SPH 336 **COMPUTING LABORATORY II**

PROJECT REPORT

PROJECT TITLE: TRAFFIC LIGHTS CONTROL SYSTEM

NAME: WANGERE JOSEPH NGIGI

REG. NUMBER: I39/2135/2012

1.COLLABORATAORS

- Alvin Mutisya -I39/2142/2012
- Gideon Kimaiyo -I39/2138/2012

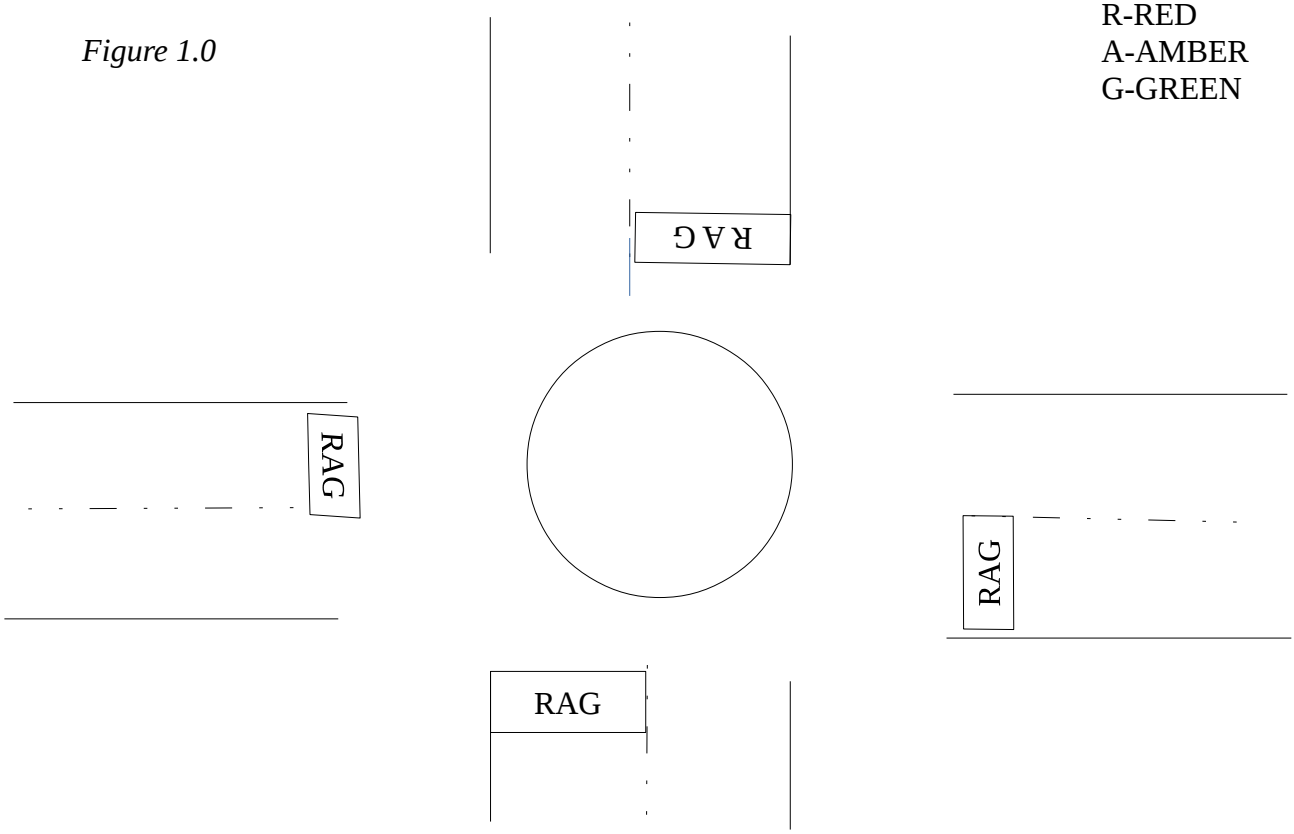
2.OBJECTIVES

a) To model the functionality of traffic lights on roundabout using hierarchically concurrent Finite State Machine model.
b) To create a model of computation for the traffic light system and simulate its functioning
c) To implement the model of computation using systemc
d) To run a simulation and analyze the terminal output as well as the vcd traces

3.INTRODUCTION:

In urban centres, it is a common scenario to find an overflow of traffic involving vehicles travelling in and out of a town. Without proper control of such traffic, the situation would be chaotic. In view of this, a traffic lights control system was modelled, taking into account a unique case of a four-way round about, each way with two lanes.

