CS 210 ASSIGNMENT – 1

GROUP MEMBERS:

Zubin Arya (120050036)

Naveen Sagar (120050026)

Pintu Lal Meena (120050018)

Bhupendra Singh Bhuarya (120050040)

Aim: Design an automated traffic system for the IIT main-gate and pizzahut interconnection points

Procedure:

1. Firstly we have defined logic vectors each denoting a specific routes (e.g ck, aj, bk, etc) , which gives the percent of traffic through that particular route
2. 00 represents 0 percent traffic, 01 represents 33 percent traffic and 11 represents 66 percent traffic
3. <u>MAIN GATE JUNCTION</u>
4. After that we have assigned weights to the traffic heading towards each route (internal weights)
    a. Traffic coming from C and D

Heading towards K	4
Heading towards F	2
Heading towards J	4

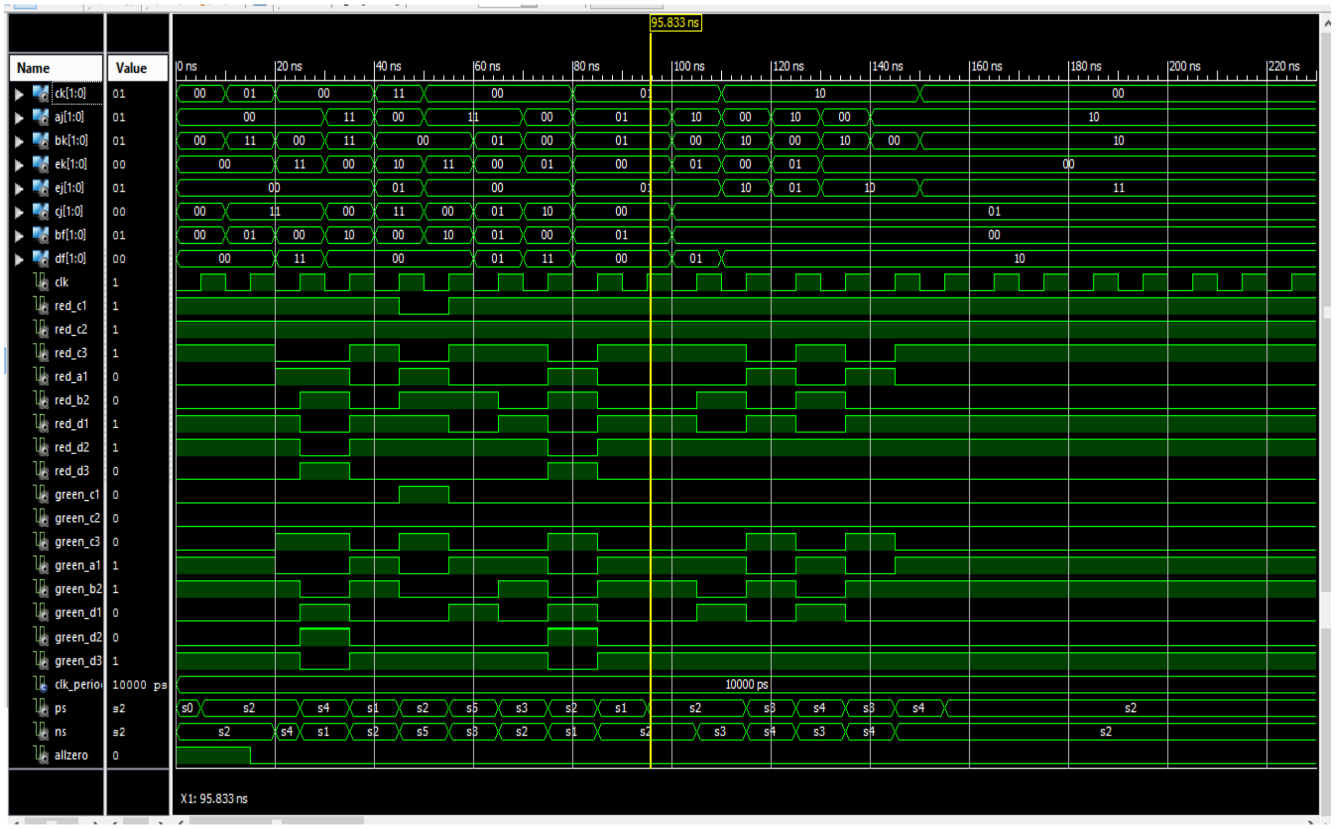
    b. Traffic coming from A(A1) and heading towards J is 4
    c. Traffic coming from B(B2) and
        i. Heading towards k is 5
        ii. Heading towards F is 1
    d. Traffic coming from E
        i. Heading towards k is 5
        ii. Heading towards j is 5
5. Further we have also assigned weights (external weights) alpha1 = 3(FOR a), alpha2 = 5(FOR b and c) and alpha3 = 2(FOR c/d) to the traffic coming from (C & D) , A and E.
6. We have considered various states with maximum traffic clearances
7. At every rising edge we compare the computed weighted sum of traffic quality weights for the blocked roads in every state and choose the next state with minimum weighted sum.
8. <u>PIZZA HUT JUNCTION</u>
9. After that we have assigned weights to the traffic heading towards each route (internal weights)

