WEEKLY REPORT – DIGITAL SYSTEM DESIGN USING VERILOG & FPGA

1. General Information

Project Title: HIỂN THỊ ĐẾM 0 ĐẾN 59 TRÊN TM1638

Students 1: Lê Quang Minh Nhật

Students 2: Nguyễn Quốc Hưng

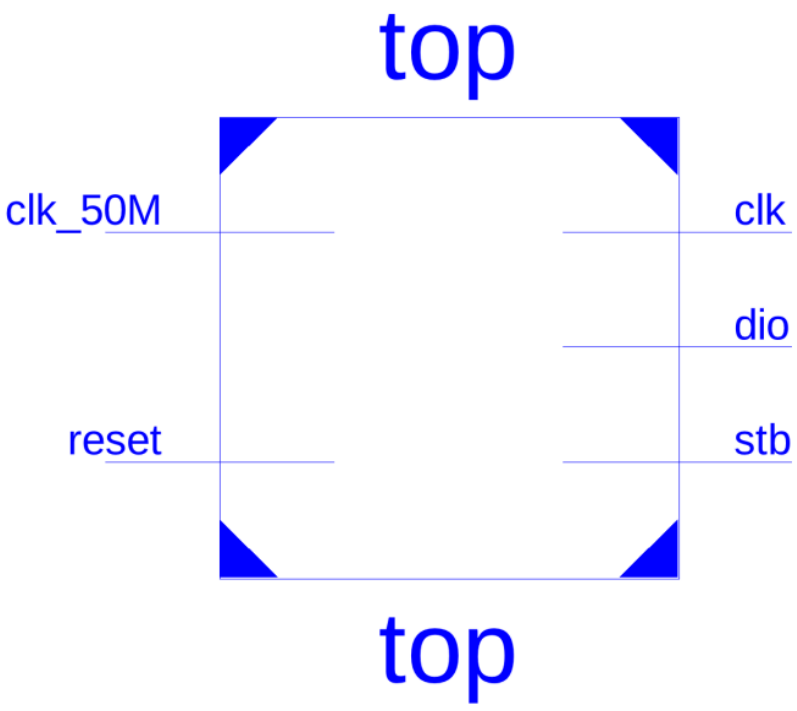
Students ID 1: 23139031

Students ID 2: 23139019

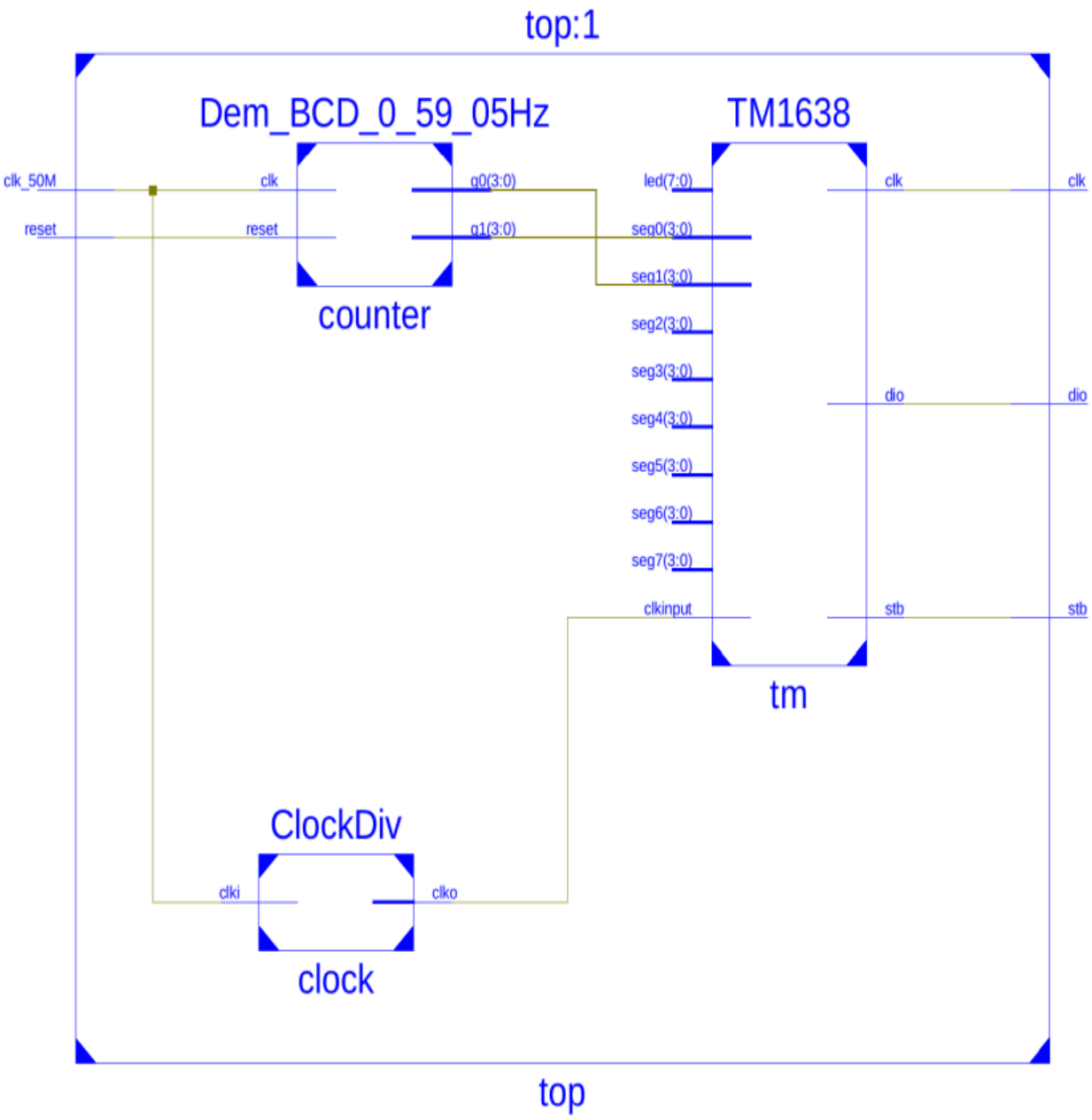
Date: 11/10/2025

2. Block Diagram

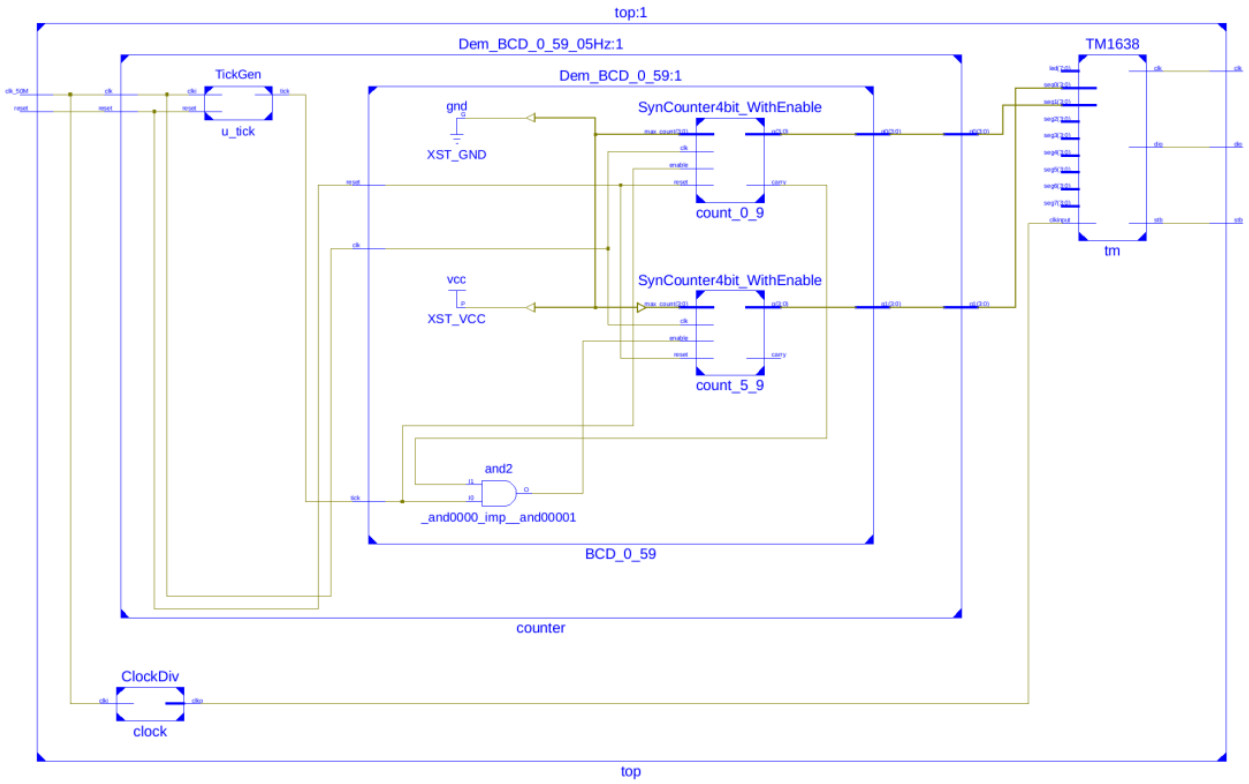
Draw block diagrams here.



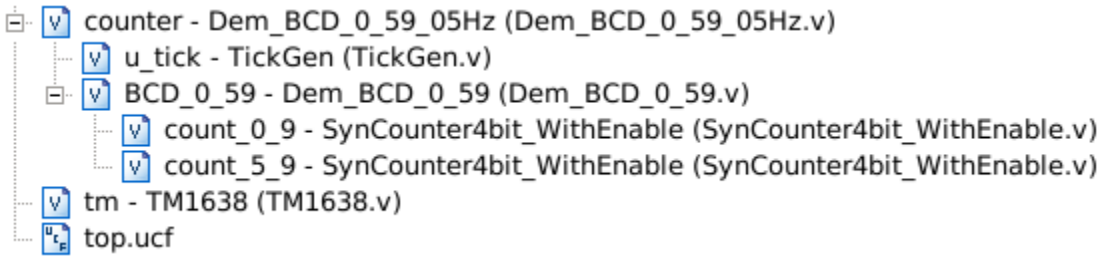
Sơ đồ khối tổng quát



Sơ đồ khối đếm và TM1638



Sơ đồ chi tiết các khối



Cấu trúc chương trình

3. State Transition Diagram

Describe Truth table or Design requirements or the FSM state diagram, etc

Mô-đun ClockDiv

Trạng thái hiện tại	Điều kiện	Trạng thái kế tiếp	Hành động	Ngõ ra (clko)

r_reg = 0 (Mặc định)	reset = 1	r_reg ← 0	Khởi tạo đếm	r_reg[5] = 0
r_reg = N (0 ≤ N < $2^{27}-1$)	reset = 0	r_reg ← N + 1	Tăng bộ đếm	clko = bit thứ 5 của r_reg
–	–	–	–	Tín hiệu clko dao động

Mô đun TickGen

Trạng thái (r_reg)	**Điều kiện vào**	**Chuyển trạng thái**	**Hành động**	**Ngõ ra (tick)**
0	reset = 1	r_reg ← 0	Xóa bộ đếm	tick = 0
0 ≤ r_reg < M−1	reset = 0	r_reg ← r_reg + 1	Tăng 1 mỗi chu kỳ xung	tick = 0
r_reg = M−1	–	r_reg ← 0	Reset đếm lại từ đầu	tick = 1 (1 chu kỳ)

Chức năng: Tạo xung "tick" tần số f_tick = f_clk / M

Mô đun SynCounter4bit_WithEnable

Trạng thái (q)	**Điều kiện**	**Trạng thái kế**	**Hành động**	**Ngõ ra (carry)**
q = 0	reset = 1	q ← 0	Khởi tạo	0

$0 \le q <$ max_count	enable = 1	$q \leftarrow q + 1$	Tăng giá trị	0
q = max_count	enable = 1	$q \leftarrow 0$	Quay về 0	1
q = X	enable = 0	$q \leftarrow q$	Giữ nguyên	carry giữ trạng thái

Chức năng: Bộ đếm 4-bit có ngõ vào enable, đếm 0 → max_count → 0.

Mô đun Dem_BCD_0_59

Trạng thái (q1:q0)	**Điều kiện vào (tick)**	**Trạng thái kế**	**Hành động**	**Ngõ ra**
q1,q0 = 00	tick = 0	Giữ nguyên	–	q1,q0
q1,q0 = XY (Y < 9)	tick = 1	$q0 \leftarrow q0 + 1$	Tăng hàng đơn vị	q1:q0+1
q1,q0 = X9 (X < 5)	tick = 1	$q0 \leftarrow 0$; $q1 \leftarrow q1 + 1$	Tăng hàng chục	q1+1:00
q1,q0 = 59	tick = 1	$q1,q0 \leftarrow 00$	Quay vòng	00
Reset	reset = 1	00	Khởi tạo	00

Chức năng: Đếm BCD từ 00 → 59 → 00.

Mô đun Dem_BCD_0_59_05Hz

Trạng thái (r_reg, q1, q0)	Điều kiện	Trạng thái kế	Hành động	Ngõ ra
r_reg < M−1	reset=0	r_reg ← r_reg + 1	Tăng TickGen	tick=0
r_reg = M−1	–	r_reg ← 0, q cập nhật như Dem_BCD_0_59	Tạo tick, cập nhật số BCD	tick=1
reset=1	–	r_reg,q ← 0	Reset toàn bộ	q1,q0 = 00

Chức năng: Kết hợp TickGen 0.5 Hz với Dem_BCD_0_59 → đếm giây/phút.

Mô đun TM1638

Trạng thái (cs)	Miêu tả / Giai đoạn	Điều kiện chuyển	Hành động / Tín hiệu xuất	Ngõ ra (clk, stb, dio)
0	Khởi tạo	cs++	Nạp command 1,2,3,... dữ liệu led	stb=0, clk=1
1–16	Gửi command1	cs++ mỗi xung	Dịch bit lệnh ra dio	dio=command1[0], clk đảo
17	Kết thúc lệnh 1	cs++	Dừng stb=1	stb=1
18	Chuẩn bị lệnh 2	cs++	stb=0	stb=0

19–34	Gửi command2	cs++	Dịch lệnh 2	dio=command2[0]
35–290	Gửi dữ liệu LED/SEG	cs++	Dịch leddatahold 128 byte	dio=leddatahold[0]
291	Kết thúc dữ liệu	cs++	stb=1	stb=1
292	Chuẩn bị lệnh 3	cs++	stb=0	stb=0
293–308	Gửi command3	cs++	Dịch lệnh cuối	dio=command3[0]
309	Kết thúc khung	cs++	stb=1	stb=1
310	Lặp lại chu kỳ	cs ← 0	Bắt đầu lại	tuần hoàn

Chức năng: Truyền nối tiếp 3 lệnh + dữ liệu LED tới TM1638, lặp liên tục.

