Led controller

**Bai 8led_4 Switch**

```
`timescale 1ns / 1ps

module LED_Controller (
    input wire clk,     // tín hiệu xung nhịp
       input wire [1:0] SW,
    input wire [1:0] sw,     // 2 switch S3, S4 để chọn mode
    output reg [7:0] led   // 8 LED điều khiển
);

reg [2:0] mode;           // Biến lưu mode hiện tại (0 - 7)
reg [3:0] brightness;     // Biến điều khiển độ sáng của LED
reg direction;            // Biến điều khiển hướng chạy của LED
wire [3:0] f;
wire clk_o ;
Clock_div clockdivider (clk, f) ;
Mux41 mux4to1 (f,SW,clk_o);
initial begin
    led = 8'b00000000;    // Ban đầu tất cả LED tắt
    mode = 3'b000;        // Mode khởi tạo
    brightness = 4'b0000; // Độ sáng ban đầu
    direction = 0;        // Hướng chạy ban đầu (0: trái sang phải)
end

// Đọc mode từ S3 và S4 (mỗi lần thay đổi giá trị của S3 hoặc S4)

// Khối điều khiển LED chạy theo mode
always @(posedge clk_o) begin
    case (sw)
        2'b00: begin // Mode 1: 8 LED chớp tắt
            led <= ~led; // Đảo trạng thái tất cả LED
        end

        2'b01: begin // Mode 2: 8 LED sáng dần, tắt dần
            if (brightness == 4'b1111) begin
                brightness <= 0; // Reset độ sáng khi đạt max
            end else begin
                brightness <= brightness + 1;
            end
                        led <= (8'b11111111 >> (4 - brightness)); // Điều chỉnh độ sáng
        end
              2'b10: begin // Mode 3: 1 LED sáng chạy từ trái sang phải, rồi từ phải sang trái
            if (direction == 0) begin
                led <= (led == 8'b10000000) ? 8'b00000001 : led << 1; // Dịch trái
                if (led == 8'b10000000) direction <= 1; // Đổi hướng khi đến LED cuối
            end else begin
                led <= (led == 8'b00000001) ? 8'b10000000 : led >> 1; // Dịch phải
                if (led == 8'b00000001) direction <= 0; // Đổi hướng khi đến LED đầu
            end
        end
              2'b11: begin // Mode 4: 8 LED sáng dồn
            led <= (led == 8'b11111111) ? 8'b00000001 : (led << 1) | 8'b00000001;
        end

        default: begin
            led <= 8'b00000000; // Tắt tất cả LED trong các trường hợp khác
        end
    endcase
end

endmodule
```

Clock divider

```
`timescale 1ns / 1ps

module Clock_div
      #(parameter N= 30, M = 500000000) // 500,000,000 for 0.1Hz
            ( input wire clk,
            output wire [3:0]q
      );
      // signal declaration
      reg [N-1:0] r_reg01H,r_reg1H,r_reg10H,r_reg100H;
      wire [N-1:0] r_next01H,r_next1H,r_next10H,r_next100H;
      // body, register
      always @(posedge clk) begin
            r_reg01H<=r_next01H;
            r_reg1H<=r_next1H;
            r_reg10H<=r_next10H;
            r_reg100H<=r_next100H;
      end
      // next state logic
      assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
      assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
      assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
      assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H + 1;
      // output logic
      assign q[0]=(r_reg01H<M/2)?0:1;
      assign q[1]=(r_reg1H<M/20)?0:1;
      assign q[2]=(r_reg10H<M/200)?0:1;
      assign q[3]=(r_reg100H<M/2000)?0:1;
endmodule
```

Mux4-1

```
`timescale 1ns / 1ps

module Mux41
            ( input wire [3:0] clk,
            input wire [1:0] sw,
            output reg clk_o
      );
      // signal declaration
      // clk_o ;
      always @(clk,sw)
            case (sw)
                  0: clk_o = clk[0];
                  1: clk_o = clk[1];
                  2: clk_o = clk[2];
                  3: clk_o = clk[3];
            endcase
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "sw<0>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "sw<1>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Chia xung**

```
`timescale 1ns / 1ps

module Counter
      #(parameter N= 26)
            ( input wire clk, reset,
            output wire [3:0] q
      );
            // signal declaration
            reg [N-1:0] r_reg;
            wire [N-1:0] r_next;
            // body, register
            always @(posedge clk, posedge reset)
                  if (reset)
                        r_reg <= 0;
                  else
                        r_reg<=r_next; // <= is non-blocking statement
                  // next state logic
                  assign r_next = r_reg + 1;
                  // output logic
            assign q=r_reg[25:22];
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Chia_xung_0.1_1_10_100Hz**

```
`timescale 1ns / 1ps

module Counter
      #(parameter N= 30, M = 500000000) // 500,000,000 for 0.1Hz
            ( input wire clk,
            output wire [3:0]q
      );
      // signal declaration
      reg [N-1:0] r_reg01H,r_reg1H,r_reg10H,r_reg100H;
      wire [N-1:0] r_next01H,r_next1H,r_next10H,r_next100H;
      // body, register
      always @(posedge clk) begin
            r_reg01H<=r_next01H;
            r_reg1H<=r_next1H;
            r_reg10H<=r_next10H;
            r_reg100H<=r_next100H;
      end
      // next state logic
      assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
      assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
      assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
      assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H +
      1;
      // output logic
      assign q[0]=(r_reg01H<M/2)?0:1;
      assign q[1]=(r_reg1H<M/20)?0:1;
      assign q[2]=(r_reg10H<M/200)?0:1;
      assign q[3]=(r_reg100H<M/2000)?0:1;
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Counter 8bit_switch_s0s1s2**

LED_counter

```
`timescale 1ns / 1ps

module LED_COUNTER(
      input wire clk, reset,
      input wire [1:0] SW,
      input wire UD,
      output wire [7:0] LED
);
// wire declaration
wire [3:0] f;
wire clk_o ;
// module instance
Clock_div clockdivider (clk, f) ;
Mux41 mux4to1 (f,SW,clk_o);
CounterUD counter (clk_o, reset,UD,LED);
endmodule
```

Clock_div

```
`timescale 1ns / 1ps

module Clock_div
      #(parameter N= 30, M = 500000000) // 500,000,000 for 0.1Hz
            ( input wire clk,
            output wire [3:0]q
      );
      // signal declaration
      reg [N-1:0] r_reg01H,r_reg1H,r_reg10H,r_reg100H;
      wire [N-1:0] r_next01H,r_next1H,r_next10H,r_next100H;
      // body, register
      always @(posedge clk) begin
            r_reg01H<=r_next01H;
            r_reg1H<=r_next1H;
            r_reg10H<=r_next10H;
            r_reg100H<=r_next100H;
      end
      // next state logic
      assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
      assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
      assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
      assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H + 1;
      // output logic
      assign q[0]=(r_reg01H<M/2)?0:1;
      assign q[1]=(r_reg1H<M/20)?0:1;
      assign q[2]=(r_reg10H<M/200)?0:1;
      assign q[3]=(r_reg100H<M/2000)?0:1;
endmodule
```

Mux41

