**NAME-SHASHANK SHEKHAR PATHAK**

**ROLL-2022ITB030 (HX)**

**ASSIGNMENT-2**

**AND GATE:**

**TestBench Code:**

-- Testbench for AND gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component and\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: and\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple AND gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity and\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end and\_gate;

architecture behavioral of and\_gate is

begin

process(a, b) is

begin

if a = '1' and b= '1' then q<='1';

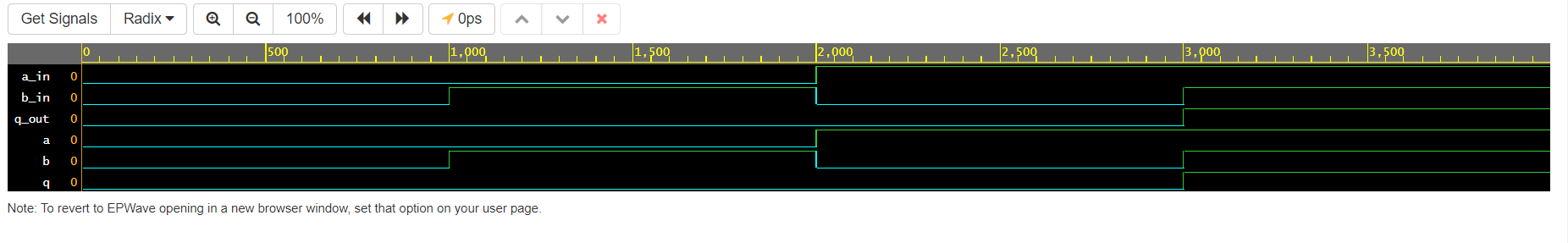
     else q <= '0';

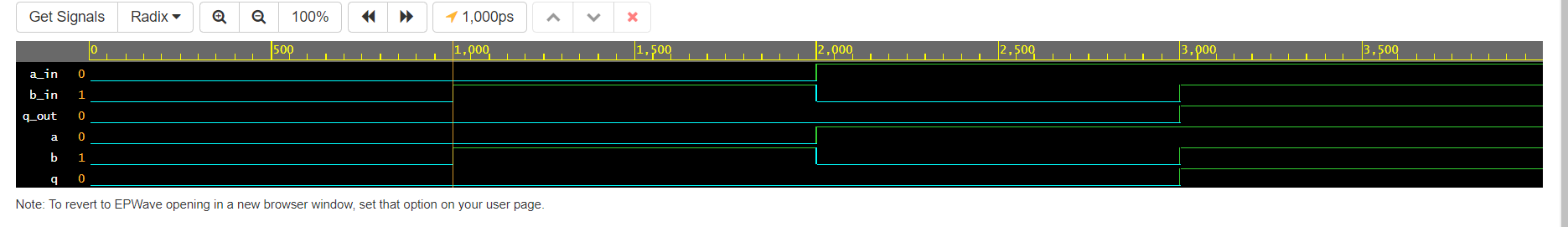
     end if;

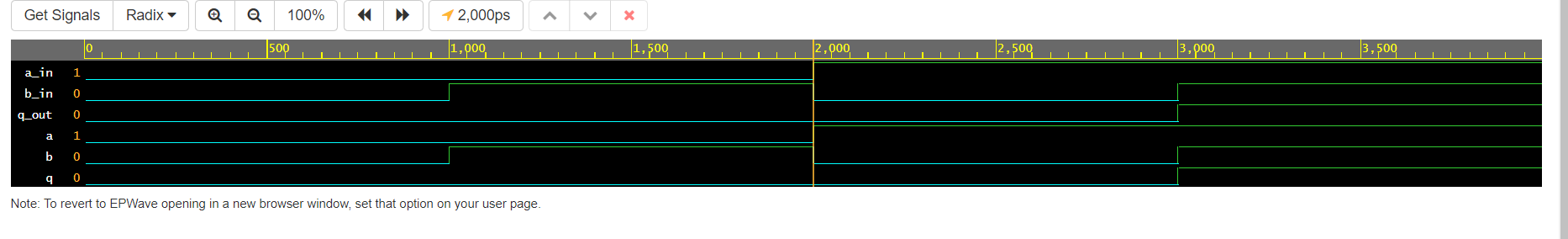
end process;