a. Traffic coming from H and heading towards J (HJ) is 5.
b. Traffic coming from H and heading towards K (HK) is 5.
c. Traffic coming from A and heading towards G(AG) is 3.
d. Traffic coming from A and heading towards J(AJ) is 7.
e. Traffic coming from B and heading towards K(BK) is 7.
f. Traffic coming from B and heading towards G(BG) is 3.

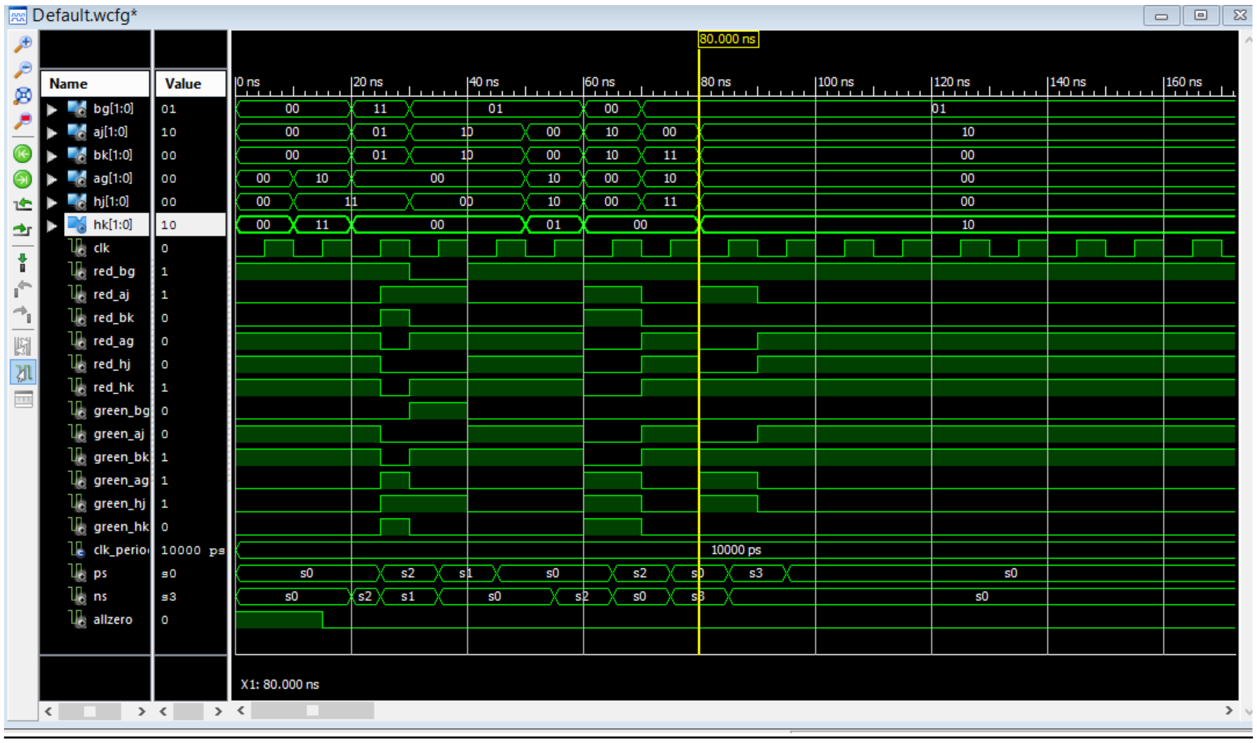
10. Further we have also assigned weights (external weights) alpha1 = 2(for a and b), alpha2 = 4(for c,d,e and f) i.e traffic coming from a and b.
11. Rest of the steps are same as before i.e considering states of maximum clearance and choosing minimum weighted sum.

