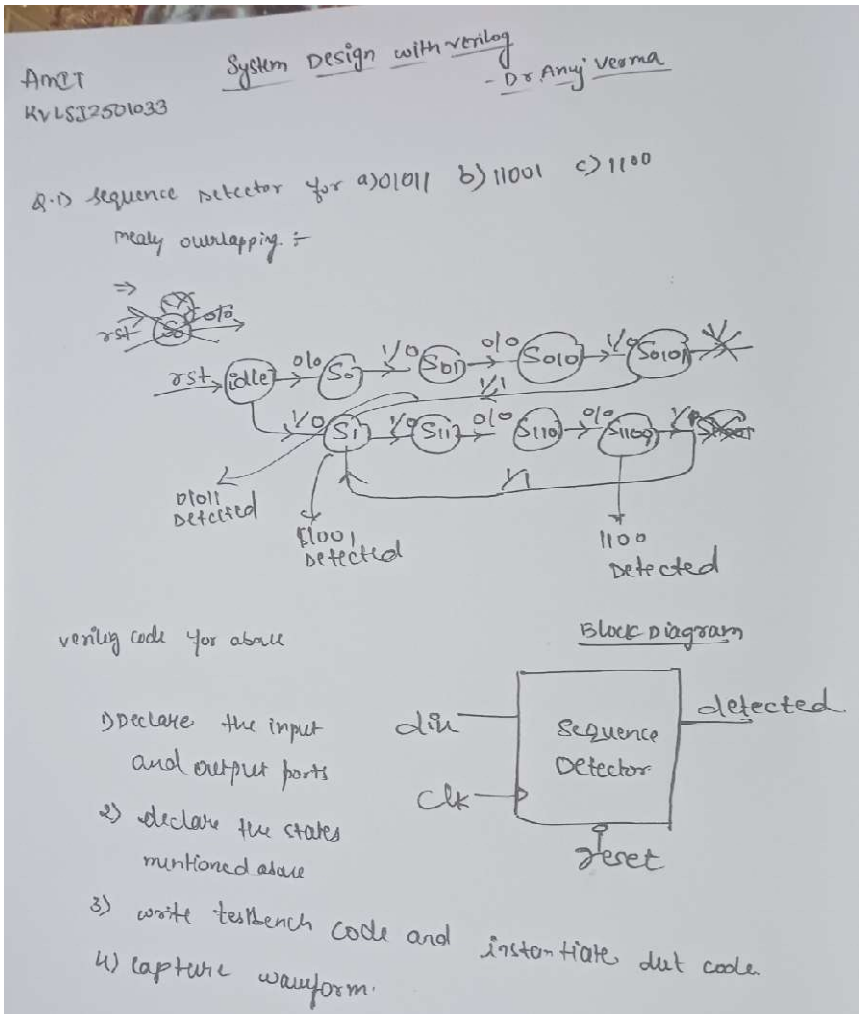
Assignment – 02

System Desing using Verilog

Submitted to : Dr. Anuj Verma **Name:** Amit

Roll No: KVLSI2501033

Question 1: Design a single State Transition Diagram (STD) for a Mealy-based sequence detector that detects overlapping occurrences of the following binary sequences:
a) 01011
b) 11001
c) 1100
Write down the Verilog code for the generated STD for the given sequences



