EE671 VSLI Design
Assignment - 3 Brunt Kung Adder
Madan Y N 200070040

Aim:
The aim of the assignment is to describe a 32-bit Brent Kung adder in VHDL and simulate it using a test bench.

Design:
We first design the basic gates required for building the adder circuit

```
library IEEE;
use IEEE.std_logic_1164.all;

entity andgate is
    port (A, B: in std_ulogic; prod: out std_ulogic);
end entity andgate;

architecture trivial of andgate is
    begin
        prod <= A AND B AFTER 340 ps;
end architecture trivial;

library IEEE;
use IEEE.std_logic_1164.all;

entity xorgate is
    port (A, B: in std_ulogic; uneq: out std_ulogic);
end entity xorgate;

architecture trivial of xorgate is
    begin
        uneq <= A XOR B AFTER 680 ps;
end architecture trivial;

library IEEE;
use IEEE.std_logic_1164.all;

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

architecture trivial of abcgate is
```

```
    begin
        abc <= A OR (B AND C) AFTER 440 ps;
end architecture trivial;

-- 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_ulogic; Bit0_G: out std_ulogic);
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 680 ps;
end architecture trivial;

library ieee;
use ieee.std_logic_1164.all;
library work;

entity G_and_P is
    port(
        G_u, P_u, G_l, P_l : in std_logic;
        Gout, Pout : out std_logic
    );
end entity G_and_P;

architecture arc of G_and_P is
    begin
        abc0 : entity work.abcgate port map (G_u, P_u, G_l, Gout);
        and0 : entity work.andgate port map (P_u, P_l, Pout);
end arc;
```

