Meng Wai Chan
Apr 10, 2023

ADD/ SUB LAB REPORT

Meng Wai Chan
Professor Gertner
CSc 343
April 10, 2023

Meng Wai Chan
Apr 10, 2023

Table of Contents

Meng Wai Chan
Apr 10, 2023

I. Objective

The Objective of this lab is to reinforce our knowledge about adders during our lab lecture, and have a deeper understanding of how they function. The first task is to design a half adder using two processes, the second task is to design a 1-bit full adder using half adder as a component. The full adder is then used to design a 4-bit adder, and 4-bit add-sub unit using the full adder by taking in operations code, if operation code = 0 perform addition, else if operation code = 1 perform subtraction. Next we package all the components created for future use. We use the package we created to design a N-bit add-sub unit using a data flow model. The seventh task is to design a N-bit add-sub with overflow, zero, and negative detection using our package components. Then we create a simulation using waveform to confirm our design is functionally using ModelSim, and testing N = 4 bit, and N = 32 bit. Later we designed a test bench to create error detection by intentionally introducing error within our designs.

Meng Wai Chan
Apr 10, 2023

II. Description of Specifications and Functionality

Task 1 - Half Adder using Two Processes.

A half adder adds two binary bits input, "A" and "B", and two outputs, "sum" and "carry", the "sum" output is the result of adding both "A" and "B", and "carry" output is an indicator for if there is a carry-over to the next bit.

		Carry	Sum
x	*y*	*c*	
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity chan_mengwai_halfadder is
5      port (
6          a, b: in std_logic;
7          sum, carry: out std_logic
8      );
9  end entity chan_mengwai_halfadder;
10
11 architecture behavioral of chan_mengwai_halfadder is
12     signal s, c: std_logic;
13 begin
14     process(a, b)
15     begin
16     s<='0';
17       if a='0' then
18          if b='1'  then
19             s<='1';
20          end if;
21       elsif a='1' then
22          if b='0' then
23             s<='1';
24          end if;
25       end if;
26
27
28     end process;
29
30     process(a, b)
31     begin
32     c<='0';
33       if a='1' then
34          if b='1' then
35             c <='1';
36          end if;
37       else
38          c <='0';
39       end if;
40     end process;
41     sum <=s;
42     carry<=c;
43
44 end architecture behavioral;
```

Figure 1 - Half Adder

Meng Wai Chan
Apr 10, 2023

Task 2 - Full Adder using half adder as components

A full adder consists of two half adders that take in 3 inputs “A”, “B”, and “Cin”. The full adder will output “sum” and “carry”. It is used to add two binary bits along with “cin” from previous addition.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY chan_mengwai_fulladder IS
    PORT
    (
        cin :  IN  STD_LOGIC;
        a :  IN  STD_LOGIC;
        b :  IN  STD_LOGIC;
        sum :  OUT  STD_LOGIC;
        cout :  OUT  STD_LOGIC
    );
END chan_mengwai_fulladder;

ARCHITECTURE structure OF chan_mengwai_fulladder IS

COMPONENT chan_mengwai_halfadder
    PORT(a : IN STD_LOGIC;
         b : IN STD_LOGIC;
         sum : OUT STD_LOGIC;
         carry : OUT STD_LOGIC
    );
END COMPONENT;

SIGNAL    s0 :  STD_LOGIC;
SIGNAL    s1 :  STD_LOGIC;
SIGNAL    c0 :  STD_LOGIC;

BEGIN

HA0 : chan_mengwai_halfadder
PORT MAP(a => cin,
         b => s0,
         sum => sum,
         carry => c0);

HA1 : chan_mengwai_halfadder
PORT MAP(a => a,
         b => b,
         sum => s0,
         carry => s1);

cout <= s1 OR c0;

END structure;
```

Figure 2 - Full Adder

Meng Wai Chan
Apr 10, 2023

Task 3 - 4-bit adder using Full Adder

A 4-bit adder adds two 4-bit binary numbers together and produces a 4-bit binary number, it is composed of 4 full adders, each full adder takes 2-bits as inputs to produce the final sum of 4-bit binary numbers.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY chan_mengwai_4bit_adder IS
    PORT
    (
        a, b :  IN  STD_LOGIC_VECTOR(3 downto 0);
        cin :  IN  STD_LOGIC;
        sum :  OUT STD_LOGIC_VECTOR(3 downto 0);
        carry: OUT STD_LOGIC
    );
END chan_mengwai_4bit_adder;

ARCHITECTURE structure OF chan_mengwai_4bit_adder IS

COMPONENT chan_mengwai_fulladder
    PORT(

          cin :  IN  STD_LOGIC;
        a :  IN  STD_LOGIC;
        b :  IN  STD_LOGIC;
        sum :  OUT  STD_LOGIC;
        cout :  OUT  STD_LOGIC
    );
END COMPONENT;

SIGNAL c1, c2, c3 :  STD_LOGIC;

BEGIN
    Chan_FA1: chan_mengwai_fulladder PORT MAP(
        cin => cin,
        a => a(0),
        b =>b(0),
        sum => sum(0),
        cout => c1
    );

    Chan_FA2: chan_mengwai_fulladder PORT MAP(
        cin => c1,
        a => a(1),
        b =>b(1),
        sum => sum(1),
        cout => c2
    );

    Chan_FA3: chan_mengwai_fulladder PORT MAP(
        cin => c2,
        a => a(2),
        b =>b(2),
        sum => sum(2),
        cout => c3
    );

    Chan_FA4: chan_mengwai_fulladder PORT MAP(
        cin => c3,
        a => a(3),
        b =>b(3),
        sum => sum(3),
        cout => carry
    );

END structure;
```

Meng Wai Chan
Apr 10, 2023

Task 4 - 4-bit Add-Sub Component

This 4-bit add-sub component is able to take an opcode to decide if it is subtracting in opcode = 1, and adding if opcode = 0. This component also consists of 4 full adders as components.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY chan_mengwai_4bit_add_sub IS
    PORT
    (
         a,b :  IN  STD_LOGIC_VECTOR(3 downto 0);
         opCode: IN STD_LOGIC;
        sum :  OUT STD_LOGIC_VECTOR(3 downto 0);
         carry, overflow: OUT STD_LOGIC
    );
END chan_mengwai_4bit_add_sub;

ARCHITECTURE structure OF chan_mengwai_4bit_add_sub IS

COMPONENT chan_mengwai_fulladder
    PORT(
       cin :  IN  STD_LOGIC;
       a :  IN  STD_LOGIC;
       b :  IN  STD_LOGIC;
       sum :  OUT  STD_LOGIC;
       cout :  OUT  STD_LOGIC
    );
END COMPONENT;

SIGNAL c1, c2, c3, c4 :  STD_LOGIC;
SIGNAL t :  STD_LOGIC_VECTOR(3 downto 0);

BEGIN

    t <= a XOR b;

    Chan_FA1: chan_mengwai_fulladder PORT MAP(
        cin => opCode,
        a => a(0),
        b =>t(0),
        sum => sum(0),
        cout => c1
    );

    Chan_FA2: chan_mengwai_fulladder PORT MAP(
        cin => c1,
        a => a(1),
        b =>t(1),
        sum => sum(1),
        cout => c2
    );

    Chan_FA3: chan_mengwai_fulladder PORT MAP(
        cin => c2,
        a => a(2),
        b =>t(2),
        sum => sum(2),
        cout => c3
    );

    Chan_FA4: chan_mengwai_fulladder PORT MAP(
        cin => c3,
        a => a(3),
        b =>t(3),
        sum => sum(3),
        cout => c4
    );

    overflow <= c3 xor c4;
    carry <= c4;

END structure;
```

Meng Wai Chan
Apr 10, 2023

Task 5 - Component Package

This package is used to store all components from above, for example half adder, full adder, 4-bit adder, and 4-bit add-sub component is stored for future use.

