EE671 VLSI Design - Assignment 5

Rohan Rajesh Kalbag, 20D170033

November 2, 2022

1 Approach

The VHDL hardware descriptions given for `andgate`, `abcgate`, `xorgate`, `Cin_map_G` in Assignment 4 were directly imported into `gates.vhdl` **and was kept unchanged**. The 16-bit Brent Kung adder `brentkung` description was moved into a file called `adders.vhdl`. Descriptions for **Full Adder** with entity name `fa`, and **Half Adder** with entity name `ha` were created using the gates provided as follows and stored in `adders.vhdl`.

1.1 Descriptions of Half Adder and Full Adder present in `adders.vhdl`

```
--Half Adder--
library IEEE;
use IEEE.std_logic_1164.all;

entity ha is
    port(
        a,b: in std_logic;
        s, c: out std_logic
    );
end entity;
```

```
architecture behave of ha is
    -- 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;
begin
    a1: andgate port map(a => a, b => b, prod => c);
    x1: xorgate port map(a => a, b => b, uneq => s);
end behave;

--Full Adder--
library IEEE;
use IEEE.std_logic_1164.all;

entity fa is
     port(
        a,b,cin: in std_logic;
        s,cout: out std_logic
    );
end entity;

architecture behave of fa is
    -- 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 Cin_map_G is
        port(A, B, Cin: in std_logic;
        Bit0_G: out std_logic);
    end component Cin_map_G;

    signal t1: std_logic;

begin
    g1: Cin_map_G port map(cin => cin, a => a, b => b, bit0_g
        ↪ => cout);
    x1: xorgate port map(a => a, b => b, uneq => t1);
    x2: xorgate port map(a => t1, b => cin, uneq => s);
end behave;
```

1.2 Designing the Multiply and Accumulate Circuit

1.2.1 Notation

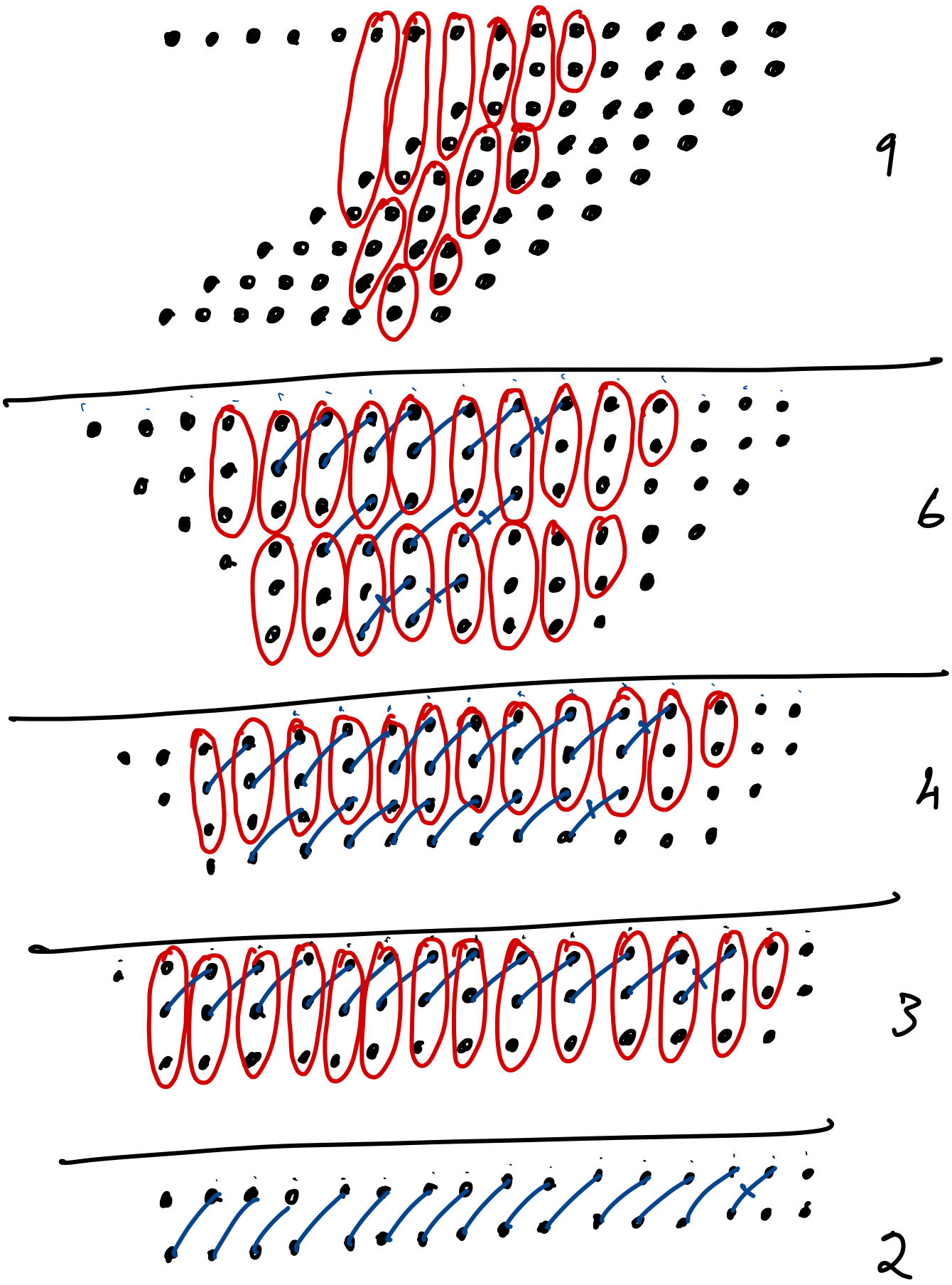
The two eight bit numbers are given by A and B. C denotes the 16 bit number to be accumulated. c_i, a_i, b_i denote the ith bit of C, A and B.

