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 2

GVHD:Phạm Công Thái

Nhóm 7: (23207)
Lê Đình Anh Tài – 2312994
Phạm Công Võ – 2313946
Lê Thanh Phong – 2312618
Trần Minh Dương – 2310609
Bùi Tiến An - 2310002

Bài 1:

Thiết kế mạch có một đầu vào 4 bit và đầu ra 4 bit với các chức năng như sau (tất cả đầu ra đều hoạt động HIGH):

- **Đầu ra 0: hoạt động khi có số bit 1 chẵn trong đầu vào.**
- **Đầu ra 1: hoạt động khi chỉ có 1 bit 1 ở đầu vào.**
- **Đầu ra 2: hoạt động khi không có bit 1 trong đầu vào**
- **Đầu ra 3: hoạt động khi tất cả các bit trong đầu vào là 1.**

Viết mã RTL Verilog HDL và test bench cho thiết kế.

Kiểm tra mạch trên FPGA bằng nút và đèn LED.

Gợi ý: Học sinh nên sử dụng các toán tử rút gọn.

Để triển khai mạch, chúng ta có thể sử dụng logic tổ hợp lấy đầu vào 4 bit và tạo ra các đầu ra cần thiết dựa trên các điều kiện đã cho. Chúng ta có thể sử dụng các toán tử rút gọn trong Verilog HDL để đơn giản hóa logic.

Đây là mã Verilog HDL RTL cho mạch:

```
module circuit(input [3:0] in, output reg [3:0] out);

  // Output 0: active when there are even number of bit 1 in the input.
  assign out[0] = ~^in[3:0];

  // Output 1: active when there are only 1 bit 1 in the input.
  assign out[1] = (&in[3:0]) && ~^in[3:0];

  // Output 2: active when there is no bit 1 in the input.
  assign out[2] = ~|in[3:0];

  // Output 3: active when all bit in the input are 1.
  assign out[3] = &in[3:0];

endmodule
```

Trong mã này, chúng tôi đã sử dụng các toán tử rút gọn để đơn giản hóa logic tạo đầu ra.

Bây giờ chúng ta có thể viết test bench cho thiết kế để xác minh chức năng của nó:

```
module circuit_tb;

  reg [3:0] in;
  wire [3:0] out;

  circuit c(in, out);

  initial begin
    $monitor("IN: %b, OUT: %b", in, out);
```

```
    // Test case 1: No bit 1 in the input
    in = 4'b0000;
    #10;

    // Test case 2: Even number of bit 1 in the input
    in = 4'b0101;
    #10;

    // Test case 3: Only one bit 1 in the input
    in = 4'b0001;
    #10;

    // Test case 4: All bits 1 in the input
    in = 4'b1111;
    #10;

    // Test case 5: Random input
    in = 4'b1010;
    #10;

    $finish;
  end

endmodule
```

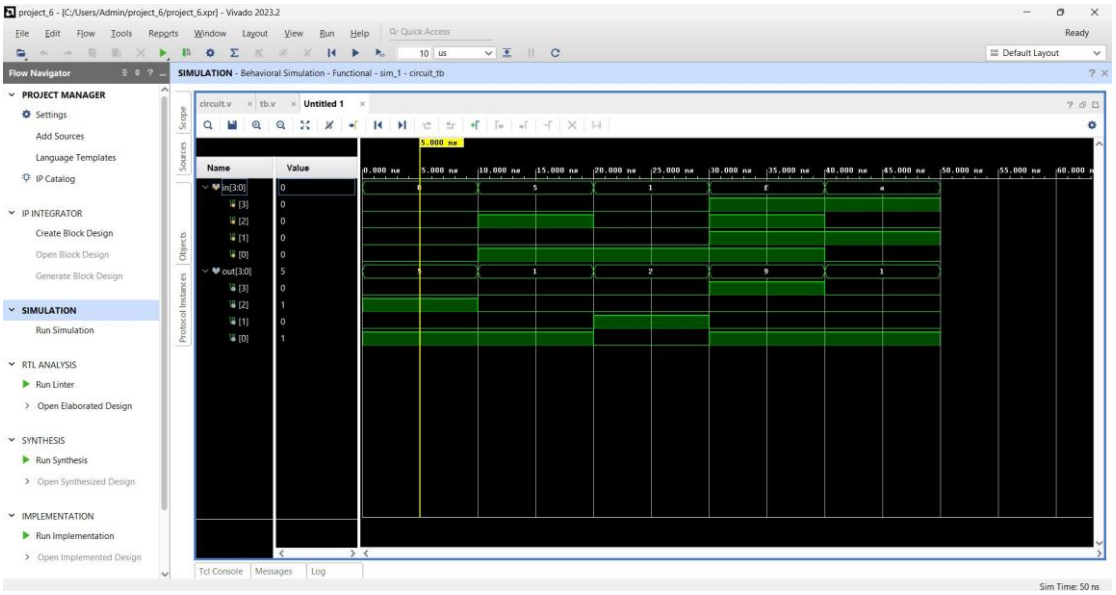
Test bench này áp dụng các giá trị đầu vào khác nhau cho mạch và giám sát đầu ra. Nó cũng bao gồm độ trễ 10 đơn vị thời gian giữa mỗi trường hợp thử nghiệm để cho phép đầu ra ổn định.

