Faculty of Engineering and Technology

Electrical and Computer Systems Engineering Department

ADVANCED DIGITAL SYSTEMS DESIGN (ENCS3310)

Course Project

Student Name: Christina Saba

Student ID: 1201255

Instructor: Dr. Abdellatif Abu-Issa

Section: 1

Date: 27-1-2023

Table of contents:

Table of figures:

Introduction:

Through the years, the development of technology applications grew and expanded in which connected to the real-life situations of millions of people. This act has revolutionized how things work making tasks more efficient and convenient. These implementations can be widely seen in transportations, healthcare, education, etc. In this course project, one of these techniques is designing a traffic light controller which has been the solution to one of the major issues many countries have faced due to the increasing number of vehicles accessing the road. This design is based on signals that control and maintain the entire functions of the traffic light and hence ensure the order and safety of the moving cars across the intersections. The aim is to guarantee the efficient movement of the vehicles based on the delays on each lane and therefore manage the flow of traffic. Engineers study and analyze some factors including the number of lanes and traffic complexity and utilize the simulation to easily adjust the signal timing and phasing patterns to make sure that it is adaptable and maintained.

Objective:

The task is to design a solution to real problem which is the traffic light design for two roads.

- **The task:**

Create a solution for the traffic light system across an intersection shown in Figure 1 and 2:

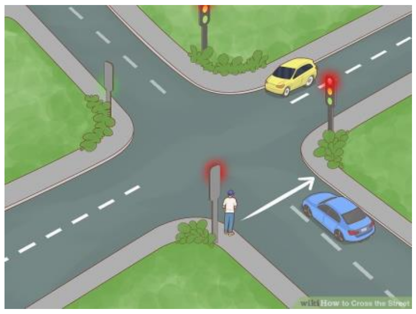


Figure1: Highway and Farm roads

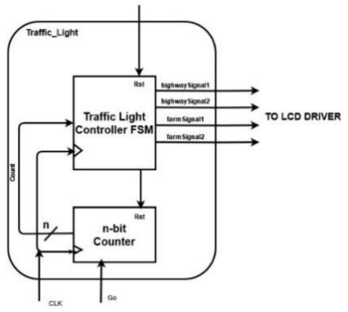


Figure 2: design block diagram

This task is solved by designing a traffic light controller module which is based on multiple inputs such as the GO, RESET, CLOCK which are classified as the inputs of the design, and therefore the outputs are the lights associated with each road based on the delay time assigned by the counter shown in Figure 2.

The design outputs four 2-bit signals of the highway and the farm road, considered as highway signal 1, highway signal 2, farm road signal 1, and farm road signal 2. The signals are encoded based on the following:

00: Green, 01: Yellow (when changing from green), 10: Red, 11: Red-Yellow (when changing from red). These light signals are divided into several states based on the time delay counted observed in the given table below. This design has three inputs are mentioned below:

1- RESET: returns the state to state 0 and counter to 0.
2- GO: controls the movement between different states, if GO=1, then the system can move across the states, while if GO=0, then the counter will freeze and stop counting.
3- CLOCK: a synchronized clock which keeps the system synchronized.

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

Table1: Table of states

• Building the project:

This project design is implemented based on the state diagram method of 18 states, each state with the traffic lights contributed to each lane (highway and farm road).

1- Implementation:

The design is implemented based on the traffic light controller module which has the RESET, GO, and CLOCK as inputs and four 2-bit outputs across the roads. As shown in Figure3, the state diagram is utilized to arrange the states. The 18 states are assigned as 5-bit registers related to the parameters s(0-17) to differentiate between the values of each one, the traffic lights are assigned as 2-bit parameters as well, the delay is considered as parameter to assign each state with its delay to change the traffic lights across the roads, and the four 2-bit outputs for the roads (highway (HW1 and HW2) and farm road (FS1 and FS2)) which indicate the traffic lights.

