# Birzeit University logo

# Faculty of Engineering &Technology

# Electrical & Computer Engineering Department

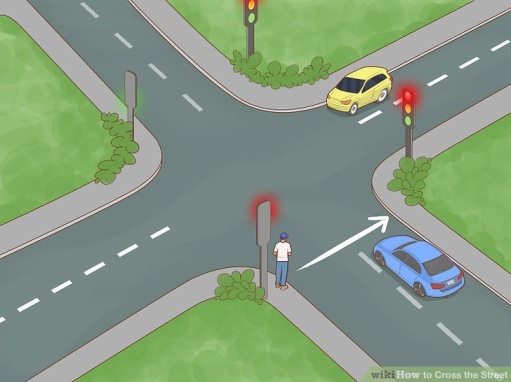
# ADVANCED DIGITAL SYSTEMS DESIGN

# ENCS3310

# PROJECT

## Introduction and background

This project is a digital circuit design for traffic light system for two roads and its test bench. The figure below shows the traffic and the table below shows the states of the traffic lights and the time they take.



Figure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State | Highway TL1 | Highway TL2 | Farm TL1 | Farm TL2 | Delay [Sec] |
| S0 | Red | Red | Red | Red | 1 |
| S1 | Red-Yellow | Red-Yellow | Red | Red | 2 |
| S2 | Green | Green | Red | Red | 30 |
| S3 | Green | Yellow | Red | Red | 2 |
| S4 | Green | Red | Red | Red | 10 |
| S5 | Yellow | Red | Red | Red | 2 |
| S6 | Red | Red | Red | Red | 1 |
| S7 | Red | Red | Red-Yellow | Red-Yellow | 2 |
| S8 | Red | Red | Green | Green | 15 |
| S9 | Red | Red | Green | Yellow | 2 |
| S10 | Red | Red | Green | Red | 5 |
| S11 | Red | Red | Yellow | Red-Yellow | 2 |
| S12 | Red | Red | Red | Green | 10 |
| S13 | Red | Red | Red | Yellow | 2 |
| S14 | Red | Red | Red | Red | 1 |
| S15 | Red | Red-Yellow | Red | Red | 2 |
| S16 | Red | Green | Red | Red | 15 |
| S17 | Red | Yellow | Red | Red | 3 |

Table

## Design philosophy

The main module traffic\_light is consists of controller and counter where counter used to know the time (number of the clock where a clock takes 1s) and controller turn on traffic lights on the proper time for the proper duration.

The main module traffic\_light code:

module traffic\_light(input clk, go, rst, output [1:0] highwaySignal1, highwaySignal2, farmSignal1, farmSignal2);

wire [6:0] count;

counter CTR(go, rst, clk, count);

controller CRL(clk, rst, count, highwaySignal1, highwaySignal2, farmSignal1, farmSignal2);

endmodule

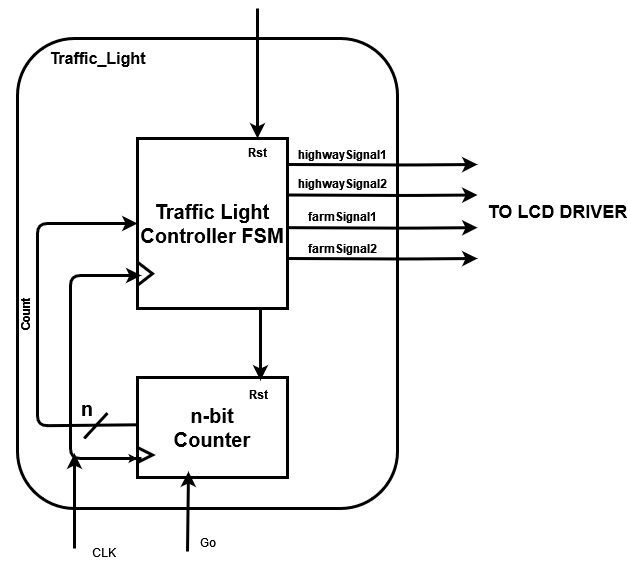


Figure - traffic\_light

Counter module when positive edge appears the output count increment by one until 107 it will reset to zero. But if ‘go’ input is low ‘0’ then the count will stop incrementing and freezes. Also when reset input ‘rst’ is high ‘1’ then the count will clear and become zero.

