ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

Thiết kế luận lý với HDL (TN) – CO1026

Báo cáo LAB 3

Nhóm 7	L01_23207
Lê Đình Anh Tài	2312994
Phạm Công Võ	2313946
Lê Thanh Phong	2312618
Trần Minh Dương	2310609
Bùi Tiến An	2310002

MỤC LỤC

MỤC LỤC **2**

BÀI 1 **3**

Code 4

Test bench 4

Mô phỏng 5

Hiện thực 5

BÀI 2 **6**

2a. Code + testbench 6

2b. Code 7

Testbench 9

Mô phỏng + hiện thực 10

BÀI 3 **11**

Code 11

Test bench 14

Mô phỏng 15

Hiện thực 16

BÀI 1

Thiết kế mạch tạo tín hiệu đầu ra 1 Hz bằng Mô hình hành vi. Tín hiệu này được nối với 2 đèn LED RGB (1 hiển thị màu xanh, 1 hiển thị màu đỏ) trên Arty-Z7 FPGA để thực hiện chớp tắt xen kẽ nhau (bật 0,5s - tắt 0,5s). Biết rằng tần số xung nhịp đầu vào là 125 MHz.

Viết các băng ghế thử nghiệm để mô phỏng các mạch.

Kiểm tra các mạch trên bo mạch FPGA bằng đèn LED và đèn LED RGB.

Bài làm:

Bài tập trên là một nhiệm vụ liên quan đến việc thiết kế mạch chia tần số xung nhịp được thiết kế riêng để tạo ra tín hiệu đầu ra 1 Hz. Đầu ra này nhằm mục đích điều khiển hai đèn LED RGB trên bo mạch Artix-7 FPGA (Arty-Z7 FPGA), trong đó một đèn LED hiển thị màu xanh lam và đèn LED còn lại hiển thị màu đỏ. Mục tiêu là làm cho các đèn LED này nhấp nháy luân phiên theo cách bắt chước tiếng còi báo động của cảnh sát – mỗi đèn LED bật trong 0,5 giây và sau đó tắt trong 0,5 giây. Nhiệm vụ này yêu cầu xử lý tần số xung nhịp đầu vào cao 125 MHz, tần số này phải được chia chính xác xuống tốc độ chậm hơn nhiều là 1 Hz.

Để tạo ra một giải pháp hoàn chỉnh cho mạch còi báo động của cảnh sát được mô tả trong bài tập của bạn, chúng tôi có thể chia nó thành một số bước chính. Chúng bao gồm thiết kế bộ chia tần số, triển khai logic điều khiển LED và thử nghiệm thiết kế trong mô phỏng và trên bo mạch FPGA. Hãy phác thảo từng phần:

1.1 Hiểu các yêu cầu

- Nhiệm vụ yêu cầu thiết kế một mạch lấy xung nhịp đầu vào 125 MHz và tạo ra tín hiệu đầu ra 1 Hz. Tín hiệu đầu ra này sẽ điều khiển hai đèn LED RGB theo cách chúng luân phiên nhau, mô phỏng còi báo động của cảnh sát với mỗi đèn LED (một màu xanh, một màu đỏ) nhấp nháy trong 0,5 giây bật và 0,5 giây tắt.

1.2 Thiết kế bộ chia tần

Nhiệm vụ chính là giảm xung nhịp đầu vào 125 MHz xuống tín hiệu đầu ra 1Hz. Điều này có thể đạt được bằng cách sử dụng bộ đếm trong FPGA:

- **Triển khai bộ đếm** : Bộ đếm cần đếm tới 2×1 Hz 125 MHz = 62,5 triệu để tạo tín hiệu 1 Hz từ đồng hồ 125 MHz vì mỗi lần chuyển đổi đầu ra yêu cầu một nửa số này, chiếm các trạng thái cao và thấp của tín hiệu 1Hz. Điều này đòi hỏi một bộ đếm có khả năng đếm từ 0 đến 62.499.999.
- **Tính toán cho Bộ chia** : Đồng hồ đầu vào chạy ở tần số 125 MHz, nghĩa là nó tích tắc 125.000.000 lần mỗi giây. Để có được tín hiệu 1 Hz, chúng ta cần chia tín hiệu này cho 125 triệu. Tuy nhiên, vì chúng ta cần đèn LED chuyển đổi cứ sau 0,5 giây (không chỉ một lần mỗi giây đối với

1 Hz), nên tần số hiệu dụng mà chúng ta cần tạo ra là 2 Hz (một lần chuyển đổi cứ sau nửa giây). Điều này yêu cầu chia đồng hồ đầu vào cho 62,5 triệu.

