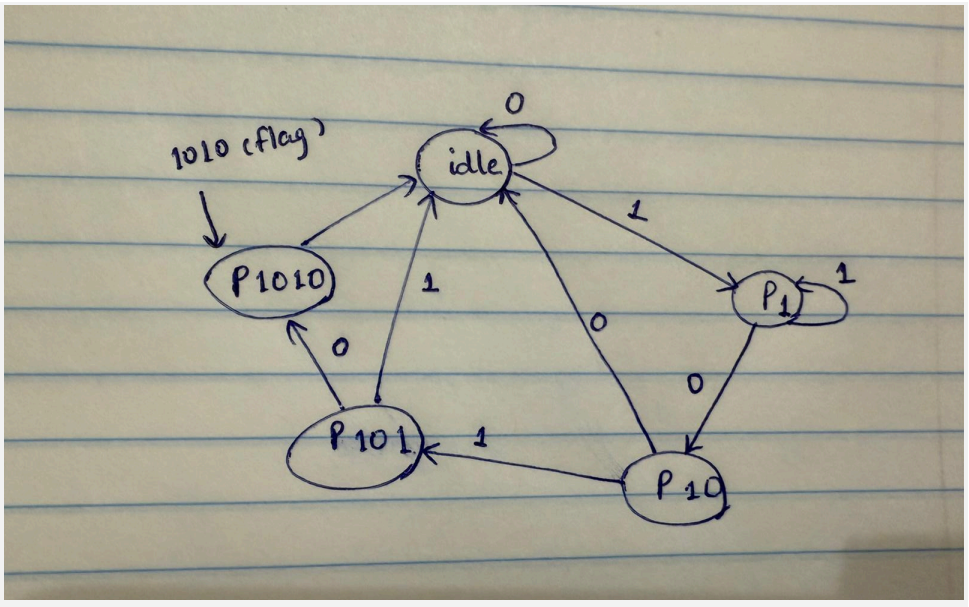
Nang Thiri Wutyi
20113
11/27/2024

Homework #5

1. **FSM logic chart to detect bit stream pattern “1010”**



Design module

```
module det_1010 (
   input clk, rst, in,
   output reg flag
);
   // States defined using one-hot encoding
   parameter pIdle   = 5'b00001;
   parameter p1      = 5'b00010;
   parameter p10     = 5'b00100;
   parameter p101    = 5'b01000;
   parameter p1010   = 5'b10000;

   reg [4:0] curSt, nxtSt;

   always @(posedge clk or posedge rst) begin
       if (rst)
           curSt <= #1 pIdle; // Reset state to pIdle
       else
           curSt <= #1 nxtSt; // Update state on clock edge
   end
```

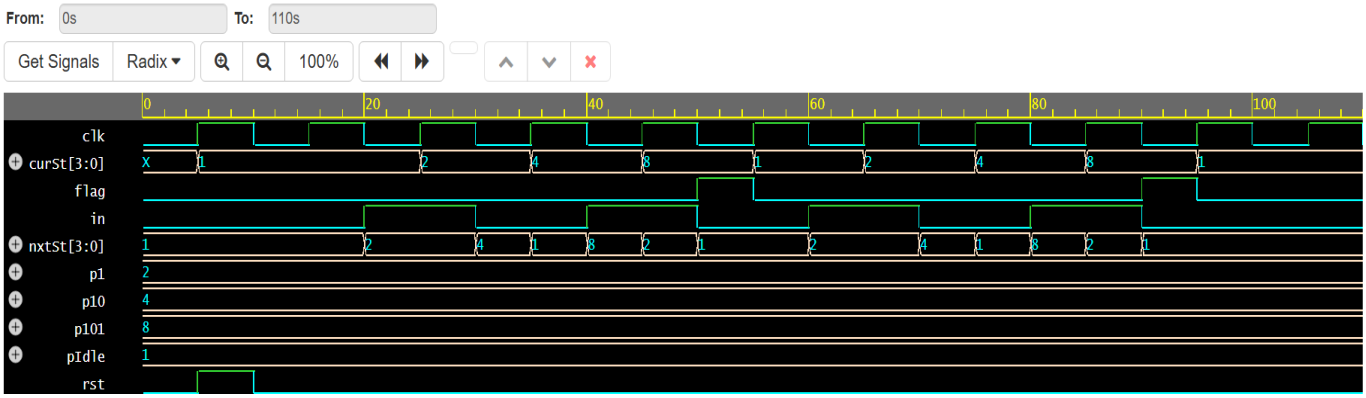
```
    always @(*) begin
        flag = 1'b0; // Default flag value
        case (curSt)
            pIdle: begin
                if (in) nxtSt = p1; // Transition to p1 on receiving '1'
                else nxtSt = pIdle; // Stay in pIdle if '0'
            end
            p1: begin
                if (in) nxtSt = p1; // Stay in p1 if '1'
                else nxtSt = p10; // Transition to p10 on receiving '0'
            end
            p10: begin
                if (in) nxtSt = p101; // Transition to p101 on receiving '1'
                else nxtSt = pIdle; // Return to pIdle if '0'
            end
            p101: begin
                if (in) nxtSt = p1010; // Transition to p1010 on receiving '1'
                else nxtSt = pIdle; // Return to pIdle if '0'
            end
            p1010: begin
                flag = 1'b1; // Set flag when `1010` is detected
                nxtSt = pIdle; // Return to pIdle state
            end
            default: nxtSt = pIdle; // Default state is pIdle
        endcase
    end
endmodule
```

Testbench Module

```
module tb_det_1010();
    reg clk, rst, in;
    wire flag;

    // Instantiate the FSM module
    det_1010 u0 (.clk(clk), .rst(rst), .in(in), .flag(flag));

    initial begin
        // Initialize signals
```

```
        clk = 0;
        rst = 0;
        in = 0;

        // Apply reset
        #2 rst = 1;
        #3 rst = 0;

        // Apply test pattern `1010`
        #5 in = 1;
        #5 in = 0;
        #5 in = 1;
        #5 in = 0;
        #5 $finish; // End simulation
    end

    // Generate clock signal
    always #3 clk = ~clk;

    // Monitor output
    initial begin
        $monitor("At time %t, flag = %b, input = %b", $time, flag, in);
    end
endmodule
```

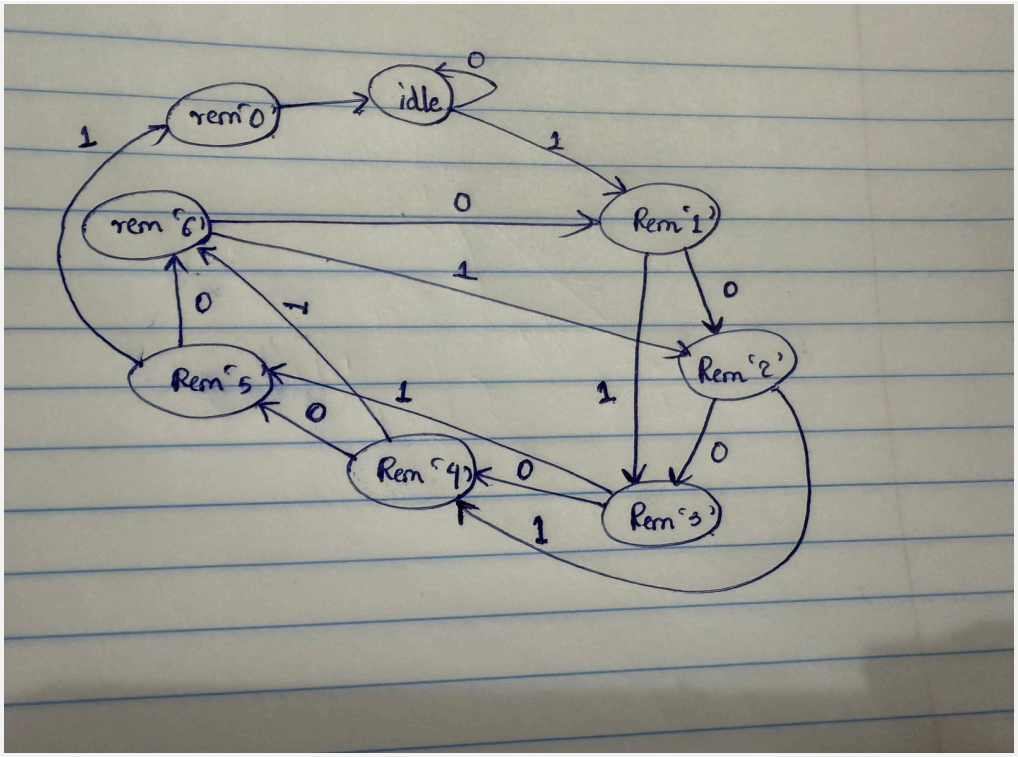
Output




```
[2024-11-27 10:12:39 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  &&
unbuffer vvp a.out
At time                    0, flag = 0, input = 0
At time                   10, flag = 0, input = 1
At time                   15, flag = 0, input = 0
```

```
At time                    20, flag = 0, input = 1
At time                    25, flag = 0, input = 0
testbench.sv:23: $finish called at 30 (1s)
Done
```

2. FSM to detect bit streams which could be divided by 7



Design Module

```
module div_7(
    input clk,
    input rst,
    input in,
    output reg flag
);

    // Define states for remainders 0 to 6 (one-hot encoding)
    parameter pRem_0 = 7'b0000001;
    parameter pRem_1 = 7'b0000010;
    parameter pRem_2 = 7'b0000100;
    parameter pRem_3 = 7'b0001000;
    parameter pRem_4 = 7'b0010000;
    parameter pRem_5 = 7'b0100000;
    parameter pRem_6 = 7'b1000000;

    reg [6:0] curSt, nxtSt;
```

```
    always @(posedge clk) begin
        if (rst) curSt <= pRem_0; // Start at remainder 0 on reset
        else curSt <= nxtSt;
    end

    always @(*) begin
        flag = 0;  // Default flag value
        case(curSt)
            pRem_0: begin
                if (in) nxtSt = pRem_1;  // Shift to remainder 1 if input is 1
                else nxtSt = pRem_0;     // Stay in remainder 0 if input is 0
            end
            pRem_1: begin
                if (in) nxtSt = pRem_2;
                else nxtSt = pRem_3;
            end
            pRem_2: begin
                if (in) nxtSt = pRem_4;
                else nxtSt = pRem_5;
            end
            pRem_3: begin
                if (in) nxtSt = pRem_6;
                else nxtSt = pRem_0;
            end
            pRem_4: begin
                if (in) nxtSt = pRem_1;
                else nxtSt = pRem_2;
            end
            pRem_5: begin
                if (in) nxtSt = pRem_3;
                else nxtSt = pRem_4;
            end
            pRem_6: begin
                if (in) nxtSt = pRem_5;
                else nxtSt = pRem_6;
                flag = 1;  // Divisible by 7 when remainder is 0 after 7th bit
            end
            default: nxtSt = pRem_0;  // Default state is remainder 0
        endcase
```

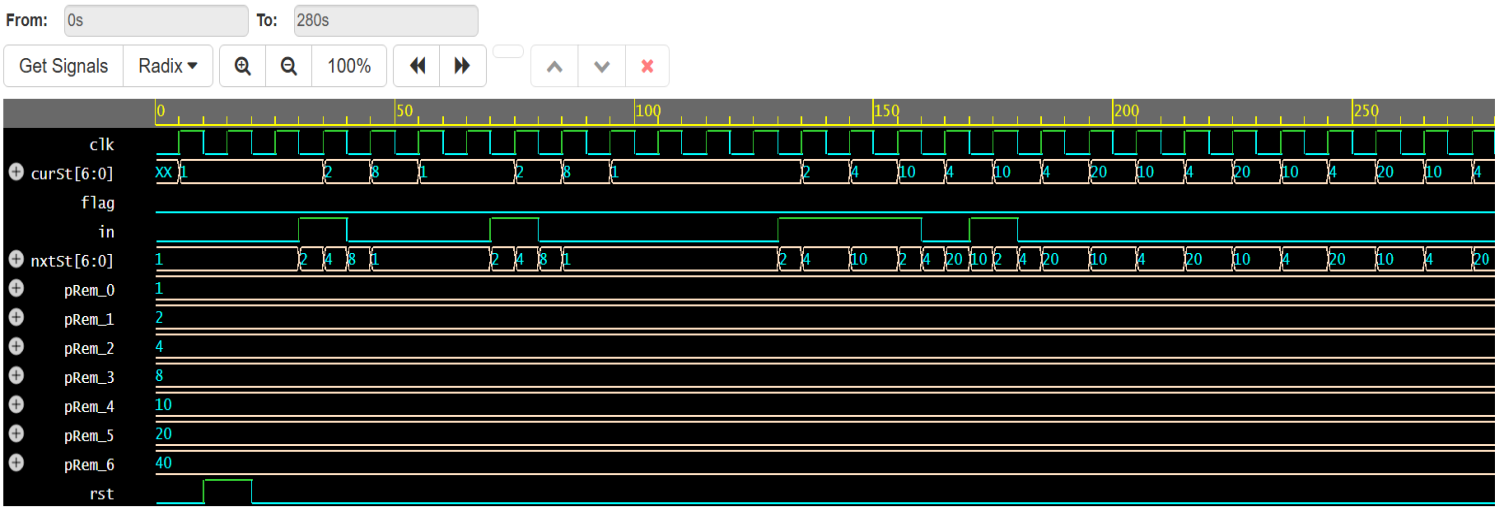
```
    end
endmodule
```

Testbench Module

```
module tb_div_7_tb;

    reg clk, rst, in;
    wire flag;

    // Instantiate the div_7 module
    div_7 uut (
        .clk(clk),
        .rst(rst),
        .in(in),
        .flag(flag)
    );

    // Clock generation
    always #5 clk = ~clk; // Clock period of 10 time units

    // Initial block to initialize signals and apply test vectors
    initial begin
        clk = 0;
        rst = 0;
        in = 0;

        // Apply reset
        #10 rst = 1;
        #10 rst = 0;

        // Test input sequence for 7 (divisible by 7)
        #10 in = 1;
        #10 in = 0;
        #10 in = 0;
        #10 in = 0;
        #10 in = 1;
        #10 in = 0;
        #10 in = 0; // Divisible by 7, flag should be high

        #20; // Wait a bit for simulation to reflect output
```

```
        // Test another input sequence for 14 (divisible by 7)
        #20 in = 1;
        #10 in = 1;
        #10 in = 1;
        #10 in = 0;
        #10 in = 1;
        #10 in = 0; // Divisible by 7, flag should be high

        // Finish simulation
        #100 $finish;
    end

    // Dump waveforms for viewing
    initial begin
        $dumpfile("div_7_tb.vcd");
        $dumpvars;
    end

endmodule
```

Output



3. **The duty cycle of PWM signal with a higher clock frequency**

Design Module

```
module duty_cycle_calculator (
    input clk,
```

```
    input reset,
    input [7:0] duty_cycle, // 0-100% duty cycle
    output reg pwm_signal
);

    reg [7:0] counter; // Counter for PWM signal generation
    reg [7:0] duty_limit;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            counter <= 8'd0;
            pwm_signal <= 0;
            duty_limit <= 8'd0;
        end else begin
            duty_limit <= duty_cycle; // Set duty cycle limit
            if (counter < 8'd100) begin
                counter <= counter + 8'd1;
            end else begin
                counter <= 8'd0;
            end

            // PWM signal logic
            if (counter < duty_limit)
                pwm_signal <= 1;
            else
                pwm_signal <= 0;
        end
    end

endmodule
```

Testbench module

```
module testbench;
    reg clk;
    reg reset;
    reg [7:0] duty_cycle;
    wire pwm_signal;

    // Instantiate the DUT (Design Under Test)
    duty_cycle_calculator dut (
```

```
        .clk(clk),
        .reset(reset),
        .duty_cycle(duty_cycle),
        .pwm_signal(pwm_signal)
    );

    // Clock generation
    always begin
        #5 clk = ~clk; // Clock period = 10 time units
    end

    initial begin
        // Initialize signals
        clk = 0;
        reset = 1;
        duty_cycle = 8'd50; // Start with 50% duty cycle

        // Dump waveform data to VCD file
        $dumpfile("waveform.vcd");
        $dumpvars(0, testbench);

        // Reset DUT
        #10 reset = 0;

        // Test different duty cycles
        #10 duty_cycle = 8'd10; // 10% duty cycle
        #100 duty_cycle = 8'd50; // 50% duty cycle
        #100 duty_cycle = 8'd90; // 90% duty cycle
        #100 duty_cycle = 8'd100; // 100% duty cycle

        // Finish simulation
        #500 $finish;
    end

    // Monitor the output for debugging
    initial begin
        $monitor("Time: %t | PWM Signal: %b | Duty Cycle: %0d%%", $time, pwm_signal, duty_cycle);
    end
endmodule
```

Output

```
[2024-11-27 22:03:11 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile waveform.vcd opened for output.
Time:                     0 | PWM Signal: 0 | Duty Cycle: 50%
Time:                    20 | PWM Signal: 0 | Duty Cycle: 10%
Time:                    25 | PWM Signal: 1 | Duty Cycle: 10%
Time:                   115 | PWM Signal: 0 | Duty Cycle: 10%
Time:                   120 | PWM Signal: 0 | Duty Cycle: 50%
Time:                   135 | PWM Signal: 1 | Duty Cycle: 50%
Time:                   220 | PWM Signal: 1 | Duty Cycle: 90%
Time:                   320 | PWM Signal: 1 | Duty Cycle: 100%
testbench.sv:40: $finish called at 820 (1s)
Done
```