

EE671 VLSI Design - Assignment 4

Rohan Rajesh Kalbag, 20D170033

October 13, 2022

1 Approach

The delays in the architecture blocks of the VHDL hardware descriptions given for `andgate`, `abcgate`, `xorgate`, `Cin_map_G` were modified according to the problem statement and the value of the last digit of my roll number which is **3**. And stored in a file called `gates.vhdl`

1.1 Code present in `gates.vhdl`

```
-- Rohan Rajesh Kalbag --  
-- Roll Number: 20D170033, last digit is 3 --  
  
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 43 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 66 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 56 ps;
    end architecture trivial;

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 66 ps;
    end architecture trivial;

```

1.2 Designing the 16-bit Brent Kung Adder

1.2.1 Notation

The two sixteen bits numbers are given by A and B. $c_{in} = c_0$ denotes the input carry. c_i denotes the generated ith carry bit. a_i, b_i denote the ith bit of a and b. $g_{x,y}^i$ denotes the ith order generate signal for the bits from x to y. Similarly $p_{x,y}^i$ denotes the ith order propagate signal for the bits from x to y.

As told in the handout for the rightmost block in the first layer, **Cin_map_G** is used to directly generate the first carry c_1 from a_0, b_0 and c_{in} . The value of g_0^1 is also assigned to be the same as c_1 .

For the rest of the generate and propagate signals these relations are used

$$g_{i,i-1}^2 = g_i^1 + p_i^1 \cdot g_{i-1}^1$$

$$p_{i,i-1}^2 = p_i^1 \cdot p_{i-1}^1$$

$$p_{i,i-3}^3 = p_{i,i-1}^2 \cdot p_{i-2,i-3}^2$$

$$g_{i,i-3}^3 = g_{i,i-1}^2 + g_{i-2,i-3}^2 \cdot p_{i,i-1}^2 \dots \text{and so on}$$

The values of carries c_i were calculated using the relations given below

$$c_2 = g_1^1 + p_1^2 \cdot c_{in} \quad c_3 = g_2^1 + p_2^1 \cdot c_2 \quad c_4 = g_{3,0}^3 + p_{3,0}^3 \cdot c_{in}$$

$$c_5 = g_4^1 + p_4^1 \cdot c_4 \quad c_6 = g_{5,4}^2 + p_{5,4}^2 \cdot c_4 \quad c_7 = g_6^1 + p_6^1 \cdot c_6$$

$$c_8 = g_{7,0}^4 + p_{7,0}^4 \cdot c_{in} \quad c_9 = g_8^1 + p_8^1 \cdot c_8 \quad c_{10} = g_{9,8}^2 + p_{9,8}^2 \cdot c_8$$

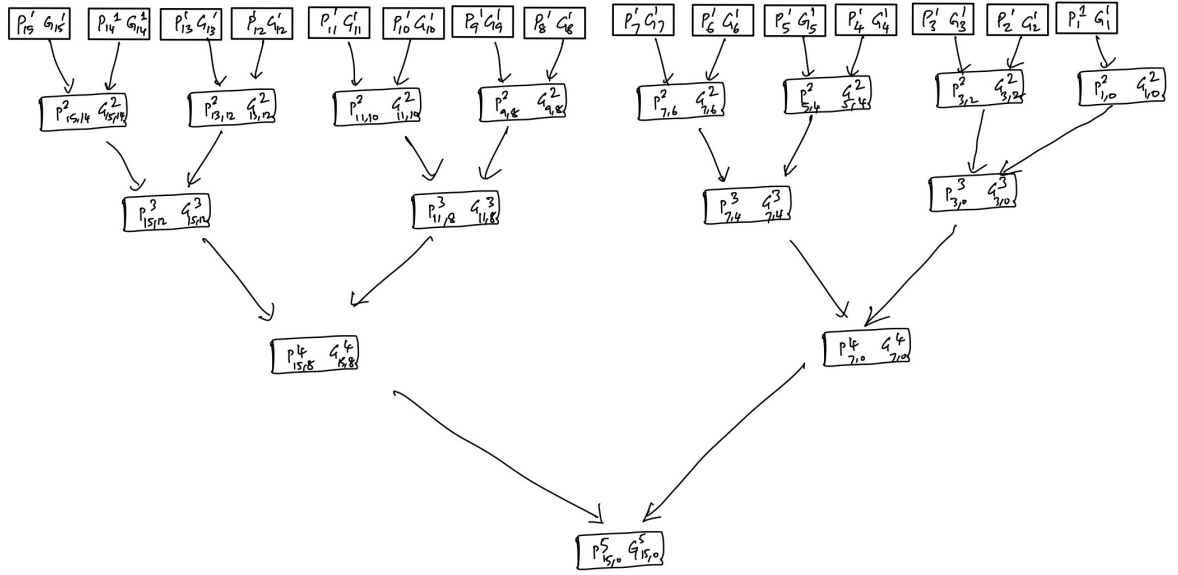
$$c_{11} = g_{10}^1 + p_{10}^1 \cdot c_{10} \quad c_{12} = g_{11,8}^3 + g_{11,8}^3 \cdot c_8 \quad c_{13} = g_{12}^1 + p_{12}^1 \cdot c_{12}$$