Solution: Design code :

```
// Code your design here
module sequence_detector (
    input clk,
    input reset,
    input din,
    output reg detected
```

```
);

    // State encoding using parameter
    parameter S0      = 4'd0,
              S1      = 4'd1,
              S11     = 4'd2,
              S110    = 4'd3,
              S1100   = 4'd4,
              S11001  = 4'd5,
              S01     = 4'd6,
              S010    = 4'd7,
              S0101   = 4'd8,
              S01011  = 4'd9;

    reg [3:0] current_state, next_state;

    // Sequential logic: state transition
    always @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    // Combinational logic: next state and output
    always @(*) begin
        detected = 0;
        case (current_state)
            S0:      next_state = din ? S1 : S01;

            // 01011 detection path
            S01:     next_state = din ? S010 : S01;
            S010:    next_state = din ? S0101 : S0;
            S0101:   next_state = din ? S01011 : S01;
            S01011: begin
                detected = 1;                // 01011 detected
                next_state = din ? S1 : S01;
            end

            // 1100 and 11001 detection path
            S1:      next_state = din ? S11 : S01;
            S11:     next_state = din ? S11 : S110;
            S110:    next_state = din ? S0101 : S1100;
            S1100: begin
                detected = 1;                // 1100 detected
                next_state = din ? S11001 : S01;
            end
            S11001: begin
                detected = 1;                // 11001 detected
                next_state = din ? S1 : S01;
            end
```

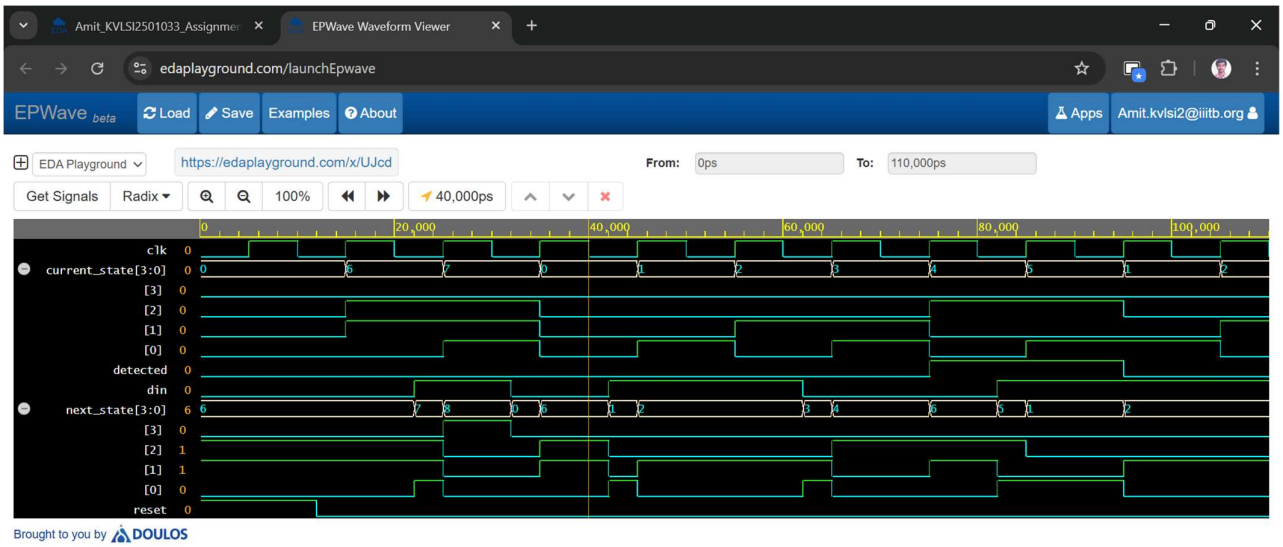
```
            default: next_state = S0;
        endcase
    end

endmodule
```

Testbench code :

```
`timescale 1ns / 1ps


module sequence_detector_tb;

    reg clk;
    reg reset;
    reg din;
    wire detected;

    // Instantiate the DUT (Device Under Test)
    sequence_detector uut (
        .clk(clk),
        .reset(reset),
        .din(din),
        .detected(detected)
    );

    // Clock generation (10ns period)
    initial clk = 0;
    always #5 clk = ~clk;

    // Bitstream that contains all 3 target sequences
    reg [7:0] seq_data = 8'b01011001;  // Contains 01011, 1100, and 11001
    integer i;

    initial begin
        // Setup for waveform dump
        $dumpfile("sequence_detector.vcd");
        $dumpvars(0, sequence_detector_tb);

        $display("Time\tClk\tReset\tInput\tDetected");
        $monitor("%0dns\t%b\t%b\t%b\t%b", $time, clk, reset, din, detected);

        // Apply reset
        reset = 1;
        din = 0;
        #12;
        reset = 0;

        // Feed input data bit-by-bit (MSB first)
        for (i = 7; i >= 0; i = i - 1) begin
            din = seq_data[i];
            #10;
        end
```

```
    #20;
    $finish;
  end

endmodule
```

