Assignment - 4

Multiply and Accumulate Circuit with Dadda Reduction Scheme

EE 671 - VLSI Design

Name:	**Satvikkumar Patel**
Roll number:	23M1117
Program:	M.tech (EE5 - Electronic Systems)
Department:	Electrical Engineering
Date of Sub:	November 2, 2023

Monday Oct. 16, 2023	**EE 671: VLSI Design** **Assignment 4**	Due on Saturday Oct. 28, 2023

Describe a Multiply-Accumulate circuit which will multiply two 16bit unsigned integer operands and add it to a 32 bit unsigned accumulator. The multiplier should use the Dadda reduction scheme and the 32 bit Brent Kung adder designed by you in the previous assignment for the final addition.

Your description should include a test bench which reads its operands from a file

Your description should use `std_logic` types for various signals. You can use the public domain tool "ghdl" for VHDL simulation.

Compose half adders and full adders using the basic gates (with roll number dependent gate delays) described for the previous assignment.

Test data should contain 10 randomly chosen input combinations. The test bench should use assert statements to flag errors if there is a mismatch between the computed sum/carry and the stored sum/carry.

Note: The pen and paper solution is there in the zip file.

Trivial components used:

Delays are taken according to the roll number scheme.

- AND gate - 317ps
- XOR gate - 634ps
- A+BC gate - 417ps
- AB+C(A+B) - 634ps

```
-- 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 317 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 634 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 417 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_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 634 ps;
```

```
end architecture trivial;

-- FUll adder

library IEEE;
use IEEE.std_logic_1164.all;

entity FA is
port(A, B, Cin: in std_logic;
sum, cout : out std_logic);
end entity FA;

architecture trivial of FA is

component xorgate is
    port (A, B: in std_logic;
    uneq: 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 temp_sum : std_logic;

begin

 FA_temp_sum : xorgate port map (A, B, temp_sum);
 FA_Sum   : xorgate port map (cin, temp_sum, sum);
 FA_carryout : cin_map_G port map (A, B, Cin, cout);

end architecture trivial;

--Half Adder

library IEEE;
use IEEE.std_logic_1164.all;

entity HA is
port(A, B : in std_logic;
sum, cout : out std_logic);
end entity HA;

architecture trivial of HA is

component andgate
 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;


begin

 HA_sum : xorgate port map (A, B, sum);
 HA_carryout : andgate port map (A, B, cout);
```

```

end architecture trivial;
```

VHDL Code for MAC with Dadda reduction scheme:

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

entity MAC_32 is
 port(a, b : in std_logic_vector(15 downto 0);
      acc : in std_logic_vector(31 downto 0);
      result : out std_logic_vector(32 downto 0));
end entity;

architecture a1 of MAC_32 is
signal cin : std_logic := '0';

-- stage0 signals

signal s0_0 : std_logic_vector(31 downto 0);
signal s0_1 : std_logic_vector(15 downto 0);
signal s0_2 : std_logic_vector(16 downto 1);
signal s0_3 : std_logic_vector(17 downto 2);
signal s0_4 : std_logic_vector(18 downto 3);
signal s0_5 : std_logic_vector(19 downto 4);
signal s0_6 : std_logic_vector(20 downto 5);
signal s0_7 : std_logic_vector(21 downto 6);
signal s0_8 : std_logic_vector(22 downto 7);
signal s0_9 : std_logic_vector(23 downto 8);
signal s0_10 : std_logic_vector(24 downto 9);
signal s0_11 : std_logic_vector(25 downto 10);
signal s0_12 : std_logic_vector(26 downto 11);
signal s0_13 : std_logic_vector(27 downto 12);
signal s0_14 : std_logic_vector(28 downto 13);
signal s0_15 : std_logic_vector(29 downto 14);
signal s0_16 : std_logic_vector(30 downto 15);

-- stage1 signals

signal s1_0 : std_logic_vector(31 downto 0);
signal s1_1 : std_logic_vector(30 downto 0);
signal s1_2 : std_logic_vector(29 downto 1);
signal s1_3 : std_logic_vector(28 downto 2);
signal s1_4 : std_logic_vector(27 downto 3);
signal s1_5 : std_logic_vector(26 downto 4);
signal s1_6 : std_logic_vector(25 downto 5);
signal s1_7 : std_logic_vector(24 downto 6);
signal s1_8 : std_logic_vector(23 downto 7);
signal s1_9 : std_logic_vector(22 downto 8);
signal s1_10 : std_logic_vector(21 downto 9);
signal s1_11 : std_logic_vector(20 downto 10);
signal s1_12 : std_logic_vector(20 downto 11);

-- stage2 signals

signal s2_0: std_logic_vector(31 downto 0);
signal s2_1: std_logic_vector(30 downto 0);
signal s2_2: std_logic_vector(29 downto 1);
signal s2_3: std_logic_vector(28 downto 2);
```

```
signal s2_4: std_logic_vector(27 downto 3);
signal s2_5: std_logic_vector(26 downto 4);
signal s2_6: std_logic_vector(25 downto 5);
signal s2_7: std_logic_vector(24 downto 6);
signal s2_8: std_logic_vector(24 downto 7);

