ECE 6213

Yihui Wang

Project-2

Mustang Taillight Sequencer Datasheet

Function Description

This sequencer controls the sequential of the taillights of a Mustang, to let the lights turn on or off with correspond status of brake and turning. The layout of taillight is as follows:

Figure 1 taillight layout

And the sequencer has the following function:

1. When only turn_right is active, right_tail_light_control[2:0] must illuminate in a repeating {001} {011} {111} {000} sequence, with each state lasting five clock cycles
2. When only turn_left is active, left_tail_light_control[2:0] must illuminate in a repeating {001} {011} {111} {000} sequence, with each state lasting five clock cycles
3. When only brake is active, all tail light must illuminate
4. When turn_right and brake is active, all left_tail_light_control lights must illuminate and right_tail_light_control[2:0] must illuminate in a repeating {111} {110} {100} {000} sequence, with each state lasting five clock cycles
5. When turn_left and brake is active, all right_tail_light_control lights must illuminate and left_tail_light_control[2:0] must illuminate in a repeating {111} {110} {100} {000} sequence, with each state lasting five clock cycles
6. If the input signal has changed, the taillight pattern should change as well, even if the light phase hasn't finished five clock cycles

I/O Signal Descriptions

Name	Type	Description
clk	In	Master clock input
rst_n	In	Reset input (active low)
brake	In	Brake operation input (active high)
turn_right	In	Turn right operation input (active high)
turn_left	In	Turn left operation input (active high)
right_tail_light_controll [2:0]	Out	Right side taillight control signal output (active high)
left_tail_light_controll [2:0]	Out	Left side taillight control signal output (active high)

Simulation Result

Simulation waveform of only turn_right signal is active:

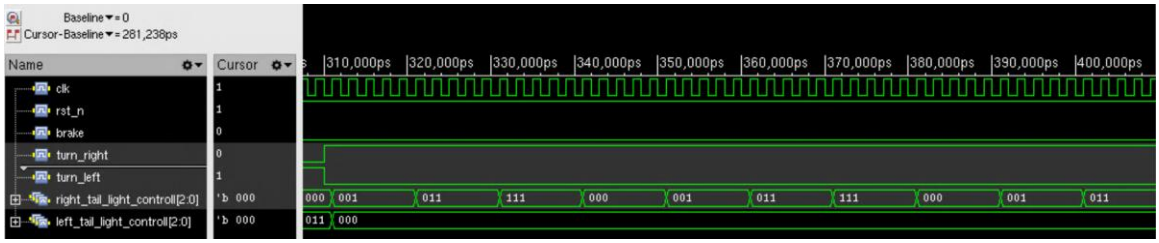

Figure 2 simulation waveform of turning right only.

Simulation waveform of only turn_left signal is active:

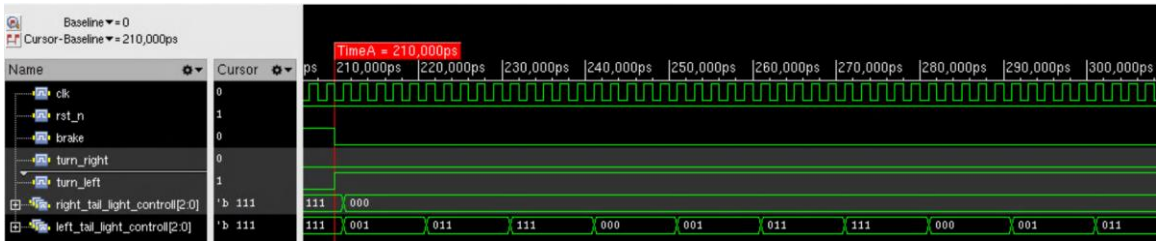

Figure 3 simulation waveform of turning left only.

Simulation waveform of only barke signal is acive:

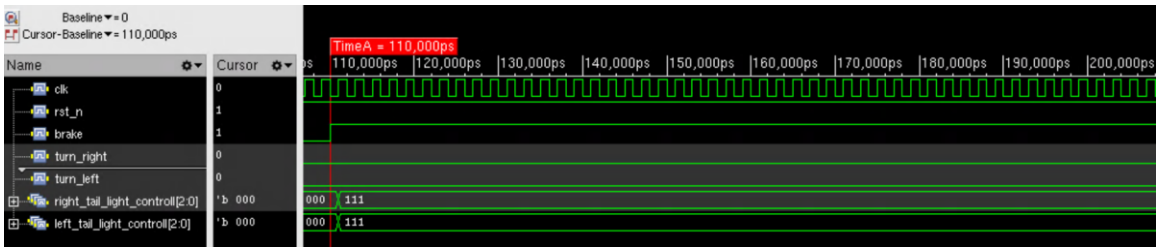

Figure 4 simulation waveform of braking only.