We wish to evaluate $A \cdot B + C$. We first find the partial products $a_i \cdot b_i$ using AND gates. Then we use Dadda's rules to reduce the number of wires using an appropriate number of Full and Half Adders, we try to minimize the number of adders used and only do so if the wire capacity of each layer in the minimization process is exceeded.

After minimizing to 32 wires, two 16 bit numbers basically, we make use of the 16 bit Brent Kung Adder from Assignment 4 for adding them up and we obtain the result we are looking for.

1.2.2 Dot Diagram

The Dot Diagram follows next page, here a red circle enclosing 2 dots denotes a HA and a red circle enclosing 3 dots is a FA. A blue line between two dots, A crossed blue line between two dots indicates sum (start of line) and carry (end of line) for FA, HA respectively. (Notation - Paper of Wallace - 1964)


9
6
4
3
2

The implementation of the MAC was done in `mac.vhdl`, where the connections were made according to the above dot diagram. The number on the right side of each reduction layer denotes the max capacity of wires for that particular layer.

1.3 Code for MAC present in `mac.vhdl`

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

entity multiply_accumulate is
    -- perform ab + c --
    port(
        a, b: in std_logic_vector(7 downto 0);
        c: in std_logic_vector(15 downto 0);
        s: out std_logic_vector(15 downto 0);
        cout: out std_logic
    );
end entity;

architecture behave of multiply_accumulate is
    -- component declarations --
    component andgate is
        port (a, b: in std_logic;
        prod: out std_logic);
    end component;

    component fa is
        port(
            a,b,cin: in std_logic;
            s,cout: out std_logic
        );
    end component;
```

```
component ha is
    port(
        a,b: in std_logic;
        s, c: out std_logic
    );
end component;

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;

-- LOW signal for carry in of Brent Kung Adder--
signal gnd_sig: std_logic := '0';

-- layer 1 signal declarations--
-- we make rows corresponding to the first layer--
signal layer1_r1: std_logic_vector(15 downto 0);
signal layer1_r2: std_logic_vector(7 downto 0);
signal layer1_r3: std_logic_vector(8 downto 1);
signal layer1_r4: std_logic_vector(9 downto 2);
signal layer1_r5: std_logic_vector(10 downto 3);
signal layer1_r6: std_logic_vector(11 downto 4);
signal layer1_r7: std_logic_vector(12 downto 5);
signal layer1_r8: std_logic_vector(13 downto 6);
signal layer1_r9: std_logic_vector(14 downto 7);

