TRAFFIC LIGHT CONTROL SYSTEM USING VHDL PROGRAMMING

KELLY ATIENO DIANG'A

12/12/2021

Table of Contents

Table of Figures

Table of Tables

PROBLEM SOLUTION DESCRIPTION

1. INTRODUCTION

To solve the above process,I used system thinking.I went through the problem description,making sure to highlight the requirements as well as the key points. I then went on to design a state transition diagram as per the problem.From the state diagram I was then able to determin what I would need to achieve the state transitions.

2. REQUIREMENTS

Input Sync Memory – Does the input synchronization and remembers when the pedestrian button has been pressed.

Counter – A counter with multiple outputs to represent various time delays. The counter would have a synchronous clear input to re-start the time interval.LED Display was used to show the different time delays for each state in my simulation.

State Machine – Provides the 'intelligence' of the system.
Controls:

- The Timer (Mealy-style outputs)
- The Lights (Moore-style outputs)

Responds to

- The Timer outputs
- The Car buttons (synchronised).
- The Pedestrian buttons (synchronised & registered).

3. CODING PROCESS

I created 3 modules. One for Input Sync Memory, another for the state machine and the final one is the Traffic Module. There are 10 inputs for the Input Synchronisation Memory,that is; CarEW, CarNS, PedEW, PedNS, CarEWClear, CarNSClear, PedEWClear, PedNSClear, clock and Reset button. Output from this module which includes Synchronised CarEW,andCarNS, and PedEW and PedNS memory is then used as input for the state machine. The state machine then provides 7 outputs and out of the 7,4 are used as inputs in the Input Sync Memory. These are; CarEWClear, CarNSClear, PedEWClear, PedNSClear which are used to clear input once the requirements have been met by the different states in the State Machine. The Traffic module combines both the Input Sync Memory and the State Machine. It has 6 inputs as per the problem description.These are CarEW, CarNS, PedEW and PedNS, clock and Reset button which are fed into to the input Sync Memory. This module also provides the final output consisting of Synchronised CarEW,andCarNS, and PedEW and PedNS memory used to analyse the simulation.It also produces the LED for the counter as well as the lights for the EW and NS traffic. Pedestrian lights are contatined in the traffic lights as will be shown in the simulations ater on.

4. SIMULATION

I then produced a test bench to simulate my module. Just to note; I used 2 bit binary for the different traffic lights;00-Red, 01-Amber, 10-Green, 11-Both Pedestrian and Traffic lights are green. To ensure the code was as per the requirements, I tested different scenarios; Traffic in the NS, Traffic in the EW,Pedestrian in the NS,Pedestrian in the EW and traffic amd Pedestrians in both directions.The sections below show the detailed description of the above process.

STATE TRANSITION DIAGRAM

The following Figure shows the state transition diagram for the traffic and Pedestrain lights as well as the time delays in each state.t also shows the number of counts it takes before transitioning to the next state.

1. DIAGRAM

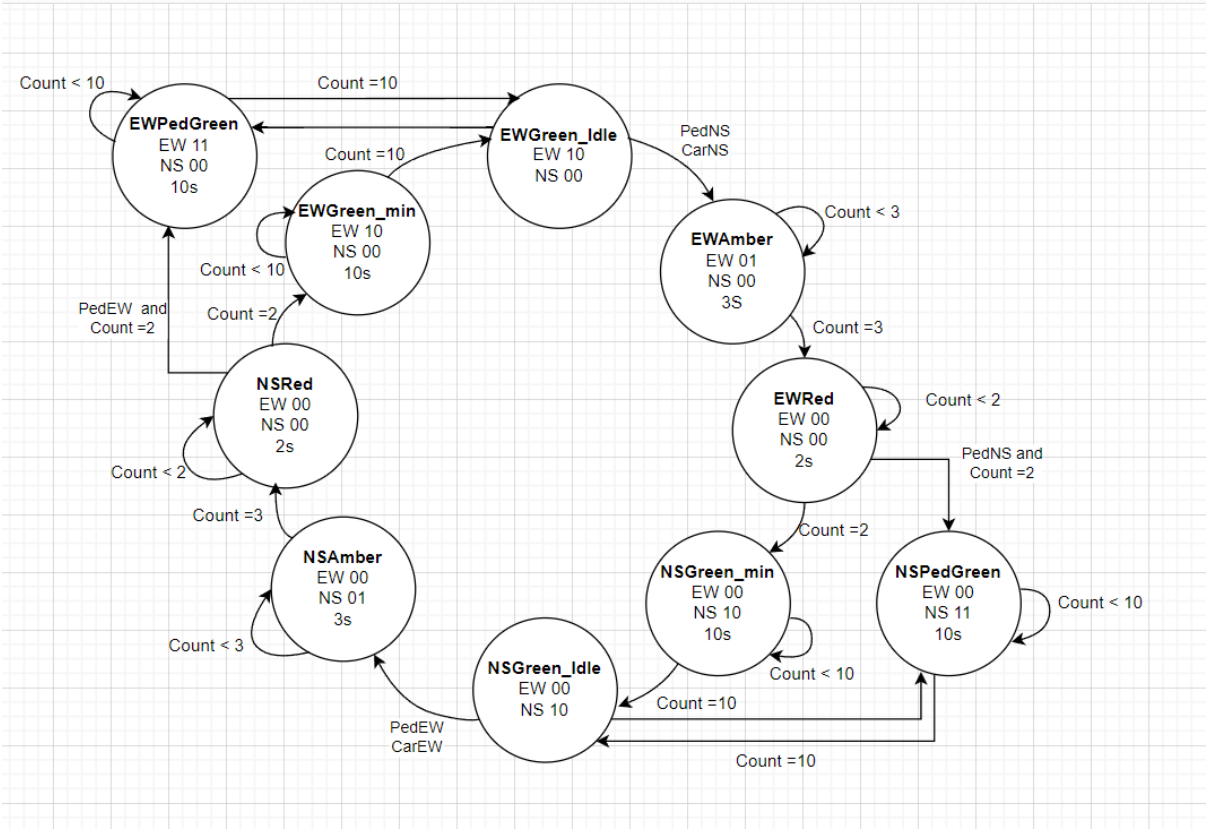

Figure 1: StateTransition diagram

2. EXPLANATION

1) EWGreen_Idle

The Traffic light is initially green at the EW traffic light and Red at the NS lane. If no car or pedestrian is detected in the NS lane and a pedestrian is detected in the EW lane,then it will move to the EWPedGreen state,stay there for 10 seconds to serve the pedestrian before returning to the EWGreen_Idle state.If a car or pedestrian is detected in the NS lane, then it will move to EWAmber state. If a pedestrian is detected in the EW lanee and a car or pedestrian has been detected in the NS lane, then it will serve the pedestrian or car from the NS lane before cycling back to serve the pedestrian in the EW lane.Therefore the next staete will be EWAmber.

2) EWPedGreen

In this state,the traffic light is Green at EW lane and Red in the NS lane.The count is cleared to 0.the timer then starts to count to 10 seconds.Once it reaches 10,it moves to the next state which is EWGreen_Idle.

3) EWAmber

At this state,the traffic light is Amber at the EW lane and Red at the NS lane.The count is cleared to 0.the timer then starts to count to 3 seconds.Once it reaches 3,it moves to the next state which is EWRed.

4) **EWRed**

At this state,the traffic light is Red at the EW lane and Red at the NS lane .The count is cleared to 0.the timer then starts to count to 2 seconds.If a Pedestrian was detected,it will move to NSPedGreen once it reaches 2 seconds. If a car was detected,it will move to NSGreen_min state once the counter reaches 2 seconds.

5) **NSGreen_min**

At this tate,the traffic light is red at the EW lane and Green at the NS lane.The timer is set to 0 and the counter starts counting till 10 seconds.at 10 seconds it moves to the next state which is the NSGreen_Idle.

6) **NSGreen_Idle**

The Traffic light is green at the NS traffic light and Red at the EW lane. If no car or pedestrian is detected in the EW lane and a pedestrian is detected in the NS lane,then it will move to the NSPedGreen state,stay there for 10 seconds to serve the pedestrian before returning to the NSGreen_Idle state.If a car or pedestrian is detected in the EW lane, then it will move to NSAmber state. If a pedestrian is detected in the NS lane and a car or pedestrian has been detected in the EW lane, then it will serve the pedestrian or car from the EW lane before cycling back to serve the pedestrian in the NS lane.Therefore the next staete will be NSAmber.

7) **NSPedGreen**

In this state,the traffic light is Red at EW lane and Green in the NS traffic and pedestrian lane.The count is cleared to 0.the timer then starts to count to 10 seconds.Once it reaches 10,it moves to the next state which is NSGreen_Idle.

8) **NSAmber**

At this state,the traffic light is Amber at the NS lane and Red at the EW lane.The count is cleared to 0.the timer then starts to count to 3 seconds.Once it reaches 3,it moves to the next state which is NSRed.

9) **NSRed**

At this state,the traffic light is Red at the NS lane and Red at the EW lane .The count is cleared to 0.the timer then starts to count to 2 seconds.If a Pedestrian was detected,it will move to EWPedGreen once it reaches 2 seconds. If a car was detected,it will move to EWGreen_min state once the counter reaches 2 seconds.

10) **EWGreen_min**

At this tate,the traffic light is red at the NS lane and Green at the EW lane.The timer is set to 0 and the counter starts counting till 10 seconds.at 10 seconds it moves to the next state which is the EWGreen_Idle.

BLOCK DIAGRAM

The diagram below shows the Top level structure-Traffic Module, and in it theres the Input Sync Memory and State machine and both are interconnected.The figure also shows the signal names as well as the port names.

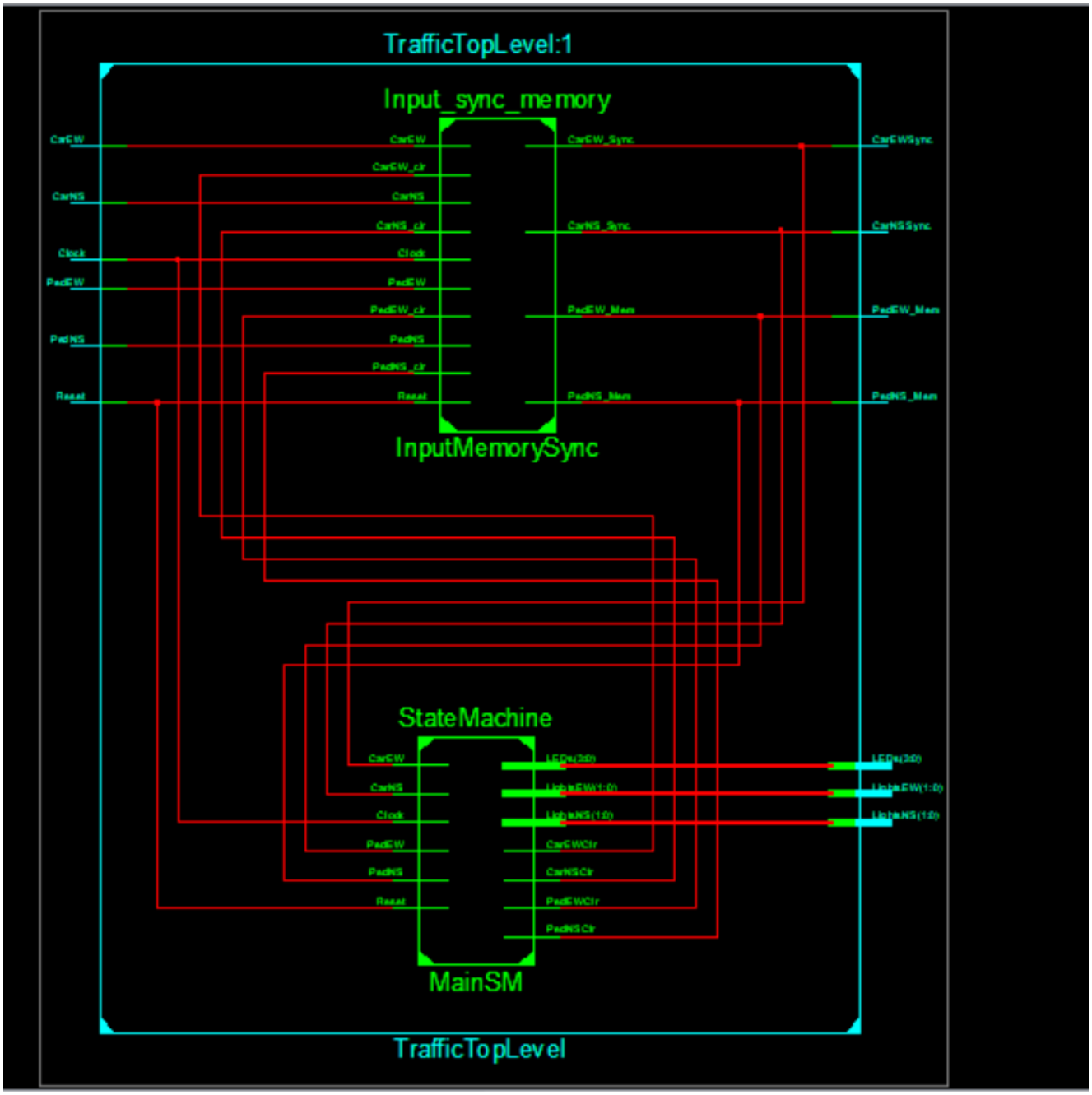

Figure 2: Top level block diagram

MODULE DESCRIPTION WITH VHDL CODES

INPUT MEMORY SYNC MODULE

1. INPUT SYNC MEMORY MODULE DIAGRAM

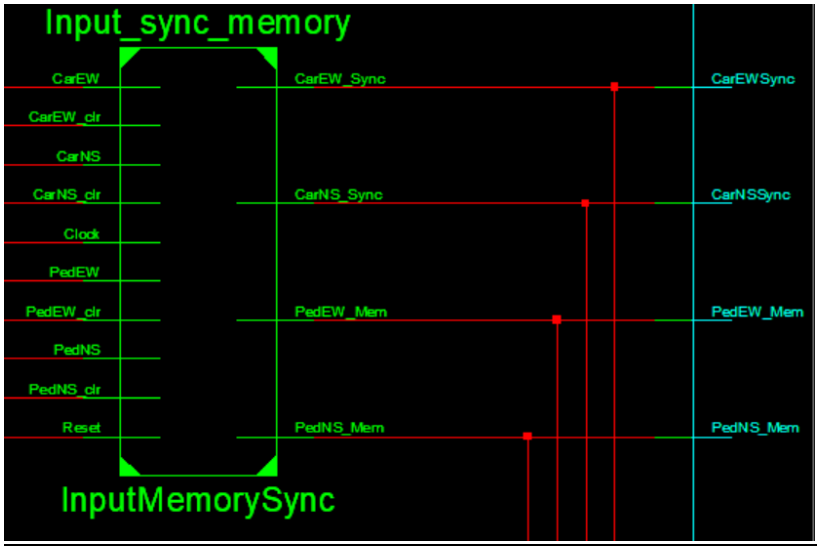

Figure 3: Input Sync Memory Module

2. MODULE DESCRIPTION

This module basically does the input synchronization and remembers when the Pedestrian button has been pressed. It has 10 inputs for the Input Synchronisation Memory. That is; CarEW, CarNS, PedEW, PedNS, CarEWClear, CarNSClear, PedEWClear, PedNSClear, clock and Reset button. It has 4 outputs which include Synchronised CarEW and CarNS, and PedEW and PedNS memory. The values for the PedEWClear, PedNSClear, CarEWClear and CarNSClear are obtained from the respective outputs of the State Machine Module.

To describle this module, I have divided it into 2 sections; the traffic lights and the pedestrian light.

