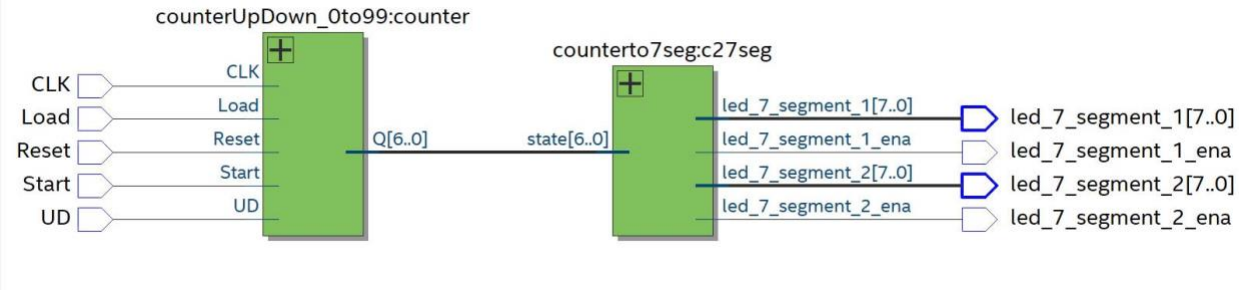
Báo cáo thực hành buổi 9+10
Thiết kế số và vi xử lý

Họ và tên : Dương Minh Vương
MSV : 22022156

I:Counter 4 bit Up and Down



1. Lý thuyết

a,Giới thiệu về Counter 4-bit Up và Down

+Bộ đếm này có hai chế độ hoạt động:

- Up (UD = 0): Đếm tăng, giá trị của bộ đếm tăng dần lên.
- Down (UD = 1): Đếm giảm, giá trị của bộ đếm giảm dần xuống.

+Cơ chế đếm dựa trên tín hiệu xung nhịp (CLK) để đồng bộ hóa các thay đổi trạng thái.

b, Hoạt động của mạch

+Tín hiệu điều khiển chính:

- CLK (Clock): Điều khiển thời điểm thay đổi trạng thái.
- UD (Up/Down): Quy định chế độ:
  - UD = 0: Đếm tăng.
  - UD = 1: Đếm giảm.
- r_next: Giá trị tiếp theo của bộ đếm (được tính dựa trên trạng thái hiện tại).
- r_reg: Giá trị hiện tại được lưu trữ trong thanh ghi.

+Phương trình logic:

- Khi UD = 0 (Up):
  r_next = r_reg + 1;
- Khi UD = 1 (Down):
  r_next=r_reg−1r

Thanh ghi sẽ lưu giá trị của r_next sau mỗi xung CLK.

c, Cấu trúc mạch

+Cấu trúc mạch bao gồm:

- Một thanh ghi để lưu trạng thái hiện tại r_reg.
- Một mạch tính toán logic để xác định giá trị tiếp theo (r_next) dựa trên chế độ UD.
- Một tín hiệu xung nhịp (CLK) để kích hoạt cập nhật giá trị.

d, Ứng dụng

- Dùng trong mạch đồng hồ (Clock circuits).
- Quản lý các tín hiệu thời gian (Timing signals).

- Các bộ đếm trong hệ thống kỹ thuật số.