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

package chan_mengwai_package IS
    component chan_mengwai_halfadder is
        port (
            a, b: in std_logic;
            sum, carry: out std_logic
        );
    end component;

    component chan_mengwai_fulladder IS
        port
        (
            cin :  IN  STD_LOGIC;
            a :  IN  STD_LOGIC;
            b :  IN  STD_LOGIC;
            sum :  OUT  STD_LOGIC;
            cout :  OUT  STD_LOGIC
        );
    end component;

    component chan_mengwai_4bit_adder IS
        port
        (
            a,b :  IN  STD_LOGIC_VECTOR(3 downto 0);
            cin :  IN  STD_LOGIC;
            sum :  OUT STD_LOGIC_VECTOR(3 downto 0);
            carry: OUT STD_LOGIC
        );
    end component;

    component chan_mengwai_4bit_add_sub IS
        port
        (
            a,b :  IN  STD_LOGIC_VECTOR(3 downto 0);
            opCode: IN STD_LOGIC;
            sum :  OUT STD_LOGIC_VECTOR(3 downto 0);
            carry, overflow: OUT STD_LOGIC
        );
    end component;

end chan_mengwai_package;
```

Meng Wai Chan
Apr 10, 2023

Task 6 - N-bit Add-Sub

N-bit add-sub unit is designed using data flow modeling, this unit is able to generate any number of bits, and it consists of full adder from the package.

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

entity chan_mengwai_nbit_add_sub is
    generic (
        N : integer := 8
    );
    port (
        a : in std_logic_vector(N-1 downto 0);
        b : in std_logic_vector(N-1 downto 0);
        cin : in std_logic;
        op: in std_logic;
        sum : out std_logic_vector(N-1 downto 0);
        cout : out std_logic
    );
end chan_mengwai_nbit_add_sub;

architecture dataflow of chan_mengwai_nbit_add_sub is
    signal carry : std_logic_vector(N-1 downto 0);
    signal temp: std_logic_vector(N-1 downto 0);

begin
    temp <= not b when op = '1' else b;

    Chan_FA0: chan_mengwai_fulladder
    PORT MAP
   (
      cin => cin,
      a =>a(0),
      b => temp(0),
      sum => sum(0),
      cout => carry(0)
   );

    Chan_generate: for i in 1 to N-1 generate
        Chan_FA: chan_mengwai_fulladder
        PORT MAP
        (
            cin => carry(i-1),
            a =>a(i),
            b => temp(i),
            sum => sum(i),
            cout => carry(i)
        );

    end generate Chan_generate;

    cout <= carry(N-1);

end dataflow;
```

Meng Wai Chan
Apr 10, 2023

Task 7 - N-Bit Add-Sub with Flags

For this N-bit Add-Sub unit with flag, we implemented using data flow modeling, with overflow, negative, and zero flags. When overflow occurs the overflow flag will be equal to 1, meaning the output is too big and it does not fit, when negative occurs the negative flag will be equal to 1, meaning the result is a negative number. When zero flags equal to 1 it means the result is equal to 0.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.chan_mengwai_package.all;
5
6
7  entity chan_mengwai_nbit_add_sub_overflow is
8      generic (
9          N : integer := 8
10     );
11     port (
12         a : in std_logic_vector(N-1 downto 0);
13         b : in std_logic_vector(N-1 downto 0);
14         cin : in std_logic;
15         op: in std_logic;
16         sum : buffer std_logic_vector(N-1 downto 0);
17         cout : out std_logic;
18         overflow,zero,negative: out std_logic
19     );
20 end chan_mengwai_nbit_add_sub_overflow;
21
22 architecture dataflow of chan_mengwai_nbit_add_sub_overflow is
23     signal carry : std_logic_vector(N-1 downto 0);
24     signal temp: std_logic_vector(N-1 downto 0);
25
26 begin
27     temp <= not b when op = '1' else b;
28     Chan_FA0: chan_mengwai_fulladder
29     PORT MAP
30     (
31        cin => cin,
32        a =>a(0),
33        b => temp(0),
34        sum => sum(0),
35        cout => carry(0)
36     );
37     Chan_generate: for i in 1 to N-1 generate
38         Chan_FA: chan_mengwai_fulladder
39         PORT MAP
40         (
41             cin => carry(i-1),
42             a =>a(i),
43             b => temp(i),
44             sum => sum(i),
45             cout => carry(i)
46         );
47
48     end generate Chan_generate;
49     cout <= carry(N-1);
50     overflow <= (carry(N-1) xor carry(N-2));
51     zero <= '1' when unsigned(sum) = 0 else '0';
52     negative <= '1' when sum(N-1) = '1' else '0';
53 end dataflow;
```

Meng Wai Chan
Apr 10, 2023

Task 9 - LPM Add-Sub

This N-bit LPM Add-Sub unit generated using LPM, is opposite compared to our n-bit add-sub unit, it performs addition when opcode = 0, and subtraction when opcode = 1.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm;
USE lpm.all;

ENTITY chan_mengwai_LPM IS
   generic (
         N : integer := 32
    );
   PORT
   (
      add_sub     : IN STD_LOGIC ;
      cin      : IN STD_LOGIC ;
      dataa    : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
      datab    : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
      cout     : OUT STD_LOGIC ;
      overflow    : OUT STD_LOGIC ;
      result      : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0)
   );
END chan_mengwai_LPM;


ARCHITECTURE SYN OF chan_mengwai_LPM IS

   SIGNAL sub_wire0  : STD_LOGIC ;
   SIGNAL sub_wire1  : STD_LOGIC ;
   SIGNAL sub_wire2  : STD_LOGIC_VECTOR (N-1 DOWNTO 0);

   COMPONENT lpm_add_sub
   GENERIC (
      lpm_direction     : STRING;
      lpm_hint    : STRING;
      lpm_representation      : STRING;
      lpm_type    : STRING;
      lpm_width      : NATURAL
   );
   PORT (
         add_sub  : IN STD_LOGIC ;
         cin   : IN STD_LOGIC ;
         dataa : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
         datab : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
         cout  : OUT STD_LOGIC ;
         overflow : OUT STD_LOGIC ;
         result   : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0)
   );
   END COMPONENT;

BEGIN
   cout    <= sub_wire0;
   overflow    <= sub_wire1;
   result    <= sub_wire2(N-1 DOWNTO 0);

   LPM_ADD_SUB_component : LPM_ADD_SUB
   GENERIC MAP (
      lpm_direction => "UNUSED",
      lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=YES",
      lpm_representation => "SIGNED",
      lpm_type => "LPM_ADD_SUB",
      lpm_width => N
   )
   PORT MAP (
      add_sub => add_sub,
      cin => cin,
      dataa => dataa,
      datab => datab,
      cout => sub_wire0,
      overflow => sub_wire1,
      result => sub_wire2
   );


END SYN;
```