1) Traffic Light control

Inputs used here are CarEW, CarNS, clock,reset, CarEWClear and CarNSClear. To synchronise the car inputs to the clock, a synchronise process is used.The inputs are initially set to 0.

Once the inputs has been recorded for CarEW and CarNS,they are synchronised to the rising edge of the clock as shown below.

```
elsif rising_edge(Clock)then --Synchronize to the clock(only output at rising edge of the clock)
      if CarEW = '1' then
         CarEW_Sync <= '1';
      end if;

      if CarNS = '1' then
         CarNS_Sync <= '1';
      end if;
```

The synchronised inputs are then stored in the CarEW_Sync and the CarNS_Sync.These will stay high until the green light changes to amber. This is because it is assumed that the light sensors remain activated as the car stays at the stop sign and will only be deactivated once the light changes to

amber,showing car has already passed and another car has been detected in the next lane. To ensure that this input cleared once the car has passed,CarEWClear and CarNSClear are used. When CarEWClear is high(1) it clears the input in the CarEW_Sync and when CarNSClear is high(1) it clears the input in the CarNS_Sync.This is so as to allow for traffic and pedestrians in the other lane to be catered to.

This is done using the following lines of codes;

```
if CarEW_clr ='1' then
   CarEW_Sync <= '0';
end if;

if CarNS_clr ='1' then
   CarNS_Sync <= '0';
end if;
```

If a car is detected in either lanes,then the CarEW and CarNS is set to high(1) and the above processs is repeated for the traffic.

Output for this part is the CarEW_Sync and CarNSSync which are used as inputs in the state machine module.

2) Pedestrian Light control

Inputs used here are the PedEW, PedNS, PedEWClear, PedNSClear, clock and Reset button. As the pedestrian is briefly pressed and released,the traffic control needs to remember this and forget this memory once the pedestrian has been catered to. A latch therefore needs to be used.

The input for Pedestrian button for both lanes are initially set to zero. Once the pedestrian button has been pressed briefly,this input is recorded in the PedEW and PedNS buttons with respect to the lane.This is then stored in the PedNS_Memory and PedEW_Memory so that when the button is released,these two stay high whereas the PedEW and PedNS inputs become low.This done using the following lines of codes.

```
if PedEW = '1' then
  PedEW_Mem <= '1';
end if;

if PedNS = '1' then
   PedNS_Mem <= '1';
end if;
```

This memory needs to be cleared once the Pedestrian has been catered for.Therefore PedEWClear and PedNSClear inputs are used for this. When PedEWClear or PedNSClear is high, PedEW_Memory and PedEW_Memory are cleared and becomes low at rising edge of the clock as per the clear input.This is done using the following lines of codes.

```
if PedEW_clr ='1' then
   PedEW_Mem <= '0';
end if;

if PedNS_clr ='1' then
   PedNS_Mem <= '0';
end if;
```

The output from this section are the PedEW_Memory and PedEW_Memory which will be used as inputs in the State Machine.

3. VHDL CODES FOR INPUT SYNC MEMORY MODULE

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: KELLY DIANGÁ
-- Tool versions:
-- Description: To remember pedestrian button press and to synchronize the inputs

----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Input_sync_memory is
    Port ( Reset : in  STD_LOGIC;
           Clock : in  STD_LOGIC;
           CarEW : in  STD_LOGIC;
           CarNS : in  STD_LOGIC;
           PedEW : in  STD_LOGIC;
           PedNS : in  STD_LOGIC;
           CarEW_Sync : out  STD_LOGIC;
           CarNS_Sync : out  STD_LOGIC;
           PedEW_Mem : out  STD_LOGIC;
           PedNS_Mem : out  STD_LOGIC;
           PedEW_clr : in  STD_LOGIC;
           PedNS_clr : in  STD_LOGIC;
           CarEW_clr : in  STD_LOGIC;
           CarNS_clr : in  STD_LOGIC
           );
end Input_sync_memory;

architecture Behavioral of Input_sync_memory is

begin

  process(Reset,Clock,CarEW,CarNS,PedEW,PedNS,PedEW_clr,PedNS_clr)
  begin
      if Reset = '1' then --Set the initial values to zero
         CarEW_Sync <= '0';
         CarNS_Sync <= '0';
         PedEW_Mem <= '0';
         PedNS_Mem <= '0';

      elsif rising_edge(Clock)then --Synchronize to the clock(only output at rising edge of the clock)
           if CarEW = '1' then   --Store value of CarEW  to carEW_Sync
              CarEW_Sync <= '1';
           end if;

           if CarNS = '1' then   --Store value of CarNS  to carNS_Sync
              CarNS_Sync <= '1';
           end if;

           if CarEW_clr ='1' then --Clear the value of CarEW_Sync to 0
              CarEW_Sync <= '0';
           end if;

           if CarNS_clr ='1' then --Clear the value of CarNS_Sync to 0
              CarNS_Sync <= '0';
           end if;

           if PedEW = '1' then  --Store value of PedEW  to PedEW_Mem once the button has been pressed briefly
             PedEW_Mem <= '1';
           end if;

           if PedNS = '1' then   --Store value of PedNS  to PedNS_Mem once the button has been pressed briefly
              PedNS_Mem <= '1';
           end if;

           if PedEW_clr ='1' then --Clear the value of PedEW_Mem to 0
              PedEW_Mem <= '0';
           end if;

           if PedNS_clr ='1' then  --Clear the value of PedNS_Mem to 0
              PedNS_Mem <= '0';
           end if;
      end if;


   end process;


end Behavioral;
```

STATE MACHINE MODULE

1. STATE MACHINE MODULE DIAGRAM

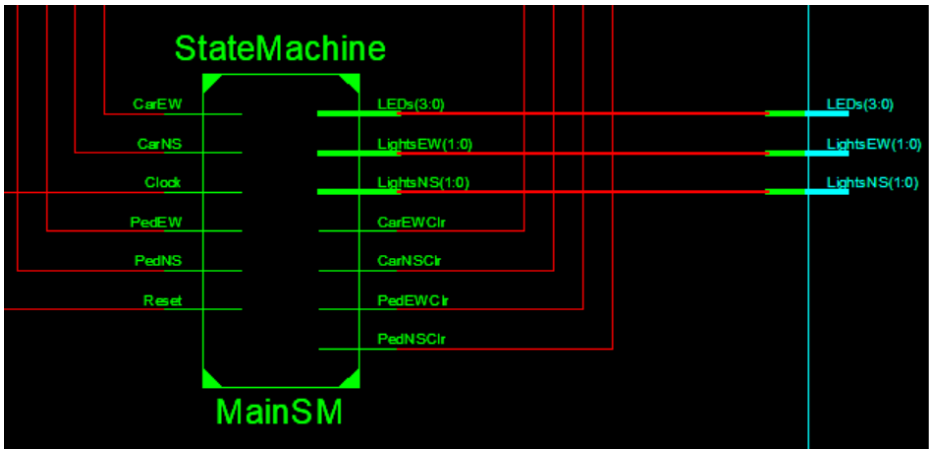

Figure 4: State Machine Module

2. MODULE DESCRIPTION

This is basically the brain of the system. It responds to the timer(counter) outputs, the Car buttons (synchronised) and the Pedestrian buttons (synchronised & registered) and changess the traffic lights respectively. The inputs for this module are CarEW, CarNS,CLOCK,PedEW,PedNS and Reset. The values for CarEW, CarNS, PedEW and PedNS are obtained from the Input Sync Memory module. The output producesd are LEDs, LightsEW, LightsNS, CarEWClr ,CarNSClr, PedEWClr and PedNSClr. , CarEWClr ,CarNSClr, PedEWClr and PedNSClr produce the values that will be used in the Input sync Memory module as mentioned earlier. LEDs output will display the time delays for each state in the simulation. LightsEW displays the traffic lights for the EW lane while the LightsNS displays the traffic lights in the NS lane.

1) COUNTER

The following screenshot shows the 4 bit signal used for the counter as well as the count of 2seconds, 3seconds and 10 seconds as indicated in the state transition diagram. These will be used well transitioning from state to state based on the timer(counter).

```
signal count:STD_LOGIC_VECTOR(3 downto 0);

constant COUNT2:STD_LOGIC_VECTOR(3 downto 0) :="0010";
constant COUNT3:STD_LOGIC_VECTOR(3 downto 0) :="0011";
constant COUNT10:STD_LOGIC_VECTOR(3 downto 0) :="1010";
```

2) STATE TRANSITION

To transition from state to state, a case statement is used. This is a synchronised Process, i.e the states will transition at the rising edge of the clock.

3) OUTPUT-LEDS AND EW,NS TRAFFIC LIGHTS.

For the outputs, a combinational process is used. This process does not depend on the clock. Acase statement is used in this process to specify the light to be displayed for each state as well as the count of the time delays.

3. VHDL CODES FOR STATE MACHINE MODULE

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  entity StateMachine is
26      Port ( Clock : in  STD_LOGIC;
27             Reset : in  STD_LOGIC;
28
29            -- Car and pedestrian buttons
30             CarEW : in STD_LOGIC; -- Car on EW road
31             CarNS : in STD_LOGIC; -- Car on NS road
32             PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
33             PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing EW road)
34
35             -- Light control
36             LightsEW : out STD_LOGIC_VECTOR (1 downto 0); -- controls EW lights
37             LightsNS : out STD_LOGIC_VECTOR (1 downto 0); -- controls NS lights
38             LEDs     : out STD_LOGIC_VECTOR (3 downto 0);   -- controls LED Lights
39             PedEWClr : out STD_LOGIC;
40             PedNSClr : out STD_LOGIC;
41             CarEWClr : out STD_LOGIC;
42             CarNSClr : out STD_LOGIC
43             );
44  end StateMachine;
45
46  architecture Behavioral of StateMachine is
47
48  type StateType is (EWGreenIdle, EWAmber, EWRed, NSGreenMin, NSPedGreen, NSGreenIdle, NSAmber, NSRed, EWGreenMin, EWPedGreen);
49
50  signal state : StateType;
51
52  --Set the different time delays
53  signal count:STD_LOGIC_VECTOR(3 downto 0);--counter
54  constant COUNT2:STD_LOGIC_VECTOR(3 downto 0) :="0010";--2 seconds
55  constant COUNT3:STD_LOGIC_VECTOR(3 downto 0) :="0011"; --3 seconds
56  constant COUNT10:STD_LOGIC_VECTOR(3 downto 0) :="1010";--10 seconds

58  signal PedEWClear,PedNSClear,CarEWClear,CarNSClear : STD_LOGIC;
59
60  begin
61
62
63   SynchronousProcess:
64   process (reset, clock,carNS,carEW,PedNS,PedEW) --Ensure the inputs synchronise with the clock
65   begin
66      if (reset = '1') then --Set values to 0
67          state <= EWGreenIdle;
68          count <= X"0";
69          PedEWClear <= '0';
70          PedNSClear <= '0';
71          CarEWClear <= '0';
72          CarNSClear <= '0';
73
74      elsif (rising_edge(clock)) then --States will transition at the rising edge of the clock
75         case state is
76            when EWGreenIdle =>                -- EWGreenIdle state
77                 if ((carNS = '1')OR (PedNS ='1')) then
78                     state<= EWAmber;
79                     count <= X"0";
80                     CarEWClear <= '0';
81
82                 elsif (PedEW = '1' and ((carNS = '0') and (PedNS = '0'))) then
83                     state<= EWPedGreen;
84                     count <= X"0";
85                     CarEWClear <= '0';
86                 else
87                     state<=EWGreenIdle;
88
89                 end if;
90
91
92             when EWPedGreen =>             -- EWPedGreen State
93                 if (count < COUNT10) then
94                     state<= EWPedGreen;
95                     count <= count + 1;
```

```
                PedEWClear <='1';
            else
                state<=EWGreenIdle;
                count <= X"0";
                PedEWClear <='0';
            end if;

        when EWAmber =>                 -- EWAmber State
            if (count < COUNT3) then
                state<= EWAmber;
                count <= count + 1;
            else
                state<=EWRed;
                 count <= X"0";
            end if;

        when EWRed =>                   -- EWRed State
            if (count < COUNT2) then
                state<= EWRed;
                count <= count + 1;
            else
                if(PedNS ='1') then
                state<=NSPedGreen;
                count <= X"0";

                else
                state<=NSGreenMin;
                count <= X"0";
                end if;
            end if;

         when NSGreenMin =>             --NSGreenMin State
            if (count < COUNT10) then
                state<= NSGreenMin;
                count <= count + 1;
                CarNSClear <= '1';


            else
                state<=NSGreenIdle;
                count <= X"0";
            end if;

        when NSGreenIdle =>           --NSGreenIdle State
          if ((carEW = '1')OR (PedEW ='1')) then
                state<= NSAmber;
                count <= X"0";
                CarNSClear <= '0';

            elsif (PedNS = '1' and ((carEW = '0') and (PedEW = '0'))) then
                state<= NSPedGreen;
                count <= X"0";
                CarNSClear <= '0';

            else
                state<=NSGreenIdle;

            end if;


        when NSPedGreen =>              --NSPedGreen  State
            if (count < COUNT10) then
                state<= NSPedGreen;
                count <= count + 1;
                PedNSClear <= '1';
            else
                state<=NSGreenIdle;
                count <= X"0";
                PedNSClear <= '0';
            end if;

        when NSAmber =>                 --NSAmber State
            if (count < COUNT3) then
                state<= NSAmber;
                count <= count + 1;
            else
```

```
                    state<=NSRed;
                    count <= X"0";
                end if;

            when NSRed =>                    --NSRed State
                if (count < COUNT2) then
                    state<= NSRed;
                    count <= count + 1;
                else
                    if(PedEW ='1') then
                    state<=EWPedGreen;
                    count <= X"0";

                    else
                    state<=EWGreenMin;
                    count <= X"0";
                    end if;
                end if;

             when EWGreenMin =>              --EWGreenMin State
                if (count < COUNT10) then
                    state<= EWGreenMin;
                    count <= count + 1;
                    CarEWClear <= '1';
                else
                    state<=EWGreenIdle;
                    count <= X"0";
                end if;

             when others =>
             state<=EWGreenIdle;

             end case;
    end if;
end process SynchronousProcess;

--OUTPUTS------
```

```
CombinationalProcess: --Not dependant on clock
process(state)
begin

  case state is

    when EWGreenIdle =>
         LightsEW <= "10";
         LightsNS <= "00";

    when EWPedGreen =>
         LightsEW <= "11";
         LightsNS <= "00";

    when EWAmber =>
         LightsEW <= "01";
         LightsNS <= "00";

    when EWRed =>
         LightsEW <= "00";
         LightsNS <= "00";

    when NSGreenMin =>
         LightsEW <= "00";
         LightsNS <= "10";

    when NSGreenIdle =>
         LightsEW <= "00";
         LightsNS <= "10";

  when NSPedGreen =>
         LightsEW <= "00";
         LightsNS <= "11";

    when NSAmber =>
         LightsEW <= "00";
         LightsNS <= "01";
```

```
245
246         when NSRed =>
247              LightsEW <= "00";
248              LightsNS <= "00";
249
250         when EWGreenMin =>
251              LightsEW <= "10";
252              LightsNS <= "00";
253
254         when others =>
255              LightsEW <="10";
256              LightsNS <="00";
257       end case;
258    end process CombinationalProcess;
259
260    LEDs <=count; --OUTPUT FOR COUNTER
261
262    --OUTPUT TO BE USED BY INPUT SYNC MEMORY MODULE
263    PedNSClr <= PedNSClear;
264    PedEWClr <= PedEWClear;
265    CarNSClr <= CarNSClear;
266    CarEWClr <= CarEWClear;
267
268 end Behavioral;
```

