Assignment - 3

32- bit Brent Kung Adder

EE 671 - VLSI Design

Name: **Sanchit Gupta.**

Roll number: **23M1114.**

Program: M.tech (EE5 - Electronic Systems).

Department: Electrical Engineering.

Date of Sub: 7th Oct, 2023.

Ques 1: Describe a 32 bit Brent Kung adder in VHDL and simulate it using a test bench. Your description should use std_logic types for various signals.

a) Logic functions AND, XOR, A + B.C and A.B + C.(A+B) are required for computing different orders of G, P and final sum and carry outputs. A template file with entity/architecture pairs for these functions is appended at the end. You should modify that code to implement delays for logic functions as given below: Function Delay in ps

Function	Delay in ps
AND	300 + last two digit of your roll number
A + B.C	400 + last two digits of your roll number
XOR	600 + 2*last two digits of your roll number
A.B + C.(A+B)	600 + 2*last two digits of your roll number

Soln: My Roll no. last two digits are 14, so I add 14ps and 28ps of delay to the respective gates.

andgate.vhdl | xorgate.vhdl | abcgate.vhdl | cin_

```
1   -- simple AND gates with trivial architectures
2   library IEEE;
3   use IEEE.std_logic_1164.all;
4
5   entity andgate is
6   port (A, B: in std_logic;
7         prod: out std_logic);
8   end entity andgate;
9
10  architecture trivial of andgate is
11  begin
12  prod <= A AND B AFTER 314 ps;
13  end architecture trivial;
14
15
```

andgate.vhdl | xorgate.vhdl | abcgate.vhdl | cir

```
1   -- simple XOR gates with trivial architectures
2
3   library IEEE;
4   use IEEE.std_logic_1164.all;
5
6   entity xorgate is
7   port (A, B: in std_logic;
8         uneq: out std_logic);
9   end entity xorgate;
10
11  architecture trivial of xorgate is
12  begin
13  uneq <= A XOR B AFTER 628 ps;
14  end architecture trivial;
15
```

andgate.vhdl | xorgate.vhdl | abcgate.vhdl | ci

```vhdl
-- simple ABC gates with trivial architectures
library IEEE;
use IEEE.std_logic_1164.all;

entity abcgate is
port (A, B, C: in std_logic;
      abc: out std_logic);
end entity abcgate;

architecture trivial of abcgate is
begin
    abc <= A OR (B AND C) AFTER 414 ps;
end architecture trivial;
```

andgate.vhdl | xorgate.vhdl | abcgate.vhdl | cin_map_G.vhdl

```vhdl
-- A + C.(A+B) with a trivial architecture
library IEEE;
use IEEE.std_logic_1164.all;

entity Cin_map_G is
port(A, B, Cin: in std_logic;
     Bit0_G: out std_logic);
end entity Cin_map_G;

architecture trivial of Cin_map_G is
begin
Bit0_G <= (A AND B) OR (Cin AND (A OR B)) AFTER 628 ps;
end architecture trivial;
```

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 rightmost blocks of all levels should use the already available value of C0 to compute the output carry directly using the logic block for A.B + C.(A+B) and use these instead of the G values for computation of P and G for the next level.

Soln: The VHDL Code is as follows:

adder.vhdl* | DUT.vhdl | Testbench_org.vhdl

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity adder is
5      port (a,b: in std_logic_vector(31 downto 0);
6            c : in std_logic;
7            sum : out std_logic_vector(31 downto 0);
8            cout : out std_logic);
9
10 end entity;
11
12 architecture trivial of adder is
13
14 signal carry : std_logic_vector(32 downto 1);
15
16 component andgate
17 port (A, B: in std_logic;
18       prod: out std_logic);
19 end component andgate;
20
21 component xorgate
22 port (A, B: in std_logic;
23       uneq: out std_logic);
24 end component xorgate;
25
26 component abcgate
27 port (A, B, C: in std_logic;
28       abc: out std_logic);
29 end component;
30
31 component Cin_map_G
32 port(A, B, Cin: in std_logic;
33      Bit0_G: out std_logic);
34 end component Cin_map_G;
35
36 signal G1, P1 :  std_logic_vector(31 downto 0);
37 signal G2, P2 :  std_logic_vector(15 downto 0);
38 signal G3, P3 :  std_logic_vector(7 downto 0);
39 signal G4, P4 :  std_logic_vector(3 downto 0);
40 signal G5, P5 :  std_logic_vector(1 downto 0);
```

adder.vhdl* | DUT.vhdl | Testbench_org.vhdl