Meng Wai Chan
Apr 10, 2023

III. Simulation

4-Bit Add-Sub and LPM Add-Sub

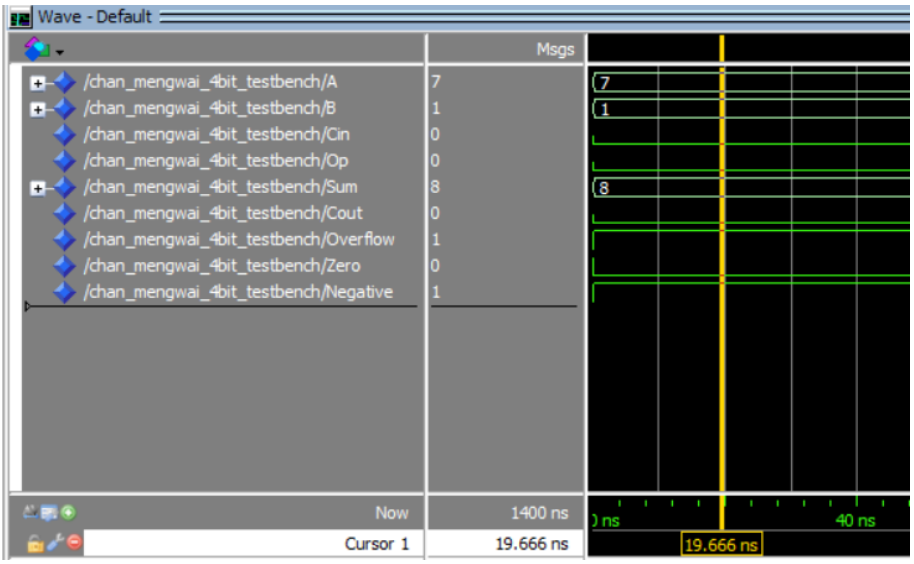
The following are test bench and simulation for 4-bit add-sub and LPM 4-bit add-sub.

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

entity chan_mengwai_4bit_testbench is
end chan_mengwai_4bit_testbench;

architecture testbench_4bit of chan_mengwai_4bit_testbench is

component chan_mengwai_nbit_add_sub_overflow
generic (n: integer := 4);
   port (
        a : in std_logic_vector(N-1 downto 0);
        b : in std_logic_vector(N-1 downto 0);
        cin : in std_logic;
      op: in std_logic;
        sum : out std_logic_vector(N-1 downto 0);
        cout : out std_logic;
      overflow, zero, negative : out std_logic
    );
end component;

    -- Entity inputs
    signal A : std_logic_vector(3 downto 0);
    signal B : std_logic_vector(3 downto 0);
    signal Cin : std_logic;
    signal Op : std_logic;

    -- Entity outputs
    signal Sum : std_logic_vector(3 downto 0) := (others => '0');
    signal Cout : std_logic;
    signal Overflow : std_logic;
    signal Zero : std_logic;
    signal Negative : std_logic;

  begin
     tb_4bit : chan_mengwai_nbit_add_sub_overflow port map(
     a => A, b => B, cin => Cin, op => Op, sum => Sum,
     cout => Cout, overflow => Overflow, zero => Zero, negative => Negative);

process
    begin
        --Most Positive N bit integer + 1
        A <= "0111";
        B <= "0001";
        Cin <= '0';
        Op <= '0';
        wait for 200 ns;

        -- Most Positive N bit integer - 1
        A <= "0111";
        B <= "0001";
        Cin <= '1';
        Op <= '1';
        wait for 200 ns;

        -- Most Negative N bit integer + 1
        A <= "1000";
        B <= "0001";
        Cin <= '0';
        Op <= '0';
        wait for 200 ns;

        -- Most Negative N bit integer - 1
        A <= "1000";
        B <= "0001";
        Cin <= '1';
        Op <= '1';
        wait for 200 ns;
```

