Christopher Lall

CSc 342/343 – Lab 3: Adder

Professor Gertner

Due: March 6th, 2022

Table of Contents

Objective:

The goal of this lab is to design half adder, 1-bit full adder, 4-bit adder, 4-bit adder/subtractor, and n-bit adder/subtractor in VHDL. We are then going to verify our output from ModelSim with the respective truth tables for each adder.

Screenshots:

Below will be the screenshots for all required tasks.

Half Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Lall_Christopher_MARCH6_2022_half_adder is

    Port ( Lall_Christopher_MARCH6_2022_inputA : in STD_LOGIC;
           Lall_Christopher_MARCH6_2022_inputB : in STD_LOGIC;
           Lall_Christopher_MARCH6_2022_Sum : out STD_LOGIC;
           Lall_Christopher_MARCH6_2022_Carry: out STD_LOGIC);
end Lall_Christopher_MARCH6_2022_half_adder;

architecture Lall_Christopher_MARCH6_2022_task1 of Lall_Christopher_MARCH6_2022_half_adder is

begin

p1: process(Lall_Christopher_MARCH6_2022_inputA, Lall_Christopher_MARCH6_2022_inputB)
    begin

       Lall_Christopher_MARCH6_2022_Sum<= Lall_Christopher_MARCH6_2022_inputA xor Lall_Christopher_MARCH6_2022_inputB

    end process;

p2: process(Lall_Christopher_MARCH6_2022_inputA, Lall_Christopher_MARCH6_2022_inputB)
    begin

       Lall_Christopher_MARCH6_2022_Carry<=Lall_Christopher_MARCH6_2022_inputA and Lall_Christopher_MARCH6_2022_input
    end process;

end Lall_Christopher_MARCH6_2022_task1;
```

Figure 1. Half adder VHDL code

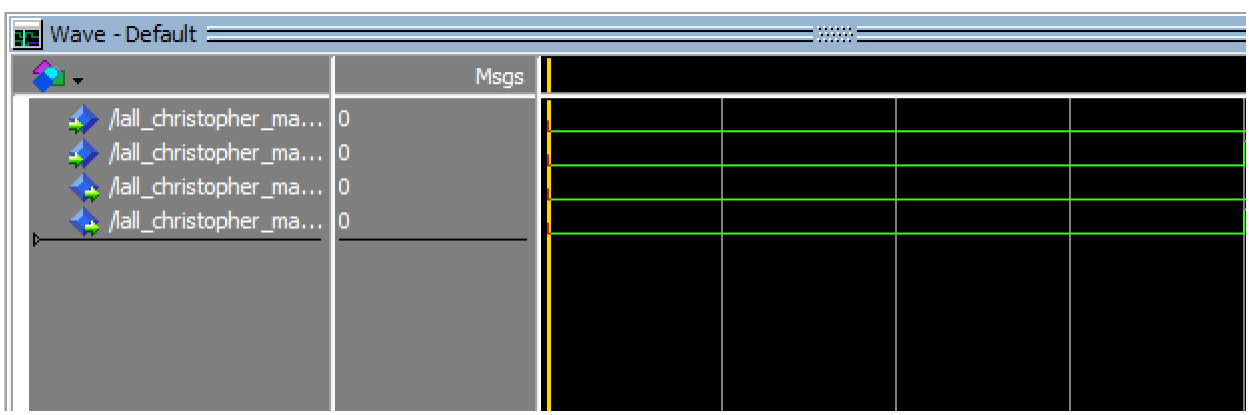

Figure 2. Half adder ouput from both inputs being 0

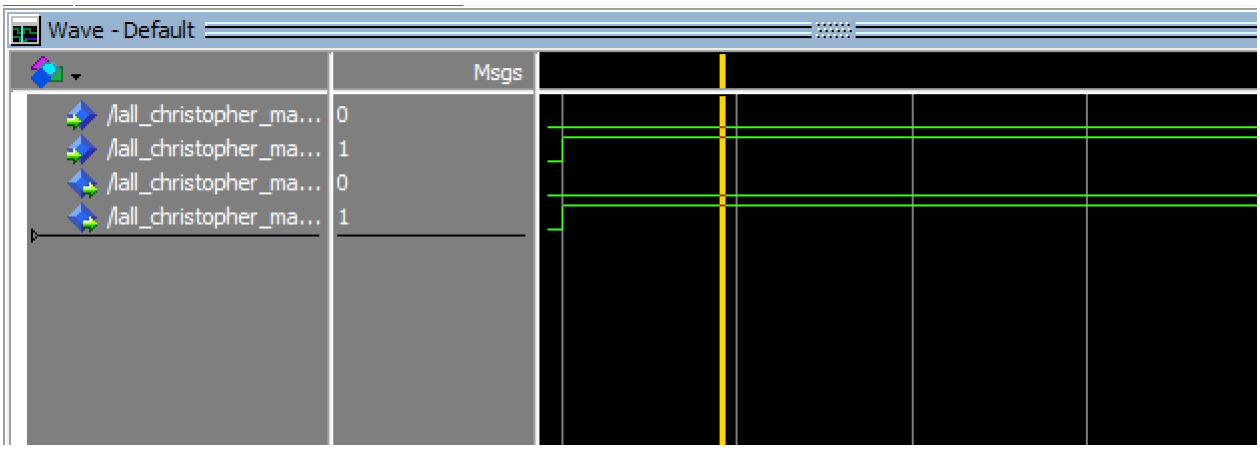

Figure 3. Half adder output where inputs are 0 and 1

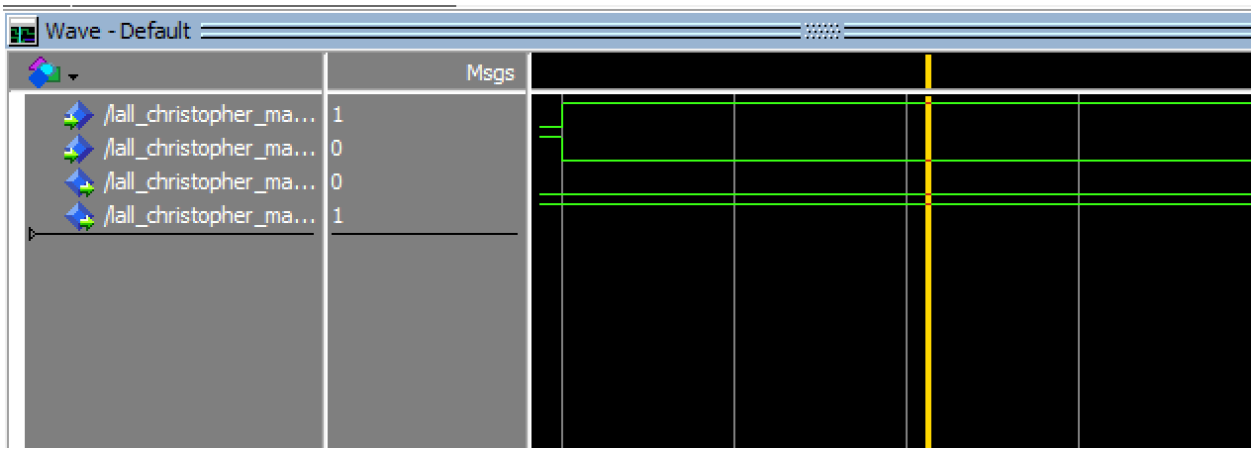

Figure 4. Half adder output where inputs are 1 and 0

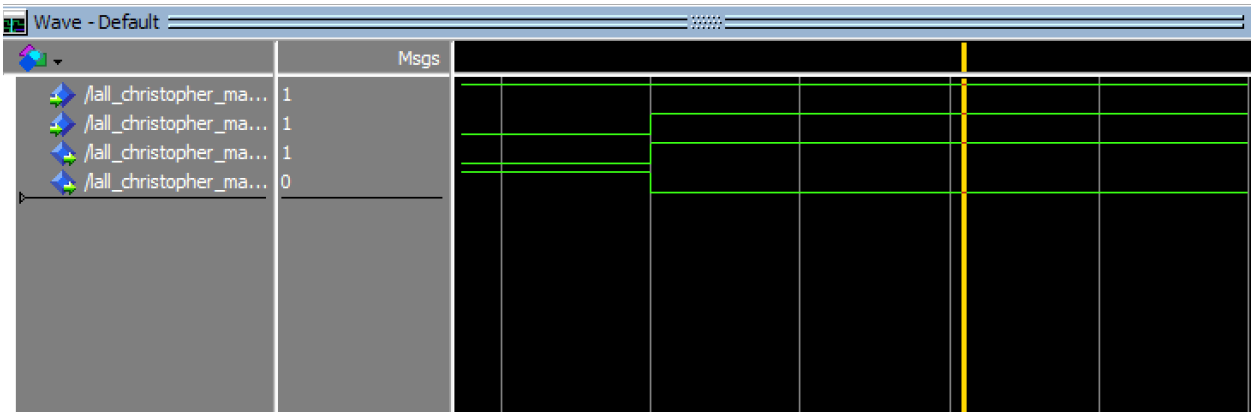

Figure 5. Half adder output where both inputs are 1

Full Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Lall_Christopher_MARCH6_2022_Full_Adder is
   port (Lall_Christopher_MARCH6_2022_inputA, Lall_Christopher_MARCH6_2022_inputB, Cin: in std_logic;
         Lall_Christopher_MARCH6_2022_sum, Cout : out std_logic);
end Lall_Christopher_MARCH6_2022_Full_Adder;

architecture Lall_Christopher_MARCH6_2022_task2 of Lall_Christopher_MARCH6_2022_Full_Adder is

-- temp variables to allow one half adder's results to be pushed into the second one

signal temp1, temp2, temp3 : std_logic;

begin

P1: process (Lall_Christopher_MARCH6_2022_inputA,Lall_Christopher_MARCH6_2022_inputB)
   begin
      temp1 <= Lall_Christopher_MARCH6_2022_inputA xor Lall_Christopher_MARCH6_2022_inputB;
      temp2 <= Lall_Christopher_MARCH6_2022_inputA and Lall_Christopher_MARCH6_2022_inputB;
end process;

P2: process (temp1, temp2, Cin)
   begin

      Lall_Christopher_MARCH6_2022_sum <= temp1 xor Cin;
      temp3 <= temp1 and Cin;
      Cout <= temp2 or temp3;
end process;
end Lall_Christopher_MARCH6_2022_task2;
```

