Assignment 1 Report

Team Members:

Shubham Jain(120050002)
Palash Kala(120050010)
Sanket Kanjalkar(120050011)
Royal Jain(120050014)
Vikas Garg(120050017)

Description of the Idea:

1. We consider the signals at two junctions, one at IIT main gate and Pizza hut are independent of each other because we did observe a direct relation between them
2. The traffic on the main road is divided into two one near the main gate and other at the pizza hut junction, and are assumed to be independent
3. There are some paths at each junction which can always be travelled without any hindrance by fellow travellers at other roads, so we don't take these states into account in the construction of a state machine i.e they are not responsible for any changes in the signals at the junction
4. We construct a FSM (Finite State Machine) for each junction
5. FSM consists of all the possible states of traffic signals at each point of time
6. There are some paths which can be allowed to take together, so they are combined to form a group and atmost one group of paths have a green signal at a time
7. We assume any state (comprising of signals of the different groups) should be achieved for at least some minimum time and not more than a specified maximum time
8. Assumption of the Q and S values: The Q values and the S values are boolean values which tells us whether the traffic and speed is above or below certain limit
9. If some vehicle's speed is higher than the limit, it tells us that the traffic is smooth while the high values of Q depicts exactly the opposite
10. We keep a function, which takes into account the values of Q1, Q2 and Q3 and S1,S2 and S3 for each path
11. Suppose for some road there are some more than one options on where to go, then its contribution to the function is given to all the options
12. From point 9, the function is proportional to Q values and inversely proportional to S values
13. This function symbolically determines the level of traffic at each road
14. We keep a timer to keep a note of time before which a state had changed

15. Before any transition from green to red or red to green it goes through a amber transition which should also happen for some minimum specified time
16. In the time where we assume the state should be constant for some time, we do not consider the input for that time because in real life if we don't do this then people will be confused about when the signal would be changed and it may be possible that they are at the mid of the junction and the signal changes to red
17. Again observed from real life that we cannot keep the signal constant for a long time. Consider a case where on one way there is a only one ambulance which has to go and on its obstructing path there is a heavy traffic always. Our assumption would allow the ambulance to pass after certain amount of time

Application of Idea to the Problem:

- A point X is assumed between A and J and a point Y is assumed between B and K

  The groups formed are:

  Main gate:-

  The paths CA, DA, CJ, BF are always possible, hence they are not considered

  g1: AJ, YK

  g2: AF, CF, DF, CK

  g3: EJ, CK, DK

  Pizza hut junction

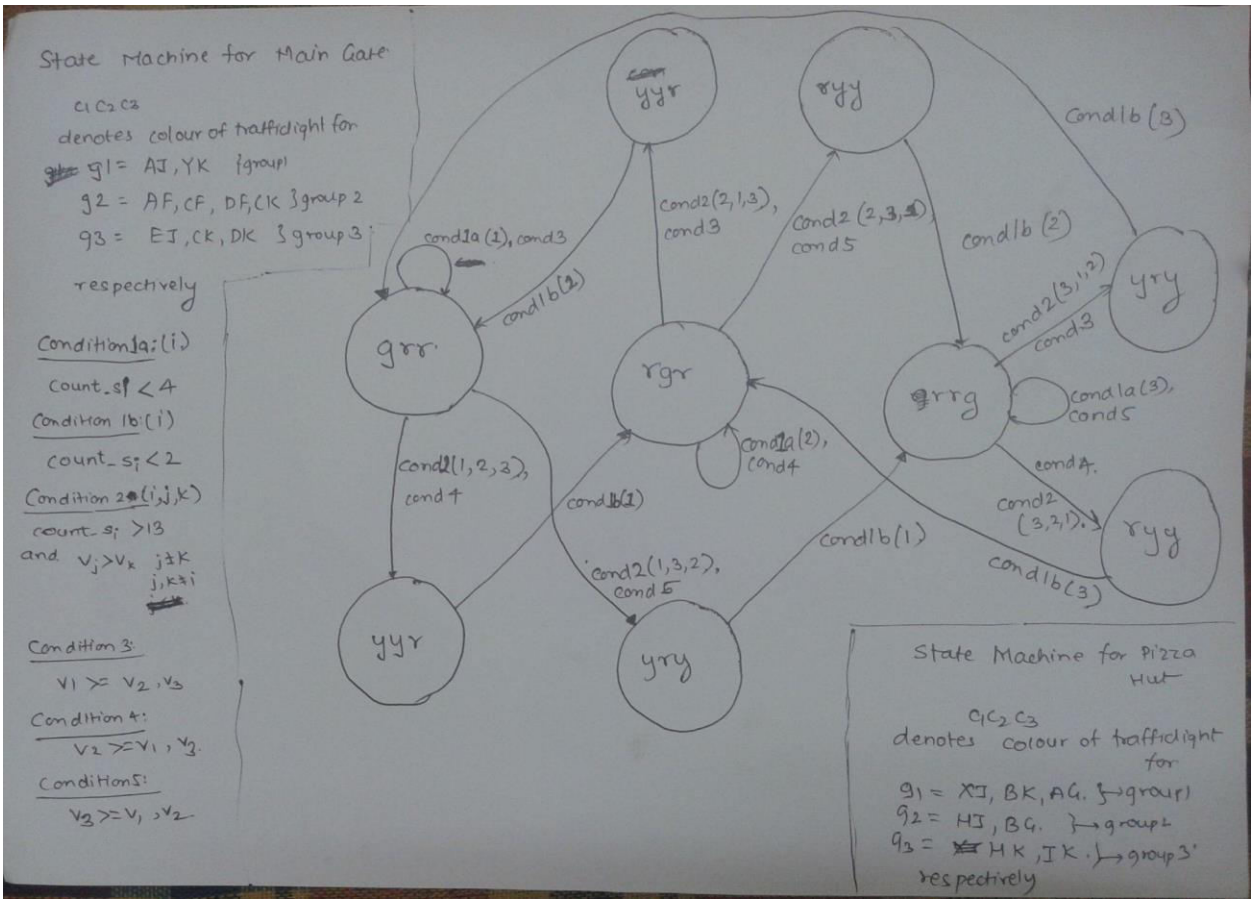
  g1: XJ, BK, AG

  g2: HJ, BG

  g3: HK, IK
- Time period of the clock= 10ns
- Inputs to the finite state machine:
  - Q1-Q9 and S1-S9 for the 9 different possible paths in the state machine
  - Each is a vector of length 3 and corresponds to q1,q2 and q3 and s1, s2 and s3 values for each
- The outputs:
  - There are nine outputs each is a vector of length 2
  - These output shows the signal for each path
    - '10'-green
    - '01'-amber
    - '00'-red
- The minimum time that the FSM should be in a state is 20 ns
- The maximum time is 70 ns
- The time in which it is in amber state is 10 ns

