**VLSI DESIGN EE671**

# Assignment #4 (Due Date - 22 October 2021)

*Professor: Dinesh K Sharma* *Submitted By: Pradumn Kumar(213070055)*

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.

**Problem 1**

1. 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.
2. 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.
3. 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.

**Solution** Brent-Kung adder is used for high performance addition operation. The Brent-Kung is the parallel prefix adder used to perform the addition operation. It is looking like tree structure to perform the arithmetic operation. The Brent-Kung adder consists of black cells and gray cells. Each black cell consists of two AND gates and one OR gate. Each gray cell consists of only one AND gate Pi denotes propagate and it consists of only one AND gate. Gi denotes generate and it consists of one AND gate and OR gate.

$$Pi = AixorBi$$

$$Gi = AiandBi$$

The Brent Kung adder computes the prefixes for 2 bit groups. These prefixes are used to find the prefixes for the 4 bit groups, which in turn are used to compute the prefixes for 8 bit groups and so on. These prefixes are then used to compute the carry out of the particular bit stage. These carries will be used along with the Group Propagate of the next stage to compute the Sum bit of that stage. Brent Kung Tree will be using 2log2N - 1 stages. Since we are designing a 32-bit adder the number of stages will be 9. The fanout for each bit stage is limited to 2. The diagram below shows the fanout being minimized and the loading on the further stages being reduced. But while actually implemented the buffers are generally omitted.