We make use of the above gates to build the adder circuit as shown below

```
library IEEE;
use IEEE.std_logic_1164.all;
library work;

entity brunt_kung_adder is
    port(
        A, B : in std_logic_vector(31 downto 0);
        Cin : in std_logic;
        Sum : out std_logic_vector(31 downto 0);
        Cout : out std_logic
    );
end entity brunt_kung_adder;

architecture arc of brunt_kung_adder is
    signal carry : std_logic_vector(31 downto 0);
    signal G0 : std_logic_vector(31 downto 0);
    signal P0 : std_logic_vector(31 downto 0);
    signal G1 : std_logic_vector(15 downto 0);
    signal P1 : std_logic_vector(15 downto 0);
    signal G2 : std_logic_vector(7 downto 0);
    signal P2 : std_logic_vector(7 downto 0);
    signal G3 : std_logic_vector(3 downto 0);
    signal P3 : std_logic_vector(3 downto 0);
    signal G4 : std_logic_vector(1 downto 0);
    signal P4 : std_logic_vector(1 downto 0);
    signal G5 : std_logic;
    signal P5 : std_logic;
begin

    --Computation of G and P
    g0_p0_generate: for i in 0 to 31 generate
        and0 : entity work.andgate port map (A => A(i), B => B(i),
prod => G0(i));
        xor0 : entity work.xorgate port map (A => A(i), B => B(i),
uneq => P0(i));
    end generate g0_p0_generate;

    g1_p1_generate: for i in 0 to 15 generate
```

```
        gen1 : entity work.G_and_P port map (G_u => G0(2*i+1), P_u
=> P0(2*i+1), G_l => G0(2*i), P_l => P0(2*i), Gout => G1(i), Pout
=> P1(i));
    end generate g1_p1_generate;

    g2_p2_generate: for i in 0 to 7 generate
        gen2 : entity work.G_and_P port map (G_u => G1(2*i+1), P_u
=> P1(2*i+1), G_l => G1(2*i), P_l => P1(2*i), Gout => G2(i), Pout
=> P2(i));
    end generate g2_p2_generate;

    g3_p3_generate: for i in 0 to 3 generate
        gen3 : entity work.G_and_P port map (G_u => G2(2*i+1), P_u
=> P2(2*i+1), G_l => G2(2*i), P_l => P2(2*i), Gout => G3(i), Pout
=> P3(i));
    end generate g3_p3_generate;

    g4_p4_generate: for i in 0 to 1 generate
        gen4 : entity work.G_and_P port map (G_u => G3(2*i+1), P_u
=> P3(2*i+1), G_l => G3(2*i), P_l => P3(2*i), Gout => G4(i), Pout
=> P4(i));
    end generate g4_p4_generate;

    gen5 : entity work.G_and_P port map (G_u => G4(1), P_u =>
P4(1), G_l => G4(0), P_l => P4(0), Gout => G5, Pout => P5);

    --Computation of carry
    carry1 : entity work.abcgate port map (A => G0(0), B => P0(0),
C => Cin, abc => carry(1));

    carry2 : entity work.abcgate port map (A => G1(0), B => P1(0),
C => Cin, abc => carry(2));

    carry3 : entity work.abcgate port map (A => G0(2), B => P0(2),
C => carry(2), abc => carry(3));
    carry4 : entity work.abcgate port map (A => G2(0), B => P2(0),
C => Cin, abc => carry(4));

    carry5 : entity work.abcgate port map (A => G0(4), B => P0(4),
C => carry(4), abc => carry(5));
    carry6 : entity work.abcgate port map (A => G1(2), B => P1(2),
C => carry(4), abc => carry(6));
```

```
    carry8 : entity work.abcgate port map (A => G3(0), B => P3(0),
C => Cin, abc => carry(8));

    carry7 : entity work.abcgate port map (A => G0(6), B => P0(6),
C => carry(6), abc => carry(7));
    carry9 : entity work.abcgate port map (A => G0(8), B => P0(8),
C => carry(8), abc => carry(9));
    carry10 : entity work.abcgate port map (A => G1(4), B =>
P1(4), C => carry(8), abc => carry(10));
    carry12 : entity work.abcgate port map (A => G2(2), B =>
P2(2), C => carry(8), abc => carry(12));
    carry16 : entity work.abcgate port map (A => G4(0), B =>
P4(0), C => Cin, abc => carry(16));

    carry11 : entity work.abcgate port map (A => G0(10), B =>
P0(10), C => carry(10), abc => carry(11));
    carry13 : entity work.abcgate port map (A => G0(12), B =>
P0(12), C => carry(12), abc => carry(13));
    carry14 : entity work.abcgate port map (A => G1(6), B =>
P1(6), C => carry(12), abc => carry(14));
    carry17 : entity work.abcgate port map (A => G0(16), B =>
P0(16), C => carry(16), abc => carry(17));
    carry18 : entity work.abcgate port map (A => G1(8), B =>
P1(8), C => carry(16), abc => carry(18));
    carry20 : entity work.abcgate port map (A => G2(4), B =>
P2(4), C => carry(16), abc => carry(20));
    carry24 : entity work.abcgate port map (A => G3(2), B =>
P3(2), C => carry(16), abc => carry(24));
    carry32 : entity work.abcgate port map (A => G5, B => P5, C =>
Cin, abc => Cout);

    carry15 : entity work.abcgate port map (A => G1(14), B =>
P1(14), C => carry(14), abc => carry(15));
    carry19 : entity work.abcgate port map (A => G0(18), B =>
P0(18), C => carry(18), abc => carry(19));
    carry21 : entity work.abcgate port map (A => G0(20), B =>
P0(20), C => carry(20), abc => carry(21));
    carry22 : entity work.abcgate port map (A => G1(10), B =>
P1(10), C => carry(20), abc => carry(22));
    carry26 : entity work.abcgate port map (A => G1(12), B =>
P1(12), C => carry(24), abc => carry(26));
    carry25 : entity work.abcgate port map (A => G0(24), B =>
```