TRAFFIC TOP LEVEL MODULE

1. TRAFFIC TOP LEVEL MODULE DIAGRAM

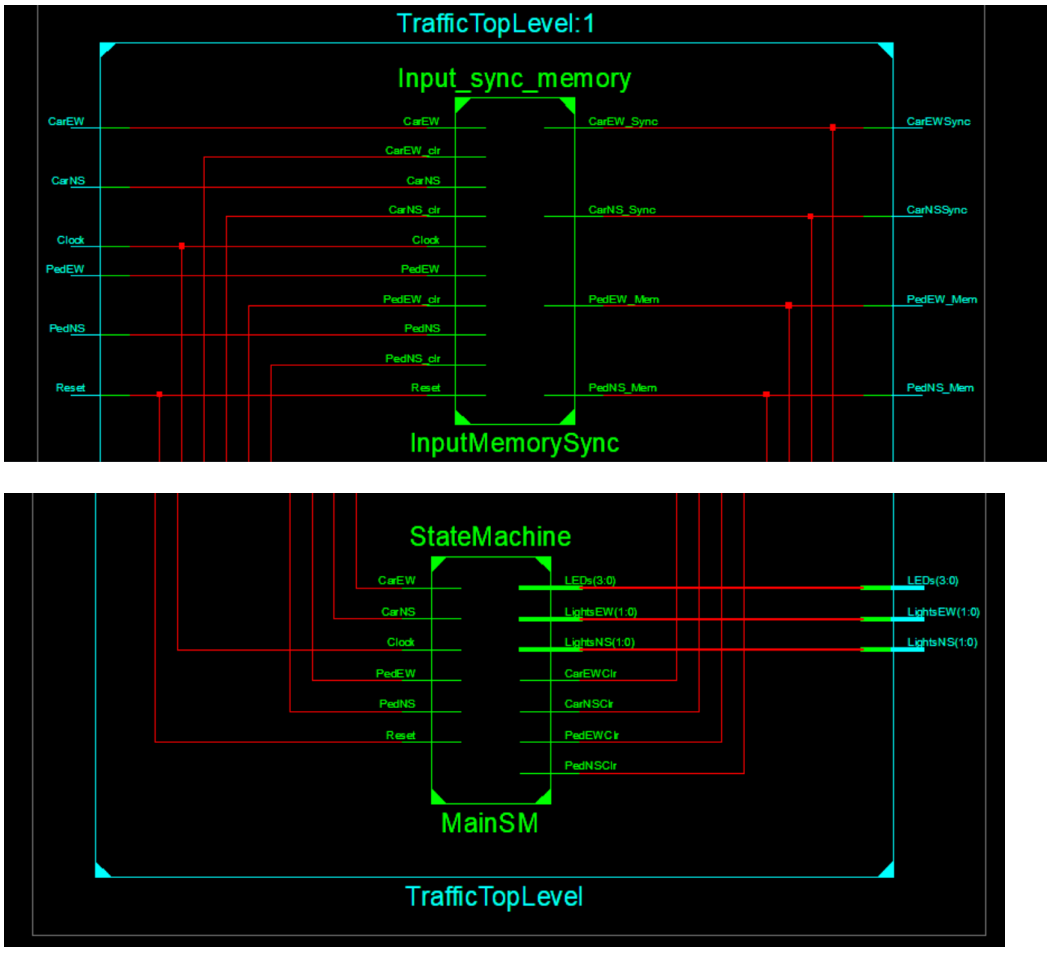

Figure 5: Traffic Top Level Module

2. MODULE DESCRIPTION

This module provides the combination of bothe the Input Sync Memory and State machine module to provide the final traffic lights control Module. This module only has 6 inputs as per the problem description. These are CarEW, CarNS, Clock, PedEW, PedNS and Reset. The clock and Reset input are both directly used in the input Sync Memory and State Machine module. The remaining inputs are only directly used in the Input Sync Memory.

3. VHDL CODES FOR TRAFFIC TOP LEVEL MODULE

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TrafficTopLevel is
    Port ( Clock : in  STD_LOGIC;
           Reset : in  STD_LOGIC;

          -- Car and pedestrian buttons
            CarEW : in STD_LOGIC; -- Car on EW road
            CarNS : in STD_LOGIC; -- Car on NS road
            PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
            PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing EW road)

            -- Light control
            LightsEW : out STD_LOGIC_VECTOR (1 downto 0); -- controls EW lights
            LightsNS : out STD_LOGIC_VECTOR (1 downto 0); -- controls NS lights
            CarEWSync : out STD_LOGIC; -- Car on EW road
            CarNSSync : out STD_LOGIC; -- Car on NS road
            PedEW_Mem : out STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
            PedNS_Mem : out STD_LOGIC; -- Pedestrian moving NS (crossing EW road)
            LEDs      : out STD_LOGIC_VECTOR (3 downto 0)    -- controls LED Lights

            );
end TrafficTopLevel;

 architecture Behavioral of TrafficTopLevel is

--COMBINE INPUT SYNC MEMORY MODULE AND STATE MACHINE MODULE IN THE TRAFFIC TTOP LEVEL MODULE
    COMPONENT Input_sync_memory
    PORT(
         Reset : IN  std_logic;
         Clock : IN  std_logic;
         CarEW : IN  std_logic;
         CarNS : IN  std_logic;
         PedEW : IN  std_logic;
```

```
         PedEW : IN  std_logic;
         PedNS : IN  std_logic;
         CarEW_Sync : OUT  std_logic;--Provide value to be used by CarEW signal in State Machine module
         CarNS_Sync : OUT  std_logic;--Provide value to be used by CarNS signal in State Machine module
         PedEW_Mem : OUT  std_logic; --Provide value to be used by PedEW signal in State Machine module
         PedNS_Mem : OUT  std_logic; --Provide value to be used by PedNS signal in State Machine module
         PedEW_clr : IN  std_logic;  --Clear PedEW Memory
         PedNS_clr : IN  std_logic;  --Clear PedNS Memory
         CarEW_clr : in  STD_LOGIC;  --Clear EW traffic
         CarNS_clr : in  STD_LOGIC   --Clear EW traffic
        );
    END COMPONENT;


    COMPONENT StateMachine
    PORT(
         Clock : IN  std_logic;
         Reset : IN  std_logic;
         CarEW : IN  std_logic;
         CarNS : IN  std_logic;
         PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
         PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing EW road)

         LightsEW : OUT  std_logic_vector(1 downto 0);
         LightsNS : OUT  std_logic_vector(1 downto 0);
         LEDs     : out STD_LOGIC_VECTOR (3 downto 0);   -- controls LED Lights
         PedEWClr : out STD_LOGIC; --Provide value to be used by PedEW_clr clear signal in input sync Memory module
         PedNSClr : out STD_LOGIC; --Provide value to be used by PedNS_clr clear signal in input sync Memory module
         CarEWClr : out STD_LOGIC; --Provide value to be used by CarEW_clr clear signal in input sync Memory module
         CarNSClr : out STD_LOGIC  --Provide value to be used by CarNS_clr clear signal in input sync Memory module
        );
    END COMPONENT;
--Wires to be used by components to transfer signals
signal CarEWshadow,CarNSshadow,PedEWshadow,PedNSshadow: STD_LOGIC;
signal PedEWClear,PedNSClear,CarEWClear,CarNSClear : STD_LOGIC;
begin

```

```
93  --COMBINING THE COMPONENT WIRES
94     InputMemorySync: Input_sync_memory PORT MAP (
95                      Reset => Reset,
96                      Clock => Clock,
97                      CarEW => CarEW,
98                      CarNS => CarNS,
99                      PedEW => PedEW,
100                     PedNS => PedNS,
101                     CarEW_Sync => CarEWshadow,
102                     CarNS_Sync => CarNSshadow,
103                     PedEW_Mem => PedEWshadow,
104                     PedNS_Mem => PedNSshadow,
105                     PedEW_clr => PedEWClear,
106                     PedNS_clr => PedNSClear,
107                     CarEW_clr => CarEWClear,
108                     CarNS_clr => CarNSClear
109         );
110
111         MainSM: StateMachine PORT MAP (
112                     Clock => Clock,
113                     Reset => Reset,
114                     CarEW => CarEWShadow,
115                     CarNS => CarNSshadow,
116                     PedEW => PedEWshadow,
117                     PedNS => PedNSshadow,
118                     LightsEW => LightsEW,
119                     LightsNS => LightsNS,
120                     LEDs =>LEDs,
121                     PedEWClr => PedEWClear,
122                     PedNSClr => PedNSClear,
123                     CarEWClr => CarEWClear,
124                     CarNSClr => CarNSClear
125         );
126
127  --OUTPUTS AS DISPLAYED IN SIMULATION
128 CarEWSync <= CarEWshadow;
129 CarNSSync <= CarNSshadow;
130 PedEW_Mem <= PedEWshadow;
131 PedNS_Mem <= PedNSshadow;
132 end Behavioral;
```

TESTING DESCRIPTION

In this section I am going to take you through the testing phase of my codes.This includes the test bench codes as well as the simulation results.

INPUT SYNC MEMORY MODULE

1. TESTBENCH CODES

AIMS AND EXPECTED RESULTS

The following table shows the aims and expected results for the following test bench.

AIM	EXPECTED RESULTS
1. Check if the CarEW_Sync and CarNS_Sync are able to synchronise with the clock.	• Values of CarEW_Sync and CarNS_Sync should change at the rising edge of the clock. The values should also remain unchanged despite CarEW and CarNS values changing.
2. Check if the CarEWClr and CarNSClr are able to clear values and set them to 0.	• When CarEWClr or CarNSClr is high(1), then CarEW_Sync or CarNS_Sync which are initially 1 become 0 at the rising edge of the clock.
3. Checking that PedEW_Mem and PedNS_Mem retain their values.	• PedEW_Mem and PedNS_Mem retain the value of 1 even after PedEW and PedNS have changed to 0 from 1.
4. Check if the PedEWClr and PedNSClr are able to clear values and set them to 0.	• The values of PedEW_Mem and PedNS_Mem are supposed to change from 1 to 0 at rising edge of the clock when PedEW_Clr and PedNS_Clr are high(1).

Table 1: Aims and Expected results for the Input Sync Testbench

TESTBENCH CODES

The following screenshot shows the codes used for this test bench as well as the explanation for each line of code.

```
15  LIBRARY ieee;
16  USE ieee.std_logic_1164.ALL;
17
18  ENTITY TB_Input_Sync_Memory IS
19  END TB_Input_Sync_Memory;
20
21  ARCHITECTURE behavior OF TB_Input_Sync_Memory IS
22
23      -- Component Declaration for the Unit Under Test (UUT)
24
25      COMPONENT Input_sync_memory
26      PORT(
27           Reset : IN  std_logic;
28           Clock : IN  std_logic;
29           CarEW : IN  std_logic;
30           CarNS : IN  std_logic;
31           PedEW : IN  std_logic;
32           PedNS : IN  std_logic;
33           CarEW_Sync : OUT  std_logic;
34           CarNS_Sync : OUT  std_logic;
35           PedEW_Mem : OUT  std_logic;
36           PedNS_Mem : OUT  std_logic;
37           PedEW_clr : IN  std_logic;
38           PedNS_clr : IN  std_logic;
39           CarEW_clr : in  STD_LOGIC;
40           CarNS_clr : in  STD_LOGIC
41          );
42      END COMPONENT;
```

```
43
44
45      --Inputs
46      signal Reset : std_logic := '0';
47      signal Clock : std_logic := '0';
48      signal CarEW : std_logic := '0';
49      signal CarNS : std_logic := '0';
50      signal PedEW : std_logic := '0';
51      signal PedNS : std_logic := '0';
52      signal PedEW_clr : std_logic := '0';
53      signal PedNS_clr : std_logic := '0';
54      signal CarEW_clr : std_logic := '0';
55      signal CarNS_clr : std_logic := '0';
56
57      --Outputs
58      signal CarEW_Sync : std_logic;
59      signal CarNS_Sync : std_logic;
60      signal PedEW_Mem : std_logic;
61      signal PedNS_Mem : std_logic;
62
63      -- Clock period definitions
64      constant Clock_period : time := 10 ns;
65
```

```
65
66  BEGIN
67
68      -- Instantiate the Unit Under Test (UUT)
69      uut: Input_sync_memory PORT MAP (
70             Reset => Reset,
71             Clock => Clock,
72             CarEW => CarEW,
73             CarNS => CarNS,
74             PedEW => PedEW,
75             PedNS => PedNS,
76             CarEW_Sync => CarEW_Sync,
77             CarNS_Sync => CarNS_Sync,
78             PedEW_Mem => PedEW_Mem,
79             PedNS_Mem => PedNS_Mem,
80             PedEW_clr => PedEW_clr,
81             PedNS_clr => PedNS_clr,
82             CarNS_clr => CarNS_clr,
83             CarEW_clr => CarEW_clr
84           );
85
86      -- Clock process definitions
87      Clock_process :process
88      begin
89         Clock <= '0';
90         wait for Clock_period/2;
91         Clock <= '1';
92         wait for Clock_period/2;
93      end process;
94
95
96      -- Stimulus process
97      stim_proc: process
98      begin
99         -- hold reset state for 100 ns.
100        Reset<='1';
101        wait for 100 ns;
102        Reset<='0';
```