```
41 signal G6, P6 :  std_logic;
42 begin
43
44 G1_0: abcgate port map (a(0), b(0), c, G1(0));
45
46 s1: for i in 31 downto 1 generate
47  G1_i : andgate port map (a(i), b(i), G1(i));
48 end generate s1;
49
50 s2 : for i in 31 downto 0 generate
51  P1_i: xorgate port map(a(i), b(i), P1(i));
52 end generate s2;
53
54 s3: for i in 15 downto 0 generate
55  G2_i: abcgate port map (G1((2*i)+1), P1((2*i)+1), G1(2*i), G2(i));
56 end generate s3;
57
58 s4 : for i in 15 downto 0 generate
59  P2_i: andgate port map (P1((2*i)+1), P1(2*i), P2(i));
60 end generate s4;
61
62 s5 : for i in 7 downto 0 generate
63  G3_i: abcgate port map (G2((2*i)+1), P2((2*i)+1), G2((2*i)), G3(i));
64 end generate s5;
65
66 s6 : for i in 7 downto 0 generate
67  P3_i: andgate port map (P2((2*i)+1), P2((2*i)), P3(i));
68 end generate s6;
69
70 s7 : for i in 3 downto 0 generate
71  G4_i: abcgate port map (G3((2*i)+1), P3((2*i)+1), G3((2*i)), G4(i));
72 end generate s7;
73
74 s8 : for i in 3 downto 0 generate
75  P4_i: andgate port map (P3((2*i)+1), P3((2*i)), P4(i));
76 end generate s8;
77
78 s9 : for i in 1 downto 0 generate
79  G5_i: abcgate port map (G4((2*i)+1), P4((2*i)+1), G4((2*i)), G5(i));
80 end generate s9;
```

adder.vhdl* DUT.vhdl Testbench_org.vhdl

```
81
82  s10 : for i in 1 downto 0 generate
83   P5_i: andgate port map (P4((2*i)+1), P4((2*i)), P5(i));
84  end generate s10;
85
86  G6_i: abcgate port map (G5(1), P5(1), G5(0), G6);
87  P6_i : andgate port map (P5(1), P5(0), P6);
88
89  carry(1) <= G1(0);
90  carry(2) <= G2(0);
91  carry_3: abcgate port map (G1(2), P1(2), carry(2), carry(3));
92  carry(4) <= G3(0);
93  carry_5: abcgate port map (G1(4), P1(4), carry(4), carry(5));
94  carry_6: abcgate port map (G2(2), P2(2), carry(4), carry(6));
95  carry_7: abcgate port map (G1(6), P1(6), carry(6), carry(7));
96  carry(8) <= G4(0);
97  carry_9: abcgate port map (G1(8), P1(8), carry(8), carry(9));
98  carry_10: abcgate port map (G2(4), P2(4), carry(8), carry(10));
99  carry_11: abcgate port map (G1(10), P1(10), carry(10), carry(11));
100 carry_12: abcgate port map (G3(2), P3(2), carry(8), carry(12));
101 carry_13: abcgate port map (G1(12), P1(12), carry(12), carry(13));
102 carry_14: abcgate port map (G2(6), P2(6), carry(12), carry(14));
103 carry_15: abcgate port map (G1(14), P1(14), carry(14), carry(15));
104 carry(16) <= G5(0);
105 carry_17: abcgate port map (G1(16), P1(16), carry(16), carry(17));
106 carry_18: abcgate port map (G2(8), P2(8), carry(16), carry(18));
107 carry_19: abcgate port map (G1(18), P1(18), carry(18), carry(19));
108 carry_20: abcgate port map (G3(4), P3(4), carry(16), carry(20));
109 carry_21: abcgate port map (G1(20), P1(20), carry(20), carry(21));
110 carry_22: abcgate port map (G2(10), P2(10), carry(20), carry(22));
111 carry_23: abcgate port map (G1(22), P1(22), carry(22), carry(23));
112 carry_24: abcgate port map (G4(2), P4(2), carry(16), carry(24));
113 carry_25: abcgate port map (G1(24), P1(24), carry(24), carry(25));
114 carry_26: abcgate port map (G2(12), P2(12), carry(24), carry(26));
115 carry_27: abcgate port map (G1(26), P1(26), carry(26), carry(27));
116 carry_28: abcgate port map (G3(6), P3(6), carry(24), carry(28));
117 carry_29: abcgate port map (G1(28), P1(28), carry(28), carry(29));
118 carry_30: abcgate port map (G2(14), P2(14), carry(28), carry(30));
119 carry_31: abcgate port map (G1(30), P1(30), carry(30), carry(31));
120 carry(32) <= G6;
```