1.3 Logic điều khiển LED

Bạn cần tạo logic để xen kẽ các đèn LED:

- **Đèn LED xen kẽ** : Sử dụng tín hiệu 1 Hz để luân phiên giữa hai đèn LED. Khi tín hiệu cao, hãy bật một đèn LED (ví dụ: màu xanh lam) và khi tín hiệu ở mức thấp, hãy bật đèn LED còn lại (ví dụ: màu đỏ).

1.4 Mã Verilog cho FPGA

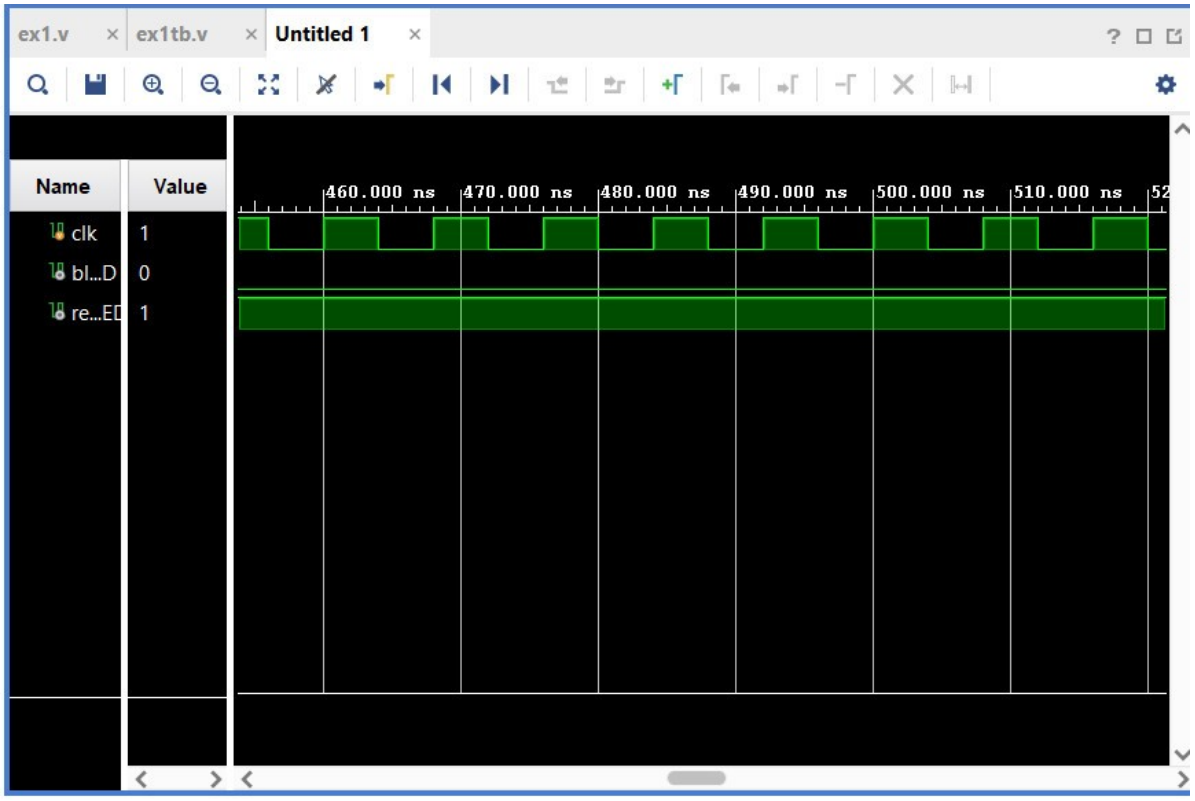
```
module PoliceSiren(
    input clk,        // 125 MHz clock input
    output reg blueLED,
    output reg redLED
);
    // Define the counter
    reg [25:0] counter = 0;  // 26-bit counter to store up to 62.5 million
    reg toggle = 0;          // Toggle signal for the LEDs

    always @(posedge clk) begin
        if (counter == 62499999) begin
            counter <= 0;
            toggle <= ~toggle;  // Toggle the LED control signal
        end else begin
            counter <= counter + 1;
        end
    end

    always @(*) begin
        if (toggle) begin
            blueLED = 1;
            redLED = 0;
        end else begin
            blueLED = 0;
            redLED = 1;
        end
    end
endmodule
```

