VLSI DESIGN

Assignment 4

NAME: Shreyansh Neekhre

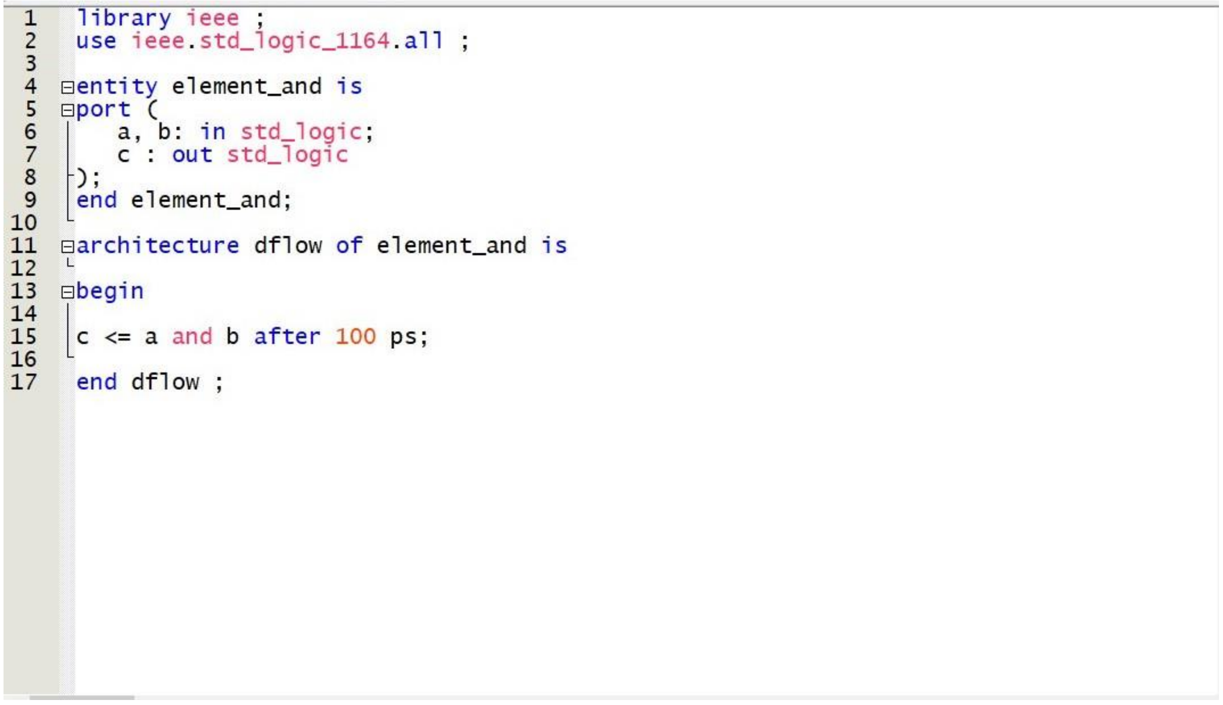
ROLL NO: 213079029

Specialization: Integrated Circuit and Systems

Q–1 Logarithmic adders use a tree structure to reduce the time taken for addition to a logarithmic function of the number of bits being added. Brent Kung adder is a simple example of this approach. Describe a 32 bit Brent Kung adder in VHDL and simulate it using a test bench.

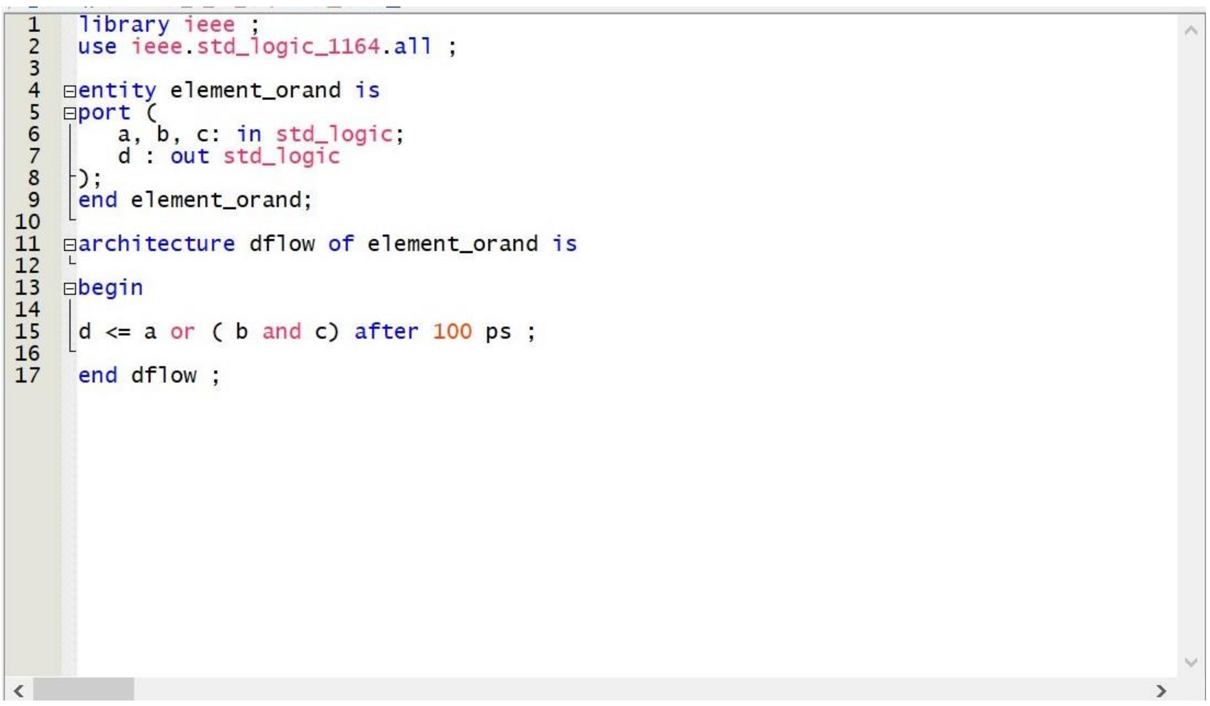
a) The adder needs logic functions AND, XOR, A + B.C to compute different orders of G, P and final sum and carry outputs. Write VHDL code in data flow style (using logic equations in assignments) to implement these functions. Each logic block should insert a delay of 100 ps in the assignments.

-------------------->>>>> here I have made and gate using dataflow style with 100 ps delay.



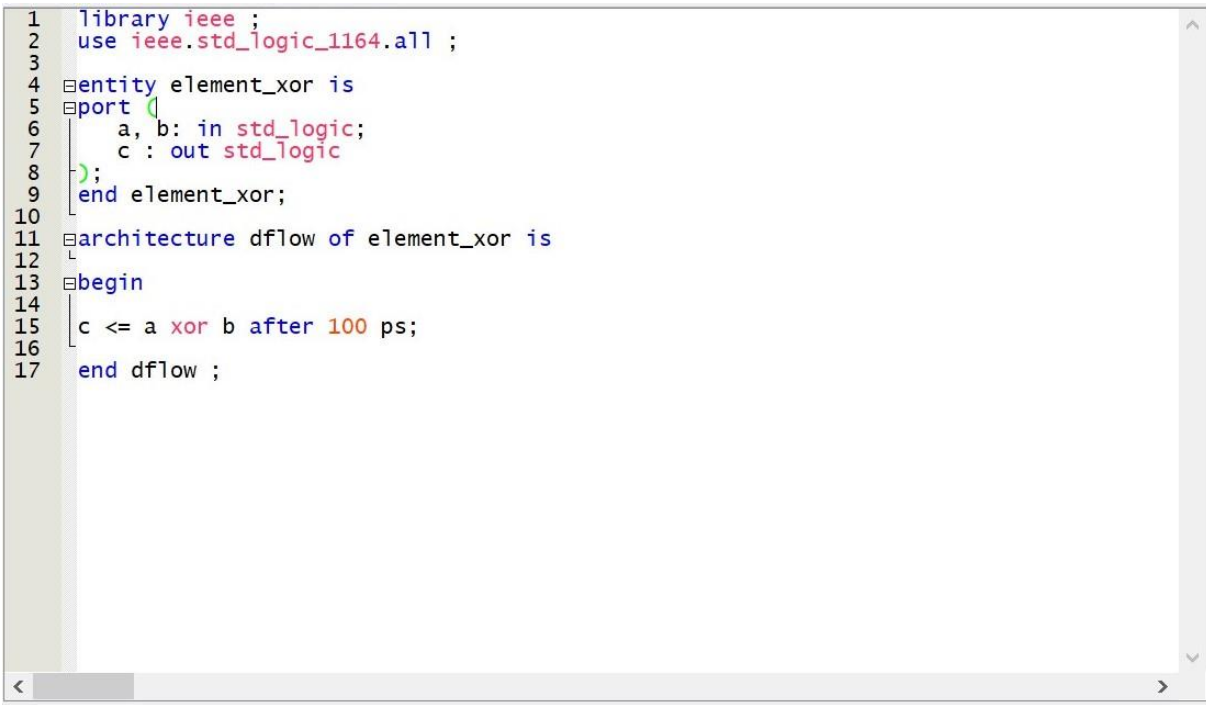
```vhdl
library ieee ;
use ieee.std_logic_1164.all ;

entity element_and is
port (
    a, b: in std_logic;
    c : out std_logic
);
end element_and;

architecture dflow of element_and is

begin

c <= a and b after 100 ps;

end dflow ;
```