Figure 1.0



The traffic light system was modelled based on the following rules:

a) Globally

-No two green lights can be on at the same time

-Green light rotates from one traffic light to the next in a clockwise direction

b) Locally

-Amber lasts for 0.5 minutes and green lasts for two minutes

-Red follows Amber and Green follows Red after trigger

Hierarchically concurrent Finite State Machine model of computation was used.

MODEL OF COMPUTATION

fig.2

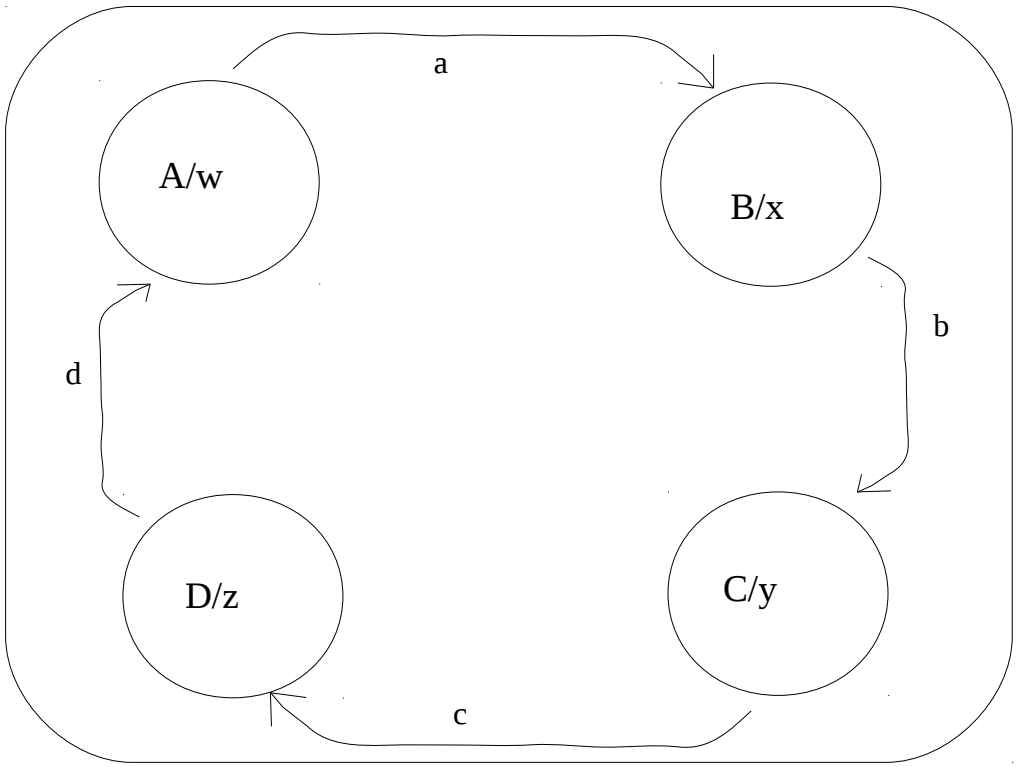
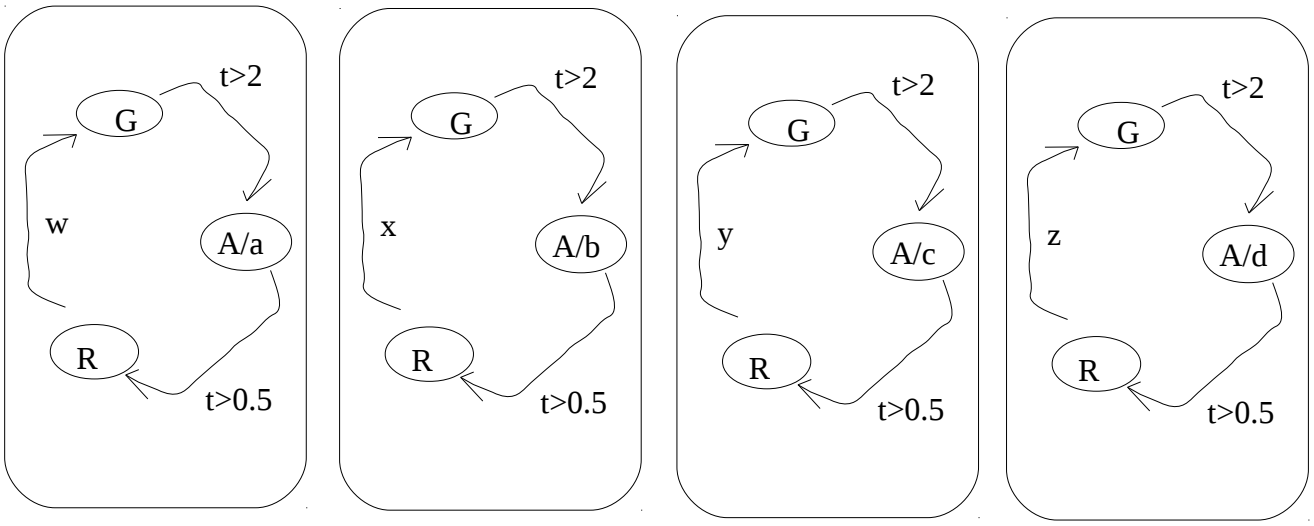