1.5 Test bench

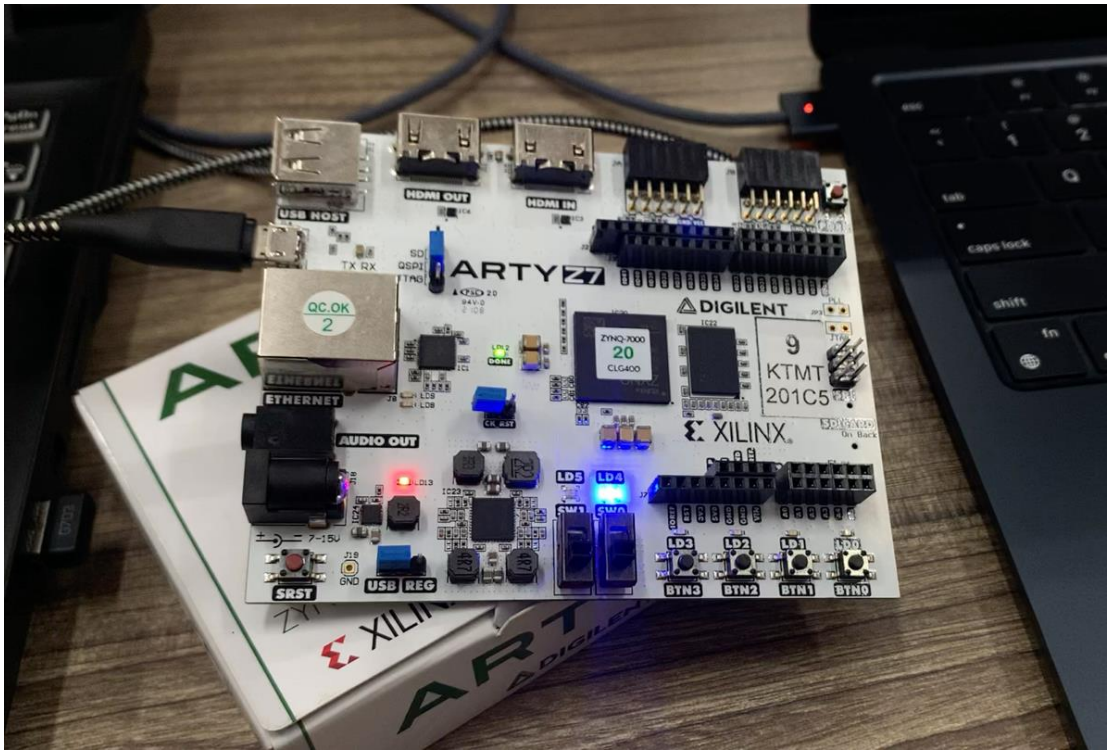
```
module testPoliceSiren;
    reg clk = 0;
    wire blueLED, redLED;

    PoliceSiren ps(clk, blueLED, redLED);

    // Generate clock
    always #4 clk = ~clk;  // Toggle every 4 ns (for simulation speed)
```

```
        initial begin
            #1000;  // Run simulation for 1000 ns to observe behavior
            $finish;
        end
    endmodule
```

1.6 Mô phỏng



1.7 Hiện thực



BÀI 2

a. Design a Rising Edge Detection circuit (Mạch phát hiện cạnh lên):

Module này sử dụng 2 flip-flop để phát hiện cạnh lên của tín hiệu đầu vào. Đầu ra (out) sẽ ở mức cao trong một chu kỳ xung nhịp khi phát hiện cạnh lên.