$$c_{11} = g_{10}^1 + p_{10}^1 \cdot c_{10} \quad c_{12} = g_{11,8}^3 + g_{11,8}^3 \cdot c_8 \quad c_{13} = g_{12}^1 + p_{12}^1 \cdot c_{12}$$

$$c_{14} = g_{13,12}^2 + p_{13,12}^2 \cdot c_{12} \quad c_{15} = g_{14}^1 + p_{14}^1 \cdot c_{14} \quad c_{16} = g_{15,0}^5 + p_{15,0}^5 \cdot c_{in}$$

1.2.2 Tree Diagram

The generate and propagate signals were created as per this tree block diagram where each node represents the generate and propagate signal and each edge denotes the generate and propagate signals needed to create the next node.



Representation of 16 bit Brent Kung Adder as Tree Diagram

Finally the sum bits were generated using the following relation

$$s_i = p_i \oplus c_i$$

1.2.3 Code for 16 bit Brent Kung Adder

The entity `brentkung` was created as follows to describe the adder and stored in `brent_kung_adder.vhdl`.

```
-- Rohan Rajesh Kalbag --
-- Roll Number: 20D170033, last digit is 3 --
-- 16 bit Brent Kung Adder --
```

```

library IEEE;
use IEEE.std_logic_1164.all;

entity brentkung is
  port(
    a,b: in std_logic_vector(15 downto 0);
    s: out std_logic_vector(15 downto 0);
    cout: out std_logic;
    cin: in std_logic
  );
end entity;

architecture behave of brentkung is
  --signal declarations --

  --leaf nodes
  signal p1_i, g1_i: std_logic_vector(15 downto 0);
  -- root node
  signal g5_15_0, p5_15_0: std_logic;
  -- right sub tree --
  signal p2_7_6, g2_7_6: std_logic;
  signal p2_5_4, g2_5_4: std_logic;
  signal p2_3_2, g2_3_2: std_logic;
  signal p2_1_0, g2_1_0: std_logic;
  signal p3_7_4, g3_7_4: std_logic;
  signal p3_3_0, g3_3_0: std_logic;
  signal p4_7_0, g4_7_0: std_logic;
  -- left sub tree --
  signal p2_15_14, g2_15_14: std_logic;
  signal p2_13_12, g2_13_12: std_logic;
  signal p2_11_10, g2_11_10: std_logic;
  signal p2_9_8, g2_9_8: std_logic;
  signal p3_15_12, g3_15_12: std_logic;
  signal p3_11_8, g3_11_8: std_logic;
  signal p4_15_8, g4_15_8: std_logic;

  --carry bits--
  signal c: std_logic_vector(16 downto 0);

```

```

-- component declarations --
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 xorgate;

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

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

begin
    --generate p1_i signals --
    xors: for i in 0 to 15 generate
        xor_i: xorgate port map(a => a(i), b => b(i), uneq =>
            ↪ p1_i(i));
    end generate xors;

    --generate g1_i signals --
    ands: for i in 1 to 15 generate
        and_i: andgate port map(a => a(i), b => b(i), prod =>
            ↪ g1_i(i));
    end generate ands;

