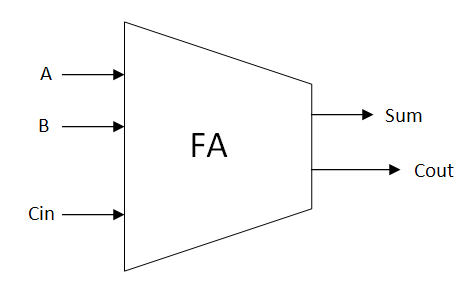
**Question 1:** Simplify fulladder

****

**Figure:** SimplifiedFull-Adder circuit

In previous lab, you have asked to design Full-Adder module by designing low-level of XOR and AND gates and construct all as a top-level to created Full-Adder. The simplified Full-Adder consists of three inputs (A, B & Cin) and two outputs (Sum & Cout). By using Boolean expression below,

Sum = (A XOR B) XOR Cin;

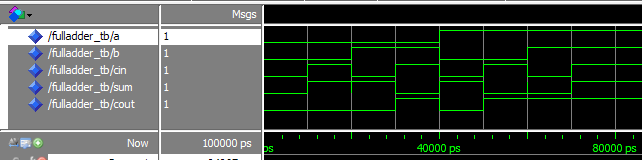
Cout = (A AND B) OR (A AND Cin) OR ( B AND Cin)

Write a VHDL code of simplified Full-Adder based on Boolean expression given. Then, create it testbench running at 80ns and your waveform result should equivalent with truth table given.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **Cin** | **Sum** | **Cout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Figure:** Full-Adder truth table

**Result:**



**Answer:**

**Behavioral code:**

library ieee;

use ieee.std\_logic\_1164.all;

entity simplify\_fulladder is

port ( A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end simplify\_fulladder;

architecture behav of simplify\_fulladder is

begin

Sum <= (A xor B) xor Cin;

Cout <= (A and B) or (A and Cin) or (B and Cin);

end behav;

**Testbench:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity simfulladder\_tb is -- entity declaration

end simfulladder\_tb;

architecture TB of simfulladder\_tb is

component simplify\_fulladder

PORT ( A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end component;

signal a : std\_logic;

signal b : std\_logic;

signal cin : std\_logic;

signal sum : std\_logic;

signal cout : std\_logic;

begin

DUT: simplify\_fulladder port map (

A=>a,

B=>b,

Cin=>cin,

Sum=>sum,

Cout=>cout);

process

begin

a<= '0'; b<= '0'; cin<='0'; wait for 10 ns;

a<= '0'; b<= '0'; cin<='1'; wait for 10 ns;

a<= '0'; b<= '1'; cin<='0'; wait for 10 ns;

a<= '0'; b<= '1'; cin<='1'; wait for 10 ns;

a<= '1'; b<= '0'; cin<='0'; wait for 10 ns;

a<= '1'; b<= '0'; cin<='1'; wait for 10 ns;

a<= '1'; b<= '1'; cin<='0'; wait for 10 ns;

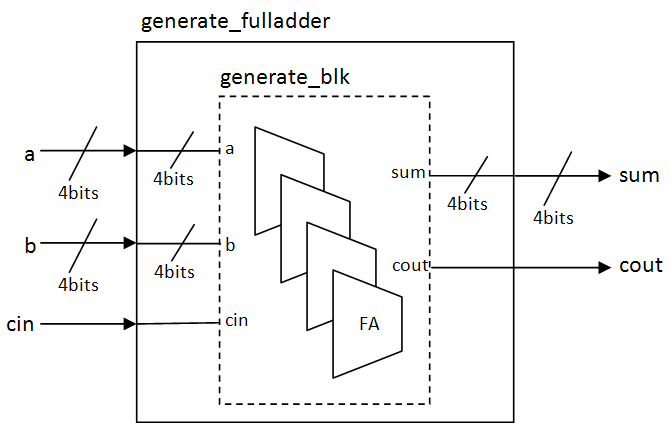
a<= '1'; b<= '1'; cin<='1'; wait for 10 ns;

wait;

end process;

end TB;

**Question 2:** Generate Block for 4-bit Full-Adder.

****

**Figure:** The generate4-bit Full-Adder module

Figure above illustrated generate 4-bit Full-Adder Module. It has low-level design of Full-Adder which is generated 4 times inside block generate\_blk. This block functional is same as 4-bit Full-Adder. Top-level (generate\_fulladder) has three inputs (a, b, cin) which two of the inputs convey 4-bits signal (e.g. “0011”) and two outputs (sum, cout) which sum convey 4-bits output signal.