Bài làm:

RTL Code for Rising Edge Detector:

```
module RisingEdgeDetector (
    input wire clk,
    input wire rst,
    input wire in,
    output reg out
);
    reg in_d1, in_d2;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            in_d1 <= 1'b0;
            in_d2 <= 1'b0;
            out <= 1'b0;
        end else begin
            in_d1 <= in;
            in_d2 <= in_d1;
            out <= in_d1 & ~in_d2;  // Detect rising edge
        end
    end
endmodule
```

Testbench:

```
module tb_RisingEdgeDetector;
    reg clk;
    reg rst;
    reg in;
    wire out;

    RisingEdgeDetector uut (
        .clk(clk),
```

```
    .rst(rst),
    .in(in),
    .out(out)
);

initial begin
    // Initialize inputs
    clk = 0;
    rst = 0;
    in = 0;

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

    // Generate test signals
    #20 in = 1; // Rising edge
    #20 in = 0;
    #40 in = 1; // Rising edge
    #20 in = 0;

    // Finish simulation
    #100;
    $finish;
end

always #5 clk = ~clk; // Clock generator with period of 10 time units
endmodule
```

b. 4-bit Binary Counter with Button Edge Detection

Module này là một bộ đếm nhị phân 4-bit đơn giản. Bộ đếm sẽ tăng giá trị đếm lên 1 ở mỗi chu kỳ xung nhịp khi tín hiệu enable ở mức cao.

RTL Code for 4-bit Binary Counter:

```
module Counter (
    input wire clk,
    input wire rst,
    input wire enable,
    output reg [3:0] count
```

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

Top Module (Module chính):

Mô-đun này kết hợp Rising Edge Detection circuit và Counter. Rising Edge Detection circuit tạo ra tín hiệu enable cho bộ đếm khi nút được nhấn.

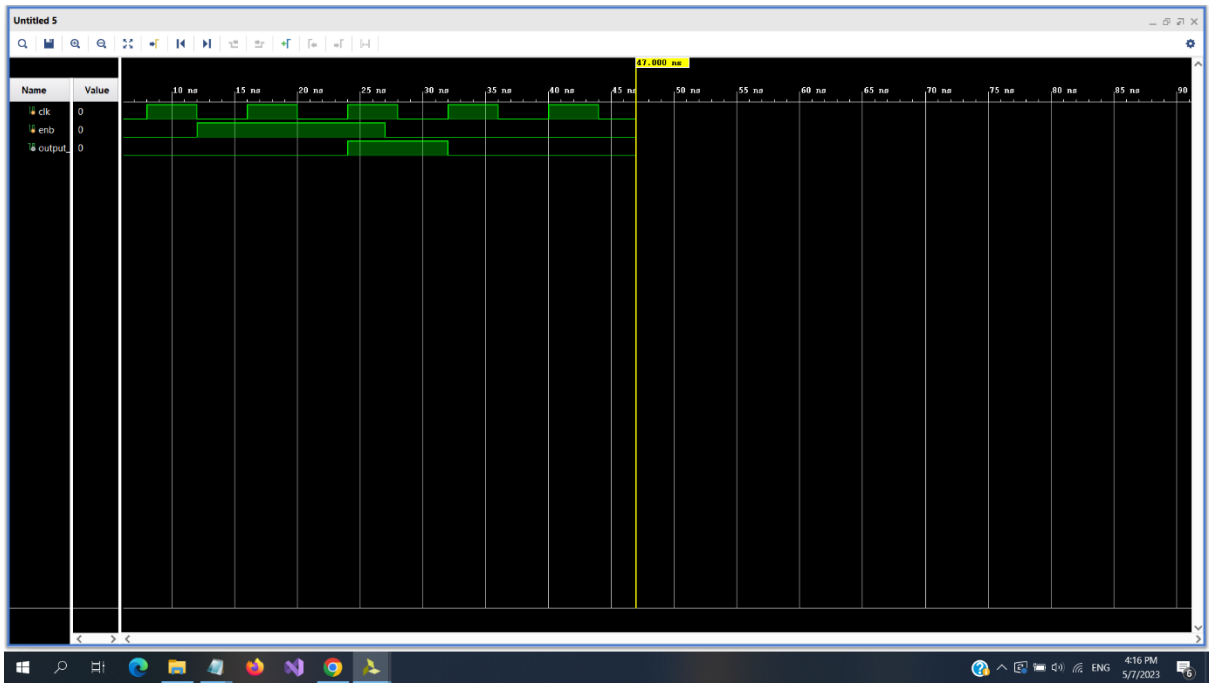
Top Module Combining Edge Detector and Counter

```
module TopModule (
    input wire clk,
    input wire rst,
    input wire button,
    output wire [3:0] count
);
    wire enable;

    RisingEdgeDetector red (
        .clk(clk),
        .rst(rst),
        .in(button),
        .out(enable)
    );

    Counter counter (
        .clk(clk),
        .rst(rst),
        .enable(enable),
        .count(count)
    );
endmodule
```