-- stage3 signals

signal s3_0 : std_logic_vector(31 downto 0);
signal s3_1 : std_logic_vector(30 downto 0);
signal s3_2 : std_logic_vector(29 downto 1);
signal s3_3 : std_logic_vector(28 downto 2);
signal s3_4 : std_logic_vector(27 downto 3);
signal s3_5 : std_logic_vector(27 downto 4);

-- stage4 signals

signal s4_0 : std_logic_vector(31 downto 0);
signal s4_1 : std_logic_vector(30 downto 0);
signal s4_2 : std_logic_vector(29 downto 1);
signal s4_3 : std_logic_vector(29 downto 2);

-- stage5 signals

signal s5_0 : std_logic_vector(31 downto 0);
signal s5_1 : std_logic_vector(30 downto 0);
signal s5_2 : std_logic_vector(30 downto 1);

-- stage6 signals

signal s6_0 : std_logic_vector(31 downto 0);
signal s6_1 : std_logic_vector(31 downto 0);


component andgate
port (A, B: in std_logic;
      prod: out std_logic);
end component andgate;

component HA is  -- used for half adder
port (A, B: in std_logic;
      sum, cout: out std_logic);
end component HA;

component FA is  -- used for full adder
port (A, B, Cin: in std_logic;
      sum, cout: out std_logic);
end component FA;

component brent_kung 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 component brent_kung;

begin

 -- stage0 signals assignment
 s0_0 <= acc;

 g1: for i in 0 to 15 generate
```

```
r1: andgate port map (a(i), b(0), s0_1(i));
end generate g1;

g2: for i in 1 to 16 generate
r2: andgate port map (a(i-1), b(1), s0_2(i));
end generate g2;

g3: for i in 2 to 17 generate
r3: andgate port map (a(i-2), b(2), s0_3(i));
end generate g3;

g4 :for i in 3 to 18 generate
r4: andgate port map (a(i-3), b(3), s0_4(i));
end generate g4;

g5: for i in 4 to 19 generate
r5: andgate port map(a(i-4), b(4), s0_5(i));
end generate g5;

g6: for i in 5 to 20 generate
r6: andgate port map (a(i-5), b(5), s0_6(i));
end generate g6;

g7: for i in 6 to 21 generate
r7: andgate port map (a(i-6), b(6), s0_7(i));
end generate g7;

g8: for i in 7 to 22 generate
r8: andgate port map (a(i-7), b(7), s0_8(i));
end generate g8;

g9: for i in 8 to 23 generate
r9: andgate port map (a(i-8), b(8), s0_9(i));
end generate g9;

g10: for i in 9 to 24 generate
r10: andgate port map (a(i-9), b(9), s0_10(i));
end generate g10;

g11: for i in 10 to 25 generate
r11: andgate port map (a(i-10), b(10), s0_11(i));
end generate;

g12: for i in 11 to 26 generate
r12: andgate port map (a(i-11), b(11), s0_12(i));
end generate g12;

g13: for i in 12 to 27 generate
r13: andgate port map (a(i-12), b(12), s0_13(i));
end generate g13;

g14: for i in 13 to 28 generate
r14: andgate port map (a(i-13), b(13), s0_14(i));
end generate g14;

g15: for i in 14 to 29 generate
r15: andgate port map (a(i-14), b(14), s0_15(i));
end generate g15;

g16: for i in 15 to 30 generate
r16: andgate port map (a(i-15), b(15), s0_16(i));
end generate g16;
```

```
s2:  HA port map (s0_0(12), s0_1(12), s1_0(12), s1_2(13));

s3: HA port map (s0_3(13), s0_4(13), s1_1(13), s1_4(14));

s4 : FA port map (s0_0(13), s0_1(13), s0_2(13),s1_0(13),s1_3(14) );

s5 : FA port map (s0_0(14), s0_1(14), s0_2(14), s1_0(14),s1_4(15));

s6: FA port map (s0_3(14), s0_4(14), s0_5(14), s1_1(14),s1_5(15));

s7 : HA port map (s0_6(14), s0_7(14), s1_2(14),s1_6(15));

s8: FA port map (s0_0(15), s0_1(15), s0_2(15), s1_0(15),s1_4(16));

s9 : FA port map (s0_3(15), s0_4(15), s0_5(15), s1_1(15),s1_5(16));

s10 : FA port map (s0_6(15), s0_7(15), s0_8(15), s1_2(15),s1_6(16));

s11 :  HA port map (s0_9(15), s0_10(15), s1_3(15),s1_7(16));

s12 : FA port map (s0_0(16), s0_2(16), s0_3(16), s1_0(16), s1_3(17));

s13 : FA port map (s0_4(16), s0_5(16), s0_6(16), s1_1(16), s1_4(17));

s14 : FA port map (s0_7(16), s0_8(16), s0_9(16), s1_2(16),s1_5(17));

s15 : HA port map (s0_10(16), s0_11(16), s1_3(16),s1_6(17));

s16 : FA port map (s0_0(17), s0_3(17), s0_4(17), s1_0(17),s1_2(18));

s17 : FA port map (s0_5(17), s0_6(17), s0_7(17), s1_1(17),s1_3(18));

s18 : FA port map (s0_8(17), s0_9(17), s0_10(17), s1_2(17),s1_4(18));

s19 : FA port map (s0_0(18), s0_4(18), s0_5(18), s1_0(18),s1_1(19));

s20 : FA port map (s0_6(18), s0_7(18), s0_8(18), s1_1(18),s1_2(19));

s21 : FA port map (s0_0(19), s0_5(19), s0_6(19), s1_0(19),s1_1(20));