Counter module code:

module counter(input go, rst, clk, output reg [6:0] count=0);

always@(posedge clk) begin

if(rst) count = 0;

else if(go)

if(count>=107) count=0;

else count++;

end

endmodule

Controller module move to the next state by use an array included the time when a specific state become (‘{1,3,33,35,45,47,48,50,65,67,72,74,84,86,87,89,104,107} this array is the same as accumulated delay). The table below shows when farms traffic lights change with orange color and when highway traffic lights change with blue color. Moreover the darker color shows the specific street light that changes on any state. Also from the table can be figure that the way the lights change is by increment there light code. So in this design as you can see in the code below, when the positive edge of the clock becomes and according to states array if new state time is become, then it cheek if this is the state that the traffic light changes on if yes increment the code of the colors of the traffic light otherwise do not anything.

00 : Green 01 : Yellow (when changing from green) 10 : Red 11 : Red-Yellow (when changing from red)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| State | Highway TL1 | Highway TL2 | Farm TL1 | Farm TL2 | Delay [Sec] | state when clk |
| S0 | 10 | 10 | 10 | 10 | 1 | 0 and 107 |
| S1 | 11 | 11 | 10 | 10 | 2 | 1 |
| S2 | 00 | 00 | 10 | 10 | 30 | 3 |
| S3 | 00 | 01 | 10 | 10 | 2 | 33 |
| S4 | 00 | 10 | 10 | 10 | 10 | 35 |
| S5 | 01 | 10 | 10 | 10 | 2 | 45 |
| S6 | 10 | 10 | 10 | 10 | 1 | 47 |
| S7 | 10 | 10 | 11 | 11 | 2 | 48 |
| S8 | 10 | 10 | 00 | 00 | 15 | 50 |
| S9 | 10 | 10 | 00 | 01 | 2 | 65 |
| S10 | 10 | 10 | 00 | 10 | 5 | 67 |
| S11 | 10 | 10 | 01 | 11 | 2 | 72 |
| S12 | 10 | 10 | 10 | 00 | 10 | 74 |
| S13 | 10 | 10 | 10 | 01 | 2 | 84 |
| S14 | 10 | 10 | 10 | 10 | 1 | 86 |
| S15 | 10 | 11 | 10 | 10 | 2 | 87 |
| S16 | 10 | 00 | 10 | 10 | 15 | 89 |
| S17 | 10 | 01 | 10 | 10 | 3 | 104 |

Table

Controller module code:

module controller(input clk, rst, [6:0] count, output reg [1:0] highwaySignal1=2, highwaySignal2=2, farmSignal1=2, farmSignal2=2);

reg [6:0] state [0:17] = '{1,3,33,35,45,47,48,50,65,67,72,74,84,86,87,89,104,107}; //ppears in any clk

reg [4:0] s = 0;

always@(posedge clk && count >= state[s] && rst == 0) begin

if(s >= 6 && s < 14) begin

if((s <= 7) || (s >= 10 && s <= 11)) farmSignal1++;

farmSignal2++;

end

else begin

if((s >= 0 && s <= 1) || (s >= 4 && s <= 5)) highwaySignal1++;

if((s >= 0 && s <= 3) || (s >= 14)) highwaySignal2++;

end

if(s == 17) s=0;

else s++;

end

endmodule

traffic\_light\_TB module includes generate\_test and analyzer. This module create 108 clocks where 107 is the last value that the counter reach it and use repeat(216) where clock has to change two times to get one positive edge (108\*2=216). Then this module test ‘rst’ input is high ‘1’ and when ‘go’ input is low ‘0’.If the generator output values match the traffic light system output values then traffic\_light\_TB print “fault free”. But if any values of results and expected results do not matched traffic\_light\_TB will receives high signal ‘1’ from the analyzer at error wire then traffic\_light\_TB will stop clocking and error message will printed by the analyzer where that needs lees code.

traffic\_light\_TB code:

module traffic\_light\_TB();

reg clk=0;

reg go=1, rst=0;

reg [7:0] expectedResult; reg [1:0] highwaySignal1; reg [1:0] highwaySignal2;

reg [1:0] farmSignal1; reg [1:0] farmSignal2; reg [6:0] count;

wire error;

generate\_test GT(clk, go, rst, count, expectedResult);

traffic\_light TL(clk, go, rst, highwaySignal1, highwaySignal2, farmSignal1, farmSignal2);

analyzer AZ(clk, count, {highwaySignal1, highwaySignal2, farmSignal1, farmSignal2}, expectedResult, error);

initial begin

repeat(216) begin //(107clk + 1)\*2

#500ms if(error) break; //500ms\*2 = 1s

clk = ~clk;

end

if(!error) begin

rst=1; #500ms clk = ~clk; #500ms clk = ~clk; //test reset

rst=0; #500ms clk = ~clk; #500ms clk = ~clk;

end

if(!error) begin

go=0; #500ms clk = ~clk; #500ms clk = ~clk; //test go

go=1; #500ms clk = ~clk; #500ms clk = ~clk;

end

if(!error) $monitor("fault free");

end

endmodule

generate\_test mudule use memory method to generate the expected results where when the counter pass the indicated time of the previous state then change state number ‘s’ and this state number is the same as the index number of the output values that stores in the memory that named ‘state’ where stores as 8bit blocks.

