

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.

```

1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4  entity element_and is
5  port (
6      a, b: in std_logic;
7      c : out std_logic
8  );
9  end element_and;
10
11 architecture dflow of element_and is
12
13 begin
14
15     c <= a and b after 100 ps;
16
17 end dflow ;

```

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

```

1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4  entity element_orand is
5  port (
6      a, b, c: in std_logic;
7      d : out std_logic
8  );
9  end element_orand;
10
11 architecture dflow of element_orand is
12
13 begin
14
15     d <= a or ( b and c) after 100 ps ;
16
17 end dflow ;

```

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

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4  entity element_xor is
5  port (
6      a, b: in std_logic;
7      c : out std_logic
8  );
9  end element_xor;
10
11 architecture dflow of element_xor is
12 begin
13     c <= a xor b after 100 ps;
14 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.

```
1  library ieee ;
2  use ieee.std_logic_1164.all ;
3
4
5  package my_pkg is
6  component element_xor is
7  port (
8      a, b: in std_logic;
9      c : out std_logic
10 );
11 end component ;
12
13 component element_and is
14 port (
15     a, b: in std_logic;
16     c : out std_logic
17 );
18 end component ;
19
20 component element_orand is
21 port (
22     a, b, c: in std_logic;
23     d : out std_logic
24 );
25 end component ;
26
27 component assign4 is
28 generic ( n : integer := 8);
29 port (
```

```

9      a, b: in std_logic;
10      c : out std_logic
11  );
12  end component ;
13
14  component element_and is
15  port (
16      a, b: in std_logic;
17      c : out std_logic
18  );
19  end component ;
20
21  component element_orand is
22  port (
23      a, b, c: in std_logic;
24      d : out std_logic
25  );
26  end component ;
27
28  component assign4 is
29  generic ( n : integer := 8);
30  port (
31      a, b: in std_logic_vector((n-1) downto 0);
32      cin : in std_logic ;
33      s : out std_logic_vector((n-1) downto 0) ;
34      cout : out std_logic
35  );
36  end component ;
37
38  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.

```

1  library ieee ;
2  use ieee.std_logic_1164.all ;
3  use work.my_pkg.all ;
4
5  entity assign4 is
6  generic ( n : integer := 32);
7  port (
8      a, b: in std_logic_vector((n-1) downto 0);
9      cin : in std_logic ;
10     s : out std_logic_vector((n-1) downto 0) ;
11     cout : out std_logic
12 );
13 end assign4;
14
15 architecture dflow of assign4 is
16     --alias std_logic_vector as slv ;
17     signal cx : std_logic_vector(32 downto 1) ;
18     signal g1, p1 : std_logic_vector(31 downto 0) ;
19     signal g2, p2 : std_logic_vector(15 downto 0) ;
20     signal g3, p3 : std_logic_vector(7 downto 0) ;
21     signal g4, p4 : std_logic_vector(3 downto 0) ;
22     signal g5, p5 : std_logic_vector(1 downto 0) ;
23     signal g6, p6 : std_logic_vector(0 downto 0) ;
24     --signal g7, p7 : std_logic_vector((n/64-1) downto 0) ;
25     signal g1c, g2c, g3c, g4c, g5c, g6c, g7c : std_logic ;
26 begin
27
28 generate1: for i in 0 to 31 generate
29     generate11: if(i=0) generate
30         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.

```

28 generate1: for i in 0 to 31 generate
29   generate11: if(i=0) generate
30     G : element_orand port map (a => a(0), b => b(0), c => g1c);
31     C: element_orand port map (a => g1c, b => p1(0), c => cin, d => cx(1));
32     p: element_xor port map(a => a(0), b=> b(0), c=> p1(i));
33     g1(0) <= cx(1);
34   end generate;
35   generate111: if(i>0) generate
36     g: element_and port map(a => a(i), b=> b(i), c=> g1(i));
37     p: element_xor port map(a => a(i), b=> b(i), c=> p1(i));
38   end generate;
39 end generate;
40
41 generate2: for i in 0 to 15 generate
42   generate22: if(i=0) generate
43     G : element_orand port map (a => g1(1), b => p1(1), c => g1(0), d => g2c);
44     C: element_orand port map (a => g2c, b => p2(0), c => cin, d => cx(2));
45     g2(0) <= cx(2);
46     p: element_and port map(a => p1(i*2+1), b=> p1(i*2), c=> p2(i));
47   end generate;
48   generate222: if(i>0) generate
49     g: element_orand port map(a => g1(i*2 + 1), b=> p1(i*2 + 1), c=> g1(i*2), d => g2c);
50     p: element_and port map(a => p1(i*2+1), b=> p1(i*2), c=> p2(i));
51   end generate;
52 end generate;
53
54
55 generate3: for i in 0 to 7 generate
56   generate33: if(i=0) generate
57     G : element_orand port map (a => g2(1), b => p2(1), c => g2(0), d => g3c);