Figure 6. Full adder VHDL code

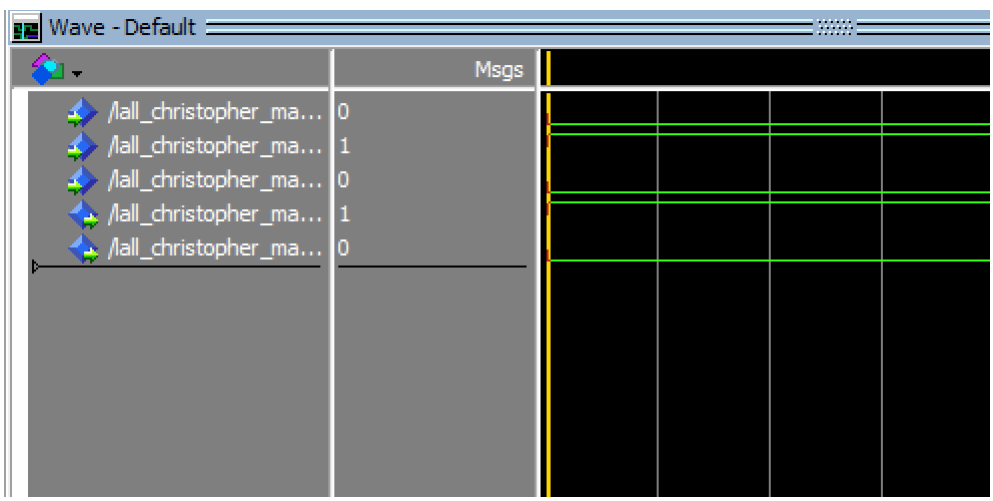

Figure 7. Full adder output where inputs are 0 1 0

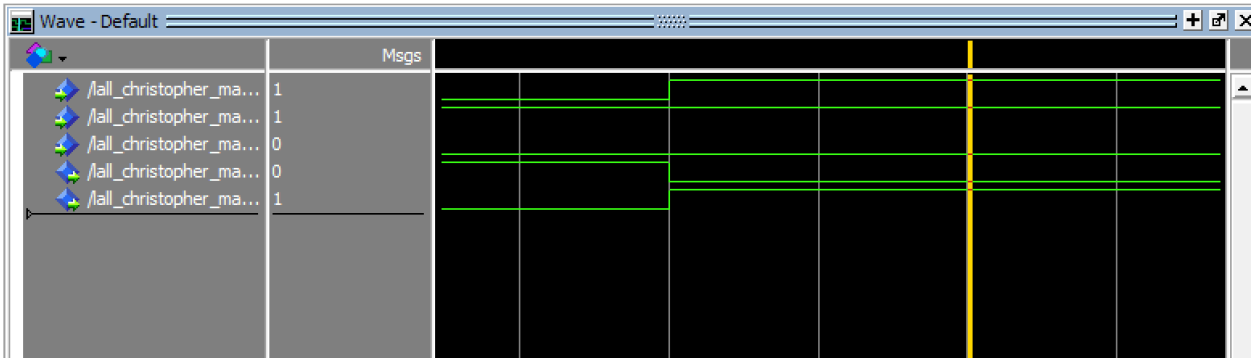

Figure 8. Full adder output where inputs are 1 1 0

Figure 9. Full adder output where input is 1 1 1

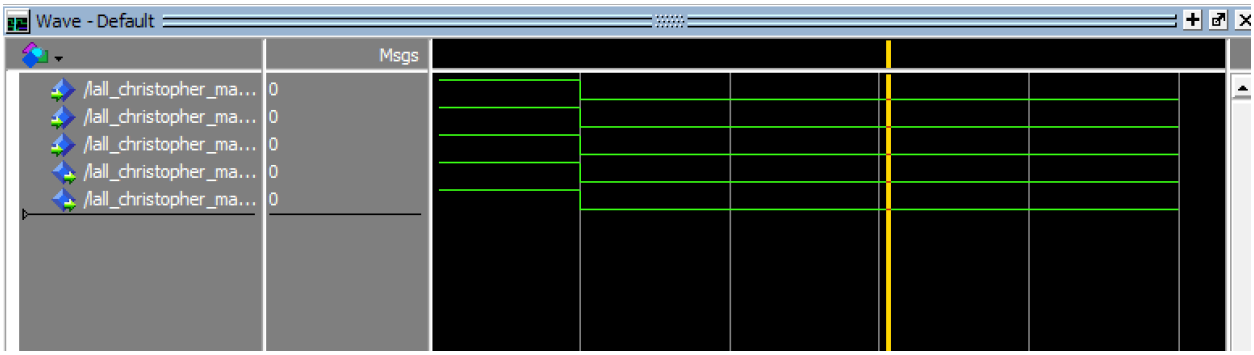

Figure 10. Full adder output where input is 0 0 0

4-bit Adder:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Lall_Christopher_MARCH6_2022_four_bit_Adder is
    Port ( Lall_Christopher_MARCH6_2022_inputA : in STD_LOGIC_VECTOR (3 downto 0);
    Lall_Christopher_MARCH6_2022_inputB : in STD_LOGIC_VECTOR (3 downto 0);
    Cin : in STD_LOGIC;
    Lall_Christopher_MARCH6_2022_sum : out STD_LOGIC_VECTOR (3 downto 0);
    Cout : out STD_LOGIC);
end Lall_Christopher_MARCH6_2022_four_bit_Adder;

architecture Lall_Christopher_MARCH6_2022_task3 of Lall_Christopher_MARCH6_2022_four_bit_adder is

component Lall_Christopher_MARCH6_2022_Full_Adder
    Port ( Lall_Christopher_MARCH6_2022_inputA : in STD_LOGIC;
    Lall_Christopher_MARCH6_2022_inputB : in STD_LOGIC;
    Cin : in STD_LOGIC;
    Lall_Christopher_MARCH6_2022_sum : out STD_LOGIC;
    Cout : out STD_LOGIC);
end component;

signal c1,c2,c3: STD_LOGIC;

begin

FA1: Lall_Christopher_MARCH6_2022_Full_Adder port map( Lall_Christopher_MARCH6_2022_inputA(0), Lall_Christopher_MARC
FA2: Lall_Christopher_MARCH6_2022_Full_Adder port map( Lall_Christopher_MARCH6_2022_inputA(1), Lall_Christopher_MARC
FA3: Lall_Christopher_MARCH6_2022_Full_Adder port map( Lall_Christopher_MARCH6_2022_inputA(2), Lall_Christopher_MARC
FA4: Lall_Christopher_MARCH6_2022_Full_Adder port map( Lall_Christopher_MARCH6_2022_inputA(3), Lall_Christopher_MARC