```

```

--generate g1_0 using Cin_map_G
cmapg_1: Cin_map_G port map(a => a(0), b => b(0), cin =>
    ↪ cin, bit0_g => g1_i(0));
--set the first carry as the value g1_0
c(1) <= g1_i(0);

--generate p2, g2 signals --
abc_1: abcgate port map(a => g1_i(1), b => p1_i(1), c =>
    ↪ g1_i(0), abc => g2_1_0);
abc_2: abcgate port map(a => g1_i(3), b => p1_i(3), c =>
    ↪ g1_i(2), abc => g2_3_2);
abc_3: abcgate port map(a => g1_i(5), b => p1_i(5), c =>
    ↪ g1_i(4), abc => g2_5_4);
abc_4: abcgate port map(a => g1_i(7), b => p1_i(7), c =>
    ↪ g1_i(6), abc => g2_7_6);
abc_5: abcgate port map(a => g1_i(9), b => p1_i(9), c =>
    ↪ g1_i(8), abc => g2_9_8);
abc_6: abcgate port map(a => g1_i(11), b => p1_i(11), c =>
    ↪ g1_i(10), abc => g2_11_10);
abc_7: abcgate port map(a => g1_i(13), b => p1_i(13), c =>
    ↪ g1_i(12), abc => g2_13_12);
abc_8: abcgate port map(a => g1_i(15), b => p1_i(15), c =>
    ↪ g1_i(14), abc => g2_15_14);

and_1: andgate port map(a =>p1_i(0), b=>p1_i(1), prod=>
    ↪ p2_1_0);
and_2: andgate port map(a =>p1_i(2), b=>p1_i(3), prod=>
    ↪ p2_3_2);
and_3: andgate port map(a =>p1_i(4), b=>p1_i(5), prod=>
    ↪ p2_5_4);
and_4: andgate port map(a =>p1_i(6), b=>p1_i(7), prod=>
    ↪ p2_7_6);
and_5: andgate port map(a =>p1_i(8), b=>p1_i(9), prod=>
    ↪ p2_9_8);
and_6: andgate port map(a =>p1_i(10), b=>p1_i(11), prod=>
    ↪ p2_11_10);
and_7: andgate port map(a =>p1_i(12), b=>p1_i(13), prod=>
    ↪ p2_13_12);

```

```

and_8: andgate port map(a =>p1_i(14), b=>p1_i(15), prod=>
    ↪ p2_15_14);

--generate p3, g3 signals --
abc_9: abcgate port map(a => g2_15_14, b => p2_15_14, c =>
    ↪ g2_13_12, abc => g3_15_12);
abc_10: abcgate port map(a => g2_11_10, b => p2_11_10, c =>
    ↪ g2_9_8, abc => g3_11_8);
abc_11: abcgate port map(a => g2_7_6, b => p2_7_6, c =>
    ↪ g2_5_4, abc => g3_7_4);
abc_12: abcgate port map(a => g2_3_2, b => p2_3_2, c =>
    ↪ g2_1_0, abc => g3_3_0);

and_9: andgate port map(a =>p2_15_14, b=>p2_13_12, prod=>
    ↪ p3_15_12);
and_10: andgate port map(a =>p2_11_10, b=>p2_9_8, prod=>
    ↪ p3_11_8);
and_11: andgate port map(a =>p2_7_6, b=>p2_5_4, prod=>
    ↪ p3_7_4);
and_12: andgate port map(a =>p2_3_2, b=>p2_1_0, prod=>
    ↪ p3_3_0);

--generate p4, g4 signals --
abc_13: abcgate port map(a => g3_15_12, b => p3_15_12, c =>
    ↪ g3_11_8, abc => g4_15_8);
abc_14: abcgate port map(a => g3_7_4, b => p3_7_4, c =>
    ↪ g3_3_0, abc => g4_7_0);

and_13: andgate port map(a =>p3_15_12, b=>p3_11_8, prod=>
    ↪ p4_15_8);
and_14: andgate port map(a =>p3_7_4, b=>p3_3_0, prod=>
    ↪ p4_7_0);

--generate g5, p5 signals --
abc_15: abcgate port map(a => g4_15_8, b => p4_15_8, c =>
    ↪ g4_7_0, abc => g5_15_0);