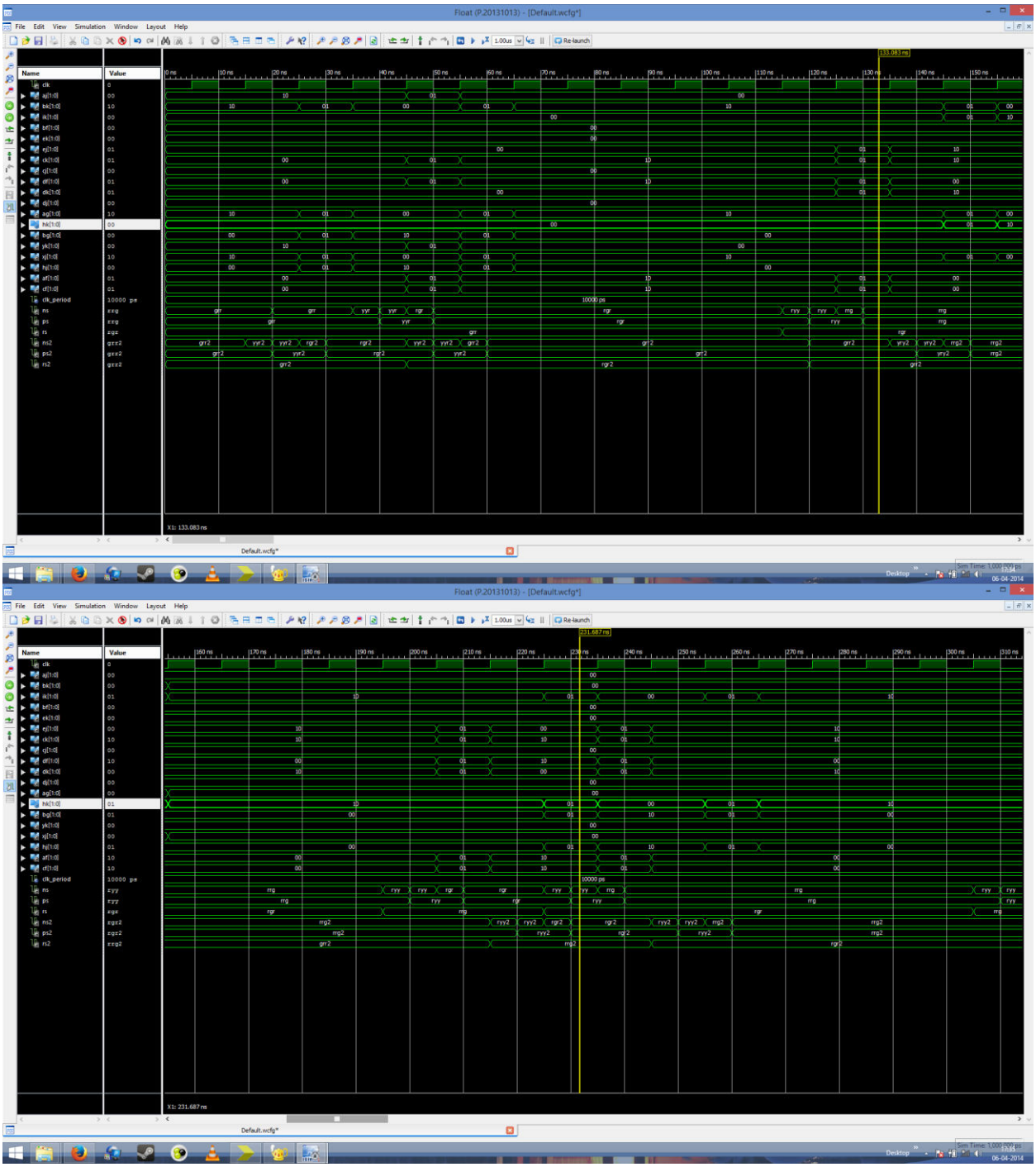
- The preference order if the values of the functions are same is g1>g2>g3 (for both the junctions)
- The function:
    - We calculate the value of each path as 00(q1q2q3s1's2's3')(q2q3s2's3')(q3s3') of 5 bits (from point 12 above)
    - Supply these values to the groups wherever they should be and add them (from point 11)
    - This addition for each group will give symbolically show the level of traffic
    - Example: For g1 add the values for AJ and YK
- States: RS-previous, PS- present, NS- next state
- The time counters:
    - Main gate:
        - g1-s1, g2-s2 and g3-s3
    - Pizza Hut junction:
        - g1-s12, g2-s22, g3-s32
- Now, at every iteration, first check
    - if the time elapsed is less than the 40 ns, then no change.
    - If it is between 40 and 130 ns, then choose the maximum value among the three
    - If it is greater than 130 ns, then choose maximum among other two
- Example: The state grr means that the g1 is green and other two are red (only one can be green at a time)
- For amber transition, two of the three groups will have amber signal and their next state will be define from the previous state
    - For example:
        - grr -> yyr -> rgr
        - rgr -> ryy -> rrg