Simulation waveform of braking during turn_left signal is active and turn_right signal is active:

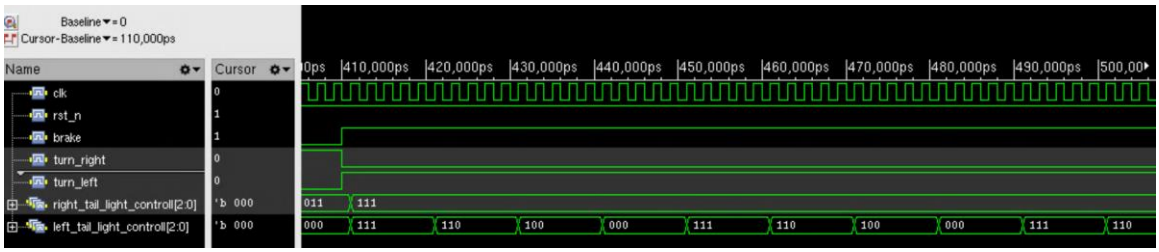

Figure 5 simulation waveform of braking during turning left.

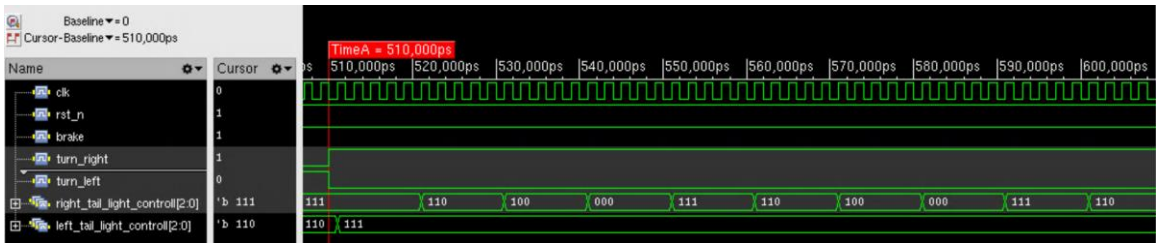

Figure 6 simulation waveform of braking during turning right.

Controller Verilog Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// ECE4150/6250
// Yihui Wang
// Taillight sequencer for Mustang
//
//
//////////////////////////////////////////////////////////////////////////////////


module tail_light_sequencer (
    input wire clk,
    input wire rst_n,
    input wire brake,
    input wire turn_right,
    input wire turn_left,

    output reg [2:0] right_tail_light_controll = 0,
    output reg [2:0] left_tail_light_controll = 0
);



    // State defination
    parameter idle = 5'b00000;
    parameter brake_only = 5'b00001;
    parameter right_only = 5'b00010;
    parameter left_only = 5'b00100;
    parameter brake_right = 5'b01000;
    parameter brake_left = 5'b10000;

    // State register
    reg [4:0] state = 0, next = 0;

    always @(posedge clk or negedge rst_n) begin
        if (rst_n == 1'b0) begin
            state <= idle;
        end
        else state <= next;
    end

    // State transition logic
    always @(*) begin
        case ({brake,turn_left,turn_right})
```

```
            3'b001: next = right_only;
            3'b010: next = left_only;
            3'b100: next = brake_only;
            3'b101: next = brake_right;
            3'b110: next = brake_left;
            default: next = idle;
        endcase
    end

    // Stage cycle counter
    wire next_stage;
    reg [4:0] counter = 5'b00001;

    always @(posedge clk or negedge rst_n) begin
        if ((rst_n == 1'b0) | (next != state)) begin
            counter <= 5'b00001;
        end
        else begin
            if (counter == 5'b10000) begin
                counter <= 5'b00001;
            end
            else begin
                counter <= counter << 1;
            end
        end
    end

    assign next_stage = (counter == 5'b10000);