### generate\_test code:

module generate\_test(input reg clk, reg go, reg rst, output reg [6:0] count=0, reg [7:0] expectedResult);

reg [6:0] state [0:17] = '{1,3,33,35,45,47,48,50,65,67,72,74,84,86,87,89,104,107};

reg [4:0] s = 0;

reg [7:0] results [0:17] = '{8'b10101010,8'b11111010,8'b00001010,8'b00011010,

8'b00101010,8'b01101010,8'b10101010,8'b10101111,8'b10100000,8'b10100001,

8'b10100010,8'b10100111,8'b10101000,8'b10101001,8'b10101010,8'b10111010,

8'b10001010,8'b10011010};

initial expectedResult = results[s];

always@(posedge clk) begin

if(rst) count=0;

else if(go) begin

count++;

if(count >= state[s])

if(s == 17) begin

s=0;

count=0;

end

else s++;

expectedResult = results[s];

end

end

endmodule

analyzer module sets the error flag to high ‘one’ and display error message with counter value when the error happens if a result value and corresponding expected value dose not matched.

### analyzer module code:

module analyzer(input clk, [6:0] count, reg [7:0] result, reg [7:0] expectedResult, output reg error = 0);

always@(posedge clk or count == 0) begin

#100ms;

if(result !== expectedResult) begin

error = 1;

$monitor("faulty when counter =%d\nresult = %b <and> expectedResult = %b", count, result, expectedResult);