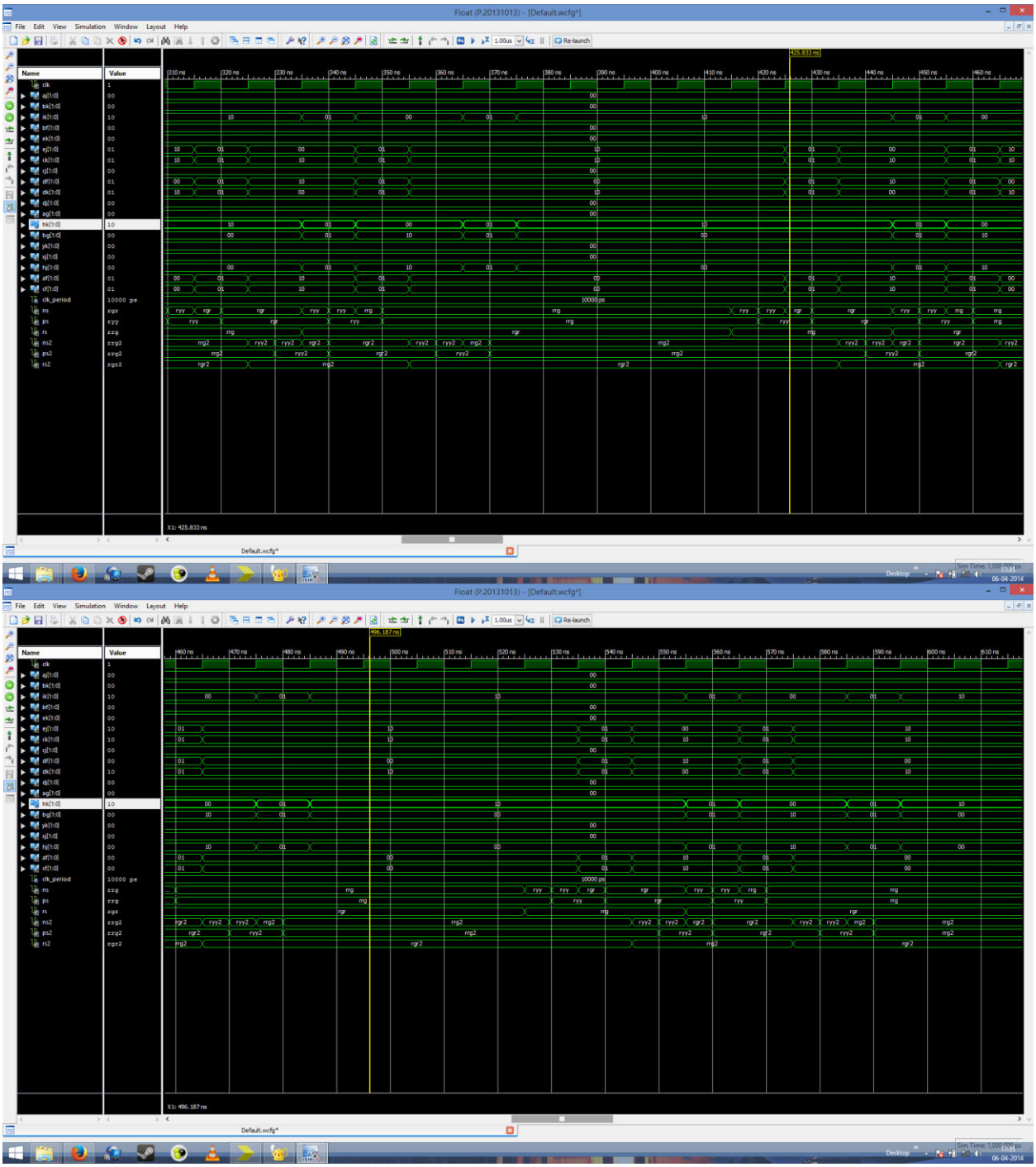
State Machine Diagram:



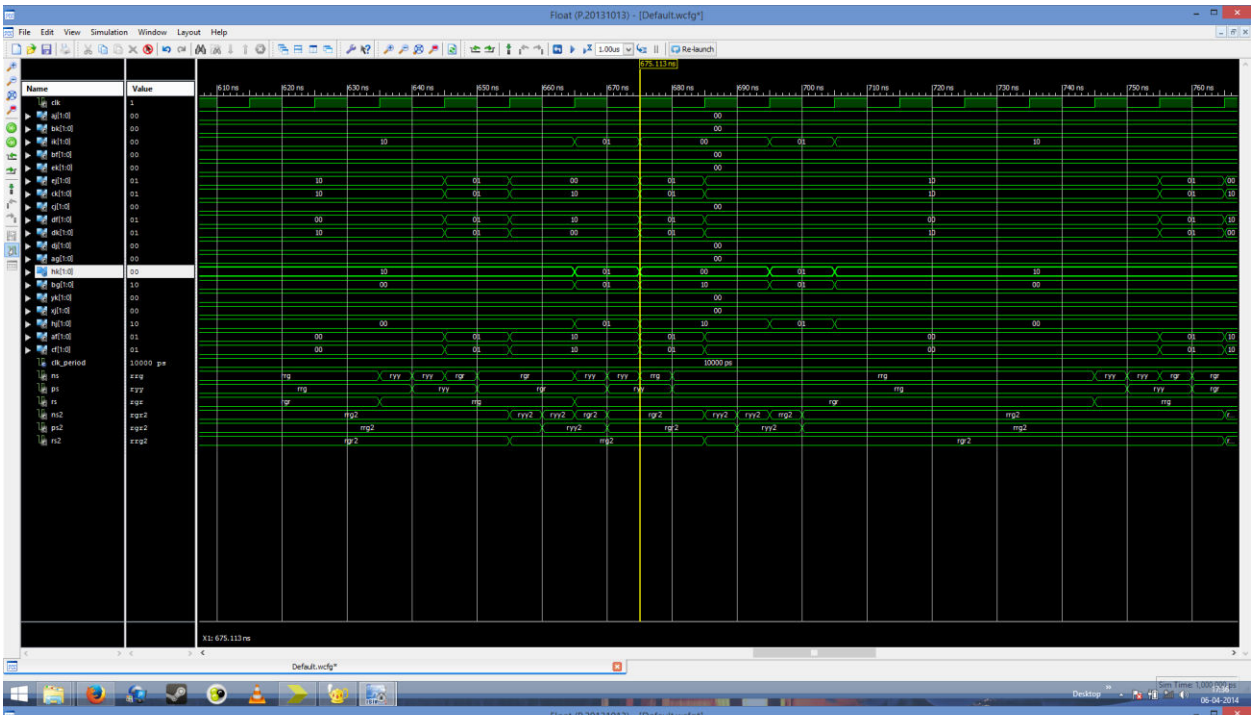
Timing Diagram:

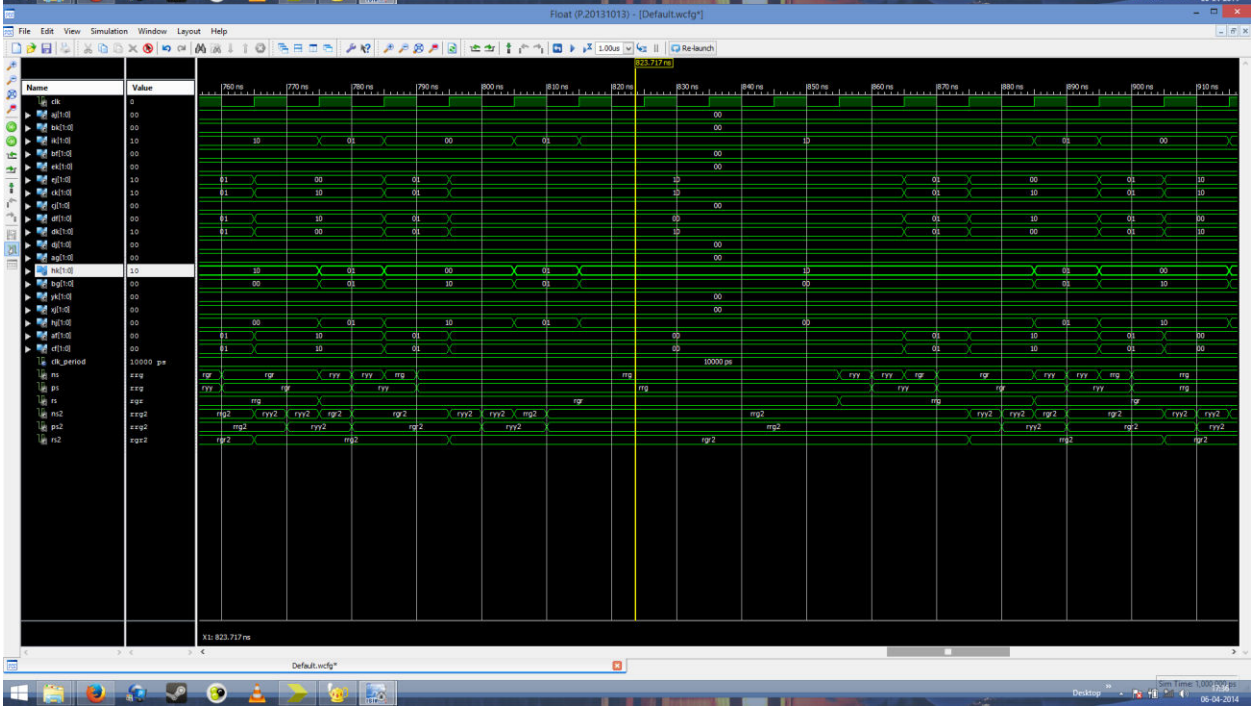
In Image 1, we can see that the FSM for pizza hut junction remains in grr state for 20 ns despite increasing the traffic for other routes because we have set the minimum transition time to be 20 ns.
Also, in the first image we can observe the amber transition for 10 ns.
In image 3, we can observe that the state for ps2 changes after 70 ns time even without changing the traffic conditions