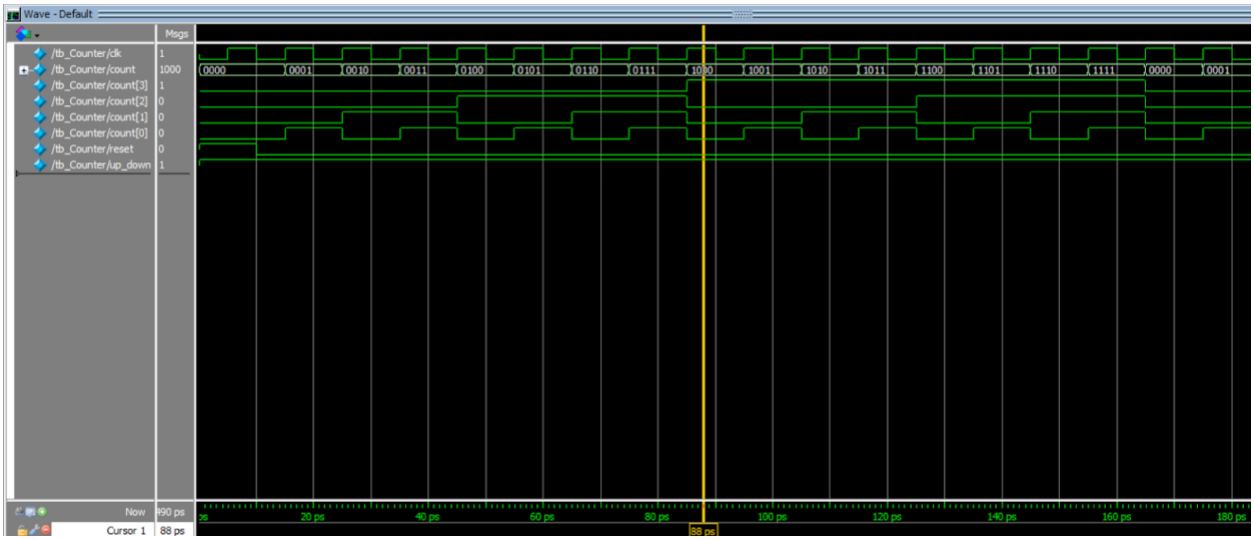
2.Code HDL

```
module Counter (
  input clk,
  input reset,
  input up_down,
  output reg [3:0] count
);


  always @(posedge clk or posedge reset) begin
    if (reset) begin
      count <= 4'b0000;
    end else begin
      if (up_down) begin
        count <= count + 1;
      end else begin
        count <= count - 1;
      end
    end
  end
endmodule
```

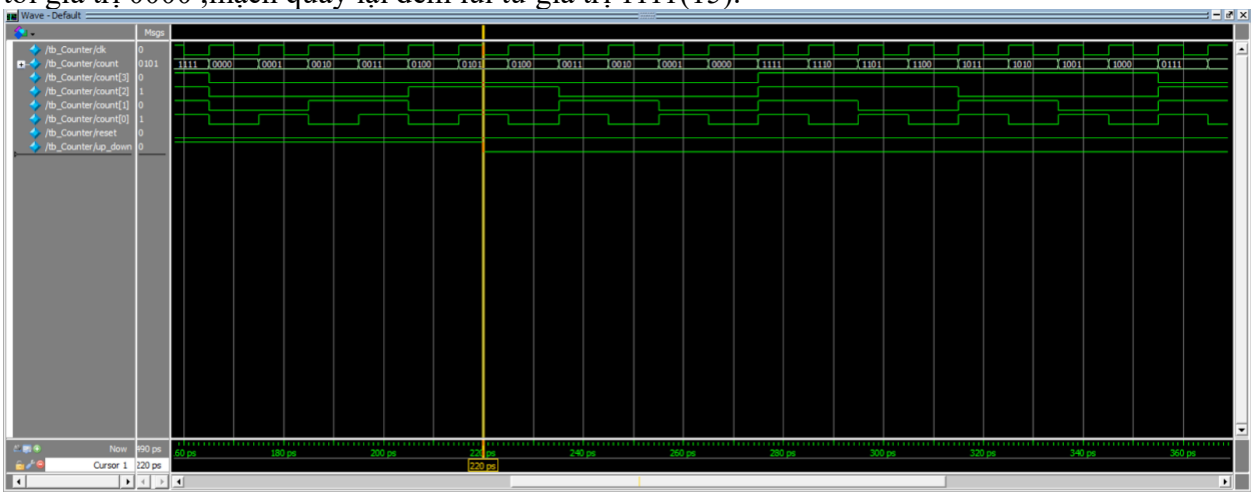
3.Testbench và mô phỏng lối ra theo lối vào

```
module tb_Counter();
    logic clk;
    logic reset;
    logic up_down;
    logic [3:0] count;

    Counter uut (
        .clk(clk),
        .reset(reset),
        .up_down(up_down),
        .count(count)
    );

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

    initial begin

        reset = 1;
        up_down = 1;
        #10;

        reset = 0;
        #10;

        up_down = 1;
        #200

        up_down = 0;
        #200

        reset = 1;
        #10;
        reset = 0;
        #10;

        up_down = 1;
        #50;
        $stop;
    end
endmodule
```

