Faculty of Engineering & Technology
Electrical & Computer Engineering Department

Advanced Digital System Design ENCS3310

Project Report

Prepared by:
Yazeed Hamdan 1201133

Instructor: Dr. Abdallatif Abuissa

Section: 1

Date: 24-1-2023

Table of contents

List of figures

List of tables

Brief introduction and background

This project aims to design a solution to real problem which is the traffic light design for two roads, by using Verilog programming. Also, to build different modules like state machine and be more aware with different design concepts by using Verilog like verification.

The task

Is to create a solution for traffic light system shown in Figure 1:

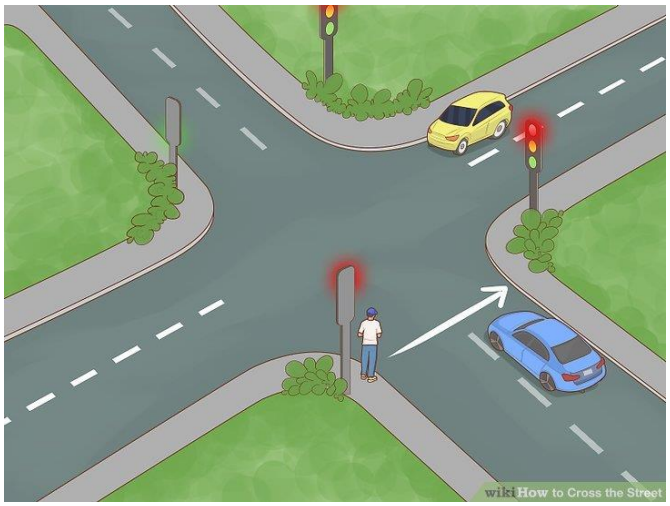

Figure 1: Highway and farm roads

The following block diagram for the traffic lights show us how the design should work:

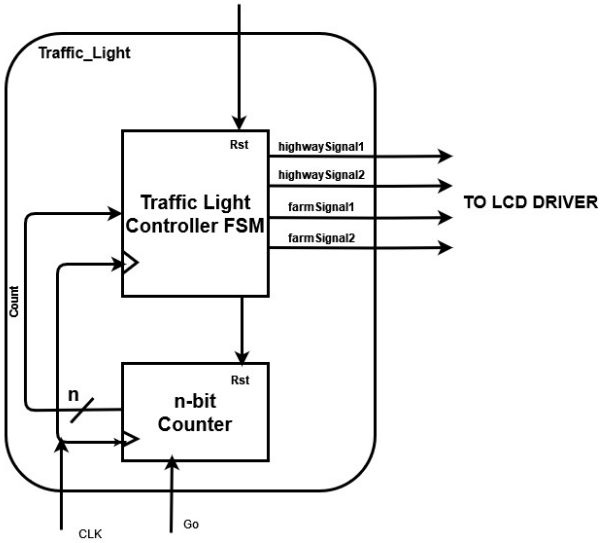

Figure 2: block diagram

We have to design a traffic light controller for the highway and farm road intersection shown in Figure 1. The traffic light controller outputs four 2-bit signals, highway Signal 1 and highway signal 2 and farm Signal 1 and 2, for the highway road and the farm road, respectively. The following encoding is used for both signals:

Table 1:encoding of signals

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

Table 2 :All the states of the traffic light

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

Design philosophy

My design was built on this principle of make only one module for state machine and the counter, I used the counter as a register 5 bits with initial value equal to zero, for each execution of always block, the counter increase by one until reach the delay value for each state, then the counter will be zero and we will move to the next state.

In design verification part, ROM was used to store the correct values for all states, then we have to compare this values with state machine output values to check if our design work correctly or not.