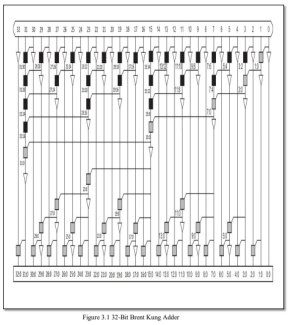

Figure 3.1 32-Bit Brent Kung Adder

The snippet of the code is given below-

```
--------  Ans(1)(a) ----------
--AND GATE
library ieee;
use ieee.std_logic_1164.all;
entity and2x1 is port(a, b : in std_logic;
     f : out std_logic);
end entity;
architecture df of and2x1 is begin
f <= a and b after 100ns;
end architecture;

--XOR GATE
library ieee;
use ieee.std_logic_1164.all;
entity xor2x1 is port(a, b : in std_logic;
     f : out std_logic);
end entity;
architecture df of xor2x1 is begin
f <= a xor b after 100ns;
end architecture;

--A + B.C GATE
library ieee;
use ieee.std_logic_1164.all;
entity func3x1 is port(a, b, c : in std_logic;
     f : out std_logic);
end entity;
architecture df of func3x1 is begin
f <= a or (b and c) after 100ns;
end architecture;

--Final Sum and Carry Box
library ieee;
use ieee.std_logic_1164.all;
entity sum_carry_box is port(gi, pi, ci : in std_logic;
     ci1, si1 : out std_logic);
end entity;
architecture df of sum_carry_box is
component func3x1 is port(a, b, c : in std_logic;
     f : out std_logic);
end component;
component xor2x1 is port(a, b : in std_logic;
     f : out std_logic);
end component;
begin
carry : func3x1 port map (gi, pi, ci, ci1);
```

```
47  sum : xor2x1 port map (pi, ci, si1);
48  end architecture;
49
50  --Gi and Pi generation
51  library ieee;
52  use ieee.std_logic_1164.all;
53  entity gen_pen_box is port(gi, pi, gi_1, pi_1 : in std_logic;
54       gi1, pi1 : out std_logic);
55  end entity;
56  architecture arc of gen_pen_box is
57  component func3x1 is port(a, b, c : in std_logic;
58       f : out std_logic);
59  end component;
60  component and2x1 is port(a, b : in std_logic;
61       f : out std_logic);
62  end component;
63  begin
64  label1 : func3x1 port map(gi, pi, gi_1, gi1);
65  label2 : and2x1 port map(pi, pi_1, pi1);
66  end architecture;
67
68  --------  Ans(1)(b) ---------
69  library ieee;
70  use ieee.std_logic_1164.all;
71  entity krung_adder_32bit is port (a, b : in std_logic_vector(31 downto 0);
72        cin : in std_logic;
73        cout : out std_logic;
74        s : out std_logic_vector(32 downto 1));
75  end entity;
76  architecture structural of krung_adder_32bit is
77  component sum_carry_box is port(gi, pi, ci : in std_logic;
78       ci1, si1 : out std_logic);
79  end component;
80  component gen_pen_box is port(gi, pi, gi_1, pi_1 : in std_logic;
81       gi1, pi1 : out std_logic);
82  end component;
83  component and2x1 is port(a, b : in std_logic;
84       f : out std_logic);
85  end component;
86  component xor2x1 is port(a, b : in std_logic;
87       f : out std_logic);
88  end component;
89  component func3x1 is port(a, b, c : in std_logic;
90       f : out std_logic);
91  end component;
92  signal g1, p1 : std_logic_vector(31 downto 0);
```

```
 93    signal g2, p2 : std_logic_vector(15 downto 0);
 94    signal g3, p3 : std_logic_vector(7 downto 0);
 95    signal g4, p4 : std_logic_vector(3 downto 0);
 96    signal g5, p5 : std_logic_vector(1 downto 0);
 97    signal g6, p6 : std_logic;
 98    signal c : std_logic_vector(32 downto 1);
 99    begin
100    stage1 : for i in 0 to 31 generate
101    label1 : and2x1 port map(a(i), b(i), g1(i));
102    label2 : xor2x1 port map(a(i), b(i), p1(i));
103    end generate;
104
105    stage2 : for i in 1 to 16 generate
106    label3 : gen_pen_box port map(g1(i), p1(i), g1(i-1), p1(i-1), g2(i-1), p2(i-1));
107    end generate;
108
109    stage3 : for i in 1 to 8 generate
110    label4 : gen_pen_box port map(g2(i), p2(i), g2(i-1), p2(i-1), g3(i-1), p3(i-1));
111    end generate;
112
113    stage4 : for i in 1 to 4 generate
114    label5 : gen_pen_box port map(g3(i), p3(i), g3(i-1), p3(i-1), g4(i-1), p4(i-1));
115    end generate;
116
117    stage5 : for i in 1 to 2 generate
118    label6 : gen_pen_box port map(g4(i), p4(i), g4(i-1), p4(i-1), g5(i-1), p5(i-1));
119    end generate;
120
121    label7 : gen_pen_box port map(g5(1), p5(1), g5(0), p5(0), g6, p6);
122
123    label8 : sum_carry_box port map(g1(0), p1(0), cin, c(1), s(1));
124
125    sum_car_generation : for i in 1 to 31 generate
126    label9 : sum_carry_box port map(g1(i), p1(i), c(i), c(i+1), s(i+1));
127    end generate;
128
129    --label10 : func3x1 port map(g6, p6, cin, cout);
130    cout <= c(32);
131
132    end architecture;
```

```
--------  Ans(1)(c) ---------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;
use std.textio.all;

ENTITY test_adder IS
END test_adder;

ARCHITECTURE tb OF test_adder IS
component krung_adder_32bit is port (a, b : in std_logic_vector(31 downto 0);
      cin : in std_logic;
      cout : out std_logic;
      s : out std_logic_vector(32 downto 1));
end component;
signal tb_a, tb_b : std_logic_vector(31 downto 0);
signal tb_cin : std_logic;
signal tb_cout : std_logic;
signal tb_s : std_logic_vector(32 downto 1);
begin

dut : krung_adder_32bit port map(tb_a, tb_b, tb_cin, tb_cout, tb_s);
process
file text_file : text open read_mode is "D:\Documents\Quartus Projects\Assign4_EE67
variable text_line : line;
variable x, y, z : std_logic_vector(31 downto 0);
variable p, q : std_logic;
variable ok : boolean;
begin

while not endfile(text_file) loop
readline(text_file, text_line);
if text_line.all'length = 0 or text_line.all(1) = '#' then next;
end if;
read(text_line, x, ok);
tb_a <= x;
read(text_line, y, ok);
tb_b <= y;
read(text_line, p, ok);
tb_cin <= p;
read(text_line, z, ok);
read(text_line, q, ok);
wait for 2000ns;
assert(tb_s = z) report "Mismatch" severity failure;
assert(tb_cout = q) report "Mismatch" severity failure;
end loop;
```

The Observation of this exercises is as follows-