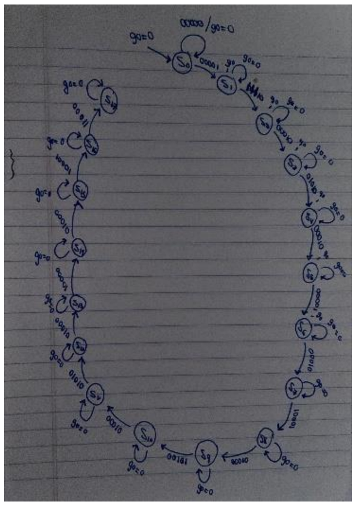
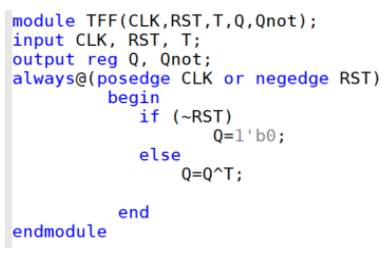
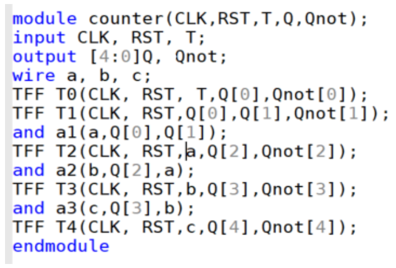


Figure3: Design's State diagram

At first I thought of having a counter module that is built using T flip-flop to count up to 5 bits wide shown in the below Figure4, but I haven't got the right results in the main module so I decided to cancel this module and use the counter as a register in the same traffic light controller module and control its value according to each state. So, the counter's bits won't be more than 5 (30 is maximum delay) since each state converts the counter back to zero in order to begin counting again in the next state for the specified delay of each one, in order to have the least number of bits used instead of incrementing the values which may consume more than 5 bits wide.



```
module TFF(CLK,RST,T,Q,Qnot);
input CLK, RST, T;
output reg Q, Qnot;
always@(posedge CLK or negedge RST)
            begin
                if (~RST)
                        Q=1'b0;
                else
                    Q=Q^T;

            end
endmodule
```



```
module counter(CLK,RST,T,Q,Qnot);
input CLK, RST, T;
output [4:0]Q, Qnot;
wire a, b, c;
TFF T0(CLK, RST, T,Q[0],Qnot[0]);
TFF T1(CLK, RST,Q[0],Q[1],Qnot[1]);
and a1(a,Q[0],Q[1]);
TFF T2(CLK, RST,a,Q[2],Qnot[2]);
and a2(b,Q[2],a);
TFF T3(CLK, RST,b,Q[3],Qnot[3]);
and a3(c,Q[3],b);
TFF T4(CLK, RST,c,Q[4],Qnot[4]);
endmodule
```

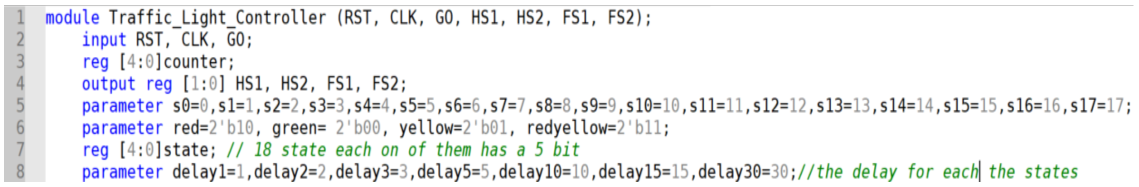
Figure4: Counter Module Code

2- Code Specification:

Four modules are used: Traffic_Light_Controller, Traffic_Light_Controller_Generator, Traffic_Light_Controller_Analyzer, Traffic_Light_Controller_tb.