Base on connection above, write VHDL code for generate 4-bit Full-Adder design. Then, create it testbench running at 80ns and your waveform result should equivalent with truth table given.

Below is the snippet how to generate the 4-bits Full-Adder:

generate\_blk : for i in 3 downto 0 generate

begin

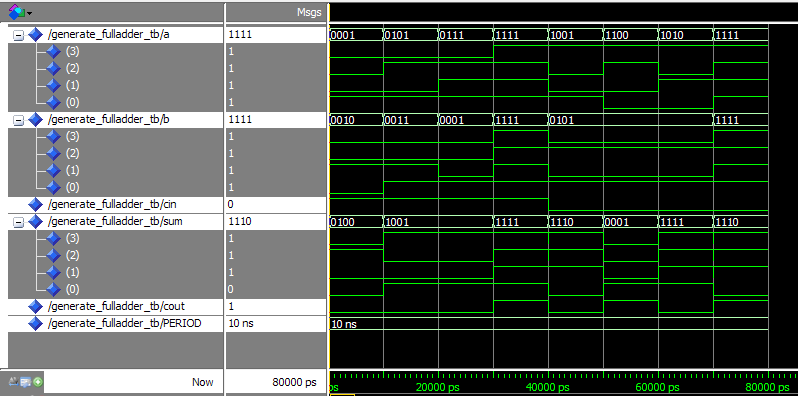
FA : simplify\_fulladder port map (a(i),b(i),c(i),sum(i),c(i+1));

end generate generate\_blk;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **a3** | **a2** | **a1** | **a0** | **b3** | **b2** | **b1** | **b0** | **Cin** | **sum3** | **sum2** | **sum1** | **sum0** | **cout** |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

**Figure:** Generate 4-bits Full-Adder truth table

**Result:**

****

**Answer:**

**Behavioral code:**

--simplify\_fulladder------------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity simplify\_fulladder is

port ( A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end simplify\_fulladder;

architecture behav of simplify\_fulladder is

begin

Sum <= (A xor B) xor Cin;

Cout <= (A and B) or (A and Cin) or (B and Cin);

end behav;

--AND gate-----------------------------------------------------

library ieee;

use ieee.std\_logic\_1164.all;

entity generate\_fulladder is

port ( a : in std\_logic\_vector(3 downto 0);

b : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

sum : out std\_logic\_vector(3 downto 0);

cout : out std\_logic);

end generate\_fulladder;

architecture behav of generate\_fulladder is

component simplify\_fulladder

port( A : in std\_logic;

B : in std\_logic;

Cin : in std\_logic;

Sum : out std\_logic;

Cout : out std\_logic);

end component;

signal c:std\_logic\_vector(4 downto 0);

begin

c(0) <= cin;

generate\_blk : for i in 3 downto 0 generate

begin

FA0 : simplify\_fulladder port map (a(i),b(i),c(i),sum(i),c(i+1));

end generate generate\_blk;

cout <= c(4);

end behav;

**Testbench:**

library ieee;

use ieee.std\_logic\_1164.all;

use std.textio.all;

use work.all;

entity generate\_fulladder\_tb is

end generate\_fulladder\_tb;

architecture tb of generate\_fulladder\_tb is

component generate\_fulladder is

port ( a : in std\_logic\_vector(3 downto 0);

b : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

sum : out std\_logic\_vector(3 downto 0);

cout : out std\_logic);

end component;

constant PERIOD: time := 10 ns;

signal a: std\_logic\_vector(3 downto 0);

signal b: std\_logic\_vector(3 downto 0);

signal cin: std\_logic;

signal sum : std\_logic\_vector(3 downto 0);

signal cout: std\_logic;

begin

DUT: generate\_fulladder port map ( a, b, cin, sum, cout);

INPUTS: process

begin

a<= "0001"; b<= "0010"; cin<='1';

wait for PERIOD;

a<= "0101"; b<= "0011"; cin<='1';

wait for PERIOD;

a<= "0111"; b<= "0001"; cin<='1';

wait for PERIOD;

a<= "1111"; b<= "1111"; cin<='1';

wait for PERIOD;

a<= "1001"; b<= "0101"; cin<='0';

wait for PERIOD;

a<= "1100"; b<= "0101"; cin<='0';

wait for PERIOD;

a<= "1010"; b<= "0101"; cin<='0';

wait for PERIOD;

a<= "1111"; b<= "1111"; cin<='0';

wait for PERIOD;

wait;

end process INPUTS;

end tb;

**Question 3:** Structure and Assert statement in Generate 4-bit Full-Adder.

Refer to the Question1, design a structure 4-bits Full-Adder by using simplify\_fulladder. Then, create a testbench for both struct\_fulladder module and generate\_fulladder module and in the end of the testbench codes, implement Assert statement to ensure the output of both are no different.