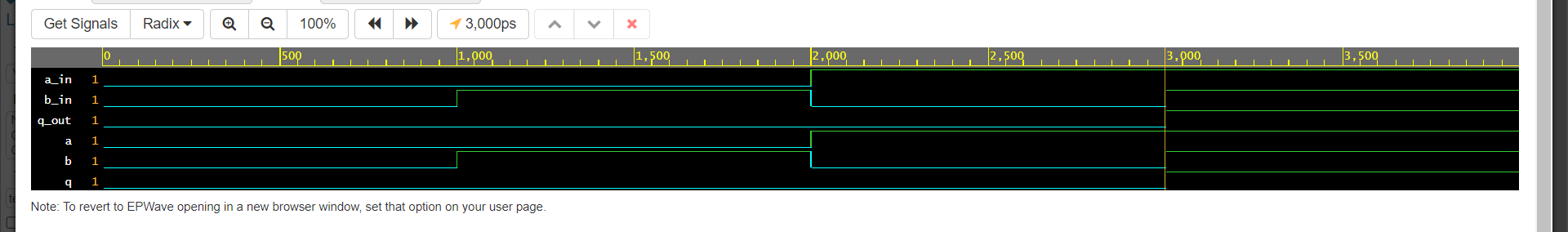
end behavioral;

**OUTPUT:**

****

****

****

****

**OR GATE:**

**TestBench Code:**

-- Testbench for OR gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component or\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: or\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple OR gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity or\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end or\_gate;

architecture behavioral of or\_gate is

begin

process(a, b) is

begin

if a = '0' and b= '0' then q<='0';

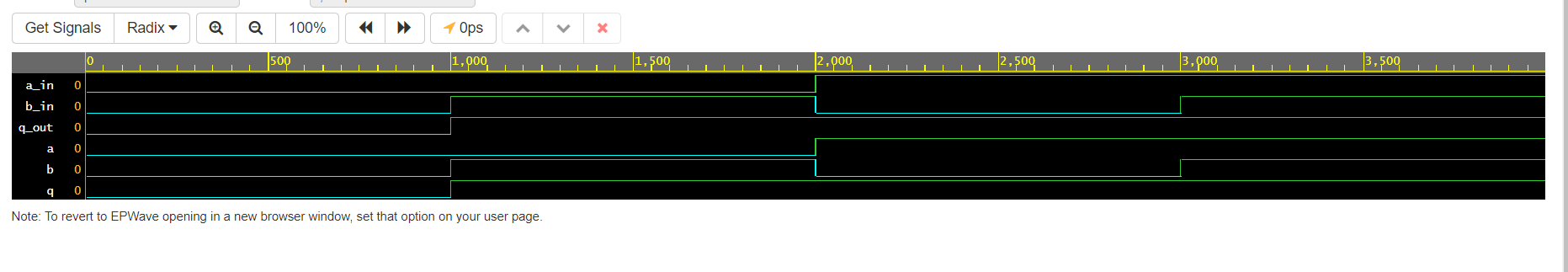
    else q <= '1';

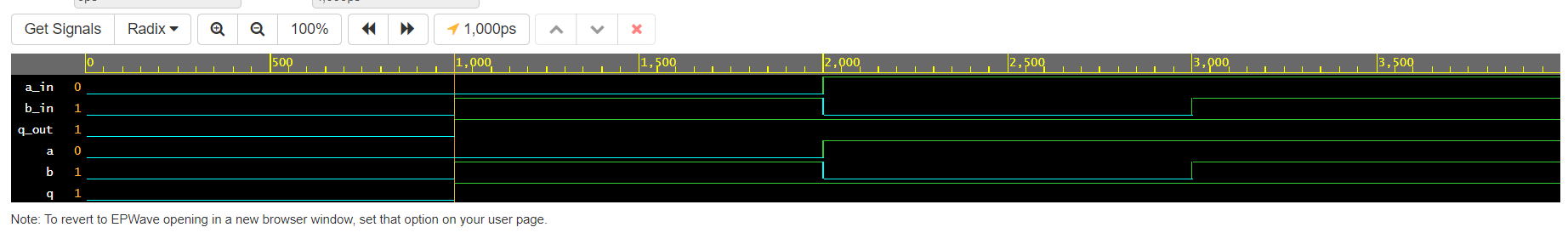
    end if;