```
`timescale 1ns / 1ps

module Mux41
      (      input wire [3:0] clk,
             input wire [1:0] sw,
             output reg clk_o
      );
      // signal declaration
      // clk_o ;
      always @(clk,sw)
            case (sw)
                  0: clk_o = clk[0];
                  1: clk_o = clk[1];
                  2: clk_o = clk[2];
                  3: clk_o = clk[3];
            endcase
endmodule
```

CounterUD

```
`timescale 1ns / 1ps

module CounterUD
      #(parameter N= 8) // 500,000,000 for 0.1Hz
            ( input wire clk,reset,ud,
            output wire [7:0]q
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk, posedge reset)
            if (reset)
                  r_reg<=0;
            else
                  r_reg<=r_next;
      // next state logic
      assign r_next = (ud==1)?r_reg + 1:r_reg - 1;
      // output logic
      assign q=r_reg;
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "UD" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
```

**Counter 8bit_ switch_ s0s1s2s3 _pause**

LED_ COUNTER

```
`timescale 1ns / 1ps

module LED_COUNTER(
      input wire clk, reset,
      input wire [2:0] SW,
      input wire pause,
      input wire toggle,
      input wire UD,
      output wire [7:0] LED
);
// wire declaration
wire [7:0] f;
wire clk_o ;
// module instance
Clock_div clockdivider (clk, f) ;
Mux81 mux8to1 (f,SW,clk_o);
CounterUD counter (clk_o, reset, UD, pause, toggle, LED);
endmodule
```

Clock_div

```
`timescale 1ns / 1ps

module Clock_div
      #(parameter N= 30, M = 500000000) // 500,000,000 for 0.1Hz
            ( input wire clk,
            output wire [7:0]q
      );
      // signal declaration
      reg [N-1:0] r_reg01H,r_reg1H,r_reg5H,r_reg10H,r_reg20H,r_reg50H,r_reg70H,r_reg100H;
      wire [N-1:0] r_next01H,r_next1H,r_next5H,r_next10H,r_next20H,r_next50H,r_next70H,r_next100H;
      // body, register
      always @(posedge clk) begin
            r_reg01H<=r_next01H;
            r_reg1H<=r_next1H;
            r_reg5H<=r_next5H;
            r_reg10H<=r_next10H;
            r_reg20H<=r_next20H;
            r_reg50H<=r_next50H;
            r_reg70H<=r_next70H;
            r_reg100H<=r_next100H;
      end
      // next state logic
      assign r_next01H = (r_reg01H==M)?0:r_reg01H + 1;
      assign r_next1H = (r_reg1H==M/10)?0:r_reg1H + 1;
      assign r_next5H = (r_reg5H==M/50)?0:r_reg5H + 1;
      assign r_next10H = (r_reg10H==M/100)?0:r_reg10H + 1;
      assign r_next20H = (r_reg20H==M/200)?0:r_reg20H + 1;
      assign r_next50H = (r_reg50H==M/300)?0:r_reg50H + 1;
      assign r_next70H = (r_reg70H==M/500)?0:r_reg70H + 1;
      assign r_next100H = (r_reg100H==M/1000)?0:r_reg100H + 1;
      // output logic
      assign q[0]=(r_reg01H<M/2)?0:1;
      assign q[1]=(r_reg1H<M/20)?0:1;
      assign q[2]=(r_reg5H<M/100)?0:1;
      assign q[3]=(r_reg10H<M/200)?0:1;
      assign q[4]=(r_reg20H<M/400)?0:1;
      assign q[5]=(r_reg50H<M/600)?0:1;
      assign q[6]=(r_reg70H<M/1000)?0:1;
      assign q[7]=(r_reg100H<M/2000)?0:1;
endmodule
```

MUX81

```
`timescale 1ns / 1ps

module Mux81(
    input wire [7:0] clk,   // 8 đầu vào
    input wire [2:0] sw,    // 3 bit để chọn đầu vào
    output reg clk_o        // Đầu ra
);
    // Logic cho bộ chọn
    always @(*) begin
        case (sw)
            3'b000: clk_o = clk[0];
            3'b001: clk_o = clk[1];
            3'b010: clk_o = clk[2];
                        3'b011: clk_o = clk[3];
            3'b100: clk_o = clk[4];
            3'b101: clk_o = clk[5];
            3'b110: clk_o = clk[6];
            3'b111: clk_o = clk[7];
            default: clk_o = 1'b0; // Giá trị mặc định
        endcase
    end
endmodule
```

CounterUD

```
`timescale 1ns / 1ps

module CounterUD
    #(parameter N = 8)  // 500,000,000 for 0.1Hz
    (
        input wire clk,       // Tín hiệu clock
        input wire reset,     // Tín hiệu reset
        input wire ud,        // Tín hiệu chọn hướng đếm (1: tăng, 0: giảm)
        input wire pause,     // Tín hiệu dừng đếm
        input wire toggle,    // Tín hiệu đảo trạng thái ngõ ra
        output reg [7:0] q    // Đầu ra 8-bit
    );

       // Khai báo thanh ghi đếm
    reg [N-1:0] r_reg;
    wire [N-1:0] r_next;
    reg toggle_state;        // Trạng thái của toggle để kiểm tra thay đổi

       // Khối thanh ghi: Cập nhật giá trị đếm ở mỗi cạnh dương của clk hoặc khi reset
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            r_reg <= 0;             // Đặt lại bộ đếm về 0 khi reset
            q <= 0;                 // Đặt lại ngõ ra về 0
            toggle_state <= 0;      // Đặt lại trạng thái toggle
        end else begin
                        // Kiểm tra tín hiệu toggle để đảo ngược ngõ ra
            if (!toggle && !toggle_state) begin
                q <= ~q;            // Đảo ngược ngõ ra q
                toggle_state <= 1;  // Đánh dấu toggle đã được kích hoạt
            end else if (!toggle) begin
                toggle_state <= 0;  // Xóa đánh dấu khi toggle không kích hoạt
            end

                        // Xử lý tín hiệu pause
            if (!pause) begin
                r_reg <= r_next;    // Cập nhật giá trị đếm nếu pause không kích hoạt
                q <= r_reg;         // Cập nhật ngõ ra với giá trị đếm
            end
        end
    end

    // Logic tính toán giá trị tiếp theo của bộ đếm
    assign r_next = (ud == 1) ? r_reg + 1 : r_reg - 1;

endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;

NET "UD" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
NET "toggle" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "reset" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
NET "pause" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
```

**Counter8b 1Hz**

TopModule

```
`timescale 1ns / 1ps

module TopModule(
    input wire clk, reset, // clk = 50 MHz
    output wire [7:0] led
);
    wire clk_1hz;

    // Module chia xung tu 50 MHz xuống 1 Hz
    ClockDivider divider(
        .clk(clk),
        .reset(reset),
        .clk_1hz(clk_1hz)
    );

    // Module đếm 8 bit
    Counter8Bit counter(
        .clk_1hz(clk_1hz),
        .reset(reset),
        .q(led)
    );
endmodule
```

Clock_div