```
104        --Testing Input Sync to see if CarNS_Sync and CarEW_Sync synchronise with the clock
105        wait for 2 ns;
106        CarNS <= '1';
107        wait for 1 ns;
108        CarEW <= '1';
109        wait for Clock_period;
110        CarNS <= '0';
111        CarEW <= '0';
112        wait for 7 ns;
113
114        --Testing Pedestrian Memory to see if PedEW_Mem and PedNS_Mem retain the value of 1
115        PedEW <='1';
116        wait for Clock_period;
117        PedEW <='0';
118        wait for Clock_period*2;
119        PedNS <='1';
120        wait for Clock_period;
121        PedNS <='0';
122        wait for Clock_period*2;
123
124        --Testing clear Memory to see if PedEW_Mem and PedNS_Mem are cleared to 0
125        PedEW_clr<='1';
126        PedNS_clr<='1';
127        wait for Clock_period;
128        PedEW_clr<='0';
129        PedNS_clr<='0';
130        wait for Clock_period;
131
132        --Testing clear Car to see if CarNS_Sync and CarEW_Sync are cleared to 0
133        CarEW_clr<='1';
134        CarNS_clr<='1';
135        wait for Clock_period;
136        CarEW_clr<='0';
137        CarNS_clr<='0';
138        wait for Clock_period;
139        wait;
140    end process;
141 END;
```

2. SIMULATION RESULTS

The following screenshot shows the results obtained from simulating the testbench above.

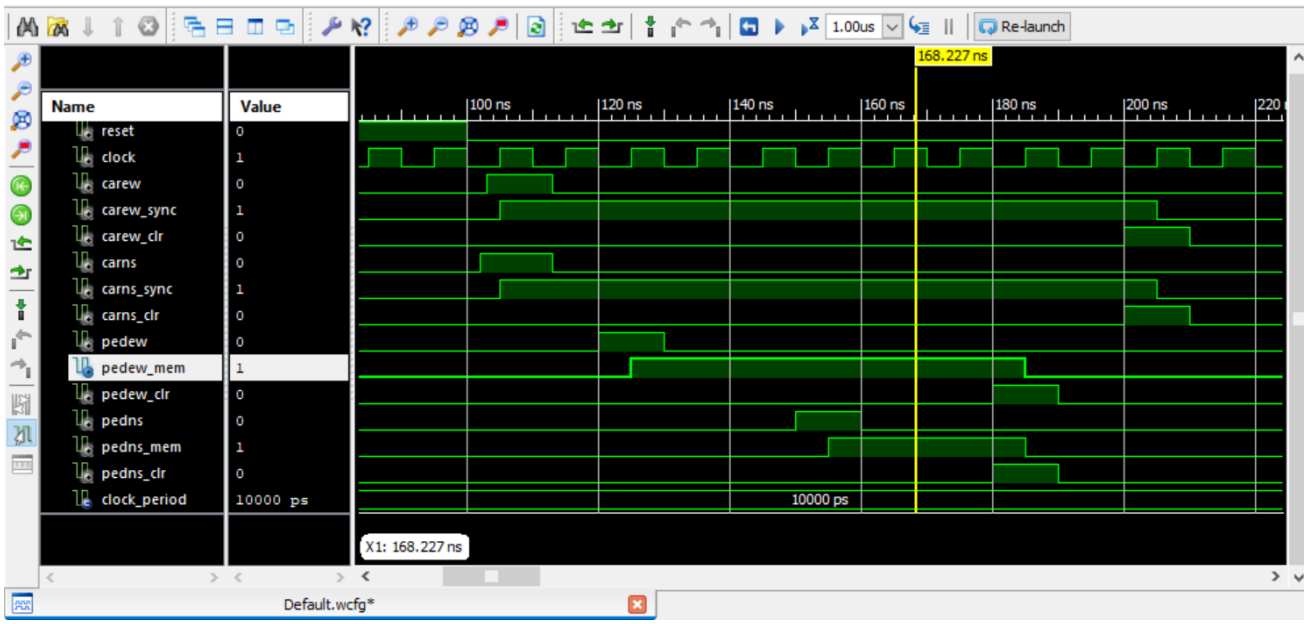

Figure 6: Test Bench Simulation for Input Sync Memory module.

From the screenshot above, the expected reults match the simulated results,showing that the codes work perfectly as intended.

STATE MACHINE MODULE

1. TESTBENCH CODES

AIMS AND EXPECTED RESULTS

The following table shows the aim and expected results of this testbench.

AIM	EXPECTED RESULTS
1. Test for the traffic in the NS.	• the lights of the EW trafffic change from Green to Amber then to red while the NS lights change from red to green and remain green.
2. Test for traffic in the EW.	• the lights of the NS trafffic change from Green to Amber then to red while the EW lights change from red to green and remain green.
3. Check if CarNS_Clr is high on a specific state and low on the other states.	• CarNS_Clr is 1 throughout NSGreenMin state, 1 for one clock cycle in the NSGreen_Idle and 0 in the other states.
4. Check if CarEW_Clr is high on a specific state and low on the other states.	• CarEW_Clr is 1 throughout EWGreenMin state, 1 for one clock cycle in the EWGreen_Idle and 0 in the other states.
5. Test for pedestrians in the NS lane.	• the lights change to both green in the traffic and pedestrian lane then back to green for the NS traffic
6. Check if the timer is working properly.	• Check if the counter counts properly as per the time delays indicated in each state before they transition to the next state.
7. Check if PedNS_Clr is high on a specific state and low on the other states.	• PedNS_Clr is 1 when both Pedestrian and traffic lights are green and 0 for the remaining states
8. Check if PedEW_Clr is high on a specific state and low on the other states.	• PedEW_Clr is 1 when both Pedestrian and traffic lights are green and 0 for the remaining states.

Table 2: Aims and expected results for State Machine testbench

NOTE: The traffic light colours are represented by 2 bit binary which are listed below;

- 00-Red
- 01-Amber
- 10-Green
- 11-Both Pedestrian nd traffic lights are green

TESTBENCH CODES

The following screenshot shows the codes used to simulate this testbench.

```
27  --------------------------------------------------------------------------------
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30
31  -- Uncomment the following library declaration if using
32  -- arithmetic functions with Signed or Unsigned values
33  --USE ieee.numeric_std.ALL;
34
35  ENTITY TB_StateMachine IS
36  END TB_StateMachine;
37
38  ARCHITECTURE behavior OF TB_StateMachine IS
39
40      -- Component Declaration for the Unit Under Test (UUT)
41
42      COMPONENT StateMachine
43      PORT(
44           Clock : IN  std_logic;
45           Reset : IN  std_logic;
46           CarEW : IN  std_logic;
47           CarNS : IN  std_logic;
48           PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
49           PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing EW road)
50
51           LightsEW : OUT  std_logic_vector(1 downto 0);
52           LightsNS : OUT  std_logic_vector(1 downto 0);
53           LEDs     : out STD_LOGIC_VECTOR (3 downto 0);   -- controls LED Lights
54           PedEWClr : out STD_LOGIC;
55           PedNSClr : out STD_LOGIC;
56           CarEWClr : out STD_LOGIC;
57           CarNSClr : out STD_LOGIC
58          );
59      END COMPONENT;
```

```
61      --Inputs
62      signal Clock : std_logic := '0';
63      signal Reset : std_logic := '0';
64      signal CarEW : std_logic := '0';
65      signal CarNS : std_logic := '0';
66      signal PedEW : std_logic := '0';
67      signal PedNS : std_logic := '0';
68      --Outputs
69      signal LightsEW : std_logic_vector(1 downto 0);
70      signal LightsNS : std_logic_vector(1 downto 0);
71      signal   LEDs   : STD_LOGIC_VECTOR (3 downto 0);
72      signal PedEWClr :  STD_LOGIC;
73      signal PedNSClr :  STD_LOGIC;
74      signal CarEWClr :  STD_LOGIC;
75      signal CarNSClr :  STD_LOGIC;
76      -- Clock period definitions
77      constant Clock_period : time := 10 ns;
78  BEGIN
79      -- Instantiate the Unit Under Test (UUT)
80      uut: StateMachine PORT MAP (
81             Clock => Clock,
82             Reset => Reset,
83             CarEW => CarEW,
84             CarNS => CarNS,
85             PedEW => PedEW,
86             PedNS => PedNS,
87             LightsEW => LightsEW,
88             LightsNS => LightsNS,
89             LEDs  =>LEDs,
90             PedEWClr => PedEWClr,
91             PedNSClr => PedNSClr,
92             CarEWClr => CarEWClr,
93             CarNSClr => CarNSClr
94           );
95
```

```
96      -- Clock process definitions
97      Clock_process :process
98      begin
99         Clock <= '0';
100        wait for Clock_period/2;
101        Clock <= '1';
102        wait for Clock_period/2;
103     end process;
104
105
106     -- Stimulus process
107     stim_proc: process
108     begin
109        -- hold reset state for 100 ns.
110        Reset <= '1';
111        wait for 100 ns;
112        Reset <= '0';
113
```

```
114      --Testing traffic in the NS and EW lanes
115       wait for 2 ns;
116       CarNS <= '1';
117       wait for 1 ns;
118       CarEW <= '1';
119       wait for Clock_period;
120       CarNS <= '0';
121       CarEW <= '0';
122       wait for 7 ns;
123
124       --Testing Pedestrian lights in the EW and NS lanes
125       PedEW <='1';
126       wait for Clock_period;
127       PedEW <='0';
128       wait for Clock_period*2;
129       PedNS <='1';
130       wait for Clock_period;
131       PedNS <='0';
132       wait for Clock_period*2;
133       wait;
134    end process;
135
136 END;
137
```

1. **Testing for traffic in the EW lane**

```
--Testing traffic in the EW  lane
 wait for 2 ns;
 CarNS <= '0';
 wait for 1 ns;
 CarEW <= '1';
 wait for Clock_period;
 CarNS <= '0';
 CarEW <= '0';
 wait for 7 ns;
```

---

2. **Testing for traffic in the NS lane**

```
--Testing traffic in the NS  lane
 wait for 2 ns;
 CarNS <= '1';
 wait for 1 ns;
 CarEW <= '0';
 wait for Clock_period;
 CarNS <= '0';
 CarEW <= '0';
 wait for 7 ns;
```

---

3. **Testing for traffic in the NS and EW lane**

```
--Testing traffic in the NS and EW lanes
 wait for 2 ns;
 CarNS <= '1';
 wait for 1 ns;
 CarEW <= '1';
 wait for Clock_period;
 CarNS <= '0';
 CarEW <= '0';
 wait for 7 ns;
```

4. **Testing for Pedestrian in the EW lane**

```
--Testing Pedestrian lights in the EW lane
PedEW <='1';
wait for Clock_period;
PedEW <='0';
wait for Clock_period*2;
PedNS <='0';
wait for Clock_period;
PedNS <='0';
wait for Clock_period*2;
```

5. **Testing for Pedestrian in the NS lane**

```
--Testing Pedestrian lights in the EW lane
PedEW <='0';
wait for Clock_period;
PedEW <='0';
wait for Clock_period*2;
PedNS <='1';
wait for Clock_period;
PedNS <='0';
wait for Clock_period*2;
wait;
```

6. **Testing for Pedestrians in the NS and EW lane**