Waveform:



Question 2: Design a Verilog module for a Washing Machine with the following specifications: Functional Requirements:

The machine has six states:

a) IDLE: Waiting for user to start

b) FILL: Filling water (2 minutes)

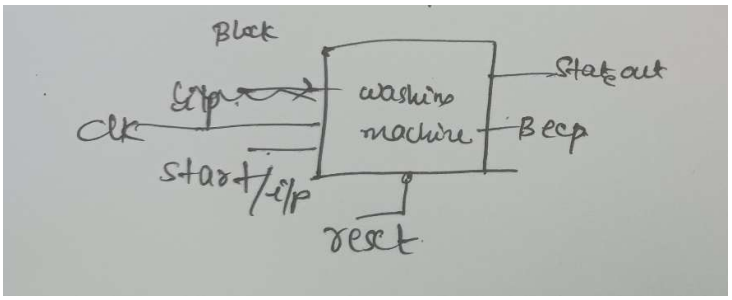
c) WASH: Washing clothes (20 minutes)

d) RINSE: Clothes Rinse (10 minutes)

e) DRAIN: Draining water (3 minutes)

f) SPIN: Spin clothes (5 minutes)

After completing all these steps, a DONE/BEEP signal should be high for 30 seconds.



Design Code:

```
// Code your design here

`timescale 1ns / 1ps

module washing_machine (
    input wire clk,
    input wire reset,
    input wire start,
    output reg [2:0] state_out,
    output reg beep
);

    // State Encoding
    parameter IDLE  = 3'd0,
              FILL  = 3'd1,
              WASH  = 3'd2,
              RINSE = 3'd3,
              DRAIN = 3'd4,
              SPIN  = 3'd5,
              DONE  = 3'd6;

    reg [2:0] state, next_state;
    reg [15:0] timer;

    // Duration Constants (in seconds)
    parameter FILL_TIME  = 2 * 60;
    parameter WASH_TIME  = 20 * 60;
    parameter RINSE_TIME = 10 * 60;
    parameter DRAIN_TIME = 3 * 60;
    parameter SPIN_TIME  = 5 * 60;
    parameter DONE_TIME  = 30;

    // State Register
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= IDLE;
            timer <= 0;
        end else begin
            state <= next_state;
            if (state != next_state)
                timer <= 0;
            else
                timer <= timer + 1;
        end
    end

    // Output and Next State Logic
    always @(*) begin
        next_state = state;
        beep = 0;
```

```
        case (state)
            IDLE: begin
                if (start)
                    next_state = FILL;
            end

            FILL: begin
                if (timer >= FILL_TIME)
                    next_state = WASH;
            end

            WASH: begin
                if (timer >= WASH_TIME)
                    next_state = RINSE;
            end

            RINSE: begin
                if (timer >= RINSE_TIME)
                    next_state = DRAIN;
            end

            DRAIN: begin
                if (timer >= DRAIN_TIME)
                    next_state = SPIN;
            end

            SPIN: begin
                if (timer >= SPIN_TIME)
                    next_state = DONE;
            end

            DONE: begin
                beep = 1;
                if (timer >= DONE_TIME)
                    next_state = IDLE;
            end

            default: next_state = IDLE;
        endcase
    end

    // For monitoring state externally
    always @(posedge clk) begin
        state_out <= state;
    end

endmodule
```