Mô đun top

Trạng thái / Chức năng	**Điều kiện**	**Tác động / Kết nối**	**Ngõ ra**
–	reset = 1	Reset toàn bộ hệ thống con	–
–	reset = 0	Nhận xung từ ClockDiv → Dem_BCD → TM1638	clk, stb, dio

–	–	Hiển thị giá trị BCD hiện tại trên TM1638	Dữ liệu LED, 7-seg

Mô đum ClockDiv

```
module ClockDiv  //200KHz
(
   input wire clki,
   output wire clko
   );

        wire [26:0] r_next ;
        reg [26:0] r_reg;

        initial r_reg =0 ;

        always @(posedge clki)
                r_reg = r_next;

        assign r_next = r_reg + 1 ;
        assign clko=r_reg[5];

endmodule
```

Mô đum TickGen

```
module TickGen
#(parameter M=50000000)
(
```

```
    input  wire clki,   // 50 MHz
    input  wire reset,
    output wire tick
);
        reg [30:0] r_reg;
        always @(posedge clki or posedge reset) begin
                if (reset)
                        r_reg <= 0;
                else if (r_reg == M-1)
                        r_reg <= 0;
                else
                        r_reg <= r_reg + 1;
        end

        assign tick = (r_reg == M-1);

endmodule
```

Mô đum SynCounter4bit_WithEnable

```
module SynCounter4bit_WithEnable(
    input wire clk,
    input wire reset,
    input wire enable,
```

```
    input wire [3:0] max_count,
    output reg [3:0] q,
    output wire carry
    );

        always @(posedge clk or posedge reset) begin
            if (reset)
                q <= 4'd0;
            else if (enable) begin
                if (q == max_count)
                    q <= 4'd0;
                else
                    q <= q + 1;
            end
        end

        assign carry = (q == max_count);

endmodule
```

Mô đum Dem_BCD_0_59

```
module Dem_BCD_0_59(
    input wire clk,
    input wire reset,
         input wire tick,
    output wire [3:0] q1,    // 0 - 5
    output wire [3:0] q0     // 0 - 9
);
```

```
    wire carry_q0;

    SynCounter4bit_WithEnable count_0_9 (
        .clk(clk),
        .reset(reset),
        .enable(tick),
        .max_count(4'd9),
        .q(q0),
        .carry(carry_q0)
    );

    SynCounter4bit_WithEnable count_5_9 (
        .clk(clk),
        .reset(reset),
        .enable(tick & carry_q0),
        .max_count(4'd5),
        .q(q1),
        .carry()
    );


endmodule
```

Mô đum Dem_BCD_0_59_05Hz

```
module Dem_BCD_0_59_05Hz(
  input clk,
  input reset,
  output [3:0] q1,  // 0 - 5
  output [3:0] q0   // 0 - 9
```

```
    );

        wire tick_05Hz;

        TickGen #(25_000_000) u_tick (.clki(clk), .reset(reset), .tick(tick_05Hz));

        Dem_BCD_0_59 BCD_0_59(.clk(clk), .reset(reset), .tick(tick_05Hz), .q1(q1), .q0(q0));

endmodule
```

Mô đum TM1638

```
module TM1638(
    input wire[7:0] led,
    input wire[3:0] seg7,    // 4 bit data for cathode common LED
    input wire[3:0] seg6,
    input wire[3:0] seg5,
    input wire[3:0] seg4,
    input wire[3:0] seg3,
    input wire[3:0] seg2,
    input wire[3:0] seg1,
    input wire[3:0] seg0,
    input clkinput,
    output reg clk,
    output reg stb,
    output reg dio
    );

        function [7:0] sseg;
        input [3:0] hex;
```

```
        begin
                case (hex)
                        4'h0: sseg[7:0] = 8'b0011_1111;
                        4'h1: sseg[7:0] = 8'b0000_0110;
                        4'h2: sseg[7:0] = 8'b0101_1011;
                        4'h3: sseg[7:0] = 8'b0100_1111;
                        4'h4: sseg[7:0] = 8'b0110_0110;
                        4'h5: sseg[7:0] = 8'b0110_1101;
                        4'h6: sseg[7:0] = 8'b0111_1101;
                        4'h7: sseg[7:0] = 8'b0000_0111;
                        4'h8: sseg[7:0] = 8'b0111_1111;
                        4'h9: sseg[7:0] = 8'b0110_1111;
                        4'hA: sseg[7:0] = 8'b0111_0111;
                        4'hB: sseg[7:0] = 8'b0111_1100;
                        4'hC: sseg[7:0] = 8'b0101_1000;
                        4'hD: sseg[7:0] = 8'b0101_1110;
                        4'hE: sseg[7:0] = 8'b0111_1001;
                        default : sseg[7:0] = 8'b0000000; // 4'hF
                endcase
        end
        endfunction

        integer cs = 0;

        reg [7:0] command1 =8'h40, command2 =8'hC0,command3 =8'h8F;
        wire [127:0] leddata; // 1,3,5,7,9,11,13,15: single led; 0,2,4,6,8,10,12,14: seg LED
(common cathode)
        reg [127:0] leddatahold;
```

```
assign leddata[0*8+7:0*8+0] = sseg(seg0);
assign leddata[2*8+7:2*8+0] = sseg(seg1);
assign leddata[4*8+7:4*8+0] = sseg(seg2);
assign leddata[6*8+7:6*8+0] = sseg(seg3);
assign leddata[8*8+7:8*8+0] = sseg(seg4);
assign leddata[10*8+7:10*8+0] = sseg(seg5);
assign leddata[12*8+7:12*8+0] = sseg(seg6);
assign leddata[14*8+7:14*8+0] = sseg(seg7);

// Single LED data (odd bytes: 1,3,5,7,9,11,13,15)
assign leddata[1*8+7:1*8+0]   = {7'b0000000, led[0]};
assign leddata[3*8+7:3*8+0]   = {7'b0000000, led[1]};
assign leddata[5*8+7:5*8+0]   = {7'b0000000, led[2]};
assign leddata[7*8+7:7*8+0]   = {7'b0000000, led[3]};
assign leddata[9*8+7:9*8+0]   = {7'b0000000, led[4]};
assign leddata[11*8+7:11*8+0] = {7'b0000000, led[5]};
assign leddata[13*8+7:13*8+0] = {7'b0000000, led[6]};
assign leddata[15*8+7:15*8+0] = {7'b0000000, led[7]};

initial begin
                clk = 1 ;
                stb = 1 ;
                dio = 0 ;
end

always @(posedge clkinput) begin
        if (cs==0) begin
                stb = 0; // initial tm1638
                command1 =8'h40;
```

```
        command2 =8'hC0;
        command3 =8'h8F;
        leddatahold = leddata ;
end
else if ((cs >=1)&&(cs<=16)) begin
        dio = command1[0];
        clk = ~clk ;
        if (clk) command1=command1>>1 ;
end
else if (cs==17)
        stb = 1; // stop tm1638
else if (cs==18)
        stb = 0; // ready to send the second command

// send second command
else if ((cs >=19)&&(cs<=34)) begin
        dio = command2[0];
        clk = ~clk ;
if (clk) command2=command2>>1 ;
end
else if ((cs >=35)&&(cs<=290)) begin
        dio = leddatahold[0];
        clk = ~clk ;
        if (clk) leddatahold=leddatahold>>1 ;
end
else if (cs==291)
        stb = 1; // stop tm1638 for end of data
else if (cs==292)
        stb = 0; // ready to send the third command
```

```
            // send last command
            else if ((cs >=293)&&(cs<=308)) begin
                dio = command3[0];
                clk = ~clk ;
            if (clk) command3=command3>>1 ;
            end
            else if (cs==309)
                stb = 1; // End
            else if (cs==310)
                cs = -1 ; //repeat

            // update cs
            cs=cs+1;
        end

endmodule
```

Mô đum top

```
module top(
    input clk_50M,
    input wire reset,
    output wire clk,
    output wire stb,
    output wire dio
    );

        wire clko;
        wire [3:0] count_chuc;   // 0 - 5
```

```
        wire [3:0] count_donvi;  // 0 - 9

        ClockDiv clock (.clki(clk_50M), .clko(clko));

        Dem_BCD_0_59_05Hz counter (
                .clk(clk_50M),
                .reset(reset),
                .q1(count_chuc),   // 0 - 5
                .q0(count_donvi)   // 0 - 9
    );

        TM1638 tm (
                .led(0),
                .seg7(4'd0),
                .seg6(4'd0),
                .seg5(4'd0),
                .seg4(4'd0),
                .seg3(4'd0),
                .seg2(4'd0),
                .seg1(count_donvi),
                .seg0(count_chuc),
                .clkinput(clko),
                .clk(clk),
                .stb(stb),
                .dio(dio)
        );

endmodule
```

4. Test Cases Overview

TC ID	Purpose	Input	Expected Output	Result (Pass/Fail)
TC1	Kiểm tra reset	reset = 1	Mạch bị reset về 0	Pass
TC2	Kiểm tra đếm BCD	clk = ~clk reset = 0	Mạch đếm lên	Pass
TC3	Kiểm tra khi quay về 0	clk = ~clk reset = 0	Mạch đếm về 0	Pass

Mô đum Testbench_Dem_BCD_0_59

```
`timescale 1ns / 1ps

module Testbench_Dem_BCD_0_59;

	// Inputs
	reg clk;
	reg reset;
	reg tick;

	// Outputs
	wire [3:0] q1;
	wire [3:0] q0;

	// Instantiate the Unit Under Test (UUT)
	Dem_BCD_0_59 uut (
		.clk(clk),
		.reset(reset),
```

```
        .tick(tick),
        .q1(q1),
        .q0(q0)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        reset = 0;
        tick = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        reset = 1;
        #100;
        reset = 0;

    end

    always forever #10 clk = ~clk;
    always forever #20 tick = ~tick;

endmodule
```

5. Testcase Details

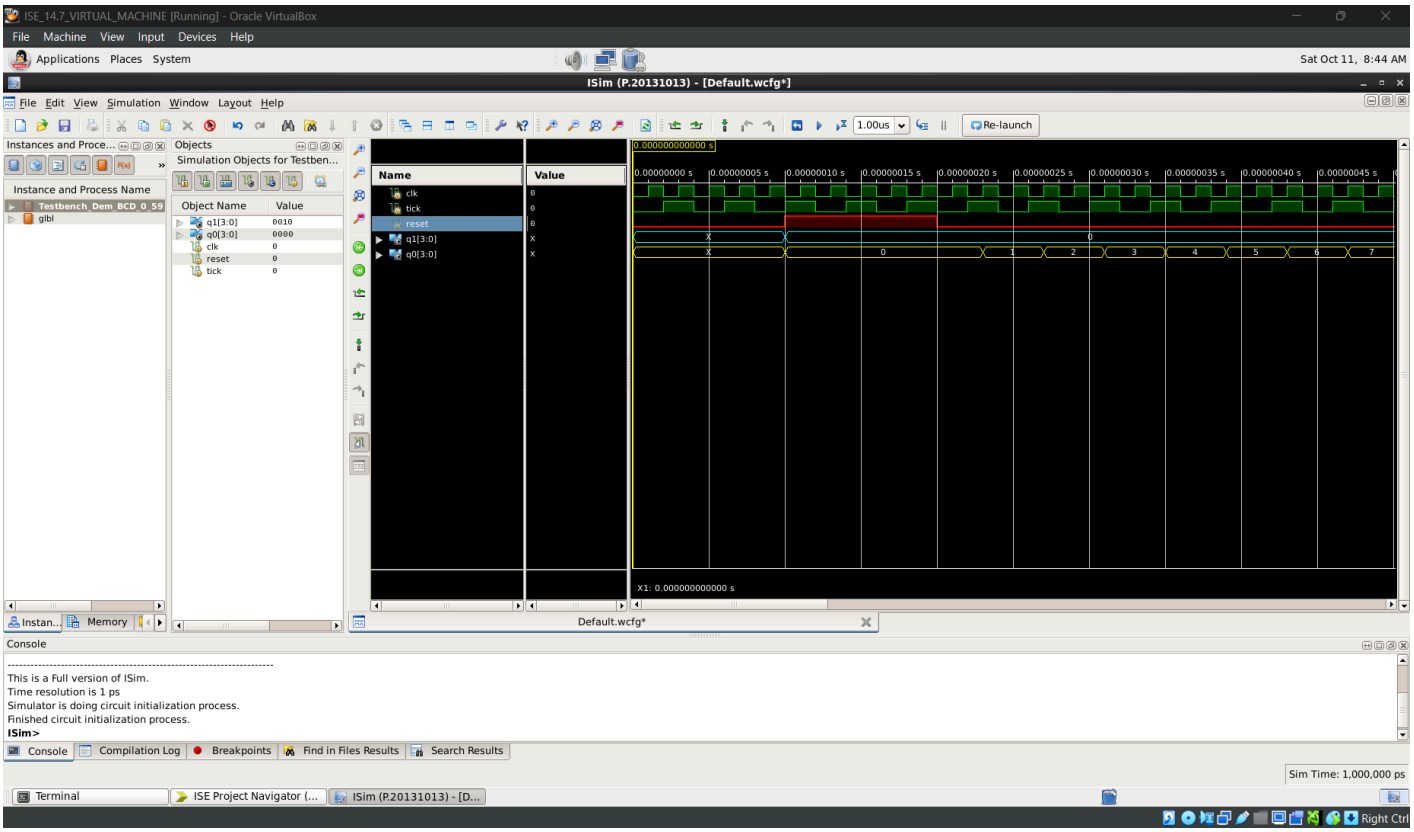
Testcase 1: TC1

Purpose: Kiểm tra reset

Input/Stimulus: reset = 1

Expected Output: Mạch bị reset về 0

Waveform Simulation (attach/insert image):



Analysis:

Kết quả mô phỏng đúng yêu cầu bảng trạng thái

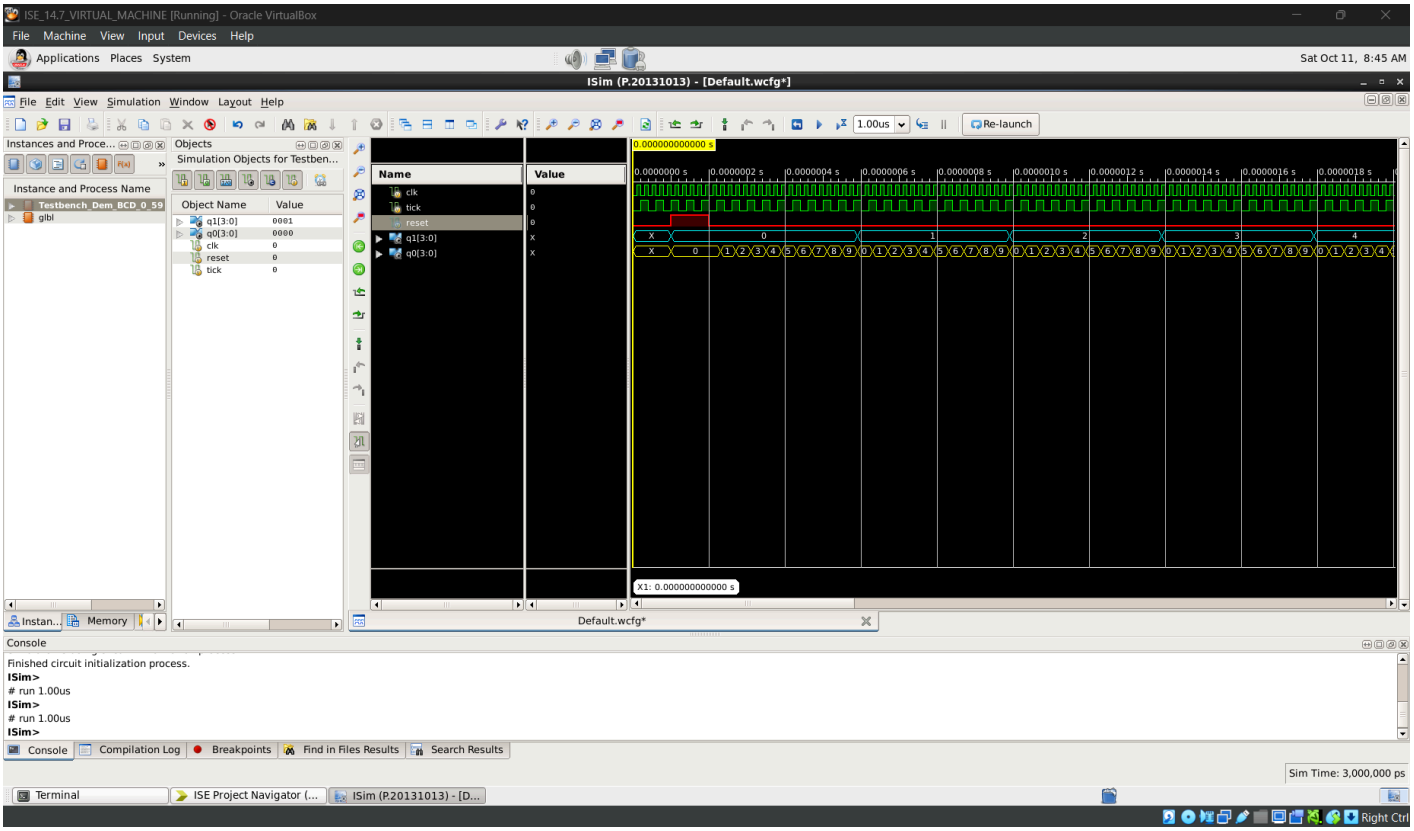
Testcase 2: TC2

Purpose: Kiểm tra đếm BCD

Input/Stimulus: clk = ~clk, reset = 0, tick = 1

Expected Output: Mạch đếm lên

Waveform Simulation (attach/insert image):



Analysis:

Kết quả mô phỏng đúng yêu cầu bảng trạng thái

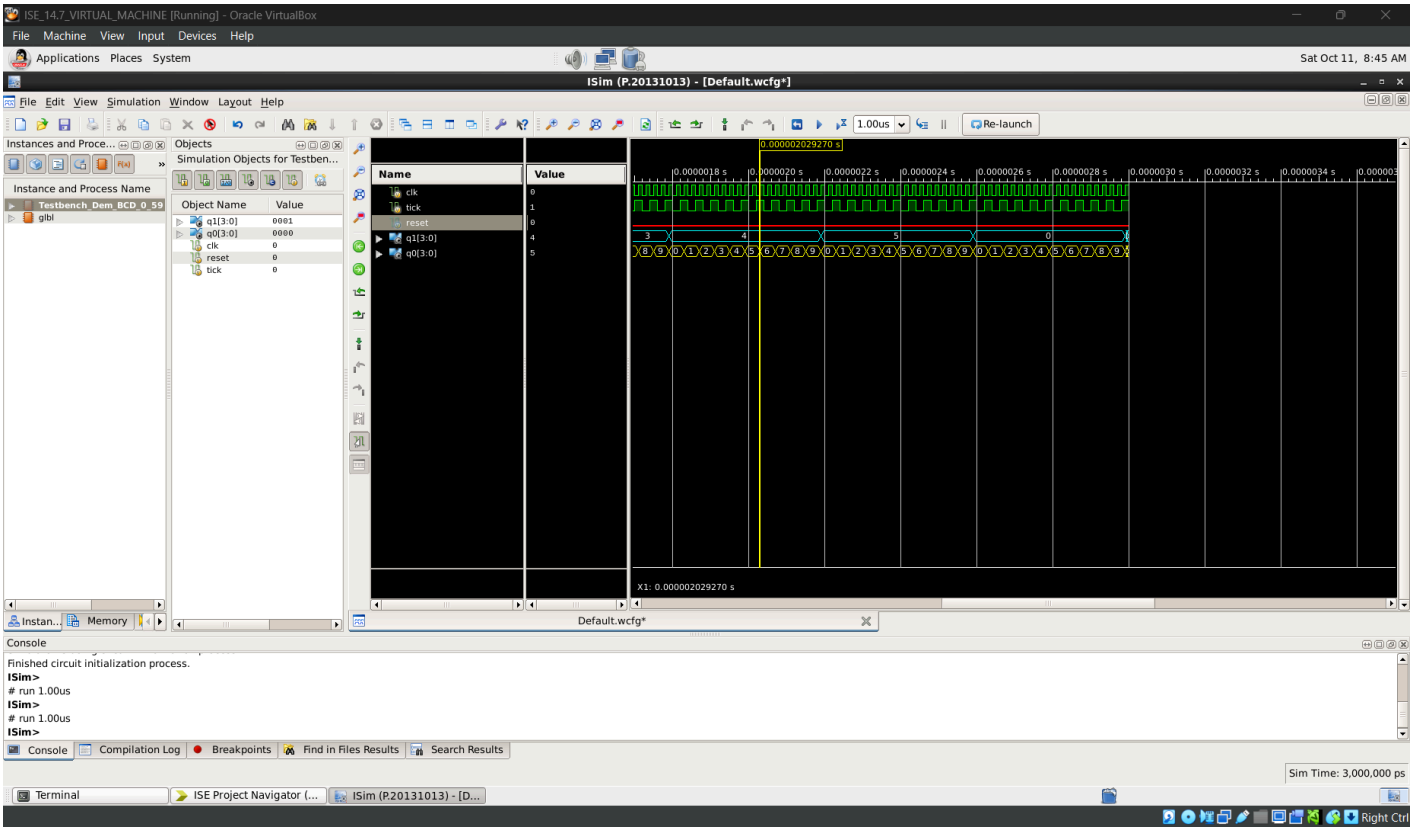
Testcase 3: TC3

Purpose: Kiểm tra khi quay về 0

Input/Stimulus: clk = ~clk, reset = 0, tick = 1

Expected Output: Mạch đếm về 0

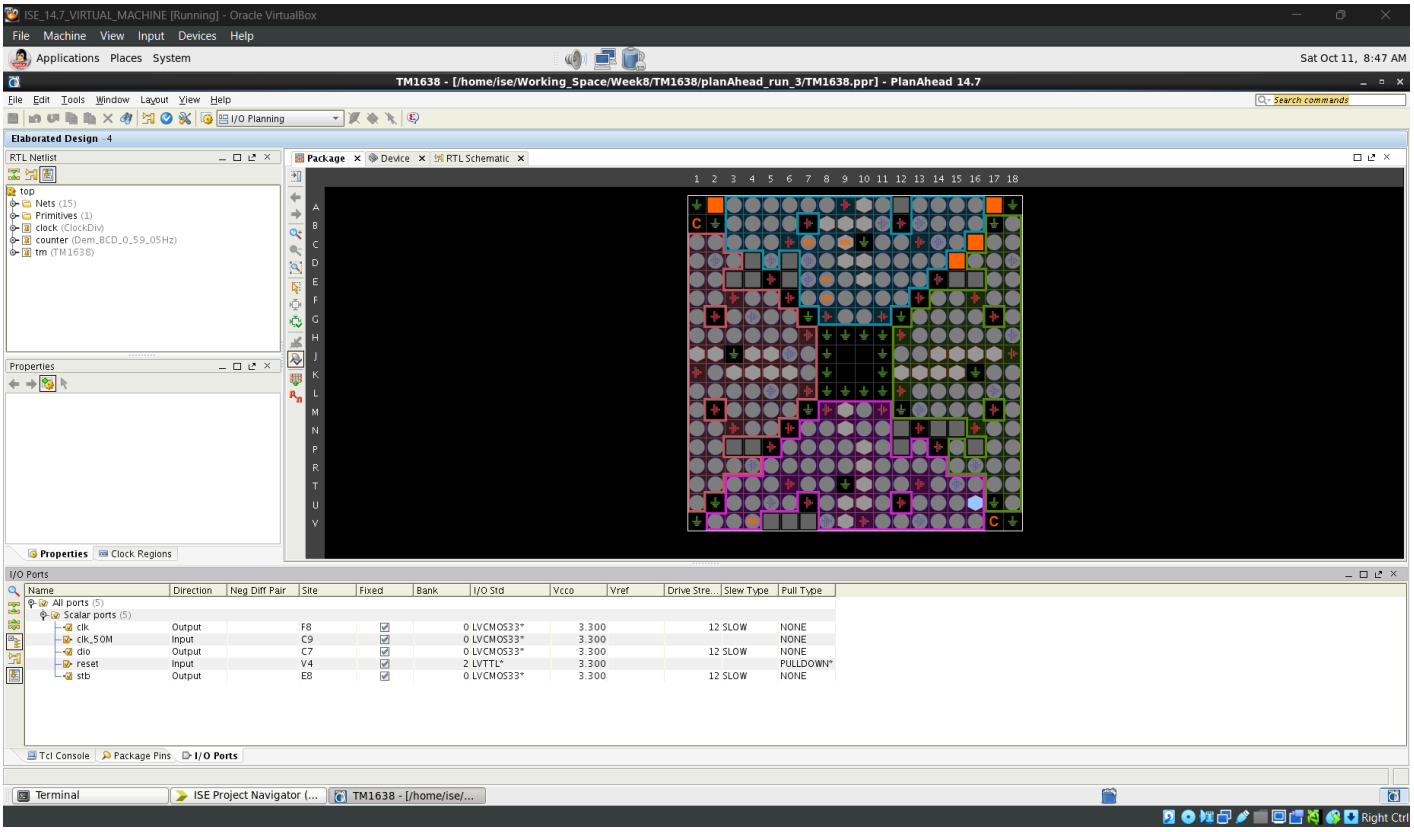
Waveform Simulation (attach/insert image):



Analysis:

Kết quả mô phỏng đúng yêu cầu bảng trạng thái

6. Summary




```
NET "clk_50M" LOC = C9;
NET "stb" LOC = E8;
NET "clk" LOC = F8;
NET "dio" LOC = C7;


NET "clk" IOSTANDARD = LVCMOS33;
NET "clk_50M" IOSTANDARD = LVCMOS33;
NET "dio" IOSTANDARD = LVCMOS33;
NET "stb" IOSTANDARD = LVCMOS33;

# PlanAhead Generated physical constraints

NET "reset" LOC = V4;
```

```
# PlanAhead Generated IO constraints

NET "reset" IOSTANDARD = LVTTL;
NET "reset" PULLDOWN;
```