```
69
70          -- Most Positive N bit integer - Most Negative N bit integer
71          A <= "0111";
72          B <= "1000";
73          Cin <= '1';
74          Op <= '1';
75          wait for 200 ns;
76
77          -- Most Positive N bit integer + Most Negative N bit integer
78          A <= "0111";
79          B <= "1000";
80          Cin <= '0';
81          Op <= '0';
82          wait for 200 ns;
83
84          -- Most Positive N bit integer - Most Positive N bit integer
85          A <= "0111";
86          B <= "0111";
87          Cin <= '1';
88          Op <= '1';
89          wait for 200 ns;
90
91
92      end process;
93
94  end testbench_4bit;
95
```

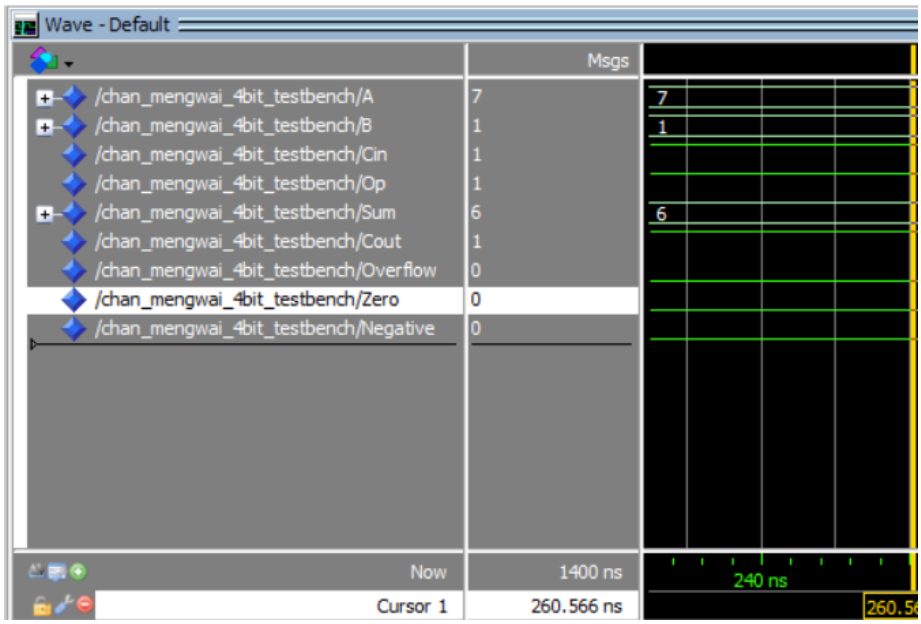
1. Most positive 4-bit integer + 1

The most positive 4-bit integer is 0111 in binary and 7 in HEX and when it adds 0001 in binary, the sum is equal to 1000, this overflow occurs and negative occurs since it needs a single bit in front to store negative sign.



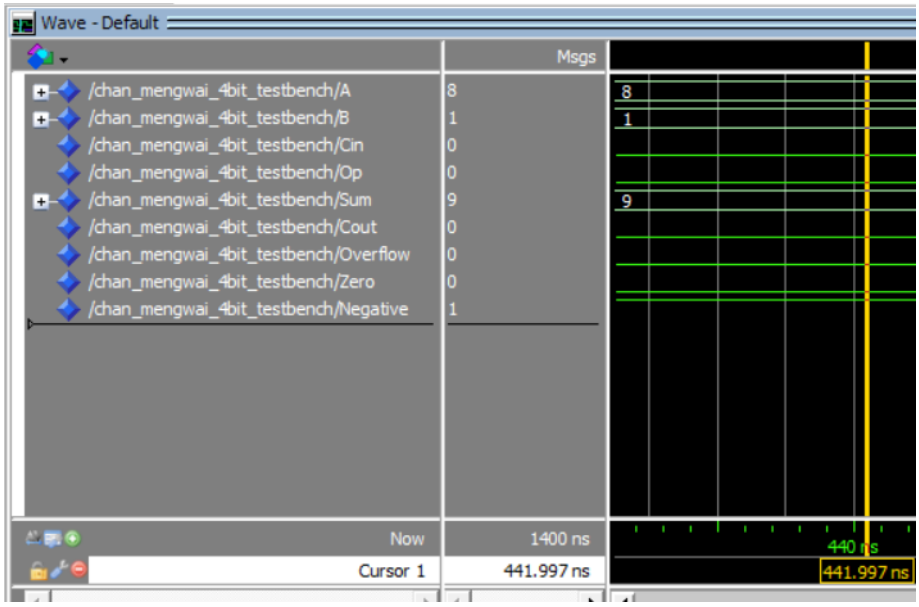
2. Most positive 4-bit integer - 1

   0111 - 0001 = 0110 in binary,since there are enough bits to store the sign and number, overflow does not occur.



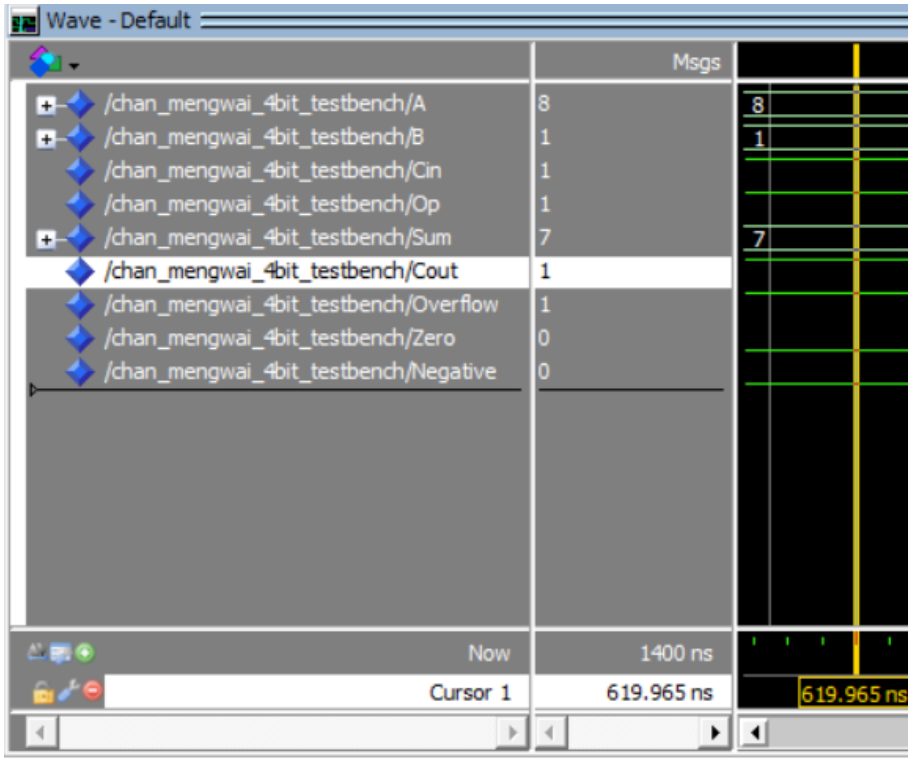
3. Most Negative 4-bit integer + 1

   1000 + 0001 = 1001



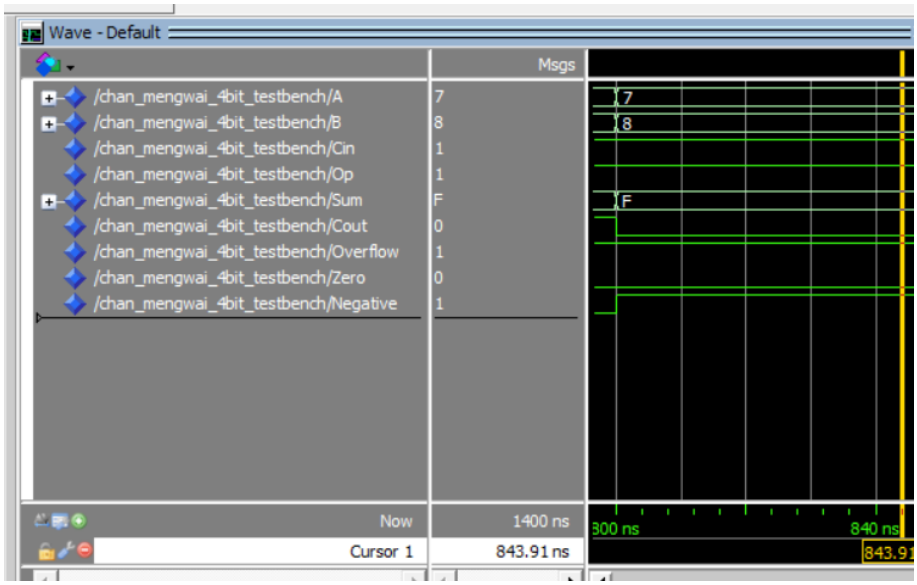
4. Most Negative 4-bit integer - 1

   1000 - 1 = 0111, overflow occurs.



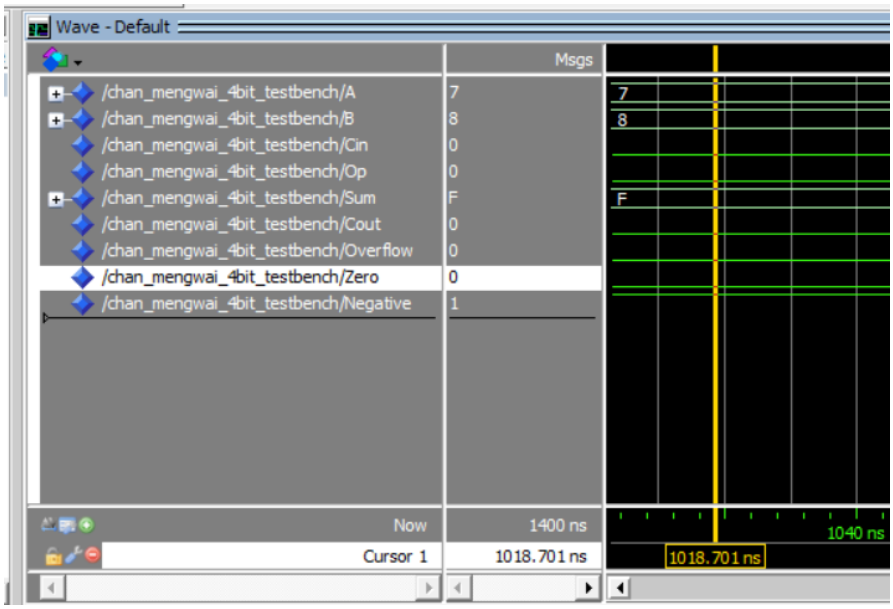
5. Most positive 4-bit integer - most negative 4-bit integer

   0111 - 1000 = 1111, overflow occurs



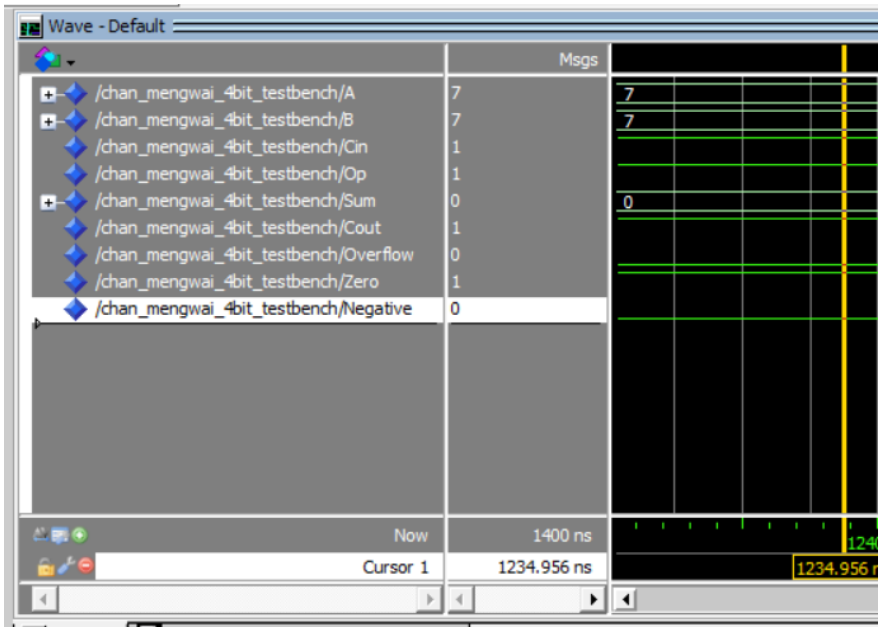
6. Most positive 4-bit integer + most negative 4-bit integer

   0111 + 1000 = 1111



7. Most positive 4-bit integer - most positive 4-bit integer

   0111 - 0111 = 0000, since this is equal to zero the zero flag occurs



Meng Wai Chan
Apr 10, 2023

LPM

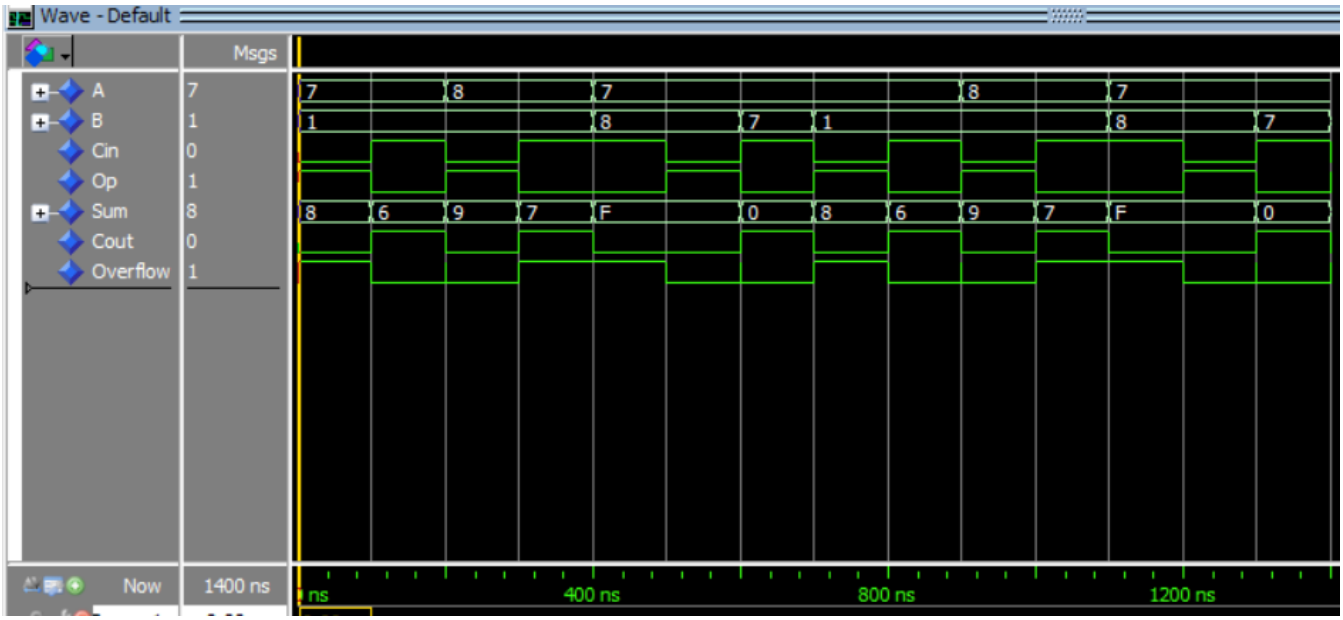
The following is a LPM 4-bit test bench

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity chan_mengwai_LPM_4bit_tb is
5  end chan_mengwai_LPM_4bit_tb;
6
7  architecture testbench_lpm_4bit of chan_mengwai_LPM_4bit_tb is
8
9  component chan_mengwai_LPM
10    generic (n: integer := 4);
11    port (
12       add_sub     : IN STD_LOGIC ;
13       cin      : IN STD_LOGIC ;