A- Traffic light controller module:

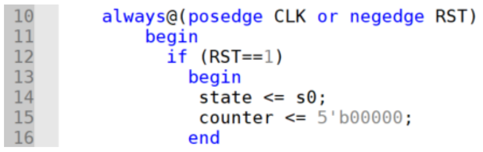
This module is based on the state diagram to switch between the cases based on the delays given and therefore, assign the right values to the outputs based on the clock's positive edge.



```
module Traffic_Light_Controller (RST, CLK, GO, HS1, HS2, FS1, FS2);
    input RST, CLK, GO;
    reg [4:0]counter;
    output reg [1:0] HS1, HS2, FS1, FS2;
    parameter s0=0,s1=1,s2=2,s3=3,s4=4,s5=5,s6=6,s7=7,s8=8,s9=9,s10=10,s11=11,s12=12,s13=13,s14=14,s15=15,s16=16,s17=17;
    parameter red=2'b10, green= 2'b00, yellow=2'b01, redyellow=2'b11;
    reg [4:0]state; // 18 state each on of them has a 5 bit
    parameter delay1=1,delay2=2,delay3=3,delay5=5,delay10=10,delay15=15,delay30=30;//the delay for each the states
```

Figure5: initiating the inputs, outputs, registers, and parameters

Since the design is synchronized based on the clock then an always@ statement is utilized based on the positive edge of the clock and the negative edge of the RST and GO value shown in the below Figure4.

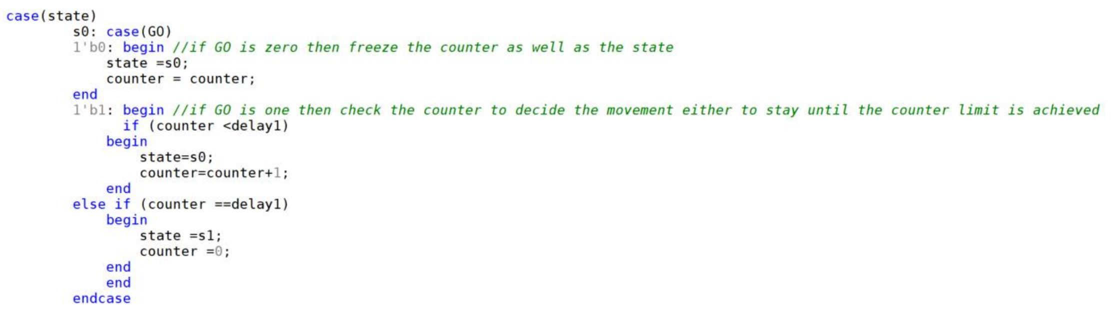


```
always@(posedge CLK or negedge RST)
    begin
      if (RST==1)
        begin
         state <= s0;
         counter <= 5'b00000;
        end
```

Figure6: Synchronizing the design

Since the reset's value restricts the movement of the states then an if statement is added to check if the value of the reset is 1 then the state remains at s0 and the counter in assigned to 0.

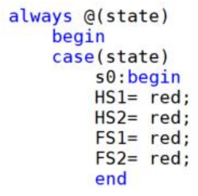
If the reset is not 1 then the design can start based on a case statement across the states with the restriction of the GO value which is then applied in a case statement, case 0 the counter is frozen in addition to the current state, case 1, then check the counter if still not equal to the next state's delay then the counter is incremented and remain in the current state, if the counter is equal to the limit then the case can move to the next state and the counter is set to 0 to begin the counter again in the next. Figure5 below is an example of the first state s0 which either can stay in s0 or move to s1 based on the counter if it is equal to 1, and the same is applied for the other 17 states.



```
case(state)
        s0: case(GO)
        1'b0: begin //if GO is zero then freeze the counter as well as the state
            state =s0;
            counter = counter;
        end
        1'b1: begin //if GO is one then check the counter to decide the movement either to stay until the counter limit is achieved
              if (counter <delay1)
            begin
                state=s0;
                counter=counter+1;
            end
        else if (counter ==delay1)
            begin
                state =s1;
                counter =0;
            end
            end
        endcase
```

Figure7: Example of state0 based on GO

After the case selected the state then it is assigned to a second case in which gives the four outputs the values (traffic light colors) according to the case the delay is assigned to. Figure6 is an example of s0 where the outputs are given red value (2’b10) for 1 second delay.

