Weifan Lin

CSc343

Lab 1

02/19/2015

Adders

Objective:

The goal of this lab was to design and understand the functioning of adders. We wanted to first build a full bit adder and use it to build a two-bit adder, and lastly build a 16-bit adder using the two-bit adder with overflow detection.

Specifications:

Full adder inputs:

A: the first operand of the adder.

B: the second operand of the adder.

Cin (carry in): the bit that is carried in from the next less significant stage.

Full adder outputs:

Cout (carry out): the bit that is carried over from an addition when necessary.

S: the sum, which is last bit of the result from the addition.

Cout and S together is a 2-bit sum of the 3 inputs.

2-bit adder inputs:

A[1…0]: the first operand of the adder which is made up of four bits: A[1], and A[0].

B[1…0]: the second operand of the adder which is made up of four bits: B[1], and B[0].

Cin: the bit that is carried in from the less significant stage.

2-bit adder outputs:

Carry: the carry output is the bit that is carried over from an addition.

S[3…0]: The sum of the two operands which is made up of the four bits S[1], and S[0].

16-bit adder inputs:

A[15…0]: the first operand of the adder which is made up of four bits: A[15], A[14], A[13], … and A[0].

B[15…0]: the second operand of the adder which is made up of four bits: B[15], B[14], B[13], … and B[0].

Cin: the bit that is carried in from the less significant stage.

16-bit adder outputs:

Carry: the carry output is the bit that is carried over from an addition.

S[15…0]: The sum of the two operands which is made up of the four bits S[15], S[14], S[13], and S[0].

Overflow: to determine whether or not there is an overflow.

Functionality:

Full Adder

Truth Table

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Full adder circuit adds binary numbers and it can both carry in and carry out values. A 1-bit full adder can add three 1-bit numbers (A, B, and Cin) and produce a 2-bit output, which is made up of Cout and S.

2-bit Adder

A 2-bit adder is made up of 2 1-bit full adders and it adds 2 4-bit numbers, results in a 2-bit sum and when the result required a 3rd bit to carry the output, Cout yells 1, however, carry output is separated from the final sum.

16-bit Adder

A 16-bit adder is made up of 8 2-bit full adders and it adds 2 16-bit numbers, results in a 16-bit sum and when the result required a 17^{th} bit to carry the output, Cout yells 1, however, carry output is separated from the final sum. This adder has an additional output to determine if there was an overflow.

Design:

Full Adder

```
1   -- 1-bit Adder
2
3   LIBRARY IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity BIT_ADDER is
7           port( a, b, cin          : in  STD_LOGIC;
8                 sum, cout          : out STD_LOGIC );
9   end BIT_ADDER;
10
11  architecture BHV of BIT_ADDER is
12  begin
13
14          sum <=  (not a and not b and cin) or
15                          (not a and b and not cin) or
16                          (a and not b and not cin) or
17                          (a and b and cin);
18
19          cout <= (not a and b and cin) or
20                          (a and not b and cin) or
21                          (a and b and not cin) or
22                          (a and b and cin);
23  end BHV;
```