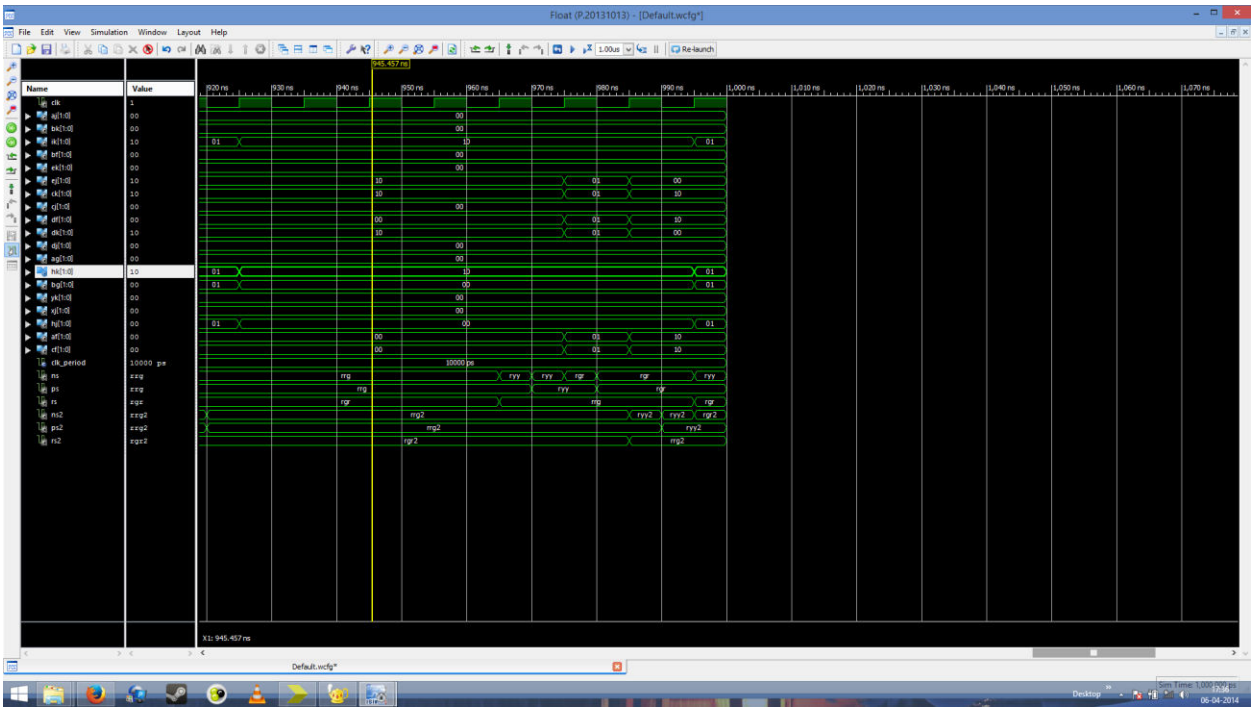

Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
clk
ak[1:0]
bk[1:0]
ik[1:0]
ek[1:0]
ck[1:0]
df[1:0]
dk[1:0]
ag[1:0]
hk[1:0]
bg[1:0]
xj[1:0]
af[1:0]
clk_period
10000 ps
X1: 133.083 ns
Default.wcfg*
Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
X1: 231.687 ns
Default.wcfg*


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
clk
a[1:0]
bk[1:0]
ik[1:0]
bf[1:0]
ek[1:0]
ej[1:0]
ck[1:0]
cj[1:0]
df[1:0]
dk[1:0]
dj[1:0]
ag[1:0]
hk[1:0]
bg[1:0]
yk[1:0]
xj[1:0]
hj[1:0]
af[1:0]
cf[1:0]
clk_period
10000 ps
ns
ps
rs
ns2
ps2
rs2
425.833 ns
10000 ps
X1: 425.833 ns
Default.wcfg*
Desktop
06-04-2014

Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
496.187 ns
Name
Value
clk
a[1:0]
bk[1:0]
ik[1:0]
bf[1:0]
ek[1:0]
ej[1:0]
ck[1:0]
cj[1:0]
df[1:0]
dk[1:0]
dj[1:0]
ag[1:0]
hk[1:0]
bg[1:0]
yk[1:0]
xj[1:0]
hj[1:0]
af[1:0]
cf[1:0]
clk_period
10000 ps
ns
ps
rs
ns2
ps2
rs2
10000 ps
X1: 496.187 ns
Default.wcfg*
Desktop
06-04-2014


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
clk
clk_period
10000 ps
X1: 675.113 ns
Default.wcfg*


Float (P.20131013) - [Default.wcfg*]
File Edit View Simulation Window Layout Help
Re-launch
Name
Value
clk
clk_period
10000 ps
X1: 823.717 ns
Default.wcfg*



VHDL Module:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:47:43 04/06/2014
-- Design Name:
-- Module Name:    traffic_signal - 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_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity traffic_signal is
    Port ( q1 : in  STD_LOGIC_VECTOR (2 downto 0);
           q2 : in  STD_LOGIC_VECTOR (2 downto 0);
           q3 : in  STD_LOGIC_VECTOR (2 downto 0);
           q4 : in  STD_LOGIC_VECTOR (2 downto 0);
           q5 : in  STD_LOGIC_VECTOR (2 downto 0);
           q6 : in  STD_LOGIC_VECTOR (2 downto 0);
           q7 : in  STD_LOGIC_VECTOR (2 downto 0);
           q8 : in  STD_LOGIC_VECTOR (2 downto 0);
           q9 : in  STD_LOGIC_VECTOR (2 downto 0);

           s1 : in  STD_LOGIC_VECTOR (2 downto 0);
           s2 : in  STD_LOGIC_VECTOR (2 downto 0);
           s3 : in  STD_LOGIC_VECTOR (2 downto 0);
           s4 : in  STD_LOGIC_VECTOR (2 downto 0);
           s5 : in  STD_LOGIC_VECTOR (2 downto 0);
           s6 : in  STD_LOGIC_VECTOR (2 downto 0);
           s7 : in  STD_LOGIC_VECTOR (2 downto 0);
           s8 : in  STD_LOGIC_VECTOR (2 downto 0);
           s9 : in  STD_LOGIC_VECTOR (2 downto 0);

                                AJ : out STD_LOGIC_VECTOR (1 downto 0);
                                BK : out STD_LOGIC_VECTOR (1 downto 0);
                                IK : out STD_LOGIC_VECTOR (1 downto 0);
                                BF : out STD_LOGIC_VECTOR (1 downto 0);
                                EK : out STD_LOGIC_VECTOR (1 downto 0);
                                EJ : out STD_LOGIC_VECTOR (1 downto 0);
                                CK : out STD_LOGIC_VECTOR (1 downto 0);
                                CJ : out STD_LOGIC_VECTOR (1 downto 0);
                                DF : out STD_LOGIC_VECTOR (1 downto 0);
                                DK : out STD_LOGIC_VECTOR (1 downto 0);
                                DJ : out STD_LOGIC_VECTOR (1 downto 0);
                                AG : out STD_LOGIC_VECTOR (1 downto 0);
                                HK : out STD_LOGIC_VECTOR (1 downto 0);
                                BG : out STD_LOGIC_VECTOR (1 downto 0);
                                YK : out STD_LOGIC_VECTOR (1 downto 0);
                                XJ : out STD_LOGIC_VECTOR (1 downto 0);
                                HJ : out STD_LOGIC_VECTOR (1 downto 0);
                                AF : out STD_LOGIC_VECTOR (1 downto 0);
                                CF : out STD_LOGIC_VECTOR (1 downto 0);
                                clk : in  STD_LOGIC);
end traffic_signal;
```

```
architecture Behavioral of traffic_signal is
Type STATE_TYPE is (grr,rgr,rrg,yyr,yry,ryy);
Type STATE_TYPE2 is (grr2,rgr2,rrg2,yyr2,yry2,ryy2);
Signal NS: STATE_TYPE:=grr;
Signal PS: STATE_TYPE:=grr;
Signal RS: STATE_TYPE:=grr;