```
`timescale 1ns / 1ps

module ClockDivider(
    input wire clk, reset, // clk = 50 MHz
    output reg clk_1hz
);
    // Chia tần số từ 50 MHz xuống 1 Hz
    reg [25:0] counter; // Số bit đủ để đếm tới 50,000,000

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            counter <= 0;
            clk_1hz <= 0;
        end
        else if (counter == 49_999_999) begin
            counter <= 0;
            clk_1hz <= ~clk_1hz; // Đảo trạng thái để tạo xung 1 Hz
        end
              else
            counter <= counter + 1;
    end
endmodule
```

Counter 8bits

```
`timescale 1ns / 1ps

module Counter8Bit(
    input wire clk_1hz, reset,
    output reg [7:0] q // 8 bit đầu ra để hiển thị trên 8 LED
);
    always @(posedge clk_1hz or posedge reset) begin
        if (reset)
            q <= 8'b0; // Reset giá trị về 0
        else
            q <= q + 1; // Đếm lên mỗi chu kỳ của xung clk_1hz
    end
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Dethi1 Adder_3bit**

Top_module

```
`timescale 1ns / 1ps

module top_module(
      input wire [2:0]A,
      input wire [2:0]B,
   input wire     Cin,
      output wire [2:0]Sum,
      output wire Cout
    );
wire Cin_01Hz;
Cin_01Hz    IC1(Cin, Cin_01Hz);
Adder_3bits IC2(A, B, Cin_01Hz, Cout, Sum);
endmodule
```

IC1

```
`timescale 1ns / 1ps

module Cin_01Hz
#(parameter N= 29, M = 500000000) // 500,000,000 for 0.1Hz
      ( input wire clk,
      output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg==M)?0:r_reg + 1;
      // output logic
      assign f=(r_reg<M/2)?0:1;
endmodule
```

IC2

```
`timescale 1ns / 1ps

module Adder_3bits(
      input wire [2:0] A,
      input wire [2:0] B,
      input wire Cin,
      output wire Cout,
      output wire [2:0] Sum
      );
wire c1, c2;
      Full_adder add0 (A[0], B[0], Cin ,Sum[0],c1 );
      Full_adder add1 (A[1], B[1], c1 ,Sum[1],c2 );
      Full_adder add3 (A[2], B[2], c2 ,Sum[2],Cout );
endmodule
```

Full_adder

```
`timescale 1ns / 1ps

module Full_adder(
input wire a,b,ci,
output wire s,co
);
assign s = a^b^ci ;
assign co = ((a^b)&ci) | (a&b) ;
endmodule
```

```
`timescale 1ns / 1ps

module Kiem_tra_1_test;

      // Inputs
      reg [2:0] A;
      reg [2:0] B;
      reg Cin;

      // Outputs
      wire [2:0] Sum;
      wire Cout;

      // Instantiate the Unit Under Test (UUT)
      top_module DUT (
            .A(A),
            .B(B),
            .Cin(Cin),
            .Sum(Sum),
            .Cout(Cout)
      );

      // Tạo tín hiệu đồng hồ 50MHz
    initial Cin = 0;
    always #10 Cin = ~Cin; // 50MHz -> 20ns chu kỳ

    // Tạo tín hiệu reset
    //initial begin
    //    reset = 1;
    //    #50 reset = 0;
    //end

    // Test case
    initial begin
        A = 3'b000; B = 3'b111; // Trạng thái ban đầu

        #100 A = 3'b111; B = 3'b000; // Thay đổi trạng thái A và B
        #100 A = 3'b000; B = 3'b111; // Quay lại trạng thái ban đầu

        #1000 $finish; // Kết thúc mô phỏng
    end
endmodule
```

```
`timescale 1ns / 1ps

module Test_2;

      // Inputs
      reg [2:0] A;
      reg [2:0] B;
      reg Cin;

      // Outputs
      wire Cout;
      wire [2:0] Sum;

      // Instantiate the Unit Under Test (UUT)
      Adder_3bits uut (
            .A(A),
            .B(B),
            .Cin(Cin),
            .Cout(Cout),
            .Sum(Sum)
      );

      always begin
            Cin=~Cin;#100;
      end
      initial begin
            // Initialize Inputs
            Cin=1'b0;
            A = 3'd0;
            B = 3'd7;
            Cin = 0;

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

            A = 3'd7;
            B = 3'd0;

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

            A = 3'd0;
            B = 3'd7;

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

            // Add stimulus here

      end

endmodule
```

**Decoder24**

```
`timescale 1ns / 1ps

module decoder24(
      input wire [1:0] w,
      output reg [3:0] y
);
      always @(w)
      case (w)
            0: y = 4'b1000;
            1: y = 4'b0100;
            2: y = 4'b0010;
            3: y = 4'b0001;
      endcase
endmodule
```

ucf

```
# PlanAhead Generated physical constraints

NET "y[3]" LOC = F12;
NET "y[2]" LOC = E12;
NET "y[1]" LOC = E11;
NET "y[0]" LOC = F11;
NET "w[1]" LOC = L14;
NET "w[0]" LOC = L13;

# PlanAhead Generated IO constraints

NET "w[1]" IOSTANDARD = LVTTL;
NET "w[0]" IOSTANDARD = LVTTL;
NET "y[3]" IOSTANDARD = LVTTL;
NET "y[2]" IOSTANDARD = LVTTL;
NET "y[1]" IOSTANDARD = LVTTL;
NET "y[0]" IOSTANDARD = LVTTL;
NET "w[1]" PULLUP;
NET "w[0]" PULLUP;
```

```
`timescale 1ns / 1ps

module test;
      // Inputs
      reg [1:0] w;
      // Outputs
      wire [3:0] y;
      // Instantiate the Unit Under Test (UUT)
      decoder24 uut (
            .w(w),
            .y(y)
      );
      initial begin
            // Initialize Inputs
            w = 0;
            #100;
            w = 1;
            #100;
            w = 2;
            #100;
            w = 3;
            #100;
            // Add stimulus here
      end
endmodule
```

**Dem_len _mod10 _5hz**

```
`timescale 1ns / 1ps

/*Thiết kế mạch đếm lên sản phẩm mod 10 (Đếm từ 0-9)
 Khi đếm đủ thiết lập LED sáng. xuất ngõ ra
 Mạch tích cực cạnh xung lên, với reset tích cực mức thấp
*/
module Top(
      input clk, data, rst,
      output [3:0] q,
      output done
      );
wire ck;
ckdiv ic1(.ckin(clk),.ckout(ck));
COUNTER_MOD10(.clk(ck),.data(data),.rst(rst),.q(q),.done(done));
endmodule
```

ckdiv

```
`timescale 1ns / 1ps

/*chia xung 5Hz*/
module ckdiv(
      input ckin,
      output ckout
      );
      reg [25:0] cnt;
      wire [25:0] cnt_next;

      always@(posedge ckin) begin
            cnt=cnt_next;
      end

      assign cnt_next=(cnt==50_000_000)?0:cnt+1;
      assign ckout = (cnt>5_000_000)?1:0;

endmodule
```