```



```

and_15: andgate port map(a => p4_15_8, b => p4_7_0, prod=>
    ↪ p5_15_0);

--generate carry signals --
abc_c2: abcgate port map(b => p2_1_0, a=>g2_1_0, c=>cin,
    ↪ abc => c(2));
abc_c3: abcgate port map(b => p1_i(2), a=>g1_i(2), c=>c(2),
    ↪ abc => c(3));
abc_c4: abcgate port map(b => p3_3_0, a=>g3_3_0, c=>cin,
    ↪ abc => c(4));
abc_c5: abcgate port map(b => p1_i(4), a=>g1_i(4), c=>c(4),
    ↪ abc => c(5));
abc_c6: abcgate port map(b => p2_5_4, a=>g2_5_4, c=>c(4),
    ↪ abc => c(6));
abc_c7: abcgate port map(b => p1_i(6), a=>g1_i(6), c=>c(6),
    ↪ abc => c(7));
abc_c8: abcgate port map(b => p4_7_0, a=>g4_7_0, c=>cin,
    ↪ abc => c(8));
abc_c9: abcgate port map(b => p1_i(8), a=>g1_i(8), c=>c(8),
    ↪ abc => c(9));
abc_c10: abcgate port map(b => p2_9_8, a=>g2_9_8, c=>c(8),
    ↪ abc => c(10));
abc_c11: abcgate port map(b => p1_i(10), a=>g1_i(10), c=>c
    ↪ (10), abc => c(11));
abc_c12: abcgate port map(b => p3_11_8, a=>g3_11_8, c=>c(8)
    ↪ , abc => c(12));
abc_c13: abcgate port map(b => p1_i(12), a=>g1_i(12), c=>c
    ↪ (12), abc => c(13));
abc_c14: abcgate port map(b => p2_13_12, a=>g2_13_12, c=>c
    ↪ (12), abc => c(14));
abc_c15: abcgate port map(b => p1_i(14), a=>g1_i(14), c=>c
    ↪ (14), abc => c(15));
abc_c16: abcgate port map(b => p5_15_0, a=>g5_15_0, c=>cin,
    ↪ abc => c(16));

```

```

--sum signals--
sumxors: for i in 0 to 15 generate
    sum_xor_i: xorgate port map(a => p1_i(i), b => c(i),
        ↪ uneq => s(i));
end generate sumxors;

--carry out and carry in--
c(0) <= cin;
cout <= c(16);
end behave;

```

2 Testing of the Circuit

2.1 Testcase Generation

A python script `testcase_generator.py` was created to randomly generate 10 testcases to test on the adder everytime and stores them in `testcases.txt`.

2.1.1 Code present in `testcase_generator.py`

```

import random
no_of_testcases = 10

with open("testcases.txt", 'w') as t:
    for i in range(10):
        cin = random.randint(0, 1)
        a = random.randint(0, 65536)
        b = random.randint(0, 65536)
        s = a + b + cin
        cout = 0 if (s <= 65536) else 1
        s = s & 0xFFFF
        line = str(cin) + str("{0:b}".format(a).zfill(16)) +
            ↪ str("{0:b}".format(b).zfill(16)) + "␣" + str(cout
            ↪ ) + str("{0:b}".format(s).zfill(16))
        t.write(line+'\n')

```

2.2 Testbench

A testbench was written in VHDL allowing taking the inputs from `testcases.txt` and tests the circuit on the inputs and compares the outputs generated with the correct outputs and stores the **outputs** and **testcases which are wrong** in `results.txt`. It uses `assert` to check whether all testcases have passed. If not it throws an exception.