Signal NS2: STATE_TYPE2:=grr2;
Signal PS2: STATE_TYPE2:=grr2;
Signal RS2: STATE_TYPE2:=grr2;

shared variable count_s1:std_logic_vector(3 downto 0):="0000";
shared variable count_s2:std_logic_vector(3 downto 0):="0000";
shared variable count_s3:std_logic_vector(3 downto 0):="0000";



shared variable count_s12:std_logic_vector(3 downto 0):="0000";
shared variable count_s22:std_logic_vector(3 downto 0):="0000";
shared variable count_s32:std_logic_vector(3 downto 0):="0000";
begin
SEQ:process(clk)
begin
--if rising_edge(clk) or falling_edge(clk) then
                                        PS <= NS;
                                        count_s1:=count_s1+"0001";
                                        count_s2:=count_s2+"0001";
                                        count_s3:=count_s3+"0001";

                                        PS2<=NS2;


        AJ<="00";BK<="00";IK<="00";BF<="00";EK<="00";EJ<="00";CK<="00";CJ<="00";DF<="00";DK<=
"00";DJ<="00";AG<="00";HK<="00";BG<="00";
                                        YK<="00";XJ<="00";HJ<="00";AF<="00";CF<="00";
                                        count_s12:=count_s12+"0001";
                                        count_s22:=count_s22+"0001";
                                        count_s32:=count_s32+"0001";
                                        case PS is
                                                when grr=>
                                                        AJ  <= "10";YK<="10";
                                                when rgr =>
                                                        AF <="10";CF<="10";DF<="10";CK<="10";
                                                when rrg=>
                                                        EJ<="10";CK<="10";DK<="10";
                                                when yyr=>
                                                        AJ  <= "01";YK<="01";AF
<="01";CF<="01";DF<="01";CK<="01";
                                                when ryy=>
```

```
                                        EJ<="01";CK<="01";DK<="01";AF
<="01";CF<="01";DF<="01";CK<="01";
                                when yry=>
                                        AJ  <= "01";YK<="01";EJ<="01";CK<="01";DK<="01";
                        end case;


                        case PS2 is
                                when grr2=>
                                        XJ  <= "10";BK<="10";AG<="10";
                                when rgr2 =>
                                        HJ <="10";BG<="10";
                                when rrg2=>
                                        HK<="10";IK<="10";
                                when yyr2=>
                                        XJ <= "01";BK<="01";AG<="01";HJ <="01";BG<="01";
                                when ryy2=>
                                        HJ <="01";BG<="01";HK<="01";IK<="01";
                                when yry2=>
                                        HK<="01";IK<="01";XJ <= "01";BK<="01";AG<="01";
                        end case;
--end if;
end process SEQ;
--PS,q1,q2,q3,q4,q5,s1,s2,s3,s4,s5,
        COM:process(clk,PS,q1,q2,q3,q4,q5,q6,q7,q8,q9,s1,s2,s3,s4,s5,s6,s7,s8,s9,PS2)
        Variable val1: STD_LOGIC_VECTOR (4 downto 0);
        Variable val2: STD_LOGIC_VECTOR (4 downto 0);
        Variable val3: STD_LOGIC_VECTOR (4 downto 0);
        Variable val4: STD_LOGIC_VECTOR (4 downto 0);
        Variable val5: STD_LOGIC_VECTOR (4 downto 0);
        Variable val6: STD_LOGIC_VECTOR (4 downto 0);
        Variable val7: STD_LOGIC_VECTOR (4 downto 0);
        Variable val8: STD_LOGIC_VECTOR (4 downto 0);
        Variable val9: STD_LOGIC_VECTOR (4 downto 0);
        Variable v1: STD_LOGIC_VECTOR (4 downto 0);
        Variable v2: STD_LOGIC_VECTOR (4 downto 0);
        Variable v3: STD_LOGIC_VECTOR (4 downto 0);
        Variable v4: STD_LOGIC_VECTOR (4 downto 0);
        Variable v5: STD_LOGIC_VECTOR (4 downto 0);
        Variable v6: STD_LOGIC_VECTOR (4 downto 0);
        begin
        val1(2):=q1(2) and q1(1) and q1(0) and (not s1(2)) and (not s1(1)) and  (not s1(0));
        val1(1):=q1(1) and q1(0) and (not s1(0)) and (not s1(1));
        val1(0):= q1(0) and (not s1(0));
        val1(3):='0';
        val1(4):='0';

        val2(2):=q2(2) and q2(1) and q2(0) and (not s2(2)) and (not s2(1)) and  (not s2(0));
```

```
val2(1):=q2(1) and q2(0) and (not s2(0)) and (not s2(1));
val2(0):= q2(0) and (not s2(0));
val2(3):='0';
val2(4):='0';

val3(2):=q3(2) and q3(1) and q3(0) and (not s3(2)) and (not s3(1)) and  (not s3(0));
val3(1):=q3(1) and q3(0) and (not s3(0)) and (not s3(1));
val3(0):= q3(0) and (not s3(0));
val3(3):='0';
val3(4):='0';

val4(2):=q4(2) and q4(1) and q4(0) and (not s4(2)) and (not s4(1)) and  (not s4(0));
val4(1):=q4(1) and q4(0) and (not s4(0)) and (not s4(1));
val4(0):= q4(0) and (not s4(0));
val4(3):='0';
val4(4):='0';

val5(2):=q5(2) and q5(1) and q5(0) and (not s5(2)) and (not s5(1)) and  (not s5(0));
val5(1):=q5(1) and q5(0) and (not s5(0)) and (not s5(1));
val5(0):= q5(0) and (not s5(0));
val5(3):='0';
val5(4):='0';


val6(2):=q5(2) and q6(1) and q6(0) and (not s6(2)) and (not s6(1)) and  (not s6(0));
val6(1):=q6(1) and q6(0) and (not s6(0)) and (not s6(1));
val6(0):= q6(0) and (not s6(0));
val6(3):='0';
val6(4):='0';


val7(2):=q7(2) and q7(1) and q7(0) and (not s7(2)) and (not s7(1)) and  (not s7(0));
val7(1):=q7(1) and q7(0) and (not s7(0)) and (not s7(1));
val7(0):= q7(0) and (not s7(0));
val7(3):='0';
val7(4):='0';


val8(2):=q8(2) and q8(1) and q8(0) and (not s8(2)) and (not s8(1)) and  (not s8(0));
val8(1):=q8(1) and q8(0) and (not s8(0)) and (not s8(1));
val8(0):= q8(0) and (not s8(0));
val8(3):='0';
val8(4):='0';


val9(2):=q9(2) and q9(1) and q9(0) and (not s9(2)) and (not s9(1)) and  (not s9(0));
val9(1):=q9(1) and q9(0) and (not s9(0)) and (not s9(1));
val9(0):= q9(0) and (not s9(0));
```