Counter Mod10

```
`timescale 1ns / 1ps

module COUNTER_MOD10(
      input clk, data, rst,
      output [3:0] q,
      output done
    );
reg [3:0]reg1;
reg reg2;
wire [3:0]reg1_next;
wire reg2_next;
always@(posedge clk, negedge rst)begin
      if(rst==0)begin
            reg1=0;
            reg2=0;
      end
      else begin
            if(data==1)begin
                  reg1=reg1_next;
                  reg2=reg2_next;
            end
      end
end

assign reg1_next=(reg1>8)?0:reg1+1;
assign reg2_next=(reg1==8)?1:0;

assign q=reg1;
assign done=reg2;

endmodule
```

```
`timescale 1ns / 1ps

module test;

      // Inputs
      reg clk;
      reg data;
      reg rst;

      // Outputs
      wire [3:0] q;
      wire done;

      // Instantiate the Unit Under Test (UUT)
      COUNTER_MOD10 uut (
            .clk(clk),
            .data(data),
            .rst(rst),
            .q(q),
            .done(done)
      );

      always begin
            clk=~clk;#1;
      end
      initial begin
            // Initialize Inputs
            clk = 0;
            data = 0;
            rst = 0;
            #1;
            rst=1;
            #1;
            data=1;
            // Wait 100 ns for global reset to finish
            #100;

            // Add stimulus here

      end

endmodule
```

**Demux18**

```
`timescale 1ns / 1ps

module demux_18_ASS(
      input [3:0] sw,
      output [7:0] led
    );
demux_18 IC(.i(sw[0]),.s(sw[3:1]),.o(led));

endmodule
```

```
`timescale 1ns / 1ps

module demux_18(
      input i,
      input [2:0] s,
      output [7:0] o
    );
assign o =  (s==3'b000) ? {7'b0000000,i}:
                        (s==3'b001) ? {6'b000000,i,1'b0}:
                        (s==3'b010) ? {5'b00000,i,2'b00}:
                        (s==3'b011) ? {4'b0000,i,3'b000}:
                        (s==3'b100) ? {3'b000,i,4'b0000}:
                        (s==3'b101) ? {2'b00,i,5'b00000}:
                        (s==3'b110) ? {1'b0,i,6'b000000}:
                                                          {i,7'b0000000};
endmodule
```

ucf

```
# PlanAhead Generated physical constraints
NET "sw[3]" LOC = N17;
NET "sw[2]" LOC = H18;
NET "sw[1]" LOC = L14;
NET "sw[0]" LOC = L13;
NET "led[7]" LOC = F9;
NET "led[6]" LOC = E9;
NET "led[5]" LOC = D11;
NET "led[4]" LOC = C11;
NET "led[3]" LOC = F11;
NET "led[2]" LOC = E11;
NET "led[1]" LOC = E12;
NET "led[0]" LOC = F12;

# PlanAhead Generated IO constraints

NET "led[7]" IOSTANDARD = LVTTL;
NET "led[6]" IOSTANDARD = LVTTL;
NET "led[5]" IOSTANDARD = LVTTL;
NET "led[4]" IOSTANDARD = LVTTL;
NET "led[3]" IOSTANDARD = LVTTL;
NET "led[2]" IOSTANDARD = LVTTL;
NET "led[1]" IOSTANDARD = LVTTL;
NET "led[0]" IOSTANDARD = LVTTL;
NET "sw[3]" IOSTANDARD = LVTTL;
NET "sw[2]" IOSTANDARD = LVTTL;
NET "sw[1]" IOSTANDARD = LVTTL;
NET "sw[0]" IOSTANDARD = LVTTL;
NET "sw[3]" PULLDOWN;
NET "sw[2]" PULLDOWN;
NET "sw[1]" PULLDOWN;
NET "sw[0]" PULLDOWN;
```

```
`timescale 1ns / 1ps

module test_demux_18;

      // Inputs
      reg i;
      reg [2:0] s;

      // Outputs
      wire [7:0] o;

      // Instantiate the Unit Under Test (UUT)
      demux_18 uut (
            .i(i),
            .s(s),
            .o(o)
      );

      // Đoạn khởi tạo testbench
    initial begin
        // In ra tiêu đề của testbench
        $display("Time\t i\t s\t o");
        $monitor("%0t\t %b\t %b\t %b", $time, i, s, o);

        // Trường hợp 1: i = 1, s = 000 (kết quả mong đợi: o[0] = 1)
        i = 1; s = 3'b000;
        #10;

        // Trường hợp 2: i = 1, s = 001 (kết quả mong đợi: o[1] = 1)
        i = 1; s = 3'b001;
        #10;

        // Trường hợp 3: i = 1, s = 010 (kết quả mong đợi: o[2] = 1)
        i = 1; s = 3'b010;
        #10;

              // Trường hợp 4: i = 1, s = 011 (kết quả mong đợi: o[3] = 1)
        i = 1; s = 3'b011;
        #10;

        // Trường hợp 5: i = 1, s = 100 (kết quả mong đợi: o[4] = 1)
        i = 1; s = 3'b100;
        #10;

        // Trường hợp 6: i = 1, s = 101 (kết quả mong đợi: o[5] = 1)
        i = 1; s = 3'b101;
        #10;

        // Trường hợp 7: i = 1, s = 110 (kết quả mong đợi: o[6] = 1)
        i = 1; s = 3'b110;
        #10;

              // Trường hợp 8: i = 1, s = 111 (kết quả mong đợi: o[7] = 1)
        i = 1; s = 3'b111;
        #10;

        // Kết thúc testbench
        $finish;
    end
endmodule
```

**Encoder42**

```
`timescale 1ns / 1ps

module Mahoa_42H_2IC(
      input [7:0] sw,
      output [3:0] led
    );
Mahoa_42H_IF   IC1(.i1(sw[3:0]), .o1(led[1:0]));
Mahoa_42H_CASE IC2(.i2(sw[7:4]), .o2(led[3:2]));
endmodule
```

IC1

```
`timescale 1ns / 1ps

module Mahoa_42H_IF(
      input [3:0] i1,
      output [1:0] o1
    );
reg [1:0] ot;
always @*
            if(i1==4'b0000)   ot=2'b00;
else  if(i1==4'b0001)   ot=2'b00;
else  if(i1==4'b0010)   ot=2'b01;
else  if(i1==4'b0100)   ot=2'b10;
else                                      ot=2'b11;

assign o1 = ot;

endmodule
```