------------------->>>> here I have made and – or logic used to generate higher order G with 100 ps delay.



```vhdl
library ieee ;
use ieee.std_logic_1164.all ;

entity element_orand is
port (
    a, b, c: in std_logic;
    d : out std_logic
);
end element_orand;

architecture dflow of element_orand is

begin

d <= a or ( b and c) after 100 ps ;

end dflow ;
```

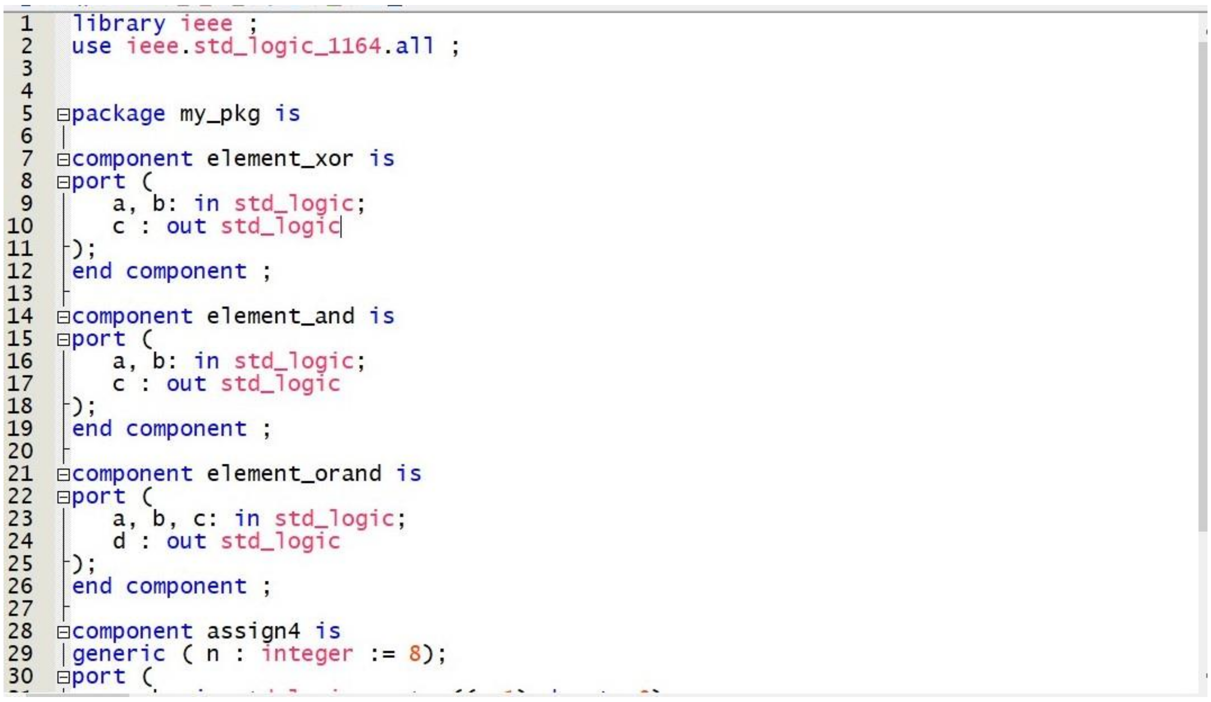
------------>>>>>> here I have made xor gate used to generate 1 order P only with 100 ps delay.



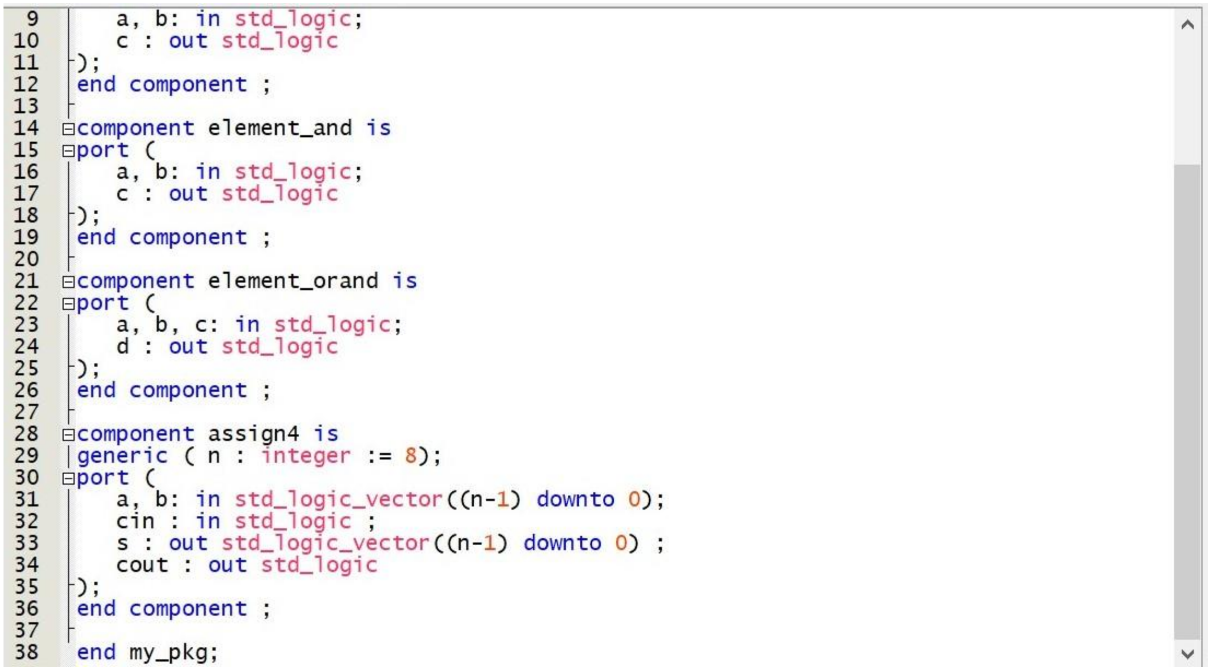
```vhdl
library ieee ;
use ieee.std_logic_1164.all ;

entity element_xor is
port (
    a, b: in std_logic;
    c : out std_logic
);
end element_xor;

architecture dflow of element_xor is

begin

c <= a xor b after 100 ps;

end dflow ;
```

b) Using the above entities as components, write structural descriptions of each level of the tree for generating various orders of G and P values. The right most blocks of all levels should use the available value of C0 to compute the output carry directly and term these as the G values for for computation of P and G for the next level.

---------->>>> here I have made a package in which all the small gates are used as components and later then instantiated in main code.