```
val9(3):='0';
val9(4):='0';

v1:=val1 + val3;
v2:=val1 + val4 +val5;
v3:=val2 + val4 +val5;

v4:=val6+val8;
v5:=val7+val8;
v6:=val7+val9;

case PS2 is
        when grr2=>
                if(count_s12 < "0100") then
                        NS2<=grr2;
                elsif (count_s12 > "1101") then
                        if(v5 > v6) then

NS2<=yyr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        else

NS2<=yry2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        end if;
                elsif(v4 >= v5 and v4 >= v6) then
                        NS2<=grr2;
                elsif(v5 >= v4 and v5 >= v6) then

NS2<=yyr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                elsif(v6>=v4 and v6 >= v5) then

NS2<=yry2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                end if;


        when rgr2=>
                if(count_s22 < "0100") then
                        NS2<=rgr2;
                elsif (count_s22 > "1101") then
                        if(v4 > v6) then

NS2<=yyr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        else

NS2<=ryy2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        end if;
                elsif(v4 >= v5 and v4 >= v6) then

NS2<=yyr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
```

```
                elsif(v5 >= v4 and v5 >= v6) then
                        NS2<=rgr2;
                elsif(v6>=v4 and v6 >= v5) then

NS2<=ryy2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                end if;


        when rrg2=>
                if(count_s32 < "0100") then
                        NS2<=rrg2;
                elsif (count_s32 > "1101") then
                        if(v5 > v4) then

NS2<=ryy2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        else

NS2<=yry2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                        end if;
                elsif(v4 >= v5 and v4 >= v6) then

NS2<=yry2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                elsif(v5 >= v4 and v5 >= v6) then

NS2<=ryy2;count_s12:="0000";count_s22:="0000";count_s32:="0000";RS2<=PS2;
                elsif(v6>=v4 and v6 >= v5) then
                        NS2<=rrg2;
                end if;



        when yyr2=>
                if(count_s12 < "0010") then
                        NS2<=yyr2;
                elsif (RS2 = grr2) then
                        NS2 <=rgr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
                else
                        NS2 <= grr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
                end if;


        when ryy2=>
                if(count_s22 < "0010") then
                        NS2<=ryy2;
                elsif (RS2 = rgr2) then
                        NS2 <=rrg2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
                else
                        NS2 <= rgr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
```

```
                end if;

        when yry2=>
                if(count_s32 < "0010") then
                        NS2<=yry2;
                elsif (RS2 = grr2) then
                        NS2 <=rrg2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
                else
                        NS2 <= grr2;count_s12:="0000";count_s22:="0000";count_s32:="0000";
                end if;
end case;


--second signal near  main gate

case PS is
        when grr=>
                if(count_s1 < "0100") then
                        NS<=grr;
                elsif (count_s1 > "1101") then
                        if(v2 > v3) then

NS<=yyr;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                        else

NS<=yry;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                        end if;
                elsif(v1 >= v2 and v1 >= v3) then
                        NS<=grr;
                elsif(v2 >= v1 and v2 >= v3) then

NS<=yyr;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                elsif(v3>=v1 and v3 >= v2) then

NS<=yry;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                end if;


        when rgr=>
                if(count_s2 < "0100") then
                        NS<=rgr;
                elsif (count_s2 > "1101") then
                        if(v1 > v3) then

NS<=yyr;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                        else

NS<=ryy;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
```

```
                    end if;
                elsif(v1 >= v2 and v1 >= v3) then

NS<=yyr;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                elsif(v2 >= v1 and v2 >= v3) then
                    NS<=rgr;
                elsif(v3>=v1 and v3 >= v2) then

NS<=ryy;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                end if;


        when rrg=>
                if(count_s3 < "0100") then
                    NS<=rrg;
                elsif (count_s3 > "1101") then
                    if(v2 > v1) then

NS<=ryy;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                    else

NS<=yry;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                    end if;
                elsif(v1 >= v2 and v1 >= v3) then

NS<=yry;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                elsif(v2 >= v1 and v2 >= v3) then

NS<=ryy;count_s1:="0000";count_s2:="0000";count_s3:="0000";RS<=PS;
                elsif(v3>=v1 and v3 >= v2) then
                    NS<=rrg;
                end if;



        when yyr=>
                if(count_s1 < "0010") then
                    NS<=yyr;
                elsif (RS = grr) then
                    NS <=rgr;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                else
                    NS <= grr;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                end if;


        when ryy=>
                if(count_s2 < "0010") then
                    NS<=ryy;
```

```
                elsif (RS = rgr) then
                        NS <=rrg;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                else
                        NS <= rgr;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                end if;

        when yry=>
                if(count_s3 < "0010") then
                        NS<=yry;
                elsif (RS = grr) then
                        NS <=rrg;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                else
                        NS <= grr;count_s1:="0000";count_s2:="0000";count_s3:="0000";
                end if;
    end case;


    end process COM;