fig.3



KEY:
w,x,y,z -These are global variables
R -Red
A -Amber
G -Green

4.TOOLS USED

The following tools were used:
-A computer for testing the solution
-Eclipse Keppler IDE for code development and debugging
-SystemC for hardware simulation

5.METHODOLOGY

The solution to the problem was subdivided into several phases. The first phase involved understanding the problem and modelling the solution using a model of computation as shown in figures 1 and 2. This was necessary to save on time that could have otherwise been wasted trying to solve the problem at a higher level of abstraction. With the help of the model of computation, the problem was broken into smaller solvable portions.

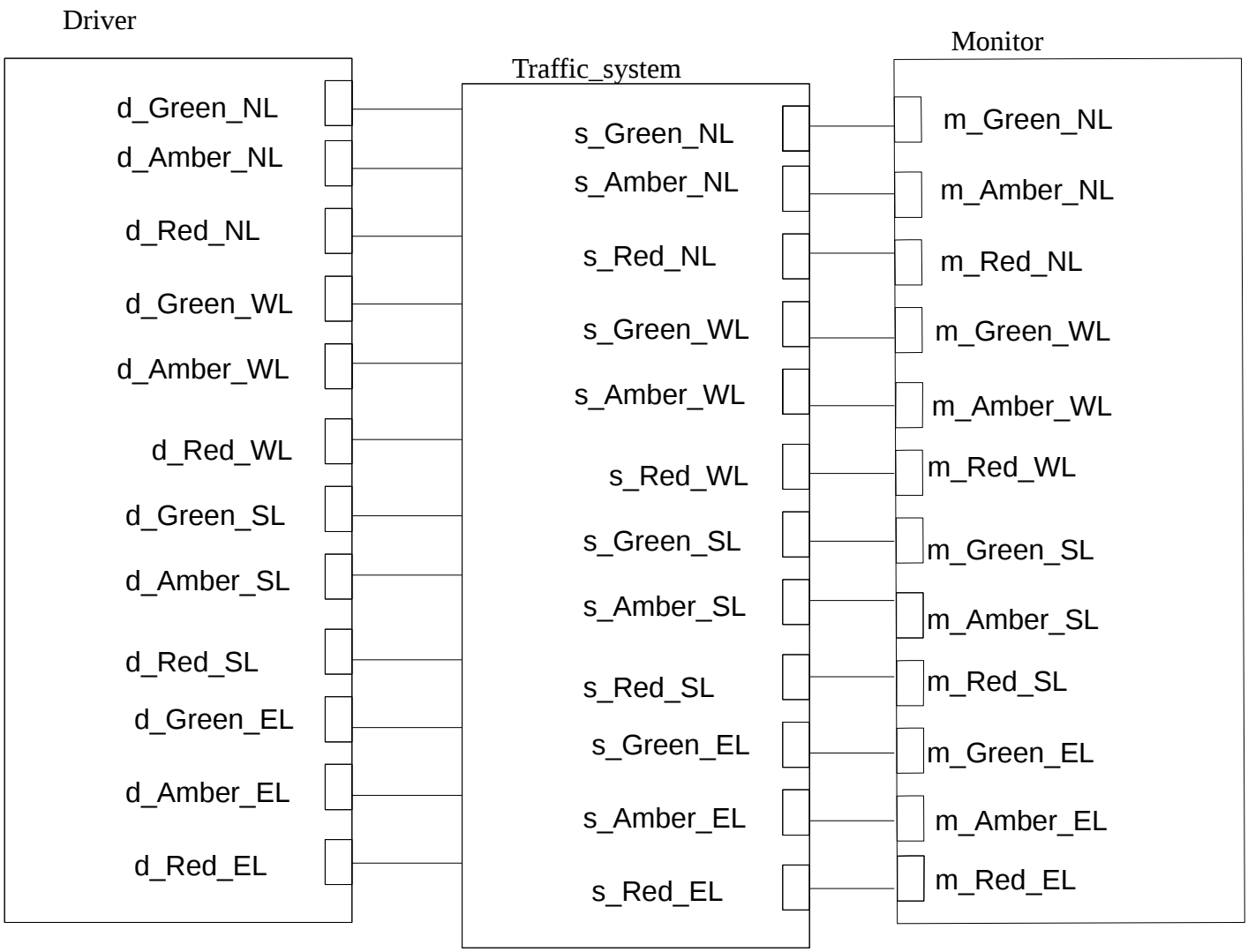
The system was decomposed into four states, taking into consideration their transitions. The four state machines were used to represent all the four Traffic lights for the four lanes. The global variables w,x,y,z were used as transition elements from one state to the next, since each state machine has three state, that is, Red, Amber and Green. According to the model, the first traffic light starts in Green state which lasts for a duration of about 2 seconds, after which the second traffic light is triggered.Meanwhile, no other traffic light is in its green state.

The second phase was to develop software solution for the problem. A model of computation was also employed to make coding simpler and faster. The model of computation entailed three modules: That is, a driver module to drive the 12 inputs into the system, a traffic_system module to process the inputs and finally a monitor module to display the processed output.

The inputs and outputs both were of boolean type, implying they can either be true or false denoted by binary digits 1 for true and 0 for false.

The outputs displayed by the monitor were also Twelve in total since there are four lanes and each lane has a traffic light consisting of Red,Amber and green lights.

Fig.4



Driver module- This was used to drive in the inputs into the traffic light system for processing.It has no input signals itself.

Traffic system module -This was used to transmit processed inputs in form of signals to the monitor.It has 12 input signals and 12 output signals as well.

Monitor module-This also has 12 input signals.Its function is to monitor both input and output signals from the traffic light system module.

6.RESULTS

Upon implementation of the above model of computation using systemc, the following trace diagram was obtained, which indicates the resultant timing diagram for all the signals. The console output was also obtained as shown below.