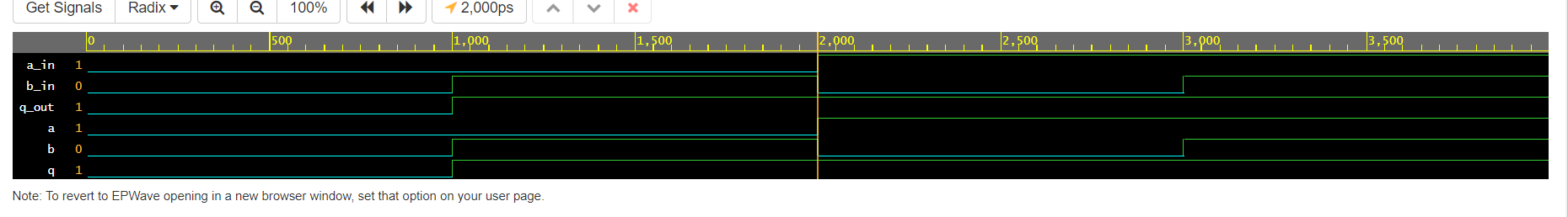
end process;

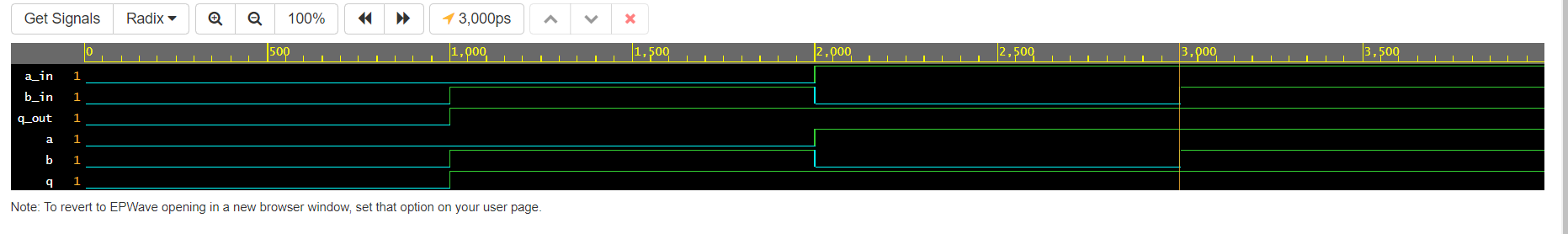
end behavioral;

**OUTPUT:**









**NOT GATE:**

**Testbench Code:**

-- Testbench for NOT gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity NOTGate is

-- empty

end NOTGate;

architecture tb of NOTGate is

-- DUT component

component not\_gate is

port(

a: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: not\_gate port map(a\_in, q\_out);

process

begin

a\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 0" severity error;

a\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 1" severity error;

-- Clear inputs

a\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

--NOT gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity not\_gate is

port(

a: in std\_logic;

q: out std\_logic);

end not\_gate;

architecture behaviour of not\_gate is

begin

process(a) is

begin

if a = '0' then q<='1';

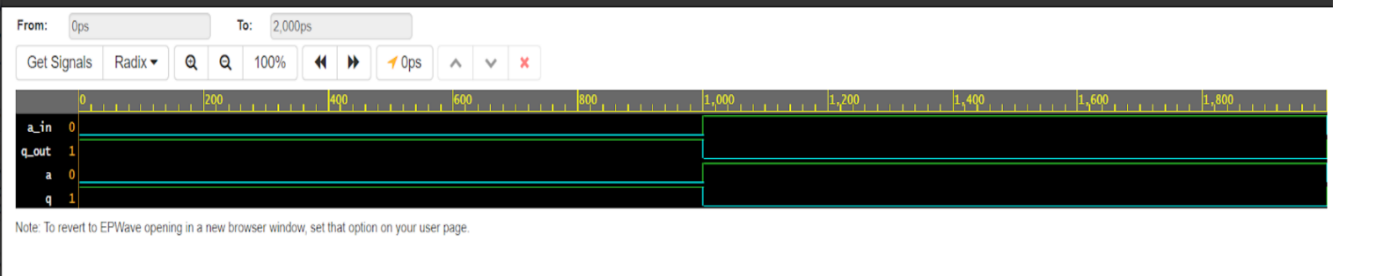
    else q <= '0';

    end if;

end process;

end behaviour;