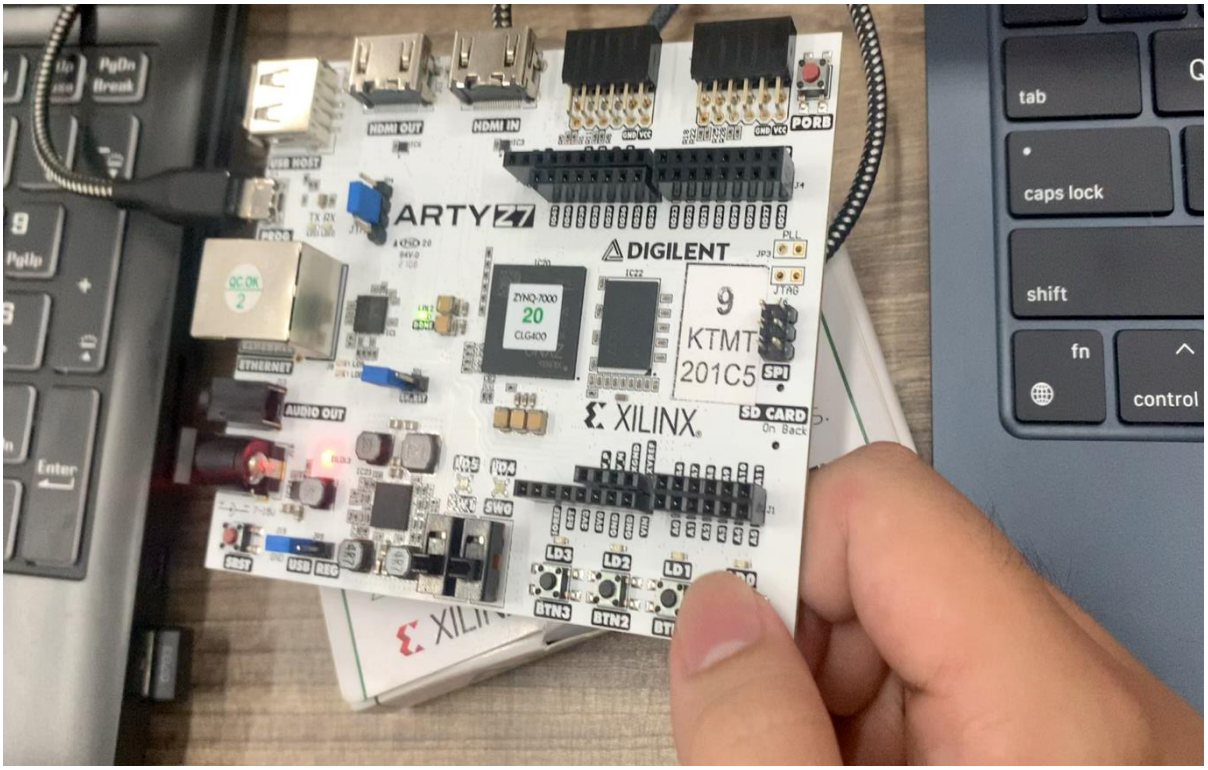
Testbench for the Top Module

```
module tb_TopModule;
    reg clk;
    reg rst;
    reg button;
    wire [3:0] count;

    TopModule uut (
        .clk(clk),
        .rst(rst),
        .button(button),
        .count(count)
    );

    initial begin
        // Initialize inputs
        clk = 0;
        rst = 0;
        button = 0;

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

        // Generate test signals for button press
        #20 button = 1; // Button press (rising edge)
        #10 button = 0;
        #40 button = 1; // Button press (rising edge)
        #10 button = 0;
        #40 button = 1; // Button press (rising edge)
        #10 button = 0;

        // Finish simulation
        #100;
        $finish;
    end

    always #5 clk = ~clk; // Clock generator with period of 10 time units
endmodule
```

Mô phỏng



Hiện thực



BÀI 3

Để thiết kế máy trạng thái (FSM) để thay đổi chế độ hiển thị của một chuỗi bit bằng cách sử dụng Verilog HDL, cần làm theo các bước sau:

Bước 1: Phân chia thiết kế thành các khối:

FSM (Finite State Machine): Quản lý các chế độ khác nhau dựa trên các nút bấm.

