**CS226 Lab 12**

**Design using HDL(Verilog)**

The goal of this lab12 is to familiarize the students with describing FSMs in verilog Hardware Description Language (HDL). In this lab presented in the form of problems and answers.

Task1: Simulate traffic light controller.

Finite state machine consists of combinational, sequential and output logic. Combinational logic is used to decide the next state of the FSM, sequential logic is used to store the current state of the FSM. The output depends on the current state of the machine in this example.

module fsm\_trafic\_light(clk,reset, red,yellow, green);

input clk, reset;

//output [1:0] state;

output red,yellow,green;

reg [1:0] next\_state;

reg[1:0] state;

parameter [1:0] FIRST= 2'b00;

parameter [1:0] SECOND= 2'b01;

parameter [1:0] THIRD = 2'b10;

parameter [1:0] FOURTH=2'b11;

always @(posedge clk) // sequential

begin

if (reset) state <= FIRST;

else state <= next\_state;

end

always @(state) // combinational

begin

case(state)

FIRST: if (reset)

next\_state = FIRST;

else next\_state = SECOND;

SECOND: if (reset)

next\_state = FIRST;

else next\_state = THIRD;

THIRD: if (reset)

next\_state = FIRST;

else next\_state = FOURTH;

FOURTH: if (reset)

next\_state = FIRST;

else next\_state = FIRST;

endcase end

// output logic described using continuous assignment

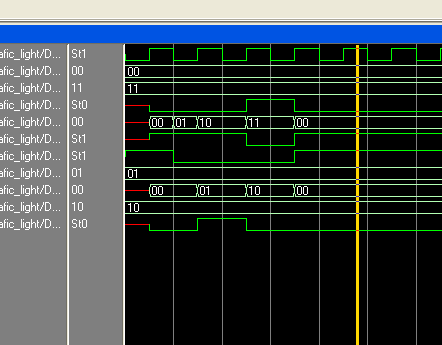
assign red = (state == FIRST) | (state == SECOND);

assign yellow = (state == SECOND | (state == FOURTH));

assign green = (state == THIRD);

endmodule





module fsm\_vendingmachine(open,clk,reset, coin\_T,coin\_F);

input clk, reset;

input coin\_T,coin\_F;

output open;

reg [1:0] next\_state;

reg[1:0] state;

parameter [1:0] FIRST= 2'b00;

parameter [1:0] SECOND= 2'b01;

parameter [1:0] THIRD = 2'b10;

parameter [1:0] FOURTH=2'b11;

always @(posedge clk) // sequential

begin

if (reset) state <= FIRST;

else state <= next\_state;

end

always @(state,coin\_T,coin\_F) // combinational

begin

case(state)

FIRST: if (coin\_F)

next\_state = SECOND;

else if (coin\_T) next\_state = THIRD;

SECOND: if (coin\_F)

next\_state = SECOND;

else next\_state = FOURTH;

THIRD: if (coin\_F)

next\_state = FOURTH;

else next\_state = FOURTH;

FOURTH: if (coin\_F)

next\_state = FOURTH;

else next\_state = FOURTH;

endcase end

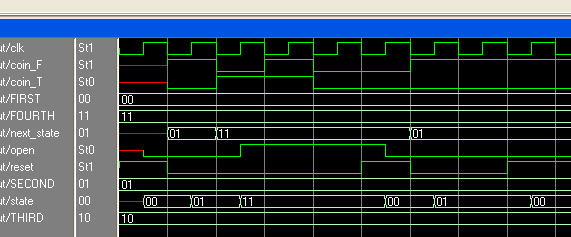
// output logic described using continuous assignment

assign open = (state == FOURTH);

endmodule

Task2: Simulate vending machine.





Task3: Simulate the state diagram description in verilog.