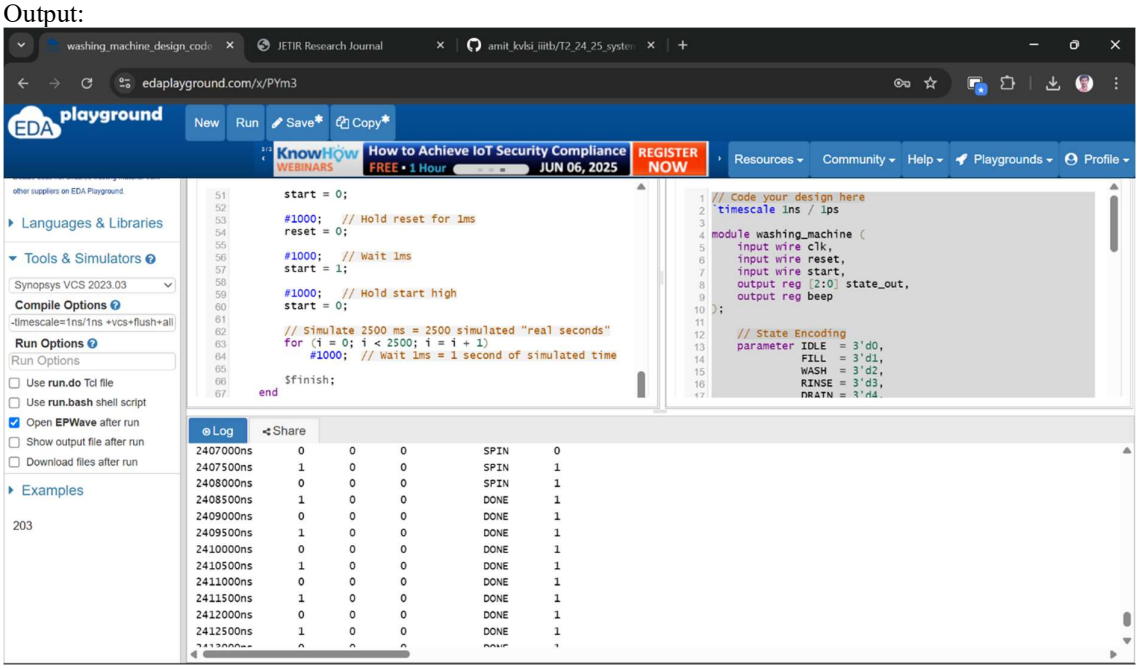
Testbench Code:

```
`timescale 1ns / 1ns
```

```
module washing_machine_tb;

    reg clk;
    reg reset;
    reg start;
    wire [2:0] state_out;
    wire beep;

    // Instantiate the DUT
    washing_machine uut (
        .clk(clk),
        .reset(reset),
        .start(start),
        .state_out(state_out),
        .beep(beep)
    );

    // Simulated Clock: 1kHz → 1ms period (1 real second ≈ 1ms)
    initial clk = 0;
    always #500 clk = ~clk;  // 1ms total cycle

    // Function to map state names
    function [79:0] get_state_name;
        input [2:0] state;
        case (state)
            3'd0: get_state_name = "IDLE ";
            3'd1: get_state_name = "FILL ";
            3'd2: get_state_name = "WASH ";
            3'd3: get_state_name = "RINSE";
            3'd4: get_state_name = "DRAIN";
            3'd5: get_state_name = "SPIN ";
            3'd6: get_state_name = "DONE ";
            default: get_state_name = "UNDEF";
        endcase
    endfunction

    integer i;

    initial begin
        // Dump for waveform
        $dumpfile("washing_machine.vcd");
        $dumpvars(0, washing_machine_tb);

        $display("Time(ns)\tClk\tReset\tStart\tState\tBeep");
        $monitor("%0dns\t%b\t%b\t%b\t%s\t%b", $time, clk, reset, start, get_state_name(state_out), beep);

        // Initial state
        reset = 1;
        start = 0;

        #1000;   // Hold reset for 1ms
```

```
    reset = 0;

    #1000;   // Wait 1ms
    start = 1;

    #1000;   // Hold start high
    start = 0;

    // Simulate 2500 ms = 2500 simulated "real seconds"
    for (i = 0; i < 2500; i = i + 1)
        #1000;  // Wait 1ms = 1 second of simulated time

    $finish;
  end

endmodule
```

Output:



Question 3: Design and implement the I2C Bus Protocol using Verilog on Modelsim.