IC2

```
`timescale 1ns / 1ps

module Mahoa_42H_CASE(
      input [3:0] i2,
      output [1:0] o2
    );
reg [1:0] ot;
always @*
      case(i2)
            4'b0001:    ot=2'b00;
            4'b0010:    ot=2'b01;
            4'b0100:    ot=2'b10;
            4'b1000:    ot=2'b11;
      endcase
assign o2 = ot;

endmodule
```

ucf

```

# PlanAhead Generated physical constraints

NET "sw[0]" LOC = L13;
NET "sw[1]" LOC = L14;
NET "sw[2]" LOC = H18;
NET "sw[3]" LOC = N17;
NET "sw[4]" LOC = H13;
NET "sw[5]" LOC = K17;
NET "sw[6]" LOC = D18;
NET "sw[7]" LOC = V4;
NET "led[0]" LOC = F12;
NET "led[1]" LOC = E12;
NET "led[2]" LOC = E9;
NET "led[3]" LOC = F9;

# PlanAhead Generated IO constraints

NET "led[3]" IOSTANDARD = LVTTL;
NET "led[2]" IOSTANDARD = LVTTL;
NET "led[1]" IOSTANDARD = LVTTL;
NET "led[0]" IOSTANDARD = LVTTL;
NET "sw[7]" IOSTANDARD = LVTTL;
NET "sw[6]" IOSTANDARD = LVTTL;
NET "sw[5]" IOSTANDARD = LVTTL;
NET "sw[4]" IOSTANDARD = LVTTL;
NET "sw[3]" IOSTANDARD = LVTTL;
NET "sw[2]" IOSTANDARD = LVTTL;
NET "sw[1]" IOSTANDARD = LVTTL;
NET "sw[0]" IOSTANDARD = LVTTL;
NET "sw[7]" PULLDOWN;
NET "sw[6]" PULLDOWN;
NET "sw[5]" PULLDOWN;
NET "sw[4]" PULLDOWN;
NET "sw[3]" PULLUP;
NET "sw[2]" PULLUP;
NET "sw[1]" PULLUP;
NET "sw[0]" PULLUP;
```

```
`timescale 1ns / 1ps

module test;

	// Inputs
	reg [7:0] sw;

	// Outputs
	wire [3:0] led;

	// Instantiate the Unit Under Test (UUT)
	Mahoa_42H_2IC uut (
		.sw(sw), 
		.led(led)
	);

	initial begin
		// Initialize Inputs
		sw = 8'b00010001;
		#100;
		sw = 8'b00100010;
		#100;
		sw = 8'b01000100;
		#100;
		sw = 8'b10001000;
		#100;

		// Add stimulus here

	end

endmodule
```

**FULL_ ADDER4bit**

```
`timescale 1ns / 1ps

module Adder_4bit(
	input wire [3:0] A,
	input wire [3:0] B,
	input wire Cin,
	output wire Cout,
	output wire [3:0] Sum
	);
wire c1, c2, c3;
FullAdder add0 (A[0], B[0], Cin ,Sum[0],c1 );
FullAdder add1 (A[1], B[1], c1 ,Sum[1],c2 );
FullAdder add2 (A[2], B[2], c2 ,Sum[2],c3 );
FullAdder add3 (A[3], B[3], c3 ,Sum[3],Cout );
endmodule
```

```
`timescale 1ns / 1ps

module FullAdder(
input wire a,b,ci,
output wire s,co);
assign s = a^b^ci ;
assign co = ((a^b)&ci) | (a&b) ;
endmodule
```

ucf

```

# PlanAhead Generated physical constraints 

NET "A[3]" LOC = N17;
NET "A[2]" LOC = H18;
NET "A[1]" LOC = L14;
NET "A[0]" LOC = L13;
NET "B[3]" LOC = V4;
NET "B[2]" LOC = D18;
NET "B[1]" LOC = K17;
NET "B[0]" LOC = H13;
NET "Sum[3]" LOC = F11;
NET "Sum[2]" LOC = E11;
NET "Sum[1]" LOC = E12;
NET "Sum[0]" LOC = F12;
NET "Cin" LOC = V16;
NET "Cout" LOC = C11;

# PlanAhead Generated IO constraints 

NET "A[3]" IOSTANDARD = LVTTL;
NET "A[2]" IOSTANDARD = LVTTL;
NET "A[1]" IOSTANDARD = LVTTL;
NET "A[0]" IOSTANDARD = LVTTL;
NET "B[3]" IOSTANDARD = LVTTL;
NET "B[2]" IOSTANDARD = LVTTL;
NET "B[1]" IOSTANDARD = LVTTL;
NET "B[0]" IOSTANDARD = LVTTL;
NET "Sum[3]" IOSTANDARD = LVTTL;
NET "Sum[2]" IOSTANDARD = LVTTL;
NET "Sum[1]" IOSTANDARD = LVTTL;
NET "Sum[0]" IOSTANDARD = LVTTL;
NET "Cin" IOSTANDARD = LVTTL;
NET "Cout" IOSTANDARD = LVTTL;
NET "A[3]" PULLUP;
NET "A[2]" PULLUP;
NET "A[1]" PULLUP;
NET "A[0]" PULLUP;
NET "B[3]" PULLDOWN;
NET "B[2]" PULLDOWN;
NET "B[1]" PULLDOWN;
NET "B[0]" PULLDOWN;
NET "Cin" PULLDOWN;
```

```
`timescale 1ns / 1ps

module Testbench_add;
      // Inputs
      reg [3:0] A;
      reg [3:0] B;
      reg Cin;
      // Outputs
      wire Cout;
      wire [3:0] Sum;
      // Instantiate the Unit Under Test (UUT)
      Adder_4bit uut (
            .A(A),
            .B(B),
            .Cin(Cin),
            .Cout(Cout),
            .Sum(Sum)
            );
      initial begin
            // Initialize Inputs
            A = 2;
            B = 3;
            Cin = 0;
            // Wait 100 ns for global reset to finish
            #100;
            // Add stimulus here
      end
endmodule
```

**LCD_hienthi haihang**

```
`timescale 1ns / 1ps

module LCDtest(
      input wire clk,
      output wire lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7
      );
      wire [256:0] chars;
      // module installation
      LCD lcd( clk,
                        chars,
                        lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7);
      // defnine data memory
      assign chars[255:128] = " HCMUTE ";
      assign chars[127:0] = " 04 - 11 2024 ";
      //assign chars[7:0] = {4'b0011,I3, I2, I1, I0};
endmodule
```

LCD

```
`timescale 1ns / 1ps

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; // next case
                                    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
```

Ucf

```
# clock
NET "clk" LOC = C9 | IOSTANDARD = LVCMOS33 ;
NET "clk" PERIOD = 20.0ns HIGH 50%;
# LCD control inputs
NET "lcd_e" LOC = M18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_rs" LOC = L18 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_rw" LOC = L17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
# The LCD four-bit data interface
NET "lcd_4" LOC = R15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_5" LOC = R16 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_6" LOC = P17 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "lcd_7" LOC = M15 | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

LCDtest

```
`timescale 1ns / 1ps

module LCDtest(
      input wire clk,
      output wire lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7
      );
      wire [256:0] chars;
      // module installation
      LCD lcd( clk,
                        chars,
                        lcd_rs, lcd_rw, lcd_e, lcd_4, lcd_5, lcd_6, lcd_7);
      // defnine data memory
      assign chars[255:128] = " HCMUTE ";
      assign chars[127:0] = " 04 - 11 2024 ";
      //assign chars[7:0] = {4'b0011,I3, I2, I1, I0};
endmodule
```

**LED_dich switch**

```
`timescale 1ns / 1ps

module LED_Controller (
    input wire clk,        // tín hiệu xung nhịp
    input wire switch,     // công tắc điều khiển hướng (0: trái sang phải, 1: phải sang trái)
    output reg [7:0] led   // 8 LED điều khiển
);
```