-- stage1 signals assignments
g17: for i in 20 to 31 generate
s1_0(i) <= s0_0(i);
end generate g17;
g18: for i in 0 to 11 generate
s1_0(i) <= s0_0(i);
s1_1(i) <= s0_1(i);
end generate g18;
g19: for i in 1 to 11 generate
s1_2(i) <= s0_2(i);
end generate g19;
g20: for i in 2 to 11 generate
s1_3(i) <= s0_3(i);
end generate g20;
g21: for i in 3 to 11 generate
s1_4(i) <= s0_4(i);
end generate g21;
g22: for i in 4 to 11 generate
s1_5(i) <= s0_5(i);
end generate g22;
```

```
g23: for i in 5 to 11 generate
s1_6(i) <= s0_6(i);
end generate g23;
g24: for i in 6 to 11 generate
s1_7(i) <= s0_7(i);
end generate g24;
g26: for i in 7 to 11 generate
s1_8(i) <= s0_8(i);
end generate g26;
g27: for i in 8 to 11 generate
s1_9(i) <= s0_9(i);
end generate g27;
g28: for i in 9 to 11 generate
s1_10(i) <= s0_10(i);
end generate g28;
g29: for i in 10 to 11 generate
s1_11(i) <= s0_11(i);
end generate g29;
s1_12(11) <= s0_12(11);

s1_1(12) <= s0_2(12);
s1_2(12) <= s0_3(12);
s1_3(12) <= s0_4(12);
s1_4(12) <= s0_5(12);
s1_5(12) <= s0_6(12);
s1_6(12) <= s0_7(12);
s1_7(12) <= s0_8(12);
s1_8(12) <= s0_9(12);
s1_9(12) <= s0_10(12);
s1_10(12) <= s0_11(12);
s1_11(12) <= s0_12(12);
s1_12(12) <= s0_13(12);


s1_3(13) <= s0_5(13);
s1_4(13) <= s0_6(13);
s1_5(13) <= s0_7(13);
s1_6(13) <= s0_8(13);
s1_7(13) <= s0_9(13);
s1_8(13) <= s0_10(13);
s1_9(13) <= s0_11(13);
s1_10(13) <= s0_12(13);
s1_11(13) <= s0_13(13);
s1_12(13) <= s0_14(13);

s1_5(14) <= s0_8(14);
s1_6(14) <= s0_9(14);
s1_7(14) <= s0_10(14);
s1_8(14) <= s0_11(14);
s1_9(14) <= s0_12(14);
s1_10(14) <= s0_13(14);
s1_11(14) <= s0_14(14);
s1_12(14) <= s0_15(14);

s1_7(15) <= s0_11(15);
s1_8(15) <= s0_12(15);
s1_9(15) <= s0_13(15);
s1_10(15) <= s0_14(15);
s1_11(15) <= s0_15(15);
s1_12(15) <= s0_16(15);

s1_8(16) <= s0_12(16);
```

```
s1_9(16) <= s0_13(16);
s1_10(16) <= s0_14(16);
s1_11(16) <= s0_15(16);
s1_12(16) <= s0_16(16);

s1_7(17) <= s0_11(17);
s1_8(17) <= s0_12(17);
s1_9(17) <= s0_13(17);
s1_10(17) <= s0_14(17);
s1_11(17) <= s0_15(17);
s1_12(17) <= s0_16(17);

s1_5(18) <= s0_9(18);
s1_6(18) <= s0_10(18);
s1_7(18) <= s0_11(18);
s1_8(18) <= s0_12(18);
s1_9(18) <= s0_13(18);
s1_10(18) <= s0_14(18);
s1_11(18) <= s0_15(18);
s1_12(18) <= s0_16(18);

s1_3(19) <= s0_7(19);
s1_4(19) <= s0_8(19);
s1_5(19) <= s0_9(19);
s1_6(19) <= s0_10(19);
s1_7(19) <= s0_11(19);
s1_8(19) <= s0_12(19);
s1_9(19) <= s0_13(19);
s1_10(19) <= s0_14(19);
s1_11(19) <= s0_15(19);
s1_12(19) <= s0_16(19);