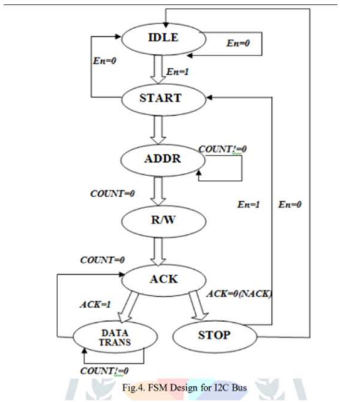


Fig.4. FSM Design for I2C Bus

FSM for I2C bus protocol .

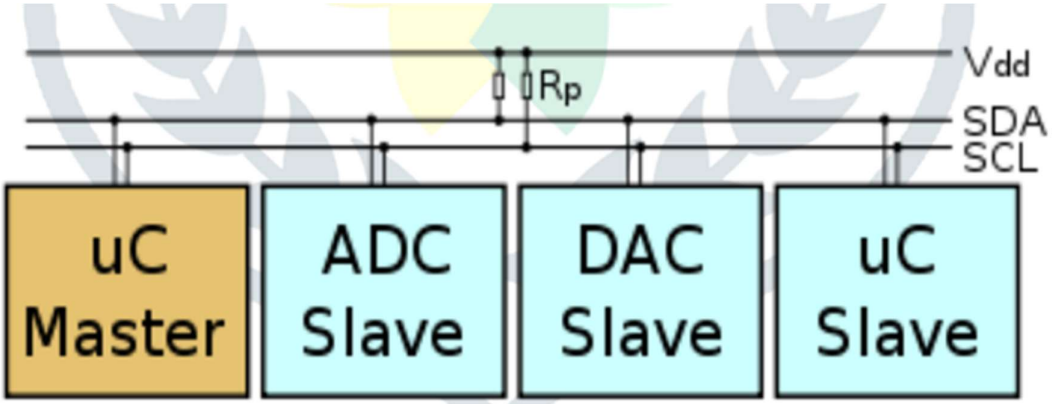


Fig.1 Block diagram of connecting devices using I2C

Design code : -

```
// ========================
// I2C Master Module
// ========================
module i2c_master (
    input wire clk,
    input wire rst,
    input wire enable,
    input wire [6:0] slave_address,
    input wire rw,          // 1 = Read, 0 = Write
    input wire [7:0] data_in,
    output reg sda,
    output reg scl,
    output reg done,
    output reg [2:0] state    // <-- Made 'state' visible to testbench
);

    reg [3:0] count;
    reg [7:0] shift_reg;
    reg ack;

    parameter IDLE      = 3'b000,
              START     = 3'b001,
              ADDR      = 3'b010,
              RW_BIT    = 3'b011,
              ACK_BIT   = 3'b100,
              DATA_TRANS = 3'b101,
              STOP      = 3'b110;

    // Clock divider for SCL
    reg scl_enable = 0;
    always @(posedge clk or posedge rst) begin
        if (rst) scl_enable <= 0;
        else scl_enable <= ~scl_enable;
    end
```

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        state <= IDLE;
        scl <= 1;
        sda <= 1;
        done <= 0;
        count <= 0;
    end else if (scl_enable) begin
        case (state)
            IDLE: begin
                scl <= 1;
                sda <= 1;
                done <= 0;
                if (enable) state <= START;
            end

            START: begin
                sda <= 0;
                scl <= 1;
                state <= ADDR;
                shift_reg <= {slave_address, rw};
                count <= 7;
            end

            ADDR: begin
                scl <= 0;
                sda <= shift_reg[count];
                scl <= 1;
                if (count == 0)
                    state <= RW_BIT;
                else
                    count <= count - 1;
            end