end

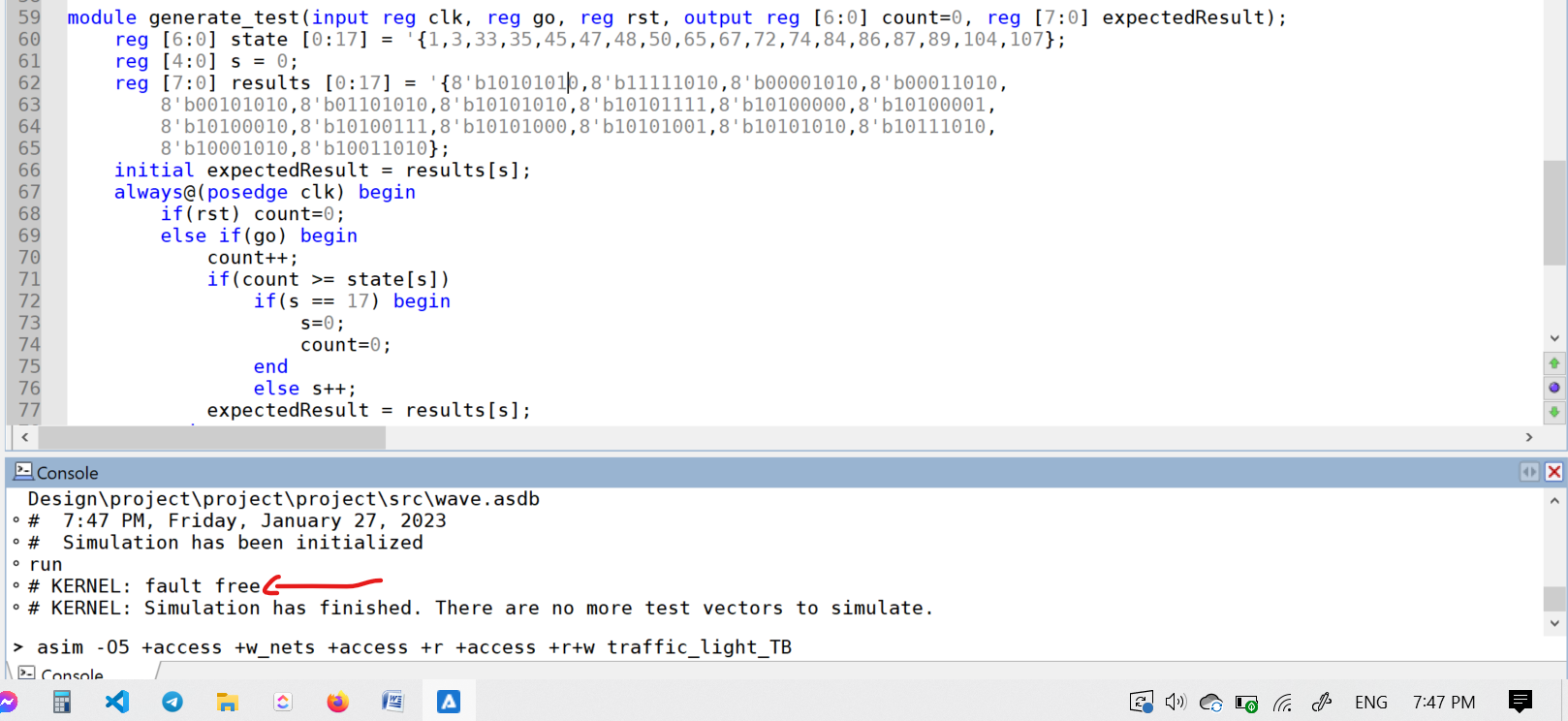
end

endmodule

## Results

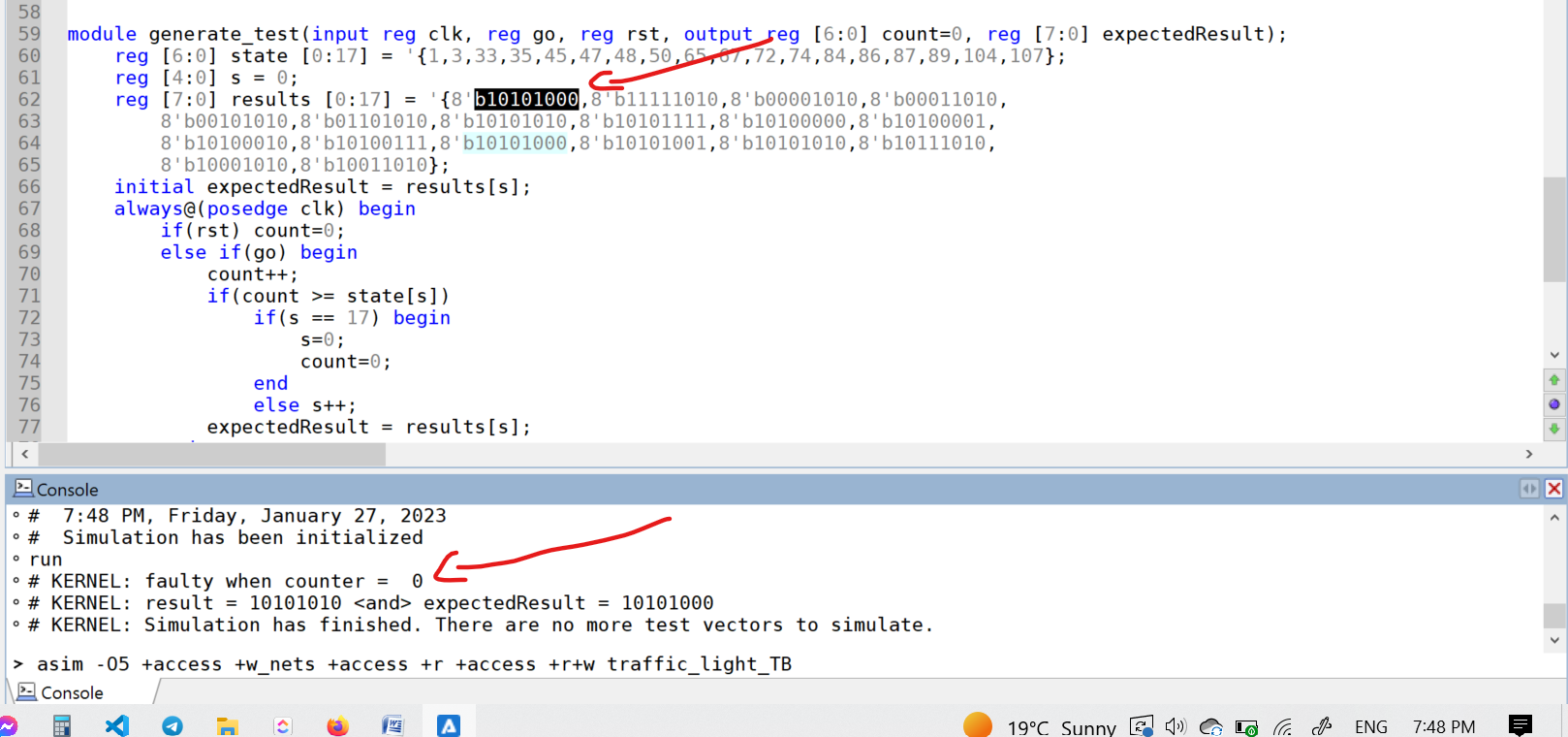
In the figures below (figure3 to 6) we can see that the test tell us that our system structure works as we want, and when added a deliberate error the test bench recognize it and tell when the error occur.