```
// Đăng ký giá trị ban đầu cho LED (ví dụ: chỉ bật LED đầu tiên)
initial begin
    led = 8'b00000001;
end
Clock_1Hz clockdivider (clk,clk_o) ;
// Khối điều khiển LED chạy
always @(posedge clk_o) begin
    if (switch == 0) begin
            // Nếu switch == 0, LED chạy từ trái sang phải
        led <= (led == 8'b10000000) ? 8'b00000001 : led << 1;
    end else begin
        // Nếu switch == 1, LED chạy từ phải sang trái
        led <= (led == 8'b00000001) ? 8'b10000000 : led >> 1;
    end
end

endmodule
```

Ckdiv

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 30, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg>=M)?0:r_reg + 1;
      // output logic
      //assign f=(r_reg>M/2)?1:0;
      assign f=r_reg[25] ;
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "switch" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Led_dich Tu_dong**

```
`timescale 1ns / 1ps

module LED_Controller (
    input wire clk,        // tín hiệu xung nhịp
    input wire switch,     // công tắc đảo trạng thái ngõ ra
    output reg [7:0] led   // 8 LED điều khiển
);

// Đăng ký giá trị ban đầu cho LED (ví dụ: chỉ bật LED đầu tiên)
reg direction; // Biến điều khiển hướng (0: trái sang phải, 1: phải sang trái)
initial begin
    led = 8'b00000001;
    direction = 0;
end
Clock_1Hz clockdivider (clk,clk_o) ;
// Khối điều khiển LED chạy
always @(posedge clk_o) begin
    // Đảo trạng thái ngõ ra nếu switch được bật (active high)
    if (switch) begin
        led <= ~led;
    end else begin
        // Điều khiển hướng chạy của LED
        if (direction == 0) begin
            // LED chạy từ trái sang phải
            if (led == 8'b10000000) begin
                direction <= 1; // Đảo hướng khi đến cuối
            end else begin
                led <= led << 1; // Dịch sang trái
            end
              end else begin
            // LED chạy từ phải sang trái
            if (led == 8'b00000001) begin
                direction <= 0; // Đảo hướng khi đến đầu
            end else begin
                led <= led >> 1; // Dịch sang phải
            end
        end
    end
end

endmodule
```

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 30, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg>=M)?0:r_reg + 1;
      // output logic
      //assign f=(r_reg>M/2)?1:0;
      assign f=r_reg[25] ;
endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "switch" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Led_sang Dan_trai phai**

```
`timescale 1ns / 1ps

module ShiftRegiter(
            input wire clk,
            output wire [7:0] q );
      wire clk_o ;
      wire s_in;
      // module instance
      Clock_1Hz clockdivider (clk,clk_o) ;
      Shift_SIPO SIPO (clk_o,s_in, q);
      assign s_in = ~q[0] ;
endmodule
```

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 30, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg>=M)?0:r_reg + 1;
      // output logic
      //assign f=(r_reg>M/2)?1:0;
      assign f=r_reg[25] ;
endmodule
```

```
`timescale 1ns / 1ps

module Shift_SIPO
(
      input wire clk,s_in,
      output wire [7:0] q_out
);
      // signal declaration
      reg [7:0] r_reg;
      wire [7:0] r_next;
      // body, register
      always@(negedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = {s_in,r_reg[7:1]};
      // output logic
      assign q_out= r_reg;
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Led_sang Dan_trai_ Phai _switch**

```
`timescale 1ns / 1ps

module ShiftRegiter(
      input wire clk, lr,
      output wire [7:0] q);
      wire clk_o ;
      wire s_in;
      // module instance
      Clock_1Hz clockdivider (clk,clk_o) ;
      Shift_SIPO SIPO (clk_o,s_in,lr, q);
      assign s_in =(lr == 1)?~q[0]: ~q[7] ;
endmodule
```

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 30, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg>=M)?0:r_reg + 1;
      // output logic
      //assign f=(r_reg>M/2)?1:0;
      assign f=r_reg[25] ;
endmodule
```

```
`timescale 1ns / 1ps

module Shift_SIPO
(
      input wire clk,s_in,lr,
      output wire [7:0] q_out
);
      // signal declaration
      reg [7:0] r_reg;
      wire [7:0] r_next;
      // body, register
      always@(negedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next =(lr==1) ? {s_in,r_reg[7:1]}:{r_reg[6:0],s_in};
      // output logic
      assign q_out= r_reg;
endmodule
```

ucf

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "lr" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**MOD10_sw**

```
`timescale 1ns / 1ps

module Mod10Counter(
    input wire clk,       // Clock input
    input wire reset,     // Reset button
    input wire button,    // Button to count
    output reg [3:0] Q    // Output for LEDs
);

// Tín hiệu trung gian cho nút nhấn
reg button_prev;
wire button_edge;
```

```
// Phát hiện cạnh lên của nút nhấn
assign button_edge = button & ~button_prev;
always @(posedge clk or posedge reset) begin
    if (reset) begin
        Q <= 4'b0000;           // Reset bộ đếm về 0
        button_prev <= 0;
    end else begin
        button_prev <= button;  // Cập nhật trạng thái trước của nút nhấn

        // Khi phát hiện cạnh lên của nút nhấn, tăng bộ đếm
        if (button_edge) begin
            if (Q == 4'b1001)   // Nếu Q = 9, quay lại 0
                Q <= 4'b0000;
            else
                Q <= Q + 1;
        end
    end
end

endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;

NET "button" LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN ;

NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**MUX41**

```
`timescale 1ns / 1ps

module mux4to1(
    input wire [1:0] sel,  // Tín hiệu chọn (2 bit)
    input wire [3:0] d,    // Các đầu vào (4 bit)
    output wire y         // Đầu ra
);
    assign y = (sel == 2'b00) ? d[0] :
               (sel == 2'b01) ? d[1] :
               (sel == 2'b10) ? d[2] :
               d[3];  // Trường hợp sel == 2'b11
endmodule
```

```

# PlanAhead Generated physical constraints

NET "d[3]" LOC = N17;
NET "d[2]" LOC = H18;
NET "d[1]" LOC = L14;
NET "d[0]" LOC = L13;
NET "sel[1]" LOC = K17;
NET "sel[0]" LOC = H13;
NET "y" LOC = F12;

# PlanAhead Generated IO constraints

NET "d[3]" IOSTANDARD = LVTTL;
NET "d[2]" IOSTANDARD = LVTTL;
NET "d[1]" IOSTANDARD = LVTTL;
NET "d[0]" IOSTANDARD = LVTTL;
NET "sel[1]" IOSTANDARD = LVTTL;
NET "sel[0]" IOSTANDARD = LVTTL;
NET "y" IOSTANDARD = LVTTL;
NET "d[3]" PULLUP;
NET "d[2]" PULLUP;
NET "d[1]" PULLUP;
NET "d[0]" PULLUP;
NET "sel[1]" PULLDOWN;
NET "sel[0]" PULLDOWN;
```

```
`timescale 1ns / 1ps

module test;

      // Inputs
      reg [1:0] sel;
      reg [3:0] d;

      // Outputs
      wire y;

      // Instantiate the Unit Under Test (UUT)
      mux4to1 uut (
            .sel(sel),
            .d(d),
            .y(y)
```

```
   );