```
--Testing Pedestrian lights in the EW and NS lanes
PedEW <='1';
wait for Clock_period;
PedEW <='0';
wait for Clock_period*2;
PedNS <='1';
wait for Clock_period;
PedNS <='0';
wait for Clock_period*2;
wait;
```

2. SIMULATION RESULTS

The following screenshots display the results obtained from the simulation of this testbench.

1. **Testing for traffic in the EW lane**

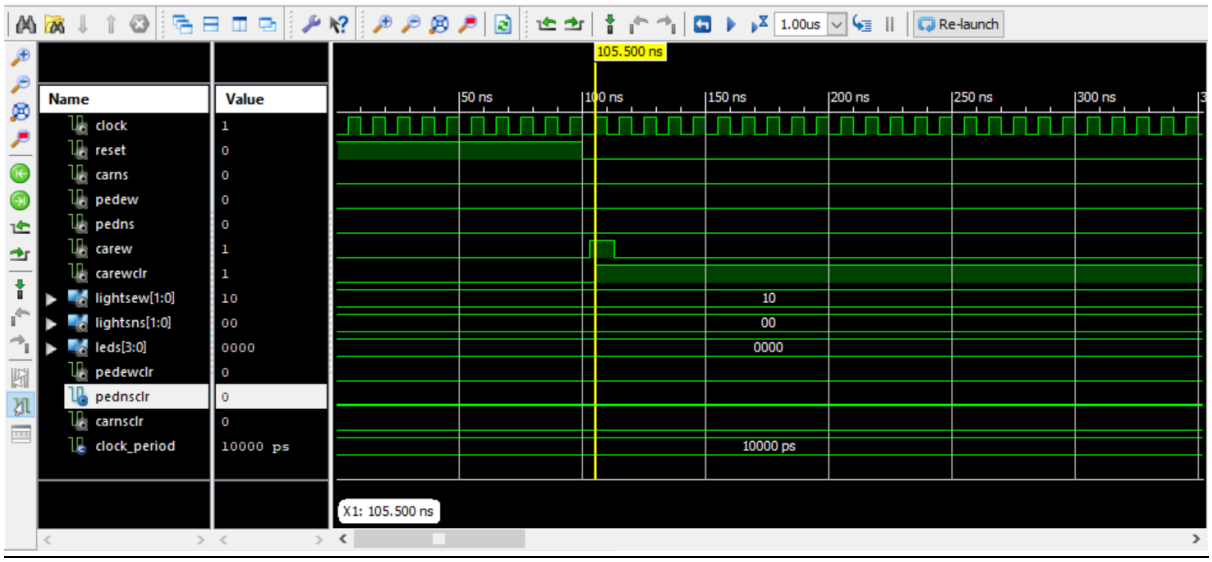

Figure 7: Test Bench Simulation for traffic in the EW lane in the State Machine Module.

From the diagram above,the smulated results match the expected results.

2. **<u>Testing for traffic in the NS lane</u>**

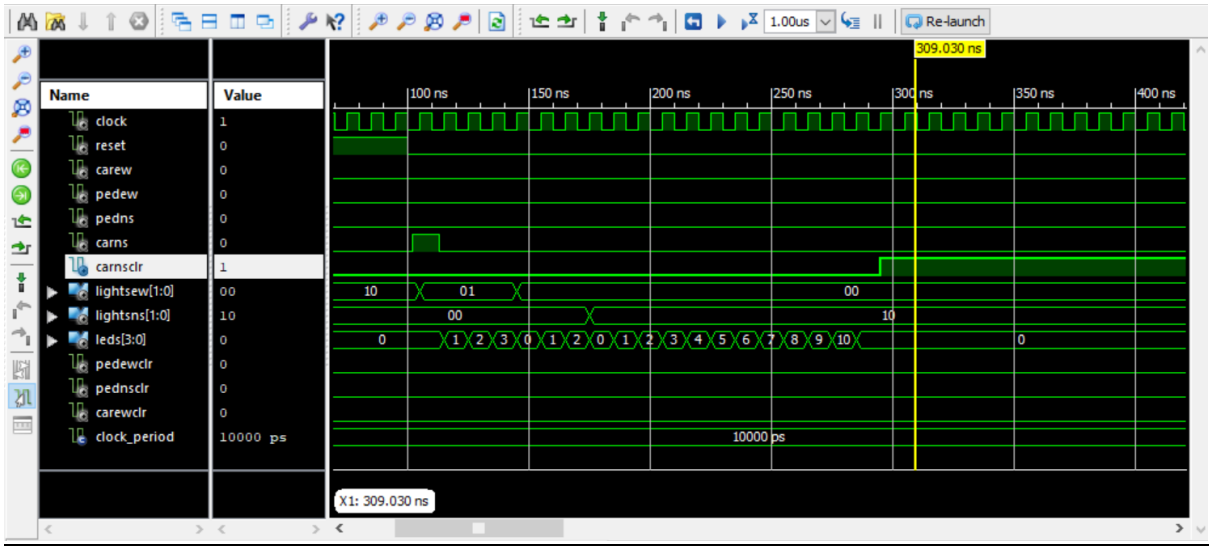

Figure 8: Test Bench Simulation for traffic in the NS lane in the State Machine Module.

From the diagram above,the smulated results match the expected results.

3. **<u>Testing for traffic in the NS and EW lane</u>**

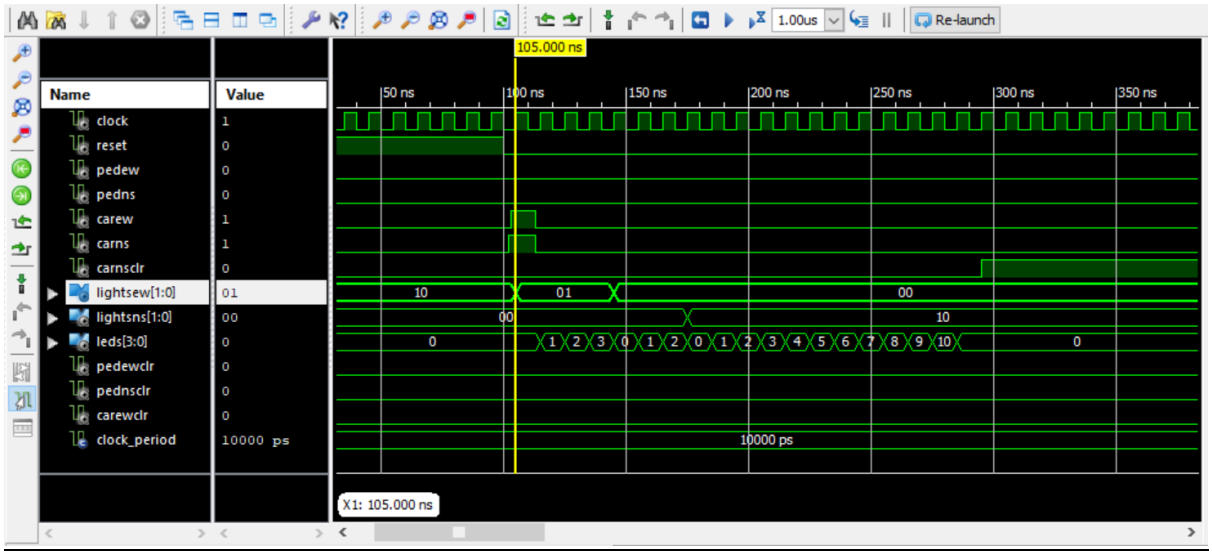

Figure 9: Test Bench Simulation for Traffc in the EW and NS lane in the State Machine *Module.*

From the diagram above,the smulated results match the expected results.The traffic in the NS lane simulates the expected light change.As the EW lights was initially green,the traffic in that lane has already been given priority and therefore it moves to NS lane to cater for the traffic in that lane.The lights remain unchanged(green) in the NS lane for an unspecified amount of time until there is traffic in the EW lane,afterwhich traffic lights will change for both lanes.

4. Testing for Pedestrian in the EW lane

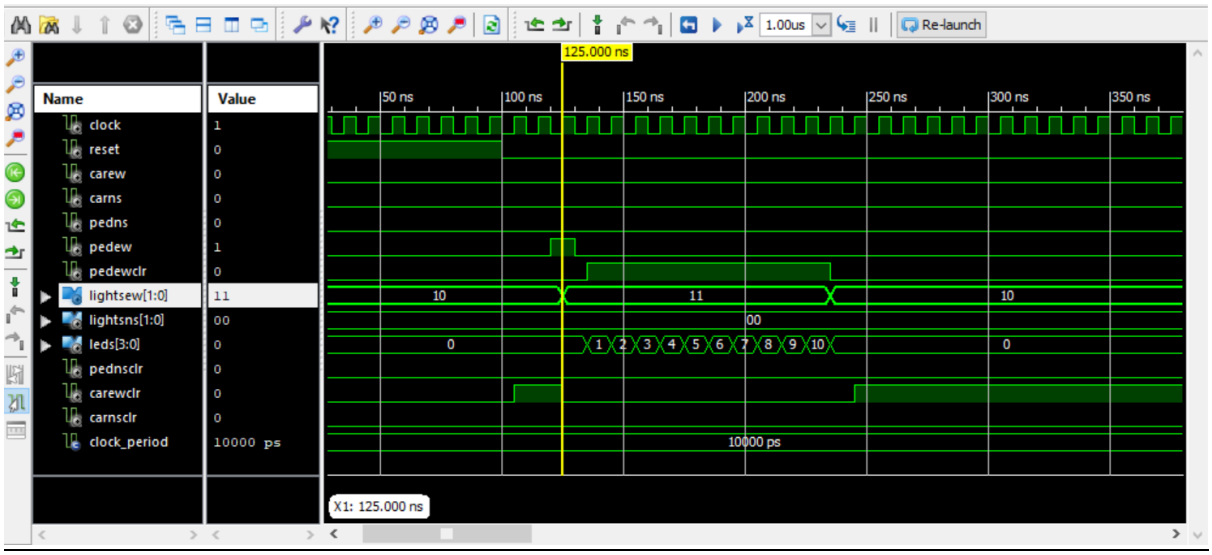

Figure 10: Test Bench Simulation for Pedestrian in the EW lane in the State Machine Module.

From the diagram above,the smulated results match the expected results.

5. Testing for Pedestrian in the NS lane

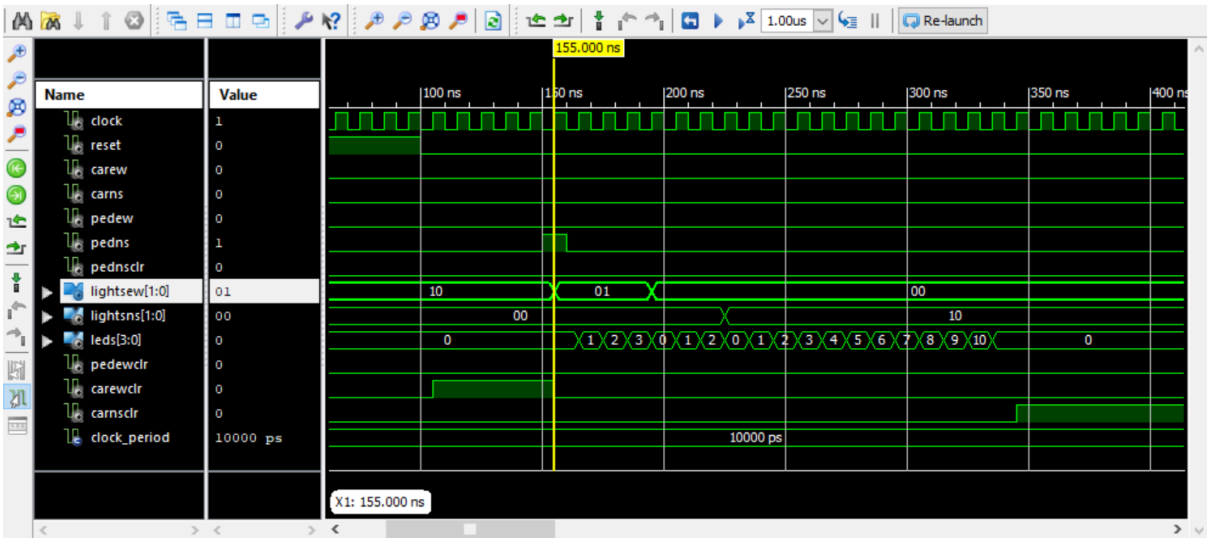

Figure 11: Test Bench Simulation for traffic in the EW lane in the State Machine Module.

From the diagram above,the smulated results do not match the expected results.This is because the pedNS input is not saved to the memory of the system(a latch has not been used).Therefore,it does not change to 11 state from 00 state for the lights in the NS traffic because based on the system,the pedNS input is taken as 0 instead of 1.(The system assumes that there is no pedestrian in the NS lane).This will be fixed in the Top Level Traffic module. The lights in the EW traffic lane match the expected results because initially,the PedNS input is 1.(System assumes there is a pedestrian in the NS lane).

6. Testing for traffic in the NS and EW lane

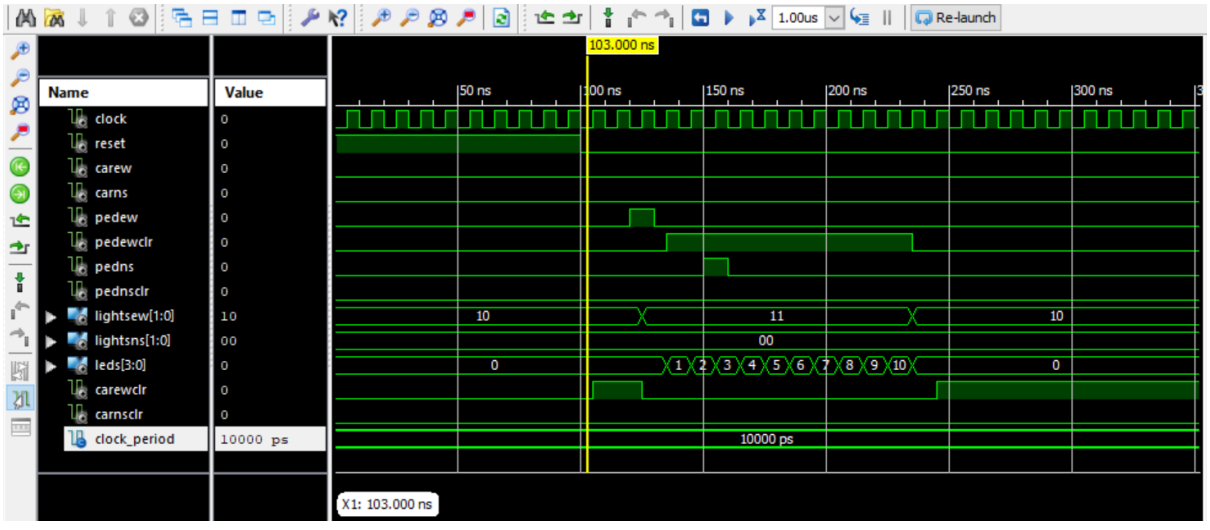

Figure 12: Test Bench Simulation for Pedestrians in the EW and NS lanes in the State Machine Module.

From the diagram above,the smulated results does not match the expected results.This is because for the State machine module, a latch was not used for the Pedestrian button and therefore the system does not remember if the Pedestian button was pressed .

TOP LEVEL TRAFFIC

1. TESTBENCH CODES

AIMS AND EXPECTED RESULTS

AIMS	EXPECTED RESULTS
1. Check if the CarEW_Sync and CarNS_Sync are able to synchronise with the clock and hold their values despite carNS and carEW values changing to 0.	• the CarEW_Sync and CarNS_Sync remain 1 until they are cleared to 0 in the NSGreen_min and EWGreen_min state.This should occur after 2 seconds
2. Check if the PedEW_Mem and PedNS_Mem are able to synchronise with the clock and hold their values despite PedNS and PedEW values changing to 0	• the PedEW_Mem and PedNS_Mem once high should remain 1 and be cleared to 0 in the EWPedGreen and NSPedGreen state ie 11 state when both traffic and Pedestrian lihts are green.
3. Check for when there is traffic in both the NS and EW lanes.	• The traffic lights should cycle from green to amber to red in the EW lane and simultaneously change from red to green in the NS traffic lane.It should remain green in the NS traffic light and red in the EW traffic light.
4. Check when there is Pedestrians in both EW and NS traffic lanes	• If The traffic lights were initially green at the EW traffic lane,then the lights should change accordingly to serve the pedestrian in the NS lane before cycling back to cater for the pedestrian in the EW lane and vice versa.
5. Check for a Pedestrian in EW lane and traffic in the NS lane at the same time.	• If the traffic lights are initially green in the EW traffic lane,the traffic lights should cycle and take care of the traffic in the NS lane before cycling back to serve the pedestrian in the EW lane.
6. Check for a Pedestrian in NS lane and traffic in the EW lane at the same time.	• If the traffic lights are initially green in the NS traffic lane,the traffic lights should cycle and take care of the traffic in the EW lane before cycling back to serve the pedestrian in the NS lane.

Table 3: Aims and expected results for Top Level Traffic testbench

TEST CODES

Below is a screenshot used to simulate the testbench of Top level Traffic module.

```
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY TB_TrafficTopLevel IS
END TB_TrafficTopLevel;

ARCHITECTURE behavior OF TB_TrafficTopLevel IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT TrafficTopLevel
    PORT(
         Clock : IN  std_logic;
         Reset : IN  std_logic;
         CarEW : IN  std_logic;
         CarNS : IN  std_logic;
         PedEW : IN  std_logic;
         PedNS : IN  std_logic;
         LightsEW : OUT  std_logic_vector(1 downto 0);
         LightsNS : OUT  std_logic_vector(1 downto 0);
         CarEWSync : out STD_LOGIC; -- Car on EW road
         CarNSSync : out STD_LOGIC; -- Car on NS road
         PedEW_Mem : out STD_LOGIC; -- Pedestrian moving EW (crossing NS road)
         PedNS_Mem : out STD_LOGIC; -- Pedestrian moving NS (crossing EW road)
         LEDs      : out STD_LOGIC_VECTOR (3 downto 0)   -- controls LED Lights
        );
    END COMPONENT;
```

```
   --Inputs
   signal Clock : std_logic := '0';
   signal Reset : std_logic := '0';
   signal CarEW : std_logic := '0';
   signal CarNS : std_logic := '0';
   signal PedEW : std_logic := '0';
   signal PedNS : std_logic := '0';

   --Outputs
   signal LightsEW : std_logic_vector(1 downto 0);
   signal LightsNS : std_logic_vector(1 downto 0);
   signal LEDs     : STD_LOGIC_VECTOR (3 downto 0);   -- controls LED Lights
   signal CarEWSync :STD_LOGIC; -- Car on EW road
   signal CarNSSync :STD_LOGIC; -- Car on NS road
   signal PedNS_Mem :STD_LOGIC;
   signal PedEW_Mem :STD_LOGIC;
   -- Clock period definitions
   constant Clock_period : time := 10 ns;

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: TrafficTopLevel PORT MAP (
          Clock => Clock,
          Reset => Reset,
          CarEW => CarEW,
          CarNS => CarNS,
          PedEW => PedEW,
          PedNS => PedNS,
          LightsEW => LightsEW,
          LightsNS => LightsNS,
          CarEWSync =>CarEWSync,
          CarNSSync =>CarNSSync,
          PedEW_Mem=>PedEW_Mem,
          PedNS_Mem=>PedNS_Mem,
          LEDS => LEDs
        );
```

```

   -- Clock process definitions
   Clock_process :process
   begin
      Clock <= '0';
      wait for Clock_period/2;
      Clock <= '1';
      wait for Clock_period/2;
   end process;



   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      Reset<='1';
      wait for 1 ns;
      Reset<='0';
```

```
117
118         --Testing if the CarEW input is Synchronised and Holds till a specific state
119         wait for 2 ns;
120         CarNS <= '1';
121         wait for 1 ns;
122         CarEW <= '1';
123         wait for Clock_period;
124
125         CarNS <= '0';
126         CarEW <= '0';
127         wait for 7 ns;
128
129         --Testing Pedestrian Memory-Check if Pedestrian Input is held until a specific state before being cleared.
130         PedEW <='1';
131         wait for Clock_period;
132         PedEW <='0';
133         wait for Clock_period*2;
134
135         PedNS <='1';
136         wait for Clock_period;
137         PedNS <='0';
138         wait for Clock_period*2;
139
140         -- insert stimulus here
141
142         wait;
143     end process;
144
145 END;
```

1. **Testing for traffic in the EW lane**

```
119         wait for 2 ns;
120         CarNS <= '0';
121         wait for 1 ns;
122         CarEW <= '1';
123         wait for Clock_period;
124
125         CarNS <= '0';
126         CarEW <= '0';
127         wait for 7 ns;
```

2. **Testing traffic in the NS lane**

```
119         wait for 2 ns;
120         CarNS <= '1';
121         wait for 1 ns;
122         CarEW <= '0';
123         wait for Clock_period;
124
125         CarNS <= '0';
126         CarEW <= '0';
127         wait for 7 ns;
128
```

3. **Testing for Pedestrian in the EW lane**

```
130         PedEW <='1';
131         wait for Clock_period;
132         PedEW <='0';
133         wait for Clock_period*2;
134
135         PedNS <='0';
136         wait for Clock_period;
137         PedNS <='0';
138         wait for Clock_period*2;
139
```

4. **Testing for Pedestrian in the NS lane**

```
130         PedEW <='0';
131         wait for Clock_period;
132         PedEW <='0';
133         wait for Clock_period*2;
134 
135         PedNS <='1';
136         wait for Clock_period;
137         PedNS <='0';
138         wait for Clock_period*2;
139 
```

5. **Testing for pedestrians in both NS and EW lanes**

a. When pedestrian in the EW traffic lane presses the button first before the one in the NS lane.

```
129         --Testing Pedestrian Memory-Check if Pedestrian Input is held until a specific state before being cleared.
130         PedEW <='1';
131         wait for Clock_period;
132         PedEW <='0';
133         wait for Clock_period*2;
134 
135         PedNS <='1';
136         wait for Clock_period;
137         PedNS <='0';
138         wait for Clock_period*2;
139 
```

b. When pedestrian in the NS traffic lane presses the button first before the one in the EW lane.