```

```

57     G : element_orand port map (a => g2(1), b => p2(1), c => g2(0), d => g3c);
58     C: element_orand port map (a => g3c, b => p3(0), c => cin, d => cx(4));
59     g3(0) <= cx(4);
60     p: element_and port map(a => p2(i*2+1), b=> p2(i*2), c=> p3(i));
61   end generate;
62   generate333: if(i>0) generate
63     g: element_orand port map(a => g2(i*2 + 1), b=> p2(i*2 + 1), c=> g2(i*2), d => g3c);
64     p: element_and port map(a => p2(i*2+1), b=> p2(i*2), c=> p3(i));
65   end generate;
66 end generate;
67
68
69 generate4: for i in 0 to 3 generate
70   generate44: if(i=0) generate
71     G : element_orand port map (a => g3(1), b => p3(1), c => g3(0), d => g4c);
72     C: element_orand port map (a => g4c, b => p4(0), c => cin, d => cx(8));
73     g4(0) <= cx(8);
74     p: element_and port map(a => p3(i*2+1), b=> p3(i*2), c=> p4(i));
75   end generate;
76   generate444: if(i>0) generate
77     g: element_orand port map(a => g3(i*2 + 1), b=> p3(i*2 + 1), c=> g3(i*2), d => g4c);
78     p: element_and port map(a => p3(i*2+1), b=> p3(i*2), c=> p4(i));
79   end generate;
80 end generate;
81
82
83 generate55: for i in 0 to 1 generate
84   generate5: if(i=0) generate
85     G : element_orand port map (a => g4(1), b => p4(1), c => g4(0), d => g5c);
86     C: element_orand port map (a => g5c, b => p5(0), c => cin, d => cx(16));

```



```

85     G : element_orand port map (a => g4(1), b => p4(1), c => g4(0), d => g5c);
86     C: element_orand port map (a => g5c, b => p5(0), c => cin, d => cx(16));
87     g5(0) <= cx(16);
88     p: element_and port map(a => p4(i*2+1), b=> p4(i*2), c=> p5(i));
89 end generate;
90 generate555: if(i>0) generate
91     g: element_orand port map(a => g4(i*2 + 1), b=> p4(i*2 + 1), c=> g4(i*2),
92     p: element_and port map(a => p4(i*2+1), b=> p4(i*2), c=> p5(i));
93 end generate;
94
95 end generate;
96
97 generate66: for i in 0 to 0 generate
98     generate666: if(i=0) generate
99         G : element_orand port map (a => g5(1), b => p5(1), c => g5(0), d => g6c);
100        C: element_orand port map (a => g6c, b => p6(0), c => cin, d => cx(32));
101        g6(0) <= cx(32);
102        p: element_and port map(a => p5(i*2+1), b=> p5(i*2), c=> p6(i));
103    end generate;
104    generate6666: if(i>0) generate
105        g: element_orand port map(a => g5(i*2 + 1), b=> p5(i*2 + 1), c=> g5(i*2),
106        p: element_and port map(a => p5(i*2+1), b=> p5(i*2), c=> p6(i));
107    end generate;
108 end generate;
109
110
111 cout <= cx(n);
112 --cx1: element_orand port map (g1(0), p1(0), cin, cx(1));
113
114

```

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.

```

120 cx3 : element_orand port map (g1(2), p1(2), cx(2), cx(3));
121 cx5 : element_orand port map (g1(4), p1(4), cx(4), cx(5));
122 cx7 : element_orand port map (g1(6), p1(6), cx(6), cx(7));
123 cx9 : element_orand port map (g1(8), p1(8), cx(8), cx(9));
124 cx11 : element_orand port map (g1(10), p1(10), cx(9), cx(11));
125 cx13 : element_orand port map (g1(12), p1(12), cx(12), cx(13));
126 cx15 : element_orand port map (g1(14), p1(14), cx(14), cx(15));
127 cx17 : element_orand port map (g1(16), p1(16), cx(16), cx(17));
128 cx19 : element_orand port map (g1(18), p1(18), cx(18), cx(19));
129 cx21 : element_orand port map (g1(20), p1(20), cx(20), cx(21));
130 cx23 : element_orand port map (g1(22), p1(22), cx(22), cx(23));
131 cx25 : element_orand port map (g1(24), p1(24), cx(24), cx(25));
132 cx27 : element_orand port map (g1(26), p1(26), cx(26), cx(27));
133 cx29 : element_orand port map (g1(28), p1(28), cx(28), cx(29));
134 cx31 : element_orand port map (g1(30), p1(30), cx(30), cx(31));
135
136 cx6 : element_orand port map (g2(2), p2(2), cx(4), cx(6));
137 cx10 : element_orand port map (g2(4), p2(4), cx(8), cx(10));
138 cx14 : element_orand port map (g2(6), p2(6), cx(12), cx(14));
139 cx18 : element_orand port map (g2(8), p2(8), cx(16), cx(18));
140 cx22 : element_orand port map (g2(10), p2(10), cx(20), cx(22));
141 cx26 : element_orand port map (g2(12), p2(12), cx(24), cx(26));
142 cx30 : element_orand port map (g2(14), p2(14), cx(28), cx(30));
143
144
145 cx12 : element_orand port map (g3(2), p3(2), cx(8), cx(12));
146 cx20 : element_orand port map (g3(4), p3(4), cx(16), cx(20));
147 cx24 : element_orand port map (g3(5), p3(5), cx(20), cx(24));
148 cx28 : element_orand port map (g3(6), p3(6), cx(24), cx(28));
149