1. State machine and simple testbench

Step 1

First, in step 1, I built state_yaz module, it has three inputs, rst represent the reset, this input returns design to state 0, and counter to 0. Clk represent the clock, and go if go is forced to 0 then the counter will stop counting (freeze). Also, there is four 2 bits outputs as described in the introduction and the block diagram, each output represents a traffic light for the road. I used two registers, first one called counter with 5 bits, second one called state with 5 bits.

```
1       // Yazeed Hamdan 1201133 Sec.1
2   module state_yaz (clk,rst,go,hiway1,hiway2,farm1,farm2);
3       input clk,rst,go;
4       output reg [1:0] hiway1 ;
5       output reg [1:0] hiway2;
6       output reg [1:0] farm1;
7       output reg [1:0] farm2;
8       reg [4:0] counter;
9       reg [4:0] state;
```

Figure 3: Code for step 1

Step 2

Second, I used parameter keyword to define all the 18 states shown in table 2, each state has 5 bits with different binary value, The values of states arranged in ascending order.

```
10      parameter s0 = 5'b00000,
11    s1 = 5'b00001,
12    s2 = 5'b00010,
13    s3 = 5'b00011,
14    s4 = 5'b00100,
15    s5 = 5'b00101,
16    s6 = 5'b00110,
17    s7 = 5'b00111,
18    s8  =5'b01000,
19    s9 = 5'b01001,
20    s10 =5'b01010,
21    s11= 5'b01011,
22    s12= 5'b01100,
23    s13= 5'b01101,
24    s14 =5'b01110,
25    s15 =5'b01111,
26    s16 =5'b10000,
27    s17= 5'b10001;
```

Figure 4: Code for step 2

Step 3

In this step, always block was used as shown in figure 5 below, with positive edge for the clock and negative edge for the reset. If reset value equal to zero, state register will store s0 and counter value will be equal to zero, else if reset value equal to 1, then, there will be 18 cases with different 18 states, the counter will count when go value equal to 1 until reach the require delay value for each state, but, when go value is zero the counter will freeze. After reach the value of delay, we will set the counter value to zero and move to the next state.

This part of the code shown in figure 5 was repeated for another 17 states with different value of delay as shown in the attached file.

```
always @(posedge clk or negedge rst or go)
    begin

        if(rst==0)
            begin
                state <= s0;
                counter<= 5'b00000;
            end
        else

                case (state)
        s0:
            if (counter <  5'b00001 || ~go)
                begin
                    state <= s0;
                    if(go)
                    counter <= counter +1;
                end
            else
                begin
                    state <= s1;
                    counter<=0;
                    end
```

Figure 5: Code for step3

Step 4

In this step, the output values were given for each state as shown in table 2, we have 18 state and each state has 4 different output value, the encoding of this values are shown as comments in the code below. The complete code of this step with the other 17 state shown in the attached file.

```
// 00 : Green
//01 : Yellow (when changing from green)
//10 : Red
//11 : Red-Yellow (when changing from red)
        always@(*)
            begin
                case(state)
        s0:
            begin
                hiway1<= 2'b10;
                hiway2<= 2'b10;
                farm1<=  2'b10;
                farm2<= 2'b10;
            end
```

Figure 6: Code for step4

Simple testbench

After building the state machine module (state_yaz), the simple testbench (traffic_tb) was made to check if the code gives us logical results or not. Inputs defines as registers, outputs define as wires, then, state machine module was called. I used monitor task to print the output values in the console. In initial block, clock, reset, and go were set to zero, after that, after 10 nano seconds reset value will change to one, after 60 nano seconds go value will be equal to one, and each 5 nano seconds clock value will change.
#600ns $finish; this task means that after 600 nano seconds the simulation will be finished.
The waveform of this testbench is shown in results part.

```
// module state_yaz (clk,rst,go,hiway1,hiway2,farm1,farm2);
//  output reg [1:0] hiway2;
//  output reg [1:0] farm1;
//  output reg [1:0] farm2;

module traffic_tb;
            reg CLK;
            reg RST;
            reg GO;
            wire [1:0]Hiway1;
            wire [1:0]Hiway2;
            wire [1:0]Farm1;
            wire [1:0]Farm2;
    state_yaz  stage(CLK,RST,GO,Hiway1,Hiway2,Farm1,Farm2);
    initial
        begin
    $monitor("when t = %0d  : Highway1 = %b Highway2 = %b farm1 =%b farm2 =%b",$time,Hiway1,Hiway2,Farm1,Farm2);
        CLK = 0 ;
        RST = 0 ;
        GO = 0 ;
        #10ns RST = ~RST;
        #60ns GO = 1;
        #600ns $finish ;
        end
        always #5ns CLK = ~CLK;

    endmodule
```