WEEKLY REPORT – DIGITAL SYSTEM DESIGN USING VERILOG & FPGA

1. General Information

Project Title: LED SÁNG DỒN HIỂN THỊ TRÊN TM1638

Students 1: Lê Quang Minh Nhật

Students 2: Nguyễn Quốc Hưng

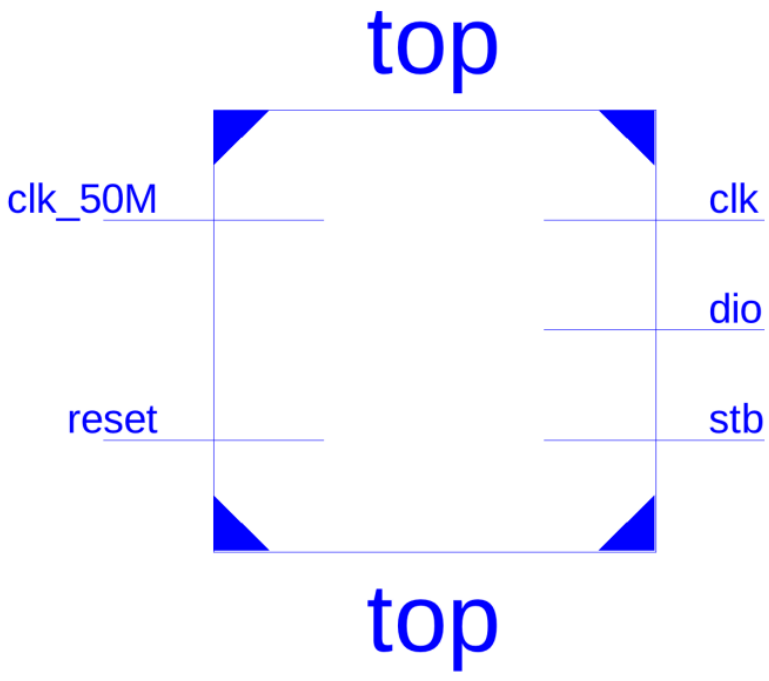
Students ID 1: 23139031

Students ID 2: 23139019

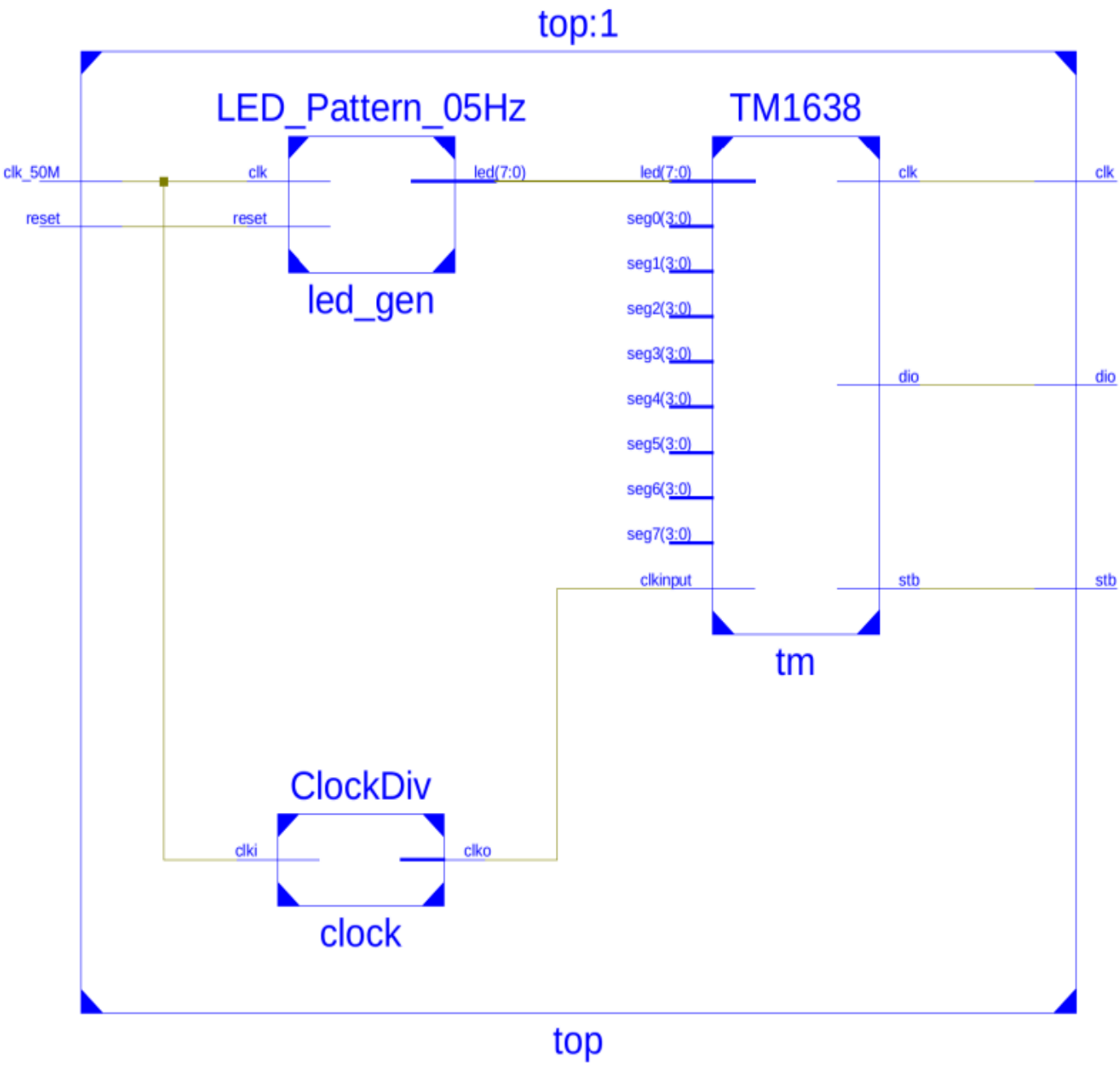
Date: 11/10/2025

2. Block Diagram

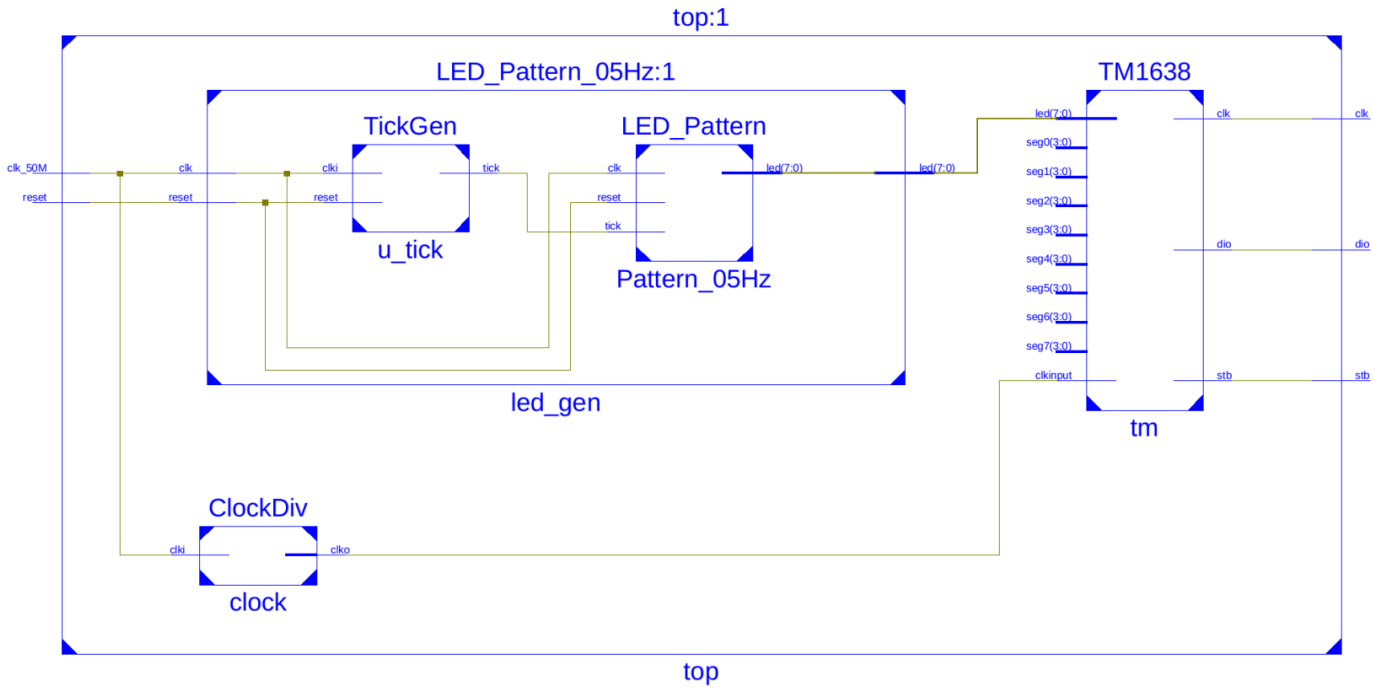
Draw block diagrams here.



Sơ đồ khối tổng quát



Sơ đồ khối điều khiển LED và TM1638



Sơ đồ chi tiết các khối

Cấu trúc chương trình

3. State Transition Diagram

Describe Truth table or Design requirements or the FSM state diagram, etc

Mô-đun ClockDiv

Trạng thái hiện tại	Điều kiện	Trạng thái kế tiếp	Hành động	Ngõ ra (clko)
r_reg = 0 (Mặc định)	reset = 1	r_reg ← 0	Khởi tạo đếm	r_reg[5] = 0

r_reg = N (0 ≤ N < $2^{27}-1$)	reset = 0	r_reg ← N + 1	Tăng bộ đếm	clko = bit thứ 5 của r_reg
–	–	–	–	Tín hiệu clko dao động

Mô đun TickGen

Trạng thái (r_reg)	**Điều kiện vào**	**Chuyển trạng thái**	**Hành động**	**Ngõ ra (tick)**
0	reset = 1	r_reg ← 0	Xóa bộ đếm	tick = 0
0 ≤ r_reg < M−1	reset = 0	r_reg ← r_reg + 1	Tăng 1 mỗi chu kỳ xung	tick = 0
r_reg = M−1	–	r_reg ← 0	Reset đếm lại từ đầu	tick = 1 (1 chu kỳ)

Chức năng: Tạo xung "tick" tần số f_tick = f_clk / M

Mô đun LED_Pattern

Trạng thái hiện tại	**Điều kiện (ngõ vào)**	**Trạng thái kế tiếp**	**Hành động & Ngõ ra**
state = 0	reset = 1	state ← 0	led ← 8'b0000_0000
state = S (0…35)	tick = 1 & posedge clk	state ← S + 1	led ← pattern[S]

state = 36	tick = 1 & posedge clk	state ← 0	led ← pattern[36] (trước khi sang 0)
bất kỳ	tick = 0	giữ nguyên	led giữ theo pattern[state]

Chức năng: ROM pattern[0…36] đã nạp ở initial. Cập nhật chỉ khi có tick.

Mô đun LED_Pattern_05Hz (TickGen + LED_Pattern)

Thành phần	**Trạng thái hiện tại**	**Điều kiện**	**Trạng thái kế tiếp**	**Hành động & Ngõ ra**
TickGen (M=25e6)	r_reg	posedge clk/reset	như bảng TickGen	Tạo tick_05Hz (thực tế 2 Hz)
LED_Pattern	state	tick_05Hz & posedge clk/reset	như bảng LED_Pattern	led = pattern[state]

Mô đun TM1638

Trạng thái (cs)	**Miêu tả / Giai đoạn**	**Điều kiện chuyển**	**Hành động / Tín hiệu xuất**	**Ngõ ra (clk, stb, dio)**
0	Khởi tạo	cs++	Nạp command 1,2,3,... dữ liệu led	stb=0, clk=1
1–16	Gửi command1	cs++ mỗi xung	Dịch bit lệnh ra dio	dio=command1[0], clk đảo
17	Kết thúc lệnh 1	cs++	Dừng stb=1	stb=1

18	Chuẩn bị lệnh 2	cs++	stb=0	stb=0
19–34	Gửi command2	cs++	Dịch lệnh 2	dio=command2[0]
35–290	Gửi dữ liệu LED/SEG	cs++	Dịch leddatahold 128 byte	dio=leddatahold[0]
291	Kết thúc dữ liệu	cs++	stb=1	stb=1
292	Chuẩn bị lệnh 3	cs++	stb=0	stb=0
293–308	Gửi command3	cs++	Dịch lệnh cuối	dio=command3[0]
309	Kết thúc khung	cs++	stb=1	stb=1
310	Lặp lại chu kỳ	cs ← 0	Bắt đầu lại	tuần hoàn

Chức năng: Truyền nối tiếp 3 lệnh + dữ liệu LED tới TM1638, lặp liên tục.

Mô đun top

Trạng thái / Chức năng	**Điều kiện**	**Tác động / Kết nối**	**Ngõ ra**
–	reset = 1	Reset toàn bộ hệ thống con	–
–	reset = 0	Nhận xung từ ClockDiv → LED_Pattern → TM1638	clk, stb, dio

–	–	Hiển thị giá trị led hiện tại trên TM1638	Dữ liệu LED, 7-seg

Mô đum ClockDiv

```
module ClockDiv  //200KHz
(
   input wire clki,
   output wire clko
   );

        wire [26:0] r_next ;
        reg [26:0] r_reg;

        initial r_reg =0 ;

        always @(posedge clki)
                r_reg = r_next;

        assign r_next = r_reg + 1 ;
        assign clko=r_reg[5];

endmodule
```

Mô đum TickGen

```
module TickGen
#(parameter M=50000000)
(
   input  wire clki,   // 50 MHz
```

```
    input  wire reset,
    output wire tick
);
        reg [30:0] r_reg;
        always @(posedge clki or posedge reset) begin
                if (reset)
                        r_reg <= 0;
                else if (r_reg == M-1)
                        r_reg <= 0;
                else
                        r_reg <= r_reg + 1;
        end

        assign tick = (r_reg == M-1);

endmodule
```

Mô đum LED_Pattern

```
module LED_Pattern(
    input wire clk,
    input wire reset,
          input wire tick,
    output reg [7:0] led
    );

        reg [7:0] pattern [0:36];

        initial begin
       pattern[0]  = 8'b0000_0000;
```

```
pattern[1]  = 8'b1000_0000;
pattern[2]  = 8'b0100_0000;
pattern[3]  = 8'b0010_0000;
pattern[4]  = 8'b0001_0000;
pattern[5]  = 8'b0000_1000;
pattern[6]  = 8'b0000_0100;
pattern[7]  = 8'b0000_0010;
pattern[8]  = 8'b0000_0001;
pattern[9]  = 8'b1000_0001;
pattern[10] = 8'b0100_0001;
pattern[11] = 8'b0010_0001;
pattern[12] = 8'b0001_0001;
pattern[13] = 8'b0000_1001;
pattern[14] = 8'b0000_0101;
pattern[15] = 8'b0000_0011;
pattern[16] = 8'b1000_0011;
pattern[17] = 8'b0100_0011;
pattern[18] = 8'b0010_0011;
pattern[19] = 8'b0001_0011;
pattern[20] = 8'b0000_1011;
pattern[21] = 8'b0000_0111;
pattern[22] = 8'b1000_0111;
pattern[23] = 8'b0100_0111;
pattern[24] = 8'b0010_0111;
pattern[25] = 8'b0001_0111;
pattern[26] = 8'b0000_1111;
pattern[27] = 8'b1000_1111;
pattern[28] = 8'b0100_1111;
pattern[29] = 8'b0010_1111;
```

```
        pattern[30] = 8'b0001_1111;
        pattern[31] = 8'b1001_1111;
        pattern[32] = 8'b0101_1111;
        pattern[33] = 8'b0011_1111;
        pattern[34] = 8'b1011_1111;
        pattern[35] = 8'b0111_1111;
        pattern[36] = 8'b1111_1111;
          end

          reg [5:0] state;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= 6'd0;
            led <= 8'b00000000;
        end
        else if (tick) begin
            if (state == 6'd36)
                state <= 6'd0;
            else
                state <= state + 1;

            led <= pattern[state];
        end
    end

endmodule
```

Mô đum LED_Pattern_05Hz

```
module LED_Pattern_05Hz(
    input wire clk,
    input wire reset,
    output wire [7:0] led
    );

        wire tick_05Hz;

        TickGen #(25_000_000) u_tick(
                .clki(clk),
                .reset(reset),
                .tick(tick_05Hz)
    );

        LED_Pattern Pattern_05Hz(.clk(clk), .reset(reset),.tick(tick_05Hz), .led(led));

endmodule
```