    // Light output controll
    always @(posedge clk, negedge rst_n) begin
        if (rst_n == 1'b0) begin
            right_tail_light_controll <= 3'b0;
            left_tail_light_controll <= 3'b0;
        end
        else begin
            case (next)

                brake_only: begin
                    right_tail_light_controll <= 3'b111;
                    left_tail_light_controll <= 3'b111;
                end

                right_only: begin
                    if (state == right_only) begin
                        if (next_stage) begin
                            right_tail_light_controll <=
{right_tail_light_controll[1:0],right_tail_light_controll[2]} + 3'd1;
                        end
                        else begin
                            right_tail_light_controll <= right_tail_light_controll;
                        end
                        left_tail_light_controll <= 3'b0;
                    end
                    else begin
                        right_tail_light_controll <= 3'b001;
                        left_tail_light_controll <= 3'b0;
                    end
                end

                left_only: begin
                    if (state == left_only) begin
                        if (next_stage) begin
                            left_tail_light_controll <=
{left_tail_light_controll[1:0],left_tail_light_controll[2]} + 3'd1;
                        end
                        else begin
                            left_tail_light_controll <= left_tail_light_controll;
                        end
                        right_tail_light_controll <= 3'b0;
                    end
                    else begin
                        left_tail_light_controll <= 3'b001;
                        right_tail_light_controll <= 3'b0;
                    end
                end

                brake_right: begin
                    if (state == brake_right) begin
```

```
                        if (next_stage) begin
                            right_tail_light_controll <=
{right_tail_light_controll[1:0],right_tail_light_controll[2]} - 3'd1;
                        end
                        else begin
                            right_tail_light_controll <= right_tail_light_controll;
                        end
                        left_tail_light_controll <= 3'b111;
                    end
                    else begin
                        right_tail_light_controll <= 3'b111;
                        left_tail_light_controll <= 3'b111;
                    end
                end

                brake_left: begin
                    if (state == brake_left) begin
                        if (next_stage) begin
                            left_tail_light_controll <=
{left_tail_light_controll[1:0],left_tail_light_controll[2]} - 3'd1;
                        end
                        else begin
                            left_tail_light_controll <= left_tail_light_controll;
                        end
                        right_tail_light_controll <= 3'b111;
                    end
                    else begin
                        left_tail_light_controll <= 3'b111;
                        right_tail_light_controll <= 3'b111;
                    end
                end

                default: begin
                    right_tail_light_controll <= 3'b0;
                    left_tail_light_controll <= 3'b0;
                end

            endcase
        end
    end


endmodule
```

Testbench Verilog Code

```
`timescale 1ns / 1ps

module tail_light_sequencer_tb ();
    reg clk = 0;
    reg rst_n = 0;
    reg brake = 0;
    reg turn_left = 0;
    reg turn_right = 0;

    wire [2:0] right_tail_light_controll;
    wire [2:0] left_tail_light_controll;

    tail_light_sequencer sequencer(
        .clk(clk),
        .rst_n(rst_n),
        .brake(brake),
        .turn_right(turn_right),
        .turn_left(turn_left),
        .right_tail_light_controll(right_tail_light_controll),
        .left_tail_light_controll(left_tail_light_controll)
    );

    always begin
        #1
        clk = ~clk;
        $display ( "Brake = %b, turn_right = %b, turn_left = %b, right_tail_light = %b, left_tail_light = %b,
Time %d" , brake, turn_right, turn_left, right_tail_light_controll, left_tail_light_controll, $time);
    end
```

```
initial begin
    #10
    rst_n = 1;

    #100
    brake = 1;
    turn_left = 0;
    turn_right = 0;

    #100
    brake = 0;
    turn_left = 1;
    turn_right = 0;

    #100
    brake = 0;
    turn_left = 0;
    turn_right = 1;

    #100
    brake = 1;
    turn_left = 1;
    turn_right = 0;

    #100
    brake = 1;
    turn_left = 0;
    turn_right = 1;

    #100
    brake = 0;
    turn_left = 1;
    turn_right = 0;

    #3
    brake = 1;
    turn_left = 1;
    turn_right = 0;

    #100
    brake = 0;
    turn_left = 0;
    turn_right = 1;

    #3
    brake = 1;
    turn_left = 0;
    turn_right = 1;

    #100
    $finish;

end

endmodule
```