Lab-3: Modeling Sequential Systems and Finite State Machine Using Verilog HDL

Objective

The main objectives of this lab are:

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

Introduction

A digital system can be either in the form of combinational logic or sequential logic. In combinational logic, the output of a circuit depends only on the presently applied inputs. On the other hand, the output of a sequential circuit depends on the applied input and the present states. Most practical digital systems are sequential. To design a digital system, the behavioral abstraction is used as a reference to create and refine a synthesizable register transfer level (RTL) abstraction that captures the desired functionality required by the design specification.

Example 01

Flip-flops are the building blocks of sequential circuits. In the following example, the Verilog HDL code of a positive edge-triggered T flip-flop with reset is demonstrated.

```
1  module T_FF(T,clk,reset,Q);
2  input T,clk,reset;
3  output reg Q;
4  always@(posedge clk)
5  begin
6  if(reset==0)
7  begin
8      if (T)
9          Q<=~Q;
10     else
11         Q<=Q;
12 end
13 else
14     Q<=0;
15 end
16 endmodule
```

Testbench Module of Example 01

The following Verilog HDL code demonstrates the **Testbench Module** of the T flip-flop of Example 01.

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

Example 02

The following example demonstrates the Verilog HDL code of a positive edge-triggered JK flip-flop with clear.

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

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

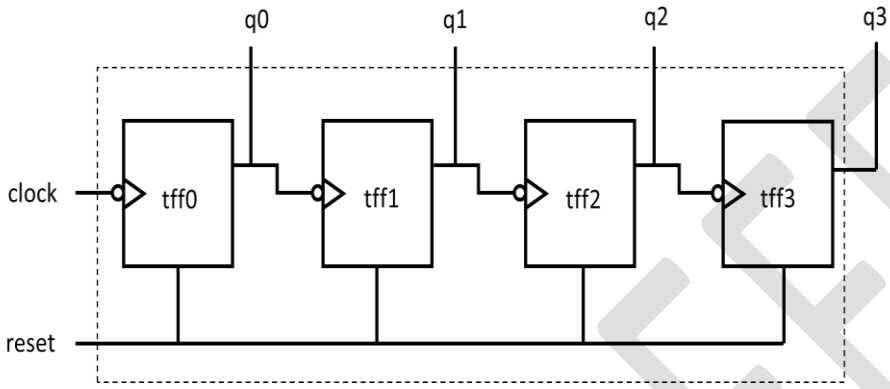
Testbench Module of Example 02

The following Verilog HDL code demonstrates the **Testbench Module** of the JK flip-flop of Example 02.

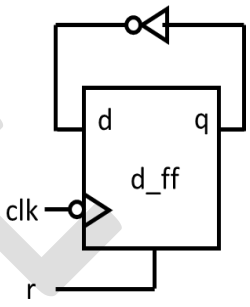
```
`timescale 1ns/1ps
module JK_FF_TB;
reg clk,J,K,clear;
wire Q;
JK_FF JK_dut(clk,J,K,Q,clear);
initial
begin
        clk=0; J=0; K=1;clear=0;
end
always
        #2 clk=~clk;
initial
begin
        #2 clear=1; J=1; K=0;
        #4 clear=0; J=0; K=1;
        #4 J=1;
         #4 J=0;
        #4 $finish;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

Example 03

In this example a 4-bit ripple carry counter will be designed using the submodule of a T flip-flop and each T filp-flop is designed using leaf module of D flip-flop. The block representation of the ripple carry counter is shown below.



4-bit ripple carry counter



T flip flop using D flip flop

The following example demonstrates the Verilog HDL code of a 4-bit asynchronous ripple-carry country as shown in the following block diagram.

```
1  module rc_counter(q,clock,reset);
2  output [3:0] q;
3  input clock,reset;
4  t_ff tff0 (q[0], clock, reset);
5  t_ff tff1 (q[1], q[0], reset);
6  t_ff tff2 (q[2], q[1], reset);
7  t_ff tff3 (q[3], q[2], reset);
8  endmodule
9
10 module t_ff (q,clk,r);//T-Flip-Flop
11 output q;
12 input clk,r;
13 wire d;
14 d_ff dff1(q,d,clk,r);
15 not n1(d,q);
16 endmodule
17
18 module d_ff (q,d,clk,r);//D-Flip-Flop
19 output reg q;
20 input d,clk,r;
21 always @(posedge r or negedge clk)
22 begin
23         if (r)
24                 q<=1'b0;
```

```
25          else
26                  q<=d;
27  end
28  endmodule
```

Testbench Module of Example 03

The following Verilog HDL code demonstrates the **Testbench Module** of the 4-bit asynchronous ripple-carry counter of Example 03.

```
1   `timescale 1ns/1ps
2   module rc_counter_TB;
3   reg clk, res;
4   wire [3:0]q;
5   rc_counter rc_counter_dut(q,clk,res);
6   initial
7   begin
8           clk=0;
9   end
10  always
11          #5 clk=~clk;
12  initial
13  begin
14          $monitor($time, " clk=%b, res=%b, q=%b", clk, res, q);
15          res=1;
16          #15   res=0;
17          #180  res=1;
18          #10   res=0;
19          #20  $stop;
20  end
21  endmodule
```

Example 04

The following example demonstrates the Verilog HDL code of a simple 8-bit accumulator. The module is designed in such a way that when reset=0 the output is set to 0 and when reset=1 the output adds the input.

```
1   module accu(in, acc, clk, reset);
2   input [7:0] in;
3   input clk, reset;
4   output reg [7:0]acc;
5   always @(posedge clk)
6   begin
7       if (reset)
```

```
8                   acc<=0;
9           else
10                  acc<=acc+in;
11  end
12  endmodule
```

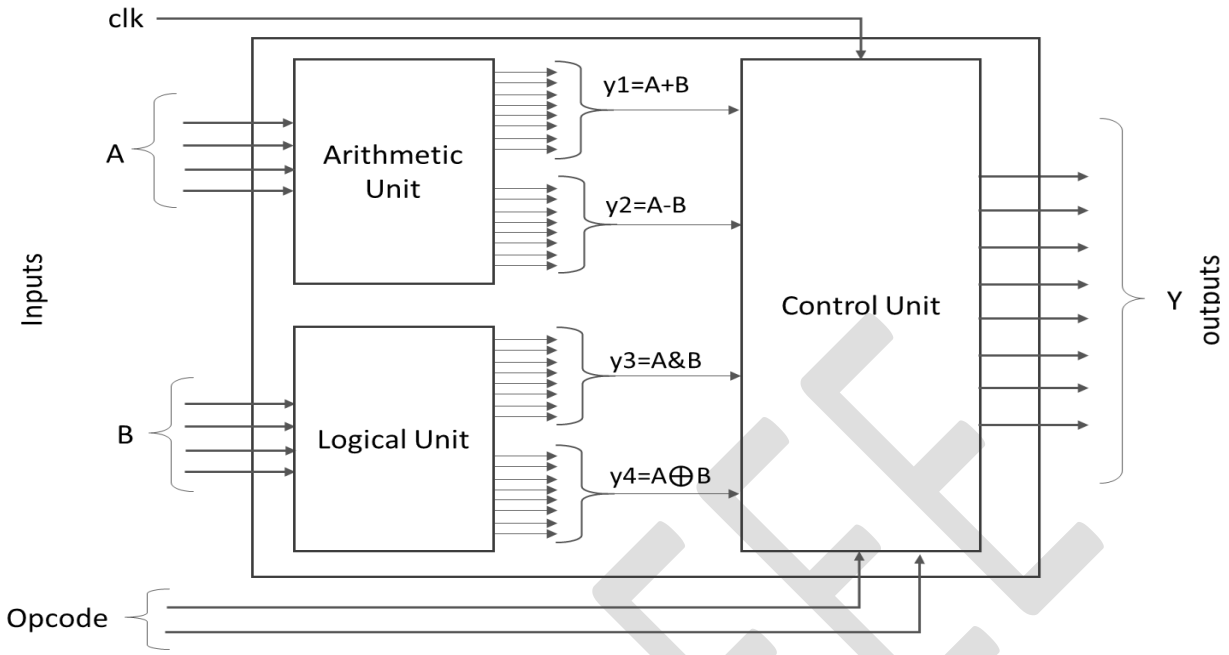
Testbench Module of Example 04

The following Verilog HDL code demonstrates the **Testbench Module** of the accumulator of Example 04.

```
1   `timescale 1ns/1ps
2   module accu_TB;
3   reg [7:0] in;
4   reg clk,reset;
5   wire [7:0] out;
6   accu dut(in, out, clk, reset);
7   initial
8           clk = 1'b0;
9   always
10          #5 clk = ~clk;
11  initial
12  begin
13          #0 reset<=1; in<=1;
14          #5 reset<=0;
15          #50 $finish;
16  end
17  endmodule
```

Example 05

In this example, a 4-bit Arithmetic Logic Unit (ALU) shown in the following figure will be designed using Verilog HDL. The top module of the ALU is ***alu_4bit*** and it is designed using three sub-modules: ***logical_unit, arithmetic_unit,*** and ***control_unit***. In the design, the two 4-bit inputs **A** and **B** are fed to the inputs of ***arithmetic_unit*** and ***logical_unit*** modules to perform two different arithmetic operations and two different logical operations according to the function table given below. Thus the **arithmetic_unit** and **logical_unit generates** four outputs y1,y2,y3, and y4 which are fed to the inputs of ***control_unit*** module which generates the 8-bit output **Y** from the y1,y2,y3 and y4 depending on its 2-bit **Opcode** input. The output **Y** is also sensitive to the positive edge of the **clk** input.



The function table of the ALU is given below.

Function Table

Opcode	Output (Y)	Description of function
00	A+B	Add A to B
01	A-B	Subtract B from A
10	A&B	Bitwise AND
11	A⊕B	Bitwise XOR

The following Verilog HDL code demonstrates the ALU mentioned in Example 05.

```
1  module alu_4bit(A,B,Y,clk,Opcode);
2  input [3:0]A,B;
3  input [1:0]Opcode;
4  input clk;
5  output [7:0]Y;
6  wire [7:0]y1,y2,y3,y4;
7  arithmetic_unit sm1(A,B,y1,y2);
8  logical_unit sm2(A,B,y3,y4);
9  control_unit sm3(y1,y2,y3,y4,clk,Opcode,Y);
10 endmodule
11
12 module arithmetic_unit(x,y,y1,y2);
13 input [3:0]x,y;
14 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
```

Testbench Module of Example 05

The following Verilog HDL code demonstrates the **Testbench Module** of the 4-bit ALU of Example 05.

```
`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
```

```
9   begin
10          clk = 1'b0; Opcode=2'b00; A=4'b0100; B=4'b1100;
11  end
12  always
13          #2.5 clk = ~clk;
14  initial
14  begin
16          #5 Opcode<=2'b01; A=4'b1000; B=4'b0111;
17          #5 Opcode<=2'b10; A=4'b1111; B=4'b1011;
18          #5 Opcode<=2'b11; A=4'b1001; B=4'b1010;
19          #5 $finish;
20  end
21  endmodule
```

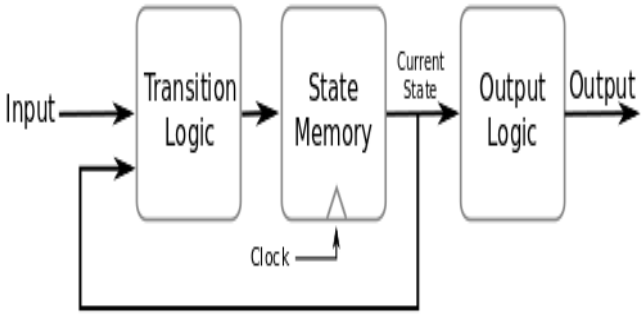
Finite State Machine Design

In a sequential circuit, outputs depend not only on the applied input values but also on the internal state. The internal state also changes with time. As the number of states in a sequential circuit is finite it is also referred to as a Finite State Machine (FSM). FSMs need memory to hold the current state and logic devices to determine the next state. Elevators(lift), vending machines, traffic signal systems, password generators etc. are examples of FSM.

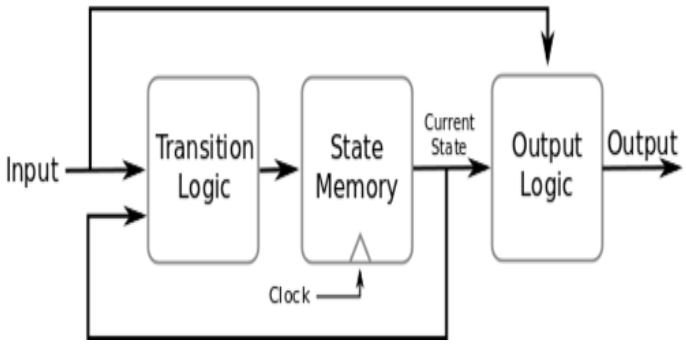
There are two types of finite state machines called the Mealy machine and the Moore Machine.

In Mealy machines, the output is a function of the current state and inputs. In Moore machines, the output is a function of only the current state. To design FSMs, we need to find the state transition diagram or the state table.

FSMs are modeled in Verilog with an always block defining the state registers and combinational logic defining the next state and output logic.



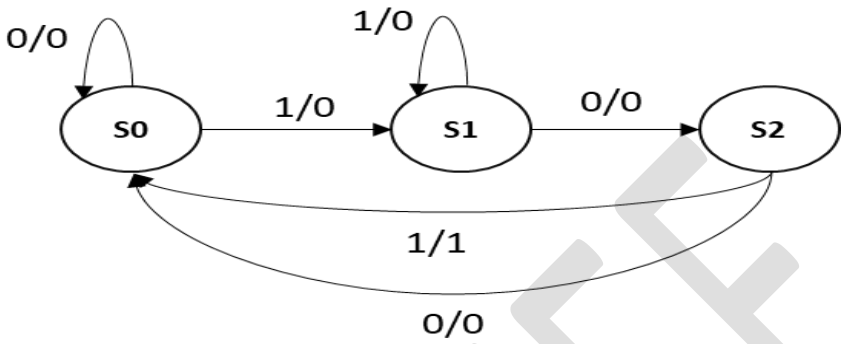
Moore Machine



Mealy Machine

Example 06

In this example, the Verilog HDL code of a Mealy machine is demonstrated that generates output '1' when sequence 101 is detected in a bitstream.



State Transition Diagram

The following Verilog HDL code demonstrates the sequence detector mentioned in Example 06.