```

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

```

131 cx25 : element_orand port map (g1(24), p1(24), cx(24), cx(25)) ;
132 cx27 : element_orand port map (g1(26), p1(26), cx(26), cx(27)) ;
133 cx29 : element_orand port map (g1(28), p1(28), cx(28), cx(29)) ;
134 cx31 : element_orand port map (g1(30), p1(30), cx(30), cx(31)) ;
135
136 cx6 : element_orand port map (g2(2), p2(2), cx(4), cx(6)) ;
137 cx10 : element_orand port map (g2(4), p2(4), cx(8), cx(10)) ;
138 cx14 : element_orand port map (g2(6), p2(6), cx(12), cx(14)) ;
139 cx18 : element_orand port map (g2(8), p2(8), cx(16), cx(18)) ;
140 cx22 : element_orand port map (g2(10), p2(10), cx(20), cx(22)) ;
141 cx26 : element_orand port map (g2(12), p2(12), cx(24), cx(26)) ;
142 cx30 : element_orand port map (g2(14), p2(14), cx(28), cx(30)) ;
143
144
145 cx12 : element_orand port map (g3(2), p3(2), cx(8), cx(12)) ;
146 cx20 : element_orand port map (g3(4), p3(4), cx(16), cx(20)) ;
147 cx24 : element_orand port map (g3(5), p3(5), cx(20), cx(24)) ;
148 cx28 : element_orand port map (g3(6), p3(6), cx(24), cx(28)) ;
149
150 sum_generate : for i in 0 to 31 generate
151   sogenerate: if(i=0) generate
152     s0 : element_xor port map (a => p1(i), b => cin, c => s(i));
153   end generate;
154   slgenerate: if(i>0) generate
155     s1 : element_xor port map (a => p1(i), b => cx(i), c => s(i));
156   end generate;
157 end generate;
158
159
160 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.

```

1  library ieee ;
2  use ieee.std_logic_1164.all ;
3  use work.my_pkg.all ;
4  use ieee.numeric_std.all ;
5  use std.textio.all ;
6
7  entity tb is
8
9  end entity;
10
11  architecture behave of tb is
12
13  signal a , b : std_logic_vector(31 downto 0) ;
14  signal cin : std_logic ;
15  signal s : std_logic_vector(31 downto 0);
16  signal cout : std_logic;
17  constant n : integer := 20000 ;
18  file vectors : text open READ_MODE is "vectors.txt";
19  constant wait_time : time := 2000 ps ;
20  begin
21
22  adder: assign4 generic map(32) port map(a => a, b =>b, cin => cin, s => s, cout => c
23
24  process
25  variable vec_a, vec_b, vec_sum : integer;
26  variable vec_cin, vec_cout : integer;
27  variable buff : line ;
28  variable s_cout: integer ;
29  begin
30  while not ENDFILE(vectors) loop

```

```

30 while not ENDFILE(vectors) loop
31   readline(vectors, buff) ;
32   if(buff(1) = '#') then
33     next ;
34   end if ;
35   read (buff, vec_a) ;
36   read (buff, vec_b) ;
37   read (buff, vec_cin) ;
38   read (buff, vec_sum) ;
39   read (buff, vec_cout) ;
40
41   a <= std_logic_vector(to_unsigned(vec_a ,32));
42   b <= std_logic_vector(to_unsigned(vec_b ,32));
43
44
45   if(vec_cin = 1) then
46     cin <= '1';
47   else
48     cin <= '0' ;
49   end if ;
50   wait for wait_time ;
51
52   --wait for wait_time ;
53
54   if(cout = '1') then
55     s_cout := 1;
56   else
57     s_cout := 0 ;
58   end if ;
59

```

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