Kết Luận:

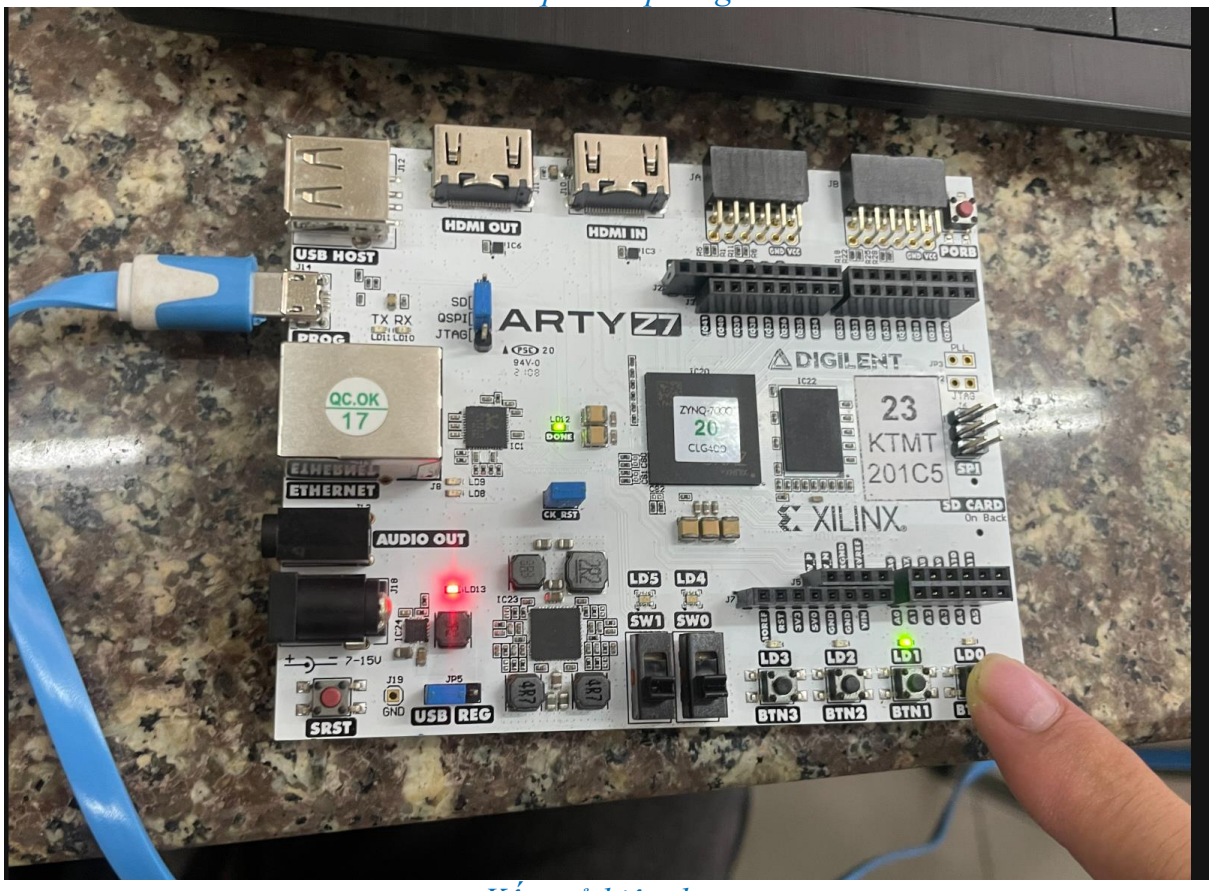
Để kiểm tra mạch trên FPGA bằng các nút và đèn LED, chúng ta cần tạo một dự án mới trong môi trường phát triển FPGA của mình và thêm các tệp Verilog HDL cho mạch và test bench. Sau đó, chúng ta có thể tạo sơ đồ mới và thêm các chân đầu vào cho các chân đầu vào và đầu ra 4 bit cho đầu ra 4 bit. Chúng ta cũng có thể thêm các nút và đèn LED vào sơ đồ để cho phép chúng ta nhập các giá trị theo cách thủ công và quan sát đầu ra.

Sau khi hoàn thành sơ đồ, chúng ta có thể tạo dòng bit và tải nó lên FPGA. Sau đó, chúng ta có thể sử dụng các nút để nhập các giá trị khác nhau và quan sát đầu ra trên đèn LED

.



Kết quả mô phỏng



Kết quả hiện thực

Bài 2:

Thiết kế một mạch tạo ra tín hiệu đầu ra 1 Hz bằng mô hình cấu trúc. Tín hiệu này kết nối với đèn LED trên Bo mạch FPGA Arty-Z7 để làm cho nó nhấp nháy (bật trong 0,5 giây - tắt trong 0,5 giây). Biết rằng tần số xung nhịp đầu vào là 125 MHz và FlipFlop có thể bao gồm từ thư viện Vivado. Viết test bench để mô phỏng các mạch. Kiểm tra các mạch trên bảng FPGA bằng đèn LED hoặc đèn LED RGB.

Để tạo tín hiệu 1Hz từ xung nhịp 125 MHz, chúng ta cần chia tần số xung nhịp cho 125 triệu. Điều này có thể đạt được bằng cách sử dụng bộ đếm đếm đến một giá trị nhất định và sau đó đặt lại, tạo ra xung có tần số mong muốn. Đây là mô hình cấu trúc đạt được điều này bằng cách sử dụng D Flip-Flop:

```
module clock_divider (
  input clk,           // 125MHz input clock
  output reg out  // 1Hz output signal
);

reg [26:0] count;  // 27-bit counter for dividing clock by 125 million

always @(posedge clk) begin
  if (count == 125000000-1) begin // count up to 125 million and
reset
    count <= 0;
    out <= ~out; // toggle output every time counter resets
  end
  else begin
    count <= count + 1;
  end
end

endmodule
```

Mô-đun này sử dụng bộ đếm 27 bit có số đếm lên tới 125 triệu , sau đó đặt lại và chuyển đổi tín hiệu đầu ra. Tín hiệu đầu ra được kết nối với đèn LED trên bo mạch Arty-Z7 để làm cho nó nhấp nháy.