Logic hiển thị chuỗi: Xử lý hiển thị của chuỗi bit dựa trên chế độ hiện tại.

Chuỗi bit được lưu trữ trong string_reg.

Tín hiệu one_sec_pulse tạo ra một xung mỗi giây bằng cách sử dụng bộ đếm.

Trong mỗi trạng thái, string_reg được cập nhật theo trạng thái hiện tại mỗi giây.

Bước 2: Sơ đồ trạng thái

FSM sẽ có các trạng thái sau:

RESET: Hiển thị chuỗi 4-bit mặc định.

SHIFT_LEFT: Dịch vòng sang trái.

SHIFT_RIGHT: Dịch vòng sang phải.

PAUSE: Tạm dừng hiển thị hiện tại.

Bước 3: Định nghĩa các đầu vào và đầu ra:

Đầu vào:

‘clk’ : Tín hiệu xung nhịp.

‘rst’ : Tín hiệu đặt lại để khởi tạo chuỗi mặc định.

‘button[3:0]’: Các nút bấm để thay đổi chế độ.

Đầu ra:

‘led[3:0]’: Đầu ra 4-bit để điều khiển các đèn LED.

FSM Design:

Định nghĩa trạng thái của FSM

```
typedef enum logic [1:0] {
   RESET       = 2'b00,
   SHIFT_LEFT  = 2'b01,
   SHIFT_RIGHT = 2'b10,
```

```
    PAUSE       = 2'b11
} state_t;
```

Verilog Code cho FSM và String Display Logic

```
module LED_Display_FSM (
    input wire clk,
    input wire rst,
    input wire [3:0] button,
    output reg [3:0] led
);
    // State encoding
    typedef enum logic [1:0] {
        RESET       = 2'b00,
        SHIFT_LEFT  = 2'b01,
        SHIFT_RIGHT = 2'b10,
        PAUSE       = 2'b11
    } state_t;

    state_t current_state, next_state;
    reg [3:0] string_reg, default_string;
    reg [31:0] counter;
    wire one_sec_pulse;

    // One second pulse generator
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            counter <= 32'd0;
        end else begin
            counter <= (counter == 32'd49_999_999) ? 32'd0 : counter + 32'd1;
        end
    end
    assign one_sec_pulse = (counter == 32'd49_999_999);

    // State transition logic
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            current_state <= RESET;
        end else begin
            current_state <= next_state;
        end
    end
```

```
// Next state logic
always @(*) begin
    case (current_state)
        RESET: begin
            if (button[1]) next_state = SHIFT_LEFT;
            else if (button[2]) next_state = SHIFT_RIGHT;
            else if (button[3]) next_state = PAUSE;
            else next_state = RESET;
        end
        SHIFT_LEFT: begin
            if (button[0]) next_state = RESET;
            else if (button[2]) next_state = SHIFT_RIGHT;
            else if (button[3]) next_state = PAUSE;
            else next_state = SHIFT_LEFT;
        end
        SHIFT_RIGHT: begin
            if (button[0]) next_state = RESET;
            else if (button[1]) next_state = SHIFT_LEFT;
            else if (button[3]) next_state = PAUSE;
            else next_state = SHIFT_RIGHT;
        end
        PAUSE: begin
            if (button[0]) next_state = RESET;
            else if (button[1]) next_state = SHIFT_LEFT;
            else if (button[2]) next_state = SHIFT_RIGHT;
            else next_state = PAUSE;
        end
    endcase
end

// Output logic and string register update
always @(posedge clk or posedge rst) begin
    if (rst) begin
        string_reg <= 4'b0011; // Default string
        default_string <= 4'b0011;
        led <= 4'b0011;
    end else if (one_sec_pulse) begin
        case (current_state)
            RESET: begin
                string_reg <= default_string;
            end
```