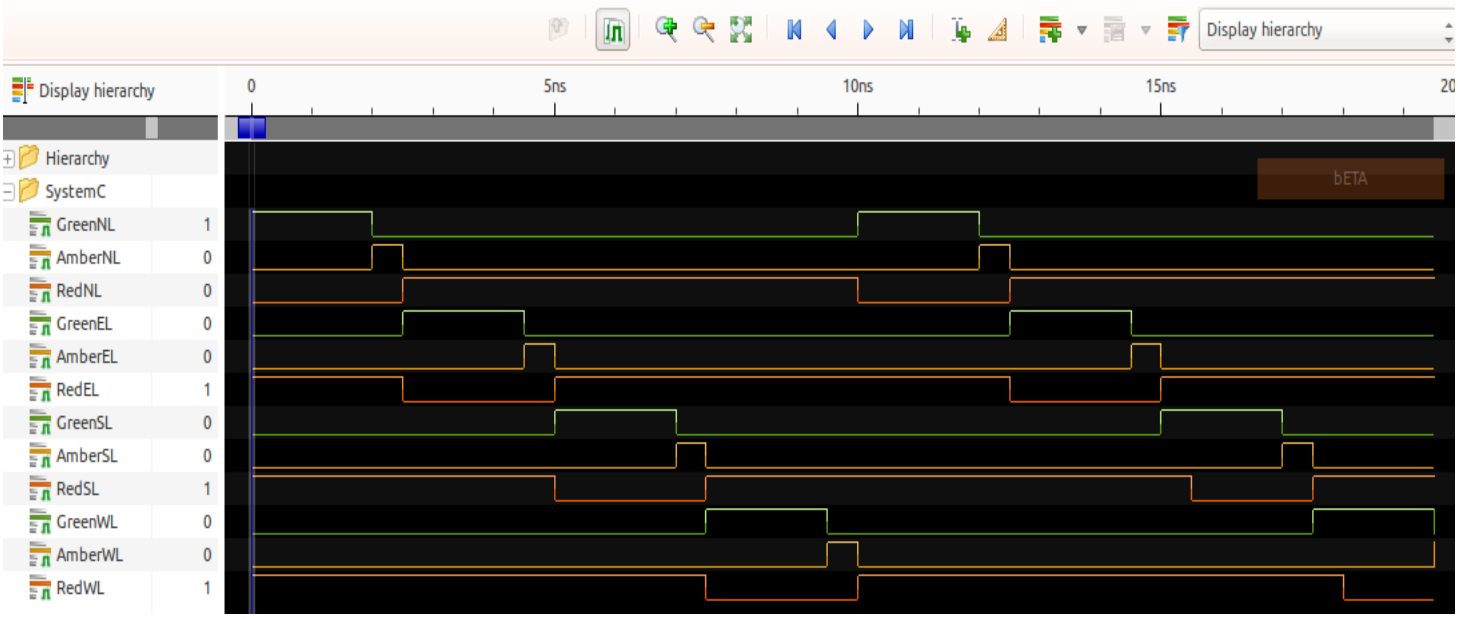


Fig.5: The trace file obtained after implementing and running the program on systemc

Each of the signals represents a specific colour of the traffic lights I.e red,amber and green. From the trace file, it can be observed that no two greens are on at the same time, which was clearly stated in the algorithm at the early stages of design.Part of the console output was obtained as shown below.For simulation purpose, very small time interval was used and the units were as well small.

```
At 0 s state is: GreenNL: AmberNL: RedNL: 100
At 0 s state is: GreenEL: AmberEL: RedEL: 001
At 0 s state is: GreenSL: AmberSL: RedSL: 001
At 0 s state is: GreenWL: AmberWL: RedWL: 001
WARNING: Default time step is used for VCD tracing.
At 2 ns state is: GreenNL: AmberNL: RedNL: 010
At 2 ns state is: GreenEL: AmberEL: RedEL: 001
At 2 ns state is: GreenSL: AmberSL: RedSL: 001
At 2 ns state is: GreenWL: AmberWL: RedWL: 001
At 2500 ps state is: GreenNL: AmberNL: RedNL: 001
At 2500 ps state is: GreenEL: AmberEL: RedEL: 100
At 2500 ps state is: GreenSL: AmberSL: RedSL: 001
At 2500 ps state is: GreenWL: AmberWL: RedWL: 001
At 4500 ps state is: GreenNL: AmberNL: RedNL: 001
At 4500 ps state is: GreenEL: AmberEL: RedEL: 010
At 4500 ps state is: GreenSL: AmberSL: RedSL: 001
At 4500 ps state is: GreenWL: AmberWL: RedWL: 001
```

Fig.5.1 A section of the console output obtained from the executed program

7.CONCLUSION

From the results observed ,the objectives of the project were achieved.The functionality of traffic light control system was modelled using hierarchically concurrent finite state machine model.The model of computation for the traffic light system was simulated and observed to work as per the expectations.The model of computation was also implemented using systemc and the terminal output analyzed as well as

the vcd traces to simulate the hardware behaviour of the system.The simulation results are as shown from the screenshots in figure 5 above.