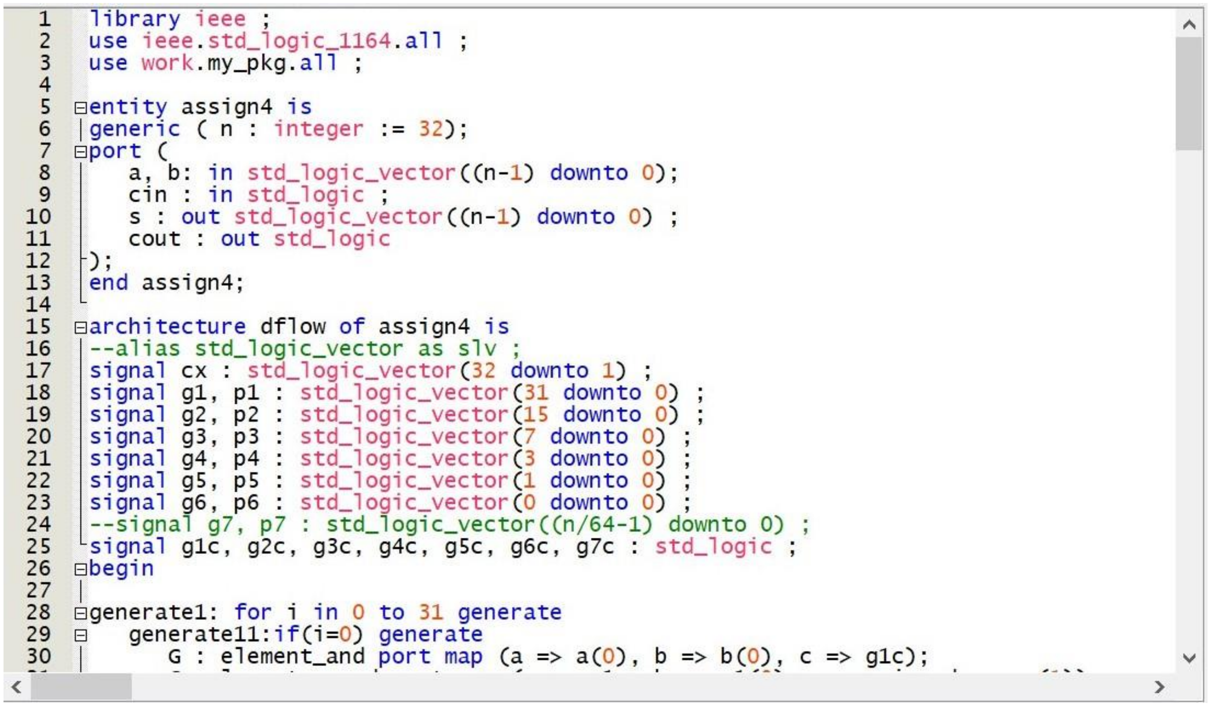
```vhdl
library ieee ;
use ieee.std_logic_1164.all ;


package my_pkg is

component element_xor is
port (
    a, b: in std_logic;
    c : out std_logic
);
end component ;

component element_and is
port (
    a, b: in std_logic;
    c : out std_logic
);
end component ;

component element_orand is
port (
    a, b, c: in std_logic;
    d : out std_logic
);
end component ;

component assign4 is
generic ( n : integer := 8);
port (
```



```vhdl
    a, b: in std_logic;
    c : out std_logic
);
end component ;

component element_and is
port (
    a, b: in std_logic;
    c : out std_logic
);
end component ;

component element_orand is
port (
    a, b, c: in std_logic;
    d : out std_logic
);
end component ;

component assign4 is
generic ( n : integer := 8);
port (
    a, b: in std_logic_vector((n-1) downto 0);
    cin : in std_logic ;
    s : out std_logic_vector((n-1) downto 0) ;
    cout : out std_logic
);
end component ;

end my_pkg;
```

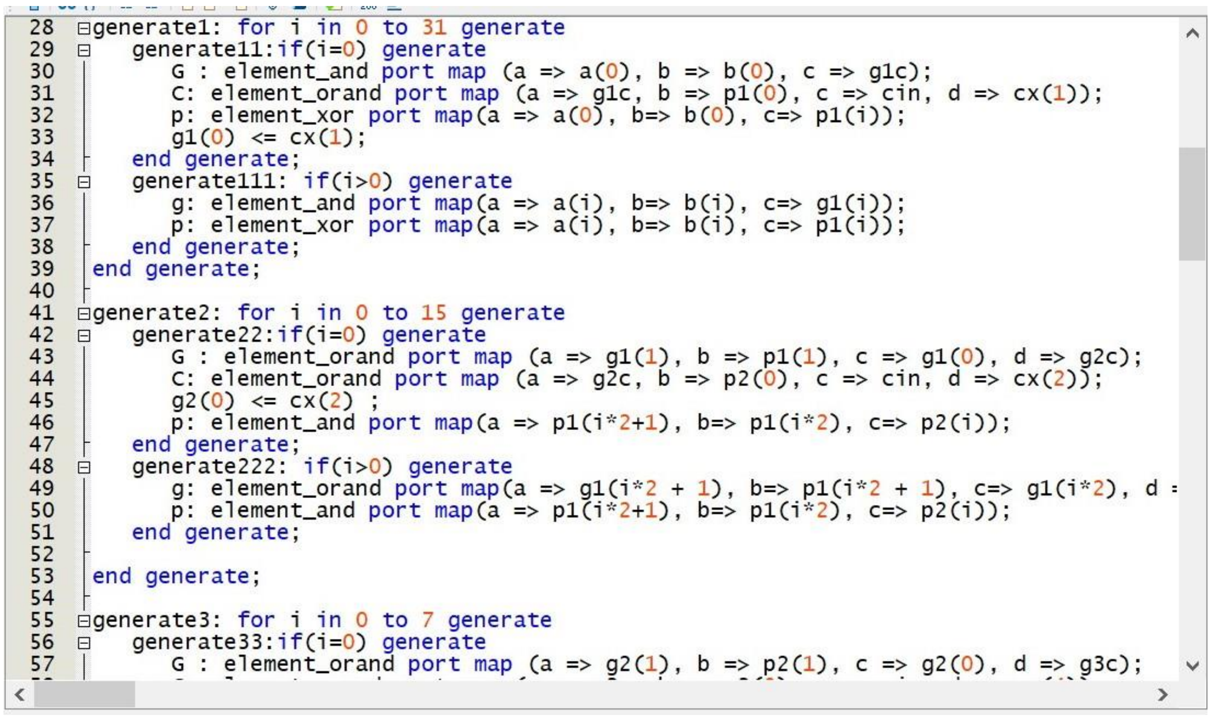
---------------------→>>>>>>

Here is the main design units which is used to generate all the higher and 1 order G and P , all the intermediates carries are generated and all the sums are also generated.