```
121
122  cout <= carry(32);
123  sum_0 : xorgate port map (P1(0), c, sum(0));
124  s11 : for i in 31 downto 1 generate
125   sum_i : xorgate port map (P1(i),carry(i), sum(i));
126  end generate s11;
127
128  end architecture;
```

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 16 bit words and a single bit input carry value from a file, applies these to the adder and compares the result with the expected 16 bit sum and 1 bit carry values stored in the same file. This test should be carried out for 10 randomly chosen input combinations. It should use assert statements to flag errors if there is a mismatch between the computed sum/carry and the stored sum/carry.

Sol: The Tracefile is as follows:

Firstly, I have given 64 bit input which is followed by 33 bit expected output and then at last i have mask all the bits as i want to compare all of them.

Here, I have given 10 randomly chosen input combinations and their respective sum along with their final carry.

Now, I will compare these stored sum/carry with the output computed sum/carry.

TRACEFILE - Notepad

File Edit Format View Help

```
11001011001010110101010110101001110010101100101011000101110010101 110010101111101100001101101110100 111111111111111111111111111111111
11010101101101101101010101101101110101011011111110101010111011010 110101011011101101000000001011010 111111111111111111111111111111111
11010101101110110100000000101101110101011011101101111110001011001 110101011011101101011111001011010 111111111111111111111111111111111
11010101101110110101111100101010110101011011101101000111001010100 110101011011101101010011001010100 111111111111111111111111111111111
