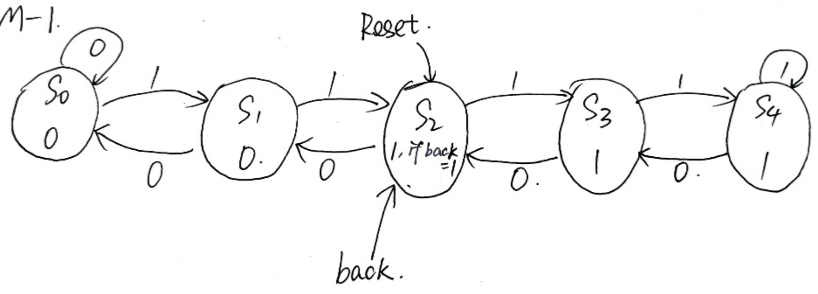
FSM-1

Sketch the state transition diagram for the FSM described by the following HDL code. An FSM of this nature is used in a branch predictor on some microprocessors.

Verilog

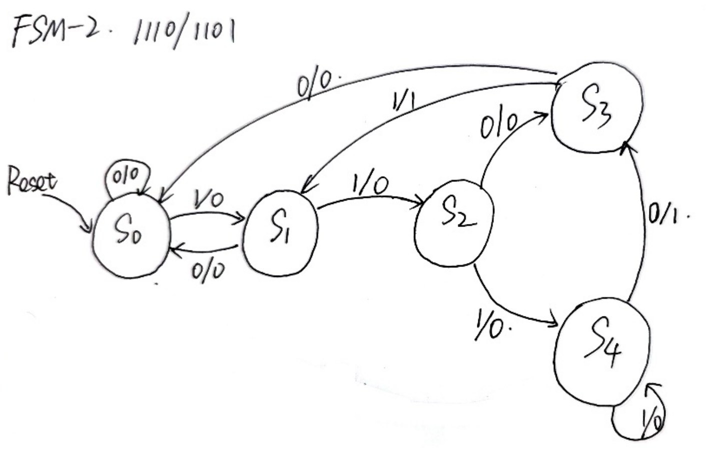
```
module fsm1 (input   clk, reset,
             input   taken, back,
             output  predicttaken);

  reg [4:0] state, nextstate;

  parameter S0 = 5'b00001;
  parameter S1 = 5'b00010;
  parameter S2 = 5'b00100;
  parameter S3 = 5'b01000;
  parameter S4 = 5'b10000;

  always @ (posedge clk, posedge reset)
    if (reset) state <= S2;
    else       state <= nextstate;

  always @ (*)
    case (state)
      S0: if (taken) nextstate = S1;
          else       nextstate = S0;
      S1: if (taken) nextstate = S2;
          else       nextstate = S0;
      S2: if (taken) nextstate = S3;
          else       nextstate = S1;
      S3: if (taken) nextstate = S4;
          else       nextstate = S2;
      S4: if (taken) nextstate = S4;
          else       nextstate = S3;
      default:       nextstate = S2;
    endcase

  assign predicttaken = (state == S4)  | |
                        (state == S3)  | |
                        (state == S2 && back);
endmodule
```

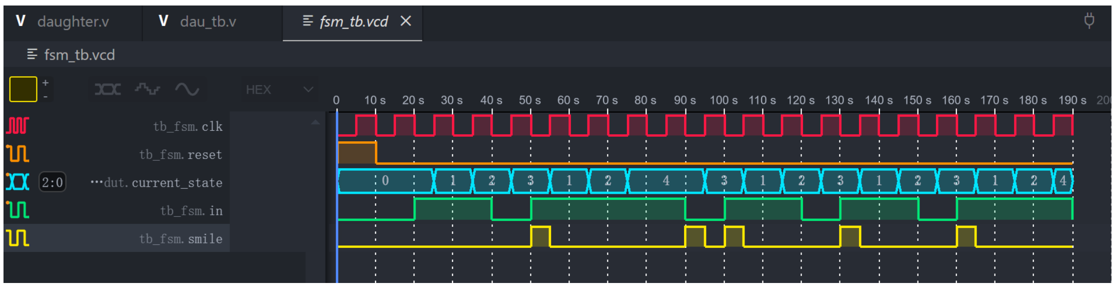
FSM-1.