            RW_BIT: begin
                scl <= 0;
                sda <= rw;
                scl <= 1;
                state <= ACK_BIT;
            end

            ACK_BIT: begin
                scl <= 0;
                sda <= 1'bz;
                scl <= 1;
                ack <= sda; // Simulated ack read
                if (ack == 0)
                    state <= DATA_TRANS;
                else
                    state <= STOP;
            end
```

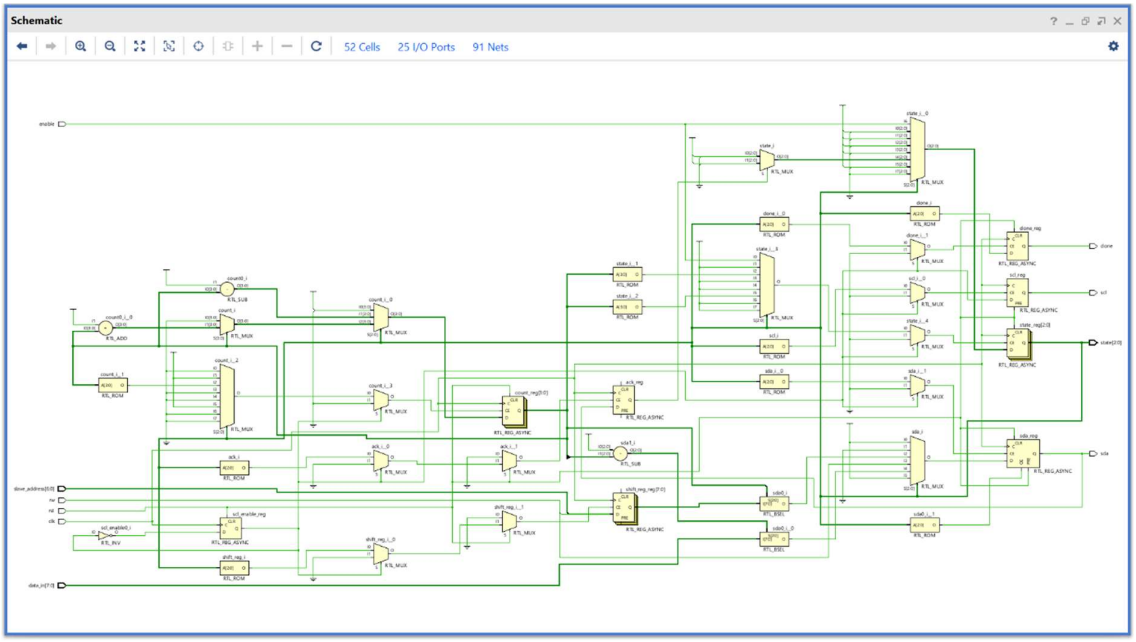
```
            DATA_TRANS: begin
                scl <= 0;
                sda <= data_in[7 - count];
                scl <= 1;
                if (count == 7) begin
                    count <= 0;
                    state <= STOP;
                end else begin
                    count <= count + 1;
                end
            end

            STOP: begin
                scl <= 1;
                sda <= 0;
                sda <= 1;
                done <= 1;
                if (!enable)
                    state <= IDLE;
                else
                    state <= START;
            end