   // Đoạn khởi tạo testbench
  initial begin
    // In ra tiêu đề của testbench
    $display("Time\t sel\t d\t y");
    $monitor("%0t\t %b\t %b\t %b", $time, sel, d, y);

    // Trường hợp 1: sel = 00, d = 4'b0001 (chọn d[0])
    sel = 2'b00; d = 4'b0001;
    #10;

    // Trường hợp 2: sel = 01, d = 4'b0010 (chọn d[1])
    sel = 2'b01; d = 4'b0010;
    #10;

       // Trường hợp 3: sel = 10, d = 4'b0100 (chọn d[2])
    sel = 2'b10; d = 4'b0100;
    #10;

    // Trường hợp 4: sel = 11, d = 4'b1000 (chọn d[3])
    sel = 2'b11; d = 4'b1000;
    #10;

    // Kết thúc testbench
    $finish;
  end
endmodule
```

**State Machine1**

```
`timescale 1ns / 1ps

module Board_test_1Hz(
   input clk_50m, reset, w,
   output z, clk_1hz,
   output [1:0] stt
   );
wire clk_i;
Chiaxung_1hz IC1(clk_50m, clk_i);
Mtt_moore IC2(clk_i, reset, w, z, stt);
assign clk_1hz = clk_i;
endmodule
```

```
`timescale 1ns / 1ps

module Chiaxung_1hz
 #(parameter N= 26)
   ( input wire clk,
   output wire q
   );
   // signal declaration
   reg [N-1:0] r_reg=0;
   wire [N-1:0] r_next;
   // body, register
   always @(posedge clk)
      r_reg<=r_next;
   // next state logic
   assign r_next = r_reg + 1;
   // output logic
   assign q=r_reg[25];
endmodule
```

```
`timescale 1ns / 1ps
module Mtt_moore(
   input Clock, Resetn, w,
   output z,
   output reg [1:0] stt
   );
reg [2:1] y, Y;
initial
   begin
      y=0;
      Y=0;
   end
parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;
// Define the next state combinational circuit
always @(w, y)
   case (y)
   A: if (w) Y = B;
   else Y = A;
   B: if (w) Y = C;
   else Y = A;
   C: if (w) Y = A;
   else Y = C;
   default: Y = 2'bxx;
endcase
```

```
// Define the sequential block
always @(negedge Resetn, posedge Clock)
      if (Resetn == 0)
            begin
                  y <= A;
                  stt<=A;
            end
      else
            begin
                  y <= Y;
                  stt<=Y;
            end
      // Define output
      assign z = (y == B);
endmodule
```

```
NET "clk_50m" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "clk_50m" CLOCK_DEDICATED_ROUTE = FALSE;

NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "w" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;

NET "z" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "clk_1hz" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "stt<0>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "stt<1>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

```
`timescale 1ns / 1ps

module Board_test;

      // Inputs
      reg Clock;
      reg Resetn;
      reg w;

      // Outputs
      wire z;
      wire [1:0] stt;

      // Instantiate the Unit Under Test (UUT)
      Mtt_moore uut (
            .Clock(Clock),
            .Resetn(Resetn),
            .w(w),
            .z(z),
            .stt(stt)
      );

      initial begin
            Clock = 0;
            Resetn = 0;
            w = 0;
            #10;
            Resetn=1;
            w=1;
            #40;
            w=0;
            #20;
            w=1;
            #20;
      end
      //***************************
      always
            begin
                  #10;
                  Clock=~Clock;
            end
      //***************************
endmodule
```

**May trang Thai 1_2**

```
`timescale 1ns / 1ps

module Board_test_1Hz_moore_mealy(
      input clk_50m, reset, a, b,
      output y1, y0, clk_1hz,
      output [1:0] stt
      );
      wire clk_i;
      Chiaxung_1hz       IC1(clk_50m, clk_i);
      MTT_moore_mealy IC2(clk_i, reset, a, b, y1, y0, stt);
      assign clk_1hz = clk_i;
endmodule
```

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

```
module Chiaxung_1hz
  #(parameter N= 26)
      ( input wire clk,
      output wire q
      );
      // signal declaration
      reg [N-1:0] r_reg=0;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = r_reg + 1;
      // output logic
      assign q=r_reg[25];
endmodule
```

```
`timescale 1ns / 1ps

module MTT_moore_mealy
      (
      input wire clk , reset ,
      input wire a , b ,
      output wire yl,y0,
      output reg [1:0] stt
      );
      // stt is current state of module//
      //symbolic state declaration
      localparam [1:0] S0 = 2'b00, S1 = 2'b01 , S2=2'b10;
      // signal declaration
      reg [1 : 0] state_reg,state_next ;
      initial
            begin
                  state_reg<=0;
                  state_next<=0;
            end
      // state register
      always @ (posedge clk ,posedge reset)
            if (reset)
                  begin
                        state_reg<=S0;
                        stt<=S0;
                  end
            else
                  begin
                        state_reg<=state_next;
                        stt<=state_next;
                  end
      //next_state logic
      always @*
            case (state_reg)
                  S0: if(a)
                              if(b)
                                    state_next=S2;
                              else
                                    state_next=S1;
                         else
                              state_next=S0;
                  S1: if(a)
                                    state_next=S1;
                         else
                              state_next=S0;
                  S2: state_next=S0;
                  default: state_next=S0;
            endcase
      //Moore outputlogic
      assign yl=(state_reg==S0);
      //Mealy outputlogic
      assign y0=(state_reg==S0)&a&b;
endmodule
```

```
NET "clk_50m" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "clk_50m" CLOCK_DEDICATED_ROUTE = FALSE;

NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "a" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
NET "b" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;

NET "clk_1hz" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8;
NET "y1" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "y0" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

NET "stt<1>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "stt<0>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

```
`timescale 1ns / 1ps

module Board_test_bench;

      // Inputs
      reg clk;
      reg reset;
      reg a;
      reg b;

      // Outputs
      wire yl;
      wire y0;
      wire [1:0] stt;

      // Instantiate the Unit Under Test (UUT)
      MTT_moore_mealy uut (
            .clk(clk),
            .reset(reset),
            .a(a),
            .b(b),
            .yl(yl),
            .y0(y0),
            .stt(stt)
      );

      initial begin
            // Initialize Inputs
            clk = 0;
            //****************************
            reset = 1; //current state is S0
            a = 1;
            b = 1;
            #10;
            //****************************
            reset = 1;
            a = 0;
            b = 0;
            #20;
            reset = 0;
            a=0;
            #20;
            a=1;
            b=1;
            #20;
            a=1;
            b=1;
            #20;
            a=1;
            b=0;
            #20;
            a=1;
            #20;
            a=0;
            #20;
            end
            //***************************
            always
                  begin
                        #10;
                        clk=~clk;
                  end
            //***************************
endmodule
```

**Syncounter 4bits**

```
`timescale 1ns / 1ps

module SynCounter4bit(
      input wire clk, reset,
      output wire [3:0] q
);
// signal declaration
reg [3:0] r_reg;
wire [3:0] r_next;
// body, register
always @(posedge clk, posedge reset)
      if (reset)
            r_reg <= 0;
      else
            r_reg<=r_next; // <= is non-blocking statement
      // next state logic
      assign r_next = r_reg + 1;
      // output logic
      assign q = r_reg;
endmodule
```