s1_2(20) <= s0_6(20);
s1_3(20) <= s0_7(20);
s1_4(20) <= s0_8(20);
s1_5(20) <= s0_9(20);
s1_6(20) <= s0_10(20);
s1_7(20) <= s0_11(20);
s1_8(20) <= s0_12(20);
s1_9(20) <= s0_13(20);
s1_10(20) <= s0_14(20);
s1_11(20) <= s0_15(20);
s1_12(20) <= s0_16(20);

s1_1(21) <= s0_7(21);
s1_2(21) <= s0_8(21);
s1_3(21) <= s0_9(21);
s1_4(21) <= s0_10(21);
s1_5(21) <= s0_11(21);
s1_6(21) <= s0_12(21);
s1_7(21) <= s0_13(21);
s1_8(21) <= s0_14(21);
s1_9(21) <= s0_15(21);
s1_10(21) <= s0_16(21);

s1_1(22) <= s0_8(22);
s1_2(22) <= s0_9(22);
s1_3(22) <= s0_10(22);
s1_4(22) <= s0_11(22);
s1_5(22) <= s0_12(22);
s1_6(22) <= s0_13(22);
s1_7(22) <= s0_14(22);
```

```
s1_8(22) <= s0_15(22);
s1_9(22) <= s0_16(22);

s1_1(23) <= s0_9(23);
s1_2(23) <= s0_10(23);
s1_3(23) <= s0_11(23);
s1_4(23) <= s0_12(23);
s1_5(23) <= s0_13(23);
s1_6(23) <= s0_14(23);
s1_7(23) <= s0_15(23);
s1_8(23) <= s0_16(23);

s1_1(24) <= s0_10(24);
s1_2(24) <= s0_11(24);
s1_3(24) <= s0_12(24);
s1_4(24) <= s0_13(24);
s1_5(24) <= s0_14(24);
s1_6(24) <= s0_15(24);
s1_7(24) <= s0_16(24);

s1_1(25) <= s0_11(25);
s1_2(25) <= s0_12(25);
s1_3(25) <= s0_13(25);
s1_4(25) <= s0_14(25);
s1_5(25) <= s0_15(25);
s1_6(25) <= s0_16(25);

s1_1(26) <= s0_12(26);
s1_2(26) <= s0_13(26);
s1_3(26) <= s0_14(26);
s1_4(26) <= s0_15(26);
s1_5(26) <= s0_16(26);

s1_1(27) <= s0_13(27);
s1_2(27) <= s0_14(27);
s1_3(27) <= s0_15(27);
s1_4(27) <= s0_16(27);

s1_1(28) <= s0_14(28);
s1_2(28) <= s0_15(28);
s1_3(28) <= s0_16(28);

s1_1(29) <= s0_15(29);
s1_2(29) <= s0_16(29);

s1_1(30) <= s0_16(30);

-- Adders in stage1
s41 : HA port map (s1_0(8), s1_1(8), s2_0(8), s2_2(9));

s42 : FA port map (s1_0(9), s1_1(9), s1_2(9), s2_0(9), s2_3(10));
s43 : HA port map (s1_3(9), s1_4(9), s2_1(9), s2_4(10));

s44 : FA port map (s1_0(10), s1_1(10), s1_2(10), s2_0(10), s2_4(11));
s45 : FA port map (s1_3(10), s1_4(10), s1_5(10), s2_1(10), s2_5(11));
s46 : HA port map (s1_6(10), s1_7(10), s2_2(10), s2_6(11));

s47 : FA port map (s1_0(11), s1_1(11), s1_2(11), s2_0(11), s2_4(12));
s48 : FA port map (s1_3(11), s1_4(11), s1_5(11), s2_1(11), s2_5(12));
s49 : FA port map (s1_6(11), s1_7(11), s1_8(11), s2_2(11), s2_6(12));
s50 : HA port map (s1_9(11), s1_10(11), s2_3(11), s2_7(12));

```

```
g50: for i in 12 to 19 generate
s51 : FA port map (s1_0(i), s1_1(i), s1_2(i), s2_0(i), s2_4(i+1));
s52 : FA port map (s1_3(i), s1_4(i), s1_5(i), s2_1(i), s2_5(i+1));
s53 : FA port map (s1_6(i), s1_7(i), s1_8(i), s2_2(i), s2_6(i+1));
s54 : FA port map (s1_9(i), s1_10(i), s1_11(i), s2_3(i), s2_7(i+1));
   end generate g50;

s83 : FA port map (s1_0(20), s1_1(20), s1_2(20), s2_0(20), s2_3(21));
s84 : FA port map (s1_3(20), s1_4(20), s1_5(20), s2_1(20), s2_4(21));
s85 : FA port map (s1_6(20), s1_7(20), s1_8(20), s2_2(20), s2_5(21));
s86 : FA port map (s1_9(20), s1_10(20), s1_11(20), s2_3(20), s2_6(21));

s87 : FA port map (s1_0(21), s1_1(21), s1_2(21), s2_0(21), s2_2(22));
s88 : FA port map (s1_3(21), s1_4(21), s1_5(21), s2_1(21), s2_3(22));
s89 : FA port map (s1_6(21), s1_7(21), s1_8(21), s2_2(21), s2_4(22));

s90 : FA port map (s1_0(22), s1_1(22), s1_2(22), s2_0(22), s2_1(23));
s91 : FA port map (s1_3(22), s1_4(22), s1_5(22), s2_1(22), s2_2(23));

s92 : FA port map (s1_0(23), s1_1(23), s1_2(23), s2_0(23), s2_1(24));