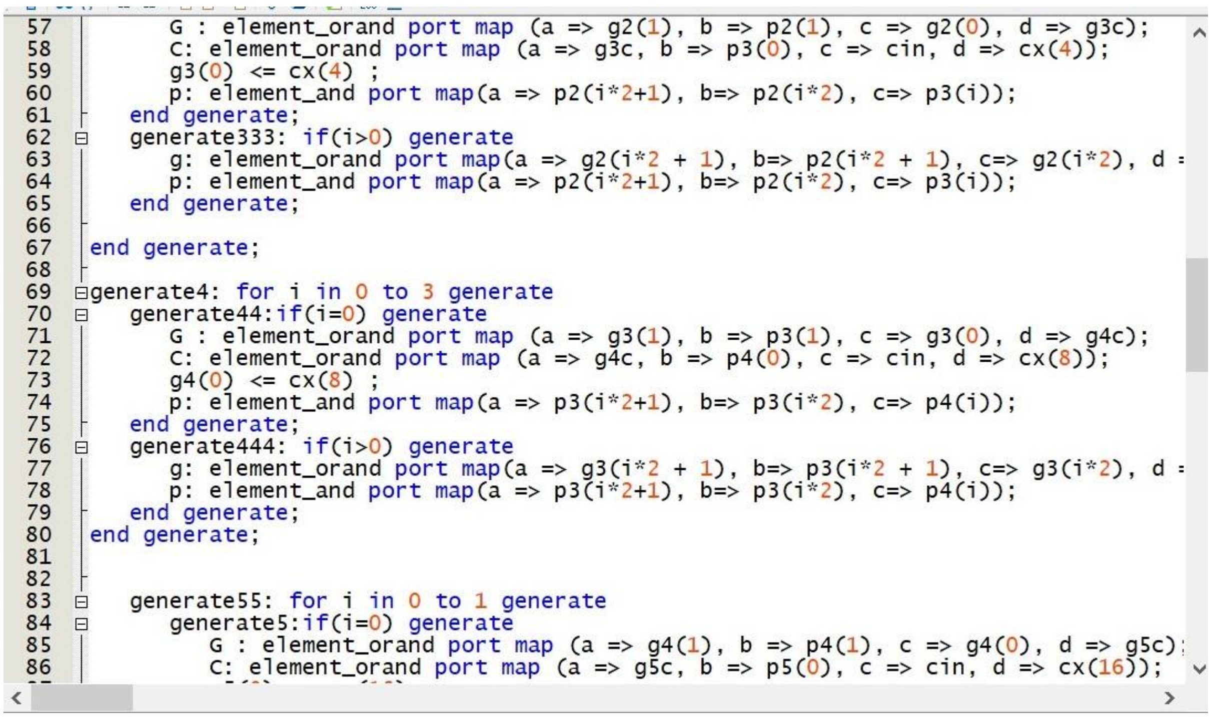


```vhdl
library ieee ;
use ieee.std_logic_1164.all ;
use work.my_pkg.all ;

entity assign4 is
generic ( n : integer := 32);
port (
    a, b: in std_logic_vector((n-1) downto 0);
    cin : in std_logic ;
    s : out std_logic_vector((n-1) downto 0) ;
    cout : out std_logic
);
end assign4;

architecture dflow of assign4 is
--alias std_logic_vector as slv ;
signal cx : std_logic_vector(32 downto 1) ;
signal g1, p1 : std_logic_vector(31 downto 0) ;
signal g2, p2 : std_logic_vector(15 downto 0) ;
signal g3, p3 : std_logic_vector(7 downto 0) ;
signal g4, p4 : std_logic_vector(3 downto 0) ;
signal g5, p5 : std_logic_vector(1 downto 0) ;
signal g6, p6 : std_logic_vector(0 downto 0) ;
--signal g7, p7 : std_logic_vector((n/64-1) downto 0) ;
signal g1c, g2c, g3c, g4c, g5c, g6c, g7c : std_logic ;
begin

generate1: for i in 0 to 31 generate
    generate11:if(i=0) generate
        G : element_and port map (a => a(0), b => b(0), c => g1c);
```

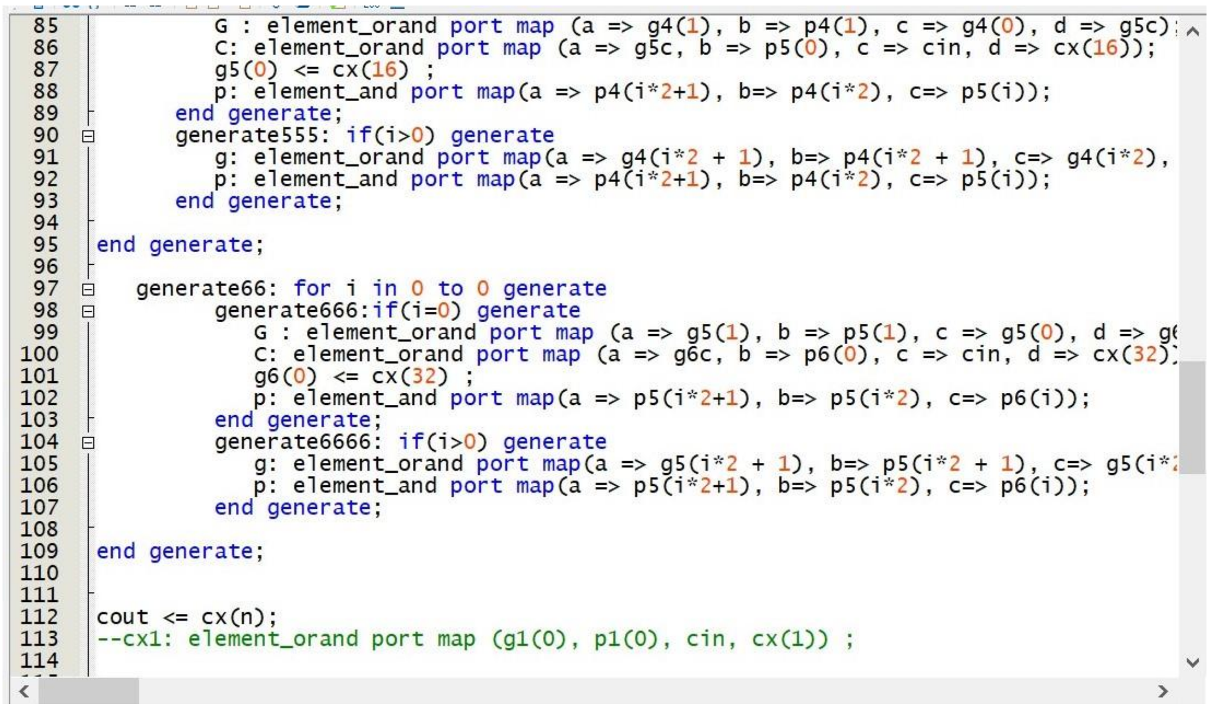
------------------→>>>>>>> **Here I have used generate statement to generate all the G and P in dataflow style modelling.**



```vhdl
generate1: for i in 0 to 31 generate
   generate11:if(i=0) generate
      G : element_and port map (a => a(0), b => b(0), c => g1c);
      C: element_orand port map (a => g1c, b => p1(0), c => cin, d => cx(1));
      p: element_xor port map(a => a(0), b=> b(0), c=> p1(i));
      g1(0) <= cx(1);
   end generate;
   generate111: if(i>0) generate
      g: element_and port map(a => a(i), b=> b(i), c=> g1(i));
      p: element_xor port map(a => a(i), b=> b(i), c=> p1(i));
   end generate;
end generate;

generate2: for i in 0 to 15 generate
   generate22:if(i=0) generate
      G : element_orand port map (a => g1(1), b => p1(1), c => g1(0), d => g2c);
      C: element_orand port map (a => g2c, b => p2(0), c => cin, d => cx(2));
      g2(0) <= cx(2) ;
      p: element_and port map(a => p1(i*2+1), b=> p1(i*2), c=> p2(i));
   end generate;
   generate222: if(i>0) generate
      g: element_orand port map(a => g1(i*2 + 1), b=> p1(i*2 + 1), c=> g1(i*2), d =
      p: element_and port map(a => p1(i*2+1), b=> p1(i*2), c=> p2(i));
   end generate;

end generate;

generate3: for i in 0 to 7 generate
   generate33:if(i=0) generate
      G : element_orand port map (a => g2(1), b => p2(1), c => g2(0), d => g3c);
```