end Lall_Christopher_MARCH6_2022_task3;
```

Figure 11. 4-bit Adder VHDL code

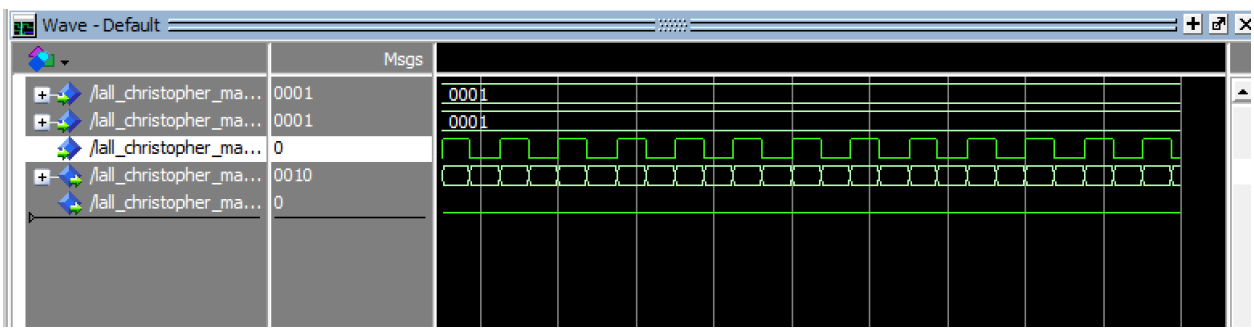

Figure 12. 4-bit Adder output where inputs are 0001 and 0001 with cin as 0

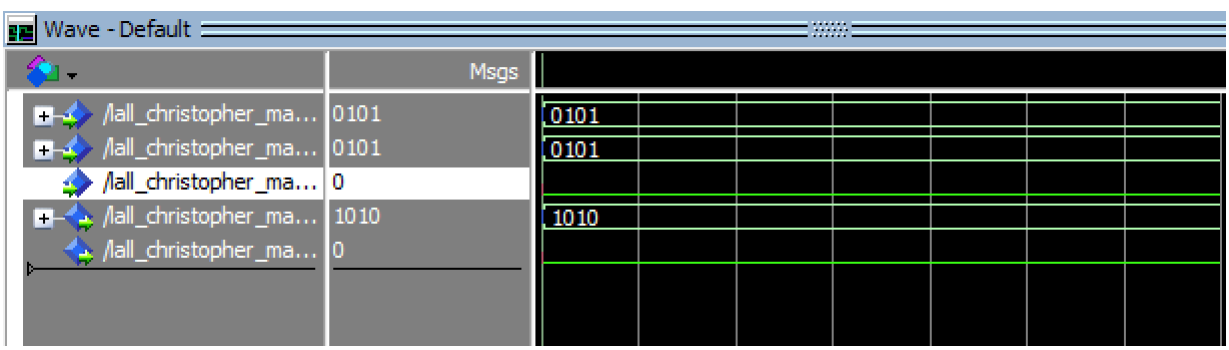

Figure 13. 4-bit Adder output where inputs are 0101 and 0101 with cin as 0

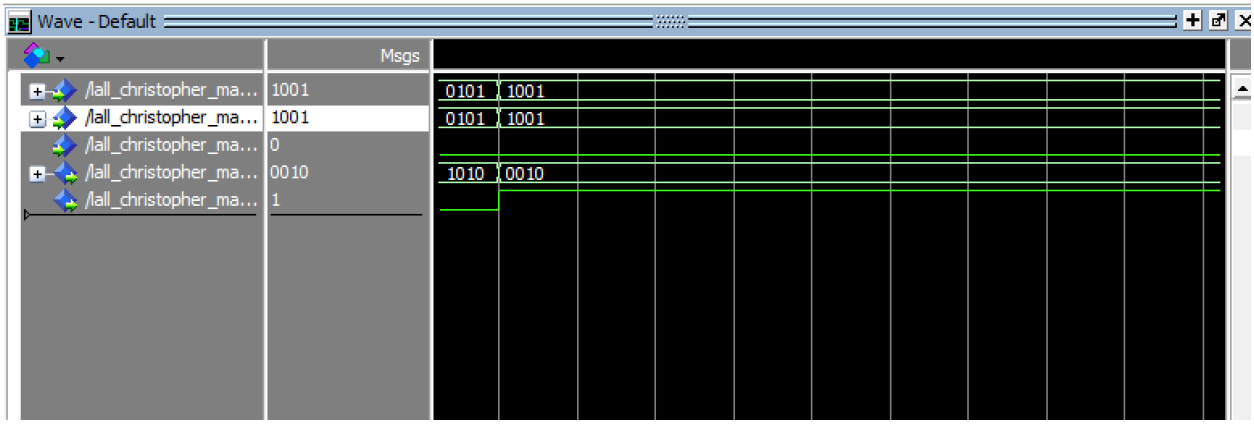

Figure 14. 4-bit Adder output where input is 1001 and 1001 with cin 0

4-bit Adder/Subtractor:

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

entity Lall_Christopher_MARCH6_2022_four_Bit_Adder_subtract is


Port (     Lall_Christopher_MARCH6_2022_inputA : in STD_LOGIC_VECTOR(3 downto 0);
           Lall_Christopher_MARCH6_2022_inputB : in STD_LOGIC_VECTOR(3 downto 0);
           Lall_Christopher_MARCH6_2022_Sum : out STD_LOGIC_VECTOR(3 downto 0);
           Lall_Christopher_MARCH6_2022_Carry: out STD_LOGIC; --carry flag
           Lall_Christopher_MARCH6_2022_Obit: in STD_LOGIC;
           Lall_Christopher_MARCH6_2022_Overflow: out STD_LOGIC);

end Lall_Christopher_MARCH6_2022_four_Bit_Adder_subtract;

architecture Lall_Christopher_MARCH6_2022_task4 of Lall_Christopher_MARCH6_2022_four_Bit_Adder_subtract is

--the carry vectory is c and the adder vector is d

signal c: STD_LOGIC_VECTOR(4 downto 0);

signal d: std_logic_vector(3 downto 0);

begin

d <= Lall_Christopher_MARCH6_2022_inputB xor Lall_Christopher_MARCH6_2022_Obit & Lall_Christopher_MARCH6_2022_Obit &
c(0) <= Lall_Christopher_MARCH6_2022_Obit;
Lall_Christopher_MARCH6_2022_Sum <= Lall_Christopher_MARCH6_2022_inputA xor d xor c(3 downto 0);
c(4 downto 1) <= (Lall_Christopher_MARCH6_2022_inputA and d) or (c(3 downto 0) and (Lall_Christopher_MARCH6_2022_inp
Lall_Christopher_MARCH6_2022_Carry <= c(3);
Lall_Christopher_MARCH6_2022_Overflow <=c(3) xor c(4);



end Lall_Christopher_MARCH6_2022_task4;
```

Figure 15. 4-bit Adder/Subtractor VHDL code

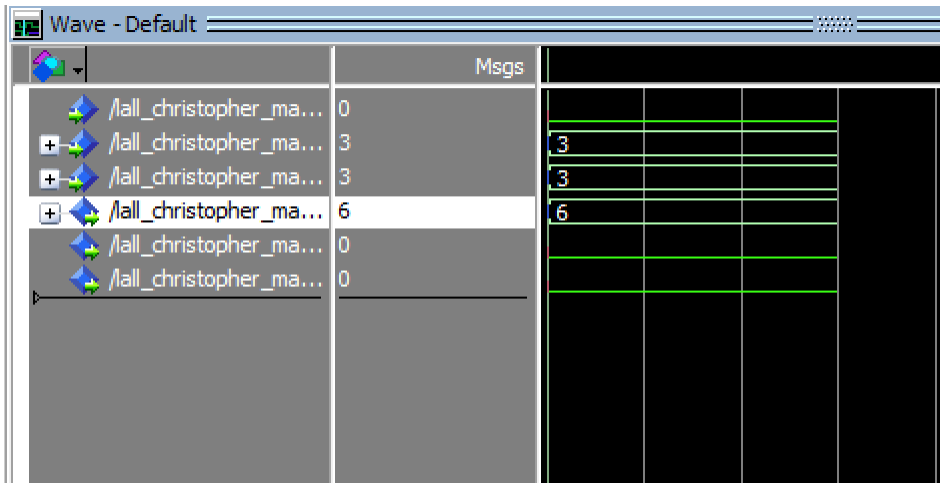

Figure 16. 4-bit Adder/Subtractor adder output where input is 0011(3) and 0011(3)

