EE671: Assignment 4
Brent Kung Adder

Vineet Ghule, 20D070089

October 21, 2022

1 Aim

To describe a 16 bit Brent Kung adder in VHDL and simulate it using a test bench

2 Design

Table 1: Delays of the required gates

Function	**Delay(ps)**
AND	49
A + B.C	68
XOR	78
A.B + C.(A+B)	78

Using the following relations:

$$C_{(1+k)2^i} = G^i_{(1+k)2^i-1:k2^i} + P^i_{(1+k)2^i-1:k2^i}.C_{k2^i} \quad ; \quad k = 0, 1, 2, ... \quad i = 0, 1, 2, ...$$

Since C_0 is available initially, we can set $P^i_{2^i-1:0} = 0$ and include its effect in $G^i_{2^i-1:0}$ by initially setting $C_1 = G^0_0 = A_0.B_0 + C_0.(A_0 + B_0)$ and further iteratively, $C_{2^i} = G^i_{2^i-1:0}$. This is what the statement 'rightmost blocks of all levels should use the available value of C0 to compute the output carry directly using the logic block for A.B + C.(A+B)' in the question meant.

Table 2: Signal availability w.r.t. time (in units of maximum delay of a gate = 78 ps)

t	**Signals generated**
0	A_i, B_i, C_0
1	G^0, P^0, C_1
2	G^1, P^1, C_2
3	G^2, P^2, C_4, C_3
4	G^3, P^3, C_8, C_6, C_5
5	G^4, P^4, C_{16}, C_{12}, C_{10}, C_9, C_7
6	C_{14}, C_{13}, C_{11}
7	C_{15}
8	S_i

3 Code

- **Gates.vhd**

```
-- simple gates with trivial architectures

library ieee;
use ieee.std_logic_1164.all;
entity andgate is
        port(
                A, B: in std_logic;
                prod: out std_logic
        );
end entity andgate;
architecture trivial of andgate is
begin
        prod <= A and B after 49 ps;
end architecture trivial;

library ieee;
use ieee.std_logic_1164.all;
entity xorgate is
        port(
                A, B: in std_logic;
                uneq: out std_logic
        );
end entity xorgate;
architecture trivial of xorgate is
begin
        uneq <= A xor B after 78 ps;
end architecture trivial;

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 68 ps;
end architecture trivial;

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 78 ps;
end architecture trivial;
```

- **BrentKungAdder16.vhd**

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

entity Brent_Kung_Adder16 is
        port(
                Cin: in std_logic;
                A, B: in std_logic_vector(15 downto 0);
                S: out std_logic_vector(15 downto 0);
                Cout: out std_logic
        );
end entity Brent_Kung_Adder16;
architecture struct of Brent_Kung_Adder16 is

   component andgate is
                port(
                        A, B: in std_logic;
                        prod: out std_logic
                );
   end component;
   component xorgate is
                port(
                        A, B: in std_logic;
                        uneq: out std_logic
                );
   end component;
   component abcgate is
                port(
                        A, B, C: in std_logic;
                        abc: out std_logic
                );
   end component;
```

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

        signal G0, P0, C: std_logic_vector(15 downto 1);
        signal G1, P1: std_logic_vector(7 downto 1);
        signal G2, P2: std_logic_vector(3 downto 1);
        signal G3, P3, P0_0: std_logic;
begin
        --t = 0
        C1_gate: Cin_map_G port map(A => A(0), B => B(0), Cin => Cin, Bit0_G => C(1));
        P0_0_xorgate: xorgate port map(A => A(0), B => B(0), uneq => P0_0);
        Level0: for i in 1 to 15 generate
                G0_andgate: andgate port map(A => A(i), B => B(i), prod => G0(i));
                P0_xorgate: xorgate port map(A => A(i), B => B(i), uneq => P0(i));
        end generate Level0;

        --t = 1
        C2_gate: abcgate port map(A => G0(1), B => P0(1), C => C(1), abc => C(2));
        Level1: for i in 1 to 7 generate
                G1_abcgate: abcgate port map(A => G0(2*i+1), B => P0(2*i+1), C => G0(2*i), abc => G1(i));
                P1_andgate: andgate port map(A => P0(2*i+1), B => P0(2*i), prod => P1(i));
        end generate Level1;

        --t = 2
        C4_gate: abcgate port map(A => G1(1), B => P1(1), C => C(2), abc => C(4));
        C3_gate: abcgate port map(A => G0(2), B => P0(2), C => C(2), abc => C(3));
        Level2: for i in 1 to 3 generate
                G2_abcgate: abcgate port map(A => G1(2*i+1), B => P1(2*i+1), C => G1(2*i), abc => G2(i));
                P2_andgate: andgate port map(A => P1(2*i+1), B => P1(2*i), prod => P2(i));
        end generate Level2;

        --t = 3
        C8_gate: abcgate port map(A => G2(1), B => P2(1), C => C(4), abc => C(8));
        C6_gate: abcgate port map(A => G1(2), B => P1(2), C => C(4), abc => C(6));
        C5_gate: abcgate port map(A => G0(4), B => P0(4), C => C(4), abc => C(5));
        G3_abcgate: abcgate port map(A => G2(3), B => P2(3), C => G2(2), abc => G3);
        P3_andgate: andgate port map(A => P2(3), B => P2(2), prod => P3);

        --t = 4
        C16_gate: abcgate port map(A => G3, B => P3, C => C(8), abc => Cout);
        C12_gate: abcgate port map(A => G2(2), B => P2(2), C => C(8), abc => C(12));
        C10_gate: abcgate port map(A => G1(4), B => P1(4), C => C(8), abc => C(10));
        C9_gate: abcgate port map(A => G0(8), B => P0(8), C => C(8), abc => C(9));
        C7_gate: abcgate port map(A => G0(6), B => P0(6), C => C(6), abc => C(7));

        --t = 5
        C14_gate: abcgate port map(A => G1(6), B => P1(6), C => C(12), abc => C(14));
        C13_gate: abcgate port map(A => G0(12), B => P0(12), C => C(12), abc => C(13));
        C11_gate: abcgate port map(A => G0(10), B => P0(10), C => C(10), abc => C(11));

        --t = 6
        C15_gate: abcgate port map(A => G0(14), B => P0(14), C => C(14), abc => C(15));
```

```
        --t = 7
        S0_xorgate: xorgate port map(A => P0_0, B => Cin, uneq => S(0));
        Sums: for i in 1 to 15 generate
                S_xorgate: xorgate port map(A => P0(i), B => C(i), uneq => S(i));
        end generate Sums;

        --t = 8
end struct;
```