2-bit Adder

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity add16 is
5     port( a, b  : in STD_LOGIC_VECTOR(15 downto 0);
6           cin   : in STD_LOGIC;
7           ans   : out STD_LOGIC_VECTOR(15 downto 0);
8           overflow  : out STD_LOGIC;
9           cout  : out STD_LOGIC  );
10  end add16;
11
12  architecture arch of add16 is
13
14  component add2
15    port( a, b  : in STD_LOGIC_VECTOR(1 downto 0);
16          cin   : in STD_LOGIC;
17          ans   : out STD_LOGIC_VECTOR(1 downto 0);
18          cout  : out STD_LOGIC  );
19  end component;
20
21  signal c1,c2,c3,c4,c5,c6,c7,c8 : STD_LOGIC;
22  signal mux : STD_LOGIC_VECTOR(15 downto 0);
23  begin
24
25    mux(0) <= cin xor b(0);
26    mux(1) <= cin xor b(1);
27    mux(2) <= cin xor b(2);
28    mux(3) <= cin xor b(3);
29    mux(4) <= cin xor b(4);
30    mux(5) <= cin xor b(5);
31    mux(6) <= cin xor b(6);
32    mux(7) <= cin xor b(7);
```

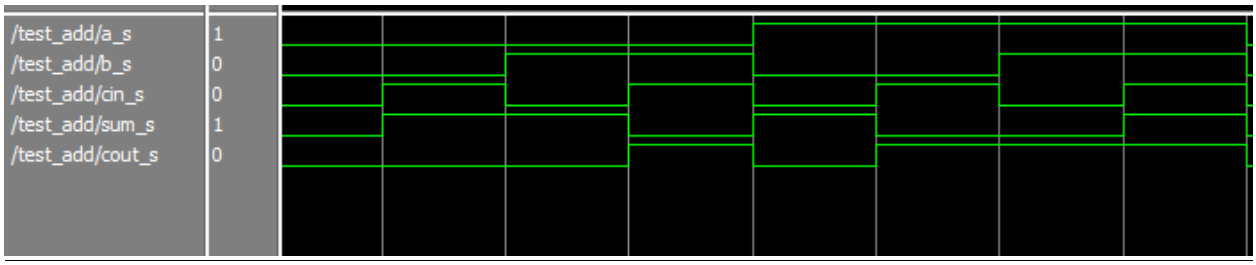
16-bit Adder

```
 1  LIBRARY IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity add2 is
 5    port( a, b  : in STD_LOGIC_VECTOR(1 downto 0);
 6          cin   : in STD_LOGIC;
 7          ans   : out STD_LOGIC_VECTOR(1 downto 0);
 8          cout  : out STD_LOGIC  );
 9  end add2;
10
11  architecture STRUCTURE of add2 is
12  component BIT_ADDER
13    port( a, b, cin  : in  STD_LOGIC;
14          sum, cout  : out STD_LOGIC );
15  end component;
16
17  signal c1 : STD_LOGIC;
18  begin
19  b_adder0: BIT_ADDER port map (a(0), b(0), cin, ans(0), c1);
20  b_adder1: BIT_ADDER port map (a(1), b(1), c1, ans(1), cout);
21
22  END STRUCTURE;
```