-- signals for connections--
signal wires: std_logic_vector(83 downto 0);
```

```
    -- final sum argument signals--
    -- first argument (top row of final layer)--
    signal arg1: std_logic_vector(15 downto 0);
    -- second argument (bottom row of final layer)--
    signal arg2: std_logic_vector(15 downto 0);

begin
    -- set first row as 16 bit sum--
    layer1_r1 <= c;

    -- [a1 a2 a3 ... a8] is the 8x8 array of AND gates--
    -- accordingly set the subsequent signals for other rows--
    a1: for i in 0 to 7 generate
        and_i: andgate port map(a => b(i), b => a(0), prod =>
            ↪ layer1_r2(i));
    end generate a1;

    a2: for i in 0 to 7 generate
        and_i: andgate port map(a => b(i), b => a(1), prod =>
            ↪ layer1_r3(i + 1));
    end generate a2;

    a3: for i in 0 to 7 generate
        and_i: andgate port map(a => b(i), b => a(2), prod =>
            ↪ layer1_r4(i + 2));
    end generate a3;

    a4: for i in 0 to 7 generate
        and_i: andgate port map(a => b(i), b => a(3), prod =>
            ↪ layer1_r5(i + 3));
    end generate a4;

    a5: for i in 0 to 7 generate
        and_i: andgate port map(a => b(i), b => a(4), prod =>
            ↪ layer1_r6(i + 4));
    end generate a5;
```

```
a6: for i in 0 to 7 generate
    and_i: andgate port map(a => b(i), b => a(5), prod =>
       ↪ layer1_r7(i + 5));
end generate a6;

a7: for i in 0 to 7 generate
    and_i: andgate port map(a => b(i), b => a(6), prod =>
       ↪ layer1_r8(i + 6));
end generate a7;

a8: for i in 0 to 7 generate
    and_i: andgate port map(a => b(i), b => a(7), prod =>
       ↪ layer1_r9(i + 7));
end generate a8;

-- connections are made column wise as per the diagram--
--column 1--
arg1(0) <= layer1_r1(0);
arg2(0) <= layer1_r2(0);

--column 2--
h1: ha port map(a => layer1_r1(1), b => layer1_r2(1), s =>
   ↪ arg1(1), c => arg2(2));
arg2(1) <= layer1_r3(1);

--column 3--
h2: ha port map(a => layer1_r1(2), b => layer1_r2(2), s =>
   ↪ wires(0), c => wires(1));
f1: fa port map(a => wires(0), b => layer1_r3(2), cin=>
   ↪ layer1_r4(2), s => arg1(2), cout => arg2(3));

--column 4--
h3: ha port map(a => layer1_r1(3), b => layer1_r2(3), s =>
   ↪ wires(2), c => wires(3));
f2: fa port map(a => wires(2), b => layer1_r3(3), cin =>
   ↪ layer1_r4(3), s => wires(4), cout => wires(5));
f3: fa port map(a => layer1_r5(3), b => wires(1), cin =>
   ↪ wires(4), s => arg1(3), cout => arg2(4));
```

```
--column 5--
f4: fa port map(a => layer1_r1(4), b => layer1_r2(4), cin
   ↪ => layer1_r3(4), s => wires(6), cout => wires(7));
h4: ha port map(a => layer1_r4(4), b => layer1_r5(4), s =>
   ↪ wires(8), c => wires(9));
f5: fa port map(a => wires(6), b => wires(8), cin => wires
   ↪ (3), s => wires(10), cout =>wires(11));
f6: fa port map(a => wires(10), b => wires(5), cin =>
   ↪ layer1_r6(4), s => arg1(4), cout =>arg2(5));

--column 6--
h5: ha port map(a => layer1_r1(5), b => layer1_r2(5), s =>
   ↪ wires(12), c =>wires(13));
f7: fa port map(a => wires(12), b => layer1_r3(5), cin =>
   ↪ layer1_r4(5), s => wires(14), cout =>wires(15));
f8: fa port map(a => layer1_r5(5), b => layer1_r6(5), cin
   ↪ => layer1_r7(5), s => wires(16), cout =>wires(17));
f9: fa port map(a => wires(14), b => wires(16), cin =>
   ↪ wires(7), s => wires(18), cout =>wires(19));
f10: fa port map(a => wires(18), b => wires(9) , cin =>
   ↪ wires(11), s => arg1(5), cout => arg2(6));