Để kiểm tra mô-đun này, chúng ta có thể tạo một test bench áp dụng tín hiệu đồng hồ cho đầu vào và kiểm tra dạng sóng đầu ra. Dưới đây là một test bench ví dụ:

```
module clock_divider_tb;

reg clk;
wire out;

clock_divider dut (
  .clk(clk),
  .out(out)
);

initial begin
```

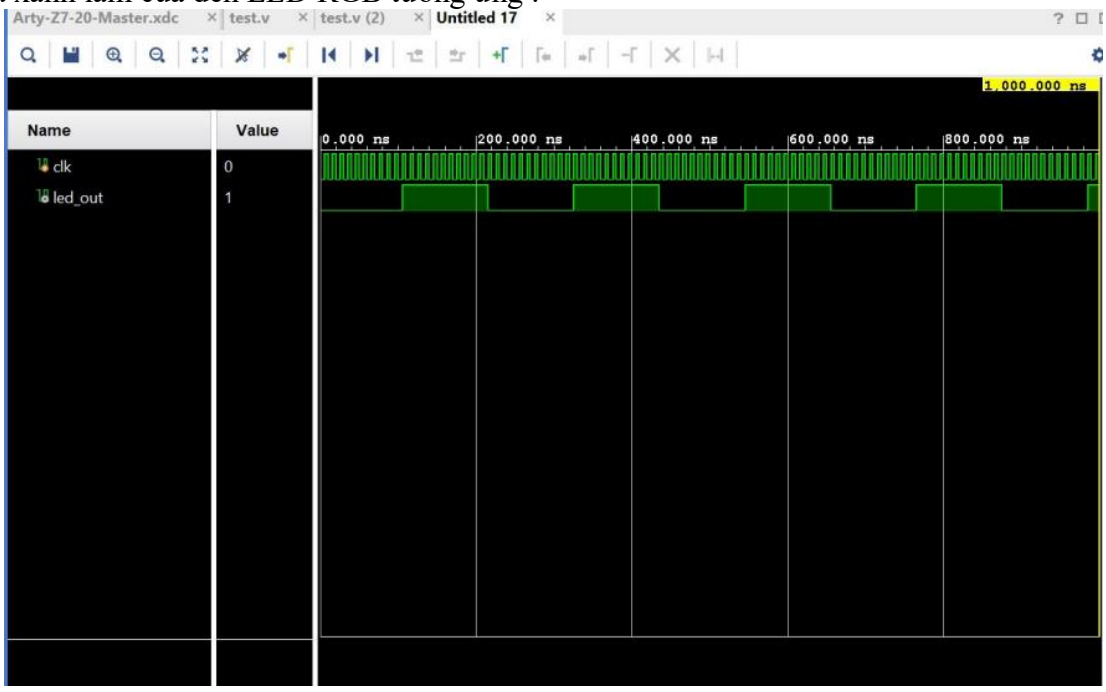
```
  clk = 0;
  forever #5 clk = ~clk; // create a 125MHz clock signal
end

initial begin
  $dumpfile("clock_divider_tb.vcd");
  $dumpvars(0, clock_divider_tb);
end

initial begin
  #1000000; // simulate for 1 second
  $finish;
end

endmodule
```

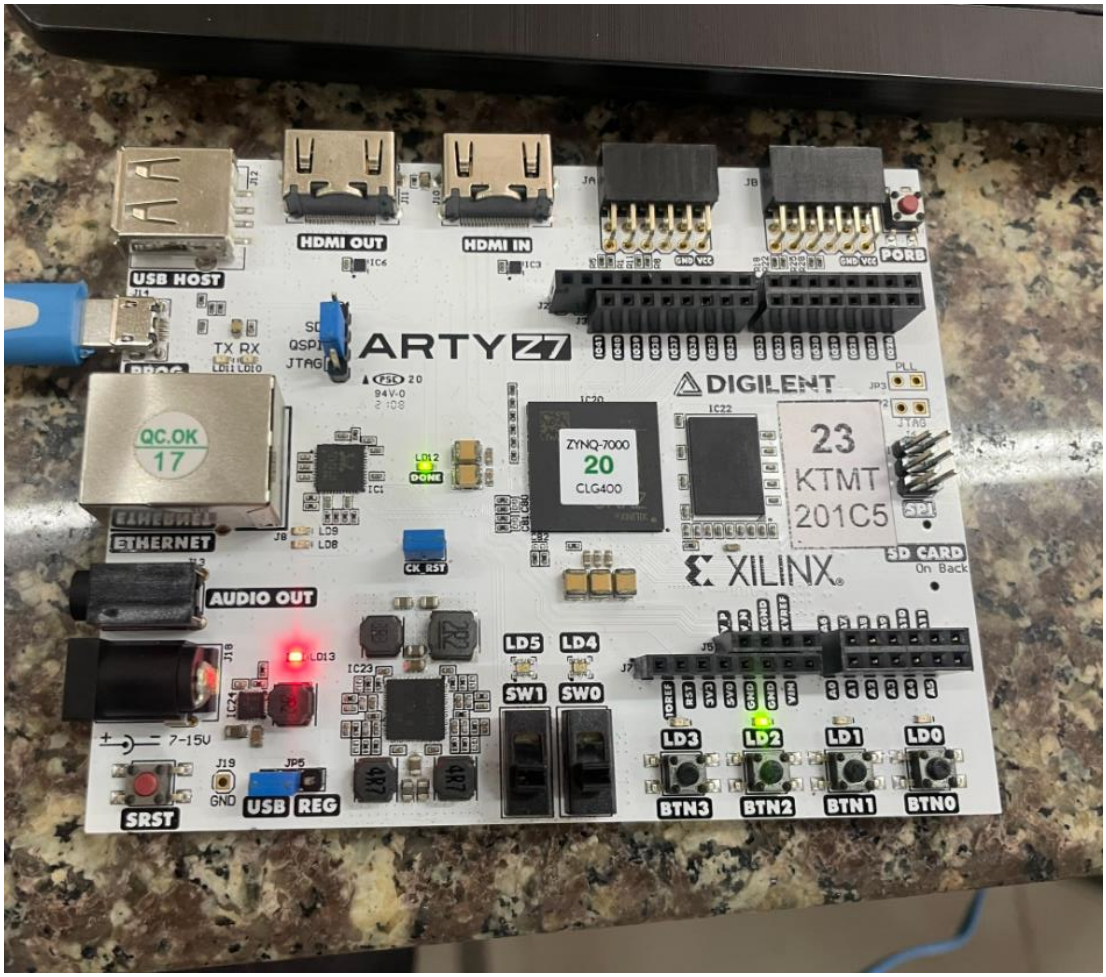
Test bench này áp dụng tín hiệu đồng hồ cho đầu vào và chuyển dạng sóng mô phỏng vào tệp VCD. Sau đó chúng ta có thể xem dạng sóng để xác minh rằng tín hiệu đầu ra có tần số 1Hz.