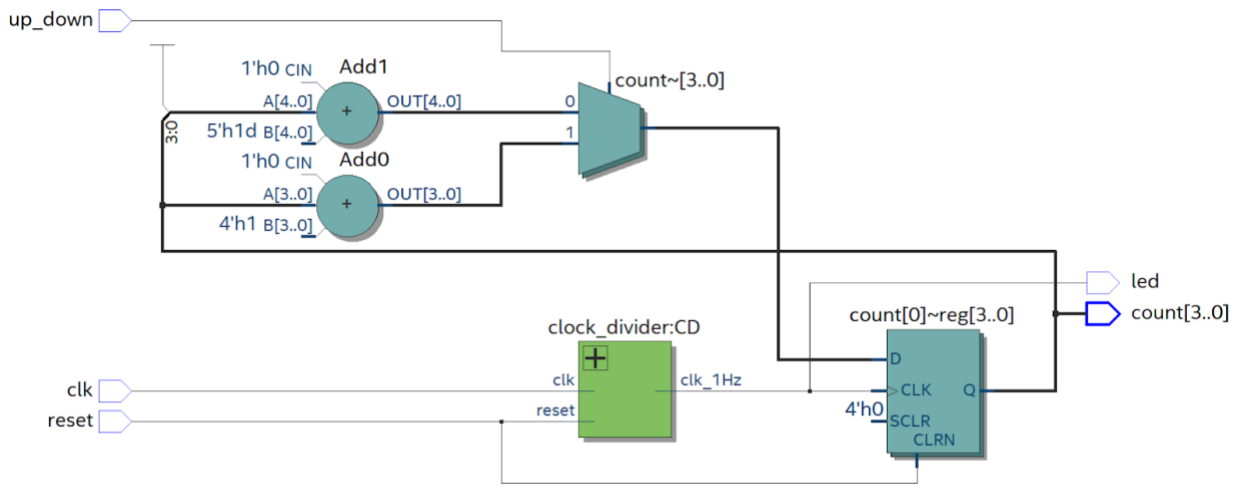
Mô phỏng:
+Khi up_down = 1 ,mạch tiến hành đếm lên từ giá trị 0-15 (0000-1111) tại thời điểm ban đầu và đếm tiến từ giá trị hiện tại khi chuyển up_down từ 0->1.Khi đếm đến giá trị 15(tức 1111) mạch quay về giá trị 0(0000) do đây là bộ đếm 4 bit.



+Khi up_down = 0,mạch tiến hành đếm lùi từ giá trị hiện theo xung clock (trong hình trạng thái hiện tại đang ở giá trị 5(tức 0101) thì up_down chuyển từ 1 về 0,mạch đếm lùi từ 5 trở về.Khi về tời giá trị 0000 ,mạch quay lại đếm lùi từ giá trị 1111(15).