Mô đum TM1638

```
module TM1638(
    input wire[7:0] led,
    input wire[3:0] seg7,    // 4 bit data for cathode common LED
    input wire[3:0] seg6,
    input wire[3:0] seg5,
    input wire[3:0] seg4,
    input wire[3:0] seg3,
    input wire[3:0] seg2,
    input wire[3:0] seg1,
```

```
input wire[3:0] seg0,
input clkinput,
output reg clk,
output reg stb,
output reg dio
);

    function [7:0] sseg;
    input [3:0] hex;
    begin
        case (hex)
            4'h0: sseg[7:0] = 8'b0011_1111;
            4'h1: sseg[7:0] = 8'b0000_0110;
            4'h2: sseg[7:0] = 8'b0101_1011;
            4'h3: sseg[7:0] = 8'b0100_1111;
            4'h4: sseg[7:0] = 8'b0110_0110;
            4'h5: sseg[7:0] = 8'b0110_1101;
            4'h6: sseg[7:0] = 8'b0111_1101;
            4'h7: sseg[7:0] = 8'b0000_0111;
            4'h8: sseg[7:0] = 8'b0111_1111;
            4'h9: sseg[7:0] = 8'b0110_1111;
            4'hA: sseg[7:0] = 8'b0111_0111;
            4'hB: sseg[7:0] = 8'b0111_1100;
            4'hC: sseg[7:0] = 8'b0101_1000;
            4'hD: sseg[7:0] = 8'b0101_1110;
            4'hE: sseg[7:0] = 8'b0111_1001;
            default : sseg[7:0] = 8'b0000000; // 4'hF
        endcase
    end
```

```
        endfunction

        integer cs = 0;

        reg [7:0] command1 =8'h40, command2 =8'hC0,command3 =8'h8F;
        wire [127:0] leddata; // 1,3,5,7,9,11,13,15: single led; 0,2,4,6,8,10,12,14: seg LED
(common cathode)
        reg [127:0] leddatahold;

        assign leddata[0*8+7:0*8+0] = sseg(seg0);
        assign leddata[2*8+7:2*8+0] = sseg(seg1);
        assign leddata[4*8+7:4*8+0] = sseg(seg2);
        assign leddata[6*8+7:6*8+0] = sseg(seg3);
        assign leddata[8*8+7:8*8+0] = sseg(seg4);
        assign leddata[10*8+7:10*8+0] = sseg(seg5);
        assign leddata[12*8+7:12*8+0] = sseg(seg6);
        assign leddata[14*8+7:14*8+0] = sseg(seg7);

        // Single LED data (odd bytes: 1,3,5,7,9,11,13,15)
        assign leddata[1*8+7:1*8+0]   = {7'b0000000, led[0]};
        assign leddata[3*8+7:3*8+0]   = {7'b0000000, led[1]};
        assign leddata[5*8+7:5*8+0]   = {7'b0000000, led[2]};
        assign leddata[7*8+7:7*8+0]   = {7'b0000000, led[3]};
        assign leddata[9*8+7:9*8+0]   = {7'b0000000, led[4]};
        assign leddata[11*8+7:11*8+0] = {7'b0000000, led[5]};
        assign leddata[13*8+7:13*8+0] = {7'b0000000, led[6]};
        assign leddata[15*8+7:15*8+0] = {7'b0000000, led[7]};

        initial begin
```

```
                clk = 1 ;
                stb = 1 ;
                dio = 0 ;
end

always @(posedge clkinput) begin
        if (cs==0) begin
                stb = 0; // initial tm1638
                command1 =8'h40;
                command2 =8'hC0;
                command3 =8'h8F;
                leddatahold = leddata ;
        end
        else if ((cs >=1)&&(cs<=16)) begin
                dio = command1[0];
                clk = ~clk ;
                if (clk) command1=command1>>1 ;
        end
        else if (cs==17)
                stb = 1; // stop tm1638
        else if (cs==18)
                stb = 0; // ready to send the second command

        // send second command
        else if ((cs >=19)&&(cs<=34)) begin
                dio = command2[0];
                clk = ~clk ;
        if (clk) command2=command2>>1 ;
        end
```

```
            else if ((cs >=35)&&(cs<=290)) begin
                dio = leddatahold[0];
                clk = ~clk ;
                if (clk) leddatahold=leddatahold>>1 ;
            end
            else if (cs==291)
                stb = 1; // stop tm1638 for end of data
            else if (cs==292)
                stb = 0; // ready to send the third command

            // send last command
            else if ((cs >=293)&&(cs<=308)) begin
                dio = command3[0];
                clk = ~clk ;
            if (clk) command3=command3>>1 ;
            end
            else if (cs==309)
                stb = 1; // End
            else if (cs==310)
                cs = -1 ; //repeat

            // update cs
            cs=cs+1;
        end

endmodule
```

Mô đum top

```
module top(
```

```
input clk_50M,
input wire reset,
output wire clk,
output wire stb,
output wire dio
);

    wire clko;
    wire [7:0] led_pattern;

    ClockDiv clock (.clki(clk_50M), .clko(clko));

    // LED Pattern Generator
    LED_Pattern_05Hz led_gen (
        .clk(clk_50M),
        .reset(reset),
        .led(led_pattern)
    );

    TM1638 tm (
        .led(led_pattern),
        .seg7(4'd0),
        .seg6(4'd0),
        .seg5(4'd0),
        .seg4(4'd0),
        .seg3(4'd0),
        .seg2(4'd0),
        .seg1(4'd0),
        .seg0(4'd0),
```

```
            .clkinput(clko),
            .clk(clk),
            .stb(stb),
            .dio(dio)
        );

endmodule
```

4. Test Cases Overview

TC ID	Purpose	Input	Expected Output	Result (Pass/Fail)
TC1	Kiểm tra reset	reset = 1	Mạch bị reset về 0	Pass
TC2	Mạch sáng dồn	reset = 0 clk = ~clk	Mạch sáng dồn Từng LED	Pass
TC3	Mạch trở về 0	reset = 0 clk = ~clk	Mạch trở về 0	Pass

Mô đun Testbench_LED_Pattern

```
`timescale 1ns / 1ps

module Testbench_LED_Pattern;

        // Inputs
        reg clk;
        reg reset;
        reg tick;

        // Outputs
        wire [7:0] led;
```

```
        // Instantiate the Unit Under Test (UUT)
        LED_Pattern uut (
                .clk(clk),
                .reset(reset),
                .tick(tick),
                .led(led)
        );

        initial begin
                // Initialize Inputs
                clk = 0;
                reset = 0;
                tick = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
                reset = 1;
                #100;
                reset = 0;

        end

                always forever #10 clk = ~clk;
                always forever #20 tick = ~tick;

endmodule
```

5. Testcase Details

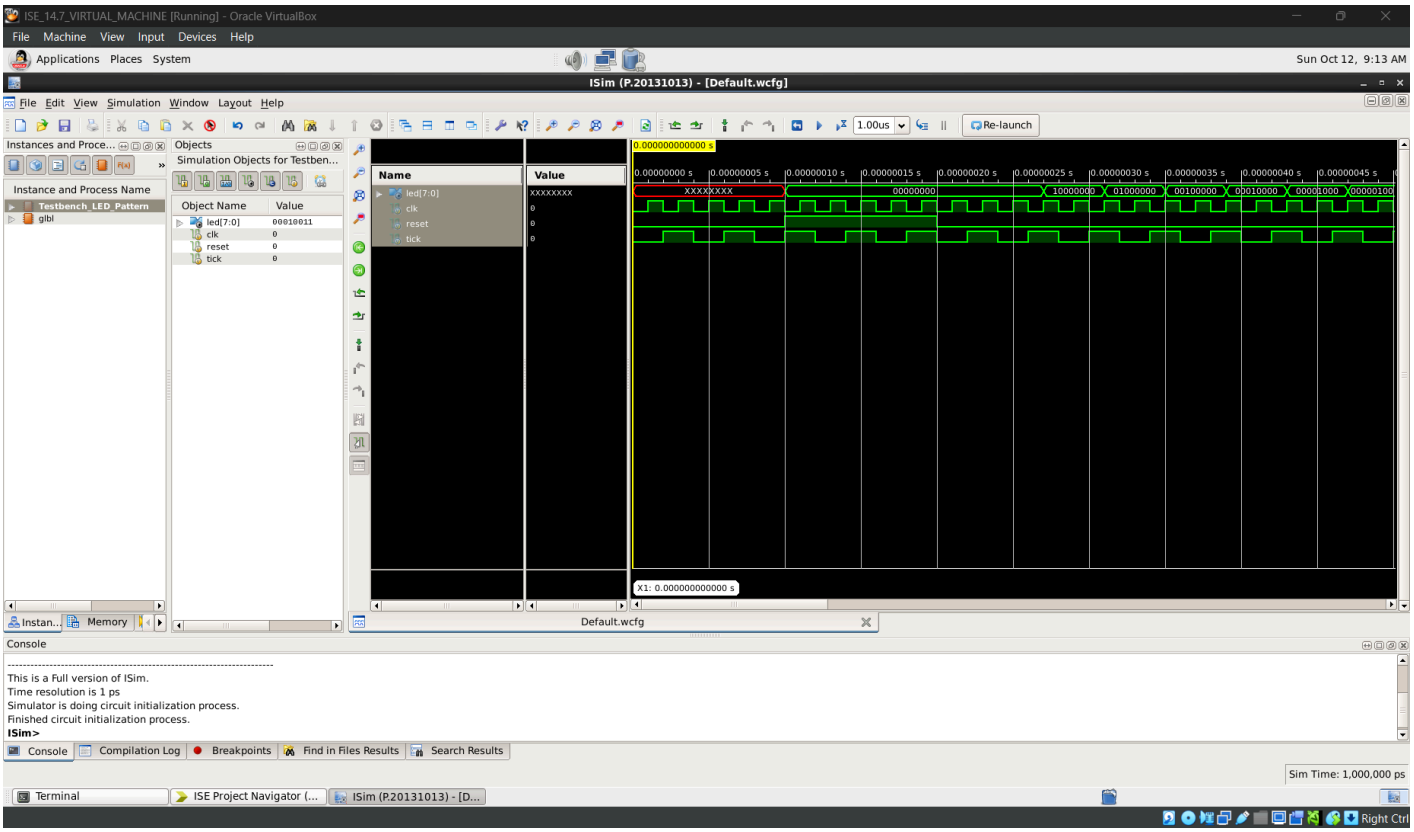
Testcase 1: TC1

Purpose: Kiểm tra reset

Input/Stimulus: reset = 1

Expected Output: Mạch bị reset về 0

Waveform Simulation (attach/insert image):



Analysis:

Mạch bị reset về 0 khi nhấn reset và bắt đầu trạng thái

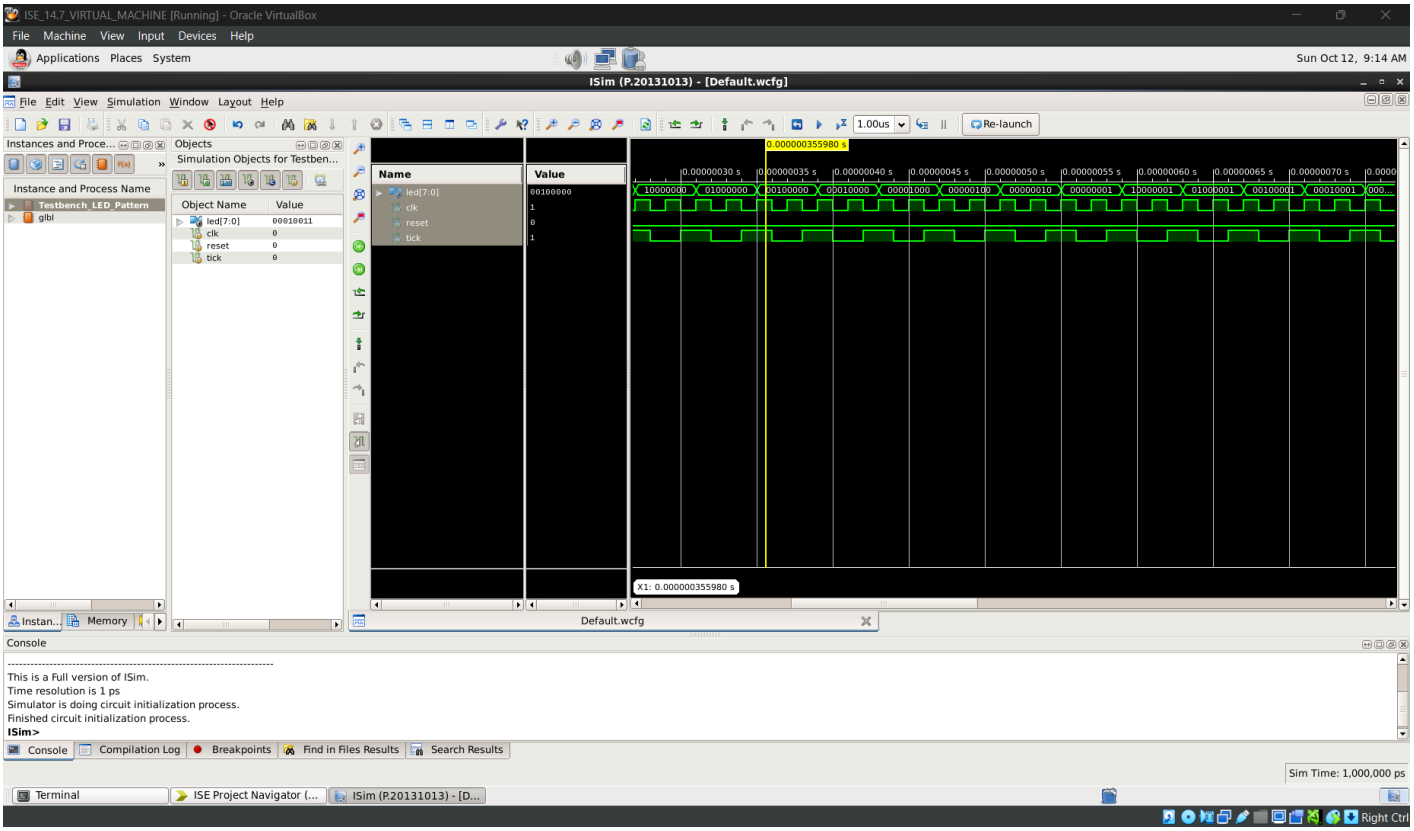
Testcase 2: TC2

Purpose: Mạch sáng dồn

Input/Stimulus: reset = 0, clk = ~clk

Expected Output: Mạch sáng dồn Từng LED

Waveform Simulation (attach/insert image):



Analysis:

Mạch thực hiện sáng dồn từng led 1 tới trạng thái 8'b1111_1111.