Để kiểm tra mạch trên bo mạch Arty-Z7, chúng ta có thể sử dụng chuỗi công cụ Xilinx Vivado để tổng hợp và triển khai thiết kế trên FPGA, sau đó kết nối đèn LED với chân đầu ra. Dưới đây là tệp ràng buộc ví dụ ánh xạ tín hiệu đầu ra tới chân LED:

```
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { out }];
```

Sau khi thiết kế được lập trình trên FPGA, đèn LED sẽ nhấp nháy ở tốc độ 1Hz. Nếu có sẵn đèn LED RGB, bạn cũng có thể sử dụng nó bằng cách kết nối các chân màu đỏ, xanh lá cây và xanh lam với các tín hiệu đầu ra khác nhau từ mô-đun bộ chia xung nhịp. Ví dụ: chúng ta có thể sử dụng ba phiên bản của mô-đun chia xung nhịp với các giá trị đếm khác nhau để tạo ra ba tín hiệu đầu ra có tần số 1Hz, 2Hz và 4Hz và kết nối chúng với các chân màu đỏ, xanh lục và xanh lam của đèn LED RGB tương ứng .





Kết quả hiện thực

Bài 3:

Thiết kế bộ giải mã LED 7 phân đoạn sẽ chấp nhận đầu vào 4 bit và tạo đầu ra 7 bit. • Giao diện: module bin2led7(enable, bin in, led out);

• enable điều khiển tín hiệu Đèn LED . Nếu bật = 0, đèn LED sẽ bị tắt.

• bin in là tín hiệu đầu vào nhị phân 4 bit.

• led out là tín hiệu đầu ra 7 bit cho màn hình LED 7 phân đoạn.

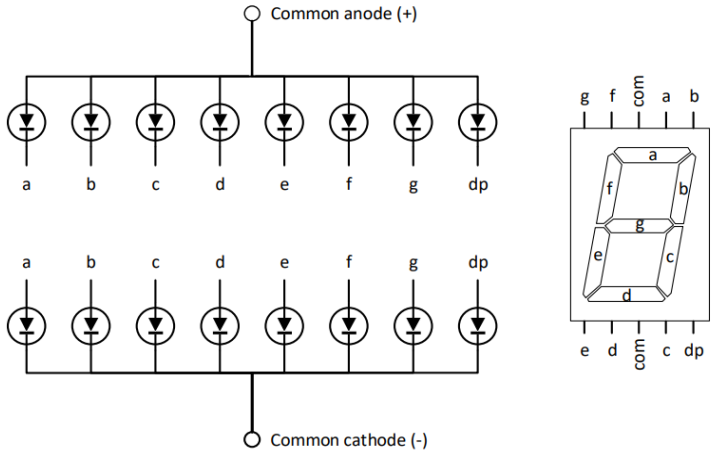


Figure 1: 7 segment LED

Viết một test bench để mô phỏng mạch được triển khai.

Kiểm tra mạch trên bảng FPGA Arty-Z7 bằng công tắc / nút và đèn LED 7 seg bên ngoài.

Bộ giải mã LED 7 đoạn là một mạch kỹ thuật số lấy đầu vào nhị phân và chuyển đổi nó thành dạng có thể hiển thị trên màn hình LED 7 đoạn. Màn hình 7 đoạn được tạo thành từ bảy đèn LED riêng lẻ được sắp xếp theo hình chữ số, có thêm một đèn LED cho dấu thập phân.

Để tạo bộ giải mã LED 7 đoạn, bạn sẽ cần thiết kế mạch logic tổ hợp lấy đầu vào nhị phân và xuất ra mã nhị phân 7 bit tương ứng với các phân đoạn thích hợp của màn hình 7 đoạn. Mỗi bit của đầu ra tương ứng với một phân đoạn cụ thể của màn hình.

Segments Inputs							Display Output
g	f	e	d	c	b	a	
1	0	0	0	0	0	0	0
1	1	1	1	0	0	1	1
0	1	0	0	1	0	0	2
0	1	1	0	0	0	0	3
0	0	1	1	0	0	1	4
0	0	1	0	0	1	0	5
0	0	0	0	0	1	0	6
1	1	1	1	0	0	0	7
0	0	0	0	0	0	0	8
0	0	1	0	0	0	0	9