<u>Timing Diagram</u>

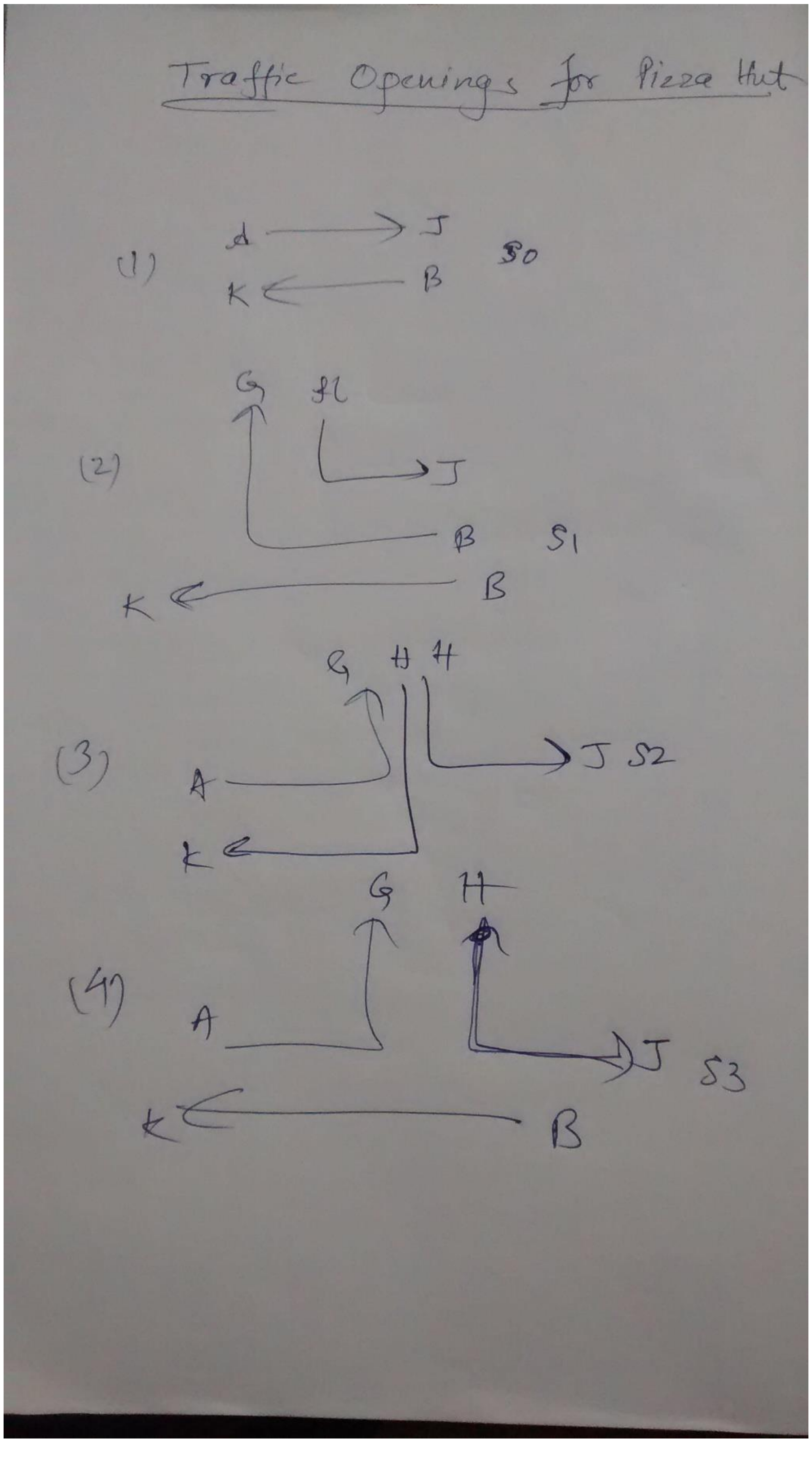
<u>Timing Diagram for Main Gate Junction</u>