8.APPENDIX

The code used to implement the model of computation is shown below:

Driver module

```
/*
 * driver.h
 *
 *  Created on: Feb 4, 2015
 *      Author: joe--ngigi
 */
#ifndef DRIVER_H_
#define DRIVER_H_
#include "systemc.h"

SC_MODULE(driver)
{
      sc_out<bool>
d_greenNL,d_amberNL,d_redNL,d_greenEL,d_amberEL,d_redEL,d_greenSL,d_amberSL,d_redSL;
      sc_out<bool> d_greenWL,d_amberWL,d_redWL;//inputs

//FOR NORTH LANE
void driveGreenNL()// this is a process
{
      for(int i=0; i<10;i++)
      {
            d_greenNL.write((bool)true);
            wait(2,SC_NS);
            d_greenNL.write((bool)false);
            wait(8,SC_NS);
      }
}

void driveAmberNL()// this is a process
{
      d_amberNL.write((bool)false);
      wait(2,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_amberNL.write((bool)true);
            wait(0.5,SC_NS);
            d_amberNL.write((bool)false);
            wait(9.5,SC_NS);
      }
}

void driveRedNL()// this is a process
{
      d_redNL.write((bool)false);
      wait(2.5,SC_NS);
      for(int i=0; i<10;i++)
      {
```

```
            d_redNL.write((bool)true);
            wait(7.5,SC_NS);
            d_redNL.write((bool)false);
            wait(2.5,SC_NS);
      }
}

//FOR EAST LANE
void driveGreenEL()// this is a process
{
      d_greenEL.write((bool)false);
      wait(2.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_greenEL.write((bool)true);
            wait(2,SC_NS);
            d_greenEL.write((bool)false);
            wait(8,SC_NS);
      }
}

void driveAmberEL()// this is a process
{
      d_amberEL.write((bool)false);
      wait(4.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_amberEL.write((bool)true);
            wait(0.5,SC_NS);
            d_amberEL.write((bool)false);
            wait(9.5,SC_NS);
      }
}

void driveRedEL()// this is a process
{
      d_redEL.write((bool)true);
      wait(2.5,SC_NS);
      d_redEL.write((bool)false);
      wait(2.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_redEL.write((bool)true);
            wait(7.5,SC_NS);
            d_redEL.write((bool)false);
            wait(2.5,SC_NS);
      }
}

//FOR SOUTH LANE
void driveGreenSL()// this is a process
{
      d_greenSL.write((bool)false);
      wait(5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_greenSL.write((bool)true);
            wait(2,SC_NS);
```

```
            d_greenSL.write((bool)false);
            wait(8,SC_NS);
      }
}

void driveAmberSL()// this is a process
{
      d_amberSL.write((bool)false);
      wait(7,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_amberSL.write((bool)true);
            wait(0.5,SC_NS);
            d_amberSL.write((bool)false);
            wait(9.5,SC_NS);
      }
}

void driveRedSL()// this is a process
{
      d_redSL.write((bool)true);
      wait(5,SC_NS);
      d_redSL.write((bool)false);
      wait(2.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_redSL.write((bool)true);
            wait(8,SC_NS);
            d_redSL.write((bool)false);
            wait(2,SC_NS);
      }
}

//FOR WEST LANE
void driveGreenWL()// this is a process
{
      d_greenWL.write((bool)false);
      wait(7.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_greenWL.write((bool)true);
            wait(2,SC_NS);
            d_greenWL.write((bool)false);
            wait(8,SC_NS);
      }
}

void driveAmberWL()// this is a process
{
      d_amberWL.write((bool)false);
      wait(9.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_amberWL.write((bool)true);
            wait(0.5,SC_NS);
            d_amberWL.write((bool)false);
            wait(9.5,SC_NS);
      }
```

```
}

void driveRedWL()// this is a process
{
      d_redWL.write((bool)true);
      wait(7.5,SC_NS);
      d_redWL.write((bool)false);
      wait(2.5,SC_NS);
      for(int i=0; i<10;i++)
      {
            d_redWL.write((bool)true);
            wait(8,SC_NS);
            d_redWL.write((bool)false);
            wait(2,SC_NS);
      }
}

SC_CTOR(driver)
{
      SC_THREAD (driveGreenNL);//process is called here
      SC_THREAD (driveAmberNL);
      SC_THREAD (driveRedNL);

      SC_THREAD (driveGreenEL);//process is called here
      SC_THREAD (driveAmberEL);
      SC_THREAD (driveRedEL);

      SC_THREAD (driveGreenSL);//process is called here
      SC_THREAD (driveAmberSL);
      SC_THREAD (driveRedSL);

      SC_THREAD (driveGreenWL);//process is called here
      SC_THREAD (driveAmberWL);
      SC_THREAD (driveRedWL);
}
};




#endif
```