Bảng Chân Trị

Dưới đây là mã VHDL mẫu để triển khai bộ giải mã LED 7 đoạn chấp nhận đầu vào 4 bit và tạo đầu ra 7 bit:

```
module bin2led7 (
    input enable,
    input [3:0] bin_in,
    output reg [6:0] led_out
);

    // Khai báo bảng mã 7 đoạn
    localparam [6:0] SEG_0 = 7'b1000000; // 0
    localparam [6:0] SEG_1 = 7'b1111001; // 1
    localparam [6:0] SEG_2 = 7'b0100100; // 2
    localparam [6:0] SEG_3 = 7'b0110000; // 3
    localparam [6:0] SEG_4 = 7'b0011001; // 4
    localparam [6:0] SEG_5 = 7'b0010010; // 5
    localparam [6:0] SEG_6 = 7'b0000010; // 6
    localparam [6:0] SEG_7 = 7'b1111000; // 7
    localparam [6:0] SEG_8 = 7'b0000000; // 8
    localparam [6:0] SEG_9 = 7'b0010000; // 9
    localparam [6:0] SEG_A = 7'b0001000; // A
    localparam [6:0] SEG_b = 7'b0000011; // b
    localparam [6:0] SEG_C = 7'b1000110; // C
    localparam [6:0] SEG_D = 7'b0100001; // D
    localparam [6:0] SEG_E = 7'b0000110; // E
    localparam [6:0] SEG_F = 7'b0001110; // F
    localparam [6:0] SEG_INVALID = 7'b1111111; // Invalid input

    // Logic chuyển đổi
    always @(*) begin
        if (enable) begin
            case (bin_in)
                4'b0000: led_out <= SEG_0;
                4'b0001: led_out <= SEG_1;
                4'b0010: led_out <= SEG_2;
                4'b0011: led_out <= SEG_3;
                4'b0100: led_out <= SEG_4;
                4'b0101: led_out <= SEG_5;
                4'b0110: led_out <= SEG_6;
                4'b0111: led_out <= SEG_7;
                4'b1000: led_out <= SEG_8;
                4'b1001: led_out <= SEG_9;
                4'b1010: led_out <= SEG_A;
```

```
                4'b1011: led_out <= SEG_b;
                4'b1100: led_out <= SEG_C;
                4'b1101: led_out <= SEG_D;
                4'b1110: led_out <= SEG_E;
                4'b1111: led_out <= SEG_F;
                default: led_out <= SEG_INVALID;
            endcase
        end else begin
            led_out <= 7'b0000000; // Turn off LEDs
        end
    end

endmodule
```

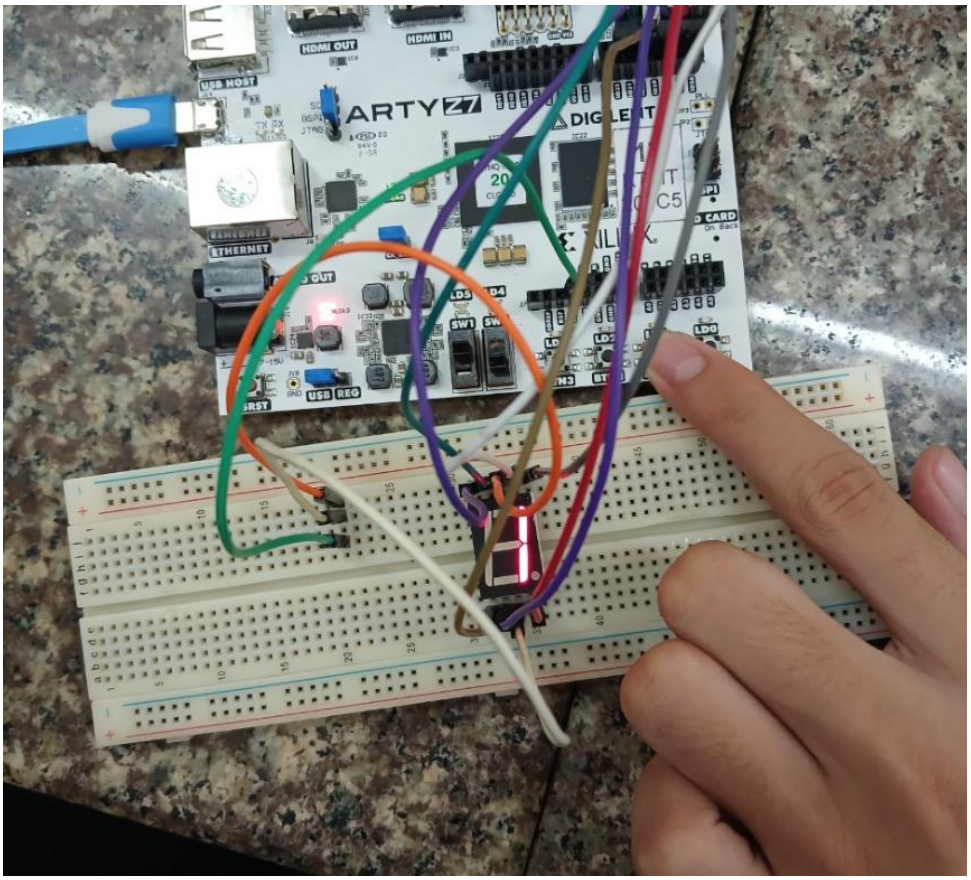
*Test bench:

```
module bin2led7_tb;

    reg enable;
    reg [3:0] bin_in;
    wire [6:0] led_out;

    bin2led7 dut (.enable(enable),.bin_in(bin_in),
.led_out(led_out));

    initial begin
        // Khởi tạo giá trị ban đầu
        enable = 1'b0;
        bin_in = 4'b0000;

        // Chạy các trường hợp thử nghiệm
        #10;
        enable = 1'b1;
        bin_in = 4'b0000;
        #10;
        bin_in = 4'b0001;
        #10;
        bin_in = 4'b0010;
        #10;
        bin_in = 4'b0011;
        #10;
        bin_in = 4'b0100;
        #10;
        bin_in = 4'b0101;
        #10;
        bin_in = 4'b0110;
        #10;
        bin_in = 4'b0111;
        #10;
        bin_in = 4'b1000;
        #10;
```