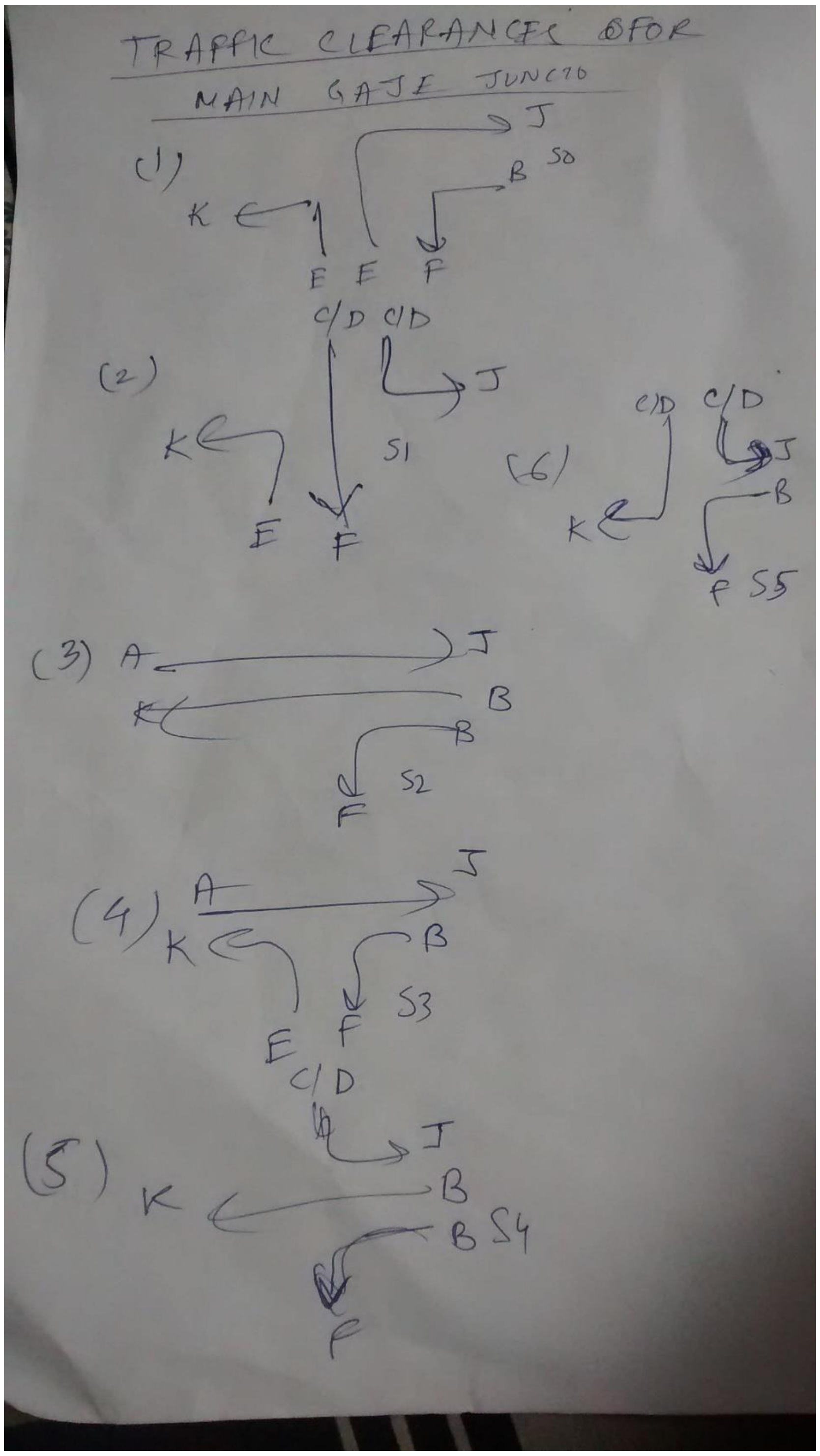
Timing Diagram for Pizza Hut Junction



Name	Value
bg[1:0]	01
aj[1:0]	10
bk[1:0]	00
ag[1:0]	00
hj[1:0]	00
hk[1:0]	10
clk	0
red_bg	1
red_aj	1
red_bk	0
red_ag	0
red_hj	0
red_hk	1
green_bg	0
green_aj	0
green_bk	1
green_ag	1