Traffic system module system

```
#ifndef TRAFFIC_H_
#define TRAFFIC_H_
#include "systemc.h"

SC_MODULE (traffic_Lights)
{
      sc_in<bool>greenNL, amberNL, redNL; //signals
```

```
sc_in<bool>greenEL, amberEL, redEL;
sc_in<bool>greenSL, amberSL, redSL;
sc_in<bool>greenWL, amberWL, redWL;

void trafficNorthLane ()
{
      if (greenNL==true)
      {
            amberNL==false;
            redNL==false;
      }
      else if (amberNL==true)
      {
            greenNL==false;
            redNL==false;
      }
      else
      {
            redNL==true;
            greenNL==false;
            amberNL==false;
      }
}

void trafficEastLane ()
{
      if (greenEL==true)
      {
            amberEL==false;
            redEL==false;
      }
      else if (amberEL==true)
      {
            greenEL==false;
            redEL==false;
      }
      else
      {
            redEL==true;
            greenEL==false;
            amberEL==false;
      }
}

void trafficSouthLane ()
{
      if (greenSL==true)
      {
            amberSL==false;
            redSL==false;
      }
      else if (amberSL==true)
      {
            greenSL==false;
            redSL==false;
      }
      else
      {
            redSL==true;
```

```
                greenSL==false;
                amberSL==false;
        }
    }

    void trafficWestLane ()
    {
        if (greenWL==true)
        {
            amberWL==false;
            redWL==false;
        }
        else if (amberWL==true)
        {
            greenWL==false;
            redWL==false;
        }
        else
        {
            redWL==true;
            greenWL==false;
            amberWL==false;
        }
    }

SC_CTOR (traffic_Lights)
{
    SC_METHOD (trafficNorthLane);
    sensitive<<greenNL<<amberNL<<redNL;
    SC_METHOD (trafficEastLane);
    sensitive<<greenEL<<amberEL<<redEL;
    SC_METHOD (trafficSouthLane);
    sensitive<<greenSL<<amberSL<<redSL;
    SC_METHOD (trafficWestLane);
    sensitive<<greenWL<<amberWL<<redWL;
}

};

#endif
```

Monitor module

```
#ifndef MONITOR_H_
#define MONITOR_H_
#include "systemc.h"

SC_MODULE(monitor)
{
    sc_in<bool> m_greenNL, m_amberNL, m_redNL; //monitor signals
    sc_in<bool> m_greenEL, m_amberEL, m_redEL;
    sc_in<bool> m_greenSL, m_amberSL, m_redSL;
    sc_in<bool> m_greenWL, m_amberWL, m_redWL;

    void monitorLane()
    {
```

```
            cout <<"At "<<sc_time_stamp()<<" state is: GreenNL: AmberNL: RedNL:
"<<m_greenNL
                        <<m_amberNL<<m_redNL<<endl;
            cout <<"At "<<sc_time_stamp()<<" state is: GreenEL: AmberEL: RedEL:
"<<m_greenEL
                        <<m_amberEL<<m_redEL<<endl;
            cout <<"At "<<sc_time_stamp()<<" state is: GreenSL: AmberSL: RedSL:
"<<m_greenSL
                        <<m_amberSL<<m_redSL<<endl;
            cout <<"At "<<sc_time_stamp()<<" state is: GreenWL: AmberWL: RedWL:
"<<m_greenWL
                        <<m_amberWL<<m_redWL<<endl;
      }

SC_CTOR(monitor)
{
      SC_METHOD(monitorLane);
      sensitive<< m_greenNL<<m_amberNL<<m_redNL;
      sensitive<< m_greenEL<<m_amberEL<<m_redEL;
      sensitive<< m_greenSL<<m_amberSL<<m_redSL;
      sensitive<< m_greenWL<<m_amberWL<<m_redWL;
}
};

#endif /* MONITOR_H_ */
```