Figure 7: simple testbench

2. Design verification

ROM (generator)

In this module, I built read only memory (ROM) to store all values of states shown in table 2, the input (Address) has 5 bits because we have 18 states so we need 5 bits to represent them. The output (Data) has 8 bits because we have 4 outputs in the system each output has 2 bits. I used a register mem to store the values in it, with 18 cell each cell has 8 bits.

```
module generateTests(Address,Data);
input [4:0] Address;
output [7:0] Data;
reg [7:0] mem [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};
assign Data = mem[Address];
    endmodule

```

Figure 8: ROM code

Generator testbench

The testbench module for generator was built as shown in figure 9 below, it takes the address as reg with 5 bits, and takes Data as wire with 8 bits.
in initial block, I set the address value to zero, and increment the value by 1, and repeat the increment another 17 times to reach for all values of states.
the waveform of this testbench is shown in result part of the report.

```
module Gen_tb;
            reg [4:0] Address;
            wire [7:0] Data;
    generateTests  stage(Address,Data);
    initial
        begin
        Address = 5'b00000 ;
        repeat(17)
        #5ns Address = Address +1'b1 ;
        #600ns $finish ;
        end

    endmodule
```

Figure 9: generator testbench

Final system module with analyzer

In this part, the modules of state machine and generator were called, each module takes parameters from the final system as inputs and outputs. Then, the analysis was done in the always block, this block works at the positive edge of the clock. In if statement we compare the value of generator output (Data) with state machine output (state_out), if the two outputs does not equal to each other, the first display task will observe in the console, else if the two outputs equal, the second display task will observe.

```
module final_system(clk,go,rst,Address,Data,state_out);
    input clk,go,rst;
    output  reg [7:0] state_out;
    input  [4:0] Address;
   output  reg [7:0] Data;

     state_yaz ss    (clk,go,rst,state_out[7:6],state_out[5:4],state_out[3:2],state_out[1:0]);
    generateTests sa  (Address,Data);
    always @(posedge clk  or Address )
            if(Data != {state_out[7:6],state_out[5:4],state_out[3:2],state_out[1:0]})
        $display ("ERROR, Your data does not equal to the true values, in address %b, generator = %b , state out = %b",Address,Data,state_out);
            else
                $display("OK, your data matched with the true values, in address %b, generator = %b , state out = %b",Address,Data,state_out);
            endmodule
```

Figure 10: final system code

Final system and analyzer testbench

In this testbench, the clock, reset, go, and address were defined as registers, and the outputs of the generator and state machine defined as wires, then the final system was called. In first initial block the value of address was set to zero then it was incremented by one until reach the last state of the all states. In second initial block, the values of clock, reset, and go were set to zero, after that, after10 nano seconds reset value will be equal to one, after 60 nano seconds go value will be equal to one, and each 5 nano seconds clock value will change.
#200ns $finish; this task means that after 200 nano seconds the simulation will be finished.

```
511  module System_tb;
512            reg  clk,rst,go;
513            reg  [4:0] Address;
514           wire [7:0] state_out;
515            wire [7:0] Data;
516
517       final_system agag(clk,go,rst,Address,Data,state_out);
518                    initial
519                begin
520                    Address = 5'b00000;
521                    repeat   (17)
522                    #5ns Address = 1'b1 + Address;
523                    end
524           initial
525                begin
526                    clk = 0 ;
527                    rst = 0 ;
528                    go = 0 ;
529                    #10ns rst = ~rst;
530                    #60ns go = 1;
531                    #200ns $finish ;
532                end
533           always #5ns clk = ~clk;
534
535                endmodule
```

Figure 11: Final system and analyzer testbench

Results

Simple testbench for state machine

After building testbench for state machine, the following waveform was observed and we can notice that it gives us true and expected results. The results were printed on the console using monitor task until reach the time of finishing the simulation.