green_hj	1
green_hk	0
clk_perio	10000 ps
ps	s0
ns	s3
allzero	0

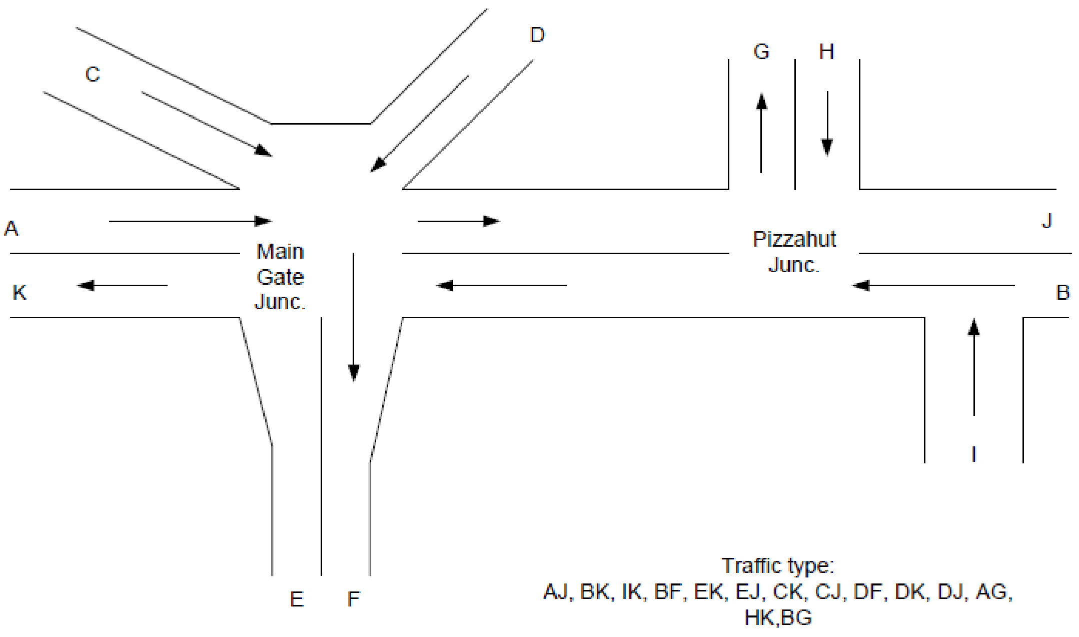


<u>States for Pizza Hut Junction</u>



<u>States for Main Gate Junction</u>

Below is the figure showing the traffic arriving a each intersection



VHDL code:

- For main-gate junction:

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.numeric_std.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

-- Uncomment the following library declaration if instantiating

-- any Xilinx primitives in this code.

```
--library UNISIM;

--use UNISIM.VComponents.all;


entity maingate is

   Port ( CK : in  STD_LOGIC_VECTOR (1 downto 0);

         AJ : in  STD_LOGIC_VECTOR (1 downto 0);

         BK : in  STD_LOGIC_VECTOR (1 downto 0);

         EK : in  STD_LOGIC_VECTOR (1 downto 0);

         EJ : in  STD_LOGIC_VECTOR (1 downto 0);

         CJ : in  STD_LOGIC_VECTOR (1 downto 0);

         BF : in  STD_LOGIC_VECTOR (1 downto 0);


         DF : in  STD_LOGIC_VECTOR (1 downto 0);


         clk : in  STD_LOGIC;

                                red_c1:out std_logic;

                                red_c2:out std_logic;

                                red_c3:out std_logic;


red_a1:out std_logic;

red_b2:out std_logic;

red_d1:out std_logic;

red_d2:out std_logic;

red_d3:out std_logic;

                               green_c1:out std_logic;

                                green_c2:out std_logic;

                                green_c3:out std_logic;


green_a1:out std_logic;
green_b2:out std_logic;
green_d1:out std_logic;
green_d2:out std_logic;
green_d3:out std_logic
                                        );
end maingate;

architecture Behavioral of maingate is
type state_type is (S0,S1,S2,S3,S4,S5);
signal PS,NS:state_type:=S0;
signal allzero:std_logic :='1';
shared variable c1,c2,c3,a1,b2,d1,d2,d3:integer:=0;

type int_array is array(0 to 5) of integer;
shared variable k:int_array;
shared variable index_min:integer:=0;
begin
process(clk,CK,DF,AJ,BK,EK,EJ,BF,PS,NS)

begin
if(rising_edge(clk) and clk='1') then
PS<=NS;

c1:=conv_integer(CK)*12;
c2:=conv_integer(DF)*6;
c3:=conv_integer(CJ)*12;
a1:=conv_integer(AJ)*20;
b2:=conv_integer(BK)*25;
d1:=conv_integer(EK)*10;
```

```
d2:=conv_integer(EJ)*10;
d3:=conv_integer(BF)*5;
k(0):=c1+c2+c3+a1+b2;
k(1):=c1+c2+a1+d3+b2;
k(2):=c1+c2+c3+d1+d2;
k(3):=c1+c2+c3+b2+d2;
k(4):=c1+c2+a1+d1+d2;
k(5):=c2+b2+d2+a1+d1;
allzero<='1';
for i in 0 to 5 loop
if(k(i)<k(index_min)) then
index_min:=i;



end if;
end loop;
for i in 0 to 5 loop
if(not(k(i)=0)) then
allzero <='0';
end if;





end loop;
end if;

case PS is

when S0 =>
```

```
case index_min is

when 0=>

NS<=S0;

when 1=>

NS<=S1;

when 2=>

NS<=S2;

when 3=>

NS<=S3;

 when 4=>

NS<=S4;

when 5=>

NS<=S5;

when others=>

null;

end case;


when S1 =>

case index_min is

when 0=>

NS<=S0;

when 1=>

NS<=S1;

when 2=>

NS<=S2;

when 3=>

NS<=S3;

 when 4=>
```

```
NS<=S4;
when 5=>
NS<=S5;
when others=>
null;
end case;
when S2 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
NS<=S3;
 when 4=>
NS<=S4;
when 5=>
NS<=S5;
when others=>
null;
end case;
when S3 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
```

```
when 2=>
NS<=S2;
when 3=>
NS<=S3;
 when 4=>
NS<=S4;
WHEN 5=>
NS<=S5;
when others=>
null;
end case;