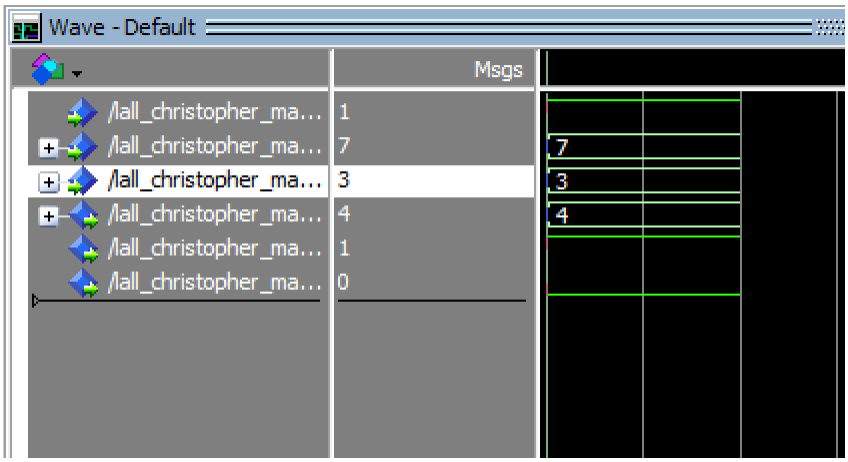

Figure 17. 4-bit Adder/Subtractor subtraction output where input is 0111 (7) and 0011(3)

N-bit adder:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;


ENTITY Lall_Christopher_MARCH6_2022_nBit_Adder_subtract IS
generic(n: natural :=4);
   PORT(Lall_Christopher_MARCH6_2022_A      : IN  std_logic_vector(n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_B      : IN  std_logic_vector(n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_OP : IN  std_logic;
        Lall_Christopher_MARCH6_2022_R     : OUT std_logic_vector (n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_O   : OUT std_logic
   );
END Lall_Christopher_MARCH6_2022_nBit_Adder_subtract;

ARCHITECTURE Lall_Christopher_MARCH6_2022_N_BIT_Adder_subtract OF Lall_Christopher_MARCH6_2022_nBit_Adder_subtract I
    SIGNAL result_temp : std_logic_vector(n-1 DOWNTO 0);
BEGIN
    result_temp <= (Lall_Christopher_MARCH6_2022_A + Lall_Christopher_MARCH6_2022_B) WHEN (Lall_Christopher_MARCH6_2
    Lall_Christopher_MARCH6_2022_O     <= '1' WHEN (Lall_Christopher_MARCH6_2022_OP='0' AND Lall_Christopher_MARCH6_2
                                (Lall_Christopher_MARCH6_2022_OP='1' AND Lall_Christopher_MARCH6_2022_A(n-1)/=Lall_Chris
    Lall_Christopher_MARCH6_2022_R       <= result_temp;
END Lall_Christopher_MARCH6_2022_N_BIT_Adder_subtract;
```

Figure 18. N-bit adder VHDL code

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;


ENTITY Lall_Christopher_MARCH6_2022_nBit_Adder_sub IS
generic(n: natural :=4);
   PORT(Lall_Christopher_MARCH6_2022_A     : IN  std_logic_vector(n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_B     : IN  std_logic_vector(n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_OP : IN  std_logic;
        Lall_Christopher_MARCH6_2022_R     : OUT std_logic_vector (n-1 DOWNTO 0);
        Lall_Christopher_MARCH6_2022_O  : OUT std_logic;
          Lall_Christopher_MARCH6_2022_Z  : OUT std_logic;
          Lall_Christopher_MARCH6_2022_N  : OUT std_logic
   );
END Lall_Christopher_MARCH6_2022_nBit_Adder_sub;

ARCHITECTURE Lall_Christopher_MARCH6_2022_nBit_Adder_sub OF Lall_Christopher_MARCH6_2022_nBit_Adder_sub IS
    SIGNAL result_temp : std_logic_vector(n-1 DOWNTO 0);
BEGIN
    result_temp <= (Lall_Christopher_MARCH6_2022_A + Lall_Christopher_MARCH6_2022_B) WHEN (Lall_Christopher_MARCH6_2
    Lall_Christopher_MARCH6_2022_O    <= '1' WHEN (Lall_Christopher_MARCH6_2022_OP='0' AND Lall_Christopher_MARCH6_2
                          (Lall_Christopher_MARCH6_2022_OP='1' AND Lall_Christopher_MARCH6_2022_A(n-1)/=Lall_Chris
     Lall_Christopher_MARCH6_2022_Z       <= '1' when result_temp = "0000" else '0';
     Lall_Christopher_MARCH6_2022_N       <= result_temp(result_temp'high);
    Lall_Christopher_MARCH6_2022_R      <= result_temp;
END Lall_Christopher_MARCH6_2022_nBit_Adder_sub;
```

Figure 19. n-bit adder with flags VHDL code

3. Waveforms for task 8 in Model-Sim for N=4, and N=32

In waveforms You must use the following operands

a. Most Positive N bit integer + 1
b. Most Positive N bit integer - 1
c. Most Negative N bit integer + 1
d. Most Negative N bit integer – 1
e. Most Positive N bit integer- Most Negative N bit integer
f. Most Positive N bit integer+ Most Negative N bit integer
g. Most Positive N bit integer- Most Positive N bit integer

Figure 20. Waveform requirements

N-bit adder as 4bit:

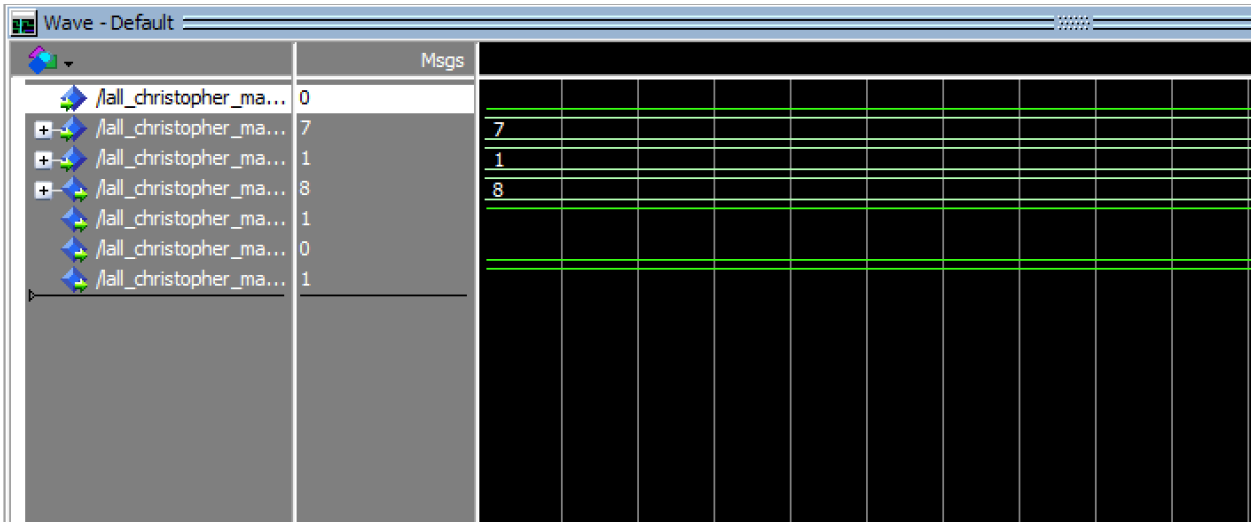

Figure 21. N-bit adder/sub as 4bit. Case 3a waveform

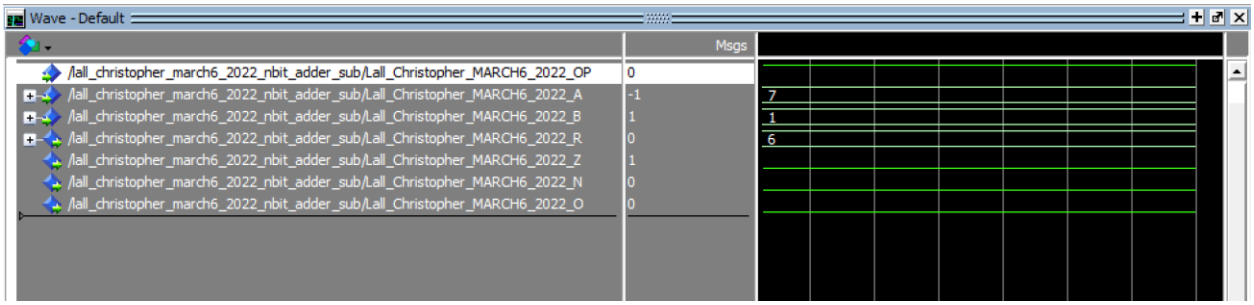

Figure 22. N-bit adder as 4bit. Case 3b waveform

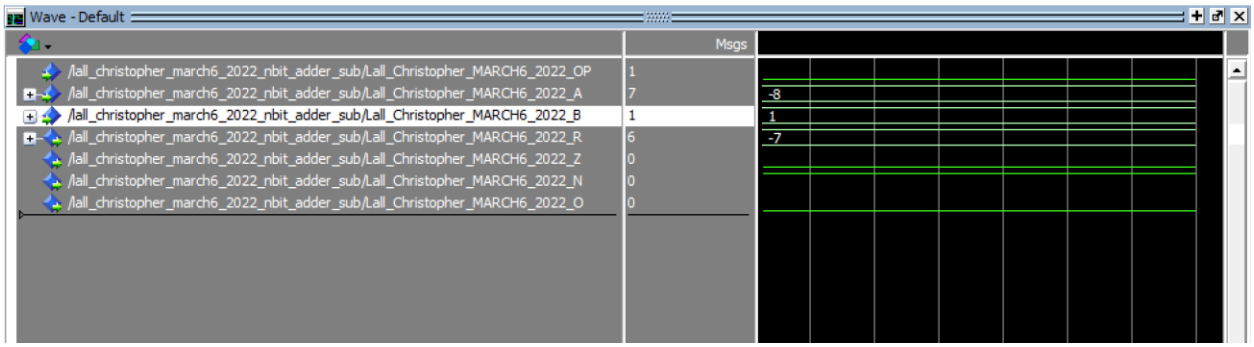

Figure 23. N-bit adder as 4bit. Case 3c waveform

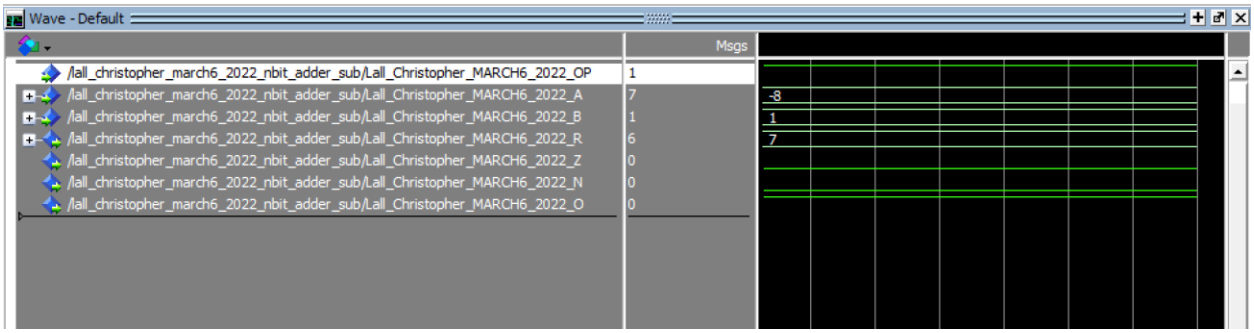

Figure 24. N-bit adder as 4bit. Case 3d waveform

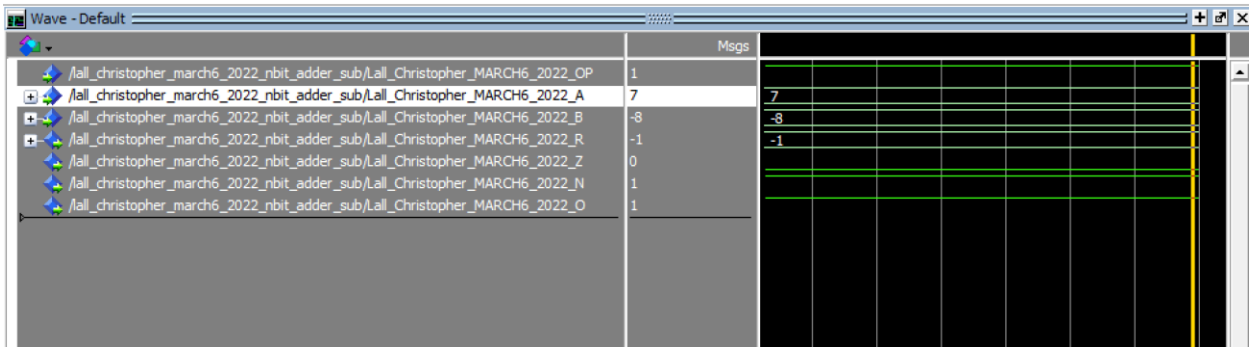

Figure 25. N-bit adder as 4bit. Case 3e waveform

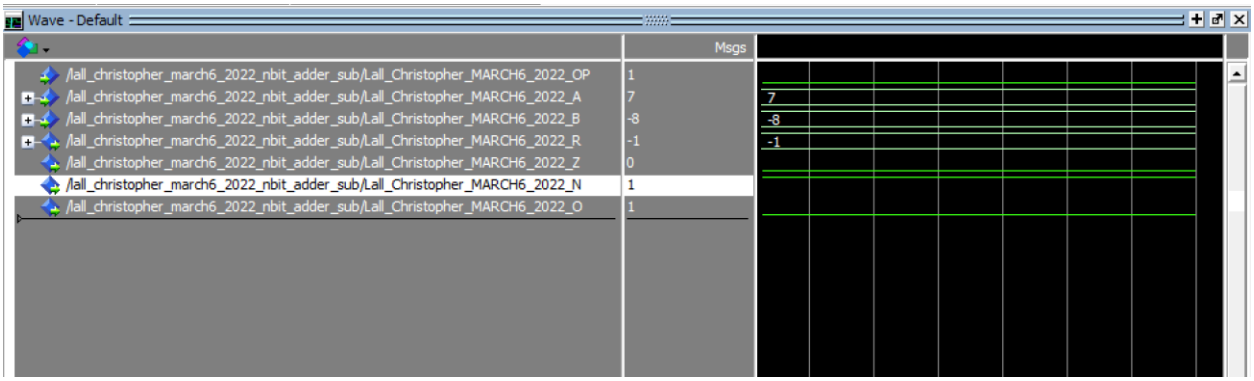

Figure 26. N-bit adder as 4bit. Case 3f waveform

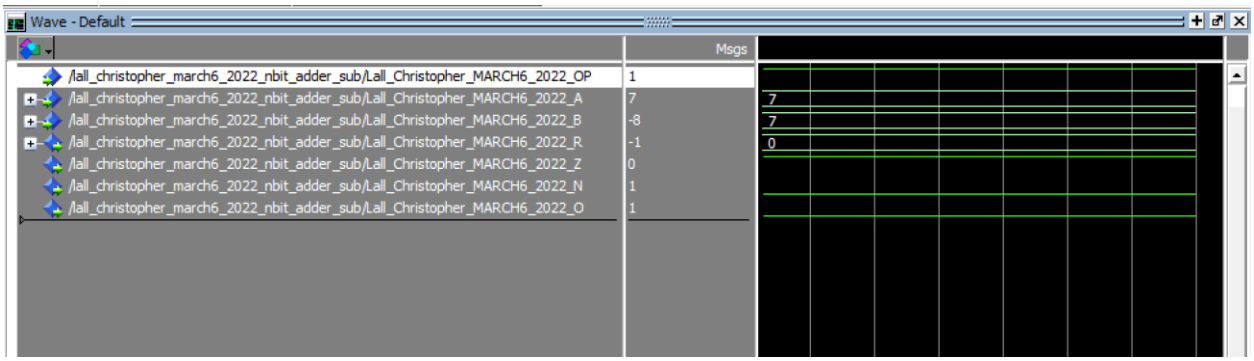

Figure 27. N-bit adder as 4bit. Case 3g waveform

N-bit adder as 32bit:

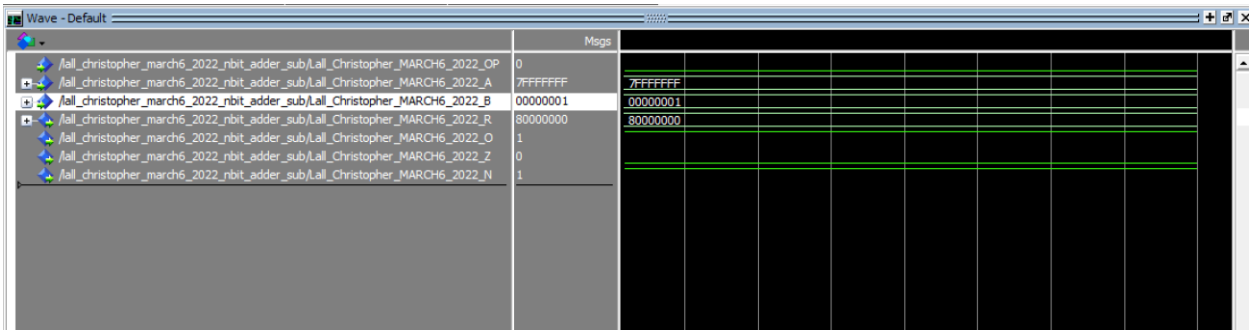

Figure 28. N-bit adder as 32bit. Case 3a waveform

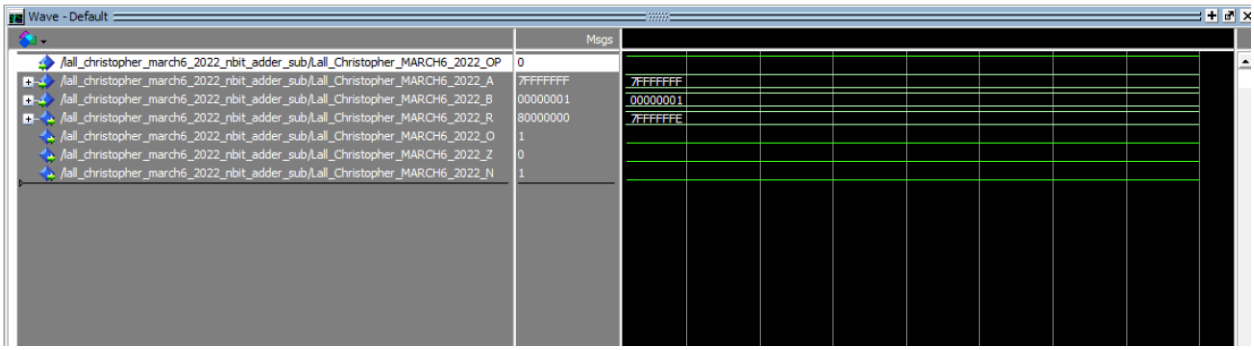

Figure 29. N-bit adder as 32bit. Case 3b waveform

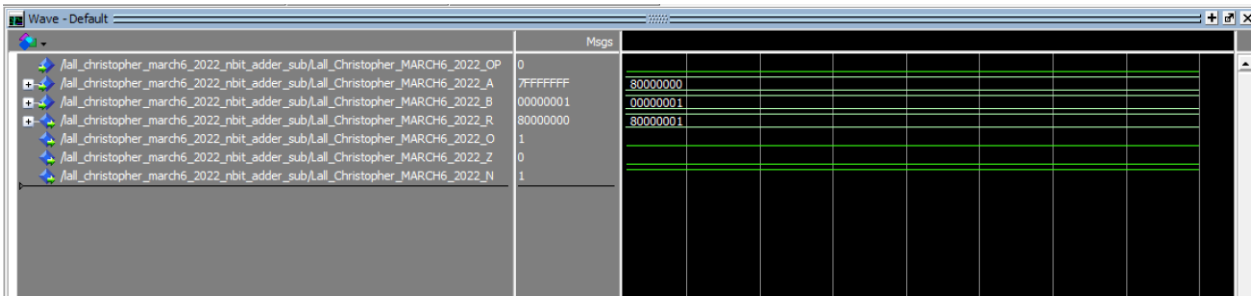

Figure 30. N-bit adder as 32bit. Case 3c waveform

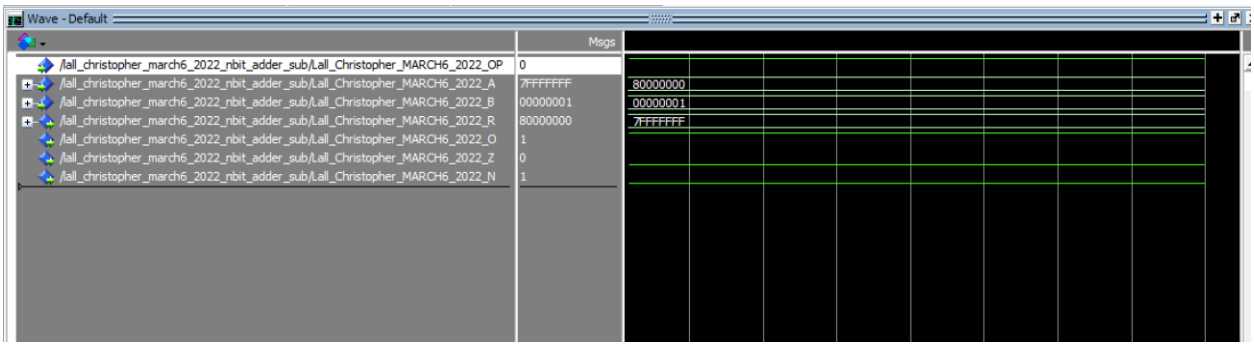

Figure 31. N-bit adder as 32bit. Case 3d waveform

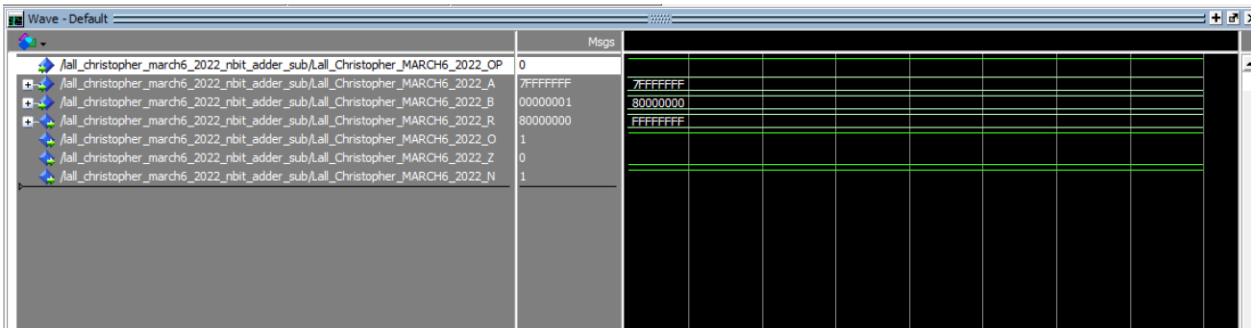

Figure 32. N-bit adder as 32bit. Case 3e waveform

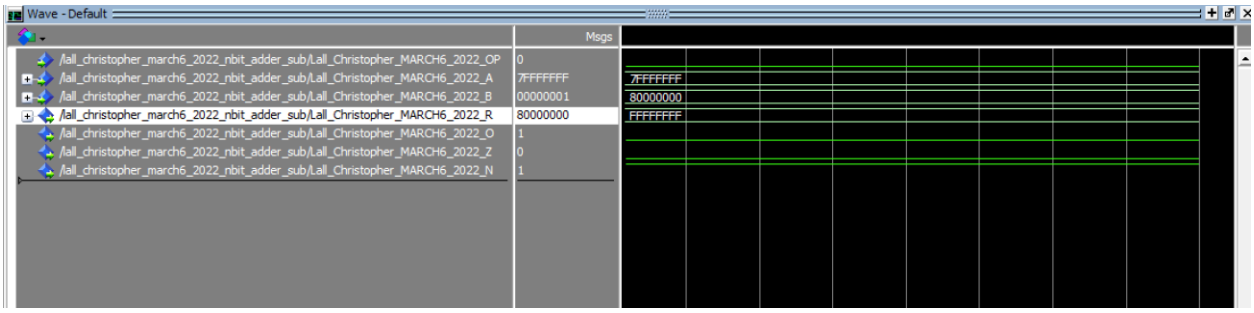

Figure 33. N-bit adder as 32bit. Case 3f waveform

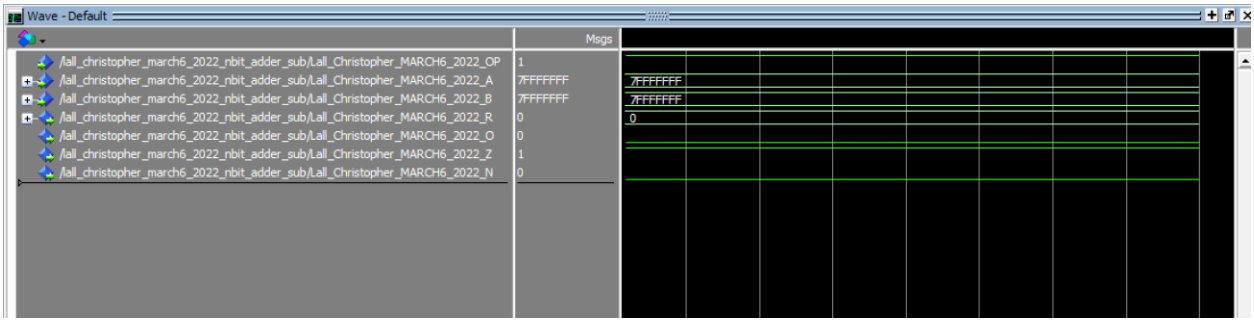

Figure 34. N-bit adder as 32bit. Case 3g waveform

LPM:

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

LIBRARY lpm;
USE lpm.all;