```
129         --Testing Pedestrian Memory-Check if Pedestrian Input is held until a specific state before being cleared.
130         PedNS <='1';
131         wait for Clock_period;
132         PedNS <='0';
133         wait for Clock_period*2;
134 
135         PedEW <='1';
136         wait for Clock_period;
137         PedEW <='0';
138         wait for Clock_period*2;
139 
```

6. **Testing for pedestrian in the EW lane and traffic in the NS lane**

```
118         --Testing for CarNS and PedEW input at the same time
119         wait for 2 ns;
120         CarEW <= '0';
121         PedNS <='0';
122         CarNS <= '1';
123         PedEW <='1';
124         wait for Clock_period;
125         CarNS <= '0';
126         CarEW <= '0';
127         PedEW <='0';
128         PedNS <='0';
129 
130         wait for Clock_period*2;
```

7. **Testing for pedestrian in the NS lane and traffic in the EW lane**

```
118         --Testing for CarEW and PedNS input at the same time
119         wait for 2 ns;
120         CarEW <= '1';
121         PedNS <='1';
122         CarNS <= '0';
123         PedEW <='0';
124         wait for Clock_period;
125         CarNS <= '0';
126         CarEW <= '0';
127         PedEW <='0';
128         PedNS <='0';
129
130         wait for Clock_period*2;
131
```

2. SIMULATION RESULTS

1. Testing for traffic in the EW lane

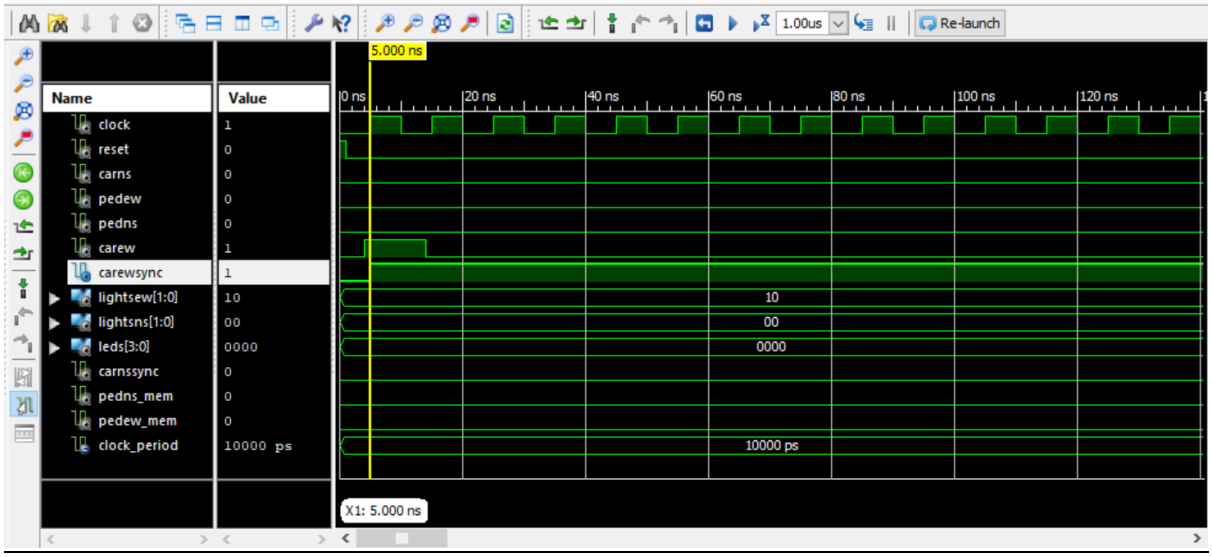

Figure 13: Test Bench Simulation for Traffic in the EW lane in the Top Level Traffic Module.

The simulated results match the expected results.CarEWSync synchronises with clock and retains its value despite the value of carEW changing from 1 to 0.

2. Testing traffic in the NS lane

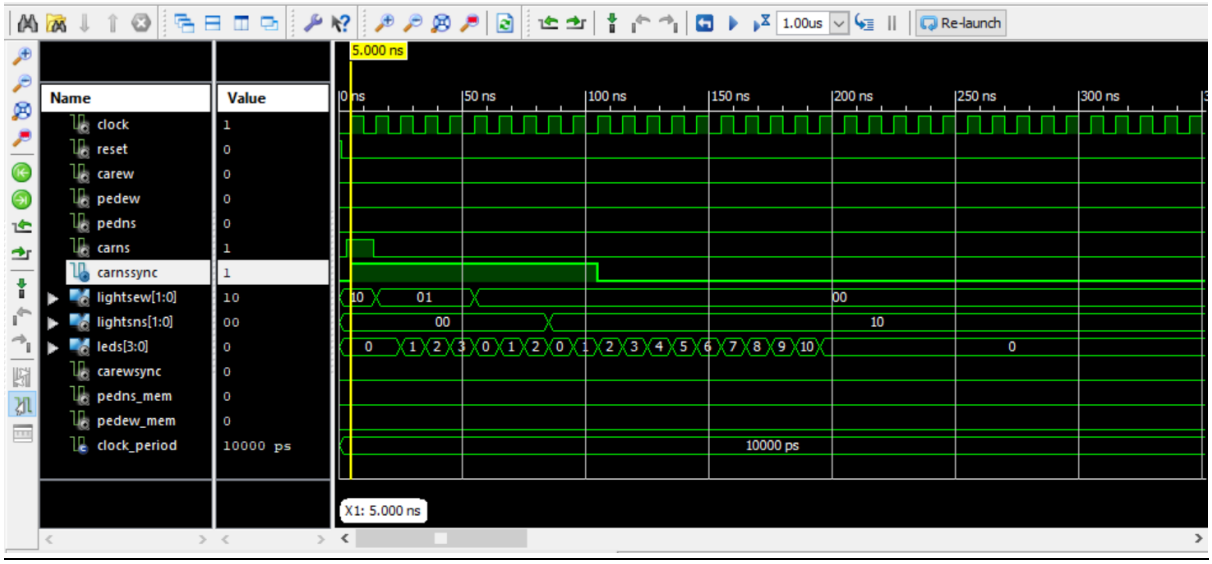

Figure 14: Test Bench Simulation for Traffic in the NS lane in the Top Level Traffic Module.

The simulated results match the expected results.CarNSSync synchronises with the clock and is able to hold its value of 1 despite the value of CarNS changing to 0.The value is cleared 0 in the NSGreen_Min state.This state can be confirmed by looking at the counter;In this state the counter counts till 10 seconds as expected.

3. Testing for Pedestrian in the EW lane

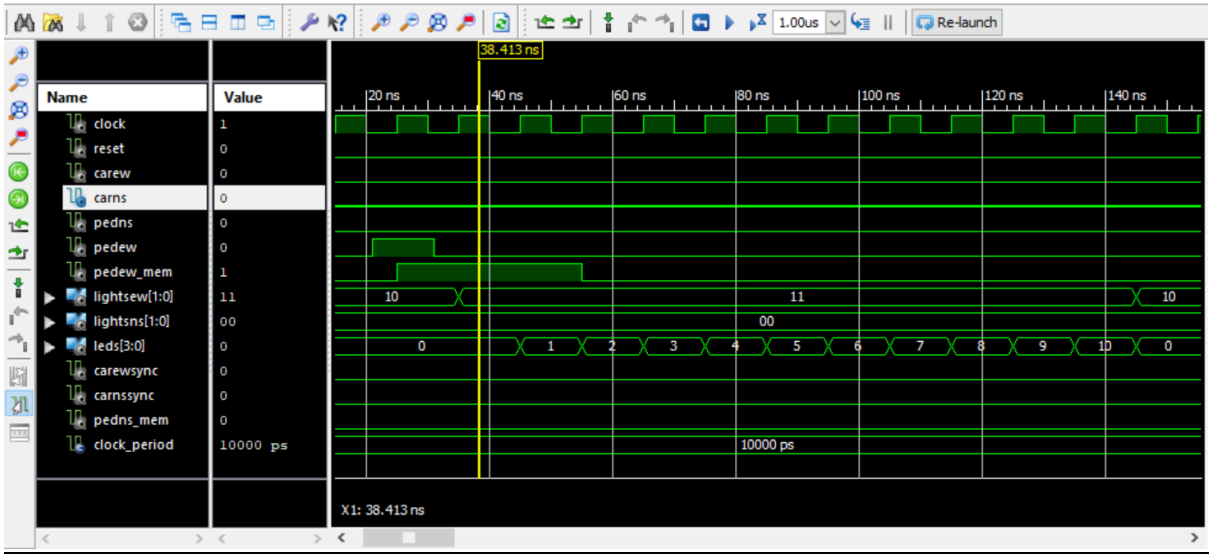

Figure 15: Test Bench Simulation for Pedestrian in the EW lane in the Top Level Traffic Module.

The simulated results match the expected results.PedEW_Mem synchronises with the clock and is able to hold its value of 1 despite the value of PedEW changing to 0.The value is cleared 0 in the EWPedGreen state(11).In this state the counter counts till 10 seconds as expected before going back to the EWGreen_Idle state which is 10.

4. **Testing for Pedestrian in the NS lane**

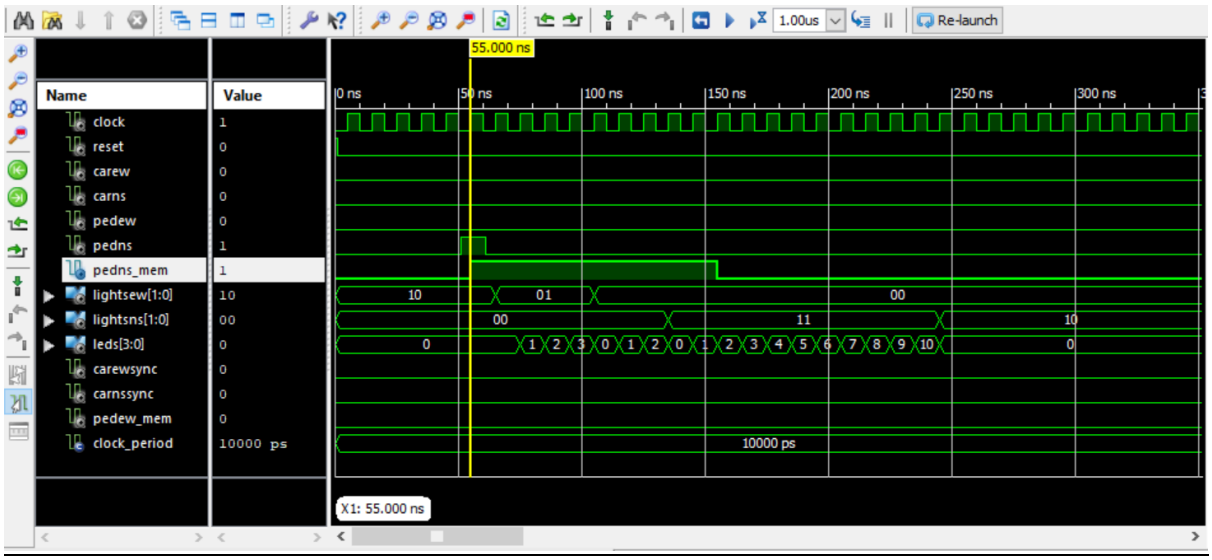

Figure 16: Test Bench Simulation for Pedestrian in the NS lane in the Top Level Traffic Module.

The simulated results match the expected results.PedNS_Mem synchronises with the clock and is able to hold its value of 1 despite the value of PedNS changing to 0.The value is cleared 0 in the NSPedGreen state(11).In this state the counter counts till 10 seconds as expected before going back to the NSGreen_Idle state which is 10.

5. **Testing for pedestrians in both NS and EW lanes**

a. When pedestrian in the EW traffic lane presses the button first before the one in the NS lane.

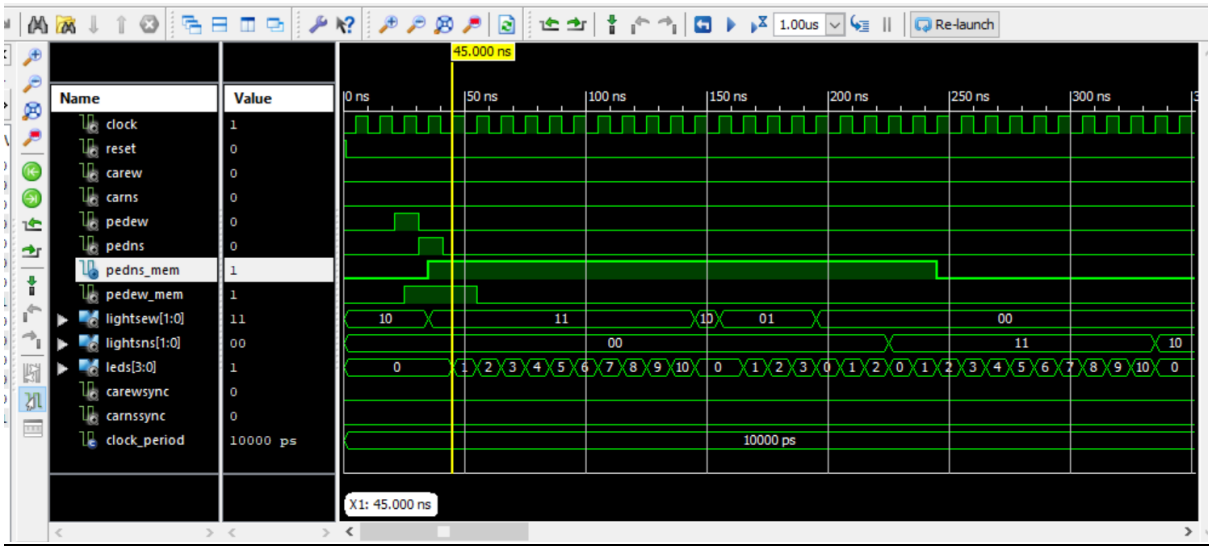

Figure 17: Test Bench Simulation for Pedestrian in the EW lane Pressing button first before NS pedestrian in the Top Level Traffic Module.

The simulated results match the expected results.This shows that the sensitivity of this traffic control system works well.When the EW pedestrian presses the button first,the traffic lights will change accordingly,serving them first before cycling back to serve the NS pedestrian who pressed the button second as shown in the above screenshot.

b. <u>When pedestrian in the NS traffic lane presses the button first before the one in the EW lane.</u>

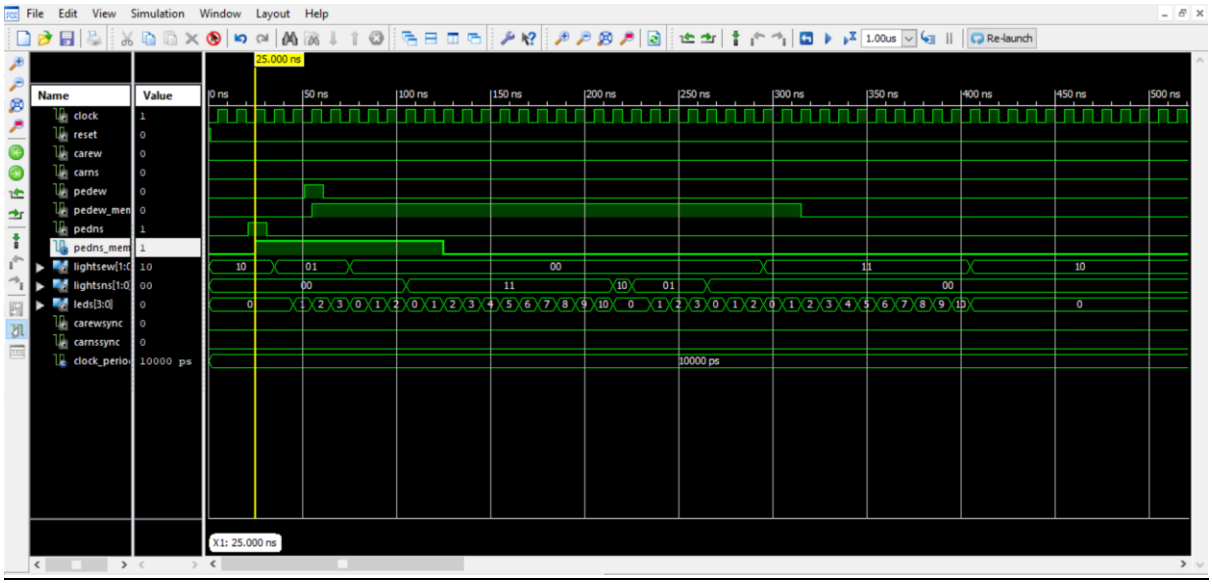

Figure 18: Test Bench Simulation for Pedestrian in the NS lane Pressing button first before EW pedestrian in the Top Level Traffic Module.

The simulated results match the expected results, showing how sensitive the traffic control system is.When the NS pedestrian presses the button first,the traffic lights will change accordingly,serving them first before cycling back to serve the EW pedestrian who pressed the button second as shown in the above screenshot.

c. <u>Testing for pedestrian in the EW lane and traffic in the NS lane</u>

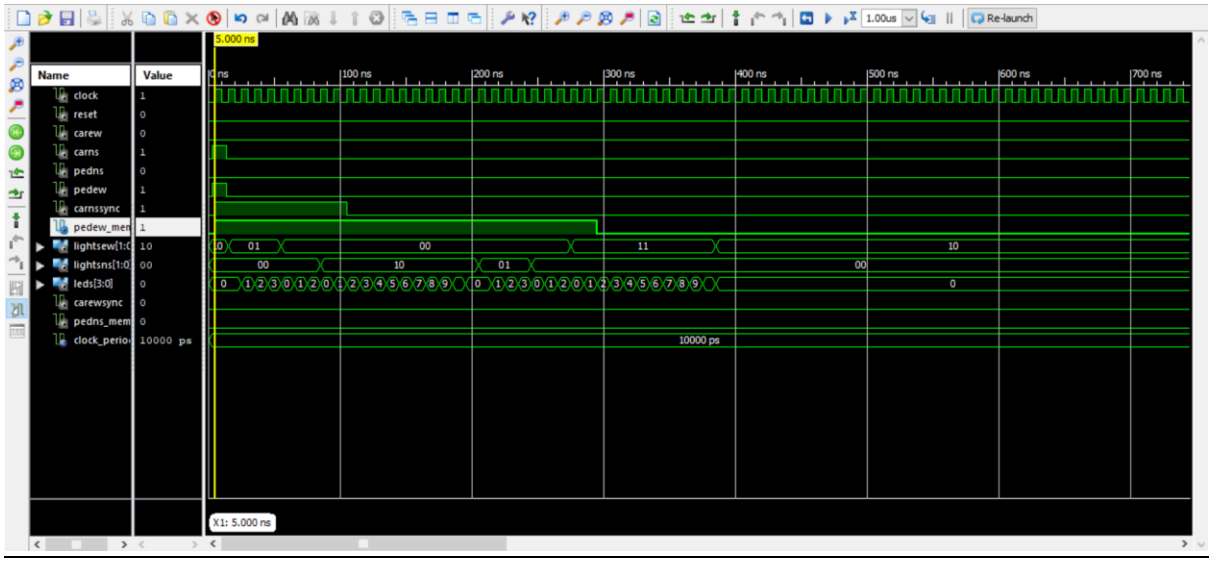

Figure 19: Test Bench Simulation for Pedestrian in the EW lane and car in the NS lane at the same time in the Top Level Traffic Module.

The simulated results match the expected results.The traffic control system gives priority to the traffic in the NS lane.The traffic lights therefore change accordingly to cater for the NS traffic before cycling back to tend to the pedestrian.

Table showing traffic lights changing as per the smulation is shown below.

EW Traffic Lights	**NS Traffic Lights**
Green	Red
Amber	Red
Red	Red
Red	Green
Red	Amber
Red	Red
Green in both Traffic and Pedestrian Lights	Red
Green	Red

d. <u>Testing for pedestrian in the NS lane and traffic in the EW lane</u>

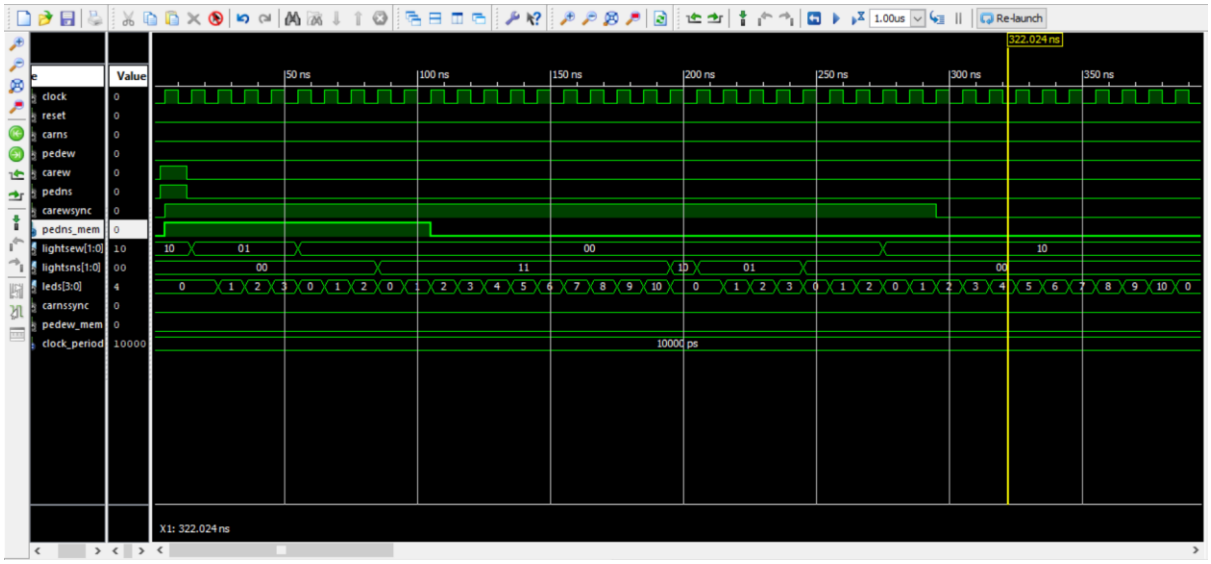

Figure 20: Test Bench Simulation for Pedestrian in the NS lane and car in the EW lane at the same time in the Top Level Traffic Module.

The simulated results match the expected results.The traffic control system gives priority to the pedestrian in the NS lane.The traffic lights therefore change accordingly to cater for the NS Pedestrian before cycling back to tend to the traffic in the EW lane.

Table showing traffic lights changing as per the smulation is shown below.

EW Traffic Lights	**NS Traffic Lights**
Green	Red
Amber	Red
Red	Red
Red	Green in both Traffic and Pedestrian lights
Red	Green
Red	Amber
Red	Red
Green	Red

DEBUGGING PROCESS

During the coding process, I came across a few bugs which led to a couple of errors in my code.They were irritating if I'm to be honest,but after going through my codes again line by line I was able to correct the mistakes and fix the bug. Xilinx proved really helful in highlighting which line had the bug. Of course there were times when the line which had the bug was above the line highlihted by Xilinx so that was a very interesting observation I made. Here are ome of the common bugs I found while writing my codes.

1. Misspelling some words

Unlike C language, when using if-else statement, in Xilinx it is elsif and not elseif.

WRONG