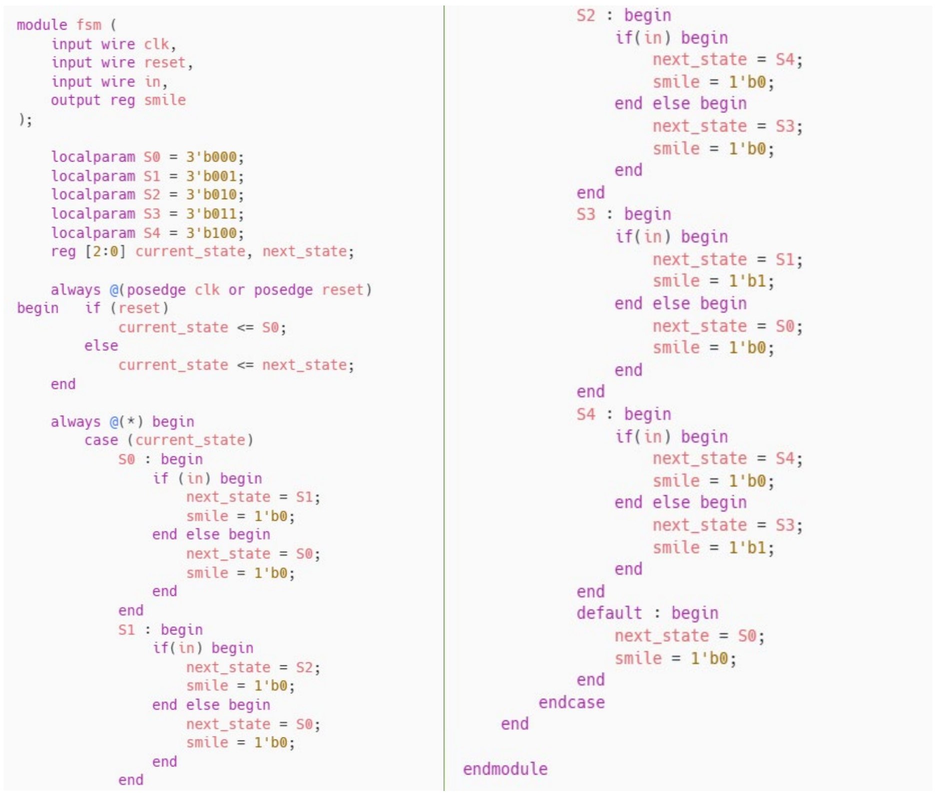


FSM-2

Alyssa P. Hacker's snail has a daughter with a Mealy machine FSM brain. The daughter snail smiles whenever she slides over the pattern 1101 or the pattern 1110. Sketch the state transition diagram for this happy snail using as few states as possible. Write Verilog code(including testbench) for this FSM, run simulation to verify you design.