```
1  module seq_101(i,clk,out);
2  input i,clk;
3  output reg out;
4  localparam S0=2'b00, S1=2'b01, S2=2'b10;
5  reg [1:0]state;
6  always@ (posedge clk)
7  begin
8  case (state)
9  S0: begin
10            out<=i?0:0;
11          state<=i?S1:S0;
12     end
13 S1: begin
14           out<=i?0:0;
15          state<=i?S1:S2;
16      end
17 S2: begin
18            out<=i?1:0;
19       state<=i?S0:S0;
20       end
21 default:
22        begin
23              out<=0;
24             state<=S0;
25         end
26 endcase
27 end
28 endmodule
```

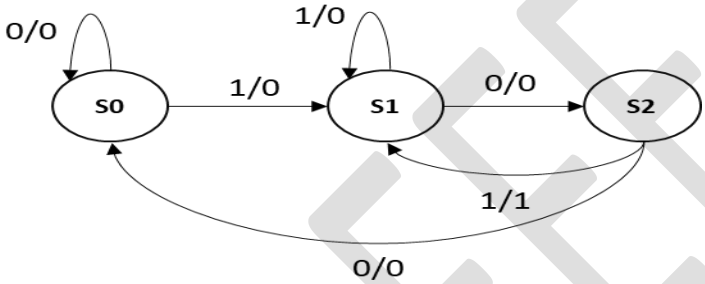
Testbench Module of Example 06

The following Verilog HDL code demonstrates the **Testbench Module** of the sequence detector of Example 06 where 0101001010-bit stream is generated.

```
1  `timescale 1ns/1ps
2  module seq_TB;
3  reg i,clk;
4  wire out;
5  seq dut(i,out,clk);
6  initial
7     clk=0;
8  always
9     #2 clk=~clk;
10 initial
11 begin
12    #0 i=0;
13    #5 i=1; #4 i=0; #4 i=1; #4 i=0;
14    #4 i=0; #4 i=1; #4 i=0; #4 i=1;
15    #4 i=0;
16    #4 $finish;
17 end
18 endmodule
```

Post Lab Tasks

1. Design a negative edge triggered D flip flop with reset and verify its functionality using testbench.
2. In Example 5 how many 1s will be generated at the output if the input bitstream is 01010100? Verify your answer using testbench.
3. Write a Verilog program to implement the digital system represented by the following state transition diagram of a Mealy machine. Assume that system has input and output variables **in** and **Y.** The system functions when the positive edge of the clock is detected.



4. Design a Mealy machine to detect the 010 sequences hence verifying its functionality using testbench.
5. The state transition diagram of a two-bit counter is shown below. Assuming that each state changes when a positive edge clock is detected. Design and verify the system using Verilog HDL.