-- stage2 signals assignments
g29_1: for i in 0 to 7 generate
s2_0(i) <= s1_0(i);
s2_1(i) <= s1_1(i);
end generate g29_1;

g30: for i in 1 to 7 generate
s2_2(i) <= s1_2(i);
end generate g30;

g31: for i in 2 to 7 generate
s2_3(i) <= s1_3(i);
end generate g31;

g32: for i in 3 to 7 generate
s2_4(i) <= s1_4(i);
end generate g32;

g33: for i in 4 to 7 generate
s2_5(i) <= s1_5(i);
end generate g33;

g34: for i in 5 to 7 generate
s2_6(i) <= s1_6(i);
end generate g34;

g35: for i in 6 to 7 generate
s2_7(i) <= s1_7(i);
end generate g35;

s2_8(7) <= s1_8(7);

g36: for i in 24 to 31 generate
s2_0(i) <= s1_0(i);
end generate g36;

g37: for i in 25 to 30 generate
s2_1(i) <= s1_1(i);
end generate g37;

```

```
g38: for i in 25 to 29 generate
s2_2(i) <= s1_2(i);
end generate g38;

g39: for i in 25 to 28 generate
s2_3(i) <= s1_3(i);
end generate g39;

g40: for i in 25 to 27 generate
s2_4(i) <= s1_4(i);
end generate g40;

g41: for i in 25 to 26 generate
s2_5(i) <= s1_5(i);
end generate g41;

s2_6(25) <= s1_6(25);

s2_1(8) <= s1_2(8);
s2_2(8) <= s1_3(8);
s2_3(8) <= s1_4(8);
s2_4(8) <= s1_5(8);
s2_5(8) <= s1_6(8);
s2_6(8) <= s1_7(8);
s2_7(8) <= s1_8(8);
s2_8(8) <= s1_9(8);

s2_3(9) <= s1_5(9);
s2_4(9) <= s1_6(9);
s2_5(9) <= s1_7(9);
s2_6(9) <= s1_8(9);
s2_7(9) <= s1_9(9);
s2_8(9) <= s1_10(9);

s2_5(10) <= s1_8(10);
s2_6(10) <= s1_9(10);
s2_7(10) <= s1_10(10);
s2_8(10) <= s1_11(10);

s2_7(11) <= s1_11(11);
s2_8(11) <= s1_12(11);

s2_8(12) <= s1_12(12);
s2_8(13) <= s1_12(13);
s2_8(14) <= s1_12(14);
s2_8(15) <= s1_12(15);
s2_8(16) <= s1_12(16);
s2_8(17) <= s1_12(17);
s2_8(18) <= s1_12(18);
s2_8(19) <= s1_12(19);
s2_8(20) <= s1_12(20);

s2_7(21) <= s1_9(21);
s2_8(21) <= s1_10(21);

s2_5(22) <= s1_6(22);
s2_6(22) <= s1_7(22);
s2_7(22) <= s1_8(22);
s2_8(22) <= s1_9(22);

s2_3(23) <= s1_3(23);
s2_4(23) <= s1_4(23);
```

```
s2_5(23) <= s1_5(23);
s2_6(23) <= s1_6(23);
s2_7(23) <= s1_7(23);
s2_8(23) <= s1_8(23);

s2_2(24) <= s1_1(24);
s2_3(24) <= s1_2(24);
s2_4(24) <= s1_3(24);
s2_5(24) <= s1_4(24);
s2_6(24) <= s1_5(24);
s2_7(24) <= s1_6(24);
s2_8(24) <= s1_7(24);

-- Adders in stage2
s93: HA port map(s2_0(5), s2_1(5), s3_0(5), s3_2(6));

s94: FA port map (s2_0(6), s2_1(6), s2_2(6), s3_0(6), s3_3(7));
s95: HA port map (s2_3(6), s2_4(6), s3_1(6), s3_4(7));

s96: FA port map (s2_0(7), s2_1(7), s2_2(7), s3_0(7), s3_3(8));
s97: FA port map (s2_3(7), s2_4(7), s2_5(7), s3_1(7), s3_4(8));
s98: HA port map (s2_6(7), s2_7(7), s3_2(7), s3_5(8));

g51: for i in 8 to 23 generate
s99 : FA port map (s2_0(i), s2_1(i), s2_2(i), s3_0(i), s3_3(i+1));
s100 : FA port map (s2_3(i), s2_4(i), s2_5(i), s3_1(i), s3_4(i+1));
s101 : FA port map (s2_6(i), s2_7(i), s2_8(i), s3_2(i), s3_5(i+1));
end generate g51;

s147: FA port map (s2_0(24), s2_1(24), s2_2(24), s3_0(24), s3_2(25));
s148: FA port map (s2_3(24), s2_4(24), s2_5(24), s3_1(24), s3_3(25));
s149: FA port map (s2_6(24), s2_7(24), s2_8(24), s3_2(24), s3_4(25));

s150: FA port map (s2_0(25), s2_1(25), s2_2(25), s3_0(25), s3_1(26));
s151: FA port map (s2_3(25), s2_4(25), s2_5(25), s3_1(25), s3_2(26));

s152: FA port map (s2_0(26), s2_1(26), s2_2(26), s3_0(26), s3_1(27));