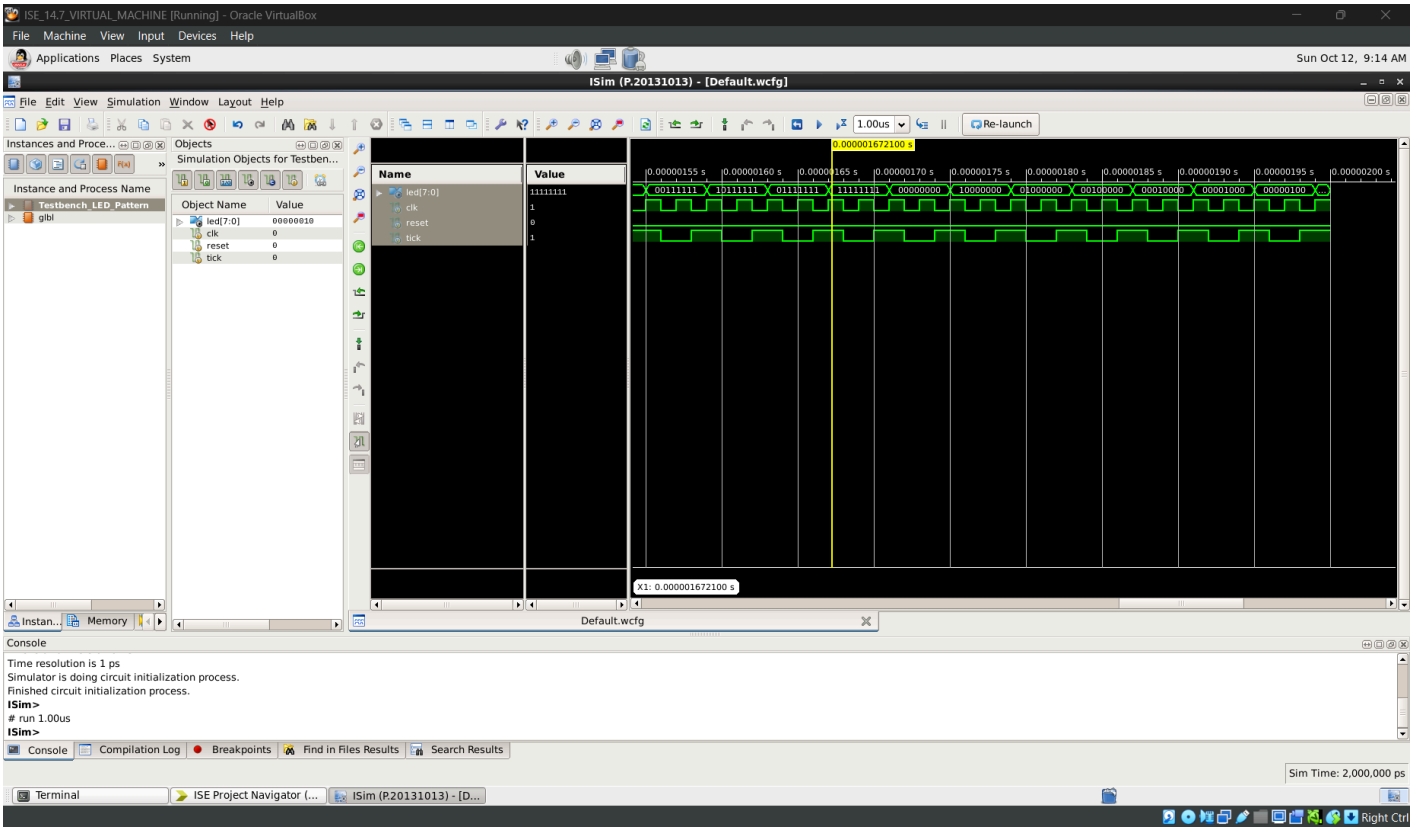
Testcase 3: TC3

Purpose: Mạch trở về 0

Input/Stimulus: reset = 0, clk = ~clk

Expected Output: Mạch trở về 0

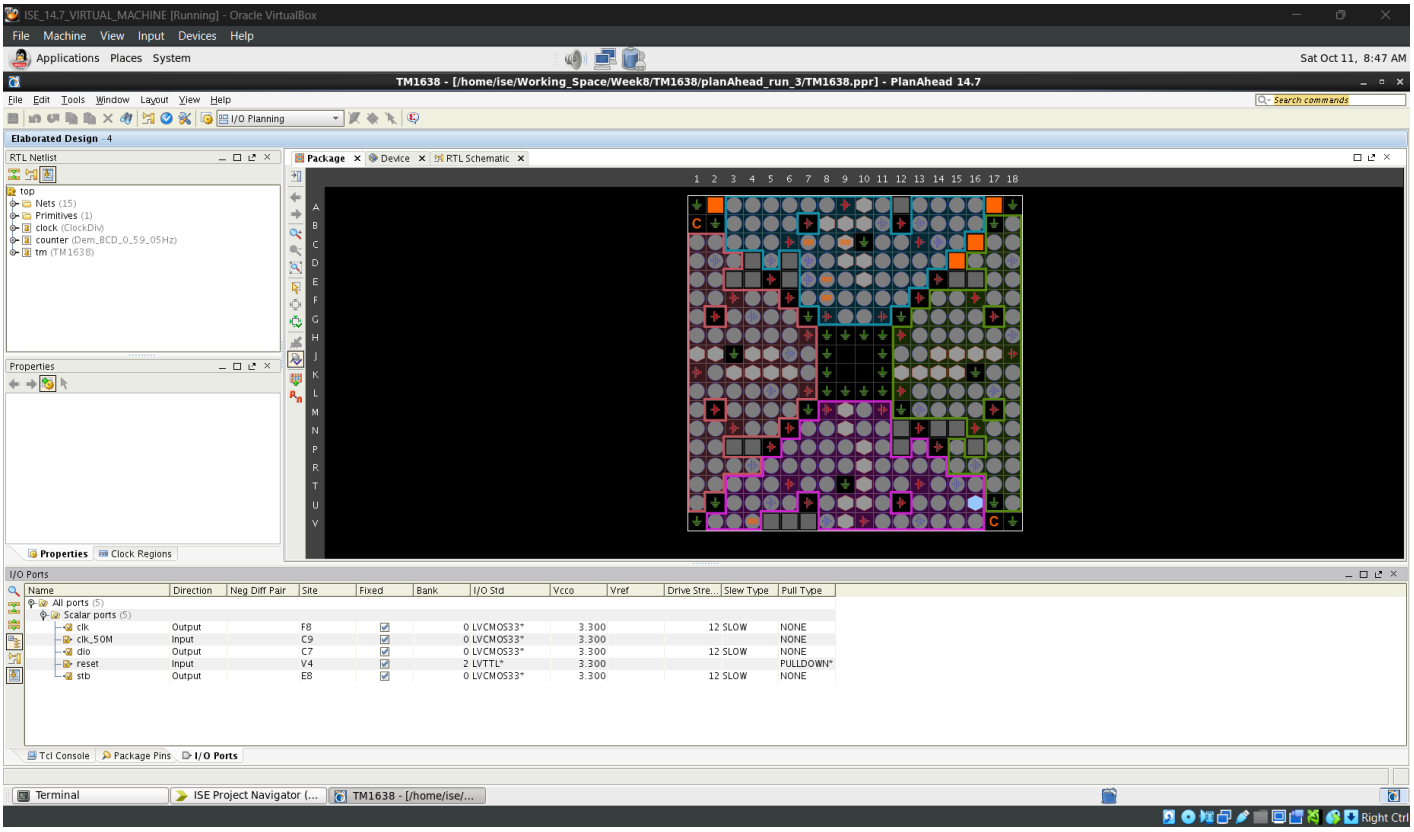
Waveform Simulation (attach/insert image):



Analysis:

Mạch từ 8’b1111_1111 về 8’b0000_0000.

6. Summary




```
NET "clk_50M" LOC = C9;
NET "stb" LOC = E8;
NET "clk" LOC = F8;
NET "dio" LOC = C7;


NET "clk" IOSTANDARD = LVCMOS33;
NET "clk_50M" IOSTANDARD = LVCMOS33;
NET "dio" IOSTANDARD = LVCMOS33;
NET "stb" IOSTANDARD = LVCMOS33;

# PlanAhead Generated physical constraints

NET "reset" LOC = V4;
```

```
# PlanAhead Generated IO constraints

NET "reset" IOSTANDARD = LVTTL;
NET "reset" PULLDOWN;
```

WEEKLY REPORT – DIGITAL SYSTEM DESIGN USING VERILOG & FPGA

1. General Information

Project Title: Điều khiển LCD hiển thị tên

Students 1: Lê Quang Minh Nhật

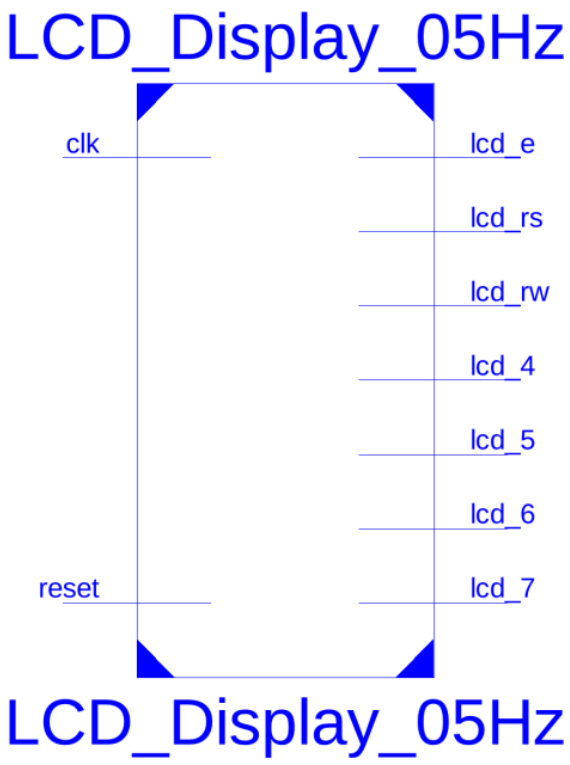
Students 2: Nguyễn Quốc Hưng

Students ID 1: 23139031

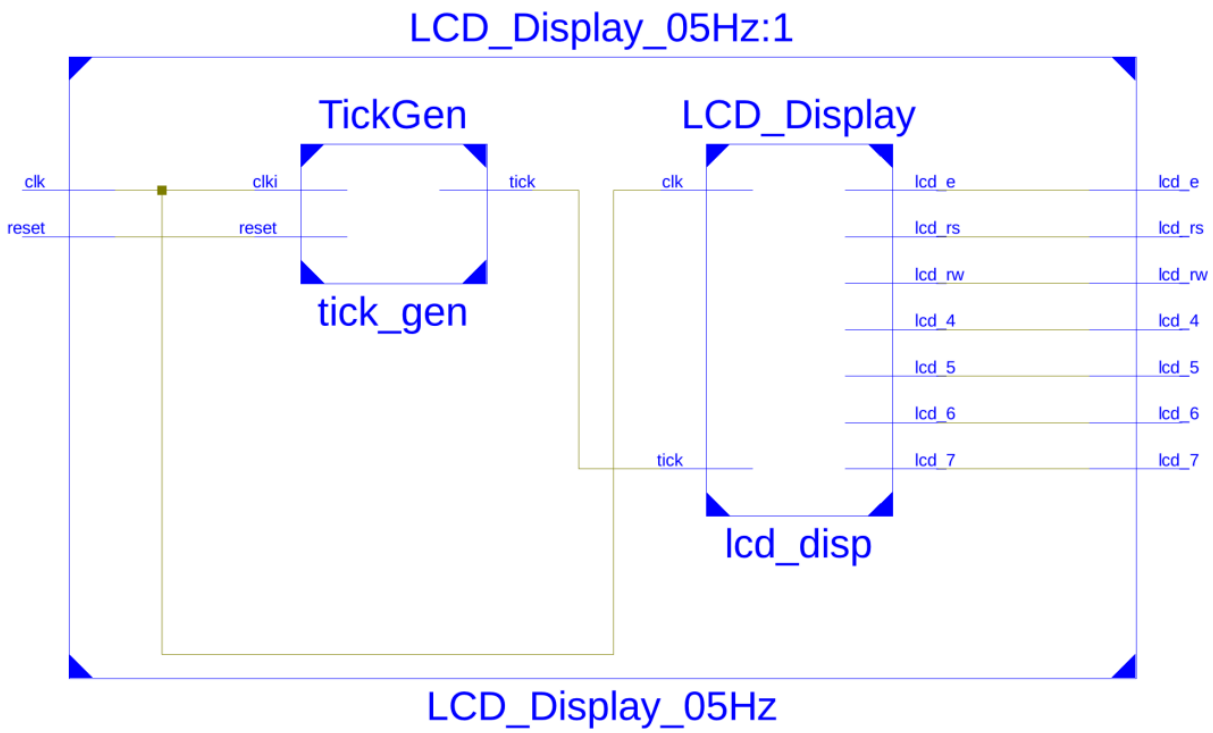
Students ID 2: 23139019

Date: 12/10/2025

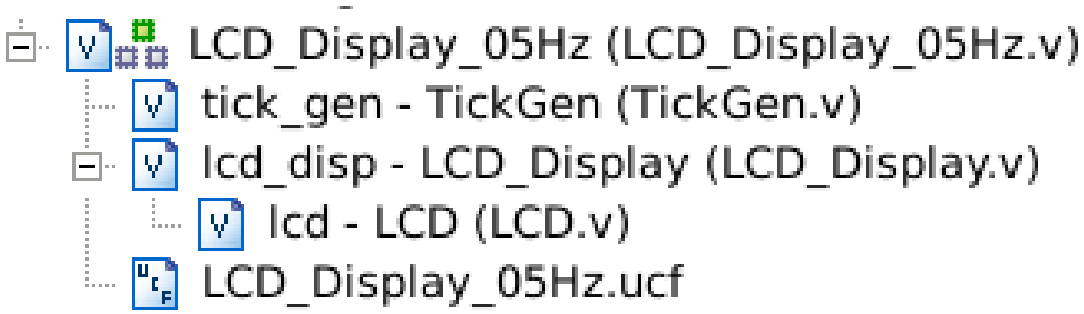
2. Block Diagram



Sơ đồ tổng quát

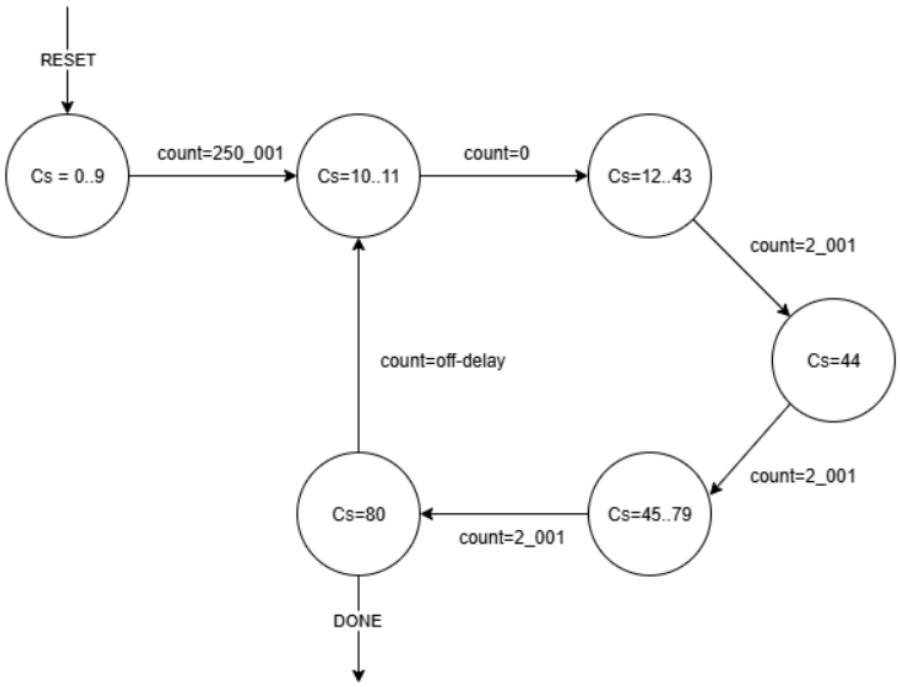


Sơ đồ xung clock 0.5Hz và Khối hiển thị



Cấu trúc chương trình

3. State Transition Diagram



Sơ đồ chuyển trạng thái

Mô đum TickGen

```
module TickGen
#(parameter M=50000000)
(
    input  wire clki,   // 50 MHz
    input  wire reset,
    output wire tick
);
        reg [30:0] r_reg;
        always @(posedge clki or posedge reset) begin
                if (reset)
                        r_reg <= 0;
                else if (r_reg == M-1)
```

```
                r_reg <= 0;
        else
                r_reg <= r_reg + 1;
    end

    assign tick = (r_reg == M-1);

endmodule
```