**OUTPUT:**

****

**NAND GATE:**

**Testbench Code:**

-- Testbench for NAND gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component nand\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: nand\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple NAND gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity nand\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end nand\_gate;

architecture behavioral of nand\_gate is

begin

process(a, b) is

begin

if a = '1' and b= '1' then q<='0';

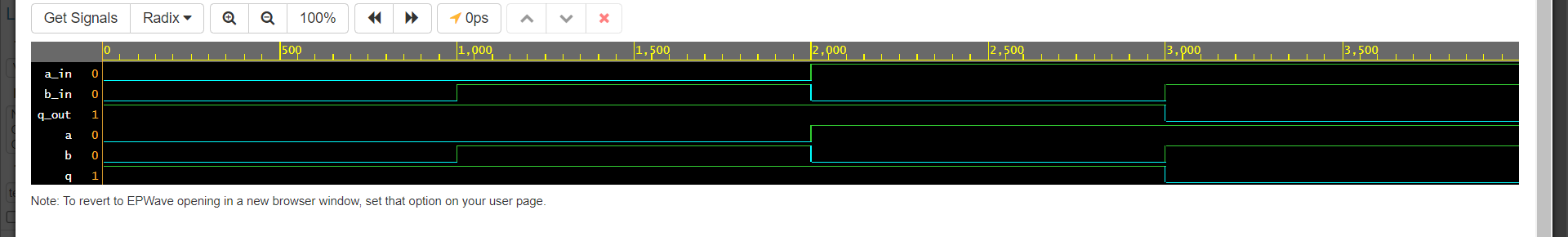
     else q <= '1';

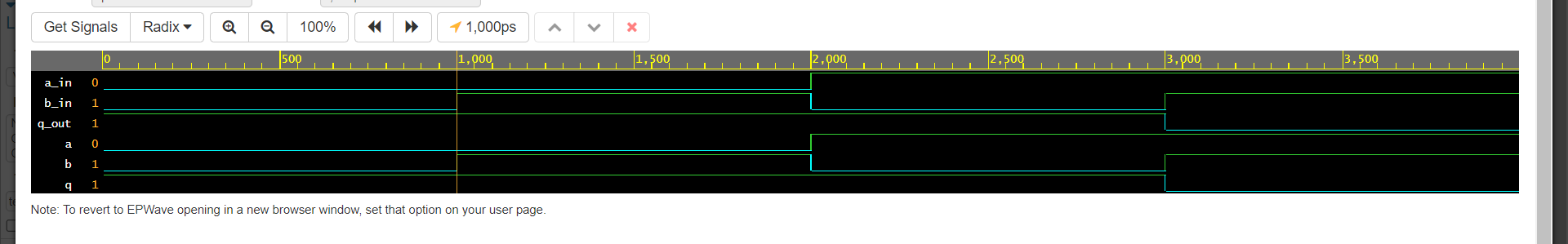
     end if;

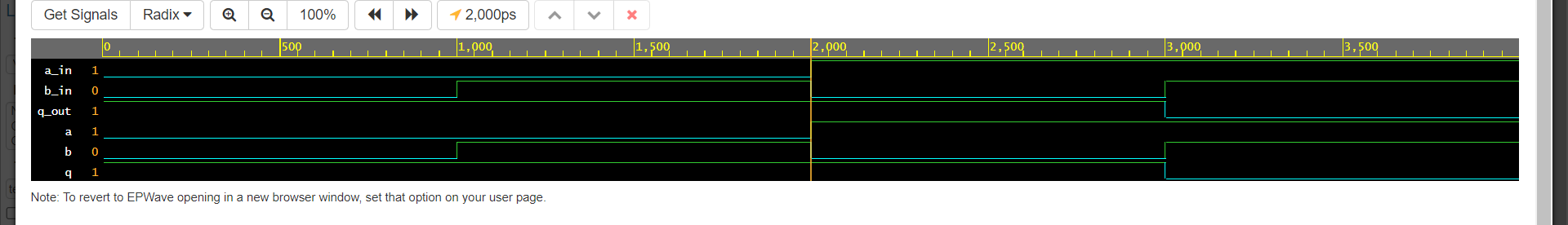
end process;