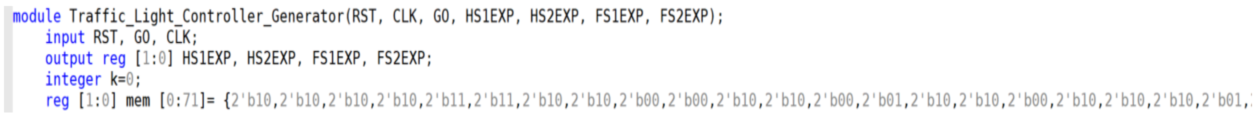


```
always @(state)
    begin
    case(state)
        s0:begin
        HS1= red;
        HS2= red;
        FS1= red;
        FS2= red;
        end
```

Figure8: Assigning the outputs according to each state

B- Traffic light controller Generator module:

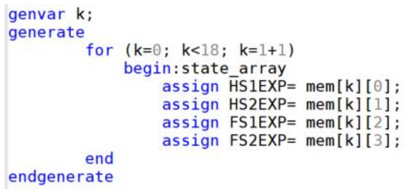
This module is utilized to save the correct data in an unpacked array of memory shown in below Figure9, array with 72 rows (each lane’s light according to each state) each value is 2-bit wide. I saved 72 because we have 18 state (s0-s17) each state with 4 outputs so all outputs are 18*4=72. The values are then observed and used to compare correct expected result with the outputs of the Traffic light controller module in the analyzer module.



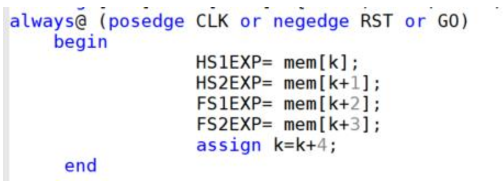
```
module Traffic_Light_Controller_Generator(RST, CLK, GO, HS1EXP, HS2EXP, FS1EXP, FS2EXP);
    input RST, GO, CLK;
    output reg [1:0] HS1EXP, HS2EXP, FS1EXP, FS2EXP;
    integer k=0;
    reg [1:0] mem [0:71]= {2'b10,2'b10,2'b10,2'b10,2'b11,2'b11,2'b10,2'b10,2'b00,2'b00,2'b10,2'b10,2'b00,2'b01,2'b10,2'b10,2'b00,2'b10,2'b10,2'b10,2'b01,
```

Figure9: Generator Code

After saving the value in the array mem I tried to do the generator method to use the behavioral way and assign the values in each row (each state) to the outputs but the simulation didn’t work and only stuck at value (10 of first state) so I switched to using the always@ according to the positive edge and utilized a for loop based on the array values. Each time the value is given the integer K is incremented by 4 in order to move across all states (rows).



```
genvar k;
generate
        for (k=0; k<18; k=1+1)
            begin:state_array
                assign HS1EXP= mem[k][0];
                assign HS2EXP= mem[k][1];
                assign FS1EXP= mem[k][2];
                assign FS2EXP= mem[k][3];
        end
endgenerate
```



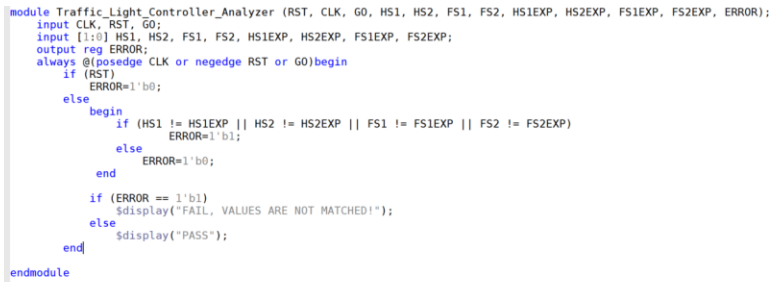
```
always@ (posedge CLK or negedge RST or GO)
    begin
                    HS1EXP= mem[k];
                    HS2EXP= mem[k+1];
                    FS1EXP= mem[k+2];
                    FS2EXP= mem[k+3];
                    assign k=k+4;
     end
```

Figure10: Generate method

Figure11: Always@ method

C- Traffic lighter controller analyzer module:

This module compares the expected data assigned as inputs (from generator) and other four inputs (from the main module) and if they are not matched then the error bit which is the output is 1 and a message is displayed that the program failed and the values don’t match, while if the values are the same then the method is correct and the error is 0 and the message “PASS” is displayed.