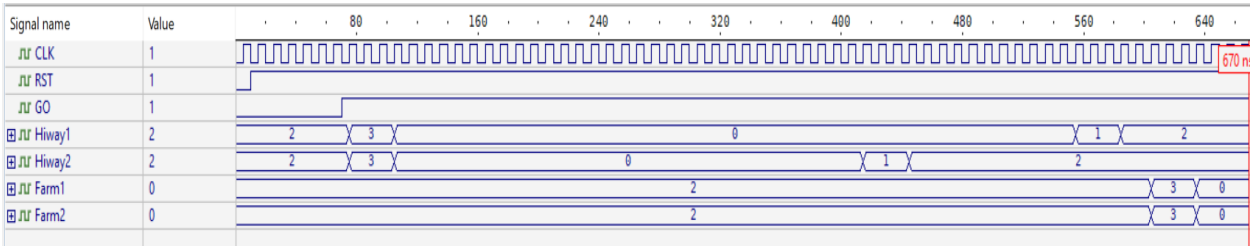

Figure 12: waveform for state machine

```
Console
run
# KERNEL: when t = 0   : Highway1 = 10 Highway2 = 10 farm1 =10 farm2 =10
# KERNEL: when t = 75000   : Highway1 = 11 Highway2 = 11 farm1 =10 farm2 =10
# KERNEL: when t = 105000   : Highway1 = 00 Highway2 = 00 farm1 =10 farm2 =10
# KERNEL: when t = 415000   : Highway1 = 00 Highway2 = 01 farm1 =10 farm2 =10
# KERNEL: when t = 445000   : Highway1 = 00 Highway2 = 10 farm1 =10 farm2 =10
# KERNEL: when t = 555000   : Highway1 = 01 Highway2 = 10 farm1 =10 farm2 =10
# KERNEL: when t = 585000   : Highway1 = 10 Highway2 = 10 farm1 =10 farm2 =10
# KERNEL: when t = 605000   : Highway1 = 10 Highway2 = 10 farm1 =11 farm2 =11
# KERNEL: when t = 635000   : Highway1 = 10 Highway2 = 10 farm1 =00 farm2 =00
```

Figure 13: output of the state machine in the console

Testbench for the generator

After building testbench for generator, the following waveform was observed and we can notice that it gives us true and expected results.

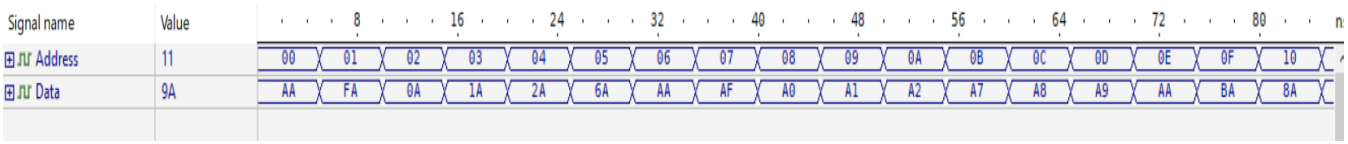

Figure 14: waveform for the generator

Final system module with analyzer results

After building module of the final system with analyzer and the test bench, the outputs were observed in the console, I got the expected results in some cases as shown in figure 15, in address 00110 the output value of generator equals to the output value from the state machine 10101010 (red red red red), and other case shown in figure 16, in address 00001 the output value from generator (11111010) does not equal to output value from state machine (10101010), so we print for the user that there is error and his data doesn't equal to the expected result.

```
# KERNEL: OK, your data matched with the true values, in address 00110, generator = 10101010 , state out = 10101010
```

Figure 15: output of final_system module

```
# KERNEL: ERROR, Your data does not equal to the true values, in address 00001, generator = 11111010 , state out = 10101010
```

Figure 16: output of final_system module

Conclusion and Future works

After working in this project, I gained experience dealing with Verilog, we designed a solution for traffic light problem, I built a different modules each one based on a different subject, for example state machine code built on the principle of RTL coding, and generator code built on the principle of ROM, this project make us more familiar with the program active HDL student edition. And we deal with design verification subject by built different modules and analyze their outputs. After simulate the different modules and make a waveform for them I get the expected results from these waveforms.