```
33    mux(8) <= cin xor b(8);
34    mux(9) <= cin xor b(9);
35    mux(10) <= cin xor b(10);
36    mux(11) <= cin xor b(11);
37    mux(12) <= cin xor b(12);
38    mux(13) <= cin xor b(13);
39    mux(14) <= cin xor b(14);
40    mux(15) <= cin xor b(15);
41    b_adder0: add2 port map (a(1 downto 0), mux(1 downto 0), cin, ans(1 downto 0), c1);
42    b_adder1: add2 port map (a(3 downto 2), mux(3 downto 2), c1, ans(3 downto 2), c2);
43    b_adder2: add2 port map (a(5 downto 4), mux(5 downto 4), c2, ans(5 downto 4), c3);
44    b_adder3: add2 port map (a(7 downto 6), mux(7 downto 6), c3, ans(7 downto 6), c4);
45    b_adder4: add2 port map (a(9 downto 8), mux(9 downto 8), c4, ans(9 downto 8), c5);
46    b_adder5: add2 port map (a(11 downto 10), mux(11 downto 10), c5, ans(11 downto 10), c6);
47    b_adder6: add2 port map (a(13 downto 12), mux(13 downto 12), c6, ans(13 downto 12), c7);
48    b_adder7: add2 port map (a(15 downto 14), mux(15 downto 14), c7, ans(15 downto 14), c8);
49
50    cout <= c8;
51    overflow <= c8 xor c7;
52
53  END arch;
```

Simulation:

Full Adder Waveform



We have:

Case 1: a=0, b=0, cin=0, sum=0 and cout=0;

Case 2: a=0, b=0, cin=1, sum=1 and cout=0;

Case 3: a=0, b=1, cin=0, sum=1 and cout=0;

Case 4: a=0, b=1, cin=1, sum=0 and cout=1;

Case 5: a=1, b=0, cin=0, sum=1 and cout=0;

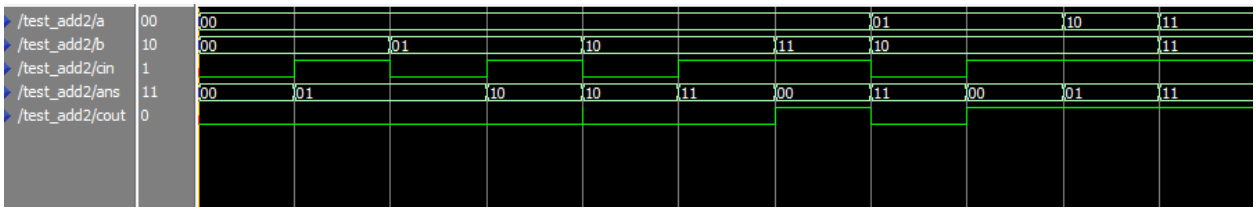
Case 6: a=1, b=0, cin=1, sum=0 and cout=1;

Case 7: a=1, b=1, cin=0, sum=0 and cout=1;

Case 8: a=1, b=1, cin=1, sum=1 and cout=1;

The results from the waveform were all correct.

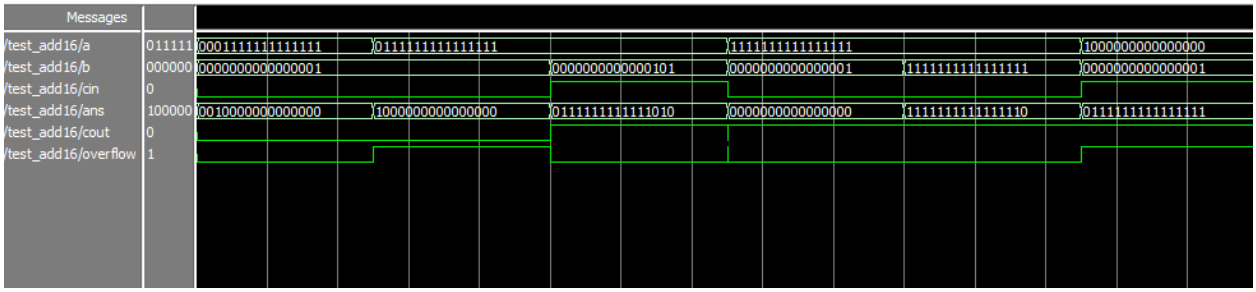
2-Bit Adder Waveform



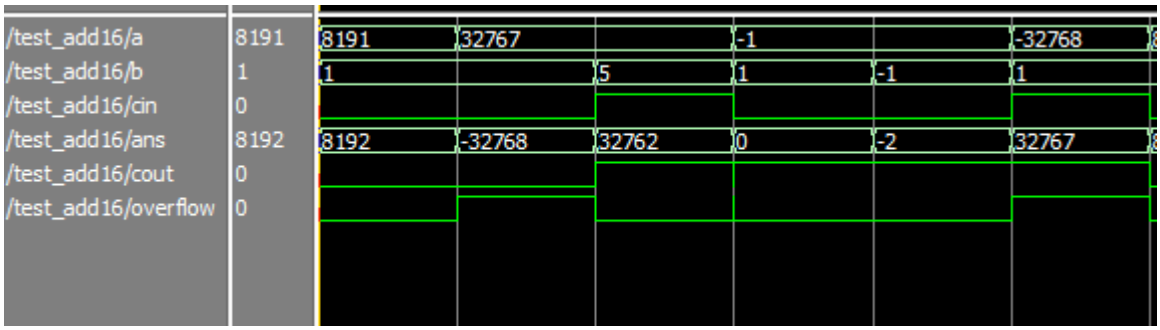
As we can see from the waveform, 2-bit adder has shown all correct results.

16-Bit Adder Waveform

Shown as binary:



Shown as decimal:



We have:

Case 1: positive number + positive number = positive number with no overflow (8191+1=8192).

Case 2: positive number + positive number = positive number with overflow (32767+1= -32768).

Case 3: positive number - positive number = positive number with no overflow (32767-5=32762).

Case 4: negative + positive number = positive number with no overflow (-1+1=0).

Case 5: negative number + negative number = negative number with no overflow (-1+ -1= -2).

Case 6: negative number - positive number = positive number with overflow (-32768-1=32767).

These results are all correct, and overflow had return right values.

Conclusion:

The completion of this lab helped me practice a lot on using Model-SIM. Building 2-bit adder and 16-bit adder subtractor taught me how to use port mapping to simplify circuits by reusing structures. Overall the lab was a bit difficult because there was no TA to help until this week.

Appendix:

```
------------------------1-bit Full Adder-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BIT_ADDER is
        port( a, b, cin       : in  STD_LOGIC;
              sum, cout       : out STD_LOGIC );
end BIT_ADDER;

architecture BHV of BIT_ADDER is
begin
        sum <=  (not a and not b and cin) or
                        (not a and b and not cin) or
                        (a and not b and not cin) or
                        (a and b and cin);
        cout <= (not a and b and cin) or
                        (a and not b and cin) or
                        (a and b and not cin) or
                        (a and b and cin);
end BHV;
```

```
-------------------------1-bit Adder Test File-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TEST_ADD is
end TEST_ADD;

-- Describes the functionality of the tesbench.
architecture TEST of TEST_ADD is
	component BIT_ADDER
		port( a, b, cin		: in  STD_LOGIC;
		sum, cout		: out STD_LOGIC );
	end component;

	for U1: BIT_ADDER use entity WORK.BIT_ADDER(BHV);
	signal A_s, B_s   : STD_LOGIC;
	signal CIN_s      : STD_LOGIC;
	signal SUM_s      : STD_LOGIC;
	signal COUT_s     : STD_LOGIC;

	begin
	U1: BIT_ADDER port map (A_s, B_s, CIN_s, SUM_s, COUT_s);

	-- The process is where the actual testing is done.
	process
	begin
		-- Case 0 : 0+0 with carry in of 0.
```

```
-- Set the signals for the inputs.
A_s <= '0';
B_s <= '0';
CIN_s <= '0';
wait for 10 ns;
assert ( SUM_s = '0'  ) report "Failed Case 0 - SUM" severity error;
assert ( COUT_s = '0' ) report "Failed Case 0 - COUT" severity error;
wait for 40 ns;

-- Carry out the same process outlined above for the other 7 cases.

-- Case 1 : 0+0 with carry in of 1.
A_s <= '0';
B_s <= '0';
CIN_s <= '1';
wait for 10 ns;
assert ( SUM_s = '1'  ) report "Failed Case 1 - SUM" severity error;
assert ( COUT_s = '0' ) report "Failed Case 1 - COUT" severity error;
wait for 40 ns;

-- Case 2 : 0+1 with carry in of 0.
A_s <= '0';
B_s <= '1';
CIN_s <= '0';
wait for 10 ns;
assert ( SUM_s = '1'  ) report "Failed Case 2 - SUM" severity error;
assert ( COUT_s = '0' ) report "Failed Case 2 - COUT" severity error;
wait for 40 ns;

-- Case 3 : 0+1 with carry in of 1.
```

```
A_s <= '0';
B_s <= '1';
CIN_s <= '1';
wait for 10 ns;
assert ( SUM_s = '0'  ) report "Failed Case 3 - SUM" severity error;
assert ( COUT_s = '1' ) report "Failed Case 3 - COUT" severity error;
wait for 40 ns;

-- Case 4 : 1+0 with carry in of 0.
A_s <= '1';
B_s <= '0';
CIN_s <= '0';
wait for 10 ns;
assert ( SUM_s = '1'  ) report "Failed Case 4 - SUM" severity error;
assert ( COUT_s = '0' ) report "Failed Case 4 - COUT" severity error;
wait for 40 ns;