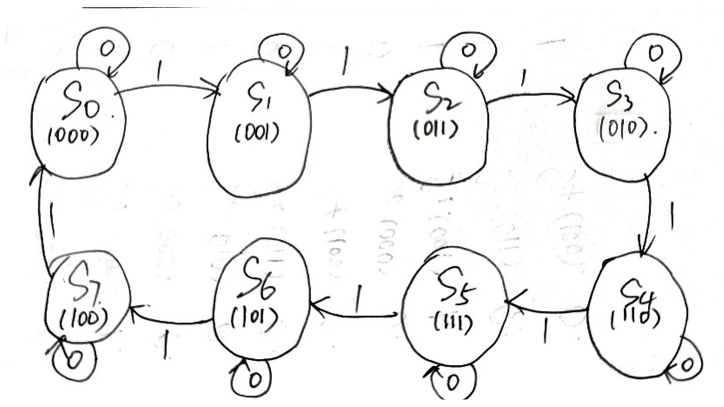


```verilog
module fsm (
    input wire clk,
    input wire reset,
    input wire in,
    output reg smile
);

    localparam S0 = 3'b000;
    localparam S1 = 3'b001;
    localparam S2 = 3'b010;
    localparam S3 = 3'b011;
    localparam S4 = 3'b100;
    reg [2:0] current_state, next_state;

    always @(posedge clk or posedge reset)
begin    if (reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    always @(*) begin
        case (current_state)
            S0 : begin
                if (in) begin
                    next_state = S1;
                    smile = 1'b0;
                end else begin
                    next_state = S0;
                    smile = 1'b0;
                end
            end
            S1 : begin
                if(in) begin
                    next_state = S2;
                    smile = 1'b0;
                end else begin
                    next_state = S0;
                    smile = 1'b0;
                end
            end
            S2 : begin
                if(in) begin
                    next_state = S4;
                    smile = 1'b0;
                end else begin
                    next_state = S3;
                    smile = 1'b0;
                end
            end
            S3 : begin
                if(in) begin
                    next_state = S1;
                    smile = 1'b1;
                end else begin
                    next_state = S0;
                    smile = 1'b0;
                end
            end
            S4 : begin
                if(in) begin
                    next_state = S4;
                    smile = 1'b0;
                end else begin
                    next_state = S3;
                    smile = 1'b1;
                end
            end
            default : begin
                next_state = S0;
                smile = 1'b0;
            end
        endcase
    end

endmodule
```

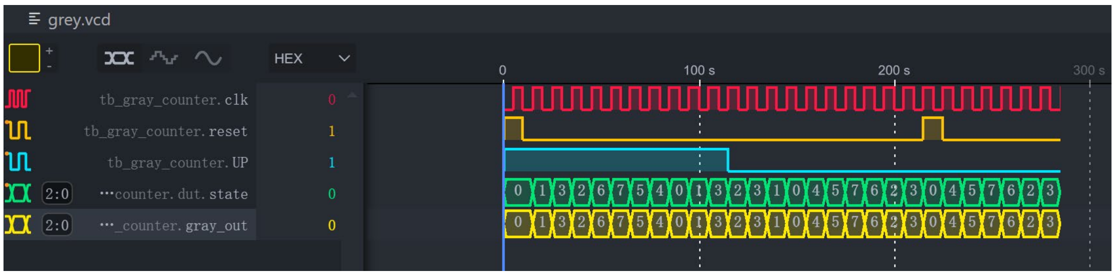
FSM-3

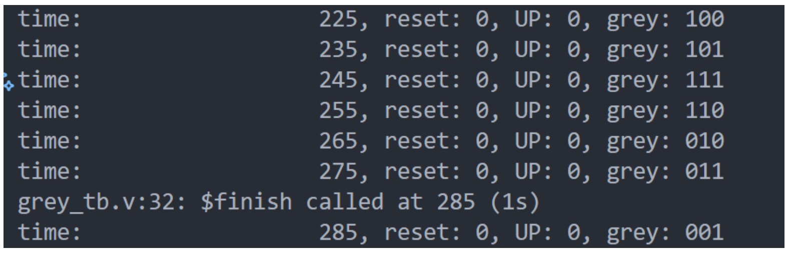
(1). Gray codes have a useful property in that consecutive numbers differ in only a single bit position. Table blow lists a 3-bit Gray code representing the numbers 0 to 7. Design a 3-bit modulo 8 Gray code counter FSM with no data inputs and three outputs. (A modulo *N* counter counts from 0 to *N* − 1, then repeats. For example, a watch uses a modulo 60 counter for the minutes and seconds that counts from 0 to 59.) When reset, the output should be 000. On each clock edge, the output should advance to the next Gray code. After reaching 100, it should repeat with 000. Draw state transition diagram for this FSM.