```

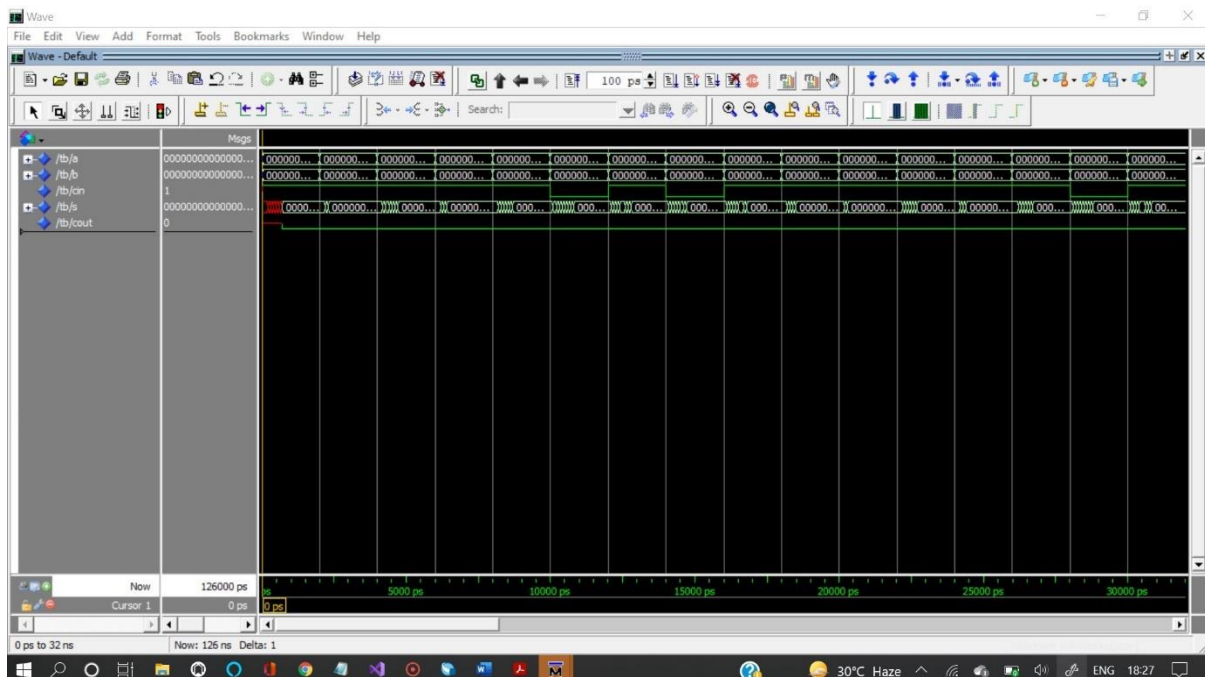
46   cin <= '1';
47   else
48     cin <= '0' ;
49   end if ;
50   wait for wait_time ;
51
52   --wait for wait_time ;
53
54   if(cout = '1') then
55     s_cout := 1;
56   else
57     s_cout := 0 ;
58   end if ;
59
60   assert((vec_sum = to_integer(unsigned(s))) and (s_cout = vec_cout))
61   report "Incorrect Result "
62   severity error ;
63
64
65   assert((vec_sum /= to_integer(unsigned(s))) and (s_cout = vec_cout))--and (std_logic
66   report "Correct Result "
67   severity NOTE ;
68
69   end loop ;
70
71   wait ;
72
73   end process;
74
75   end architecture ;

```


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

```
1 #a b cin s cout
2 4 6 1 11 0
3 45 65 1 111 0
4 455 655 1 1111 0
5 111 232 1 344 0
6 32433 43233 1 75667 0
7 6474 86585 0 93059 0
8 48574 8765875 1 8814450 0
9 387586 475764 0 863350 0
10 90898 08965 1 99864 0
11 4 6 1 11 0
12 45 65 1 111 0
13 455 655 1 1111 0
14 111 232 1 344 0
15 32433 43233 1 75667 0
16 6474 86585 0 93059 0
17 48574 8765875 1 8814450 0
18 387586 475764 0 863350 0
19 90898 08965 1 99864 0
20 4 6 1 11 0
21 45 65 1 111 0
22 455 655 1 1111 0
23 111 232 1 344 0
24 32433 43233 1 75667 0
25 6474 86585 0 93059 0
26 48574 8765875 1 8814450 0
27 387586 475764 0 863350 0
28 90898 08965 1 99864 0
29 4 6 1 11 0
30 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.



Transcript

File Edit View Bookmarks Window Help

Transcript

```
## 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>

Windows taskbar: 30°C Haze, ENG 18:28