2.2.1 Code present in `testbench.vhdl`

```
-- Rohan Rajesh Kalbag --  
-- Roll Number: 20D170033, last digit is 3 --  
-- Testbench --  
  
library std;  
use std.textio.all;  
  
library ieee;  
use ieee.std_logic_1164.all;  
  
entity tb is  
end entity;  
  
architecture behave of tb is  
    -- component declaration --  
    component brentkung is  
        port(  
            a,b: in std_logic_vector(15 downto 0);  
            s: out std_logic_vector(15 downto 0);  
            cout: out std_logic;  
            cin: in std_logic  
        );  
    end component;  
  
    -- signal declaration --  
    signal input_vector: std_logic_vector(32 downto 0);  
    signal output_vector: std_logic_vector(16 downto 0);
```

```

-- function to convert bit_string to 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;

-- function to convert bit_string to std_logic_vector --
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 convert std_logic_vector to bit_string
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
    dut1: brentkung
    port map(
        a => input_vector(31 downto 16),
        b => input_vector(15 downto 0),
        cin => input_vector(32),
        s => output_vector(15 downto 0),
        cout => output_vector(16)
    );

    main: process
        -- interface with files --
        file infile: text open read_mode is "testcases.txt";
        file outfile: text open write_mode is "results.txt";

        -- variable declaration --
        variable in_var: bit_vector (32 downto 0);
        variable out_var: bit_vector (16 downto 0);
        variable flag : boolean := true;
        variable testcase : integer := 0;
        variable input_line: Line;
        variable output_line: Line;

    begin
        while not endfile(infile) loop
            testcase := testcase + 1;
            readLine(infile, input_line);
            read(input_line, in_var);
            read(input_line, out_var);

            -- apply inputs to the DUT --
            input_vector <= to_std_logic_vector(in_var);
            wait for 10 ns;
        end loop;
    end
end

```

```

-- check if the outputs are correct --
if(output_vector = to_std_logic_vector(out_var))
    ↪ then
        flag := flag and true;
    else
        flag := false;
        write(output_line, to_string("Error:␣Testcase␣"
            ↪ & integer'image(testcase)));
        writeline(outfile, output_line);
    end if;

-- write to results.txt --
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);
wait for 5 ns;
end loop;

-- assert for check if all testcases passed --
assert (not flag) report "SUCCESS,␣All␣Testcases␣out␣of␣"
    ↪ ␣" & integer'image(testcase) & "␣Passed!"
    ↪ severity note;
assert (flag) report "FAILURE,␣Few␣Testcases␣out␣of␣" &
    ↪ integer'image(testcase) & "␣Failed" severity
    ↪ error;
report "Design␣verification␣completed";
wait;
end process;
end;

```

2.3 Simulation of DUT using GHDL

The latest version of `ghdl` was installed and the bash script `test.sh` was created to minimize the commands to be entered on terminal while testing the circuit and preparing the testcases.

2.3.1 Code present in test.sh

```
echo "generating_random_testcases"
python testcase_generator.py
echo "testcases_primed"

echo "starting_testing"
ghdl -a gates.vhdl
ghdl -e andgate
ghdl -e xorgate
ghdl -e abcgate

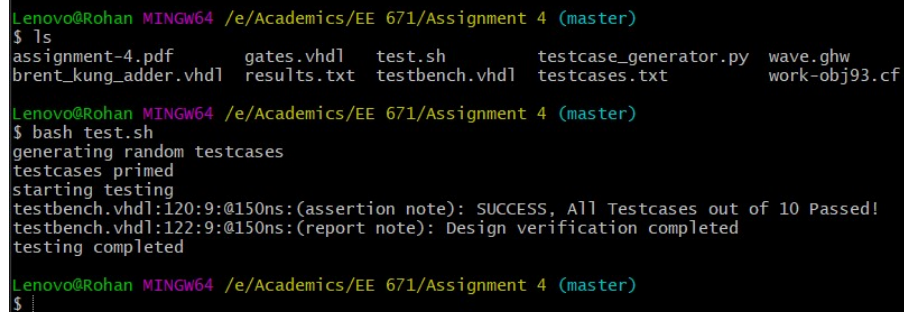
ghdl -a brent_kung_adder.vhdl
ghdl -e brentkung

ghdl -a testbench.vhdl
ghdl -e tb
ghdl -r tb --wave=wave.ghw
echo "testing_completed"
```

This script provides reports on whether the testcases have passed or not. Additionally creates a `wave.ghw` file which can be used to study all the signals and the waveform produced.