when S4 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
NS<=S3;
 when 4=>
NS<=S4;
when 5=>
NS<=S5;
when others=>
null;
end case;
```

```
when S5 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
NS<=S3;
 when 4=>
NS<=S4;
when 5=>
NS<=S5;
when others=>
null;
end case;
when others=>
null;
end case;
if(allzero='1') then
NS<=S2;
end if;
case NS is

when S0 =>
 red_c1<='1';
                                        red_c2<='1';
                                        red_c3<='1';
 red_a1<='1';
 red_b2<='1';
 red_d1<='0';
 red_d2<='0';
 red_d3<='0';
                                  green_c1<='0';
                                   green_c2<='0';
                                   green_c3<='0';

 green_a1<='0';
 green_b2<='0';
 green_d1<='1';
green_d2<='1';
green_d3<='1';
when S1 =>
red_c1<='1';
                                  red_c2<='1';
                                  red_c3<='0';

red_a1<='1';
red_b2<='1';
red_d1<='0';
red_d2<='0';
red_d3<='1';
                                 green_c1<='0';
                                  green_c2<='0';
                                  green_c3<='1';
```

```
green_a1<='0';
green_b2<='0';
green_d1<='1';
green_d2<='1';
green_d3<='0';
when S2 =>
red_c1<='1';
                                        red_c2<='1';
                                        red_c3<='1';

red_a1<='0';
red_b2<='0';
red_d1<='1';
red_d2<='1';
red_d3<='0';
                                       green_c1<='0';
                                        green_c2<='0';
                                        green_c3<='0';

green_a1<='1';
green_b2<='1';
green_d1<='0';
green_d2<='0';
green_d3<='1';
when S3 =>
red_c1<='1';
                                        red_c2<='1';
                                        red_c3<='1';
```

```
red_a1<='0';
red_b2<='1';
red_d1<='0';
red_d2<='1';
red_d3<='0';
                                  green_c1<='0';
                                   green_c2<='0';
                                   green_c3<='0';

green_a1<='1';
green_b2<='0';
green_d1<='1';
green_d2<='0';
green_d3<='1';
when S4=>
 red_c1<='1';
                                   red_c2<='1';
                                   red_c3<='0';

red_a1<='1';
red_b2<='0';
red_d1<='1';
red_d2<='1';
red_d3<='0';
                                  green_c1<='0';
                                   green_c2<='0';
                                   green_c3<='1';

green_a1<='0';
```

```
green_b2<='1';
green_d1<='0';
green_d2<='0';
green_d3<='1';
when S5=>
 red_c1<='0';
                                            red_c2<='1';
                                            red_c3<='0';

red_a1<='1';
red_b2<='1';
red_d1<='1';
red_d2<='1';
red_d3<='0';
                                           green_c1<='1';
                                            green_c2<='0';
                                            green_c3<='1';

green_a1<='0';
green_b2<='0';
green_d1<='0';
green_d2<='0';
green_d3<='1';
when others=> null;
end case;

end process;
end Behavioral;
```