### Normal condition:



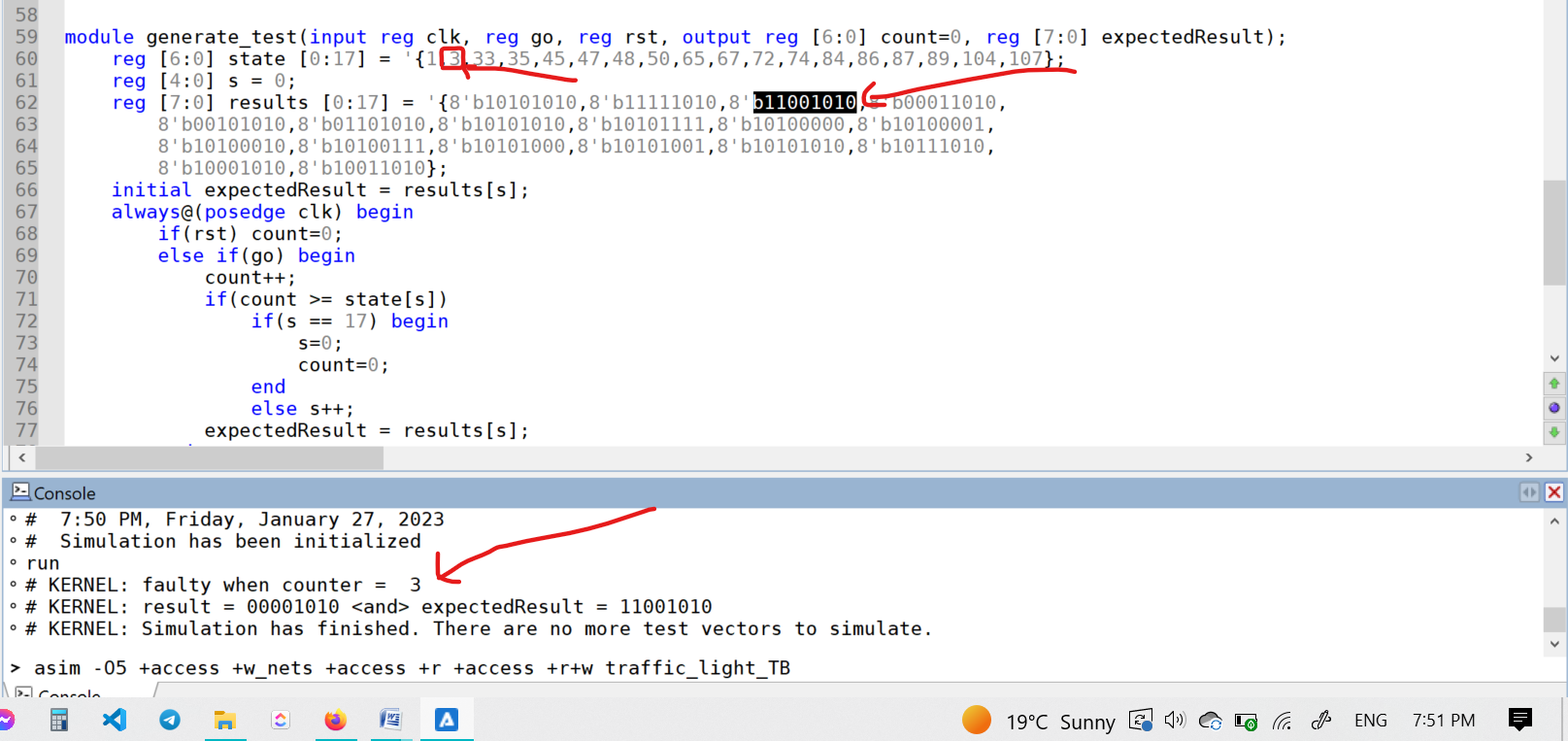
Figure

### Deliberate error



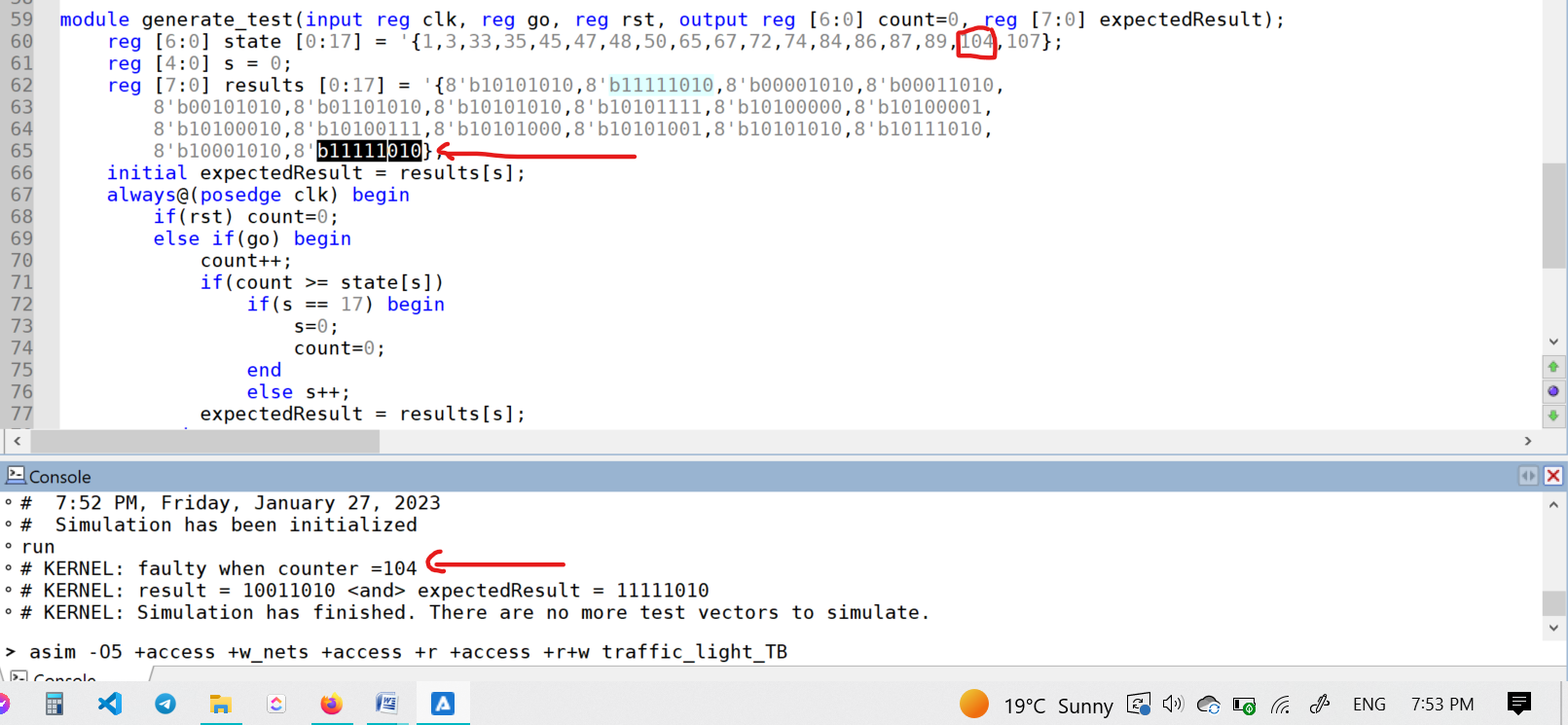
Figure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S0 | 10 | 10 | 10 | 10 | 1 | 0 and 107 |



Figure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S2 | 00 | 00 | 10 | 10 | 30 | 3 |



Figure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S17 | 10 | 01 | 10 | 10 | 3 | 104 |

## Conclusion and Future works

This system works with best effort way as possible but the test bench use the more confident way. Where the system use if statement side by side with increment to color code and test bench use memory. This to make the IC chip of the system needs less space so less cost and to make us surer with test bench results.

In future this system may be more scalable with time of color lights changes, for example may have two inputs works as multiplier factor number, one with highway streets open and farm streets close and other for farm streets open and highway streets close.