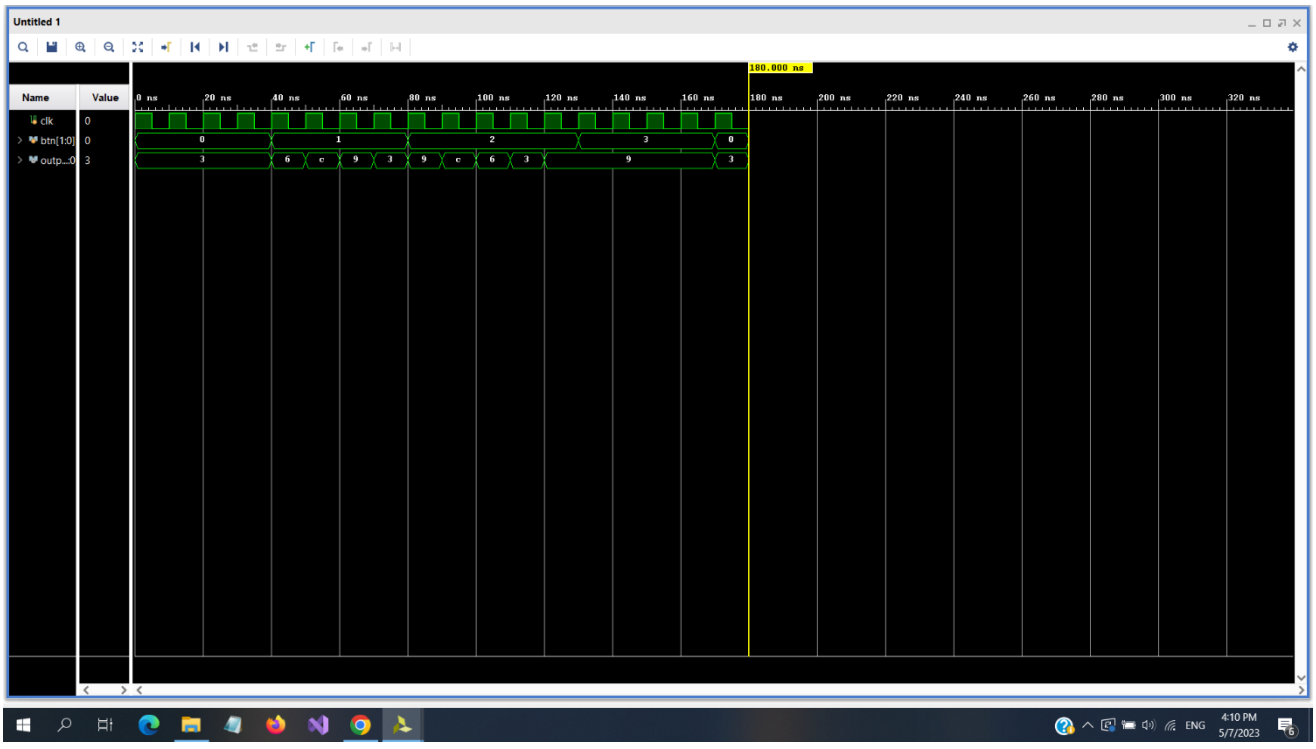
```
            SHIFT_LEFT: begin
                string_reg <= {string_reg[2:0], string_reg[3]};
            end
            SHIFT_RIGHT: begin
                string_reg <= {string_reg[0], string_reg[3:1]};
            end
            PAUSE: begin
                // Do nothing, keep current state
            end
        endcase
        led <= string_reg;
    end
  end
endmodule
```

Testbench:

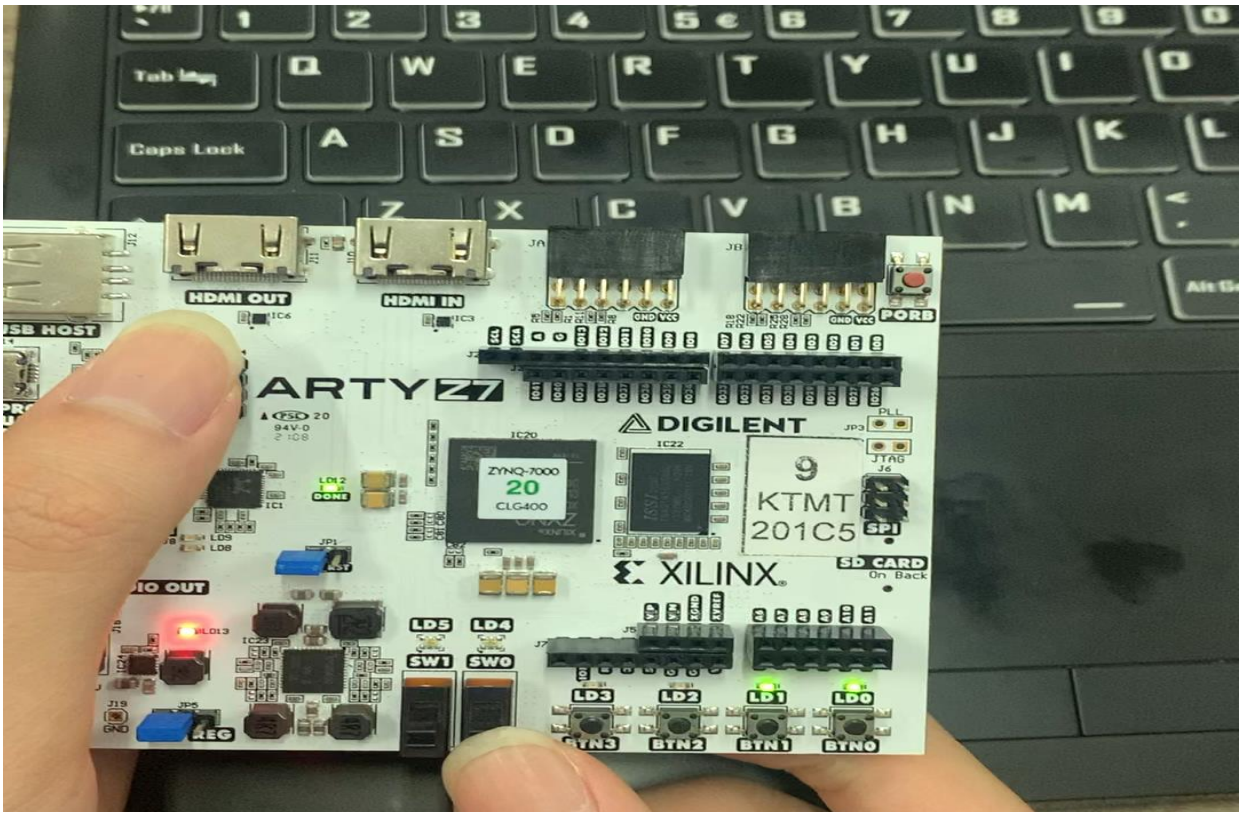
```
module tb_LED_Display_FSM;
  reg clk;
  reg rst;
  reg [3:0] button;
  wire [3:0] led;

  LED_Display_FSM uut (
    .clk(clk),
    .rst(rst),
    .button(button),
    .led(led)
  );

  initial begin
    // Initialize inputs
    clk = 0;
    rst = 0;
    button = 4'b0000;

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

```
        // Test Reset mode
        #20 button[0] = 1; #10 button[0] = 0;

        // Test Circular Shift Left mode
        #20 button[1] = 1; #10 button[1] = 0;
        #50000000; // Wait for 1 second

        // Test Circular Shift Right mode
        #20 button[2] = 1; #10 button[2] = 0;
        #50000000; // Wait for 1 second

        // Test Pause mode
        #20 button[3] = 1; #10 button[3] = 0;
        #50000000; // Wait for 1 second

        // Finish simulation
        #100000000;
        $finish;
    end

    always #5 clk = ~clk; // Clock generator with period of 10 time units
endmodule
```

Mô phỏng



Hiện thực


HDMI OUT
HDMI IN
ARTY Z7
DIGILENT
ZYNQ-7000
20
CLG400
XILINX
KTMT
201C5
SD CARD
LD5
LD4
SW1
SW0
LD3
LD2
LD1
LD0
BTN3
BTN2
BTN1
BTN0
PORB
JTAG
SPI