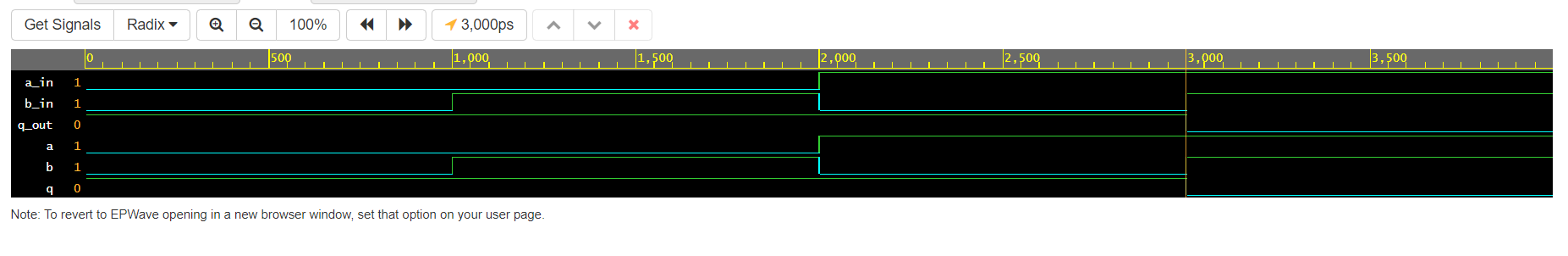
end behavioral;

**OUTPUT:**

****

****

****

****

**NOR GATE:**

**Testbench Code:**

-- Testbench for NOR gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component nor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: nor\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple NOR gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity nor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end nor\_gate;

architecture behavioral of nor\_gate is

begin

process(a, b) is

begin

if a = '0' and b= '0' then q<='1';

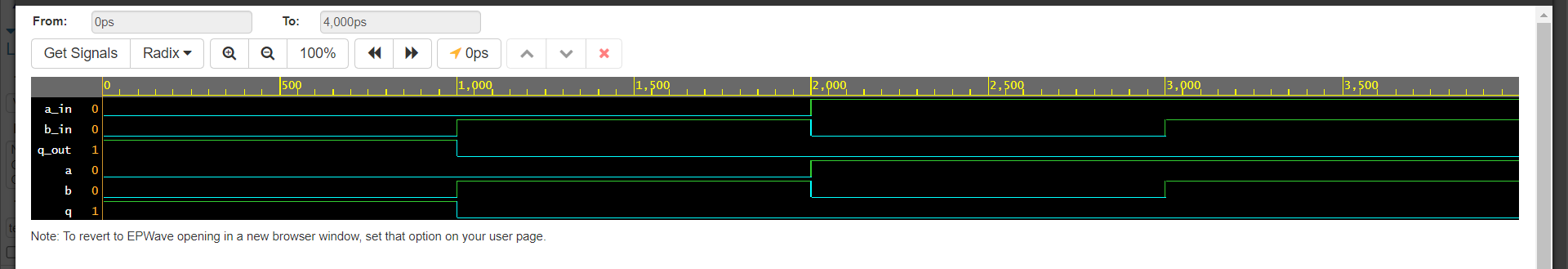
     else q <= '0';