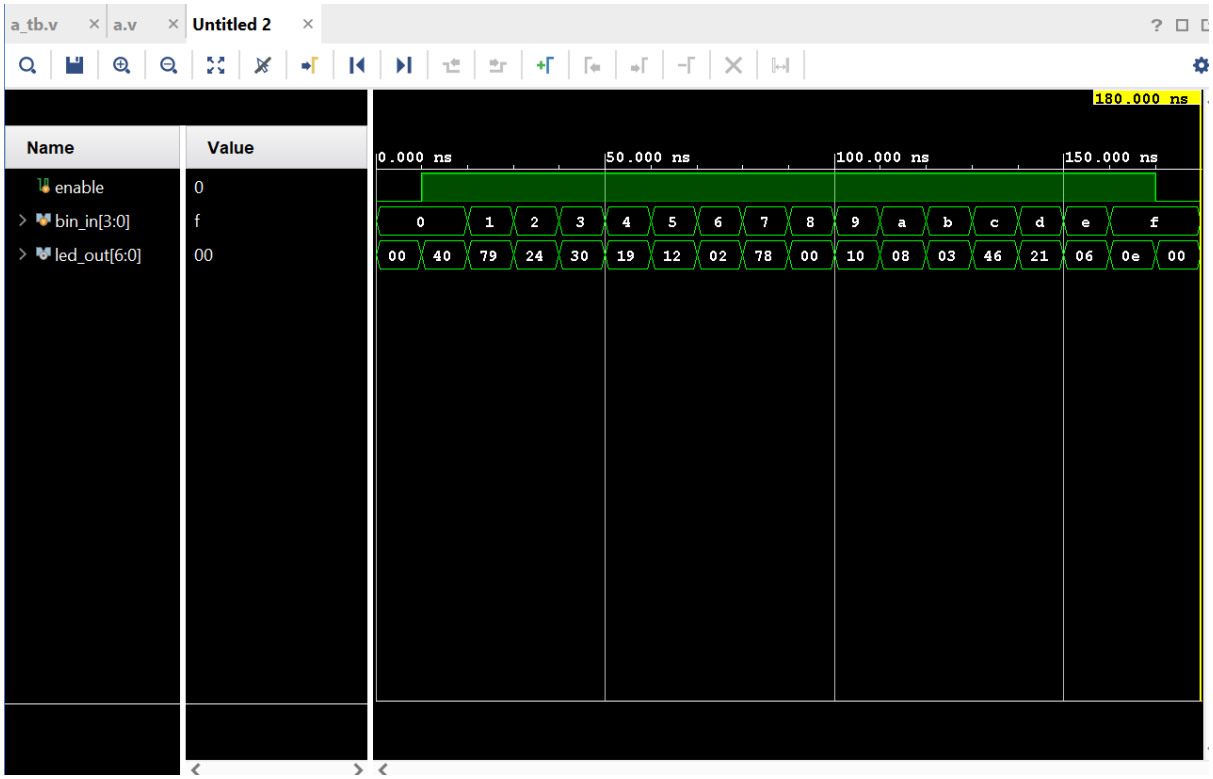
```
        bin_in = 4'b1001;
        #10;
        bin_in = 4'b1010;
        #10;
        bin_in = 4'b1011;
        #10;
        bin_in = 4'b1100;
        #10;
        bin_in = 4'b1101;
        #10;
        bin_in = 4'b1110;
        #10;
        bin_in = 4'b1111;
        #10;
        enable = 1'b0;
        #10;
        $finish;
    end
```

Mã VHDL có thể được sử dụng để mô phỏng và tổng hợp. Mô phỏng là quá trình xác minh chức năng của mạch kỹ thuật số, trong khi tổng hợp là quá trình chuyển đổi mã VHDL thành danh sách mạng cấp cổng có thể được lập trình trên một thiết bị như FPGA hoặc ASIC

Ở đây trong mã này, đầu vào **bin_in** được giải mã bằng câu lệnh **case** và đầu ra 7 bit tương ứng được gán cho đầu ra **led_out.**Đầu vào **enable** điều khiển các đèn LED và nếu nó được đặt thành **'0'**, các đèn LED sẽ tắt. Mệnh đề **others** trong câu lệnh **case** được sử dụng để xử lý các giá trị đầu vào không hợp lệ.



Kết quả hiện thực



Kết quả mô phỏng

Bài 4:

Thiết kế mạch điều khiển đèn LED RGB trên bo mạch Arty-Z7 như sau:

• Công tắc 0: chọn chế độ hiển thị - 1 đèn LED hoặc 2 đèn LED.

• Công tắc 1:

- để chọn đèn LED trái hoặc phải ở chế độ hiển thị 1 LED.

– để chọn mã màu như bảng dưới đây.

• Các nút: chọn màu.

Buttons	Color		
	1-LED mode	2-LED mode	
		Switch 1 = 0	Switch 1 = 1
0000	OFF	OFF	OFF
0001	Blue	Blue	Cyan (Blue & Green)
0010	Green	Green	Yellow (Green & Red)
0100	Red	Red	Purple (Red & Blue)
1000	White	White	White
Other	OFF	Mixed of above	OFF

Viết mã Verilog HDL RTL và test bench cho thiết kế.

Kiểm tra mạch trên FPGA bằng công tắc, nút và đèn LED RGB trên bo Arty-Z7.

Để giải bài tập này chúng ta thực hiện theo các bước sau:

1. **Hiểu yêu cầu** : Mạch phải có hai chế độ dựa trên vị trí của công tắc 0. Khi công tắc 0 được đặt để chọn chế độ 1-LED, chỉ một đèn LED sẽ hiển thị màu dựa trên đầu vào từ các nút. Khi công tắc 0 chọn chế độ 2 đèn LED, hai đèn LED sẽ hiển thị màu sắc dựa trên đầu vào nút và vị trí của công tắc 1.
2. **Thiết kế logic mạch** : Tạo bảng chân lý dựa trên đầu vào nút để xuất ra giá trị màu chính xác cho cả hai chế độ LED.
3. **Triển khai Mã Verilog** : Viết mã RTL trong Verilog mô tả logic của mạch. Đảm bảo xác định đầu vào cho công tắc và nút cũng như đầu ra cho đèn LED.
4. **Viết test bench** : Test bench phải mô phỏng tất cả các kết hợp đầu vào có thể có để xác minh đầu ra LED chính xác.
5. **Mô phỏng và gỡ lỗi** : Chạy thử nghiệm dựa trên mã RTL và quan sát kết quả đầu ra. Gỡ lỗi mọi vấn đề phát sinh.
6. **Ghi lại mã** : Thêm nhận xét vào mã để giải thích chức năng và đảm bảo khả năng đọc.