-- stage3 signals assignments
g42: for i in 0 to 4 generate
s3_0(i) <= s2_0(i);
s3_1(i) <= s2_1(i);
end generate g42;

g43: for i in 1 to 4 generate
s3_2(i) <= s2_2(i);
end generate g43;

g44: for i in 2 to 4 generate
s3_3(i) <= s2_3(i);
end generate g44;

g45: for i in 3 to 4 generate
s3_4(i) <= s2_4(i);
end generate g45;

s3_5(4) <= s2_5(4);

g46: for i in 27 to 31 generate
s3_0(i) <= s2_0(i);
end generate g46;

```

```
g47: for i in 28 to 30 generate
s3_1(i) <= s2_1(i);
end generate g47;

g48: for i in 28 to 29 generate
s3_2(i) <= s2_2(i);
end generate g48;

s3_3(28) <= s2_3(28);

s3_1(5) <= s2_2(5);
s3_2(5) <= s2_3(5);
s3_3(5) <= s2_4(5);
s3_4(5) <= s2_5(5);
s3_5(5) <= s2_6(5);

s3_3(6) <= s2_5(6);
s3_4(6) <= s2_6(6);
s3_5(6) <= s2_7(6);

s3_5(7) <= s2_8(7);

s3_5(25) <= s2_6(25);

s3_3(26) <= s2_3(26);
s3_4(26) <= s2_4(26);
s3_5(26) <= s2_5(26);

s3_2(27) <= s2_1(27);
s3_3(27) <= s2_2(27);
s3_4(27) <= s2_3(27);
s3_5(27) <= s2_4(27);

-- Adders in stage3 starts here
s153: HA port map (s3_0(3), s3_1(3), s4_0(3), s4_2(4));

s154: FA port map (s3_0(4), s3_1(4), s3_2(4), s4_0(4), s4_2(5));
s155: HA port map (s3_3(4), s3_4(4), s4_1(4), s4_3(5));

g49: for i in 5 to 26 generate
s156: FA port map (s3_0(i), s3_1(i), s3_2(i), s4_0(i), s4_2(i+1));
s157: FA port map (s3_3(i), s3_4(i), s3_5(i), s4_1(i), s4_3(i+1));
end generate g49;

s158: FA port map (s3_0(27), s3_1(27), s3_2(27), s4_0(27), s4_1(28));
s159: FA port map (s3_3(27), s3_4(27), s3_5(27), s4_1(27), s4_2(28));

s160: FA port map (s3_0(28), s3_1(28), s3_2(28), s4_0(28), s4_1(29));

-- stage4 signals assignments
g52: for i in 0 to 2 generate
s4_0(i) <= s3_0(i);
s4_1(i) <= s3_1(i);
end generate g52;

g53: for i in 1 to 2 generate
s4_2(i) <= s3_2(i);
end generate g53;

s4_3(2) <= s3_3(2);

s4_1(3) <= s3_2(3);
```

```
s4_2(3) <= s3_3(3);
s4_3(3) <= s3_4(3);

s4_3(4) <= s3_5(4);

s4_3(28) <= s3_3(28);

s4_0(29) <= s3_0(29);
s4_2(29) <= s3_1(29);
s4_3(29) <= s3_2(29);

s4_0(30) <= s3_0(30);
s4_1(30) <= s3_1(30);

s4_0(31) <= s3_0(31);

-- Adders in stage4
 s161: HA port map (s4_0(2),s4_1(2), s5_0(2), s5_1(3));

 g54: for i in 3 to 29 generate
 s162: FA port map (s4_0(i), s4_1(i), s4_2(i), s5_0(i), s5_1(i+1));
 end generate g54;

-- stage5 signals assignments
s5_0(0) <= s4_0(0);
s5_0(1) <= s4_0(1);

s5_1(0) <= s4_1(0);
s5_1(1) <= s4_1(1);
s5_1(2) <= s4_2(2);

s5_2(1) <= s4_2(1);
s5_2(2) <= s4_3(2);

g55: for i in 3 to 29 generate
s5_2(i) <= s4_3(i);
end generate g55;

s5_0(30) <= s4_0(30);
s5_2(30) <= s4_1(30);
s5_0(31) <= s4_0(31);

-- Adders in stage5 starts here
s163: HA port map (s5_0(1),s5_1(1), s6_0(1), s6_1(2));

g56: for i in 2 to 30 generate
s164: FA port map (s5_0(i), s5_1(i), s5_2(i), s6_0(i), s6_1(i+1));
end generate g56;

-- stage6 signals assignments starts here
s6_0(0) <= s5_0(0);
s6_0(31) <= s5_0(31);
s6_1(0) <= s5_1(0);
s6_1(1) <= s5_2(1);

-- Brent Kung adder for final addition for remaining two rows
brent_kung_inst : brent_kung port map (s6_0(31 downto 0), s6_1(31 downto 0),
  cin, result(31 downto 0), result(32));
```

```
end architecture;
```

The 32-bit Brent Kung adder is used for the final addition of the remaining two rows.