Main module

```
#include "systemc.h"
#include "traffic.h" //including header files
#include "monitor.h"
#include "driver.h"

int sc_main(int argc, char*argv[])
{
      sc_signal<bool> s_greenNL, s_amberNL, s_redNL; //signals, North Lane
      sc_signal<bool> s_greenEL, s_amberEL, s_redEL; //signals, East Lane
      sc_signal<bool> s_greenSL, s_amberSL, s_redSL; //signals, South Lane
      sc_signal<bool> s_greenWL, s_amberWL, s_redWL; //signals, west Lane

      sc_trace_file *tf;                              //trace file(pointer), to trace
signals
      tf=sc_create_vcd_trace_file("Display");         //create VCD trace. Traffic Light
Display

      sc_trace(tf, s_greenNL, "GreenNL");             //signals to be traced.
      sc_trace(tf, s_amberNL, "AmberNL");
      sc_trace(tf, s_redNL, "RedNL");

      sc_trace(tf, s_greenEL, "GreenEL");
      sc_trace(tf, s_amberEL, "AmberEL");
      sc_trace(tf, s_redEL, "RedEL");

      sc_trace(tf, s_greenSL, "GreenSL");
```

```
sc_trace(tf, s_amberSL, "AmberSL");
sc_trace(tf, s_redSL, "RedSL");

sc_trace(tf, s_greenWL, "GreenWL");
sc_trace(tf, s_amberWL, "AmberWL");
sc_trace(tf, s_redWL, "RedWL");

traffic_Lights tra("traffic");
monitor mon("monitor");
driver dri("driver");

tra.greenNL(s_greenNL);       //below is binding, named
tra.amberNL(s_amberNL);
tra.redNL(s_redNL);

tra.greenEL(s_greenEL);
tra.amberEL(s_amberEL);
tra.redEL(s_redEL);

tra.greenSL(s_greenSL);
tra.amberSL(s_amberSL);
tra.redSL(s_redSL);

tra.greenWL(s_greenWL);
tra.amberWL(s_amberWL);
tra.redWL(s_redWL);


dri.d_greenNL(s_greenNL);
dri.d_amberNL(s_amberNL);
dri.d_redNL(s_redNL);

dri.d_greenEL(s_greenEL);
dri.d_amberEL(s_amberEL);
dri.d_redEL(s_redEL);

dri.d_greenSL(s_greenSL);
dri.d_amberSL(s_amberSL);
dri.d_redSL(s_redSL);

dri.d_greenWL(s_greenWL);
dri.d_amberWL(s_amberWL);
dri.d_redWL(s_redWL);

mon.m_greenNL(s_greenNL);
mon.m_amberNL(s_amberNL);
mon.m_redNL(s_redNL);

mon.m_greenEL(s_greenEL);
mon.m_amberEL(s_amberEL);
mon.m_redEL(s_redEL);

mon.m_greenSL(s_greenSL);
mon.m_amberSL(s_amberSL);
mon.m_redSL(s_redSL);

mon.m_greenWL(s_greenWL);
mon.m_amberWL(s_amberWL);
mon.m_redWL(s_redWL);
```

```
        sc_start(20,SC_NS);             //start simulation, for 100
        sc_close_vcd_trace_file(tf);

        return 0;
}
```