14       dataa    : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
15       datab    : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
16       cout     : OUT STD_LOGIC ;
17       overflow    : OUT STD_LOGIC ;
18       result      : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0)
19    );
20 end component;
21
22     -- Entity inputs
23     signal A : std_logic_vector(3 downto 0);
24     signal B : std_logic_vector(3 downto 0);
25     signal Cin : std_logic;
26     signal Op : std_logic;
27
28     -- Entity outputs
29     signal Sum : std_logic_vector(3 downto 0) := (others => '0');
30     signal Cout : std_logic;
31     signal Overflow : std_logic;
32
33    begin
34       tb_4bit : chan_mengwai_LPM port map(
35       add_sub => Op, cin => Cin, dataa=> A, datab=>B,
36       cout => Cout, overflow => Overflow, result => Sum);
37
```

```
38 process
39     begin
40         --Most Positive N bit integer + 1
41         A <= "0111";
42         B <= "0001";
43         Cin <= '0';
44         Op <= '1';
45         wait for 100 ns;
46
47         -- Most Positive N bit integer - 1
48         A <= "0111";
49         B <= "0001";
50         Cin <= '1';
51         Op <= '0';
52         wait for 100 ns;
53
54         -- Most Negative N bit integer + 1
55         A <= "1000";
56         B <= "0001";
57         Cin <= '0';
58         Op <= '1';
59         wait for 100 ns;
60
61         -- Most Negative N bit integer - 1
62         A <= "1000";
63         B <= "0001";
64         Cin <= '1';
65         Op <= '0';
66         wait for 100 ns;
67
68         -- Most Positive N bit integer- Most Negative N bit integer
69         A <= "0111";
70         B <= "1000";
71         Cin <= '1';
72         Op <= '0';
73         wait for 100 ns;
74
```

```
74
75          -- Most Positive N bit integer+ Most Negative N bit integer
76          A <= "0111";
77          B <= "1000";
78          Cin <= '0';
79          Op <= '1';
80          wait for 100 ns;
81
82          -- Most Positive N bit integer- Most Positive N bit integer
83          A <= "0111";
84          B <= "0111";
85          Cin <= '1';
86          Op <= '0';
87          wait for 100 ns;
88
89
90      end process;
91
92  end testbench_lpm_4bit;
93
```