Below is the snippet how to insert Assert statement:

verify : process is

begin

wait for PERIOD;

assert sim\_sum = gen\_sum and sim\_cout = gen\_cout

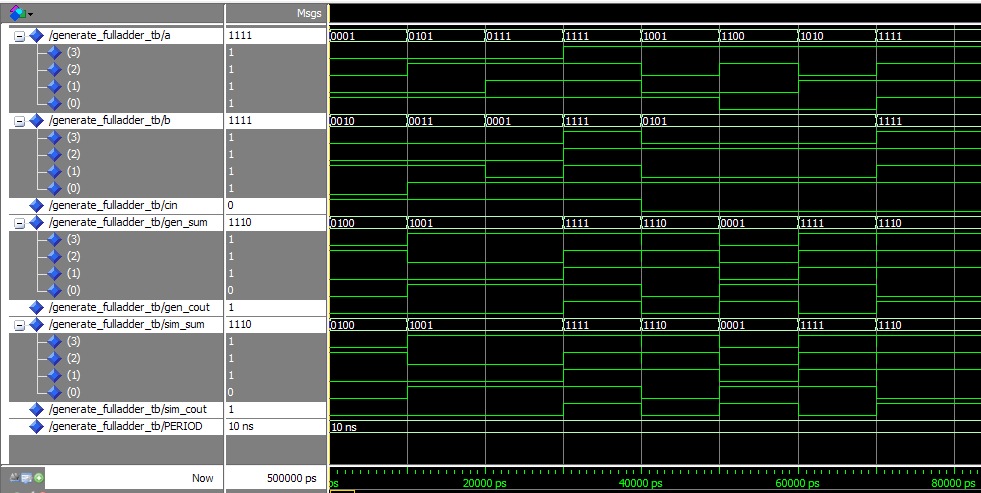
report "implementations have different outputs"

severity error;

-- wait on a,b,cin;

end process verify;

**Answer:**

****

**Testbench:**

library ieee;

use ieee.std\_logic\_1164.all;

use std.textio.all;

use work.all;

entity generate\_fulladder\_tb is

end generate\_fulladder\_tb;

architecture tb of generate\_fulladder\_tb is

component struct\_fulladder is

port ( a : in std\_logic\_vector(3 downto 0);

b : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

sum : out std\_logic\_vector(3 downto 0);

cout: out std\_logic);

end component;

component generate\_fulladder is

port ( a : in std\_logic\_vector(3 downto 0);

b : in std\_logic\_vector(3 downto 0);

cin : in std\_logic;

sum : out std\_logic\_vector(3 downto 0);

cout : out std\_logic);

end component;

constant PERIOD: time := 10 ns;

signal a: std\_logic\_vector(3 downto 0);

signal b: std\_logic\_vector(3 downto 0);

signal cin: std\_logic;

signal gen\_sum : std\_logic\_vector(3 downto 0);

signal gen\_cout: std\_logic;

signal sim\_sum : std\_logic\_vector(3 downto 0);

signal sim\_cout: std\_logic;

begin

DUT0: struct\_fulladder port map ( a, b, cin, sim\_sum, sim\_cout);

DUT1: generate\_fulladder port map ( a, b, cin, gen\_sum, gen\_cout);

INPUTS: process

begin

a<= "0001";

b<= "0010";

cin<='1';

wait for PERIOD;

a<= "0101";

b<= "0011";

cin<='1';

wait for PERIOD;

a<= "0111";

b<= "0001";

cin<='1';

wait for PERIOD;

a<= "1111";

b<= "1111";

cin<='1';

wait for PERIOD;

a<= "1001";

b<= "0101";

cin<='0';

wait for PERIOD;

a<= "1100";

b<= "0101";

cin<='0';

wait for PERIOD;

a<= "1010";

b<= "0101";

cin<='0';

wait for PERIOD;

a<= "1111";

b<= "1111";

cin<='0';

wait for PERIOD;

wait;

end process INPUTS;

verify : process is

begin

wait for PERIOD;

assert sim\_sum = gen\_sum and sim\_cout = gen\_cout

report "implementations have different outputs"

severity error;

-- wait on a,b,cin;

end process verify;

end tb;