11010101101110110101001100101011110101011011101110110011001010101 110101011011101110000011001010110 111111111111111111111111111111111
11010101101101101101010101101101110101011011111110101010111011010 110101011011101101000000001011010 111111111111111111111111111111111
11010101101110110100000000101101110101011011101101111110001011001 110101011011101101011111001011010 111111111111111111111111111111111
11001011001010110101010110101001110010101100101011000101110010101 110010101111101100001101101110100 111111111111111111111111111111111
11010101101101101101010101101101110101011011111110101010111011010 110101011011101101000000001011010 111111111111111111111111111111111
11010101101110110100000000101101110101011011101101111110001011001 110101011011101101011111001011010 111111111111111111111111111111111
```

The code for DUT is given below:

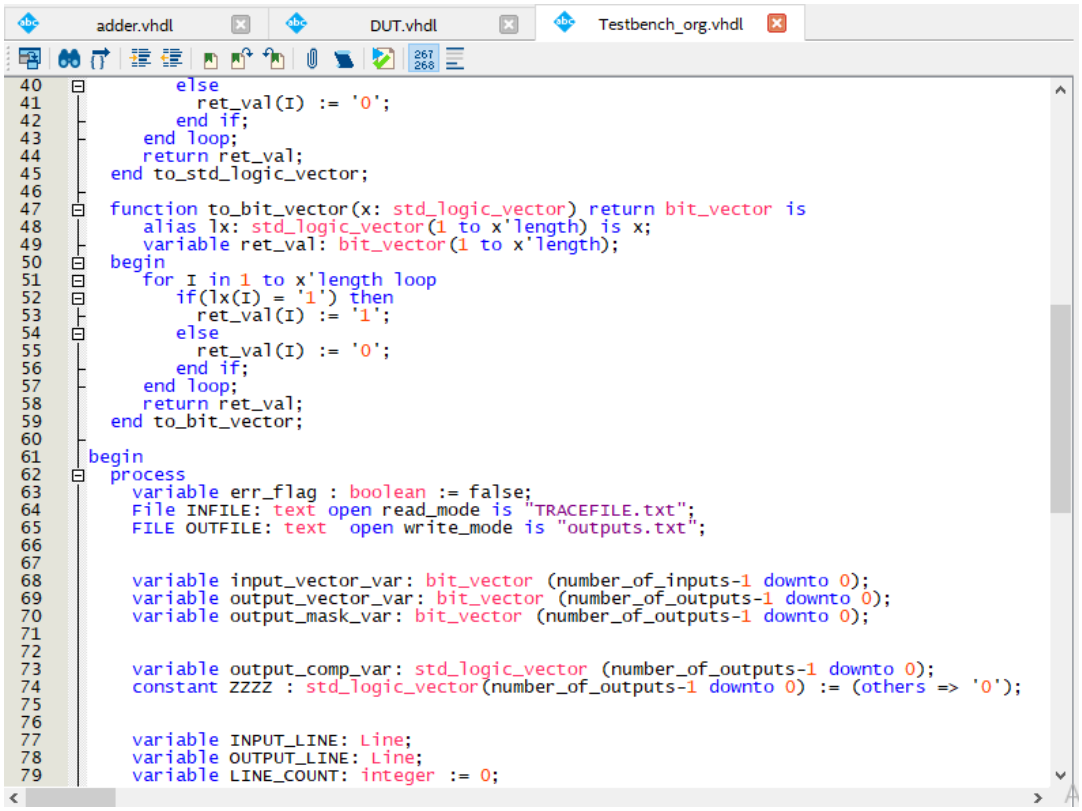
adder.vhdl* | DUT.vhdl | Testbench_org.vhdl

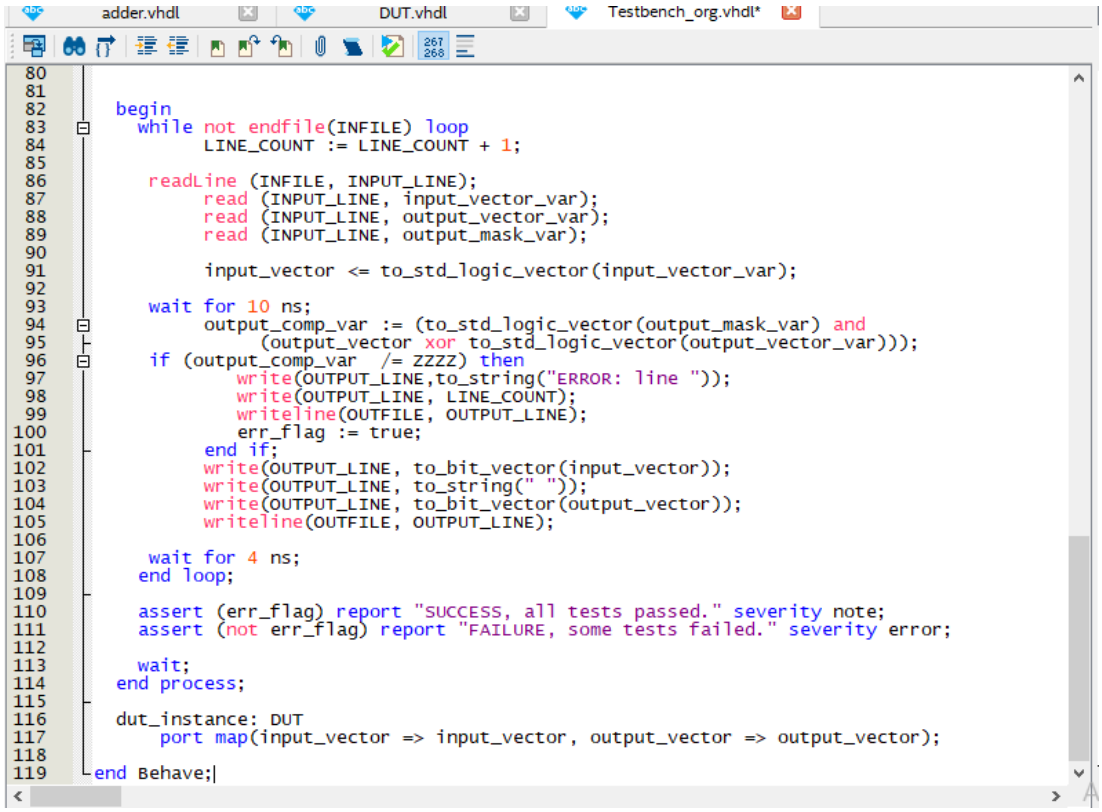
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity DUT is
   port(input_vector: in std_logic_vector(64 downto 0);
        output_vector: out std_logic_vector(32 downto 0));
end entity;

architecture DutWrap of DUT is

  component adder is
        port (
            a,b: in std_logic_vector(31 downto 0);
         c : in std_logic;
      sum : out std_logic_vector(31 downto 0);
         cout : out std_logic );
     end component adder;

begin

   add_instance: adder port map (a => input_vector(64 downto 33),
        b => input_vector(32 downto 1),
        c => input_vector(0),
        sum => output_vector(31 downto 0),
        cout => output_vector(32));
end DutWrap;
```

The Testbench Code is as follows:

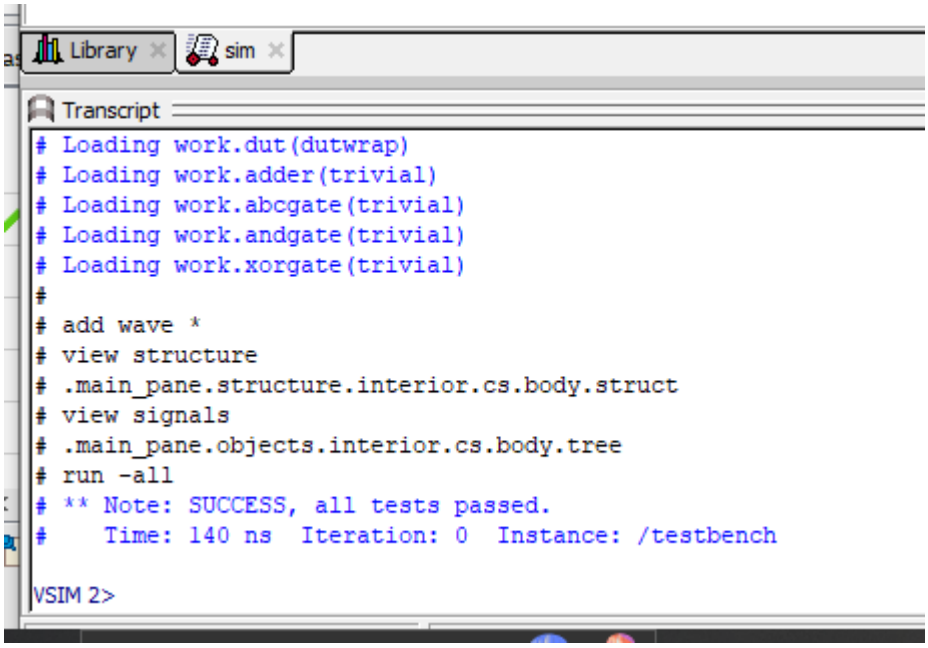
adder.vhdl | DUT.vhdl | Testbench_org.vhdl

```vhdl
library std;
use std.textio.all;

library ieee;
use ieee.std_logic_1164.all;

entity Testbench is
end entity;
architecture Behave of Testbench is


  constant number_of_inputs  : integer := 65;  -- input bits
  constant number_of_outputs : integer := 33;  -- output bits

  component DUT is
   port(input_vector: in std_logic_vector(number_of_inputs-1 downto 0);
         output_vector: out std_logic_vector(number_of_outputs-1 downto 0));
  end component;


  signal input_vector  : std_logic_vector(number_of_inputs-1 downto 0);
  signal output_vector : std_logic_vector(number_of_outputs-1 downto 0);

  function to_string(x: string) return string is
      variable ret_val: string(1 to x'length);
      alias lx : string (1 to x'length) is x;
  begin
      ret_val := lx;
      return(ret_val);
  end to_string;

  function to_std_logic_vector(x: bit_vector) return std_logic_vector is
     alias lx: bit_vector(1 to x'length) is x;
     variable ret_val: std_logic_vector(1 to x'length);
  begin
     for I in 1 to x'length loop
        if(lx(I) = '1') then
          ret_val(I) := '1';
        else
```

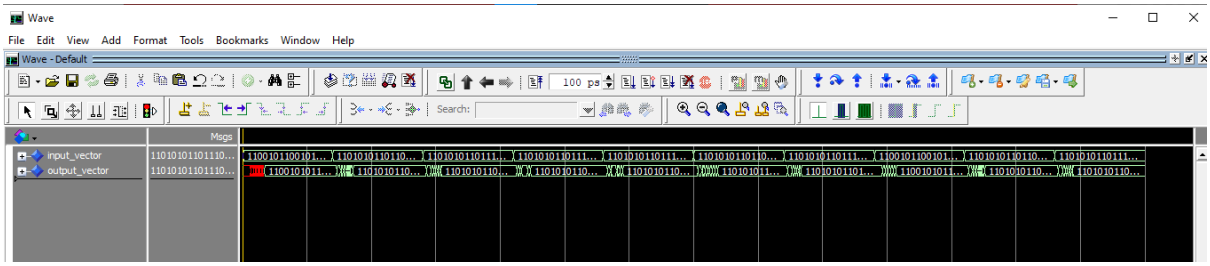




So, when I perform RTL Simulation then it assert SUCCESS which means that there is no mismatch between the computed sum/carry and the stored sum/carry.



This is the respective input output waveform.



The Output File generated by Modelsim is shown below:

outputs - Notepad

File Edit Format View Help

```
1100101100101011010101011010010011100101011001010110001011100101011100010111001010111111011000011011011101001100101100101011010101011010010011100101011001010110001011100101011100010111001010111111011000011011011101001101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011101011011001011010101010110110111010101101111111101010101111011010100000000010110111010101101110110111111100010110011
```