```
elseif (PedEW = '1' and ((carNS = '0') and (PedNS = '0'))) then
```

CORRECT

```
elsif (PedEW = '1' and ((carNS = '0') and (PedNS = '0'))) then
```

2. Using wrong signs when assigning values to signals

When assigning values to the output, I used => instead of <= which was wrong. Also using == in the if statement instead of just = which was wrong.

WRONG

```
 LEDs =>count;

if(PedNS =='1') then
```

CORRECT

```
LEDs <=count;

if(PedNS ='1') then
```

3. Forgeting to add then after if statement

Unlike C++ language where there is no 'then' after if statement,in Xilinx,there is a 'then'after the if statement.

WRONG

```
   if(PedNS ='1')
```

CORRECT

```
if(PedNS ='1') then
```

4. Using the wrong binary bit

When writing the values for the different time delays,I confused the binary number for 10 with that of 5 which ended up giving me wrong time delays.This was a little hard to find as this was not an actual error in the codes,so Xilinx did not actually highlight the line.I therefore had to go throughthe codes line by line until I found and corrected this.

WRONG

```
constant COUNT10:STD_LOGIC_VECTOR(3 downto 0) :="0101";
```

CORRECT

```
constant COUNT10:STD_LOGIC_VECTOR(3 downto 0) :="1010";
```

VHDL LISTING

1. INPUT SYC MEMORY MODULE

```
----------------------------------------------------------------------------------
--
-- Company:
-- Engineer: KELLY DIANGÁ
-- Tool versions:
-- Description: To remember pedestrian button press and to synchronize the
inputs
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Input_sync_memory is
    Port ( Reset : in  STD_LOGIC;
           Clock : in  STD_LOGIC;
           CarEW : in  STD_LOGIC;
           CarNS : in  STD_LOGIC;
           PedEW : in  STD_LOGIC;
           PedNS : in  STD_LOGIC;
           CarEW_Sync : out  STD_LOGIC;
           CarNS_Sync : out  STD_LOGIC;
           PedEW_Mem : out  STD_LOGIC;
           PedNS_Mem : out  STD_LOGIC;
           PedEW_clr : in  STD_LOGIC;
           PedNS_clr : in  STD_LOGIC;
                    CarEW_clr : in  STD_LOGIC;
           CarNS_clr : in  STD_LOGIC
                    );
end Input_sync_memory;

architecture Behavioral of Input_sync_memory is

begin

  process(Reset,Clock,CarEW,CarNS,PedEW,PedNS,PedEW_clr,PedNS_clr)
  begin
      if Reset = '1' then --Set the initial values to zero
               CarEW_Sync <= '0';
               CarNS_Sync <= '0';
               PedEW_Mem <= '0';
               PedNS_Mem <= '0';

           elsif rising_edge(Clock)then --Synchronize to the clock(only output
at rising edge of the clock)
                    if CarEW = '1' then   --Store value of CarEW  to carEW_Sync
                          CarEW_Sync <= '1';
                end if;

                    if CarNS = '1' then   --Store value of CarNS  to carNS_Sync
                       CarNS_Sync <= '1';
                    end if;

                if CarEW_clr ='1' then --Clear the value of CarEW_Sync to 0
                       CarEW_Sync <= '0';
                    end if;

                    if CarNS_clr ='1' then --Clear the value of CarNS_Sync to 0
                       CarNS_Sync <= '0';
                    end if;

                if PedEW = '1' then  --Store value of PedEW  to PedEW_Mem once
the button has been pressed briefly
                      PedEW_Mem <= '1';
```

```
                    end if;

                    if PedNS = '1' then   --Store value of PedNS  to PedNS_Mem
once the button has been pressed briefly
                          PedNS_Mem <= '1';
                    end if;

                    if PedEW_clr ='1' then --Clear the value of PedEW_Mem to 0
                       PedEW_Mem <= '0';
                    end if;

                    if PedNS_clr ='1' then  --Clear the value of PedNS_Mem to 0
                       PedNS_Mem <= '0';
                    end if;
            end if;

  end process;


end Behavioral;
```

2. STATE MACHINE MODULE

```
----------------------------------------------------------------------------------
--
-- Company:
-- Engineer:
--
-- Create Date:    15:45:15 12/10/2021
-- Design Name:
-- Module Name:    StateMachine - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity StateMachine is
    Port ( Clock : in  STD_LOGIC;
           Reset : in  STD_LOGIC;

          -- Car and pedestrian buttons
                       CarEW : in STD_LOGIC; -- Car on EW road
                       CarNS : in STD_LOGIC; -- Car on NS road
                       PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing
NS road)
                       PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing
EW road)

                       -- Light control
                       LightsEW : out STD_LOGIC_VECTOR (1 downto 0); --
controls EW lights
                       LightsNS : out STD_LOGIC_VECTOR (1 downto 0); --
controls NS lights
                       LEDs     : out STD_LOGIC_VECTOR (3 downto 0);   --
controls LED Lights
                       PedEWClr : out STD_LOGIC;
                       PedNSClr : out STD_LOGIC;
                       CarEWClr : out STD_LOGIC;
                       CarNSClr : out STD_LOGIC
                       );
end StateMachine;

architecture Behavioral of StateMachine is

type StateType is (EWGreenIdle, EWAmber, EWRed, NSGreenMin, NSPedGreen,
NSGreenIdle, NSAmber, NSRed, EWGreenMin, EWPedGreen);

signal state : StateType;

--Set the different time delays
signal count:STD_LOGIC_VECTOR(3 downto 0);--counter
constant COUNT2:STD_LOGIC_VECTOR(3 downto 0) :="0010";--2 seconds
constant COUNT3:STD_LOGIC_VECTOR(3 downto 0) :="0011"; --3 seconds
```

```
constant COUNT10:STD_LOGIC_VECTOR(3 downto 0) :="1010";--10 seconds

signal PedEWClear,PedNSClear,CarEWClear,CarNSClear : STD_LOGIC;

begin


 SynchronousProcess:
 process (reset, clock,carNS,carEW,PedNS,PedEW) --Ensure the inputs synchronise
with the clock
 begin
    if (reset = '1') then --Set values to 0
        state <= EWGreenIdle;
              count <= X"0";
              PedEWClear <= '0';
              PedNSClear <= '0';
              CarEWClear <= '0';
              CarNSClear <= '0';

    elsif (rising_edge(clock)) then --States will transition at the rising edge
of the clock
          case state is
             when EWGreenIdle =>               -- EWGreenIdle state
                       if ((carNS = '1')OR (PedNS ='1')) then
                              state<= EWAmber;
                              count <= X"0";
                    CarEWClear <= '0';

                               elsif (PedEW = '1' and ((carNS = '0') and (PedNS =
'0'))) then
                                state<= EWPedGreen;
                                count <= X"0";
                                      CarEWClear <= '0';
                             else
                                state<=EWGreenIdle;

                             end if;


                     when EWPedGreen =>            -- EWPedGreen State
                         if (count < COUNT10) then
                                state<= EWPedGreen;
                                count <= count + 1;
                    PedEWClear <='1';
                            else
                                state<=EWGreenIdle;
                                count <= X"0";
                                      PedEWClear <='0';
              end if;

                     when EWAmber =>               -- EWAmber State
                         if (count < COUNT3) then
                                state<= EWAmber;
                                count <= count + 1;
                            else
                                state<=EWRed;
                                       count <= X"0";
              end if;

                     when EWRed =>                 -- EWRed State
                         if (count < COUNT2) then
                                state<= EWRed;
                                count <= count + 1;
                            else
```

```
                                if(PedNS ='1') then
                                state<=NSPedGreen;
                                count <= X"0";

                                else
                            state<=NSGreenMin;
                            count <= X"0";
                                end if;
        end if;

                    when NSGreenMin =>          --NSGreenMin State
                    if (count < COUNT10) then
                            state<= NSGreenMin;
                            count <= count + 1;
             CarNSClear <= '1';

                        else
                            state<=NSGreenIdle;
                            count <= X"0";
        end if;

                    when NSGreenIdle =>         --NSGreenIdle State
                   if ((carEW = '1')OR (PedEW ='1')) then
                            state<= NSAmber;
                            count <= X"0";
             CarNSClear <= '0';

                       elsif (PedNS = '1' and ((carEW = '0') and (PedEW =
'0'))) then
                            state<= NSPedGreen;
                            count <= X"0";
             CarNSClear <= '0';

                        else
                            state<=NSGreenIdle;

                        end if;


              when NSPedGreen =>             --NSPedGreen  State
                  if (count < COUNT10) then
                            state<= NSPedGreen;
                            count <= count + 1;
                                  PedNSClear <= '1';
                        else
                            state<=NSGreenIdle;
                            count <= X"0";
                                  PedNSClear <= '0';
        end if;

              when NSAmber =>                                 --NSAmber State
                  if (count < COUNT3) then
                            state<= NSAmber;
                            count <= count + 1;
                        else
                            state<=NSRed;
                            count <= X"0";
        end if;

              when NSRed =>                                   --NSRed State
                  if (count < COUNT2) then
                            state<= NSRed;
                            count <= count + 1;
                        else
```