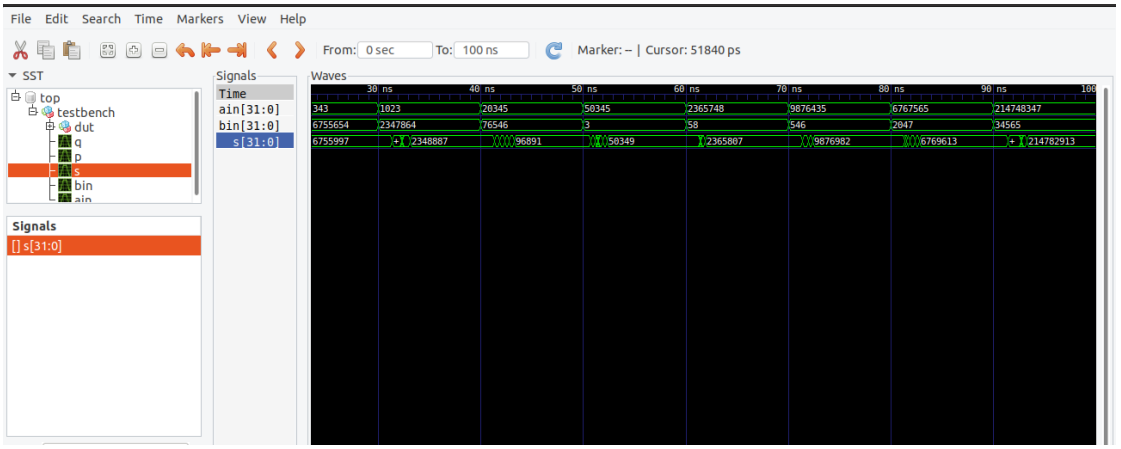
```
P0(24), C => carry(24), abc => carry(25));
    carry28 : entity work.abcgate port map (A => G2(6), B =>
P2(6), C => carry(24), abc => carry(28));

    carry23 : entity work.abcgate port map (A => G0(22), B =>
P0(22), C => carry(22), abc => carry(23));
    carry27 : entity work.abcgate port map (A => G0(26), B =>
P0(26), C => carry(26), abc => carry(27));
    carry29 : entity work.abcgate port map (A => G0(28), B =>
P0(28), C => carry(28), abc => carry(29));
    carry30 : entity work.abcgate port map (A => G1(14), B =>
P1(14), C => carry(28), abc => carry(30));

    carry31 : entity work.abcgate port map (A => G0(30), B =>
P0(30), C => carry(30), abc => carry(31));


    --Computing sum
    sum0 : entity work.xorgate port map (A => P0(0), B => Cin,
uneq => Sum(0));
    sum_generate : for i in 1 to 31 generate
        sum1 : entity work.xorgate port map (A => P0(i), B =>
carry(i), uneq => Sum(i));
    end generate sum_generate;
end arc;
```

Results:
In order to test the circuit, a testbench is created which will take in two integers and a std_logic as input and feed into the device-under-test(the brunt_kung_adder entity) and verify that the result is matching with the expected result.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;

entity testbench is
end;

architecture arc of testbench is
    signal Ain, Bin, S : std_logic_vector(31 downto 0);
```

```
    signal Cin : std_logic;
    signal Cout : std_logic;
    type test_inputs is array(0 to 9) of integer;
    signal p : test_inputs := (0, 64, 343, 1023, 20345, 50345,
2365748, 9876435, 6767565, 214748347);
    signal q : test_inputs := (3, 58, 546, 2047, 34565, 76546,
2347864, 6755654, 8776984, 214618332);

    begin
        dut : entity work.brunt_kung_adder port map (A => Ain, B
=> Bin, Cin => Cin, Sum => S, Cout => Cout);

        process
        begin
        --for input carry = 0
            for i in 0 to 4 loop
                Ain <= std_logic_vector(to_signed(p(i), 32));
                Bin <= std_logic_vector(to_signed(q(9-i), 32));
                Cin <= '0';
                wait for 10 ns;
                assert to_integer(signed(S)) = p(i) + q(9-i)
report "Failed";
            end loop;
         --for input carry = 1
            for i in 0 to 4 loop
                Ain <= std_logic_vector(to_signed(p(5+i), 32));
                Bin <= std_logic_vector(to_signed(q(i), 32));
                Cin <= '1';
                wait for 10 ns;
                assert to_integer(signed(S)) = p(5+i) + q(i) + 1
report "Failed";
            end loop;
         wait;
        end process;
end arc;
```

Simulation results:



Conclusions:

The 32-bit Brunt Kung adder is implemented in VHDL and successfully verified using a testbench