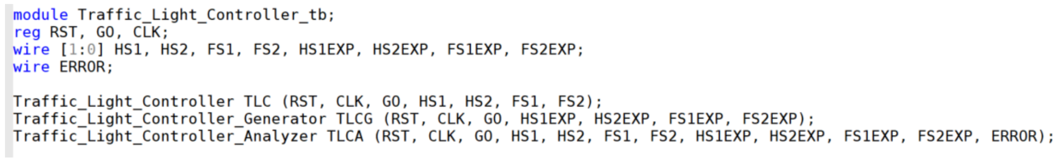
```verilog
module Traffic_Light_Controller_Analyzer (RST, CLK, GO, HS1, HS2, FS1, FS2, HS1EXP, HS2EXP, FS1EXP, FS2EXP, ERROR);
    input CLK, RST, GO;
    input [1:0] HS1, HS2, FS1, FS2, HS1EXP, HS2EXP, FS1EXP, FS2EXP;
    output reg ERROR;
    always @(posedge CLK or negedge RST or GO)begin
        if (RST)
            ERROR=1'b0;
        else
            begin
                if (HS1 != HS1EXP || HS2 != HS2EXP || FS1 != FS1EXP || FS2 != FS2EXP)
                        ERROR=1'b1;
                else
                    ERROR=1'b0;
             end

            if (ERROR == 1'b1)
                $display("FAIL, VALUES ARE NOT MATCHED!");
            else
                $display("PASS");
        end

endmodule
```

Figure12: Analyzer code

Based on the positive edge of the clock the analyzer checks each state and compares the outputs with the expected using the if statement to raise the error if not matched.

D- Traffic light controller test bench module:

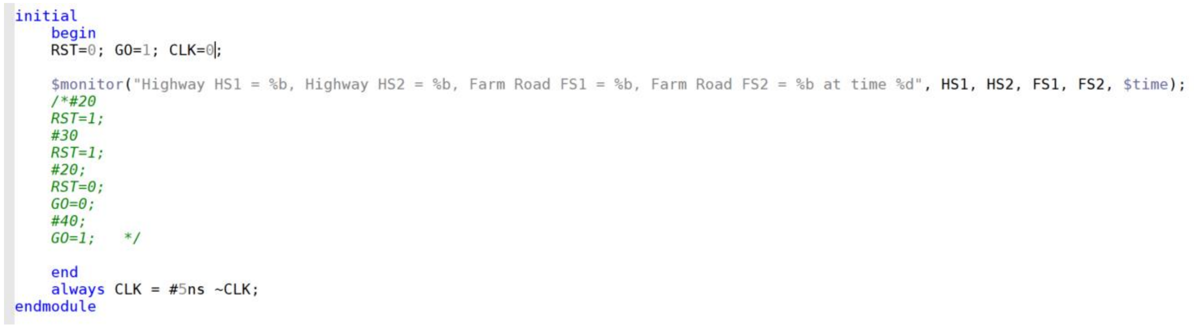
This module tests the effectiveness of each module by calling them in a structural method then set a duration of time and change the inputs in order to observe how outputs differ. As it is seen from the below Figure, the previous modules are called and the reg and wire values are assigned to each one where the outputs of traffic light controller and generator are both inputs to the analyzer module.



```verilog
module Traffic_Light_Controller_tb;
reg RST, GO, CLK;
wire [1:0] HS1, HS2, FS1, FS2, HS1EXP, HS2EXP, FS1EXP, FS2EXP;
wire ERROR;

Traffic_Light_Controller TLC (RST, CLK, GO, HS1, HS2, FS1, FS2);
Traffic_Light_Controller_Generator TLCG (RST, CLK, GO, HS1EXP, HS2EXP, FS1EXP, FS2EXP);
Traffic_Light_Controller_Analyzer TLCA (RST, CLK, GO, HS1, HS2, FS1, FS2, HS1EXP, HS2EXP, FS1EXP, FS2EXP, ERROR);
```