```
-- Brent Kung Adder

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity brent_kung 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;

architecture rtl of brent_kung is

component andgate
 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 g1, p1 : std_logic_vector (31 downto 0);
signal g2, p2 : std_logic_vector (15 downto 0);
signal g3, p3 : std_logic_vector (7 downto 0);
signal g4, p4 : std_logic_vector (3 downto 0);
signal g5, p5 : std_logic_vector (1 downto 0);
signal g6, p6 : std_logic;

signal temp_C : std_logic_vector (32 downto 0);
signal temp_s : std_logic_vector (31 downto 0);

begin

-- Finding different order Generate and Propogate values.

-- 1st Order:

order_1 : for i in 0 to 31 generate
  if_gen :if i=0 generate
  G_1_0 : Cin_map_G port map (a(i),b(i), cin, g1(i));
end generate if_gen;
  if_gen2 : if i>0 generate
    G_1 : andgate port map (a(i), b(i), g1(i));

end generate if_gen2;
```

```
P_1 : xorgate port map (a(i), b(i), p1(i));
end generate order_1;

-- 2nd Order:

order_2 : for i in 0 to 15 generate
    G_2 : abcgate port map (g1((2*i)+1), p1((2*i)+1), g1(2*i), g2(i));
    P_2 : andgate port map (p1((2*i)+1), p1(2*i), p2(i));
end generate order_2;


-- 3rd Order:
order_3 : for i in 0 to 7 generate
    G_3 : abcgate port map (g2((2*i)+1), p2((2*i)+1), g2(2*i), g3(i));
    P_3 : andgate port map (p2((2*i)+1), p2(2*i), p3(i));
end generate order_3;


-- 4th Order:
order_4 : for i in 0 to 3 generate
    G_4 : abcgate port map (g3((2*i)+1), p3((2*i)+1), g3(2*i), g4(i));
    P_4 : andgate port map (p3((2*i)+1), p3(2*i), p4(i));
end generate order_4;


--5th Order:
order_5 : for i in 0 to 1 generate
    G_5 : abcgate port map (g4((2*i)+1), p4((2*i)+1), g4((2*i)), g5(i));
    P_5 : andgate port map (p4((2*i)+1), p4((2*i)), p5(i));
end generate order_5;



--6th order:

order_6_G_5 : abcgate port map (g5(1), p5(1), g5(0), g6);
order_6_P_5 : andgate port map (p5(1), p5(0), p6);

-- Generate different Carry

temp_c(0) <= cin;
temp_c(1) <= G1(0);
temp_c(2) <= G2(0);
temp_c_3: abcgate port map (G1(2), P1(2), temp_c(2), temp_c(3));
temp_c(4) <= G3(0);
temp_c_5: abcgate port map (G1(4), P1(4), temp_c(4), temp_c(5));
temp_c_6: abcgate port map (G2(2), P2(2), temp_c(4), temp_c(6));
temp_c_7: abcgate port map (G1(6), P1(6), temp_c(6), temp_c(7));
temp_c(8) <= G4(0);
temp_c_9: abcgate port map (G1(8), P1(8), temp_c(8), temp_c(9));
temp_c_10: abcgate port map (G2(4), P2(4), temp_c(8), temp_c(10));
temp_c_11: abcgate port map (G1(10), P1(10), temp_c(10), temp_c(11));
temp_c_12: abcgate port map (G3(2), P3(2), temp_c(8), temp_c(12));
temp_c_13: abcgate port map (G1(12), P1(12), temp_c(12), temp_c(13));
temp_c_14: abcgate port map (G2(6), P2(6), temp_c(12), temp_c(14));
temp_c_15: abcgate port map (G1(14), P1(14), temp_c(14), temp_c(15));
temp_c(16) <= G5(0);
temp_c_17: abcgate port map (G1(16), P1(16), temp_c(16), temp_c(17));
temp_c_18: abcgate port map (G2(8), P2(8), temp_c(16), temp_c(18));
temp_c_19: abcgate port map (G1(18), P1(18), temp_c(18), temp_c(19));
temp_c_20: abcgate port map (G3(4), P3(4), temp_c(16), temp_c(20));
temp_c_21: abcgate port map (G1(20), P1(20), temp_c(20), temp_c(21));
```