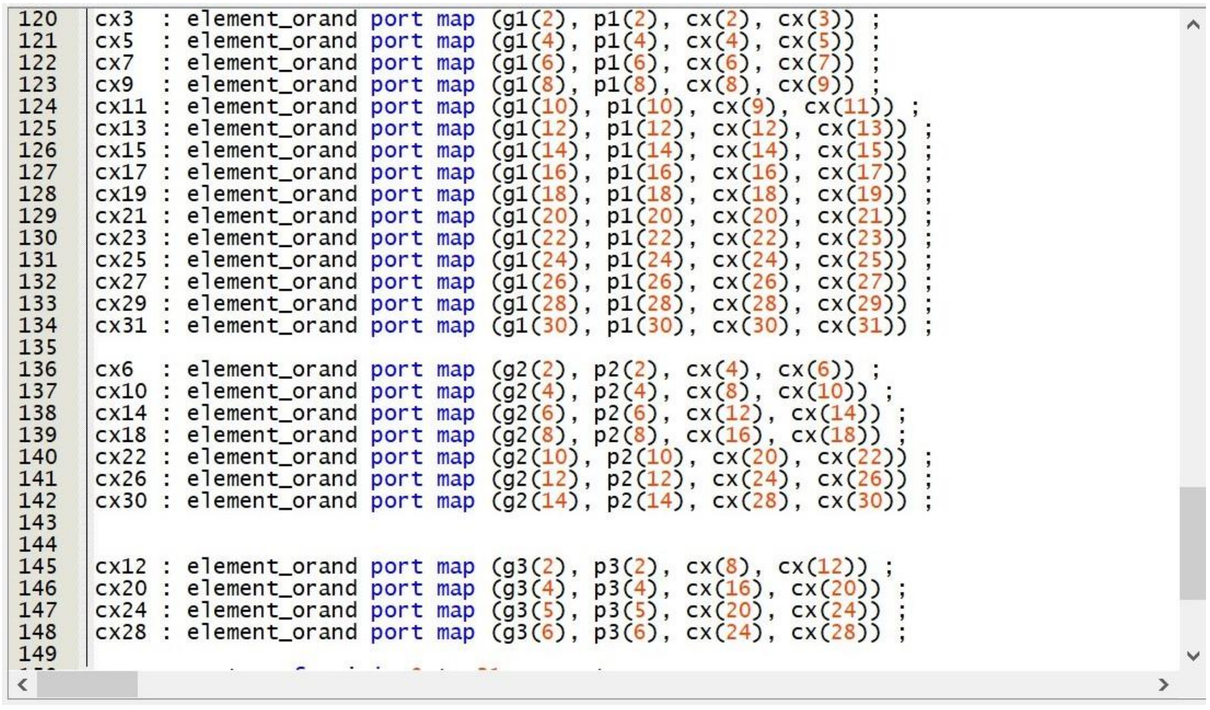


```vhdl
      G : element_orand port map (a => g2(1), b => p2(1), c => g2(0), d => g3c);
      C: element_orand port map (a => g3c, b => p3(0), c => cin, d => cx(4));
      g3(0) <= cx(4) ;
      p: element_and port map(a => p2(i*2+1), b=> p2(i*2), c=> p3(i));
   end generate;
   generate333: if(i>0) generate
      g: element_orand port map(a => g2(i*2 + 1), b=> p2(i*2 + 1), c=> g2(i*2), d =
      p: element_and port map(a => p2(i*2+1), b=> p2(i*2), c=> p3(i));
   end generate;

end generate;

generate4: for i in 0 to 3 generate
   generate44:if(i=0) generate
      G : element_orand port map (a => g3(1), b => p3(1), c => g3(0), d => g4c);
      C: element_orand port map (a => g4c, b => p4(0), c => cin, d => cx(8));
      g4(0) <= cx(8) ;
      p: element_and port map(a => p3(i*2+1), b=> p3(i*2), c=> p4(i));
   end generate;
   generate444: if(i>0) generate
      g: element_orand port map(a => g3(i*2 + 1), b=> p3(i*2 + 1), c=> g3(i*2), d =
      p: element_and port map(a => p3(i*2+1), b=> p3(i*2), c=> p4(i));
   end generate;
end generate;


   generate55: for i in 0 to 1 generate
      generate5:if(i=0) generate
         G : element_orand port map (a => g4(1), b => p4(1), c => g4(0), d => g5c);
         C: element_orand port map (a => g5c, b => p5(0), c => cin, d => cx(16));
```



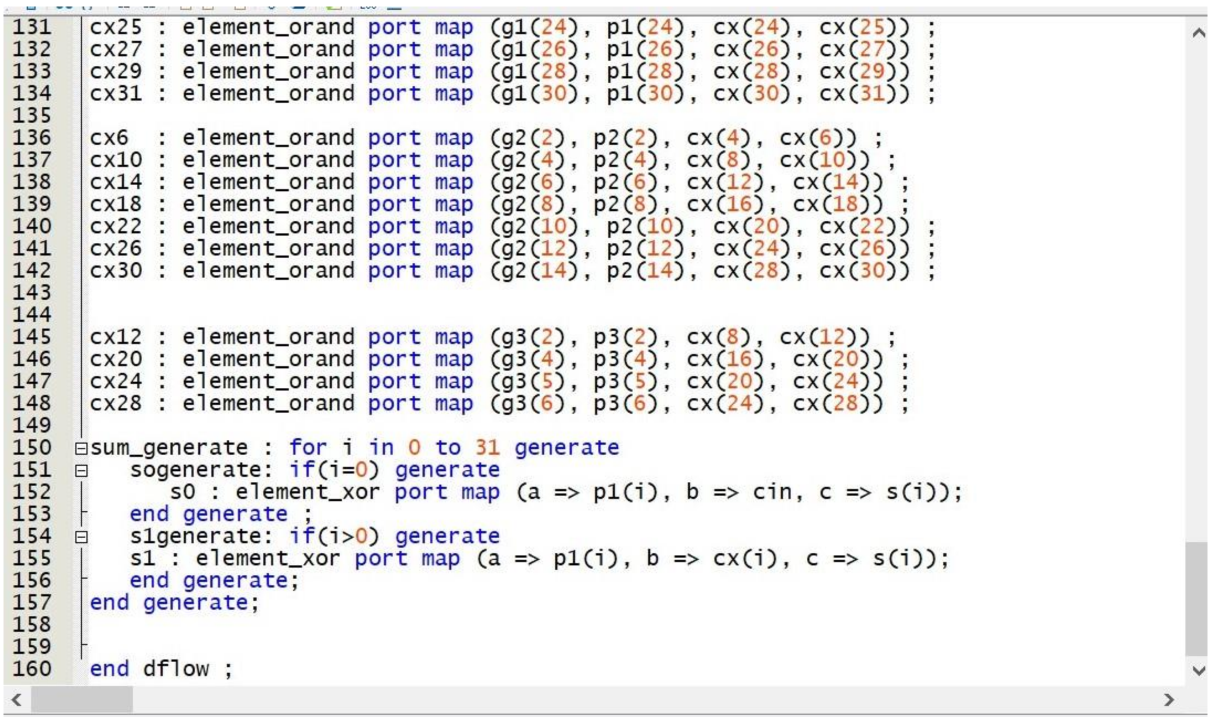
```
         G : element_orand port map (a => g4(1), b => p4(1), c => g4(0), d => g5c);
         C: element_orand port map (a => g5c, b => p5(0), c => cin, d => cx(16));
         g5(0) <= cx(16) ;
         p: element_and port map(a => p4(i*2+1), b=> p4(i*2), c=> p5(i));
      end generate;
      generate555: if(i>0) generate
         g: element_orand port map(a => g4(i*2 + 1), b=> p4(i*2 + 1), c=> g4(i*2),
         p: element_and port map(a => p4(i*2+1), b=> p4(i*2), c=> p5(i));
      end generate;

end generate;

   generate66: for i in 0 to 0 generate
         generate666:if(i=0) generate
            G : element_orand port map (a => g5(1), b => p5(1), c => g5(0), d => g
            C: element_orand port map (a => g6c, b => p6(0), c => cin, d => cx(32)
            g6(0) <= cx(32) ;
            p: element_and port map(a => p5(i*2+1), b=> p5(i*2), c=> p6(i));
         end generate;
         generate6666: if(i>0) generate
            g: element_orand port map(a => g5(i*2 + 1), b=> p5(i*2 + 1), c=> g5(i*
            p: element_and port map(a => p5(i*2+1), b=> p5(i*2), c=> p6(i));
         end generate;

end generate;


cout <= cx(n);
--cx1: element_orand port map (g1(0), p1(0), cin, cx(1)) ;
```

Here I have used all the small gates that I have defined as a component in the my_pkg and used here to generate all the carries.