end Behavioral;
```

VHDL Test Bench:

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   16:10:22 04/06/2014
-- Design Name:
-- Module Name:   C:/Users/Sanket/project_288/t_traffic.vhd
-- Project Name:  project_288
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: traffic_signal
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
```

```
--------------------------------------------------------------------------------
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 t_traffic IS
END t_traffic;

ARCHITECTURE behavior OF t_traffic IS

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

    COMPONENT traffic_signal
    PORT(
        q1 : IN  std_logic_vector(2 downto 0);
        q2 : IN  std_logic_vector(2 downto 0);
        q3 : IN  std_logic_vector(2 downto 0);
        q4 : IN  std_logic_vector(2 downto 0);
        q5 : IN  std_logic_vector(2 downto 0);
        q6 : IN  std_logic_vector(2 downto 0);
        q7 : IN  std_logic_vector(2 downto 0);
        q8 : IN  std_logic_vector(2 downto 0);
        q9 : IN  std_logic_vector(2 downto 0);
        s1 : IN  std_logic_vector(2 downto 0);
        s2 : IN  std_logic_vector(2 downto 0);
        s3 : IN  std_logic_vector(2 downto 0);
        s4 : IN  std_logic_vector(2 downto 0);
        s5 : IN  std_logic_vector(2 downto 0);
        s6 : IN  std_logic_vector(2 downto 0);
        s7 : IN  std_logic_vector(2 downto 0);
        s8 : IN  std_logic_vector(2 downto 0);
        s9 : IN  std_logic_vector(2 downto 0);
        AJ : OUT  std_logic_vector(1 downto 0);
        BK : OUT  std_logic_vector(1 downto 0);
        IK : OUT  std_logic_vector(1 downto 0);
        BF : OUT  std_logic_vector(1 downto 0);
        EK : OUT  std_logic_vector(1 downto 0);
        EJ : OUT  std_logic_vector(1 downto 0);
        CK : OUT  std_logic_vector(1 downto 0);
        CJ : OUT  std_logic_vector(1 downto 0);
        DF : OUT  std_logic_vector(1 downto 0);
        DK : OUT  std_logic_vector(1 downto 0);
        DJ : OUT  std_logic_vector(1 downto 0);
        AG : OUT  std_logic_vector(1 downto 0);
        HK : OUT  std_logic_vector(1 downto 0);
```

```
    BG : OUT  std_logic_vector(1 downto 0);
    YK : OUT  std_logic_vector(1 downto 0);
    XJ : OUT  std_logic_vector(1 downto 0);
    clk : IN  std_logic
   );
END COMPONENT;


--Inputs
signal q1 : std_logic_vector(2 downto 0) := (others => '0');
signal q2 : std_logic_vector(2 downto 0) := (others => '0');
signal q3 : std_logic_vector(2 downto 0) := (others => '0');
signal q4 : std_logic_vector(2 downto 0) := (others => '0');
signal q5 : std_logic_vector(2 downto 0) := (others => '0');
signal q6 : std_logic_vector(2 downto 0) := (others => '0');
signal q7 : std_logic_vector(2 downto 0) := (others => '0');
signal q8 : std_logic_vector(2 downto 0) := (others => '0');
signal q9 : std_logic_vector(2 downto 0) := (others => '0');
signal s1 : std_logic_vector(2 downto 0) := (others => '0');
signal s2 : std_logic_vector(2 downto 0) := (others => '0');
signal s3 : std_logic_vector(2 downto 0) := (others => '0');
signal s4 : std_logic_vector(2 downto 0) := (others => '0');
signal s5 : std_logic_vector(2 downto 0) := (others => '0');
signal s6 : std_logic_vector(2 downto 0) := (others => '0');
signal s7 : std_logic_vector(2 downto 0) := (others => '0');
signal s8 : std_logic_vector(2 downto 0) := (others => '0');
signal s9 : std_logic_vector(2 downto 0) := (others => '0');
signal clk : std_logic := '0';

        --Outputs
signal AJ : std_logic_vector(1 downto 0);
signal BK : std_logic_vector(1 downto 0);
signal IK : std_logic_vector(1 downto 0);
signal BF : std_logic_vector(1 downto 0);
signal EK : std_logic_vector(1 downto 0);
signal EJ : std_logic_vector(1 downto 0);
signal CK : std_logic_vector(1 downto 0);
signal CJ : std_logic_vector(1 downto 0);
signal DF : std_logic_vector(1 downto 0);
signal DK : std_logic_vector(1 downto 0);
signal DJ : std_logic_vector(1 downto 0);
signal AG : std_logic_vector(1 downto 0);
signal HK : std_logic_vector(1 downto 0);
signal BG : std_logic_vector(1 downto 0);
signal YK : std_logic_vector(1 downto 0);
signal XJ : std_logic_vector(1 downto 0);
        signal HJ : std_logic_vector(1 downto 0);
        signal AF : std_logic_vector(1 downto 0);
```

```
        signal CF : std_logic_vector(1 downto 0);



   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: traffic_signal PORT MAP (
        q1 => q1,
        q2 => q2,
        q3 => q3,
        q4 => q4,
        q5 => q5,
        q6 => q6,
        q7 => q7,
        q8 => q8,
        q9 => q9,
        s1 => s1,
        s2 => s2,
        s3 => s3,
        s4 => s4,
        s5 => s5,
        s6 => s6,
        s7 => s7,
        s8 => s8,
        s9 => s9,
        AJ => AJ,
        BK => BK,
        IK => IK,
        BF => BF,
        EK => EK,
        EJ => EJ,
        CK => CK,
        CJ => CJ,
        DF => DF,
        DK => DK,
        DJ => DJ,
        AG => AG,
        HK => HK,
        BG => BG,
        YK => YK,
        XJ => XJ,
                        HJ => HJ,
                        AF => AF,
                        CF => CF,
```

```
      clk => clk
    );

-- Clock process definitions
clk_process :process
begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.

                q1<="111";s1<="000";
                q3<="111";s3<="000";
                q7<="111";s7<="000";
                q8<="111";s8<="000";

                wait for 20 ns;

                q3<="000";s3<="111";
                q7<="000";s7<="111";

                q1<="111";s1<="000";
                q4<="111";s4<="000";
                q5<="111";s5<="000";
                q6<="111";s6<="000";
                q8<="111";s8<="000";

                wait for 100 ns;

                q1<="000";s1<="111";
                q6<="000";s6<="111";
                q8<="000";s8<="111";

                q2<="111";s2<="000";
                q4<="111";s4<="000";
                q5<="111";s5<="000";
                q7<="111";s7<="000";
                q9<="111";s9<="000";
```

```
      -- insert stimulus here

      wait;
   end process;

END;
```