Number	Gray code		
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

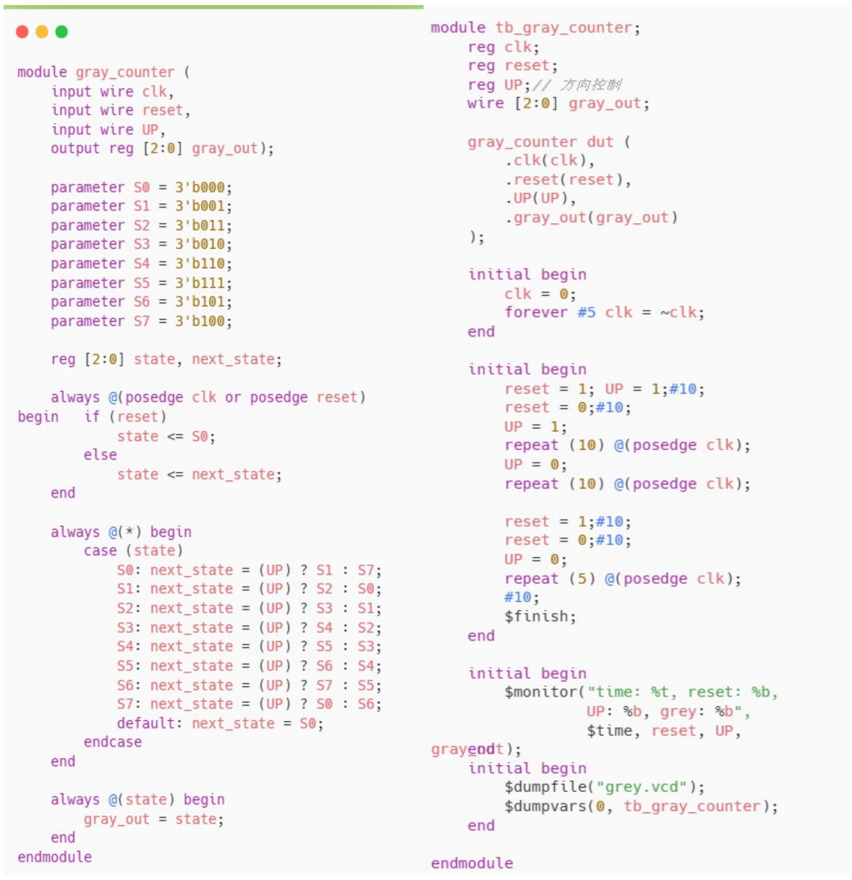


(2). Extend your modulo 8 Gray code counter to be an UP/DOWN counter by adding an UP input. If UP = 1, the counter advances to the next number. If UP = 0, the counter retreats to the previous number. write Verilog code and testbench for this FSM, compile and simulate your design.





```
time:                    225, reset: 0, UP: 0, grey: 100
time:                    235, reset: 0, UP: 0, grey: 101
time:                    245, reset: 0, UP: 0, grey: 111
time:                    255, reset: 0, UP: 0, grey: 110
time:                    265, reset: 0, UP: 0, grey: 010
time:                    275, reset: 0, UP: 0, grey: 011
grey_tb.v:32: $finish called at 285 (1s)
time:                    285, reset: 0, UP: 0, grey: 001
```



```verilog
module gray_counter (
    input wire clk,
    input wire reset,
    input wire UP,
    output reg [2:0] gray_out);

    parameter S0 = 3'b000;
    parameter S1 = 3'b001;
    parameter S2 = 3'b011;
    parameter S3 = 3'b010;
    parameter S4 = 3'b110;
    parameter S5 = 3'b111;
    parameter S6 = 3'b101;
    parameter S7 = 3'b100;

    reg [2:0] state, next_state;

    always @(posedge clk or posedge reset)
begin   if (reset)
            state <= S0;
        else
            state <= next_state;
    end

    always @(*) begin
        case (state)
            S0: next_state = (UP) ? S1 : S7;
            S1: next_state = (UP) ? S2 : S0;
            S2: next_state = (UP) ? S3 : S1;
            S3: next_state = (UP) ? S4 : S2;
            S4: next_state = (UP) ? S5 : S3;
            S5: next_state = (UP) ? S6 : S4;
            S6: next_state = (UP) ? S7 : S5;
            S7: next_state = (UP) ? S0 : S6;
            default: next_state = S0;
        endcase
    end

    always @(state) begin
        gray_out = state;
    end
endmodule
```