The following is the waveform for LPM, this confirms that our n-bit add-sub unit is correct.



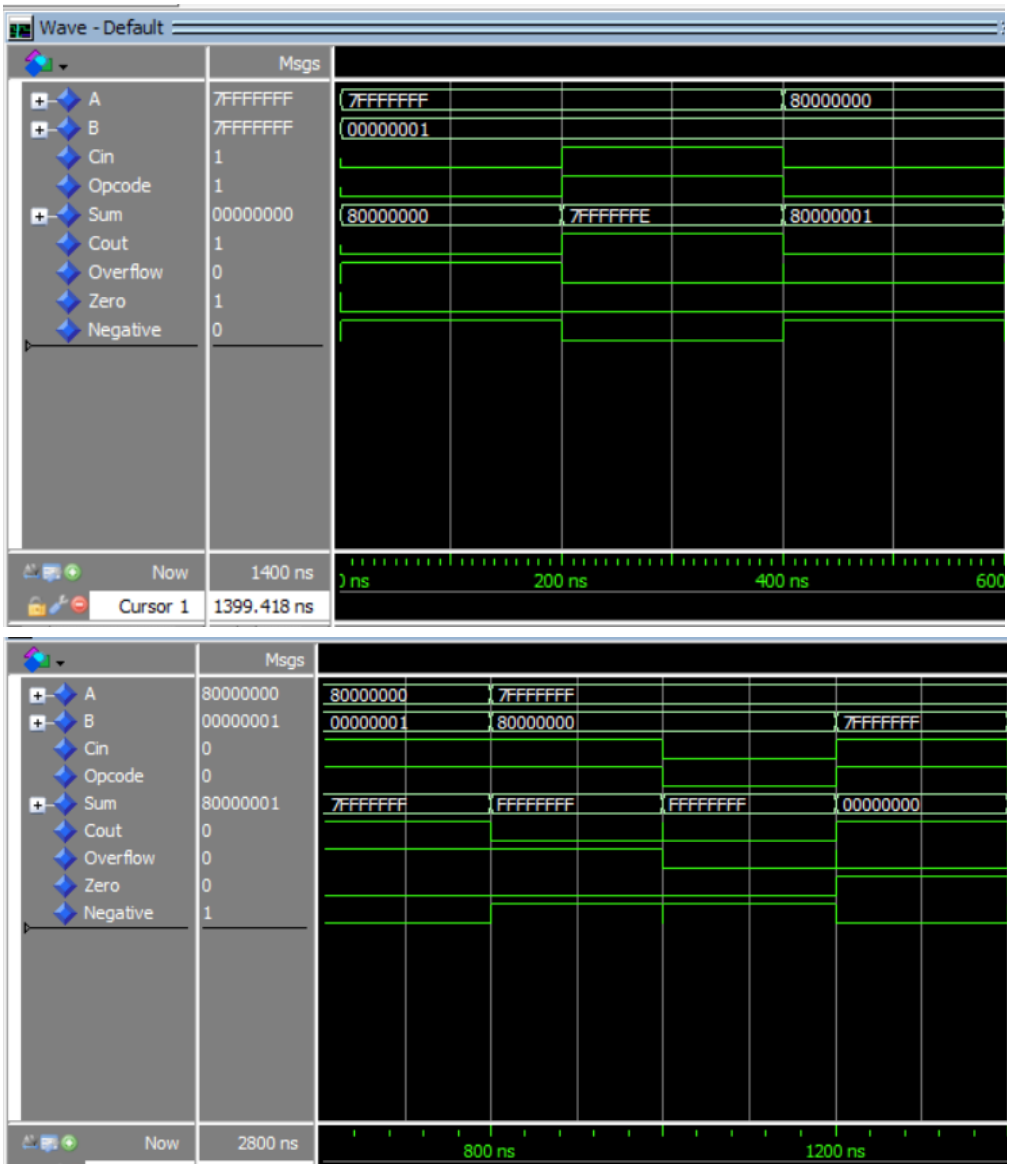
Meng Wai Chan
Apr 10, 2023

32-Bit Add-Sub and LPM Add-Sub

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

entity chan_mengwai_32bit_testbench is
end chan_mengwai_32bit_testbench;

architecture testbench_32bit of chan_mengwai_32bit_testbench is

component chan_mengwai_nbit_add_sub_overflow
generic (n: integer := 32);
   port (
        a : in std_logic_vector(N-1 downto 0);
        b : in std_logic_vector(N-1 downto 0);
        cin : in std_logic;
      op: in std_logic;
        sum : out std_logic_vector(N-1 downto 0);
        cout : out std_logic;
      overflow, zero, negative : out std_logic
    );
end component;

    -- Entity inputs
    signal A : std_logic_vector(31 downto 0);
    signal B : std_logic_vector(31 downto 0);
    signal Cin : std_logic;
    signal Opcode : std_logic;

    -- Entity outputs
    signal Sum : std_logic_vector(31 downto 0);
    signal Cout : std_logic;
    signal Overflow : std_logic;
    signal Zero : std_logic;
    signal Negative : std_logic;

   begin
      tb_32bit : chan_mengwai_nbit_add_sub_overflow port map(
      a => A, b => B, cin => Cin, op => Opcode, sum => Sum,
      cout => Cout, overflow => Overflow, zero => Zero, negative => Negative);

process
    begin
        --Most Positive N bit integer + 1
        A <= "01111111111111111111111111111111";
        B <= "00000000000000000000000000000001";
        Cin <= '0';
        Opcode <= '0';
        wait for 200 ns;

        -- Most Positive N bit integer - 1
        A <= "01111111111111111111111111111111";
        B <= "00000000000000000000000000000001";
        Cin <= '1';
        Opcode <= '1';
        wait for 200 ns;

        -- Most Negative N bit integer + 1
        A <= "10000000000000000000000000000000";
        B <= "00000000000000000000000000000001";
        Cin <= '0';
        Opcode <= '0';
        wait for 200 ns;

        -- Most Negative N bit integer - 1
        A <= "10000000000000000000000000000000";
        B <= "00000000000000000000000000000001";
        Cin <= '1';
        Opcode <= '1';
        wait for 200 ns;

        -- Most Positive N bit integer- Most Negative N bit integer
        A <= "01111111111111111111111111111111";
        B <= "10000000000000000000000000000000";
        Cin <= '1';
        Opcode <= '1';
        wait for 200 ns;