ENTITY Lall_Christopher_MARCH6_2022_lpm IS
   PORT
   (
      add_sub      : IN STD_LOGIC ;
      dataa     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      datab     : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      cout      : OUT STD_LOGIC ;
      overflow     : OUT STD_LOGIC ;
      result       : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );
END Lall_Christopher_MARCH6_2022_lpm;


ARCHITECTURE SYN OF lall_christopher_march6_2022_lpm IS

   SIGNAL sub_wire0  : STD_LOGIC ;
   SIGNAL sub_wire1  : STD_LOGIC ;
   SIGNAL sub_wire2  : STD_LOGIC_VECTOR (7 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 ;
         dataa : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         datab : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         cout  : OUT STD_LOGIC ;
         overflow : OUT STD_LOGIC ;
         result   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
   );
```

```
   END COMPONENT;

BEGIN
   cout     <= sub_wire0;
   overflow     <= sub_wire1;
   result     <= sub_wire2(7 DOWNTO 0);

   LPM_ADD_SUB_component : LPM_ADD_SUB
   GENERIC MAP (
      lpm_direction => "UNUSED",
      lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO",
      lpm_representation => "SIGNED",
      lpm_type => "LPM_ADD_SUB",
      lpm_width => 8
   )
   PORT MAP (
      add_sub => add_sub,
      dataa => dataa,
      datab => datab,
      cout => sub_wire0,
      overflow => sub_wire1,
      result => sub_wire2
   );



END SYN;
```

Figure 35. VHDL Code lpm

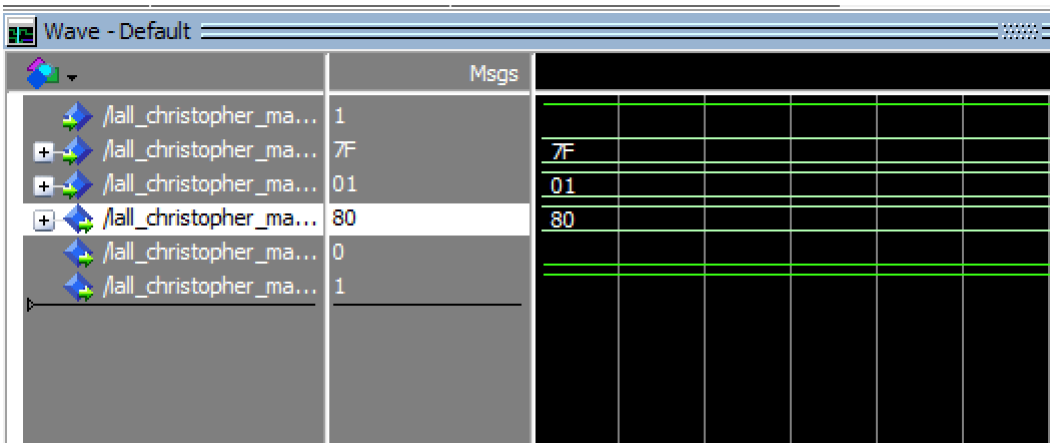

Figure 36. LPM case 3a

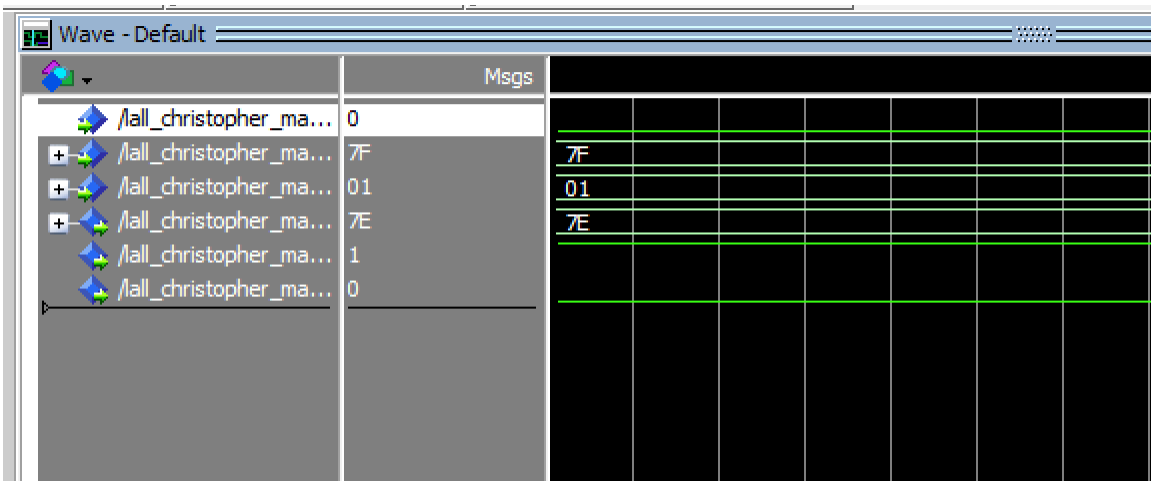

Figure 37. LPM case 3b

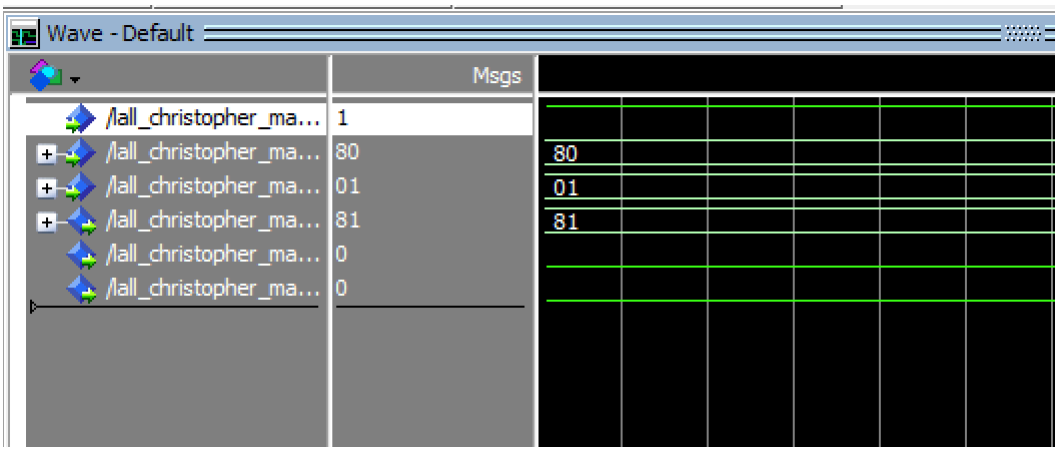

Figure 38. LPM case 3c

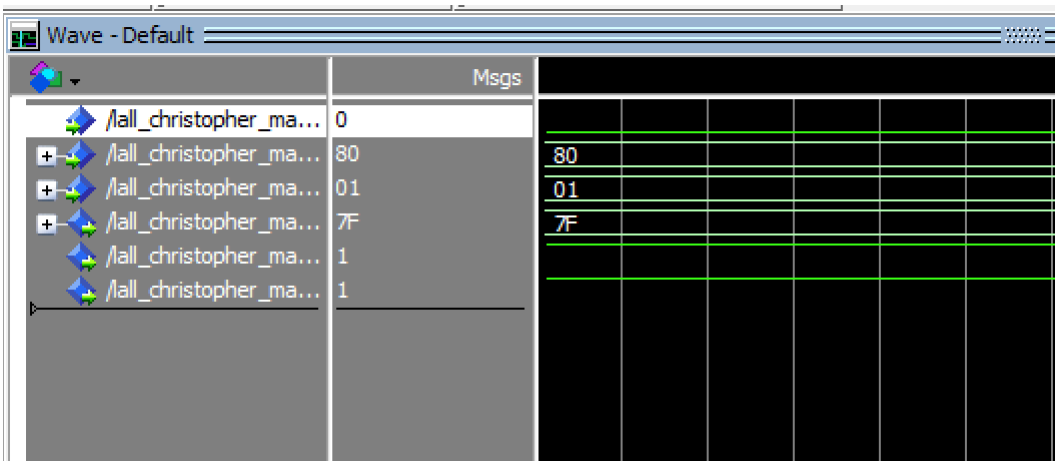

Figure 39. LPM case 3d

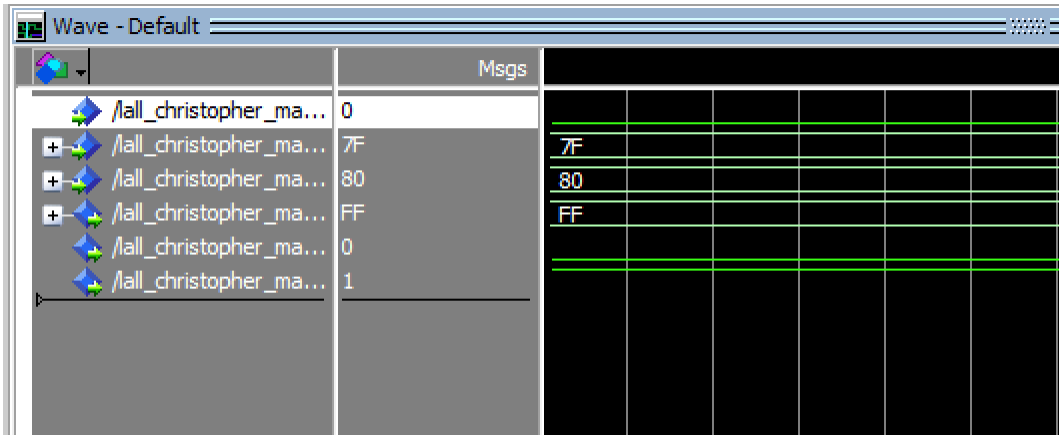

Figure 40. LPM case 3e

Figure 41. LPM case 3f

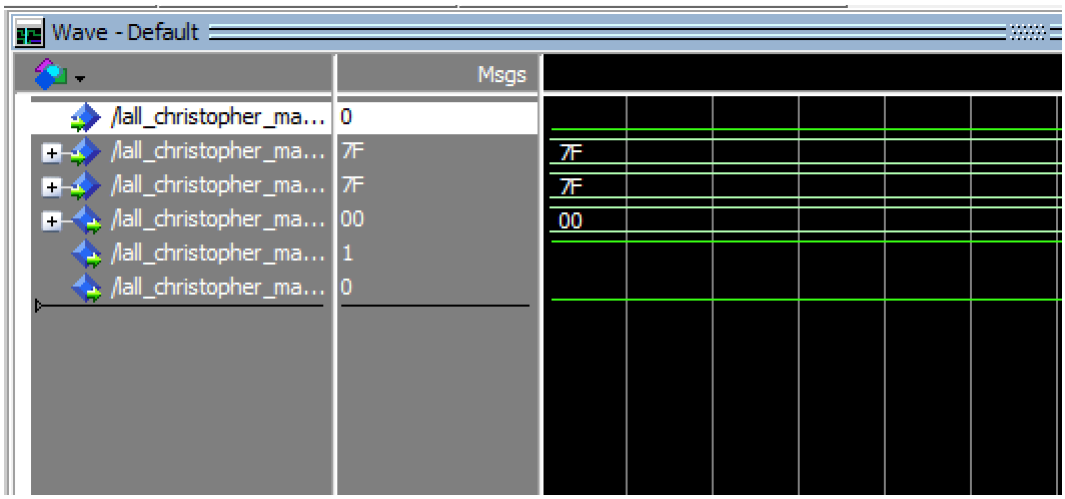

Figure 42. LPM case 3g

Test Bench:

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

entity Lall_Christopher_MARCH6_2022_16bit is
end Lall_Christopher_MARCH6_2022_16bit;

architecture arch_test of Lall_Christopher_MARCH6_2022_16bit is
component Lall_Christopher_MARCH6_2022_add_sub
PORT(
A : IN std_logic_vector(15 downto 0); --16 bit input A
B : IN std_logic_vector(15 downto 0); --16 bit input B
X : OUT std_logic_vector(15 downto 0); --Output
C : OUT std_logic; --carry out
S : IN std_logic --Select Bit
);
END COMPONENT;

--Define Inputs as internal signal
signal A : std_logic_vector(15 downto 0) := (others => '0');
signal B : std_logic_vector(15 downto 0) := (others => '0');
signal S : std_logic := '0';

--Define Outputs as internal signal
signal X : std_logic_vector(15 downto 0);
signal C : std_logic;
signal error : std_logic := '0';

begin
uut: Lall_Christopher_MARCH6_2022_add_sub
PORT MAP (
A => A, -- Definition of the Unit Under Test
B => B,
X => X,
C => C,
S => S
);
process
begin
S<='1';
A<="0000000000000011";
B<="0000000000000011";
wait for 100 ns;
if (X /= "0000000000000111") then
error <= '1';
end if;
wait for 100 ns;
if (error = '0') then
report "No errors detected. Simulation successful " & " A: " & integer'image(to_integer(unsigned(A))) & " B: " & integer'image(to_int
failure;
    else
        report "Error detected " & " A: " & integer'image(to_integer(unsigned(A))) & " B: " & integer'image(to_integer(unsigned(B)))
end if;

end process;
end arch_test;
```