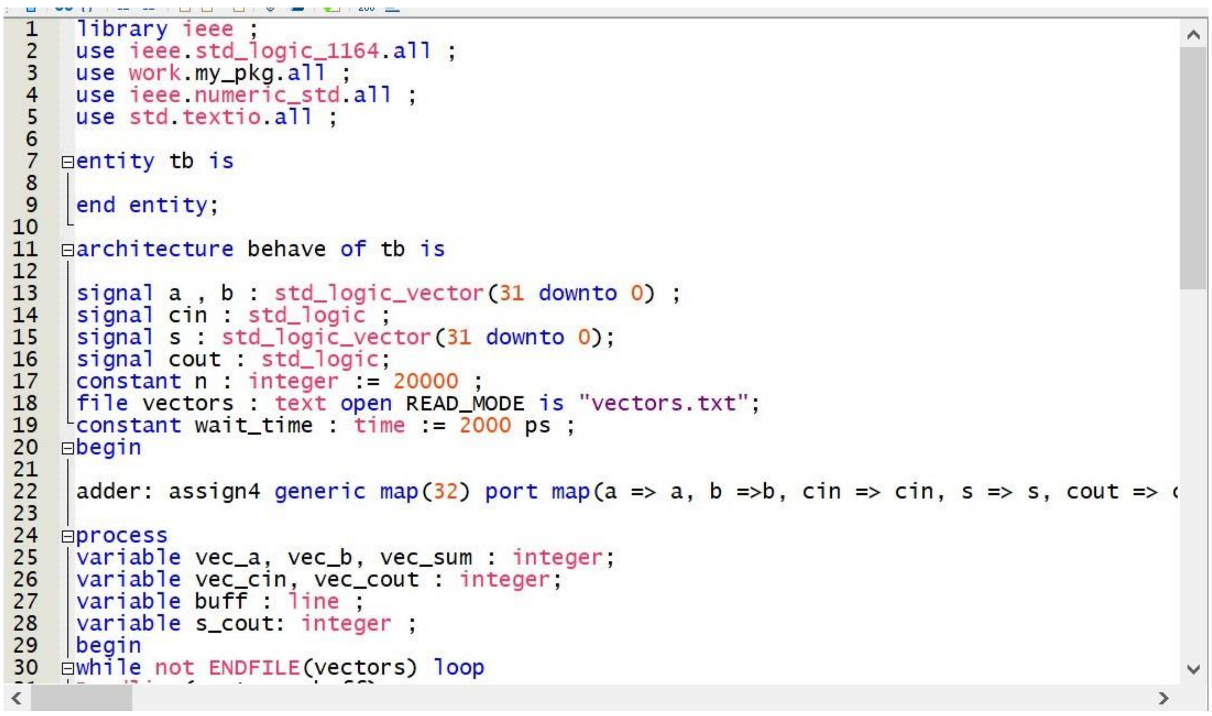
```
cx3  : element_orand port map (g1(2), p1(2), cx(2), cx(3)) ;
cx5  : element_orand port map (g1(4), p1(4), cx(4), cx(5)) ;
cx7  : element_orand port map (g1(6), p1(6), cx(6), cx(7)) ;
cx9  : element_orand port map (g1(8), p1(8), cx(8), cx(9)) ;
cx11 : element_orand port map (g1(10), p1(10), cx(9), cx(11)) ;
cx13 : element_orand port map (g1(12), p1(12), cx(12), cx(13)) ;
cx15 : element_orand port map (g1(14), p1(14), cx(14), cx(15)) ;
cx17 : element_orand port map (g1(16), p1(16), cx(16), cx(17)) ;
cx19 : element_orand port map (g1(18), p1(18), cx(18), cx(19)) ;
cx21 : element_orand port map (g1(20), p1(20), cx(20), cx(21)) ;
cx23 : element_orand port map (g1(22), p1(22), cx(22), cx(23)) ;
cx25 : element_orand port map (g1(24), p1(24), cx(24), cx(25)) ;
cx27 : element_orand port map (g1(26), p1(26), cx(26), cx(27)) ;
cx29 : element_orand port map (g1(28), p1(28), cx(28), cx(29)) ;
cx31 : element_orand port map (g1(30), p1(30), cx(30), cx(31)) ;

cx6  : element_orand port map (g2(2), p2(2), cx(4), cx(6)) ;
cx10 : element_orand port map (g2(4), p2(4), cx(8), cx(10)) ;
cx14 : element_orand port map (g2(6), p2(6), cx(12), cx(14)) ;
cx18 : element_orand port map (g2(8), p2(8), cx(16), cx(18)) ;
cx22 : element_orand port map (g2(10), p2(10), cx(20), cx(22)) ;
cx26 : element_orand port map (g2(12), p2(12), cx(24), cx(26)) ;
cx30 : element_orand port map (g2(14), p2(14), cx(28), cx(30)) ;


cx12 : element_orand port map (g3(2), p3(2), cx(8), cx(12)) ;
cx20 : element_orand port map (g3(4), p3(4), cx(16), cx(20)) ;
cx24 : element_orand port map (g3(5), p3(5), cx(20), cx(24)) ;
cx28 : element_orand port map (g3(6), p3(6), cx(24), cx(28)) ;
```

here I have used generate statements again to generate all sum bits

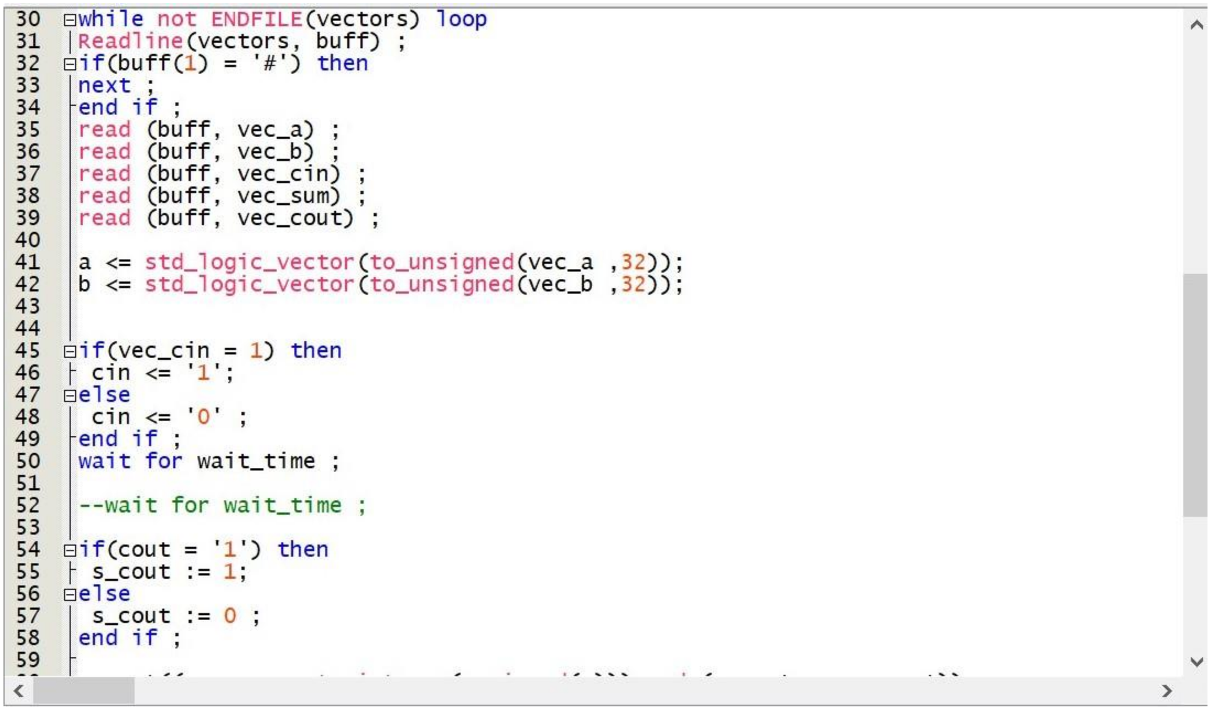


```vhdl
cx25 : element_orand port map (g1(24), p1(24), cx(24), cx(25)) ;
cx27 : element_orand port map (g1(26), p1(26), cx(26), cx(27)) ;
cx29 : element_orand port map (g1(28), p1(28), cx(28), cx(29)) ;
cx31 : element_orand port map (g1(30), p1(30), cx(30), cx(31)) ;

cx6  : element_orand port map (g2(2), p2(2), cx(4), cx(6)) ;
cx10 : element_orand port map (g2(4), p2(4), cx(8), cx(10)) ;
cx14 : element_orand port map (g2(6), p2(6), cx(12), cx(14)) ;
cx18 : element_orand port map (g2(8), p2(8), cx(16), cx(18)) ;
cx22 : element_orand port map (g2(10), p2(10), cx(20), cx(22)) ;
cx26 : element_orand port map (g2(12), p2(12), cx(24), cx(26)) ;
cx30 : element_orand port map (g2(14), p2(14), cx(28), cx(30)) ;


cx12 : element_orand port map (g3(2), p3(2), cx(8), cx(12)) ;
cx20 : element_orand port map (g3(4), p3(4), cx(16), cx(20)) ;
cx24 : element_orand port map (g3(5), p3(5), cx(20), cx(24)) ;
cx28 : element_orand port map (g3(6), p3(6), cx(24), cx(28)) ;

sum_generate : for i in 0 to 31 generate
   sogenerate: if(i=0) generate
      s0 : element_xor port map (a => p1(i), b => cin, c => s(i));
   end generate ;
   s1generate: if(i>0) generate
   s1 : element_xor port map (a => p1(i), b => cx(i), c => s(i));
   end generate;
end generate;


end dflow ;
```