Mô đum LCD

```
module LCD(
        clk,
        chars,
        lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7);
        input clk;
        input [256:0] chars;
        output lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7;
        wire [256:0] chars;
        reg lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7;
        reg [5:0] lcd_code;
        reg [1:0] write = 2'b10; // write code has 10 for rs rw
        // delays
        reg [1:0] before_delay = 3; // time before on
        reg [3:0] on_delay = 13; // time on
        reg [23:0] off_delay = 750_001; // time off
        // states and counters
        reg [6:0] Cs = 0;
        reg [19:0] count = 0;
        reg [1:0] delay_state = 0;
```

```
        // character data
        reg [256:0] chars_hold = " ";
        wire [3:0] chars_data [63:0]; // array of characters

// redirects characters data to an array
        generate
        genvar i;
        for (i = 64; i > 0; i = i-1)
                begin : for_name
                assign chars_data[64-i] = chars_hold[i*4-1:i*4-4];
                end
        endgenerate



always @ (posedge clk) begin
// store character data
        if (Cs == 10 && count == 0) begin
        chars_hold <= chars;
        end
// set time when enable is off
        if (Cs < 3) begin

                case (Cs)
                0: off_delay <= 750_001; // 15ms delay
                1: off_delay <= 250_001; // 5ms delay
                2: off_delay <= 5_001; // 0.1ms delay
                endcase
            end
        else begin
```

```
                    if (Cs > 12) begin
                            off_delay <= 2_001; // 40us delay
                    end else begin
                            off_delay <= 250_001; // 5ms delay
                    end
            end
// delays during each state
if (Cs < 80) begin
        case (delay_state)
        0: begin
                            // enable is off
                    lcd_e <= 0;
                    {lcd_rs,lcd_rw,lcd_7,lcd_6,lcd_5,lcd_4} <= lcd_code;
                    if (count == off_delay) begin
                            count <= 0;
                            delay_state <= delay_state + 1;
                    end else begin
                            count <= count + 1;
                            end
            end
        1: begin
                    // data set before enable is on
                    lcd_e <= 0;
                    if (count == before_delay) begin
                            count <= 0;
                            delay_state <= delay_state + 1;
                    end else begin
                            count <= count + 1;
                            end
```

```
            end
        2: begin
        // enable on
            lcd_e <= 1;
            if (count == on_delay) begin
                count <= 0;
                delay_state <= delay_state + 1;
            end else begin
                count <= count + 1;
            end
        end
        3: begin
            // enable off with data set
            lcd_e <= 0;
            if (count == before_delay) begin
                count <= 0;
                delay_state <= 0;
                Cs <= Cs + 1; // nextcase
            end else begin
            count <= count + 1;
            end
            end
        endcase
end
//--------------Cs = 0 - 11 ------------------------------------------------------
// set lcd_code
if (Cs < 12) begin
        // initialize LCD
        case (Cs)
```

```
                0: lcd_code <= 6'h03; // power-on initialization
                1: lcd_code <= 6'h03;
                2: lcd_code <= 6'h03;
                3: lcd_code <= 6'h02;
                4: lcd_code <= 6'h02; // function set
                5: lcd_code <= 6'h08;
                6: lcd_code <= 6'h00; // entry mode set
                7: lcd_code <= 6'h06;
                8: lcd_code <= 6'h00; // display on/off control
                9: lcd_code <= 6'h0C;
                10:lcd_code <= 6'h00; // display clear
                11:lcd_code <= 6'h01;
        default: lcd_code <= 6'h10;
        endcase
        end else begin
 //---------Cs = 44-----------------------------------------------------------------
// set character data to lcd_code
        if (Cs == 44) begin // change address at end of first line



                lcd_code <= {2'b00, 4'b1100}; // 01000000 address change
        end else if (Cs == 45) begin
                lcd_code <= {2'b00, 4'b0000};
        end else begin
                if (Cs < 44) begin
                        lcd_code <= {write, chars_data[Cs-12]};
                end else begin
                        lcd_code <= {write, chars_data[Cs-14]};
                end
```

```
        end
        end
        // hold and loop back
        if (Cs == 80) begin
                lcd_e <= 0;
                if (count == off_delay) begin
                        Cs <= 10;
                        count <= 0;
                end else begin
                        count <= count + 1;
                end
        end
end
endmodule
```

Mô đum LCD_display

```
module LCD_Display(
        input wire clk,
        input wire tick,
        output wire lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7
);

        reg [255:0] chars;
        reg [4:0] shift_pos = 0;

        reg [7:0] line1 [0:31];
        reg [7:0] line2 [0:31];

        reg [7:0] line1_shifted [0:31];
```

```
reg [7:0] line2_shifted [0:31];

integer i, idx;

initial begin
        line1[0]  = "L"; line1[1]  = "E"; line1[2]  = " "; line1[3]  = "Q";
        line1[4]  = "U"; line1[5]  = "A"; line1[6]  = "N"; line1[7]  = "G";
        line1[8]  = " "; line1[9]  = "M"; line1[10] = "I"; line1[11] = "N";
        line1[12] = "H"; line1[13] = " "; line1[14] = "N"; line1[15] = "H";
        line1[16] = "A"; line1[17] = "T";
        line1[18] = " "; line1[19] = " "; line1[20] = " "; line1[21] = " ";
        line1[22] = " "; line1[23] = " "; line1[24] = " "; line1[25] = " ";
        line1[26] = " "; line1[27] = " "; line1[28] = " "; line1[29] = " ";
        line1[30] = " "; line1[31] = " ";

        line2[0]  = "N"; line2[1]  = "G"; line2[2]  = "U"; line2[3]  = "Y";
        line2[4]  = "E"; line2[5]  = "N"; line2[6]  = " "; line2[7]  = "Q";
        line2[8]  = "U"; line2[9]  = "O"; line2[10] = "C"; line2[11] = " ";
        line2[12] = "H"; line2[13] = "U"; line2[14] = "N"; line2[15] = "G";
        line2[16] = " "; line2[17] = " "; line2[18] = " "; line2[19] = " ";
        line2[20] = " "; line2[21] = " "; line2[22] = " "; line2[23] = " ";
        line2[24] = " "; line2[25] = " "; line2[26] = " "; line2[27] = " ";
        line2[28] = " "; line2[29] = " "; line2[30] = " "; line2[31] = " ";
end

LCD lcd(
        .clk(clk),
        .chars(chars),
        .lcd_rs(lcd_rs),
```

```
                .lcd_rw(lcd_rw),
                .lcd_e(lcd_e),
                .lcd_4(lcd_4),
                .lcd_5(lcd_5),
                .lcd_6(lcd_6),
                .lcd_7(lcd_7)
);

        always @(posedge clk) begin
                if (tick) begin
                        if (shift_pos == 31)
                                shift_pos <= 0;
                        else
                                shift_pos <= shift_pos + 1;
                end

                for (i = 0; i < 16; i = i + 1) begin
                        idx = shift_pos + i;
                        if (idx >= 32)
                                idx = idx - 32;

                        chars[255 - i*8 -: 8] <= line1[idx];
                        chars[127 - i*8 -: 8] <= line2[idx];
                end
        end
endmodule
```

Mô đum LCD_Display_05Hz

```
module LCD_Display_05Hz(
```

```
    input wire clk,
    input wire reset,
    output wire lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7
);

        wire tick_05Hz;

        TickGen #(.M(25_000_000)) tick_gen (
                .clki(clk),
                .reset(reset),
                .tick(tick_05Hz)
    );

        // LCD Display
        LCD_Display lcd_disp (
                .clk(clk),
                .tick(tick_05Hz),
                .lcd_rs(lcd_rs),
                .lcd_rw(lcd_rw),
                .lcd_e(lcd_e),
                .lcd_4(lcd_4),
                .lcd_5(lcd_5),
                .lcd_6(lcd_6),
                .lcd_7(lcd_7)
        );

endmodule
```

4. Test Cases Overview

TC ID	Purpose	Input	Expected Output	Result (Pass/Fail)
TC1	Kiểm tra khi Bật LCD	clk = ~clk	Hiện tên 2 sinh viên	Pass
TC2	Chữ di chuyển từ trái qua phải	clk = ~clk	Hiển thị dịch trái	Pass
TC3	Chữ di chuyển lại từ trái	clk = ~clk	Hiển thị quay lại từ đầu	Pass

Mô đum testmodule

```
`timescale 1ns / 1ps

module Testbench_LCD_Display;

    // Inputs
    reg clk;
    reg tick;

    // Outputs
    wire lcd_rs;
    wire lcd_rw;
    wire lcd_e;
    wire lcd_4;
    wire lcd_5;
    wire lcd_6;
    wire lcd_7;

    reg [7:0] display_line1 [0:15];
    reg [7:0] display_line2 [0:15];
```

```
integer i;

// Instantiate the Unit Under Test (UUT)
LCD_Display uut (
        .clk(clk),
        .tick(tick),
        .lcd_rs(lcd_rs),
        .lcd_rw(lcd_rw),
        .lcd_e(lcd_e),
        .lcd_4(lcd_4),
        .lcd_5(lcd_5),
        .lcd_6(lcd_6),
        .lcd_7(lcd_7)
);

initial begin
        // Initialize Inputs
        clk = 0;
        tick = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

end

always forever #10 clk = ~clk;
always forever #20 tick = ~tick;
```

```
task display_lcd;
        begin
                for (i = 0; i < 16; i = i + 1) begin
                        display_line1[i] = uut.chars[255 - i*8 -: 8];
                        display_line2[i] = uut.chars[127 - i*8 -: 8];
                end
                $display("\n====================");
                $write  (" ");
                for (i = 0; i < 16; i = i + 1) begin
                        $write("%c", display_line1[i]);
                end
                $display("");

                $write  (" ");
                for (i = 0; i < 16; i = i + 1) begin
                        $write("%c", display_line2[i]);
                end
                $display("");
                $display("====================");
        end
endtask

initial begin
        #500;
        repeat(35) begin
                @(posedge tick);
                        #100;
                        display_lcd();
```

```
                    #1000;
            end
        end

endmodule
```

5. Testcase Details

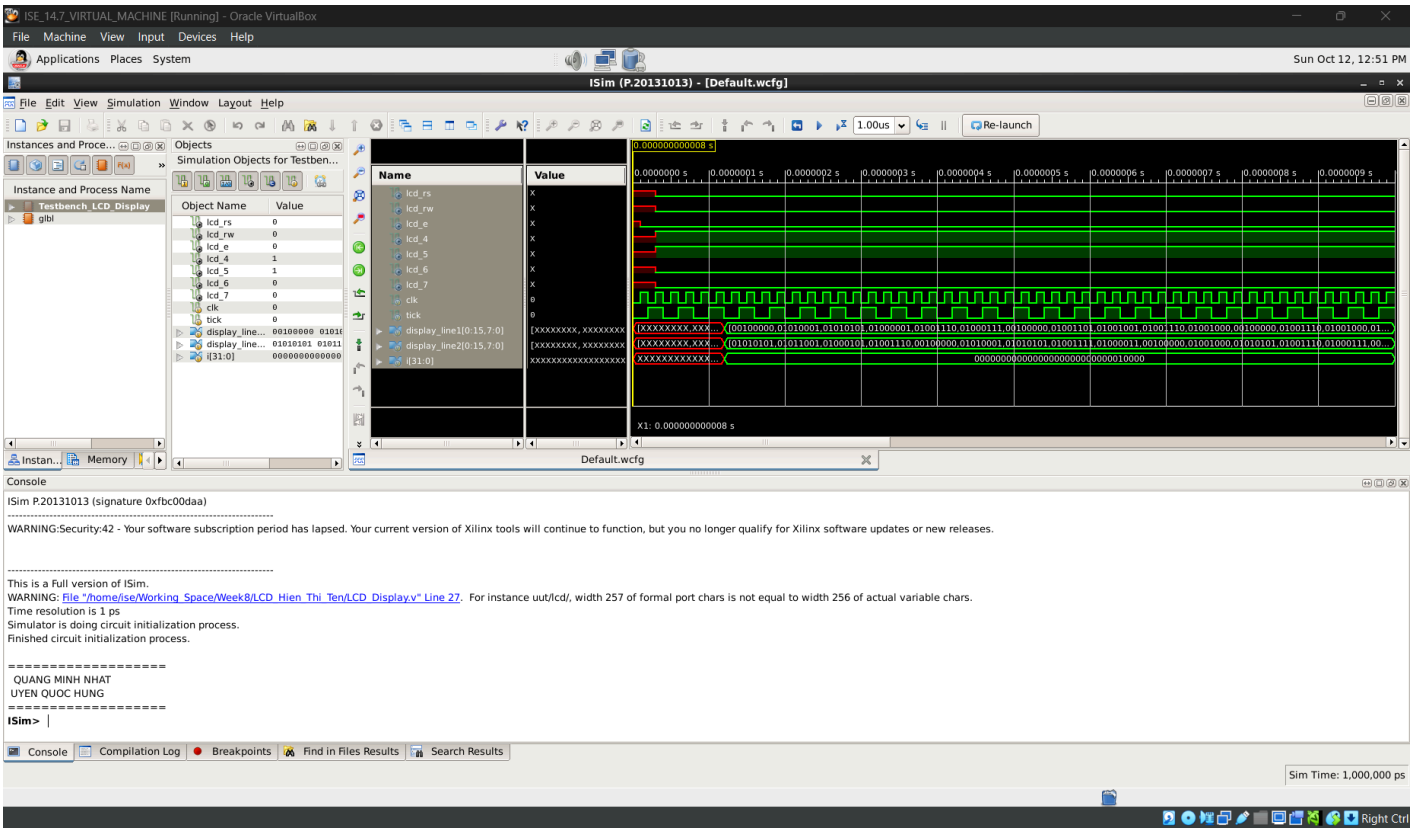
Testcase 1: TC1

Purpose: Kiểm tra khi Bật LCD

Input/Stimulus: clk = ~clk

Expected Output: Hiện tên 2 sinh viên

Waveform Simulation (attach/insert image):