-- Case 5 : 1+0 with carry in of 1.
A_s <= '1';
B_s <= '0';
CIN_s <= '1';
wait for 10 ns;
assert ( SUM_s = '0'  ) report "Failed Case 5 - SUM" severity error;
assert ( COUT_s = '1' ) report "Failed Case 5 - COUT" severity error;
wait for 40 ns;

-- Case 6 : 1+1 with carry in of 0.
A_s <= '1';
B_s <= '1';
CIN_s <= '0';
```

```
                wait for 10 ns;
                assert ( SUM_s = '0'  ) report "Failed Case 6 - SUM" severity error;
                assert ( COUT_s = '1' ) report "Failed Case 6 - COUT" severity error;
                wait for 40 ns;

                -- Case 7 : 1+1 with carry in of 1.
                A_s <= '1';
                B_s <= '1';
                CIN_s <= '1';
                wait for 10 ns;
                assert ( SUM_s = '1'  ) report "Failed Case 7 - SUM" severity error;
                assert ( COUT_s = '1' ) report "Failed Case 7 - COUT" severity error;
                wait for 40 ns;

        end process;
END TEST;


-------------------------2-bit Adder-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add2 is
        port( a, b                  : in    STD_LOGIC_VECTOR(1 downto 0);
            cin   : in STD_LOGIC;
            ans             : out   STD_LOGIC_VECTOR(1 downto 0);
            cout            : out   STD_LOGIC                   );
end add2;

architecture STRUCTURE of add2 is
```

```
component BIT_ADDER
        port( a, b, cin             : in  STD_LOGIC;
            sum, cout               : out STD_LOGIC );
end component;

signal c1 : STD_LOGIC;
begin

b_adder0: BIT_ADDER port map (a(0), b(0), cin, ans(0), c1);
b_adder1: BIT_ADDER port map (a(1), b(1), c1, ans(1), cout);
END STRUCTURE;



-------------------------2-bit Adder Test File-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TEST_ADD2 is
end TEST_ADD2;

architecture TEST of TEST_ADD2 is

        component add2
                port( a, b                  : in   STD_LOGIC_VECTOR(1 downto 0);
                    cin   : in   STD_LOGIC;
                    ans             : out               STD_LOGIC_VECTOR(1 downto 0);
                    cout            : out     STD_LOGIC                 );
        end component;

        for U1: add2 use entity WORK.ADD2(STRUCTURE);
```

```
signal a, b                    : STD_LOGIC_VECTOR(1 downto 0);
signal cin   : STD_LOGIC;
signal ans                     : STD_LOGIC_VECTOR(1 downto 0);
signal cout                    : STD_LOGIC;

begin
U1: add2 port map (a,b,cin,ans,cout);

        process
        begin

                a <= "00";
                b <= "00";
                cin <= '0';
                wait for 10 ns;
                assert ( ans = "00" )       report "Failed Case 1 - ans" severity error;
                assert ( cout = '0' ) report "Failed Case 1 - cout" severity error;
                wait for 40 ns;


                a <= "00";
                b <= "00";
                cin <= '1';
                wait for 10 ns;
                assert ( ans = "01" )       report "Failed Case 2 - ans" severity error;
                assert ( cout = '0' ) report "Failed Case 2 - cout" severity error;
                wait for 40 ns;


                a <= "00";
```

```
b <= "01";
cin <= '0';
wait for 10 ns;
assert ( ans = "01" )       report "Failed Case 3 - ans" severity error;
assert ( cout = '0' ) report "Failed Case 3 - cout" severity error;
wait for 40 ns;


a <= "00";
b <= "01";
cin <= '1';
wait for 10 ns;
assert ( ans = "10" )       report "Failed Case 4 - ans" severity error;
assert ( cout = '0' ) report "Failed Case 4 - cout" severity error;
wait for 40 ns;


a <= "00";
b <= "10";
cin <= '0';
wait for 10 ns;
assert ( ans = "10" )       report "Failed Case 5 - ans" severity error;
assert ( cout = '0' ) report "Failed Case 5 - cout" severity error;
wait for 40 ns;

a <= "00";
b <= "10";
cin <= '1';
wait for 10 ns;
assert ( ans = "11" )       report "Failed Case 6 - ans" severity error;
```