c) Using the outputs of the tree above, write structural VHDL code for generating the bit wise sum and carry values. Test the final adder with a test bench which reads pairs of 32 bit words and a single bit input carry from a file, adds them and compares the result with the expected 32 bit sum and 1 bit carry values stored in the same file. It should use assert statements to flag errors if there is a mismatch between the computed sum/carry and the stored sum/carry. Test the design with 64 randomly chosen pairs of numbers and input carry to be added.

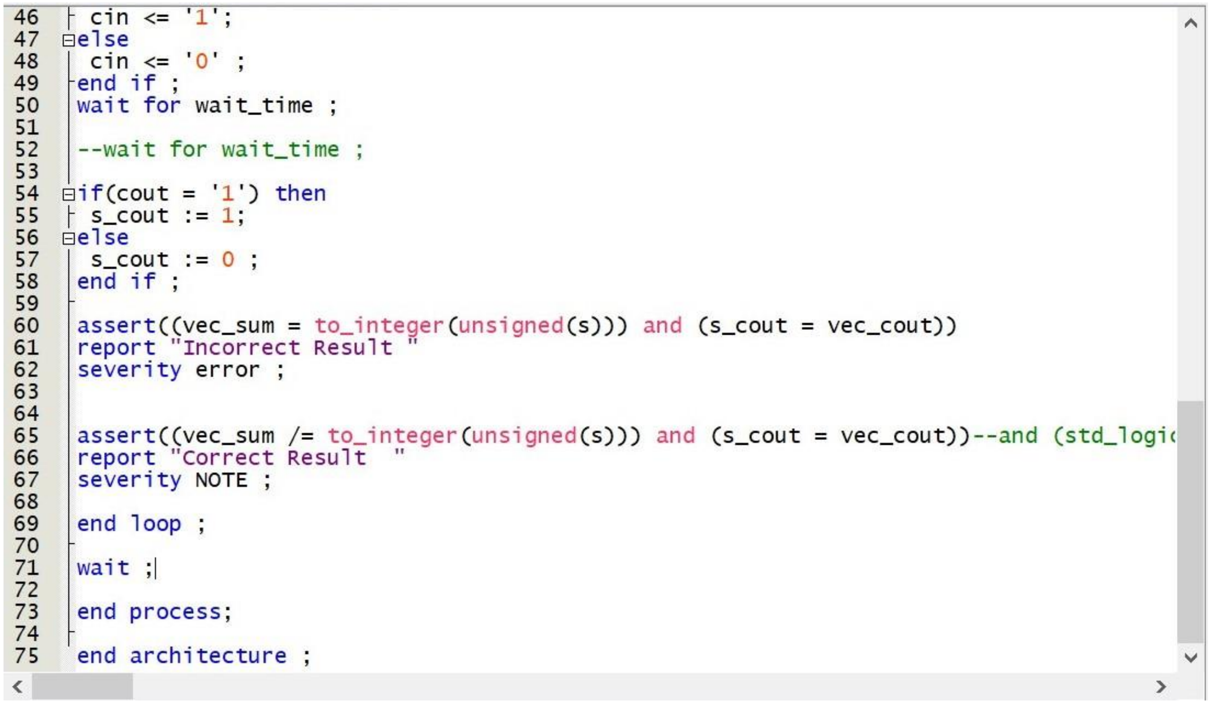


```vhdl
library ieee ;
use ieee.std_logic_1164.all ;
use work.my_pkg.all ;
use ieee.numeric_std.all ;
use std.textio.all ;

entity tb is

end entity;

architecture behave of tb is

signal a , b : std_logic_vector(31 downto 0) ;
signal cin : std_logic ;
signal s : std_logic_vector(31 downto 0);
signal cout : std_logic;
constant n : integer := 20000 ;
file vectors : text open READ_MODE is "vectors.txt";
constant wait_time : time := 2000 ps ;
begin

adder: assign4 generic map(32) port map(a => a, b =>b, cin => cin, s => s, cout =>

process
variable vec_a, vec_b, vec_sum : integer;
variable vec_cin, vec_cout : integer;
variable buff : line ;
variable s_cout: integer ;
begin
while not ENDFILE(vectors) loop
```



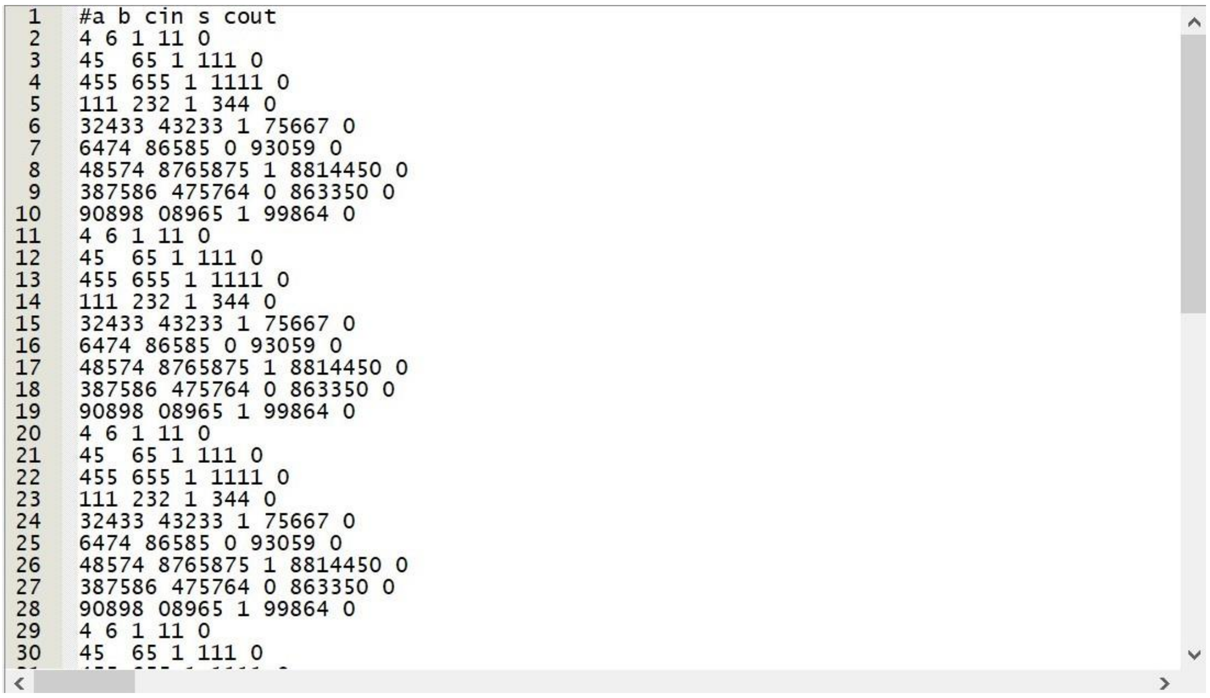
```
while not ENDFILE(vectors) loop
Readline(vectors, buff) ;
if(buff(1) = '#') then
next ;
end if ;
read (buff, vec_a) ;
read (buff, vec_b) ;
read (buff, vec_cin) ;
read (buff, vec_sum) ;
read (buff, vec_cout) ;

a <= std_logic_vector(to_unsigned(vec_a ,32));
b <= std_logic_vector(to_unsigned(vec_b ,32));


if(vec_cin = 1) then
 cin <= '1';
else
 cin <= '0' ;
end if ;
wait for wait_time ;

--wait for wait_time ;

if(cout = '1') then
 s_cout := 1;
else
 s_cout := 0 ;
end if ;