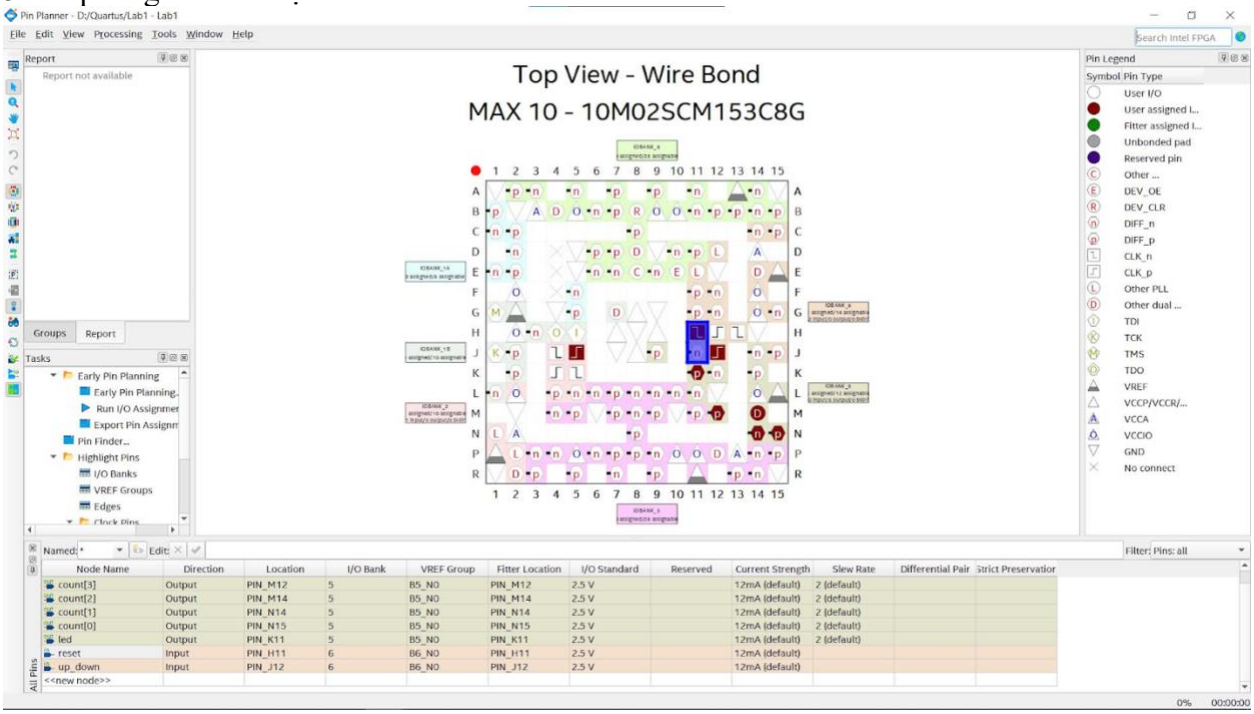


4.Mô phỏng mạch trên Quartus



Giải thích:Trong hình trên,có một bộ clock_divider thực hiện chia xung clock và hiển thị trên 1 con led cho phép quan sát sự thay đổi tín hiệu xung cũng như sẽ làm đầu vào cho bộ đếm counter_4bit.

5.Mô phỏng trên bo mạch FPGA



Link video chi tiết mô phỏng:
👉 Video báo cáo thực hành

II:Bộ đếm Full Counter từ 0 - 99 hiển thị trên led 7 thông qua Binary to BCD

1:Lý thuyết

+Bộ đếm Full Counter có đầu vào là tín hiệu xung CLK,reset,start và ud.Chân start có tác dụng cho phép bắt đầu thực hiện đếm lên hoặc xuống.Các chân còn lại có tác dụng tương tự như phần I.

+Bộ Full Counter bao gồm các thành phần:

1.Bộ đếm tuần tự (counter_99)

2.Chuyển đổi nhị phân sang BCD(binary to bcd)

3.Hiển thị kết quả lên led 7 đoạn(counter to 7seg)

2:Code HDL

+Binary to BCD

```
module binarytobcd (
 input [6:0] binary,
 output reg [7:0] bcd
);
 reg [4:0] i;
 always @*
 begin
 bcd = 8'b0;
 for (i = 0 ; i < 7 ; i = i + 1)
    begin
    if (bcd[7:4] >= 5)
        bcd[7:4] = bcd[7:4] + 3;
    if (bcd[3:0] >= 5)
        bcd[3:0] = bcd[3:0] + 3;
    bcd = {bcd[7:0],binary[6-i]};
    end
 end
endmodule
```

Chức năng:

Chuyển đổi một số nhị phân (binary) 7-bit sang mã BCD (8-bit), mỗi 4-bit đại diện cho một chữ số thập phân.

Các thành phần:

- Input:
  binary (7-bit): Giá trị đầu vào nhị phân.
- Output:
  bcd (8-bit): Giá trị BCD.

Giải thích logic:

- Cơ chế cộng 3 (Add-3): Được dùng để chuyển đổi từ nhị phân sang BCD.
- Trong vòng lặp, nếu nhóm 4-bit cao nhất hoặc thấp nhất của bcd lớn hơn hoặc bằng 5, cộng thêm 3 vào nhóm đó để chuẩn hóa mã BCD.
- Dịch chuyển bit từ giá trị binary sang giá trị bcd qua từng bước của vòng lặp (for).