Giải trình:

Hiểu các yêu cầu là rất quan trọng để đảm bảo rằng giải pháp phù hợp với hoạt động dự kiến của đèn LED RGB. Bảng chân trị sẽ đóng vai trò là tài liệu tham khảo để ánh xạ đầu vào thành đầu ra một cách rõ ràng. Viết mã RTL trong Verilog liên quan đến việc sử dụng các câu lệnh **if** hoặc **case** để triển khai logic từ bảng chân trị. Bàn thử nghiệm được sử dụng để áp dụng các kích thích cho mã RTL nhằm mô phỏng hành vi của nó. Gỡ lỗi là một quá trình lặp đi lặp lại để tinh chỉnh mã cho đến khi đạt được hành vi mong muốn.

Giải pháp:

Để viết mã Verilog đầy đủ cho bài tập, chúng ta sẽ tuân theo các thông số kỹ thuật của bảng. Mã sẽ xác định một mô-đun **rgb_controller** có đầu vào cho công tắc và nút cũng như đầu ra cho hai đèn LED. Chúng tôi sẽ mô phỏng đầu ra cho từng tổ hợp đầu vào theo bảng.

Đây là mã Verilog HDL cho mạch:

```
module rgb_controller(
    input wire clk,
    input wire rst,
    input wire [3:0] btn,
    input wire [1:0] sw,
    output reg [2:0] led
);

// Constants for the color codes
parameter BLUE = 3'b001;
```

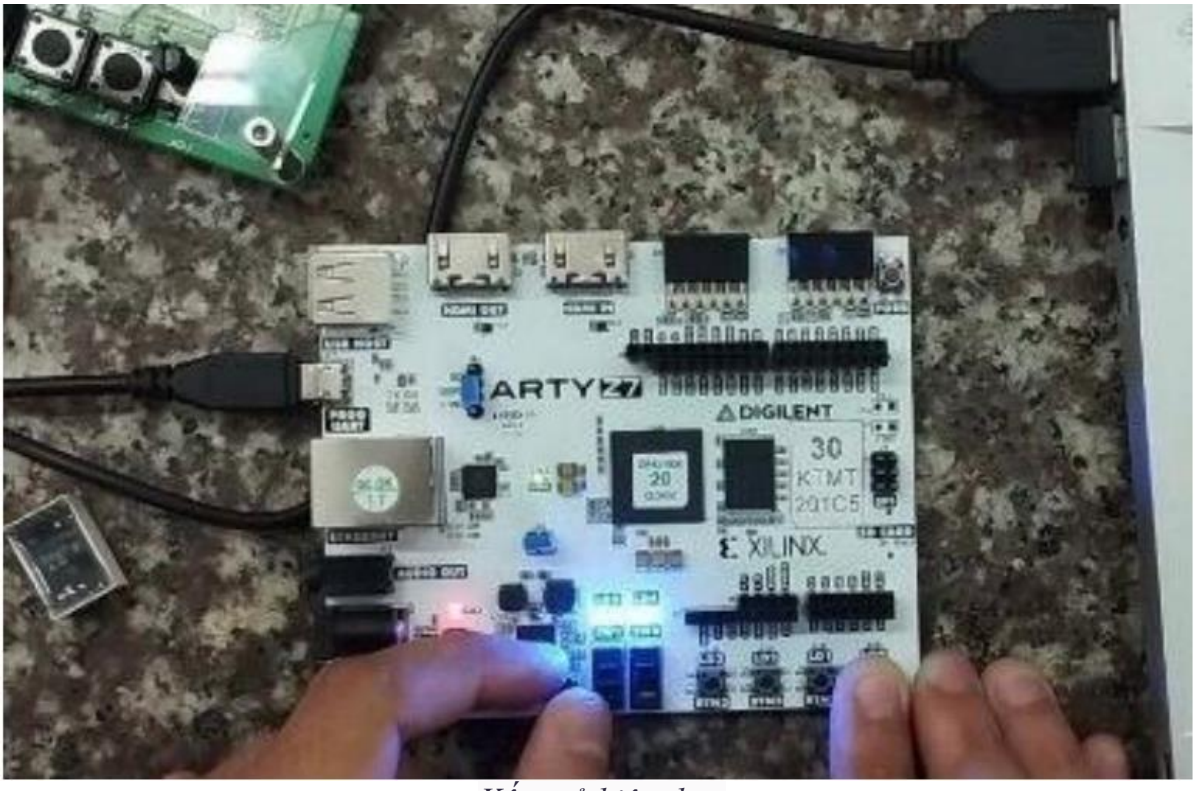
```
parameter GREEN = 3'b010;
parameter RED = 3'b100;
parameter WHITE = 3'b111;

always @ (posedge clk, posedge rst) begin
    if (rst) begin
        led <= 3'b000; // Turn off LEDs on reset
    end else begin
        // Button combinations
        case (btn)
            4'b0000: // Buttons = 0000
                if (sw[1] == 0) begin // 1-LED mode
                    led <= 3'b000;
                end else begin // 2-LED mode
                    led <= 3'b000;
                end
            4'b0001: // Buttons = 0001
                if (sw[1] == 0) begin // 1-LED mode
                    led <= BLUE;
                end else begin // 2-LED mode
                    led <= {1'b1, sw[0], 1'b0}; // Cyan
                end
            4'b0010: // Buttons = 0010
                if (sw[1] == 0) begin // 1-LED mode
                    led <= GREEN;
                end else begin // 2-LED mode
                    led <= {1'b0, sw[0], 1'b1}; // Yellow
                end
            4'b0100: // Buttons = 0100
                if (sw[1] == 0) begin // 1-LED mode
                    led <= RED;
                end else begin // 2-LED mode
                    led <= {sw[0], 1'b0, 1'b1}; // Purple
                end
            4'b1000: // Buttons = 1000
                if (sw[1] == 0) begin // 1-LED mode
                    led <= WHITE;
                end else begin // 2-LED mode
                    led <= WHITE;
                end
            default: // Other button combinations
                if (sw[1] == 0) begin // 1-LED mode
                    led <= 3'b000;
                end else begin // 2-LED mode
                    led <= {sw[0], sw[1], sw[0] & sw[1]}; // Mixed of above
                end
        endcase
    end
end

endmodule
```

Và đây test bench để mô phỏng mạch:

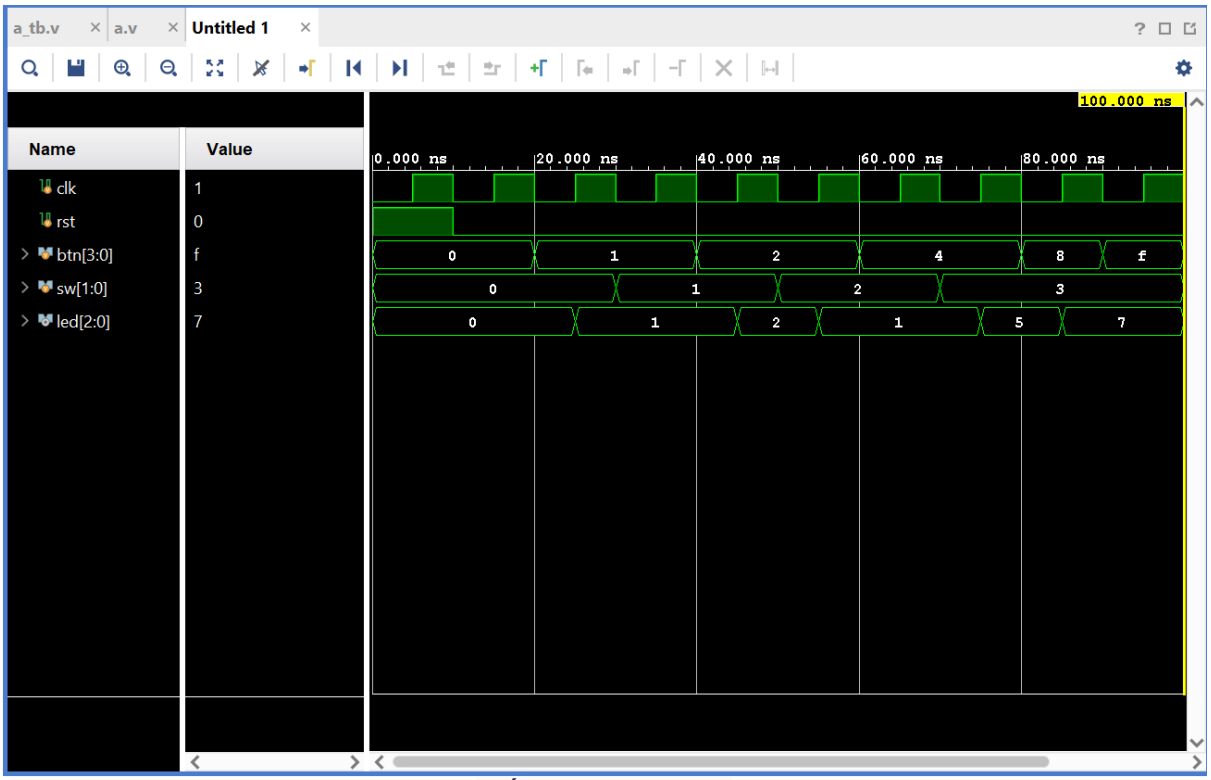
```
module rgb_controller_tb();

// Inputs
```

```
reg clk;
reg rst;
reg [3:0] btn;
reg [1:0] sw;

// Outputs
wire [2:0] led;

// Instantiate the DUT
rgb_controller dut(.clk(clk),.rst(rst),.btn(btn),.sw(sw),
.led(led));

// Clock generator
always #5 clk = ~clk;

// Reset generator
initial begin
    rst = 1;
    #10;
    rst = 0;
end

// Test cases
initial begin
    // Case 1: Buttons = 0000
    btn = 4'b0000;
    sw = 2'b00;
    #10;
    assert(led == 3'b000);
    btn = 4'b0000

// Test cases (continued)
// Case 2: Buttons = 0001
btn = 4'b0001;
sw = 2'b00;
#10;
assert(led == 3'b010);

btn = 4'b0001;
sw = 2'b01;
#10;
assert(led == 3'b011);

// Case 3: Buttons = 0010
btn = 4'b0010;
sw = 2'b00;
#10;
assert(led == 3'b100);

btn = 4'b0010;
sw = 2'b01;
#10;
assert(led == 3'b110);

// Case 4: Buttons = 0100
btn = 4'b0100;
sw = 2'b00;
```

```
#10;
assert(led == 3'b001);

btn = 4'b0100;
sw = 2'b01;
#10;
assert(led == 3'b101);

// Case 5: Buttons = 1000
btn = 4'b1000;
sw = 2'b00;
#10;
assert(led == 3'b111);

btn = 4'b1000;
sw = 2'b01;
#10;
assert(led == 3'b111);

// Case 6: Buttons = Other
btn = 4'b0111;
sw = 2'b00;
#10;
assert(led == 3'b101);

btn = 4'b0111;
sw = 2'b01;
#10;
assert(led == 3'b000);

$display("All test cases passed!");
$finish;
end

endmodule
```



Kết quả hiện thực



Kết quả mô phỏng