        -- Most Positive N bit integer+ Most Negative N bit integer
        A <= "01111111111111111111111111111111";
        B <= "10000000000000000000000000000000";
        Cin <= '0';
        Opcode <= '0';
        wait for 200 ns;

```

```
84          -- Most Positive N bit integer- Most Positive N bit integer
85          A <= "01111111111111111111111111111111";
86          B <= "01111111111111111111111111111111";
87          Cin <= '1';
88          Opcode <= '1';
89          wait for 200 ns;
90
91      end process;
92
93  end testbench_32bit;
94
```

The following is the simulation for 32-bits add-sub unit test bench, the result is very similar to 4-bit add-sub test bench.



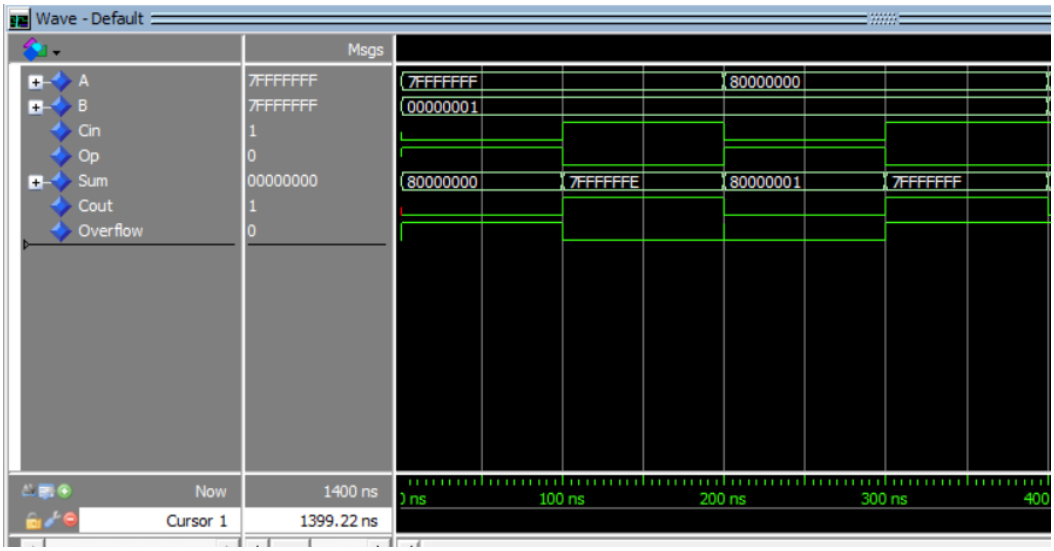
Meng Wai Chan
Apr 10, 2023

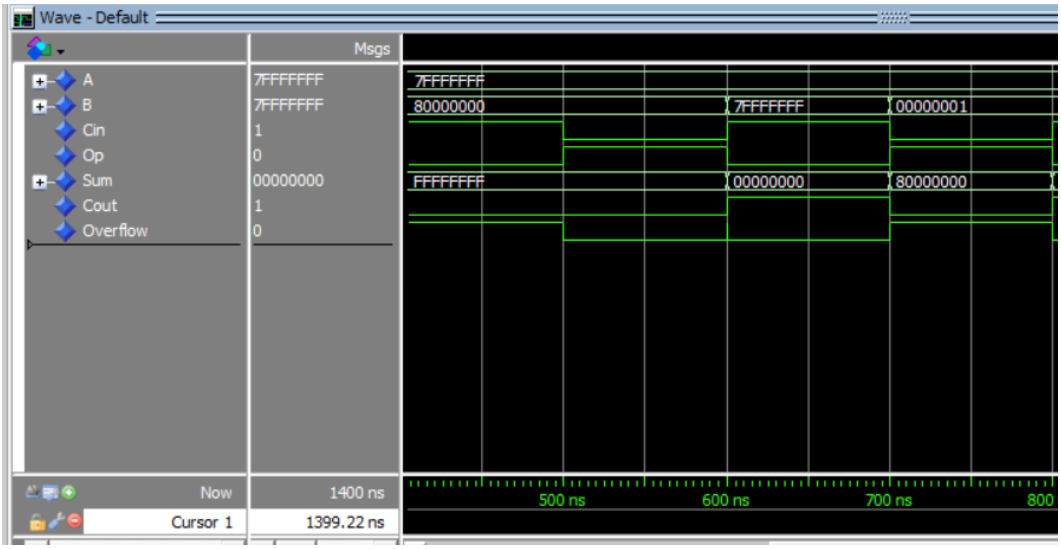
LPM 32-bit

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

entity chan_mengwai_LPM_32bit_tb is
end chan_mengwai_LPM_32bit_tb;

architecture testbench_lpm_32bit of chan_mengwai_LPM_32bit_tb is

component chan_mengwai_LPM
    generic (n: integer := 32);
    port (
        add_sub     : IN STD_LOGIC ;
        cin       : IN STD_LOGIC ;
        dataa     : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
        datab     : IN STD_LOGIC_VECTOR (N-1 DOWNTO 0);
        cout      : OUT STD_LOGIC ;
        overflow    : OUT STD_LOGIC ;
        result      : OUT STD_LOGIC_VECTOR (N-1 DOWNTO 0)
    );
end component;

    -- Entity inputs
    signal A : std_logic_vector(31 downto 0);
    signal B : std_logic_vector(31 downto 0);
    signal Cin : std_logic;
    signal Op : std_logic;

    -- Entity outputs
    signal Sum : std_logic_vector(31 downto 0) := (others => '0');
    signal Cout : std_logic;
    signal Overflow : std_logic;

    begin
        tb_32bit : chan_mengwai_LPM port map(
        add_sub => Op, cin => Cin, dataa=> A, datab=>B,
        cout => Cout, overflow => Overflow, result => Sum);

process
    begin
        --Most Positive N bit integer + 1
        A <= "01111111111111111111111111111111";
        B <= "00000000000000000000000000000001";
        Cin <= '0';
        Op <= '1';
        wait for 100 ns;

        -- Most Positive N bit integer - 1
        A <= "01111111111111111111111111111111";
        B <= "00000000000000000000000000000001";
        Cin <= '1';
        Op <= '0';
        wait for 100 ns;

        -- Most Negative N bit integer + 1
        A <= "10000000000000000000000000000000";
        B <= "00000000000000000000000000000001";
        Cin <= '0';
        Op <= '1';
        wait for 100 ns;

        -- Most Negative N bit integer - 1
        A <= "10000000000000000000000000000000";
        B <= "00000000000000000000000000000001";
        Cin <= '1';
        Op <= '0';
        wait for 100 ns;

        -- Most Positive N bit integer- Most Negative N bit integer
        A <= "01111111111111111111111111111111";
        B <= "10000000000000000000000000000000";
        Cin <= '1';
        Op <= '0';
        wait for 100 ns;
```