3 Results

3.1 Terminal Output After Executing test.sh



```
Lenovo@Rohan MINGW64 /e/Academics/EE 671/Assignment 4 (master)
$ ls
assignment-4.pdf    gates.vhdl    test.sh       testcase_generator.py  wave.ghw
brent_kung_adder.vhdl  results.txt  testbench.vhdl  testcases.txt         work-obj93.cf

Lenovo@Rohan MINGW64 /e/Academics/EE 671/Assignment 4 (master)
$ bash test.sh
generating random testcases
testcases primed
starting testing
testbench.vhdl:120:9:@150ns:(assertion note): SUCCESS, All Testcases out of 10 Passed!
testbench.vhdl:122:9:@150ns:(report note): Design verification completed
testing completed

Lenovo@Rohan MINGW64 /e/Academics/EE 671/Assignment 4 (master)
$ |
```

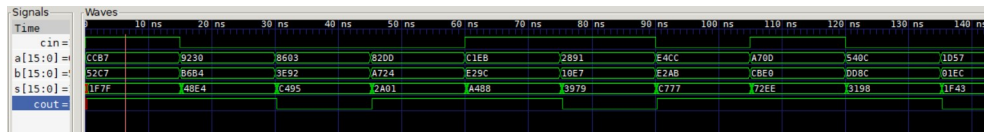
3.2 The Testcases Generated for the above Test

The testcases generated in `testcases.txt` were

```
111001100101101110101001011000111 10001111101111111
010010010001100001011011010110100 10100100011100100
010000110000000110011111010010010 01100010010010101
010000010110111011010011100100100 10010101000000001
111000001111010111110001010011100 11010010010001000
100101000100100010001000011100111 00011100101111001
011100100110011001110001010101011 11100011101110111
110100111000011011100101111100000 10111001011101110
001010100000011001101110110001100 10011000110011000
000011101010101110000000111101100 00001111101000011
```

3.3 Waveform Obtained on GTKWave

3.3.1 For all testcases

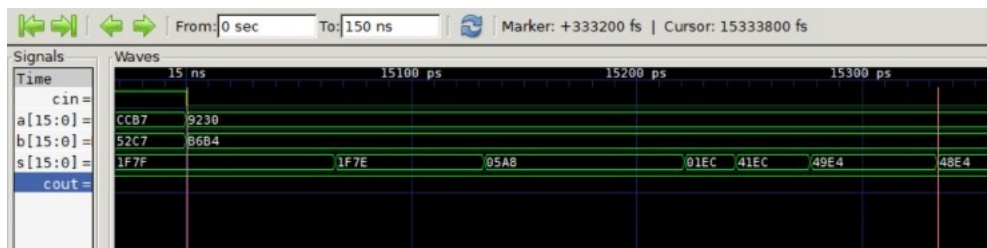


3.3.2 Transitions before achieving final value due to gate delays

For this case the values inputted to the DUT were

`a[15:0] = 0x9230`, `b[15:0] = 0xB6B4`, `cin = 0`

The delay obtained was **333.2 ps**



Hence we see the design for the **Brent Kung Adder** works as intended.

4 Submission Details

The submission contains `brent_kung_adder.vhdl`, `gates.vhdl`, `testbench.vhdl`, `testcase_generator.py`, the bash script `test.sh`, the outputs and testcases for the test simulated above and waveform generated `wave.ghw`.

5 System Requirements

- Linux Operating System or Git Bash/WSL on Windows
- GHDL - Open Source VHDL Simulator
- GTKwave - Open Source Waveform Analyser
- Python 3.x

6 Instructions to Test

- Make sure all system requirements are satisfied
- Open a terminal in the project directory
- Type the following - `bash test.sh`
- The waveform can be accessed by opening `wave.ghw` using GTKWave