```

here I have used assert statements to check whether the calculated results are correct or not



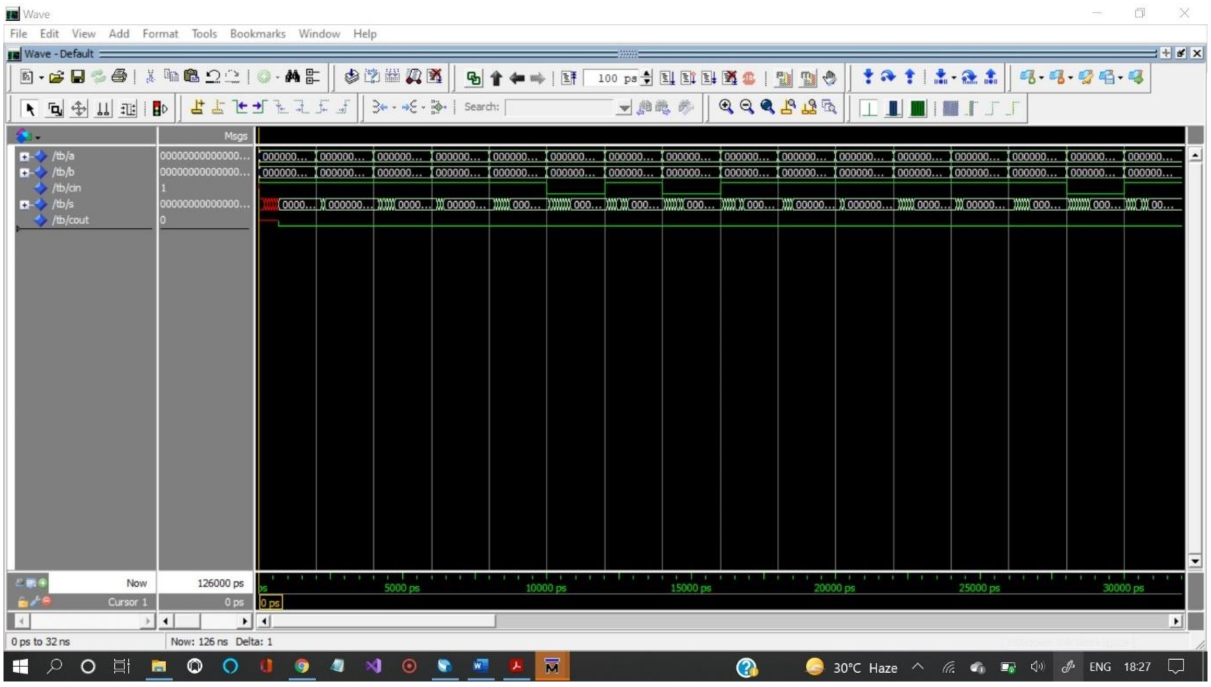
```
 cin <= '1';
else
 cin <= '0' ;
end if ;
wait for wait_time ;

--wait for wait_time ;

if(cout = '1') then
 s_cout := 1;
else
 s_cout := 0 ;
end if ;

assert((vec_sum = to_integer(unsigned(s))) and (s_cout = vec_cout))
report "Incorrect Result "
severity error ;


assert((vec_sum /= to_integer(unsigned(s))) and (s_cout = vec_cout))--and (std_logi
report "Correct Result  "
severity NOTE ;

end loop ;

wait ;

end process;

end architecture ;
```

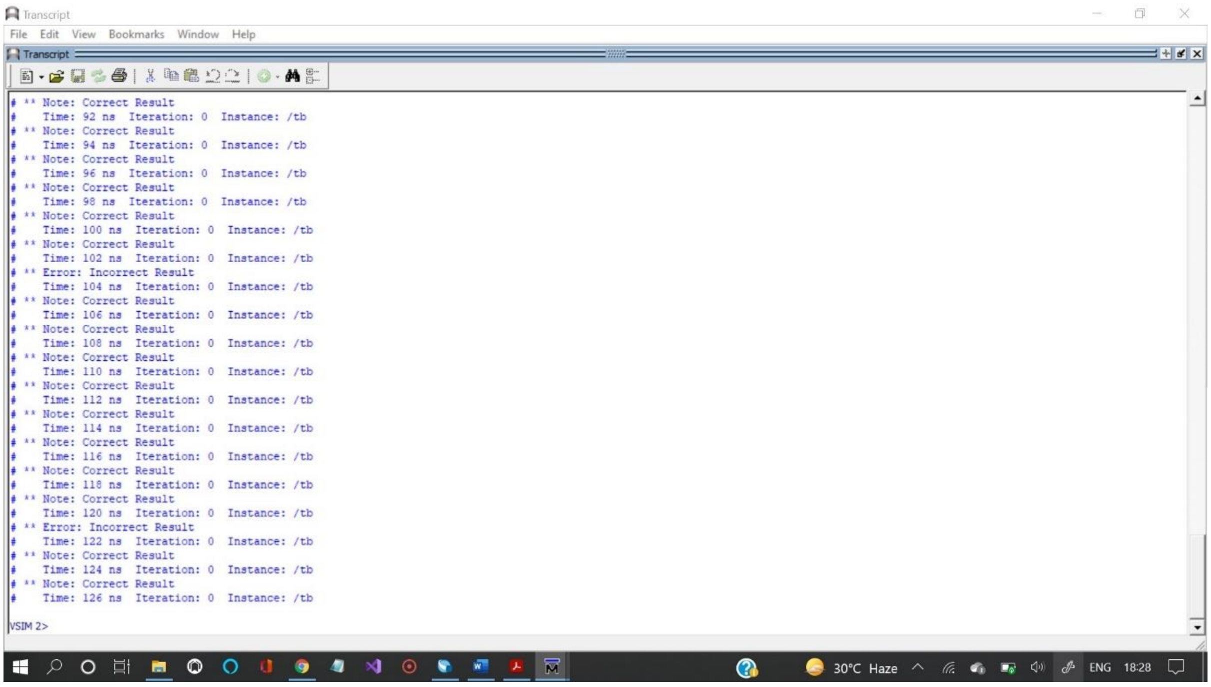
here is the text file I have used to compare the results with the calculated results.



```
#a b cin s cout
4 6 1 11 0
45  65 1 111 0
455 655 1 1111 0
111 232 1 344 0
32433 43233 1 75667 0
6474 86585 0 93059 0
48574 8765875 1 8814450 0
387586 475764 0 863350 0
90898 08965 1 99864 0
4 6 1 11 0
45  65 1 111 0
455 655 1 1111 0
111 232 1 344 0
32433 43233 1 75667 0
6474 86585 0 93059 0
48574 8765875 1 8814450 0
387586 475764 0 863350 0
90898 08965 1 99864 0
4 6 1 11 0
45  65 1 111 0
455 655 1 1111 0
111 232 1 344 0
32433 43233 1 75667 0
6474 86585 0 93059 0
48574 8765875 1 8814450 0
387586 475764 0 863350 0
90898 08965 1 99864 0
4 6 1 11 0
45  65 1 111 0
```

Here is the simulation wave diagram and the following simulation result to check the results with results of the file.





```
# ** Note: Correct Result
#    Time: 92 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 94 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 96 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 98 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 100 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 102 ns  Iteration: 0  Instance: /tb
# ** Error: Incorrect Result
#    Time: 104 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 106 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 108 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 110 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 112 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 114 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 116 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 118 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 120 ns  Iteration: 0  Instance: /tb
# ** Error: Incorrect Result
#    Time: 122 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 124 ns  Iteration: 0  Instance: /tb
# ** Note: Correct Result
#    Time: 126 ns  Iteration: 0  Instance: /tb

VSIM 2>
```