Figure13: Test bench code

As seen from the below figure, I scheduled the values to work normally without changing any input for 20s then after various timings I changed the RST value to 1 where the outputs should be as state s0, then changed GO to check if the outputs freeze at the state.



```verilog
initial
    begin
    RST=0; GO=1; CLK=0;

    $monitor("Highway HS1 = %b, Highway HS2 = %b, Farm Road FS1 = %b, Farm Road FS2 = %b at time %d", HS1, HS2, FS1, FS2, $time);
    /*#20
    RST=1;
    #30
    RST=1;
    #20;
    RST=0;
    GO=0;
    #40;
    GO=1;    */

    end
    always CLK = #5ns ~CLK;
endmodule
```

Figure14: Test Bench TIMING

At first, I commented the edits in order to check the efficiency of the modules and compare their outputs then I removed the comment in order to obtain the different outputs how they are changing.

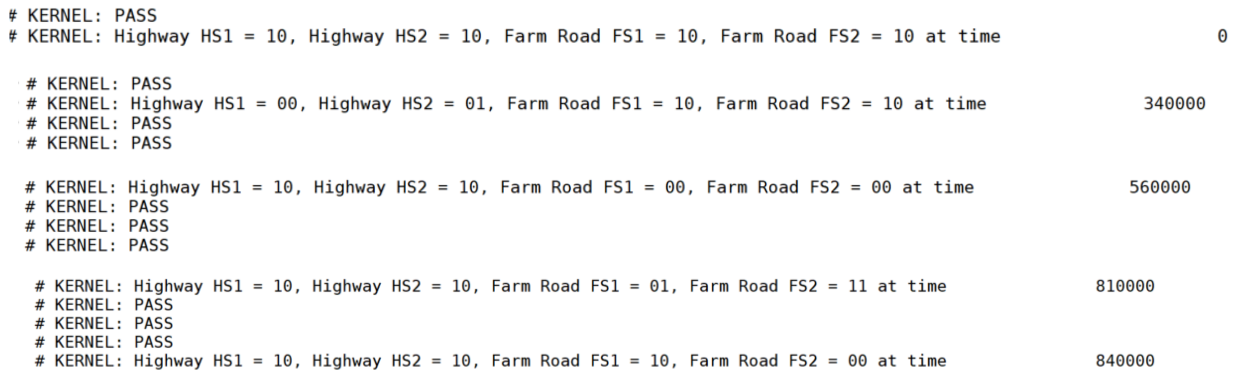
3- Results:

Obtaining the results through the waveform was a little complex since the program is synchronized and triggered based on the positive edge of the clock and different delays assigned.

a- Compile Results:

These screenshots are results that are displayed based on the changing of the states, at time 0 then state is 0 and outputs are 2'b10 (RED), another result is state 3 (GREEN YELLOW RED RED) after 33 seconds, third one is state 8 (RED RED GREEN GREEN) after 50 seconds, fourth one is state 11 (RED RED YELLOW REDYELLOW) after 72 seconds, and state 12 (RED RED RED GREEN) after 74 seconds.

There can be seen a difference in time intervals as a result of changing the RST and GO previously which affected the delays of each state.



```
# KERNEL: PASS
# KERNEL: Highway HS1 = 10, Highway HS2 = 10, Farm Road FS1 = 10, Farm Road FS2 = 10 at time                    0

  # KERNEL: PASS
  # KERNEL: Highway HS1 = 00, Highway HS2 = 01, Farm Road FS1 = 10, Farm Road FS2 = 10 at time              340000
  # KERNEL: PASS
  # KERNEL: PASS

  # KERNEL: Highway HS1 = 10, Highway HS2 = 10, Farm Road FS1 = 00, Farm Road FS2 = 00 at time              560000
  # KERNEL: PASS
  # KERNEL: PASS
  # KERNEL: PASS

   # KERNEL: Highway HS1 = 10, Highway HS2 = 10, Farm Road FS1 = 01, Farm Road FS2 = 11 at time             810000
   # KERNEL: PASS
   # KERNEL: PASS
   # KERNEL: PASS
   # KERNEL: Highway HS1 = 10, Highway HS2 = 10, Farm Road FS1 = 10, Farm Road FS2 = 00 at time             840000
```

Figure15: Figures of results displayed

b- Waveform Results:

At first, the results are normally examined, this can be seen when the state s0 moves to state s1 in Figure16 and moving from state s1 to state s2 in figure17 and compared the expected with the outputs (based on the positive edge of the clock).

