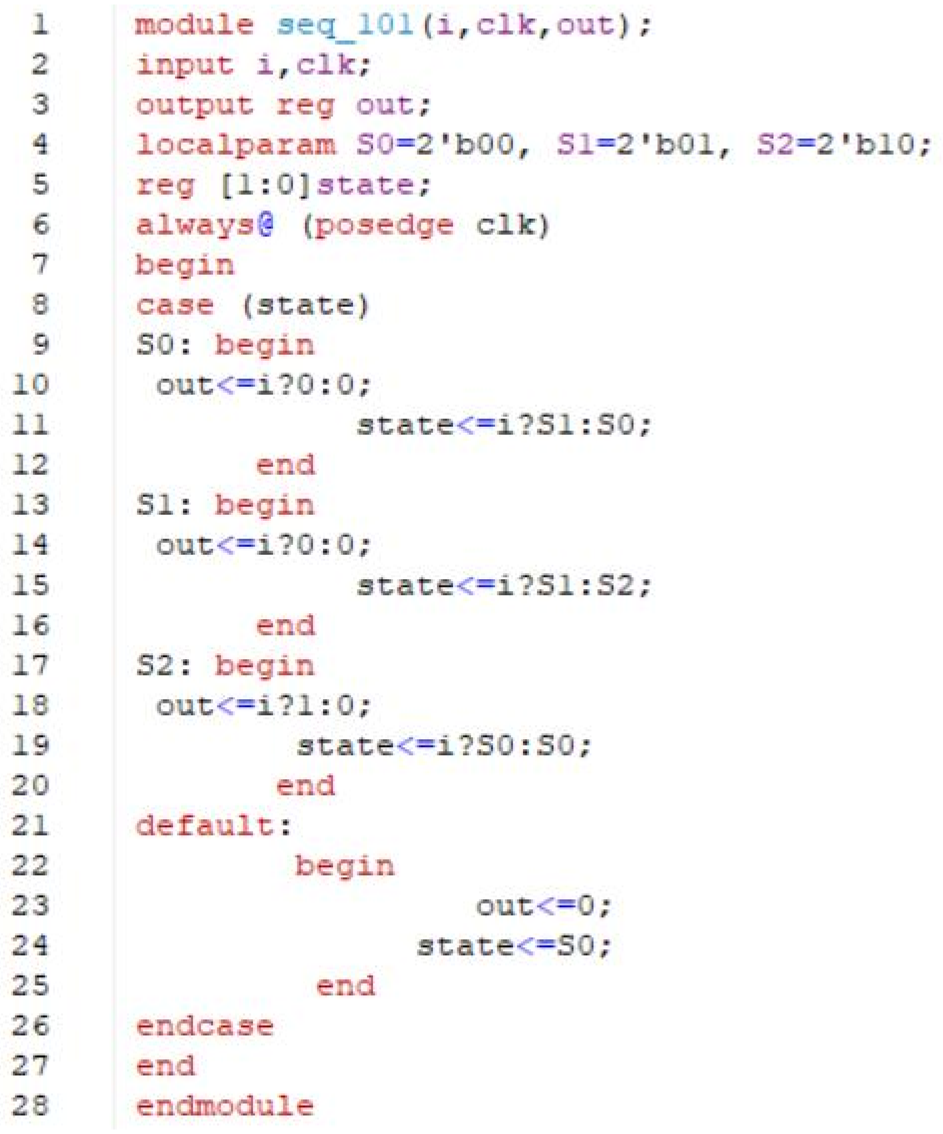


Figure 13: output for 4-bit ALU

Example 06 : Output ‘1’ when sequence 101 is detected

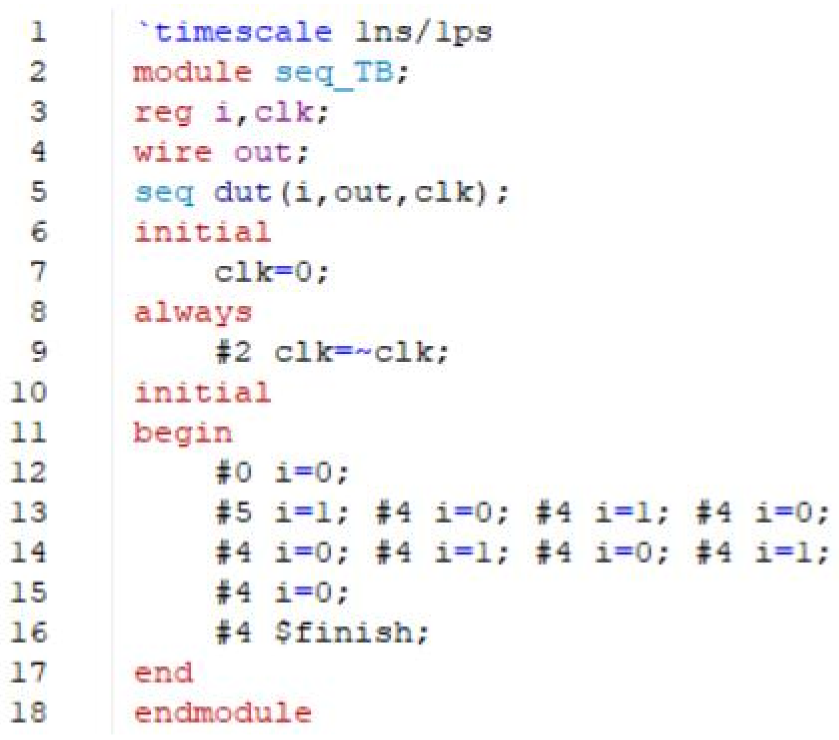
Verilog Code:



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

Figure 14: Verilog code for output ‘1’ when sequence 101 is detected

Testbench Code:



```
`timescale 1ns/1ps
module seq_TB;
reg i,clk;
wire out;
seq dut(i,out,clk);
initial
    clk=0;
always
    #2 clk=~clk;
initial
begin
    #0 i=0;
    #5 i=1; #4 i=0; #4 i=1; #4 i=0;
    #4 i=0; #4 i=1; #4 i=0; #4 i=1;
    #4 i=0;
    #4 $finish;
end
endmodule
```

Figure 15: Testbench code for output ‘1’ when sequence 101 is detected

Output:

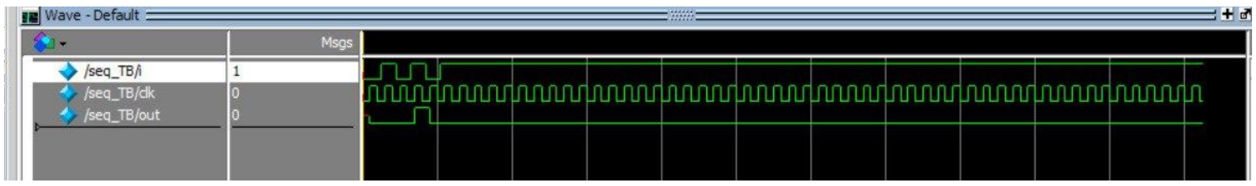


Figure 16: Output ‘1’ when sequence 101

Discussion:

1.The examples provided demonstrate fundamental digital logic design using Verilog, covering memory elements like flip-flops and moving towards more complex components such as counters and ALUs.
2.Flip-flops like T-flip flop and JK-flip flop are essential for data storage, acting as building blocks in sequential circuits by retaining binary states based on clock inputs.
3.The 4-bit ripple carry counter showcases a basic counting mechanism, illustrating how flip-flops can be connected to perform sequential counting with the ripple carry effect impacting timing.
4.The 8-bit accumulator and 4-bit ALU highlight arithmetic capabilities, with the accumulator suited for repetitive additions and the ALU performing a range of operations essential for computational tasks.
5. The last example, sequence detection, demonstrates the use of state machines, which are essential for identifying particular bit patterns and opening up applications in fields like signal processing and communication.