**module fsm(pause,restart,clk, rst, state,odd,even, terminal);**

**input pause,restart,clk, rst, odd,even;**

**output [1:0] state;**

**output terminal;**

**reg[1:0] next\_state,state;**

**parameter [1:0] FIRST= 2'b11;**

**parameter [1:0] SECOND= 2'b01;**

**parameter [1:0] THIRD = 2'b10;**

**always @(posedge clk) // sequential**

**begin**

**if (rst) state <= FIRST;**

**else state <= next\_state;**

**end**

**always @\* // combinational**

**begin**

**case(state)**

**FIRST: if (restart | pause)**

**next\_state = FIRST;**

**else next\_state = SECOND;**

**SECOND: if (restart)**

**next\_state = FIRST;**

**else if (pause) next\_state = SECOND;**

**else next\_state = THIRD;**

**THIRD: if (!restart & pause) next\_state = THIRD;**

**else next\_state = FIRST;**

**default: next\_state = FIRST;**

**endcase end**

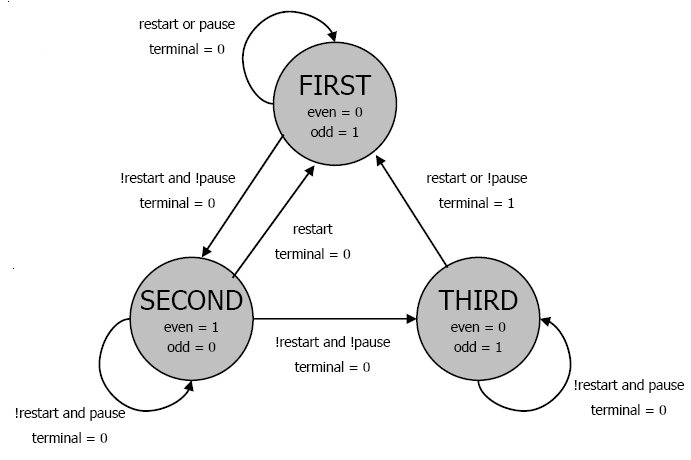
**// output logic described using continuous assignment**

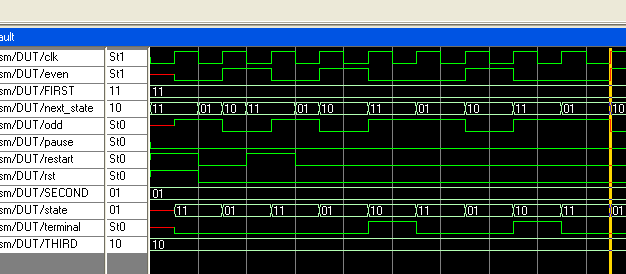
**assign odd = (state == FIRST) | (state == THIRD);**

**assign even = (state == SECOND);**

**assign terminal = (state == THIRD) & (restart | !pause);**

**endmodule**

**



**Your Lab Assignment 12 (60 points)**

**Submission** Your submission must contain- Verilog description, simulation waveforms of the exercises and problems given and test in a single file:



Lab submission through [cs225.iitp@gmail.com](mailto:cs225.iitp@gmail.com) with subject: **Yourroll\_No\_Lab12**

**Question 1**

Bob owns a pet snail with an FSM brain. The snail crawls from left to right along a paper tape containing a sequence of 1’s and 0’s. On each clock cycle, the snail crawls to the next bit. The snail smiles when the last four bits that it has crawled over are, from left to right, 1101. Design the FSM to compute when the snail should smile. The input A is the bit underneath the snail’s antennae. The output Y is ‘1’ when the snail smiles. Sketch a timing diagram (modelsim simulation output) showing the input, states, and output as your snail crawls along the sequence 11101101001001101101. **(20 points)**



**Hint : The state diagram has been solved for you ( see below)**

****

# Q2.

Design a 4-bit up-counter with an additional output *upper*. *upper* outputs a 1 whenever he counter is within the upper half of the counter’s range, 8 to 15. Use a basic 4-bit up-counter .

(hint: basic architecture is given below) . Create a Verilog model.

# 

**(20 points)**

**Q3: Create a Verilog Model for a 3-bit counter with following behavior**

**C0C1: 00 Stop counting**

**C0C1: 01 count up**

**C0C1:10 Count down**

**C0C2:11 : count by 2.**

**(C0 & C1 are the control inputs) (20 points)**

: