

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



CC01 Digital design with HDL

Assignment Report

Crossroad traffic light

Lecturer: Nguyen Thanh Loc
Class: CC01

Tran Minh Tuong	2353304
Nguyen Hieu Trung	2353244
Ho Ngoc Quynh Anh	2352033
Phuong Xuong Duc	2352270
Pham Minh Nhat	2352863

Contents

1	Introduction	2
2	Design	3
2.1	Present the idea:	3
2.2	Block diagram:	3
3	Implementation:	4
3.1	TopModudel.v:	4
3.2	state of light.v:	5
3.3	Traffic light 1.v:	6
3.4	Traffic light 2.v:	8
3.5	seven segment display:	9
4	Result:	12
4.1	Test bench:	12
4.2	Explanation:	14
5	Conclusion	14
5.1	Working achievement:	14
5.2	Advantage of this system:	14
6	References	15

1 Introduction

Traffic lights are a fundamental component of urban traffic management systems, ensuring the safe and efficient movement of vehicles and pedestrians through intersections. By controlling the flow of traffic, traffic lights prevent accidents and reduce congestion. This project aims to implement a functional traffic light control system using an FPGA board, specifically the Arty-7 FPGA board. The system will be designed to manage traffic at a four-way intersection..

The primary goals of this project are to enhance skills in using Verilog HDL for digital circuit design and to improve ability to analyze and design systems using a structured, hierarchical approach. By working on this traffic light controller, we will gain practical experience with Verilog HDL, a vital tool in digital circuit design. Additionally, understanding the operation and construction of traffic light controllers will broaden our knowledge of commonly used circuits in digital systems. This project will also help us develop research, self-study, documentation, and presentation skills.

Verilog HDL is a hardware description language used to model electronic systems. It is particularly effective for designing complex digital systems, allowing designers to describe the structure and behavior of the system at various levels of abstraction. By practicing with Verilog HDL, we enhance our coding skills and learn to think in terms of hardware implementation, crucial for creating efficient digital circuits.

Our project will follow a hierarchical design approach, breaking down the traffic light controller into smaller, manageable modules, each responsible for a specific function. This method simplifies the design process, makes the system easier to understand and debug, allows for modular testing and verification. By adopting this approach, we practice analyzing the system at different levels and integrating the modules to form a cohesive whole.

The report covers the complete lifecycle of designing a crossroad traffic light controller using Verilog HDL. This includes the initial conceptualization, hierarchical design, coding, simulation, and testing phases. Emphasis is placed on modular design and the use of state machines to manage traffic light sequences efficiently. We provide an overview of the problem and the significance of effective traffic light control systems, detail the hierarchical approach taken to design the system, and include the Verilog code with thorough explanations for each module. The simulation results are presented and analyzed to verify the design's correctness and efficiency, concluding with a summary of the findings, potential improvements, and future work.

By the end of this project, we aim to have a solid grasp of digital design with Verilog HDL, deeper insights into traffic light control systems, and enhanced research, documentation, and presentation skills. This hands-on experience will prepare us to tackle more complex digital design challenges in the future and contribute effectively to the field of digital electronics.

2 Design

2.1 Present the idea:

- Using Verilog code to create a light control circuit by utilizing multiple modules with different functions.
- Develop an advanced and versatile traffic light with more functionalities.
- Capable of handling emergency situations.
- Enhance traffic safety for participants.

2.2 Block diagram:

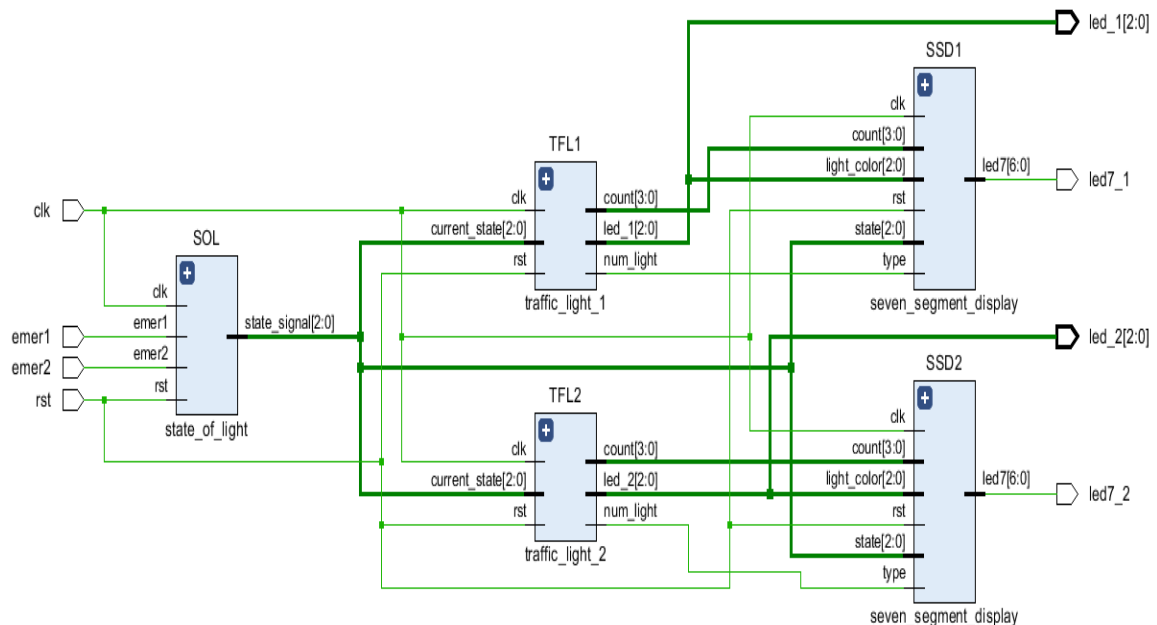


Figure 2.2: block diagram of the crossed traffic light.

3 Implementation:

3.1 TopModudel.v:

```
module TopModule(  
    input clk,  
    input rst,  
    input emer1,  
    input emer2,  
    output [2:0] RGB_LED_1,  
    output [2:0] RGB_LED_2,  
    output [6:0] num_1,  
    output [6:0] num_2  
);  
  
//connect state signal  
    wire [1:0] state_out;  
  
    state_of_light SOL (.clk(clk), .rst(rst), .emer1(emer1),  
        .emer2(emer2), .state_signal(state_out));  
  
//count and type signal 1  
    wire [4:0] count_1;  
    wire type_1;  
    wire [6:0] LED_1;  
  
    traffic_light_1 TFL1(.clk(clk), .rst(rst), .current_state(state_out),  
        .count(count_1), .led_1(LED_1), .num_light(type_1));  
  
//count and type signal 2  
  
    wire [4:0] count_2;  
    wire type_2;  
    wire [6:0] LED_2;  
  
    traffic_light_2 TFL2(.clk(clk), .rst(rst), .current_state(state_out),  
        .count(count_2), .led_2(LED_2), .num_light(type_2));  
  
//seven-segment for light 1  
    seven_segment_display SSD1 (.clk(clk), .rst(rst), .type(type_1),  
        .light_color(LED_1), .count(count_1), .state(state_out),  
        .led7(num_1));  
  
//seven-segment for light 2  
    seven_segment_display SSD2 (.clk(clk), .rst(rst), .type(type_2),  
        .light_color(LED_2), .count(count_2), .state(state_out),  
        .led7(num_2));  
  
    assign RGB_LED_1 = LED_1;
```

```
assign RGB_LED_2 = LED_2;
```

```
endmodule
```

3.2 state of light.v:

Used to determine the state of a traffic light.

```
'timescale 1ns / 1ps

// green = 3'b010, yellow = 3'b110, red = 3'b100
module state_of_light(
    input wire clk,
    input wire rst,
    input wire emer1,
    input wire emer2,
    output reg [1:0] state_signal
);

    wire [1:0] current_state;
    assign current_state = {emer1, emer2};
    parameter normal = 2'b00,
        emergency_1 = 2'b01,
        emergency_2 = 2'b10;
    reg [1:0] state, next_state;

    always @(posedge clk or posedge rst) begin
        if(rst == 1'b1) begin
            state <= normal;
        end else begin
            state <= next_state;
        end
    end

    //Determination the next_state respect to the input
    always @(state or emer1 or emer2) begin
        case (state)
            normal: begin
                if(current_state == 2'b10 || current_state == 2'b11) begin
                    next_state = emergency_1;
                end else if (current_state == 2'b01 || current_state ==
                    2'b11) begin
                    next_state = emergency_2;
                end else begin
                    next_state = normal;
                end
            end
            emergency_1: begin
                if(current_state == 2'b10 || current_state == 2'b11) begin
```

```
        next_state = emergency_1;
    end else if (current_state == 2'b01 || current_state ==
        2'b11) begin
        next_state = emergency_2;
    end else begin
        next_state = normal;
    end
end
emergency_2: begin
    if(current_state == 2'b10 || current_state == 2'b11) begin
        next_state = emergency_1;
    end else if (current_state == 2'b01 || current_state ==
        2'b11) begin
        next_state = emergency_2;
    end else begin
        next_state = normal;
    end
end
default: begin
    next_state = normal;
end
endcase
end
always @(state) begin
    case (state)
        normal: state_signal = 2'b00;
        emergency_1: state_signal = 2'b10;
        emergency_2: state_signal = 2'b01;
        default: state_signal = 2'b00;
    endcase
end
endmodule
```

3.3 Traffic light 1.v:

Used to control colors of the traffic 1 light based on states.

```
'timescale 1ns / 1ps

// green = 3'b010, yellow = 3'b110, red = 3'b100

module traffic_light_1(
    input wire clk,
    input wire rst,
    input wire [1:0] current_state,
    output reg [4:0] count,
    output reg [2:0] led_1,
    output wire num_light
);
```

```
parameter red = 3'b100,
          green = 3'b010,
          yellow = 3'b110,
          off = 3'b000,
          emer1 = 2'b10,
          emer2 = 2'b01,
          normal = 2'b00;

//because count is 4 bits
reg [4:0] red_duty = 6;
reg [4:0] green_duty = 3;
reg [4:0] yellow_duty = 3;
wire [4:0] total_duty;
assign num_light = 1'b0;
assign total_duty = red_duty + green_duty + yellow_duty;

//counter
always @(posedge clk or posedge rst) begin
    if(rst) begin
        count <= 0;
    end else if(count >= total_duty - 1) begin
        count <= 0;
    end else begin
        count <= count + 1;
    end
end

//logic of light controller
always @(posedge clk or posedge rst) begin
    if(rst) begin
        led_1 <= off;
    end else begin
        case (current_state) // green -> yellow -> red
            normal: begin
                if(count < green_duty) begin
                    led_1 <= green;
                end else if(count < (green_duty + yellow_duty)) begin
                    led_1 <= yellow;
                end else begin
                    led_1 <= red;
                end
            end
            emer1: begin
                led_1 <= green;
            end
            emer2: begin
                led_1 <= red;
            end
            default: begin
```

```
        led_1 <= off;
    end
endcase
end
end
endmodule
```

3.4 Traffic light 2.v:

Used to control colors of traffic light 2 based on states.

```
'timescale 1ns / 1ps

module traffic_light_2(
    input wire clk,
    input wire rst,
    input wire [1:0] current_state,
    output reg [4:0] count,
    output reg [2:0] led_2,
    output wire num_light
);

    parameter red = 3'b100,
               green = 3'b010,
               yellow = 3'b110,
               off = 3'b000,
               emer1 = 2'b10,
               emer2 = 2'b01,
               normal = 2'b00;

    //because count is 4 bits
    reg [4:0] red_duty = 6;
    reg [4:0] green_duty = 3;
    reg [4:0] yellow_duty = 3;
    wire [4:0] total_duty;
    assign num_light = 1'b1;
    assign total_duty = red_duty + green_duty + yellow_duty;

    //counter
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            count <= 0;
        end else if(count >= total_duty - 1) begin
            count <= 0;
        end else begin
            count <= count + 1;
        end
    end
end
```

```
//logic of light controller
always @(posedge clk or posedge rst) begin
    if(rst) begin
        led_2 <= off;
    end else begin
        case (current_state) // red -> green -> yellow
            normal: begin
                if(count < red_duty) begin
                    led_2 <= red;
                end else if(count < (green_duty + red_duty)) begin
                    led_2 <= green;
                end else begin
                    led_2 <= yellow;
                end
            end
            emer1: begin
                led_2 <= red;
            end
            emer2: begin
                led_2 <= green;
            end
            default: begin
                led_2 <= off;
            end
        endcase
    end
end
endmodule
```

3.5 seven segment display:

Used for display the countdown number by 7-segment LED.

```
'timescale 1ns / 1ps

// green = 3'b010, yellow = 3'b110, red = 3'b100

module seven_segment_display (
    input wire clk,
    input wire rst,
    input wire type,
    input wire [2:0] light_color,
    input wire [4:0] count,
    input wire [1:0] state,
    output reg [6:0] led7
);

    reg [3:0] current_count;
```

```

parameter OFF = 7'b1111111,
        ONE = 7'b1111001,
        TWO = 7'b0100100,
        THREE = 7'b0110000,
        FOUR = 7'b0011001,
        FIVE = 7'b0010010,
        SIX = 7'b0000010,
        normal = 3'b000, // emer1 and emer2 are special situations
        red = 3'b100,
        green = 3'b010,
        yellow = 3'b110;

reg [4:0] red_duty = 6;
reg [4:0] green_duty = 3;
reg [4:0] yellow_duty = 3;

// Making the countdown number respect to the count and led color
always @(posedge clk or posedge rst) begin
    if(rst) begin
        current_count <= 0;
    end else if(state == normal) begin
        if(type == 1'b0) begin // green -> yellow -> red (light 1)
            if(count < green_duty) begin
                current_count = green_duty - count; //countdown
            end else if (count < (green_duty + yellow_duty)) begin
                current_count = (green_duty + yellow_duty) - count;
            end else begin
                current_count = (green_duty + yellow_duty + red_duty) -
                    count;
            end
        end
        end else begin // red -> green -> yellow (light 2)
            if(count < red_duty) begin
                current_count = red_duty - count;
            end else if(count < (red_duty + green_duty)) begin
                current_count = (red_duty + green_duty) - count;
            end else begin
                current_count = (red_duty + green_duty + yellow_duty) -
                    count;
            end
        end
        end else begin
            current_count = 0;
        end
    end
end

//setup the output for led 7-segment respect to current_count
always @(posedge clk or posedge rst) begin
    if(rst) begin
        led7 <= OFF;
    end else if(state == normal) begin

```

```
        case(current_count)
            6: begin
                led7 <= SIX;
            end
            5: begin
                led7 <= FIVE;
            end
            4: begin
                led7 <= FOUR;
            end
            3: begin
                led7 <= THREE;
            end
            2: begin
                led7 <= TWO;
            end
            1: begin
                led7 <= ONE;
            end
            default: begin
                led7 <= OFF;
            end
        endcase
    end else begin
        led7 <= OFF;
    end
end
endmodule
```

4 Result:

4.1 Test bench:

```
'timescale 1ns / 1ps

module tb_TopModule;
    reg clk;
    reg rst;
    reg emer1;
    reg emer2;
    wire [2:0] RGB_LED_1;
    wire [2:0] RGB_LED_2;
    wire [6:0] num_1;
    wire [6:0] num_2;

    TopModule uut(.clk(clk), .rst(rst), .emer1(emer1), .emer2(emer2),
        .RGB_LED_1(RGB_LED_1),
        .RGB_LED_2(RGB_LED_2), .num_1(num_1), .num_2(num_2));

    always #5 clk = ~clk;

    initial begin

        $monitor("rst = %b, emer1 = %b, emer2 = %b, RGB_LED_1 = %b, RGB_LED_2
            = %b, num_1 = %h, num_2 = %h ", clk, rst, emer1, emer2, RGB_LED_1,
            RGB_LED_2, num_1, num_2);

    // initialize
        clk = 0;
        rst = 1;
        emer1 = 0;
        emer2 = 0;
        #10 rst = 0;

    // normal state
        #205

    // test emer1
        #50 emer1 = 1;
        #50 emer1 = 0;

    // test emer2
        #50 emer2 = 1;
        #50 emer2 = 0;

    // test both emer
        #50 emer1 = 1; emer2 = 1;
        #50 emer1 = 0; emer2 = 0;
```

```
//normal
    #500;
    #50 rst = 1;
    #50 rst = 0;

    #200;

    $finish;
end
endmodule
```

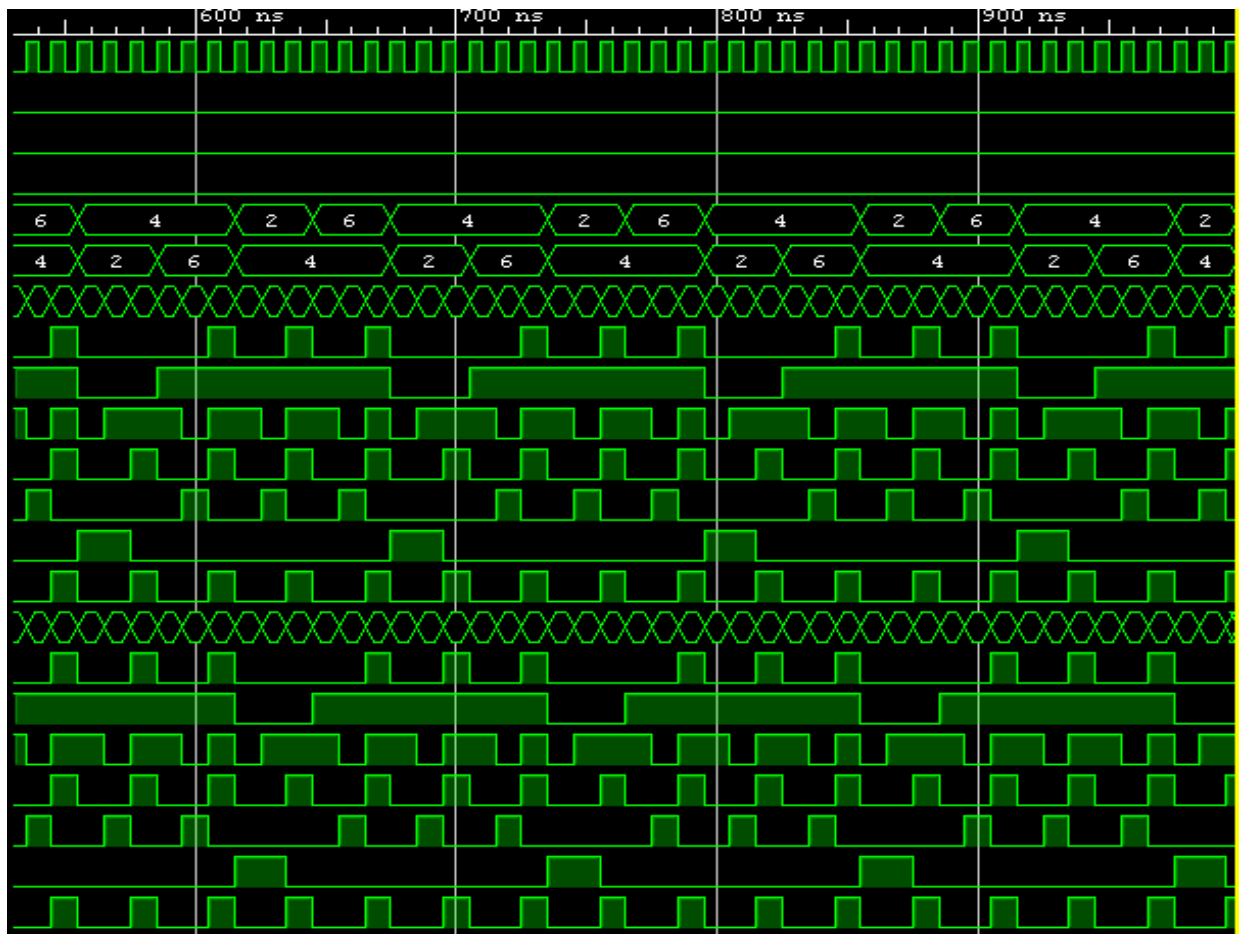


Figure 4.1: testbench illustration.

4.2 Explanation:

- Clock generation: the clock signal is toggled every 5 nanoseconds creating a clock period of 10 nano-seconds (100MHz clock frequency).
- Test sequence:
 - The clock is initialized to 0.
 - The reset ('rst') is asserted for 10 nano-seconds and then deasserted to initialize the module.
- Wait for 205 nano-seconds to allow the module to test in normal state.
- The 'emer1' signal is asserted and the traffic light 1 turns green, therefore, traffic light 2 will be turned red for 50 nano-seconds.
- The 'emer2' signal is asserted and the traffic light 2 turns green, therefore, traffic light 1 will be turned red for 50 nano-seconds.
- Both 'emer1' and 'emer2' are asserted, so both traffic lights turn green for 50 nano-seconds and then off.
- Wait for 200 nano-seconds to allow the module to test in normal state.

5 Conclusion

5.1 Working achievement:

- State machine design: successfully designed a state machine to handle normal state and some special states such as emergency traffic light state.
- Traffic light control: Implemented traffic light control logic to handle the correct sequencing of traffic lights in different states.
- Countdown timer: designed a countdown timer logic for displaying time remaining on a 7-segment display
- Module combination: successfully combined multiple modules into a cohesive module.
- Testing: developed a test bench to simulate and verify the functionality of the entire system.

5.2 Advantage of this system:

- Used for one-way streets or crossroads.
- Having an emergency state for both sides.
- Can be changed between states.

6 References

1. https://www.youtube.com/watch?v=RSz70d_VH3o&t=73s
2. <https://www.youtube.com/watch?v=Y6mbTDPlt3o>