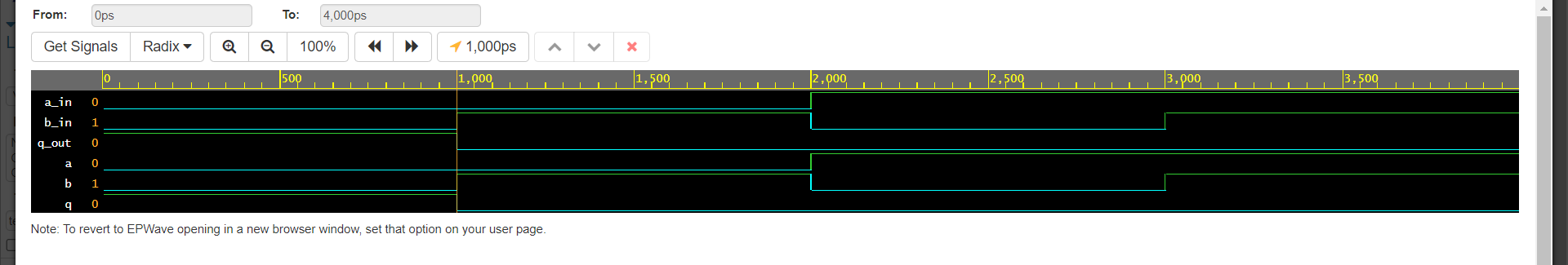
     end if;

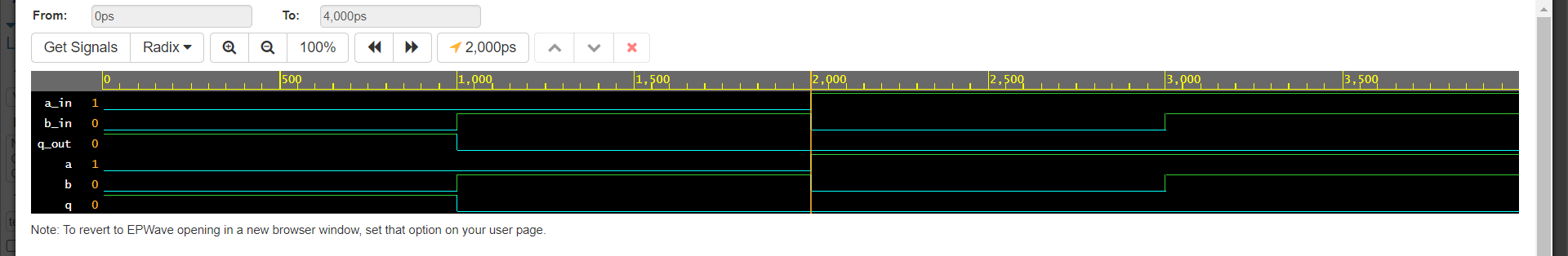
end process;

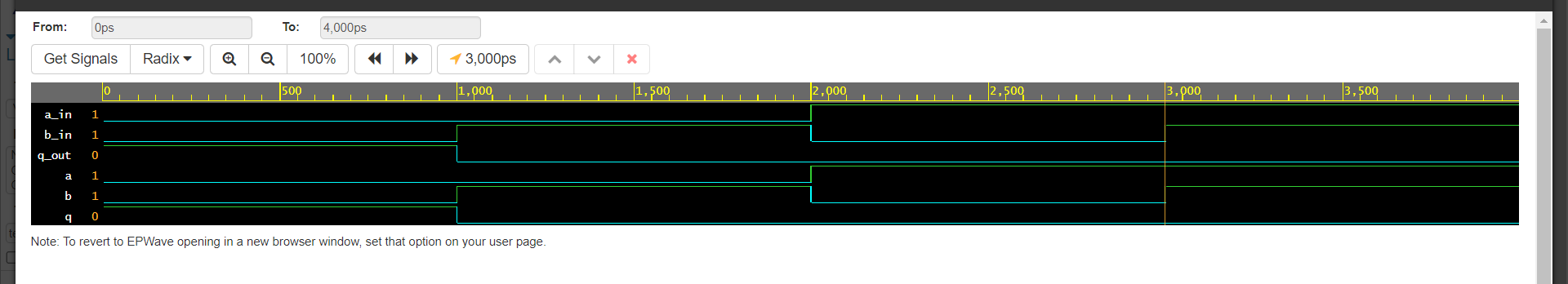
end behavioral;

**OUTPUT:**

****

****

****

****

**XOR GATE:**

**Testbench Code:**

-- Testbench for XOR gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component xor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: xor\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple XOR gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity xor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end xor\_gate;

architecture behavioral of xor\_gate is

begin

process(a, b) is

begin

if a = b then q<= '1';