```
                                    if(PedEW ='1') then
                                    state<=EWPedGreen;
                                    count <= X"0";

                                    else
                               state<=EWGreenMin;
                               count <= X"0";
                                    end if;
             end if;

                        when EWGreenMin =>                          --EWGreenMin
State
                        if (count < COUNT10) then
                               state<= EWGreenMin;
                               count <= count + 1;
                                    CarEWClear <= '1';
                           else
                               state<=EWGreenIdle;
                               count <= X"0";
             end if;

                 when others =>
                        state<=EWGreenIdle;

                        end case;
   end if;
 end process SynchronousProcess;

 --OUTPUTS------
 CombinationalProcess: --Not dependant on clock
 process(state)
 begin

      case state is

     when EWGreenIdle =>
             LightsEW <= "10";
                LightsNS <= "00";

       when EWPedGreen =>
             LightsEW <= "11";
                LightsNS <= "00";

       when EWAmber =>
             LightsEW <= "01";
                LightsNS <= "00";

       when EWRed =>
             LightsEW <= "00";
                LightsNS <= "00";

       when NSGreenMin =>
             LightsEW <= "00";
                LightsNS <= "10";

       when NSGreenIdle =>
             LightsEW <= "00";
                LightsNS <= "10";

      when NSPedGreen =>
             LightsEW <= "00";
                LightsNS <= "11";

       when NSAmber =>
```

```
            LightsEW <= "00";
               LightsNS <= "01";

        when NSRed =>
            LightsEW <= "00";
               LightsNS <= "00";

        when EWGreenMin =>
            LightsEW <= "10";
               LightsNS <= "00";

        when others =>
            LightsEW <="10";
                  LightsNS <="00";
      end case;
 end process CombinationalProcess;

 LEDs <=count; --OUTPUT FOR COUNTER

 --OUTPUT TO BE USED BY INPUT SYNC MEMORY MODULE
 PedNSClr <= PedNSClear;
 PedEWClr <= PedEWClear;
 CarNSClr <= CarNSClear;
 CarEWClr <= CarEWClear;

end Behavioral;
```

3. TOP LEVEL TRAFFIC MODULE

```
----------------------------------------------------------------------------------
--
-- Company:
-- Engineer:
--
-- Create Date:    15:45:15 12/10/2021
-- Design Name:
-- Module Name:    StateMachine - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TrafficTopLevel is
    Port ( Clock : in  STD_LOGIC;
           Reset : in  STD_LOGIC;

          -- Car and pedestrian buttons
                       CarEW : in STD_LOGIC; -- Car on EW road
                       CarNS : in STD_LOGIC; -- Car on NS road
                       PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing
NS road)
                       PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing
EW road)

                       -- Light control
                       LightsEW : out STD_LOGIC_VECTOR (1 downto 0); --
controls EW lights
                       LightsNS : out STD_LOGIC_VECTOR (1 downto 0); --
controls NS lights
                       CarEWSync : out STD_LOGIC; -- Car on EW road
                       CarNSSync : out STD_LOGIC; -- Car on NS road
                       PedEW_Mem : out STD_LOGIC; -- Pedestrian moving EW
(crossing NS road)
                       PedNS_Mem : out STD_LOGIC; -- Pedestrian moving NS
(crossing EW road)
                       LEDs     : out STD_LOGIC_VECTOR (3 downto 0)   --
controls LED Lights

                       );
end TrafficTopLevel;

 architecture Behavioral of TrafficTopLevel is

--COMBINE INPUT SYNC MEMORY MODULE AND STATE MACHINE MODULE IN THE TRAFFIC TTOP
LEVEL MODULE
    COMPONENT Input_sync_memory
    PORT(
         Reset : IN  std_logic;
         Clock : IN  std_logic;
```

```
         CarEW : IN  std_logic;
         CarNS : IN  std_logic;
         PedEW : IN  std_logic;
         PedNS : IN  std_logic;
         CarEW_Sync : OUT  std_logic;--Provide value to be used by CarEW signal
in State Machine module
         CarNS_Sync : OUT  std_logic;--Provide value to be used by CarNS signal
in State Machine module
         PedEW_Mem : OUT  std_logic; --Provide value to be used by PedEW signal
in State Machine module
         PedNS_Mem : OUT  std_logic; --Provide value to be used by PedNS signal
in State Machine module
         PedEW_clr : IN  std_logic;  --Clear PedEW Memory
         PedNS_clr : IN  std_logic;  --Clear PedNS Memory
                  CarEW_clr : in  STD_LOGIC;  --Clear EW traffic
         CarNS_clr : in  STD_LOGIC   --Clear EW traffic
        );
    END COMPONENT;


    COMPONENT StateMachine
    PORT(
         Clock : IN  std_logic;
         Reset : IN  std_logic;
         CarEW : IN  std_logic;
         CarNS : IN  std_logic;
                  PedEW : in STD_LOGIC; -- Pedestrian moving EW (crossing NS
road)
                  PedNS : in STD_LOGIC; -- Pedestrian moving NS (crossing EW
road)

         LightsEW : OUT  std_logic_vector(1 downto 0);
         LightsNS : OUT  std_logic_vector(1 downto 0);
                  LEDs     : out STD_LOGIC_VECTOR (3 downto 0);   -- controls
LED Lights
                  PedEWClr : out STD_LOGIC; --Provide value to be used by
PedEW_clr clear signal in input sync Memory module
                  PedNSClr : out STD_LOGIC; --Provide value to be used by
PedNS_clr clear signal in input sync Memory module
                  CarEWClr : out STD_LOGIC; --Provide value to be used by
CarEW_clr clear signal in input sync Memory module
                  CarNSClr : out STD_LOGIC  --Provide value to be used by
CarNS_clr clear signal in input sync Memory module
        );
    END COMPONENT;
--Wires to be used by components to transfer signals
signal CarEWshadow,CarNSshadow,PedEWshadow,PedNSshadow: STD_LOGIC;
signal PedEWClear,PedNSClear,CarEWClear,CarNSClear : STD_LOGIC;
begin

--COMBINING THE COMPONENT WIRES
   InputMemorySync: Input_sync_memory PORT MAP (
                     Reset => Reset,
                     Clock => Clock,
                     CarEW => CarEW,
                     CarNS => CarNS,
                     PedEW => PedEW,
                     PedNS => PedNS,
                     CarEW_Sync => CarEWshadow,
                     CarNS_Sync => CarNSshadow,
                     PedEW_Mem => PedEWshadow,
                     PedNS_Mem => PedNSshadow,
                     PedEW_clr => PedEWClear,
                     PedNS_clr => PedNSClear,
```

```
                                    CarEW_clr => CarEWClear,
                        CarNS_clr => CarNSClear
        );

              MainSM: StateMachine PORT MAP (
                        Clock => Clock,
                        Reset => Reset,
                        CarEW => CarEWShadow,
                        CarNS => CarNSshadow,
                                 PedEW => PedEWshadow,
                                 PedNS => PedNSshadow,
                        LightsEW => LightsEW,
                        LightsNS => LightsNS,
                                          LEDs =>LEDs,
                                          PedEWClr => PedEWClear,
                                 PedNSClr => PedNSClear,
                                          CarEWClr => CarEWClear,
                        CarNSClr => CarNSClear
        );
 --OUTPUTS AS DISPLAYED IN SIMULATION
CarEWSync <= CarEWshadow;
CarNSSync <= CarNSshadow;
PedEW_Mem <= PedEWshadow;
PedNS_Mem <= PedNSshadow;
end Behavioral;
```

```
             ISE SYNTHESIS REPORT
================================================================================
Release 14.7 - xst P.20131013 (nt64)
Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp


Total REAL time to Xst completion: 1.00 secs
Total CPU time to Xst completion: 0.82 secs

--> Parameter xsthdpdir set to xst


Total REAL time to Xst completion: 1.00 secs
Total CPU time to Xst completion: 0.82 secs

--> Reading design: TrafficTopLevel.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report

================================================================================
*                      Synthesis Options Summary                               *
================================================================================
---- Source Parameters
Input File Name                    : "TrafficTopLevel.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO

---- Target Parameters
Output File Name                   : "TrafficTopLevel"
Output Format                      : NGC
Target Device                      : CoolRunner2 CPLDs

---- Source Options
Top Module Name                    : TrafficTopLevel
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
Mux Extraction                     : Yes
Resource Sharing                   : YES

---- Target Options
Add IO Buffers                     : YES
MACRO Preserve                     : YES
XOR Preserve                       : YES
Equivalent register Removal        : YES

---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Keep Hierarchy                     : Yes
Netlist Hierarchy                  : As_Optimized
```

```
RTL Output                          : Yes
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : Maintain
Verilog 2001                        : YES

---- Other Options
Clock Enable                        : YES
wysiwyg                             : NO

=========================================================================


=========================================================================
*                            HDL Compilation                            *
=========================================================================
Compiling vhdl file "C:/Users/HP ENVY/Xilinx/LAB5/Input_sync_memory.vhd" in
Library work.
Entity <input_sync_memory> compiled.
Entity <input_sync_memory> (Architecture <behavioral>) compiled.
Compiling vhdl file "C:/Users/HP ENVY/Xilinx/LAB5/StateMachine.vhd" in Library
work.
Entity <statemachine> compiled.
Entity <statemachine> (Architecture <behavioral>) compiled.
Compiling vhdl file "C:/Users/HP ENVY/OneDrive - Swinburne University Of
Technology Sarawak Campus/School Units/Digital Electronics Design/LAB/Lab
5/Project VHDL Files/TrafficTopLevel.vhd" in Library work.
Entity <traffictoplevel> compiled.
Entity <traffictoplevel> (Architecture <behavioral>) compiled.

=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for entity <TrafficTopLevel> in library <work> (architecture
<behavioral>).

Analyzing hierarchy for entity <Input_sync_memory> in library <work>
(architecture <behavioral>).

Analyzing hierarchy for entity <StateMachine> in library <work> (architecture
<behavioral>).


=========================================================================
*                             HDL Analysis                              *
=========================================================================
Analyzing Entity <TrafficTopLevel> in library <work> (Architecture
<behavioral>).
Entity <TrafficTopLevel> analyzed. Unit <TrafficTopLevel> generated.

Analyzing Entity <Input_sync_memory> in library <work> (Architecture
<behavioral>).
Entity <Input_sync_memory> analyzed. Unit <Input_sync_memory> generated.

Analyzing Entity <StateMachine> in library <work> (Architecture <behavioral>).
Entity <StateMachine> analyzed. Unit <StateMachine> generated.


=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...
```

```
Synthesizing Unit <Input_sync_memory>.
    Related source file is "C:/Users/HP ENVY/Xilinx/LAB5/Input_sync_memory.vhd".
    Found 1-bit register for signal <PedNS_Mem>.
    Found 1-bit register for signal <CarNS_Sync>.
    Found 1-bit register for signal <CarEW_Sync>.
    Found 1-bit register for signal <PedEW_Mem>.
    Summary:
      inferred   4 D-type flip-flop(s).
Unit <Input_sync_memory> synthesized.


Synthesizing Unit <StateMachine>.
    Related source file is "C:/Users/HP ENVY/Xilinx/LAB5/StateMachine.vhd".
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------------
    | States             | 10                                             |
    | Transitions        | 26                                             |
    | Inputs             | 7                                              |
    | Outputs            | 14                                             |
    | Clock              | Clock                     (rising_edge)        |
    | Reset              | Reset                     (positive)           |
    | Reset type         | asynchronous                                   |
    | Reset State        | ewgreenidle                                    |
    | Power Up State     | ewgreenidle                                    |
    | Encoding           | automatic                                      |
    | Implementation     | automatic                                      |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <CarEWClear>.
    Found 1-bit register for signal <CarNSClear>.
    Found 4-bit register for signal <count>.
    Found 4-bit adder for signal <count$share0000> created at line 75.
    Found 1-bit register for signal <PedEWClear>.
    Found 1-bit register for signal <PedNSClear>.
    Found 4-bit comparator less for signal <state$cmp_lt0000> created at line
93.
    Found 4-bit comparator less for signal <state$cmp_lt0001> created at line
104.
    Found 4-bit comparator less for signal <state$cmp_lt0002> created at line
113.
    Summary:
      inferred   1 Finite State Machine(s).
      inferred   4 D-type flip-flop(s).
      inferred   1 Adder/Subtractor(s).
      inferred   3 Comparator(s).
Unit <StateMachine> synthesized.


Synthesizing Unit <TrafficTopLevel>.
    Related source file is "C:/Users/HP ENVY/OneDrive - Swinburne University Of
Technology Sarawak Campus/School Units/Digital Electronics Design/LAB/Lab
5/Project VHDL Files/TrafficTopLevel.vhd".
Unit <TrafficTopLevel> synthesized.

INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some
arithmetic operations in this design can share the same physical resources for
reduced device utilization. For improved clock frequency you may try to disable
resource sharing.

=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 4-bit adder                                           : 1
```

```
# Registers                                             : 9
 1-bit register                                        : 8
 4-bit register                                        : 1
# Comparators                                           : 3
 4-bit comparator less                                 : 3

=========================================================================

=========================================================================
*                        Advanced HDL Synthesis                         *
=========================================================================

Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <MainSM/state/FSM> on signal <state[1:10]> with one-hot encoding.
---------------------------
 State       | Encoding
---------------------------
 ewgreenidle | 0000000001
 ewamber     | 0000000010
 ewred       | 0000001000
 nsgreenmin  | 0000100000
 nspedgreen  | 0000010000
 nsgreenidle | 0001000000
 nsamber     | 0010000000
 nsred       | 0100000000
 ewgreenmin  | 1000000000
 ewpedgreen  | 0000000100
---------------------------

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# FSMs                                                  : 1
# Adders/Subtractors                                    : 1
 4-bit adder                                           : 1
# Registers                                             : 8
 Flip-Flops                                            : 8

=========================================================================

=========================================================================
*                         Low Level Synthesis                           *
=========================================================================

Optimizing unit <TrafficTopLevel> ...

Optimizing unit <Input_sync_memory> ...

Optimizing unit <StateMachine> ...
  implementation constraint: INIT=r  : state_FSM_FFd1
  implementation constraint: INIT=r  : state_FSM_FFd2
  implementation constraint: INIT=r  : state_FSM_FFd3
  implementation constraint: INIT=r  : state_FSM_FFd4
  implementation constraint: INIT=r  : state_FSM_FFd5
  implementation constraint: INIT=r  : state_FSM_FFd6
  implementation constraint: INIT=r  : state_FSM_FFd7
  implementation constraint: INIT=r  : state_FSM_FFd8
  implementation constraint: INIT=r  : state_FSM_FFd9
  implementation constraint: INIT=s  : state_FSM_FFd10

=========================================================================
*                           Partition Report                            *
=========================================================================
```

```
Partition Implementation Status
-------------------------------

  No Partitions were found in this design.

-------------------------------

=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : TrafficTopLevel.ngr
Top Level Output File Name         : TrafficTopLevel
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : Yes
Target Technology                  : CoolRunner2 CPLDs
Macro Preserve                     : YES
XOR Preserve                       : YES
Clock Enable                       : YES
wysiwyg                            : NO

Design Statistics
# IOs                              : 18

Cell Usage :
# BELS                             : 197
#      AND2                        : 69
#      AND3                        : 7
#      AND4                        : 3
#      INV                         : 73
#      OR2                         : 30
#      OR3                         : 12
#      XOR2                        : 3
# FlipFlops/Latches                : 22
#      FDC                         : 15
#      FDCE                        : 6
#      FDP                         : 1
# IO Buffers                       : 18
#      IBUF                        : 6
#      OBUF                        : 12
=========================================================================


Total REAL time to Xst completion: 7.00 secs
Total CPU time to Xst completion: 6.97 secs

-->

Total memory usage is 4497352 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    1 (   0 filtered)
```