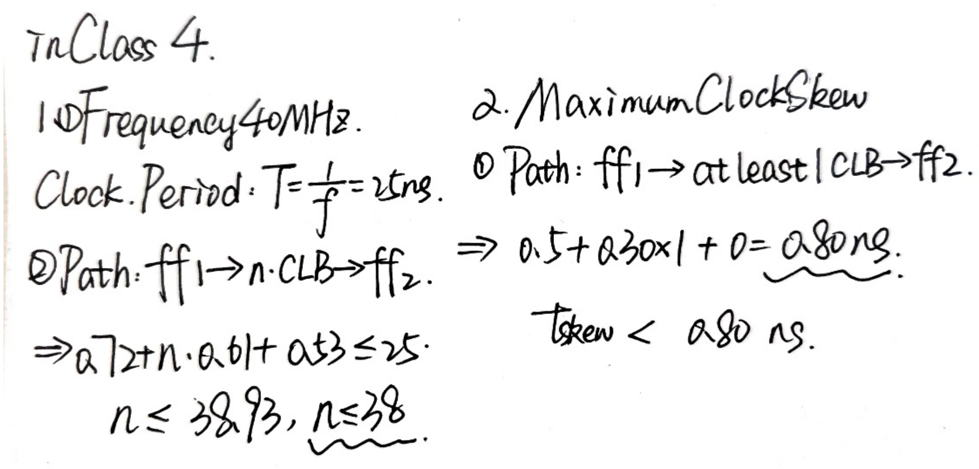
```verilog
module tb_gray_counter;
    reg clk;
    reg reset;
    reg UP;// 方向控制
    wire [2:0] gray_out;

    gray_counter dut (
        .clk(clk),
        .reset(reset),
        .UP(UP),
        .gray_out(gray_out)
    );

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 1; UP = 1;#10;
        reset = 0;#10;
        UP = 1;
        repeat (10) @(posedge clk);
        UP = 0;
        repeat (10) @(posedge clk);

        reset = 1;#10;
        reset = 0;#10;
        UP = 0;
        repeat (5) @(posedge clk);
        #10;
        $finish;
    end

    initial begin
        $monitor("time: %t, reset: %b,
                 UP: %b, grey: %b",
                 $time, reset, UP,
grayendt);
    initial begin
        $dumpfile("grey.vcd");
        $dumpvars(0, tb_gray_counter);
    end

endmodule
```

4. Timing

A field programmable gate array (FPGA) uses configurable logic blocks (CLBs) rather than logic gates to implement combinational logic. The Xilinx Spartan 3 FPGA has propagation and contamination delays of 0.61 and 0.30 ns, respectively, for each CLB. It also contains flip-flops with propagation and contamination delays of 0.72 and 0.50 ns, and setup and hold times of 0.53 and 0 ns, respectively.

(a) If you are building a system that needs to run at 40 MHz, how many consecutive CLBs can you use between two flip-flops? Assume there is no clock skew and no delay through wires between CLBs.

(b) Suppose that all paths between flip-flops pass through at least one CLB. How much clock skew can the FPGA have without violating the hold time?



InClass 4.

1 ①Frequency 40MHz.

Clock Period: $T=\frac{1}{f}=25ns$.

②Path: ff1 → n·CLB → ff2.

$\Rightarrow 0.72+n\cdot 0.61+0.53 \le 25.$

$n \le 38.93$, $\underline{n\le 38}$.

2. Maximum Clock Skew

① Path: ff1 → at least 1 CLB → ff2.

$\Rightarrow 0.5+0.30\times 1+0=\underline{0.80ns}$.

$t_{skew} < 0.80$ ns.