+Counter 99

```
module counter_99 (
    input clk,
    input reset_n,
    input start,
    input ud,
    output reg [6:0] state
);
    localparam clk_div_period = 12_000_000;
    reg [24:0] counter_clk;

    always @(posedge clk, negedge reset_n) begin
        if (~reset_n)
            counter_clk <= 0;
        else if (counter_clk == clk_div_period - 1)
            counter_clk <= 0;
        else
            counter_clk = counter_clk + 1;
    end

    always @(posedge clk, negedge reset_n) begin
        if (~reset_n) begin
            if (ud)
                state <= 0;
            else
                state <= 99;
        end else begin
            if (start && counter_clk == clk_div_period - 1) begin
                if (ud)
                    state <= state + 1;
                else
                    state <= state - 1;
            end else
                state <= state;
        end
    end
endmodule
```

Chức năng:
Tạo bộ đếm từ 0 đến 99 hoặc ngược lại, có thể thay đổi chế độ tăng (up) hoặc giảm (down).
Các thành phần:

- Input:
  - clk: Tín hiệu xung nhịp.
  - reset_n: Tín hiệu reset (active-low).
  - start: Tín hiệu khởi động bộ đếm.
  - ud: Chọn chế độ (0: tăng, 1: giảm).
- Output:
  - state: Giá trị hiện tại của bộ đếm (7-bit).

Giải thích logic:

1. Bộ chia xung nhịp (counter_clk):
    - Giảm tốc độ của tín hiệu clk để phù hợp với tốc độ bộ đếm.
    - Sử dụng hằng số clk_div_period = 12_000_000 để đặt chu kỳ chia xung (giả định tín hiệu clk ban đầu là 12 MHz).
2. Đếm tăng/giảm (state):
    - Khi ud = 0 (tăng): Giá trị state tăng lên đến 99, sau đó quay về 0.
    - Khi ud = 1 (giảm): Giá trị state giảm về 0, sau đó quay về 99.
    - Giá trị chỉ cập nhật khi tín hiệu start được kích hoạt và xung nhịp được chia hoàn tất.

+Counter to 7seg

```
module counterto7seg (
    input [6:0] state,
    output reg [7:0] led_7_segment_1,
    output led_7_segment_1_ena,
    output reg [7:0] led_7_segment_2,
    output led_7_segment_2_ena
);
    wire [7:0] bcd;

    assign led_7_segment_1_ena = 0;
    assign led_7_segment_2_ena = 0;
    // Chuy?n ??i t? nh? phân sang BCD
    binarytobcd b2bcd (
        .binary(state),
        .bcd(bcd)
    );

    always @* begin
        case (bcd[7:4])
            4'd0: led_7_segment_1 = 8'h3F;
            4'd1: led_7_segment_1 = 8'h06;
            4'd2: led_7_segment_1 = 8'h5B;
            4'd3: led_7_segment_1 = 8'h4F;
            4'd4: led_7_segment_1 = 8'h66;
            4'd5: led_7_segment_1 = 8'h6D;
            4'd6: led_7_segment_1 = 8'h7D;
            4'd7: led_7_segment_1 = 8'h07;
            4'd8: led_7_segment_1 = 8'h7F;
            4'd9: led_7_segment_1 = 8'h6F;
            default: led_7_segment_1 = 8'hFF;
        endcase
    end
```

```
    always @* begin
        case (bcd[3:0])
            4'd0: led_7_segment_2 = 8'h3F;
            4'd1: led_7_segment_2 = 8'h06;
            4'd2: led_7_segment_2 = 8'h5B;
            4'd3: led_7_segment_2 = 8'h4F;
            4'd4: led_7_segment_2 = 8'h66;
            4'd5: led_7_segment_2 = 8'h6D;
            4'd6: led_7_segment_2 = 8'h7D;
            4'd7: led_7_segment_2 = 8'h07;
            4'd8: led_7_segment_2 = 8'h7F;
            4'd9: led_7_segment_2 = 8'h6F;
            default: led_7_segment_2 = 8'hFF;
        endcase
    end

endmodule
```

Chức năng:
Hiển thị giá trị của bộ đếm (đầu vào state) lên hai LED 7 đoạn, tương ứng với chữ số hàng chục và hàng đơn vị.
Các thành phần:

- Input:
  - state (7-bit): Giá trị đầu vào từ bộ đếm.
- Output:
  - led_7_segment_1: Giá trị mã hóa để hiển thị chữ số hàng chục.
  - led_7_segment_2: Giá trị mã hóa để hiển thị chữ số hàng đơn vị.
  - led_7_segment_1_ena và led_7_segment_2_ena: Kích hoạt LED (đang để mặc định vô hiệu hóa).

Giải thích logic:

1. Chuyển đổi sang BCD:
   Sử dụng module binarytobcd để chuyển giá trị nhị phân sang mã BCD.
2. Mã hóa LED 7 đoạn:
   - Giá trị BCD [7:4] (hàng chục) được ánh xạ sang mã hiển thị cho led_7_segment_1.
   - Giá trị BCD [3:0] (hàng đơn vị) được ánh xạ sang mã hiển thị cho led_7_segment_2.