Analysis:

Ban đầu LCD hiển thị tên sinh viên

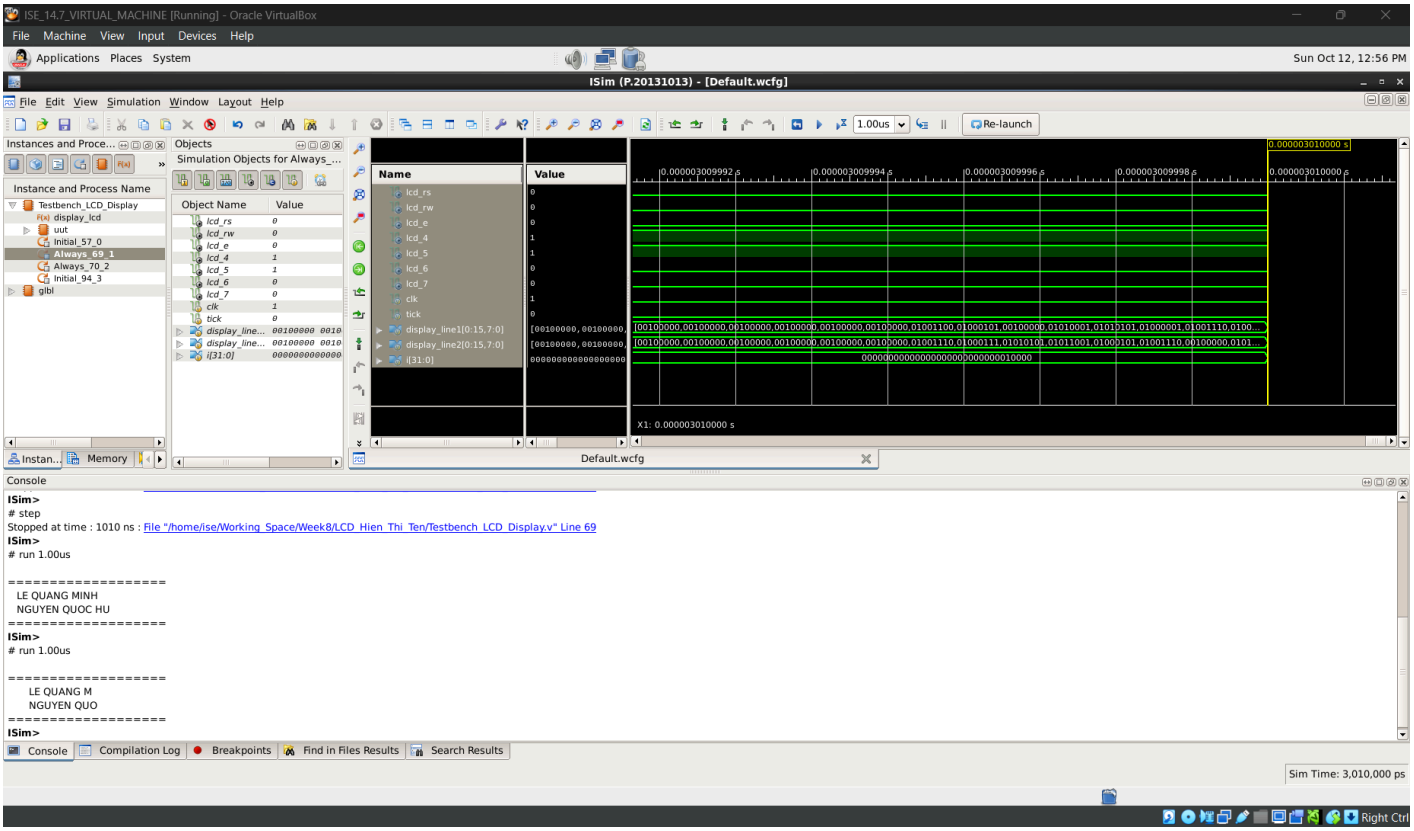
Testcase 2: TC2

Purpose: Chữ di chuyển từ trái qua phải

Input/Stimulus: clk = ~clk

Expected Output: Hiển thị dịch trái

Waveform Simulation (attach/insert image):



Analysis:

Dịch tên sinh viên qua phải từng chút một.

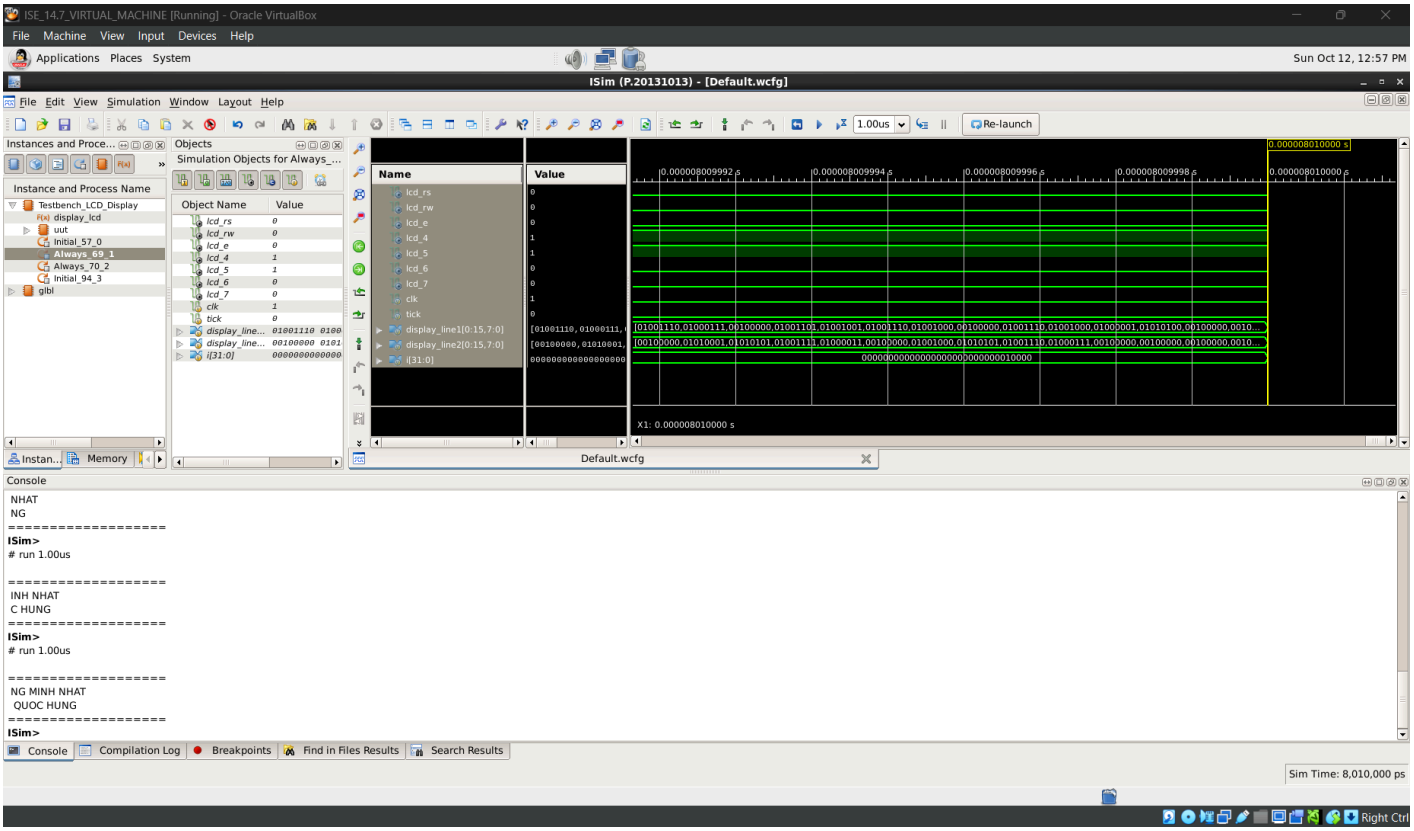
Testcase 3: TC3

Purpose: Chữ di chuyển lại từ trái

Input/Stimulus: clk = ~clk

Expected Output: Hiển thị quay lại từ đầu

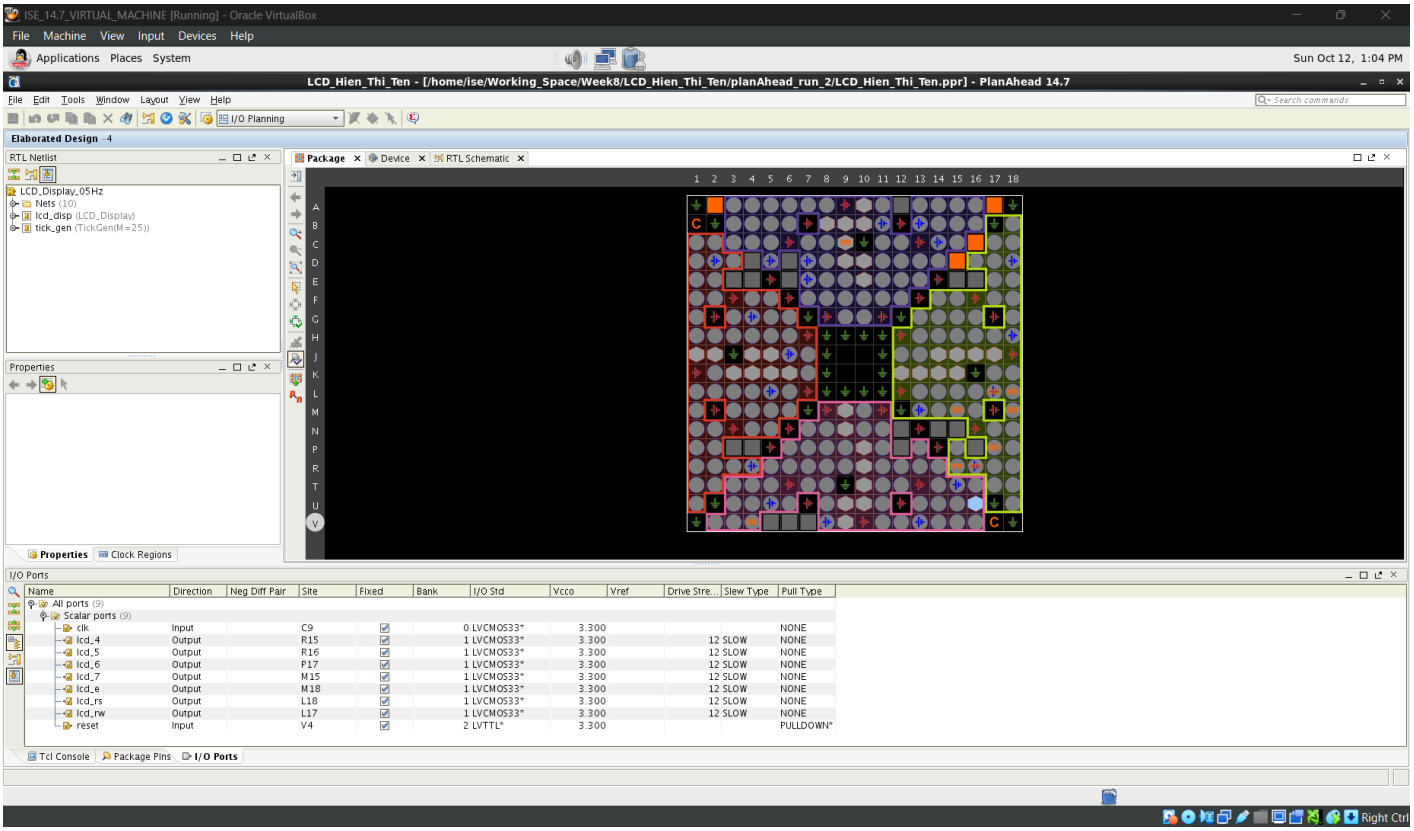
Waveform Simulation (attach/insert image):



Analysis:

Tên sinh viên bắt đầu lại từ đầu khi chữ đã di chuyển hết màn hình.

6. Summary




```
# PlanAhead Generated IO constraints

NET "clk" IOSTANDARD = LVCMOS33;

# PlanAhead Generated physical constraints

NET "clk" LOC = C9;
NET "lcd_e" LOC = M18;
NET "lcd_rs" LOC = L18;
NET "lcd_rw" LOC = L17;
NET "reset" LOC = V4;

# PlanAhead Generated IO constraints
```

```
NET "reset" IOSTANDARD = LVTTL;
NET "reset" PULLDOWN;
NET "lcd_rs" IOSTANDARD = LVCMOS33;
NET "lcd_rw" IOSTANDARD = LVCMOS33;
NET "lcd_e" IOSTANDARD = LVCMOS33;

# PlanAhead Generated physical constraints

NET "lcd_4" LOC = R15;
NET "lcd_5" LOC = R16;
NET "lcd_6" LOC = P17;
NET "lcd_7" LOC = M15;

# PlanAhead Generated IO constraints

NET "lcd_7" IOSTANDARD = LVCMOS33;
NET "lcd_6" IOSTANDARD = LVCMOS33;
NET "lcd_5" IOSTANDARD = LVCMOS33;
NET "lcd_4" IOSTANDARD = LVCMOS33;
```