- **DUT.vhdl**

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

entity DUT is
   port(
                input_vector: in std_logic_vector(32 downto 0);
      output_vector: out std_logic_vector(16 downto 0)
        );
end entity;
architecture DutWrap of DUT is
   component Brent_Kung_Adder16 is
                port(
                        Cin: in std_logic;
                        A, B: in std_logic_vector(15 downto 0);
                        S: out std_logic_vector(15 downto 0);
                        Cout: out std_logic
                );
   end component;
begin
   add_instance: Brent_Kung_Adder16 port map(
                A => input_vector(32 downto 17),
                B => input_vector(16 downto 1),
                Cin => input_vector(0),
                Cout => output_vector(16),
                S => output_vector(15 downto 0)
        );
end DutWrap;
```

- **Testbench.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

  ----------------------------------------------------------------
  --  edit the following lines to set the number of i/o's of your
  --  DUT.
  ----------------------------------------------------------------
  constant number_of_inputs  : integer := 33;  -- # input bits to your design.
```

```
  constant number_of_outputs : integer := 17;  -- # output bits from your design.
  ------------------------------------------------------------------
  ------------------------------------------------------------------

  -- Note that you will have to wrap your design into the DUT
  -- as indicated in class.
  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);

  -- create a constrained string
  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;

  -- bit-vector to std-logic-vector and vice-versa
  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
          ret_val(I) := '0';
        end if;
     end loop;
     return ret_val;
  end to_std_logic_vector;

  function to_bit_vector(x: std_logic_vector) return bit_vector is
     alias lx: std_logic_vector(1 to x'length) is x;
     variable ret_val: bit_vector(1 to x'length);
  begin
     for I in 1 to x'length loop
        if(lx(I) = '1') then
          ret_val(I) := '1';
        else
          ret_val(I) := '0';
        end if;
     end loop;
     return ret_val;
  end to_bit_vector;

begin
  process
    variable err_flag : boolean := false;
    File INFILE: text open read_mode is "TRACEFILE.txt";
```

```
  FILE OUTFILE: text  open write_mode is "outputs.txt";

  -- bit-vectors are read from the file.
  variable input_vector_var: bit_vector (number_of_inputs-1 downto 0);
  variable output_vector_var: bit_vector (number_of_outputs-1 downto 0);
  variable output_mask_var: bit_vector (number_of_outputs-1 downto 0);

  -- for comparison of output with expected-output
  variable output_comp_var: std_logic_vector (number_of_outputs-1 downto 0);
  constant ZZZZ : std_logic_vector(number_of_outputs-1 downto 0) := (others => '0');

  -- for read/write.
  variable INPUT_LINE: Line;
  variable OUTPUT_LINE: Line;
  variable LINE_COUNT: integer := 0;


begin
  while not endfile(INFILE) loop
        -- will read a new line every 5ns, apply input,
        -- wait for 1 ns for circuit to settle.
        -- read output.


        LINE_COUNT := LINE_COUNT + 1;


        -- read input at current time.
        readLine (INFILE, INPUT_LINE);
        read (INPUT_LINE, input_vector_var);
        read (INPUT_LINE, output_vector_var);
        read (INPUT_LINE, output_mask_var);

        -- apply input.
        input_vector <= to_std_logic_vector(input_vector_var);

        -- wait for the circuit to settle
        wait for 800 ps;

        -- check output.
        output_comp_var := (to_std_logic_vector(output_mask_var) and
                                  (output_vector xor to_std_logic_vector(output_vector_var)));
        if (output_comp_var  /= ZZZZ) then
           write(OUTPUT_LINE,to_string("ERROR: line "));
           write(OUTPUT_LINE, LINE_COUNT);
           writeline(OUTFILE, OUTPUT_LINE);
           err_flag := true;
        end if;

        write(OUTPUT_LINE, to_bit_vector(input_vector));
        write(OUTPUT_LINE, to_string(" "));
        write(OUTPUT_LINE, to_bit_vector(output_vector));
        writeline(OUTFILE, OUTPUT_LINE);

        -- advance time by 200 ps.
        wait for 200 ps;
  end loop;
```

```
    assert (err_flag) report "SUCCESS, all tests passed." severity note;
    assert (not err_flag) report "FAILURE, some tests failed." severity error;

    wait;
  end process;

  dut_instance: DUT
            port map(input_vector => input_vector, output_vector => output_vector);

end Behave;
```

4 Simulation Result

Output written to '$\backslash Assignment4\backslash simulation\backslash modelsim\backslash outputs.txt$'.
Circuit outputs would be calculated in $8 \times 78\ ps = 624\ ps$.
Circuit settling time specified in *Testbench.vhdl* is 800 ps.
For RTL simulation all test cases pass.
For Gate-Level simulation circuit settling time needs to be increased for test cases to pass (around 40 ns works).