```
assert ( cout = '0' ) report "Failed Case 6 - cout" severity error;
wait for 40 ns;



a <= "00";
b <= "11";
cin <= '1';
wait for 10 ns;
assert ( ans = "00" )        report "Failed Case 7 - ans" severity error;
assert ( cout = '1' ) report "Failed Case 7 - cout" severity error;
wait for 40 ns;



a <= "01";
b <= "10";
cin <= '0';
wait for 10 ns;
assert ( ans = "11" )        report "Failed Case 8 - ans" severity error;
assert ( cout = '0' ) report "Failed Case 8 - cout" severity error;
wait for 40 ns;



a <= "01";
b <= "10";
cin <= '1';
wait for 10 ns;
assert ( ans = "00" )        report "Failed Case 9 - ans" severity error;
assert ( cout = '1' ) report "Failed Case 9 - cout" severity error;
wait for 40 ns;
```

```
                a <= "10";
                b <= "10";
                cin <= '1';
                wait for 10 ns;
                assert ( ans = "01" )       report "Failed Case 10 - ans" severity error;
                assert ( cout = '1' ) report "Failed Case 10 - cout" severity error;
                wait for 40 ns;

                a <= "11";
                b <= "11";
                cin <= '1';
                wait for 10 ns;
                assert ( ans = "11" )       report "Failed Case 11 - ans" severity error;
                assert ( cout = '1' ) report "Failed Case 11 - cout" severity error;
                wait for 40 ns;


        end process;
END TEST;


------------------------16-bit Adder Sub-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add16 is
    port( a, b                  : in    STD_LOGIC_VECTOR(15 downto 0);
        cin   : in STD_LOGIC;--_VECTOR(15 downto 0);
        ans             : out   STD_LOGIC_VECTOR(15 downto 0);
```

```
        overflow        : out   STD_LOGIC;
        cout        : out   STD_LOGIC           );
end add16;

architecture arch of add16 is

component add2
    port( a, b          : in    STD_LOGIC_VECTOR(1 downto 0);
        cin   : in STD_LOGIC;
        ans         : out   STD_LOGIC_VECTOR(1 downto 0);
        cout        : out   STD_LOGIC           );
end component;

signal c1,c2,c3,c4,c5,c6,c7,c8 : STD_LOGIC;
signal mux : STD_LOGIC_VECTOR(15 downto 0);
begin

  mux(0) <= cin xor b(0);
  mux(1) <= cin xor b(1);
  mux(2) <= cin xor b(2);
  mux(3) <= cin xor b(3);
  mux(4) <= cin xor b(4);
  mux(5) <= cin xor b(5);
  mux(6) <= cin xor b(6);
  mux(7) <= cin xor b(7);
  mux(8) <= cin xor b(8);
  mux(9) <= cin xor b(9);
  mux(10) <= cin xor b(10);
  mux(11) <= cin xor b(11);
  mux(12) <= cin xor b(12);
```

```
  mux(13) <= cin xor b(13);
  mux(14) <= cin xor b(14);
  mux(15) <= cin xor b(15);
  b_adder0: add2 port map (a(1 downto 0), mux(1 downto 0), cin, ans(1 downto 0), c1);
  b_adder1: add2 port map (a(3 downto 2), mux(3 downto 2), c1, ans(3 downto 2), c2);
  b_adder2: add2 port map (a(5 downto 4), mux(5 downto 4), c2, ans(5 downto 4), c3);
  b_adder3: add2 port map (a(7 downto 6), mux(7 downto 6), c3, ans(7 downto 6), c4);
  b_adder4: add2 port map (a(9 downto 8), mux(9 downto 8), c4, ans(9 downto 8), c5);
  b_adder5: add2 port map (a(11 downto 10), mux(11 downto 10), c5, ans(11 downto 10), c6);
  b_adder6: add2 port map (a(13 downto 12), mux(13 downto 12), c6, ans(13 downto 12), c7);
  b_adder7: add2 port map (a(15 downto 14), mux(15 downto 14), c7, ans(15 downto 14), c8);

  cout <= c8;
  --overflow <= ((a(14)and (b(14) or c6)) or (c6 and b(14)))xor  c7;
  overflow <= c8 xor c7;