+Full Counter

```
module full_counter (
    input clk,
    input reset_n,
    input start,
    input ud,
    output [7:0] led_7_segment_1,
    output led_7_segment_1_ena,
    output [7:0] led_7_segment_2,
    output led_7_segment_2_ena
);

    wire [6:0] state;

    counter_99 counter (
        .clk(clk),
        .reset_n(reset_n),
        .start(start),
        .ud(ud),
        .state(state)
    );

    counterto7seg c27seg (
        .state(state),
        .led_7_segment_1(led_7_segment_1),
        .led_7_segment_1_ena(led_7_segment_1_ena),
        .led_7_segment_2(led_7_segment_2),
        .led_7_segment_2_ena(led_7_segment_2_ena)
    );

endmodule
```

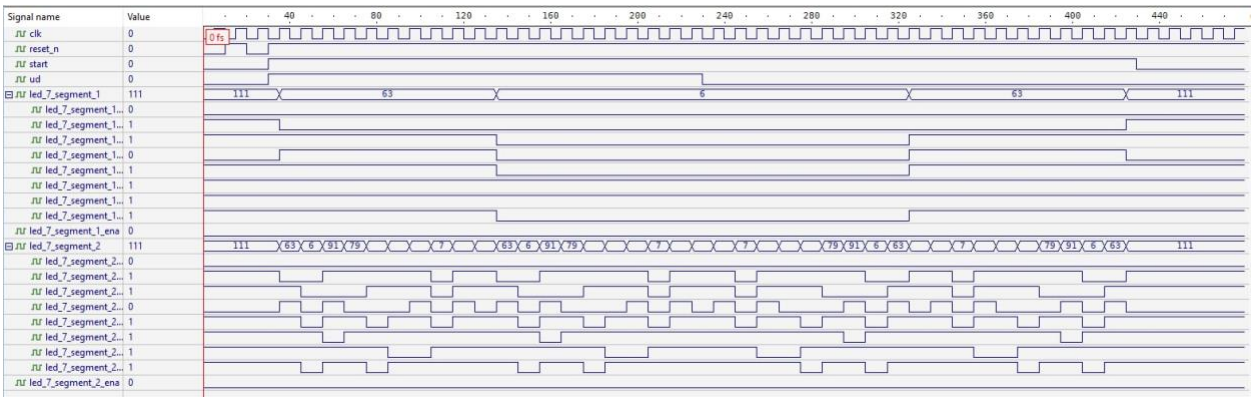
Chức năng:
Module chính để liên kết các thành phần:

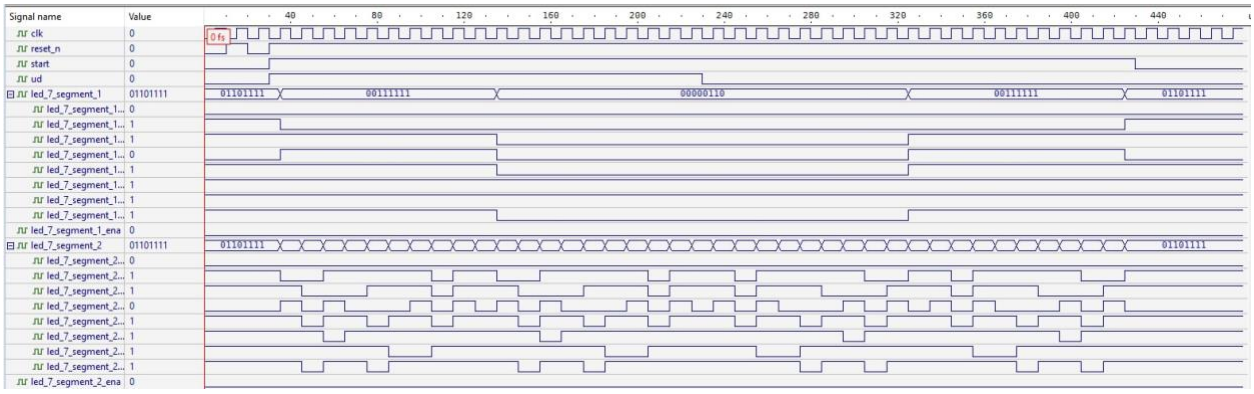
- counter_99: Bộ đếm giá trị.
- binarytobcd và counterto7seg: Hiển thị kết quả lên LED 7 đoạn.