```
temp_c_22: abcgate port map (G2(10), P2(10), temp_c(20), temp_c(22));
temp_c_23: abcgate port map (G1(22), P1(22), temp_c(22), temp_c(23));
temp_c_24: abcgate port map (G4(2), P4(2), temp_c(16), temp_c(24));
temp_c_25: abcgate port map (G1(24), P1(24), temp_c(24), temp_c(25));
temp_c_26: abcgate port map (G2(12), P2(12), temp_c(24), temp_c(26));
temp_c_27: abcgate port map (G1(26), P1(26), temp_c(26), temp_c(27));
temp_c_28: abcgate port map (G3(6), P3(6), temp_c(24), temp_c(28));
temp_c_29: abcgate port map (G1(28), P1(28), temp_c(28), temp_c(29));
temp_c_30: abcgate port map (G2(14), P2(14), temp_c(28), temp_c(30));
temp_c_31: abcgate port map (G1(30), P1(30), temp_c(30), temp_c(31));
temp_c(32) <= G6;


sum_out : for i in 0 to 31 generate
  sum_is : xorgate port map (P1(i), temp_c(i), temp_s(i));
end generate sum_out;

sum <= temp_s;
cout <= temp_c(32);

end architecture rtl;
```

Testbench:

```
library std;
use std.textio.all;

library ieee;
use ieee.std_logic_1164.all;

entity Testbench is
end entity;
architecture Behave of Testbench is


  constant number_of_inputs  : integer := 64;  -- input bits
  constant number_of_outputs : integer := 33;  -- output bits

  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);

  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_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";


    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);


    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');


    variable INPUT_LINE: Line;
    variable OUTPUT_LINE: Line;
    variable LINE_COUNT: integer := 0;


  begin
    while not endfile(INFILE) loop
          LINE_COUNT := LINE_COUNT + 1;

    readLine (INFILE, INPUT_LINE);
          read (INPUT_LINE, input_vector_var);
          read (INPUT_LINE, output_vector_var);
          read (INPUT_LINE, output_mask_var);

          input_vector <= to_std_logic_vector(input_vector_var);

    wait for 10 ns;
          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);

    wait for 4 ns;
    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;
```

The tracefile has three inputs.

- First column is of 64 bits, the first 16 bits are multiplicand, the next 16 bits are multipliers and the next 32 bits are of accumulators.
- The Second column is 33 bits, which is the output known to us. These are the bits we compare the output of the designed MAC.
- The Third column is called masking bits. We enable the bits we want to compare by placing '1' at that place. Since we want all bits to be compared, we use all 1s.

```
0000000000000001000000000000000100000000000000000000000000000000 000000000000000000000000000000001 111111111111111111111111111111111
0000000000000001000000000000000100000000000000000000000000000001 000000000000000000000000000000010 111111111111111111111111111111111
0000000000000001000000000000000100000000000000000000000000000001 000000000000000000000000000000011 111111111111111111111111111111111
0000000000000001000000000000001100000000000000000000000000000000 000000000000000000000000000000011 111111111111111111111111111111111
0000000000000001000000000000001000000000000000000000000000000000 000000000000000000000000000000100 111111111111111111111111111111111
0000000000000001000000000000001000000000000000000000000000000001 000000000000000000000000000000101 111111111111111111111111111111111
0000000000000001000000000000001010000000000000000000000000000000 000000000000000000000000000000101 111111111111111111111111111111111
0000000000000001000000000000001100000000000000000000000000000000 000000000000000000000000000000110 111111111111111111111111111111111
0000000000000001000000000000001110000000000000000000000000000000 000000000000000000000000000000111 111111111111111111111111111111111
0000000000000001000000000000001000000000000000000000000000000000 000000000000000000000000000001000 111111111111111111111111111111111
```

DUT being tested:

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

entity DUT is
   port(input_vector: in std_logic_vector(63 downto 0);
        output_vector: out std_logic_vector(32 downto 0));
end entity;

architecture DutWrap of DUT is

  component MAC_32 is
       port(a, b : in std_logic_vector(15 downto 0);
      acc : in std_logic_vector(31 downto 0);
      result : out std_logic_vector(32 downto 0));
end component MAC_32;
begin

  MAC_instance: MAC_32 port map (a => input_vector(63 downto 48),b => input_vector(47 downto 32),
                                 acc => input_vector(31 downto 0), result => output_vector(32 downto 0));
end DutWrap;
```

Result of the assert flags:

```
Transcript
# Loading ieee.numeric_std(body)
# Loading work.mac_32(a1)
# Loading work.andgate(trivial)
# Loading work.ha(trivial)
# Loading work.xorgate(trivial)
# Loading work.fa(trivial)
# Loading work.cin_map_g(trivial)
# Loading work.brent_kung(rtl)
# Loading work.abcgate(trivial)
# ** Warning: Design size of 16864 statements exceeds ModelSim-Intel FPGA Starter Edition recommended capacity.
# Expect performance to be adversely affected.
#
# add wave *
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#           File in use by: satvi  Hostname: MSI  ProcessID: 28288
#           Attempting to use alternate WLF file "./wlftqv3vi5".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlftqv3vi5
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 140 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```