END arch;



-------------------------16-bit Adder Sub Test File-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Decare a testbench.  Notice that the testbench does not have any
-- input or output ports.

entity TEST_ADD16 is
end TEST_ADD16;

-- Describes the functionality of the tesbench.
```

```
architecture TEST of TEST_ADD16 is

 component add16
  port( a, b                  : in   STD_LOGIC_VECTOR(15 downto 0);
       cin   : in   STD_LOGIC;--_VECTOR(15 downto 0);
       ans                    : out              STD_LOGIC_VECTOR(15 downto 0);
       cout                   : out     STD_LOGIC;
       overflow               : out     STD_LOGIC);
 end component;

 --for U1: add16 use entity WORK.add16(arch);
 signal a, b                  : STD_LOGIC_VECTOR(15 downto 0);
 signal cin   : STD_LOGIC;--_VECTOR(15 downto 0);
 signal ans                   : STD_LOGIC_VECTOR(15 downto 0);
 signal cout                  : STD_LOGIC;
 signal overflow                      : STD_LOGIC;

 begin
 U1: add16 port map (a,b,cin,ans,cout,overflow);

  process
  begin
     -- pos + pos: no overflow
   a <= "0001111111111111";
   b <= "0000000000000001";
   cin <= '0';
   wait for 10 ns;
   assert ( ans = "0010000000000000" )report "Failed Case 1 - ans" severity error;
   assert ( cout = '0' ) report "Failed Case 1 - cout" severity error;
```

```
assert ( overflow = '0' ) report "Failed Case 1 - overflow" severity error;
wait for 40 ns;
  -- pos + pos: overflow
a <= "0111111111111111";
b <= "0000000000000001";
cin <= '0';
wait for 10 ns;
assert ( ans = "1000000000000000" )report "Failed Case 2 - ans" severity error;
assert ( cout = '0' ) report "Failed Case 2 - cout" severity error;
assert ( overflow = '1' ) report "Failed Case 2 - overflow" severity error;
wait for 40 ns;

  -- pos - pos: no overflow
a <= "0111111111111111";
b <= "0000000000000101";
cin <= '1';
wait for 10 ns;
assert ( ans = "0111111111111010" )report "Failed Case 3 - ans" severity error;
assert ( cout = '1' ) report "Failed Case 3 - cout" severity error;
assert ( overflow = '0' ) report "Failed Case 3 - overflow" severity error;
wait for 40 ns;

  -- neg + pos: no overflow
a <= "1111111111111111";
b <= "0000000000000001";
cin <= '0';
wait for 10 ns;
assert ( ans = "0000000000000000" )report "Failed Case 4 - ans" severity error;
assert ( cout = '1' ) report "Failed Case 4 - cout" severity error;
assert ( overflow = '0' ) report "Failed Case 4 - overflow" severity error;
```

```
    wait for 40 ns;

     -- neg + neg: no overflow
    a <= "1111111111111111";
    b <= "1111111111111111";
    cin <= '0';
    wait for 10 ns;
    assert ( ans = "1111111111111110" )report "Failed Case 5 - ans" severity error;
    assert ( cout = '1' ) report "Failed Case 5 - cout" severity error;
    assert ( overflow = '0' ) report "Failed Case 5 - overflow" severity error;
    wait for 40 ns;

     -- neg - pos: overflow
    a <= "1000000000000000";
    b <= "0000000000000001";
    cin <= '1';
    wait for 10 ns;
    assert ( ans = "0111111111111111" )report "Failed Case 6 - ans" severity error;
    assert ( cout = '1' ) report "Failed Case 6 - cout" severity error;
    assert ( overflow = '1' ) report "Failed Case 6 - overflow" severity error;
    wait for 40 ns;
  end process;
END TEST;
```