--column 7--
f11: fa port map(a => layer1_r1(6), b => layer1_r2(6), cin
   ↪ => layer1_r3(6), s => wires(20), cout => wires(83));
h6: ha port map(a => layer1_r4(6), b => layer1_r5(6), s =>
   ↪ wires(21), c =>wires(22));
f12: fa port map(a => wires(20), b => wires(21), cin =>
   ↪ wires(13), s => wires(23), cout => wires(24));
f13: fa port map(a => layer1_r6(6), b => layer1_r7(6), cin
   ↪ => layer1_r8(6), s => wires(25), cout => wires(26));
f14: fa port map(a => wires(23), b => wires(25), cin =>
   ↪ wires(15), s => wires(27), cout => wires(28));
f15: fa port map(a => wires(27), b => wires(17), cin =>
   ↪ wires(19), s => arg1(6), cout => arg2(7));
```

```
--column 8--
f16: fa port map(a => layer1_r1(7), b => layer1_r2(7), cin
   ↪ => layer1_r3(7), s => wires(29), cout => wires(30));
f17: fa port map(a => layer1_r4(7), b => layer1_r5(7), cin
   ↪ => layer1_r6(7), s => wires(31), cout => wires(32));
h7: ha port map(a => layer1_r7(7), b => layer1_r8(7), s =>
   ↪ wires(33), c => wires(34));
f18: fa port map(a=>wires(29), b=>wires(31), cin=>wires(83)
   ↪ , s=>wires(35), cout=>wires(36));
f19: fa port map(a=>wires(33), b=>layer1_r9(7), cin=>wires
   ↪ (22), s=>wires(37), cout=>wires(38));
f20: fa port map(a=>wires(35), b=>wires(37), cin=>wires(24)
   ↪ , s=>wires(39), cout=>wires(40));
f21: fa port map(a=>wires(39), b=>wires(26), cin=>wires(28)
   ↪ , s=>arg1(7), cout=>arg2(8));

--column 9--
f22: fa port map(a=>layer1_r1(8), b=>layer1_r3(8), cin=>
   ↪ layer1_r4(8), s=>wires(41), cout=>wires(42));
f23: fa port map(a=>layer1_r5(8), b=>layer1_r6(8), cin=>
   ↪ layer1_r7(8), s=>wires(43), cout=>wires(44));
h8: ha port map(a=>layer1_r8(8), b=>layer1_r9(8), s=>wires
   ↪ (45), c=>wires(46));
f24: fa port map(a=>wires(41), b=>wires(30), cin=>wires(43)
   ↪ , s=>wires(47), cout=>wires(48));
f25: fa port map(a=>wires(45), b=>wires(32), cin=>wires(34)
   ↪ , s=>wires(49), cout=>wires(50));
f26: fa port map(a=>wires(47), b=>wires(49), cin=>wires(36)
   ↪ , s=>wires(51), cout=>wires(52));
f27: fa port map(a=>wires(38), b=>wires(51), cin=>wires(40)
   ↪ , s=>arg1(8), cout=>arg2(9));
```

```
--column 10--
f28: fa port map(a=>layer1_r1(9), b=>layer1_r4(9), cin=>
   ↪ layer1_r5(9), s=>wires(53), cout=>wires(54));
f29: fa port map(a=>layer1_r6(9), b=>layer1_r7(9), cin=>
   ↪ layer1_r8(9), s=>wires(55), cout=>wires(56));
f30: fa port map(a=>wires(53), b=>wires(55), cin=>wires(42)
   ↪ , s=>wires(57), cout=>wires(58));
f31: fa port map(a=>layer1_r9(9), b=>wires(44), cin=>wires
   ↪ (46), s=>wires(59), cout=>wires(60));
f32: fa port map(a=>wires(57), b=>wires(59), cin=>wires(48)
   ↪ , s=>wires(61), cout=>wires(62));
f33: fa port map(a=>wires(50), b=>wires(52), cin=>wires(61)
   ↪ , s=>arg1(9), cout=>arg2(10));