Logic:

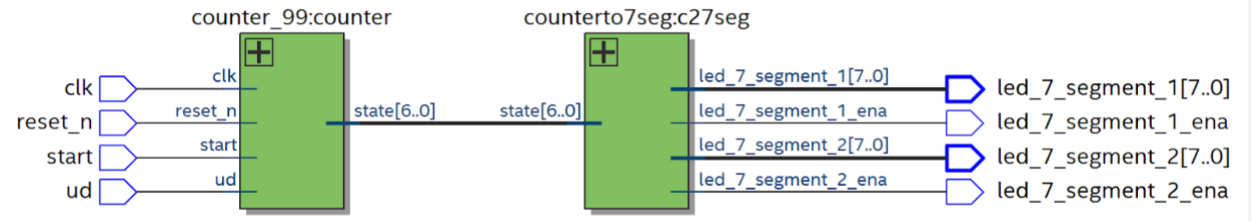
- Giá trị từ counter_99 được truyền vào counterto7seg.
- Mã hóa giá trị và hiển thị kết quả tương ứng.

3:Testbench và mô phỏng lối ra theo lối vào
Code Testbench:(code chi tiết trong link cuối báo cáo)

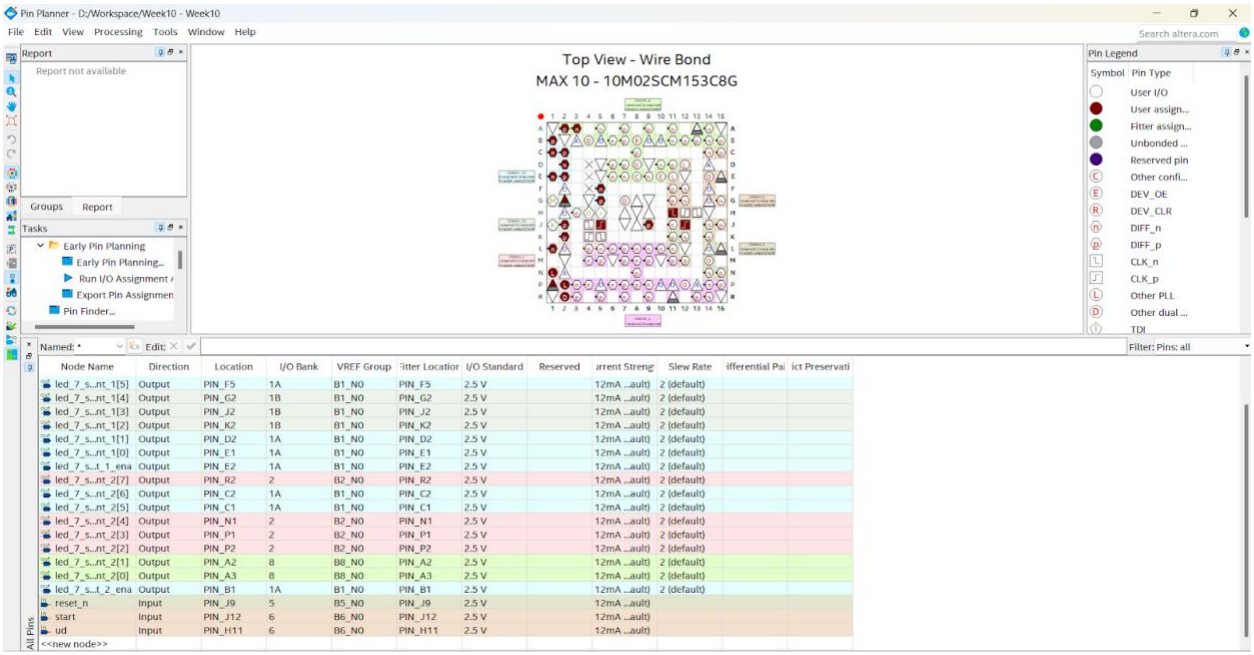




4.Mô phỏng mạch trên Quartus



5.Mô phỏng trên bo mạch FPGA



Link video chi tiết mô phỏng:

👉 Video báo cáo thực hành