- For pizzahut junction:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.ALL;
use IEEE.STD_LOGIC_ARITH.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 pizzahut is
   Port ( BG : in  STD_LOGIC_VECTOR (1 downto 0);
          AJ : in  STD_LOGIC_VECTOR (1 downto 0);
          BK : in  STD_LOGIC_VECTOR (1 downto 0);
          AG : in  STD_LOGIC_VECTOR (1 downto 0);
          HJ : in  STD_LOGIC_VECTOR (1 downto 0);
          HK : in  STD_LOGIC_VECTOR (1 downto 0);
          clk : in  STD_LOGIC;
                              red_BG:OUT STD_LOGIC;
                              red_AJ:OUT STD_LOGIC;
                              red_BK:OUT STD_LOGIC;
                              red_AG:OUT STD_LOGIC;
                              red_HJ:OUT STD_LOGIC;
                              red_HK:OUT STD_LOGIC;
```

```
                green_BG:OUT STD_LOGIC;
                green_AJ:OUT STD_LOGIC;
                green_BK:OUT STD_LOGIC;
                green_AG:OUT STD_LOGIC;
                green_HJ:OUT STD_LOGIC;
                green_HK:OUT STD_LOGIC

                );
end pizzahut;

architecture Behavioral of pizzahut is
type state_type is (S0,S1,S2,S3);
SIGNAL PS,NS:state_type:=S0;
shared variable  BG1,AJ1,BK1,AG1,HJ1,HK1:integer;
signal allzero:std_logic :='1';
type int_array is array(0 to 3) of integer;
shared variable k:int_array;
shared variable index_min:integer:=0;
begin
process(clk)
begin
if(rising_edge(clk) and clk='1') then
PS<=NS;
BG1:=12*conv_integer(BG);
AJ1:=28*conv_integer(AJ);
BK1:=28*conv_integer(BK);
AG1:=12*conv_integer(AG);
HJ1:=10*conv_integer(HJ);
HK1:=10*conv_integer(HK);
```

```
k(0):=AG1+BG1+HK1+HJ1;
k(1):=AG1+AJ1+HK1;
k(2):=AJ1+BK1+BG1;
k(3):=AJ1+BG1+HK1;
allzero<='1';
for i in 0 to 3 loop
if(k(i)<k(index_min)) then
index_min:=i;
END IF;
END LOOP;
for i in 0 to 3 loop
if(not(k(i)=0)) then
allzero<='0';
end if;



end loop;
END IF;
case PS is
when S0 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
```

```
NS<=S3;


when others=>
null;
end case;

when S1 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
NS<=S3;

when others=>
null;
end case;
when S2 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
```

```
when 3=>
NS<=S3;

when others=>
null;
end case;
when S3 =>
case index_min is
when 0=>
NS<=S0;
when 1=>
NS<=S1;
when 2=>
NS<=S2;
when 3=>
NS<=S3;

when others=>
null;
end case;
when others=>
null;
end case;
if(allzero='1') then
NS<=S0;
end if;
case NS is
when S0 =>
red_BG<='1';
```

```
                                        red_AJ<='0';
                                        red_BK<='0';
                                        red_AG<='1';
                                        red_HJ<='1';
                                        red_HK<='1';
                                        green_BG<='0';
                                        green_AJ<='1';
                                        green_BK<='1';
                                        green_AG<='0';
                                        green_HJ<='0';
                                        green_HK<='0';
when S1 =>
red_BG<='0';
                                        red_AJ<='1';
                                        red_BK<='0';
                                        red_AG<='1';
                                        red_HJ<='0';
                                        red_HK<='1';
                                        green_BG<='1';
                                        green_AJ<='0';
                                        green_BK<='1';
                                        green_AG<='0';
                                        green_HJ<='1';
                                        green_HK<='0';
when S2 =>
red_BG<='1';
                                        red_AJ<='1';
                                        red_BK<='1';
                                        red_AG<='0';
```

```
                    red_HJ<='0';
                    red_HK<='0';
                    green_BG<='0';
                    green_AJ<='0';
                    green_BK<='0';
                    green_AG<='1';
                    green_HJ<='1';
                    green_HK<='1';
when S3 =>
red_BG<='1';
                    red_AJ<='1';
                    red_BK<='0';
                    red_AG<='0';
                    red_HJ<='0';
                    red_HK<='1';
                    green_BG<='0';
                    green_AJ<='0';
                    green_BK<='1';
                    green_AG<='1';
                    green_HJ<='1';
                    green_HK<='0';
when others=>
null;
end case;
end process;
end Behavioral;
```

INFERENCE

The idea of state machines can be extended to Traffic System Design which relies on Moore Model.A synchronous clock can be used and can be altered to invoke duty cycle to cater to different traffic systems coexisting together in terms of density,frequency etc. Multilane system despite its complications is very apt for reducing traffic and can be easily implemented using digital logic