            default: state <= IDLE;
        endcase
    end
  end
endmodule
```

Testbench code:-

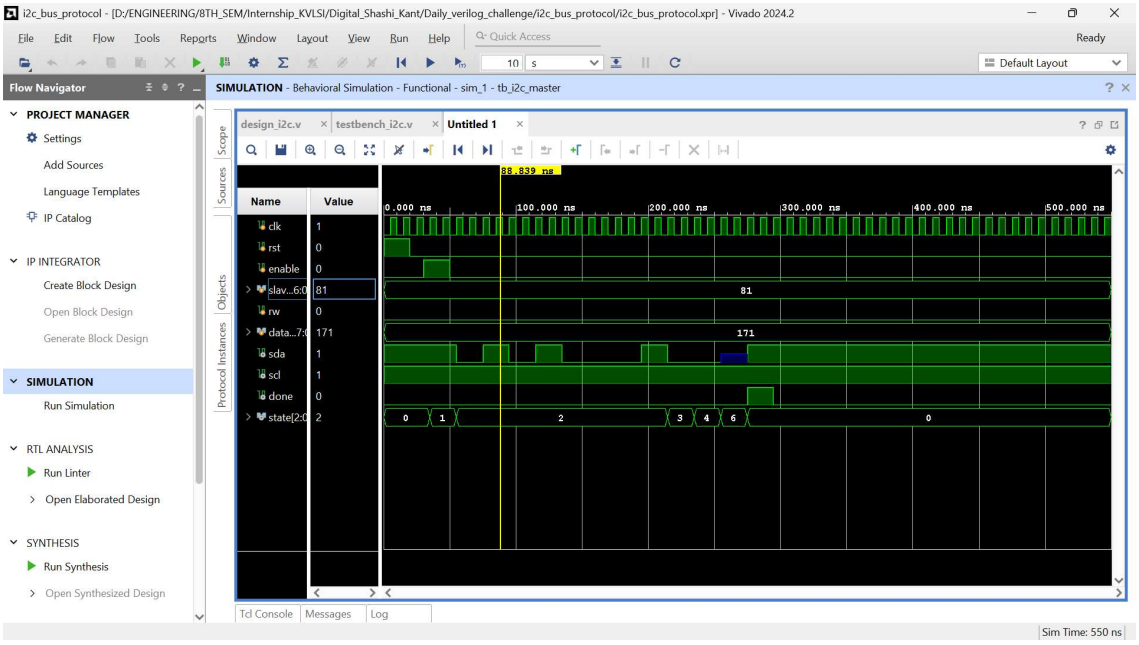
```
// ========================
// Testbench for I2C Master
// ========================
module tb_i2c_master;

  // Inputs
  reg clk;
  reg rst;
  reg enable;
  reg [6:0] slave_address = 7'b1010001;
  reg rw = 0;
  reg [7:0] data_in = 8'hAB;

  // Outputs
  wire sda;
  wire scl;
  wire done;
  wire [2:0] state;

  // Instantiate DUT
```

```
 i2c_master uut (
   .clk(clk),
   .rst(rst),
   .enable(enable),
   .slave_address(slave_address),
   .rw(rw),
   .data_in(data_in),
   .sda(sda),
   .scl(scl),
   .done(done),
   .state(state)
 );

 // Clock generation (10ns period = 100 MHz)
 always #5 clk = ~clk;

 initial begin
   // Initialize
   clk = 0;
   rst = 1;
   enable = 0;

   // Reset
   #20 rst = 0;

   // Start transaction
   #10 enable = 1;
   #20 enable = 0;

   // Wait and finish
   #500 $finish;
 end

 // Console monitor
 initial begin
   $monitor("Time=%0t | SDA=%b SCL=%b DONE=%b STATE=%d", $time, sda, scl, done, state);
 end

 // Dump for waveform (for ModelSim)
 initial begin
   $dumpfile("i2c_wave.vcd");
   $dumpvars(0, tb_i2c_master);
 end

endmodule
```

Schematic


Schematic
52 Cells
25 I/O Ports
91 Nets

Waveform


i2c_bus_protocol - [D:/ENGINEERING/8TH_SEM/Internship_KVLSI/Digital_Shashi_Kant/Daily_verilog_challenge/i2c_bus_protocol/i2c_bus_protocol.xpr] - Vivado 2024.2
File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_i2c_master
design_i2c.v
testbench_i2c.v
Untitled 1
Scope
Sources
Objects
Protocol Instances
Name
Value
clk
rst
enable
slav...6:0
rw
data...7:0
sda
scl
done
state[2:0
88.839 ns
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
81
171
Tcl Console
Messages
Log
Sim Time: 550 ns