--column 11--
f34: fa port map(a=>layer1_r1(10), b=>layer1_r5(10), cin=>
   ↪ layer1_r6(10), s=>wires(63), cout=>wires(64));
f35: fa port map(a=>wires(63), b=>layer1_r7(10), cin=>wires
   ↪ (54), s=>wires(65), cout=>wires(66));
f36: fa port map(a=>wires(56), b=>layer1_r8(10), cin=>
   ↪ layer1_r9(10), s=>wires(67), cout=>wires(68));
f37: fa port map(a=>wires(65), b=>wires(67), cin=>wires(58)
   ↪ , s=>wires(69), cout=>wires(70));
f38: fa port map(a=>wires(60), b=>wires(62), cin=>wires(69)
   ↪ , s=>arg1(10), cout=>arg2(11));

--column 12--
f39: fa port map(a=>layer1_r1(11), b=>layer1_r6(11), cin=>
   ↪ wires(64), s=>wires(71), cout=>wires(72));
f40: fa port map(a=>layer1_r7(11), b=>layer1_r8(11), cin=>
   ↪ layer1_r9(11), s=>wires(73), cout=>wires(74));
f41: fa port map(a=>wires(71), b=>wires(73), cin=>wires(66)
   ↪ , s=>wires(75), cout=>wires(76));
f42: fa port map(a=>wires(68), b=>wires(70), cin=>wires(75)
   ↪ , s=>arg1(11), cout=>arg2(12));
```

```
    --column 13--
    f43: fa port map(a=>layer1_r1(12), b=>layer1_r7(12), cin=>
       ↪ layer1_r8(12), s=>wires(77), cout=>wires(78));
    f44: fa port map(a=>layer1_r9(12), b=>wires(77), cin=>wires
       ↪ (72), s=>wires(79), cout=>wires(80));
    f45: fa port map(a=>wires(74), b=>wires(76), cin=>wires(79)
       ↪ , s=>arg1(12), cout=>arg2(13));

    --column 14--
    f46: fa port map(a=>layer1_r1(13), b=>layer1_r8(13), cin=>
       ↪ wires(78), s=>wires(81), cout=>wires(82));
    f47: fa port map(a=>layer1_r9(13), b=>wires(80), cin=>wires
       ↪ (81), s=>arg1(13), cout=>arg2(14));

    --column 15--
    f48: fa port map(a=>layer1_r1(14), b=>layer1_r9(14), cin=>
       ↪ wires(82), s=>arg1(14), cout=>arg2(15));

    --column 16--
    arg1(15) <= layer1_r1(15);

    --final adder to compute arg1 + arg2--
    bkadder: brentkung port map(a =>arg1, b=>arg2, cin=>gnd_sig
       ↪ , s=>s, cout=>cout);

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(no_of_testcases):
        a = random.randint(0, 255)
        b = random.randint(0, 255)
        c = random.randint(0, 65536)
        s = a*b + c
        cout = 0 if (s <= 65536) else 1
        s = s & 0xFFFF
        line = str("{0:b}".format(a).zfill(8)) + str("{0:b}".
            ↪ format(b).zfill(8)) + str("{0:b}".format(c).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`

```
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 multiply_accumulate is
        --perform ab + c --
        port(
            a, b: in std_logic_vector(7 downto 0);
            c: in std_logic_vector(15 downto 0);
            s: out std_logic_vector(15 downto 0);
            cout: out std_logic
        );
    end component;

    -- signal declaration --
    signal input_vector: std_logic_vector(31 downto 0);
    signal output_vector: std_logic_vector(16 downto 0);
    signal correct_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: multiply_accumulate
port map(
  a => input_vector(31 downto 24),
  b => input_vector(23 downto 16),
  c => input_vector(15 downto 0),
  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 (31 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;
        correct_output_vector <= to_std_logic_vector(out_var
            ↪ );
        -- 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 -e Cin_map_G

ghdl -a adders.vhdl
ghdl -e brentkung
ghdl -e fa
ghdl -e ha
```