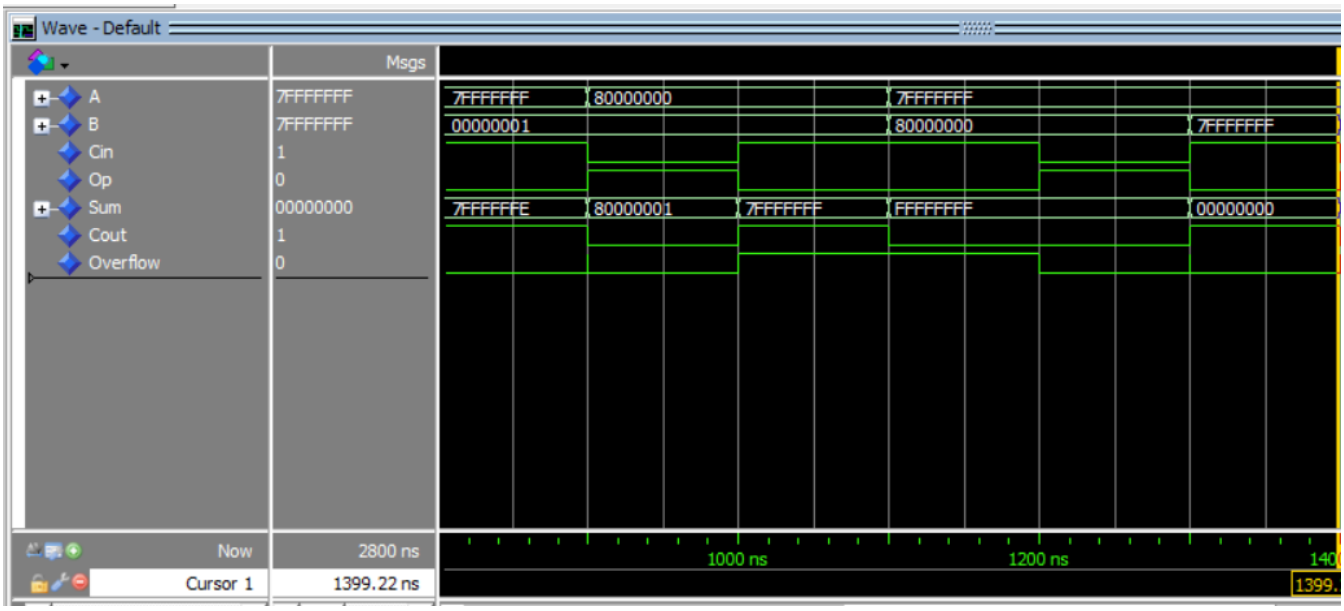
```
75          -- Most Positive N bit integer+ Most Negative N bit integer
76          A <= "01111111111111111111111111111111";
77          B <= "10000000000000000000000000000000";
78          Cin <= '0';
79          Op <= '1';
80          wait for 100 ns;
81
82          -- Most Positive N bit integer- Most Positive N bit integer
83          A <= "01111111111111111111111111111111";
84          B <= "01111111111111111111111111111111";
85          Cin <= '1';
86          Op <= '0';
87          wait for 100 ns;
88
89
90      end process;
91
92  end testbench_lpm_32bit;
93
```

The following is the LPM 32-bits simulation, this confirms that our 32-bits is also correct.





Meng Wai Chan
Apr 10, 2023

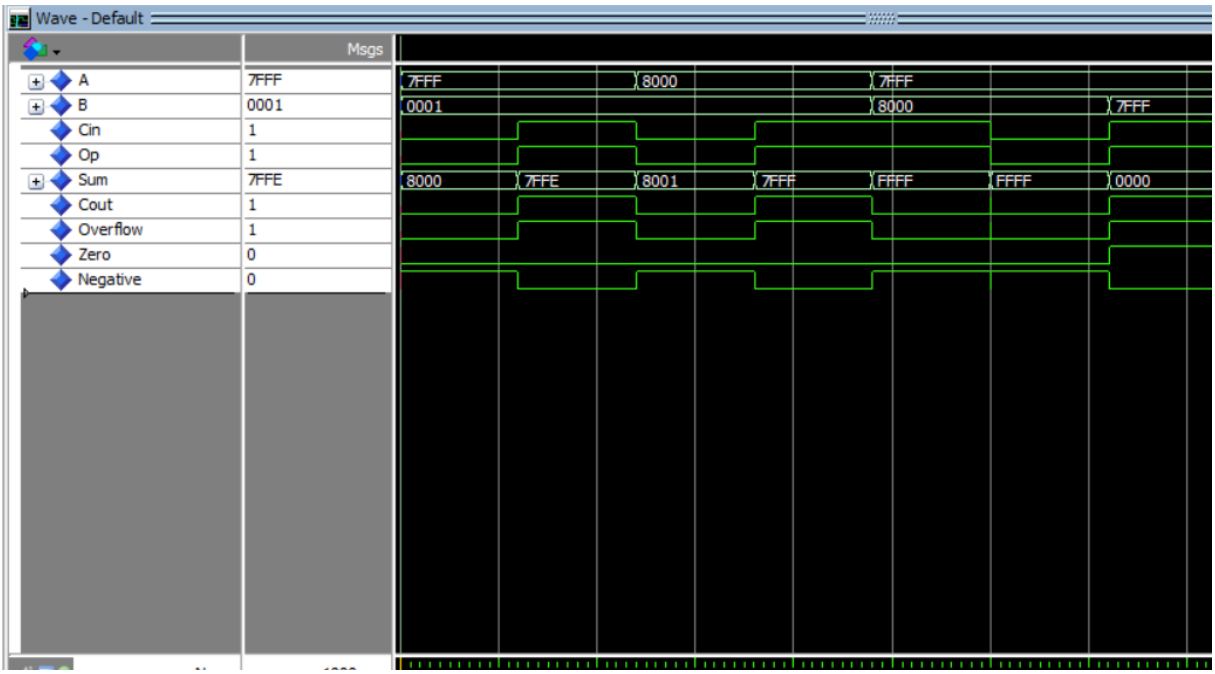


Meng Wai Chan
Apr 10, 2023

Task 10 Waveform

An error was intentionally created in the N-bit add/sub unit using the data flow modeling where the sum is equal to the first input which is not correct.

```
#    Time: 0 ps  Iteration: 0  Instance: /chan_mengwai_16bit_testbench/t4bit
# ** Note: Error for case (a)
#    Time: 20 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Time: 20000 ps
#    Time: 20 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Overflow(expected 1):'0'
#    Time: 20 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Negative(expected 1):'1'
#    Time: 20 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Error for case (b)
#    Time: 140 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Time: 140000 ps
#    Time: 140 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Overflow(expected 0):'1'
#    Time: 140 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Negative(expected 0):'0'
#    Time: 140 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Error for case (e)
#    Time: 500 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Time: 500000 ps
#    Time: 500 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Overflow(expected 1):'0'
#    Time: 500 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Negative(expected 1):'1'
#    Time: 500 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Error for case (g)
#    Time: 740 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Time: 740000 ps
#    Time: 740 ns  Iteration: 0  Instance: /chan_mengwai_16bit_testbench
# ** Note: Overflow(expected 0):'1'
```



IV. Conclusion

I gained an important lesson from this lab, which emphasized the significance of commencing with simpler tasks and gradually progressing towards more intricate ones. The lab showcased this principle by starting with a half-adder and then employing it to construct a 1-bit full adder. Later, the 1-bit full adder was utilized to design a 4-bit adder as well as a 4-bit add/sub unit. Additionally, I acquired knowledge of developing an N-bit add/sub unit utilizing data flow modeling techniques. Lastly, I learned to create testbench files to confirm the accuracy of my designs and detect any potential errors.