```
NET "clk" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
NET "reset" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "q<0>" LOC = "F12"| IOSTANDARD = LVTTL;
NET "q<1>" LOC = "E12"| IOSTANDARD = LVTTL;
NET "q<2>" LOC = "E11"| IOSTANDARD = LVTTL ;
NET "q<3>" LOC = "F11"| IOSTANDARD = LVTTL;
```

```
`timescale 1ns / 1ps

module TestCounter;
      // Inputs
      reg clk;
      reg reset;
      // Outputs
      wire [3:0] q;
      // Instantiate the Unit Under Test (UUT)
      SynCounter4bit uut (
            .clk(clk),
            .reset(reset),
            .q(q)
      );
      integer i;

      initial begin
            // Initialize Inputs
            clk = 0;
            reset = 0;
            #10;
            reset = 1;
            #10;
            reset = 0;
            // Wait 10 ns for global reset to finish
            #10;
            for (i=0;i<32;i=i+1)
            #10 clk = ~clk;
            // Add stimulus here
      end
endmodule
```

**8led_button FSM**

```
`timescale 1ns / 1ps

module FSM(
            input wire reset,clk,btn,
            output wire [7:0] q
      );
      wire tick;
      button btn1(reset, clk,btn,tick);
      Counter8bs counter(tick, reset,q);
endmodule
```

```
`timescale 1ns / 1ps

module button(
            input wire reset,clk,btn,
            output reg db
      );
      localparam [2:0]
      zero = 3'b000,
      wait1_1 = 3'b001,
      wait1_2 = 3'b010,
      wait1_3= 3'b011,
      one = 3'b100,
      wait0_1= 3'b101,
      wait0_2 = 3'b110,
      wait0_3 = 3'b111;
      localparam N = 13;
      //signal declaration
      reg [N-1:0] q_reg;
      wire [N-1:0] q_next;
      wire m_tick;
      reg [2:0] state_reg, state_next;
      // counter to generate 10ms tick
      always @(posedge clk)
            q_reg <=q_next;
      // next state logic
      assign q_next = q_reg +1 ;
      // output tick
      assign m_tick = (q_reg==0)?1'b1:1'b0;
      //debouncing FSM
      //state register
      always @(posedge clk, posedge reset)
```

```
        if(reset)
            state_reg <= zero;
        else
            state_reg<= state_next;
    // next state logic and output logic
    always @*
        begin state_next = state_reg;// default state
        db = 1'b0;
        case (state_reg)
            zero:
                if(btn)
                    state_next = wait1_1;
            wait1_1:
                if (~btn)
                    state_next = zero;
                else
                    if (m_tick)
                        state_next = wait1_2;
            wait1_2:
                if (~btn)
                    state_next = zero;
                else
                    if (m_tick)
                        state_next = wait1_3;
            wait1_3:
                    if (~btn)
                        state_next = zero;
                    else
                        if (m_tick)
                            state_next = one;
            one:
                begin
                    db = 1'b1;
                    if(~btn)
                        state_next = wait0_1;
                end
            wait0_1:
                begin
                    db = 1'b1;
                    if (btn)
                        state_next = one;
                    else
                        if (m_tick)
                            state_next = wait0_2;
                end
            wait0_2:
                begin
                    db = 1'b1;
                    if (btn)
                        state_next = one;
                    else
                        if (m_tick)
                            state_next = wait0_3;
                end
            wait0_3:
                begin
                    db = 1'b1;
                    if(btn)
                        state_next = one;
                    else
                        if (m_tick)
                            state_next = zero;
                end
            default: state_next = zero;
        endcase
    end
endmodule
```

```
`timescale 1ns / 1ps

module Counter8bs
    #(parameter N= 8) // 500,000,000 for 0.1Hz
        ( input wire clk,reset,
        output wire [7:0]q
    );
    // signal declaration
    reg [N-1:0] r_reg;
    wire [N-1:0] r_next;
    // body, register
    always @(posedge clk, posedge reset)
        if (reset)
            r_reg<=0;
            else
                  r_reg<=r_next;
      // next state logic
      assign r_next = r_reg + 1;
      // output logic
      assign q=r_reg;
endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP ;
NET "btn" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Tao_xung_1Hz**

```
`timescale 1ns / 1ps

module Counter
      #(parameter N= 26, M = 50000000)
            ( input wire clk, reset,
            output wire q
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk, posedge reset)
            if (reset)
                  r_reg <= 0;
            else
                  r_reg<=r_next;
            // next state logic
            assign r_next = (r_reg==M)?0:r_reg + 1;
      // output logic
      assign q=(r_reg<M/2)?0:1;
endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
NET "q" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```

**Thanh_ghi_Dich_ntnt dongbo**

```
`timescale 1ns / 1ps

module ShiftRegiter(
      input wire clk,
      input wire s_in,
      output wire s_out
);
wire clk_o ;
// module instance
Clock_1Hz clockdivider (clk, clk_o) ;
Shift_SISO SISO (clk_o,s_in,s_out);
endmodule
```

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 26, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg==M)?0:r_reg + 1;
      // output logic
      assign f=(r_reg<M/2)?0:1;
endmodule
```

```
`timescale 1ns / 1ps

module Shift_SISO
      #(parameter N= 4) // 500,000,000 for 0.1Hz
            ( input wire clk,reset,s_in,
            output wire s_out
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk, posedge reset)
            if (reset)
                  r_reg<=0;
            else
                  r_reg<=r_next;
      // next state logic
      assign r_next = {s_in,r_reg[N-1: 1]};
      // output logic
      assign s_out= r_reg[0];
endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "s_out" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "s_in" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
```

**Thanh_ghi_Dich_ntss Dong bo**

```
`timescale 1ns / 1ps

module ShiftRegiter(
      input wire clk,
      input wire s_in,
      output wire [7:0] q
);
wire clk_o ;
// module instance
Clock_1Hz clockdivider (clk,clk_o) ;
Shift_SIPO SIPO (clk_o,s_in, q);
endmodule
```

```
`timescale 1ns / 1ps

module Clock_1Hz
      #(parameter N= 30, M = 50000000) // for 50Mhz
            ( input wire clk,
            output wire f
      );
      // signal declaration
      reg [N-1:0] r_reg;
      wire [N-1:0] r_next;
      // body, register
      always @(posedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = (r_reg>=M)?0:r_reg + 1;
      // output logic
      //assign f=(r_reg>M/2)?1:0;
      assign f=r_reg[25] ;
endmodule
```

```
`timescale 1ns / 1ps

module Shift_SIPO
(
      input wire clk,s_in,
      output wire [7:0] q_out
);
      // signal declaration
      reg [7:0] r_reg;
      wire [7:0] r_next;
      // body, register
      always@(negedge clk)
            r_reg<=r_next;
      // next state logic
      assign r_next = {s_in,r_reg[7:1]};
      // output logic
      assign q_out= r_reg;
endmodule
```

```
NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
NET "s_in" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ; // SW2
NET "q<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "q<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
```