Figure 43. VHDL code for test bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Lall_Christopher_MARCH6_2022_add_sub is
port(A,B:in std_logic_vector(15 downto 0); --two 16 bit inputs
X:out std_logic_vector(15 downto 0); --output
C:out std_logic; --carry output
S:in std_logic); --Select Bit to define Addition or subtraction

end Lall_Christopher_MARCH6_2022_add_sub;

architecture Behavioral of Lall_Christopher_MARCH6_2022_add_sub is
signal Y:std_logic_vector(16 downto 0); -- Define internal signal Y
begin
process (A,B,S) --Process with process variables A, B, and S
begin

case S is
when '0' => --S=0 defines the subtraction operation
Y<=('0' & A)-('0' & B);
when others => --S=1 defines the Addition operation
Y<=('0' & A)+('0' & B);
end case;
end process;
C<=Y(16); -- Assignment of the 17th bit as carry out
X<=Y(15 downto 0); -- Assignment of the 1 to 16th bit as 16-bit sum or difference
end Behavioral;
```

Figure 44. VHDL code for add_sub

```
A<="0000000000000011";
B<="0000000000000011";
wait for 100 ns;
if (X /= "0000000000000111") then
error <= '1';
end if;
wait for 100 ns;
if (error = '0') then
report "No errors detected. Simulation successful " & " A: " & integer'image(to_integer(unsigned(A))) & " B: " & integer'image
failure;
    else
        report "Error detected " & " A: " & integer'image(to_integer(unsigned(A))) & " B: " & integer'image(to_integer(unsigne
end if;
```

Figure 45. Error code lines in code

```
** Failure: No errors detected. Simulation successful  A: 3 B: 3 Result: 6
   Time: 200 ns  Iteration: 0  Process: /lall_christopher_march6_2022_16bit/line__40 File: C:/Desktop/Lall_Christopher_MARCH6_2022_ADDER/Lall_Christopher_MARCH6_2022_16bit.vhd
```

Figure 46. test bench success

```
Failure: Error detected  A: 7 B: 3 Incorrect Result: 9 Expected Result: 10
Time: 200 ns  Iteration: 0  Process: /lall_christopher_march6_2022_16bit/line__40 File: C:/Desktop/Lall_Christopher_MARCH6_2022_ADDER/Lall_Christopher_MARCH6_2022_16bit.vhd
```

Figure 47. test bench fail case

Analysis & Explanations:

Each of the VHDL codes allowed us to run a simulation in ModelSim to provide the operand cases. For some files such as the 4bit adder, we are required to add a subtractor which allow us to use operands in ModelSim. With these operands, we can change the type of math we want to be done. In these cases, except for the lpm file, 0 is addition and 1 is subtraction, whereas in the lpm file, it is the opposite. We also have two input waves which allow us to set each operand case and an output which we set to print in hexadecimal.

4Bit Explanation for each operand case:

3a) Most positive 4-bit integer is 0111 which is 7, 7+1 is 8

3b) Most positive 4-bit integer is 0111 which is 7, 7-1 is 6

3c) Most negative 4-bit integer is 1000 which is 8, 8+1 is 9

3d) Most negative 4-bit integer is 1000 which is 8, 8-1 is 7

3e) Most positive 4-bit int – most negative 4-bit int is -1

3f) Most positive 4-bit int + most negative 4-bit int is -1

3g) Most positive 4-bit int - most positive 4-bit int is 0

32Bit Explanation for each operand case:

3a) Most positive 32-bit integer is 7FFFFFFF + 00000001 = 80000000

3b) Most positive 32-bit integer is 7FFFFFFF – 00000001 = 7FFFFFFE

3c) Most negative 32-bit integer is 8000000 + 00000001 = 80000001

3d) Most negative 32-bit integer is 8000000 – 00000001 = 80000001

3e) Most positive 32-bit integer 7FFFFFFF - most negative 32-bit integer 80000000 = FFFFFFFF

3f) Most positive 32-bit integer 7FFFFFFF + most negative 32-bit integer 8000000 = FFFFFFFF

3g) Most positive 32-bit integer 7FFFFFFF - most positive 32-bit integer 7FFFFFFF = 0

Conclusion:

In conclusion, I was able to create each required tasks VHDL file. I then imported the files to ModelSim in which I got my wave files. I was able to successfully do each task.