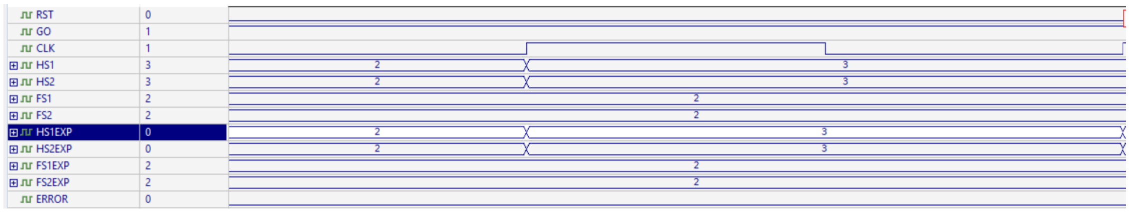


Figure16: Waveform 1

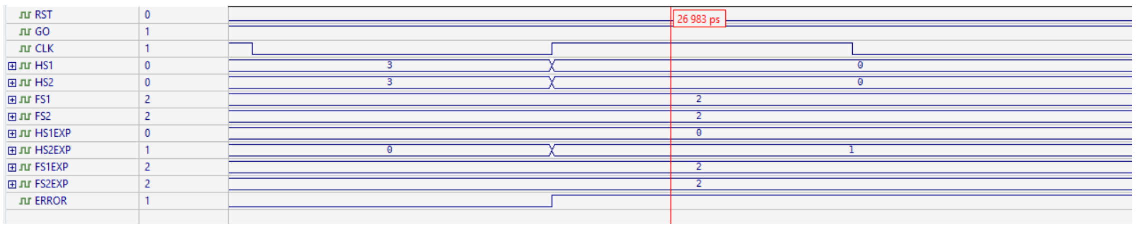


Figure17: Waveform 2

Since the test bench had some editings on the RST and GO values this can be seen in the below figure of the waveform, the error bit wasn't resulting a correct result even when the values are different it didn't convert to 1, but when GO was 0 then the state is frozen which is seen when the values stayed at (10) RED and then after GO was raised to 1, the outputs changed to (11 11 10 10) REDYELLOW REDYELLOW RED RED (from state 0 to state 1) but the results weren't matching so error was 1.

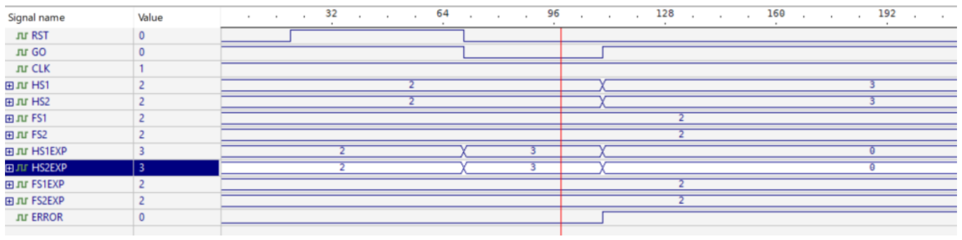


Figure18: Waveform 3

Conclusion

This project has helped me improve my skills using Verilog HDL coding and implement it with different states (state diagram) and build a way to move across one another by synchronizing the clock and arrange a limit for each one to give a certain output and check the results if they are matching with the expected ones or not.

This project has some issues especially by saving the expected data and the comparing process between them since the expected data was given only based on the positive edge of the clock and as a result the timing of each state was different from the time data was saved, and therefore the effectivity and validity of the outputs would be affected. If I had more time, I would think of a better way to implement the counter independently and develop another method to save the data based on the counter and compare them in order to give the right answers.