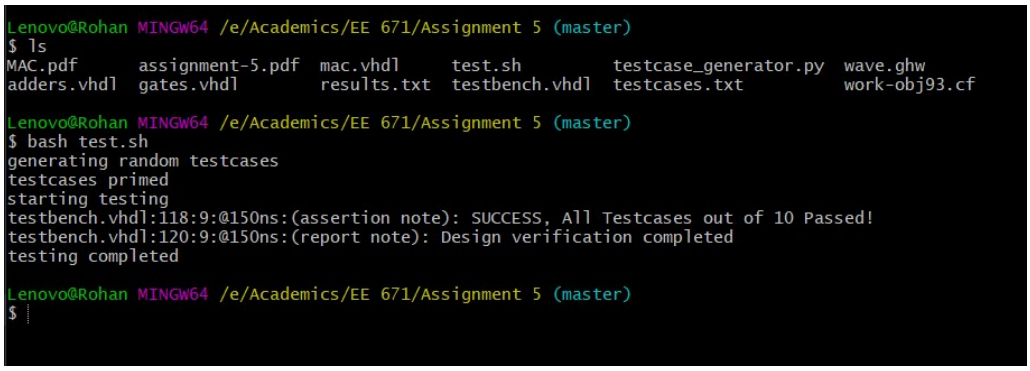
```
ghdl -a mac.vhdl
ghdl -e multiply_accumulate

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 makes `wave.ghw` file which can be used to study all the signals

3 Results

3.1 Terminal Output After Executing `test.sh`

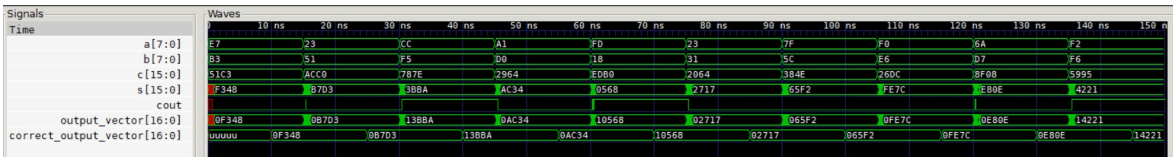


3.2 The Testcases Generated for the above Test

```
11100111101100110101000111000011 01111001101001000
00100011010100011010110011000000 01011011111010011
11001100111101010111100001111110 10011101110111010
10100001110100000010100101100100 01010110000110100
11111101000110001110110110110000 10000010101101000
00100011001100010010000001100100 00010011100010111
01111111010111000011100001001110 00110010111110010
11110000111001100010011011011100 01111111001111100
01101010110101111000111100001000 01110100000001110
11110010111101100101100110010101 10100001000100001
```

3.3 Waveform Obtained on GTKWave

3.3.1 For all testcases

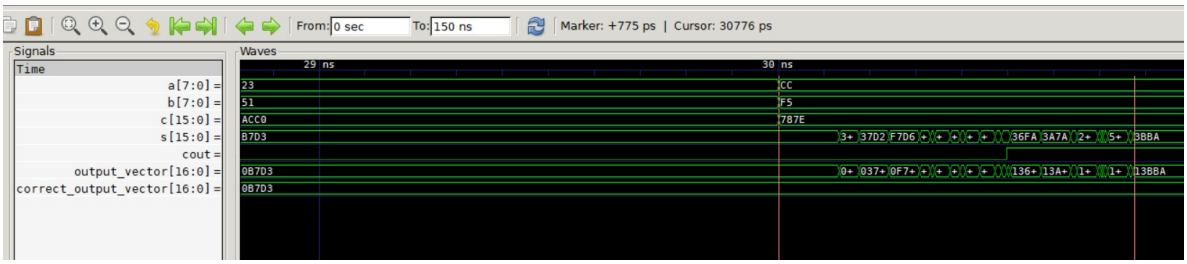


3.3.2 Transitions before achieving final value due to gate delays

For this case the values inputted to the DUT were

`a = 0xCC`, `b = 0xF5`, `c = 0x787E`

The delay obtained was **775 ps**



Hence we see the design for the **Multiply and Accumulate Circuit** works as intended.

4 Submission Details

The submission contains `mac.vhdl`, `adders.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`. The RTL was synthesised using Intel Quartus, its project files have been included in `rtl_files.zip`

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

7 Intel Quartus Generated RTL

Just for the sake of completeness, the **RTL** was generated for the `mac` entity using **Intel Quartus**. The RTL netlist viewer has been included in the next page.