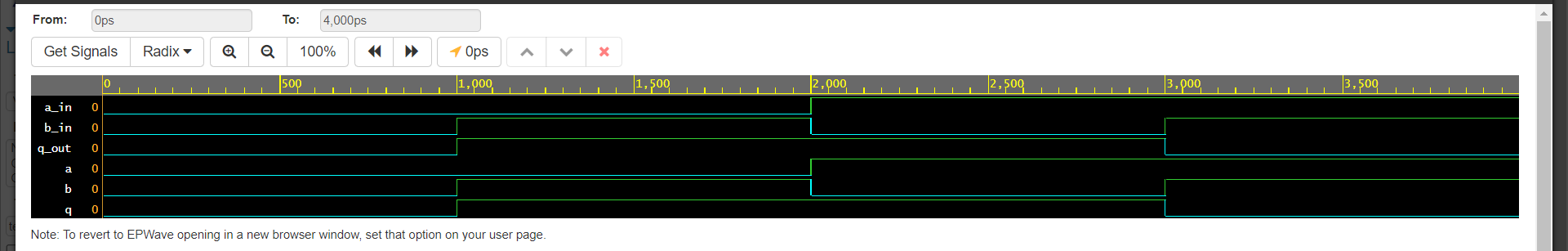
    else q <='0';

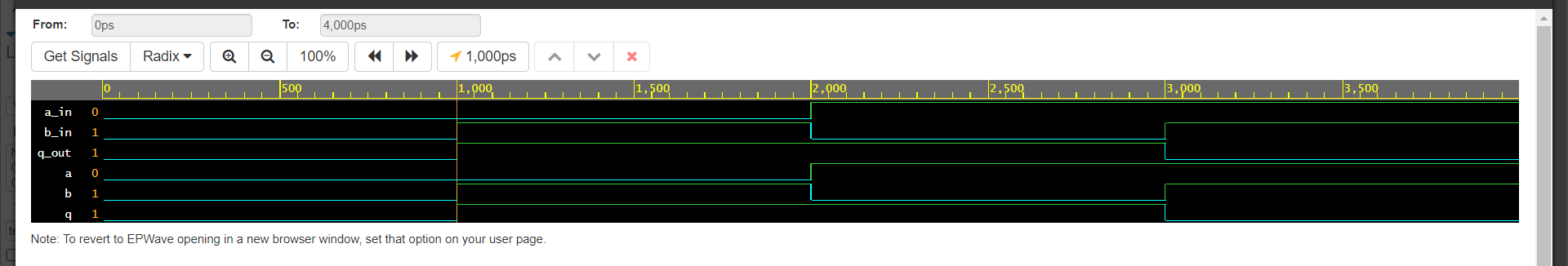
    end if;

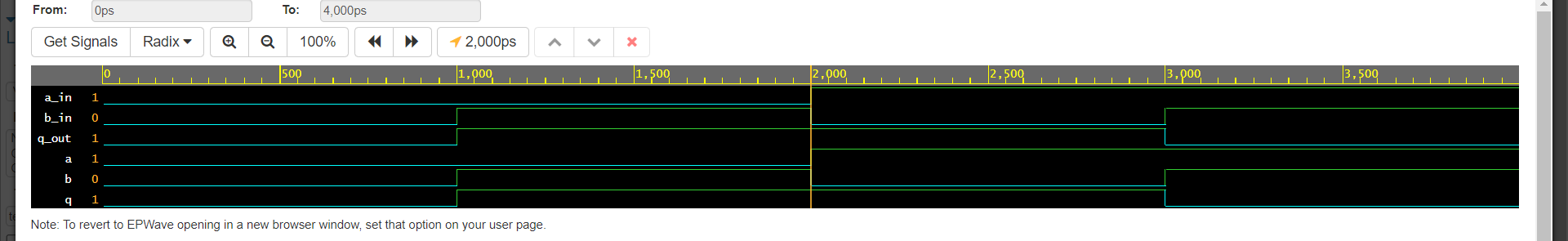
end process;

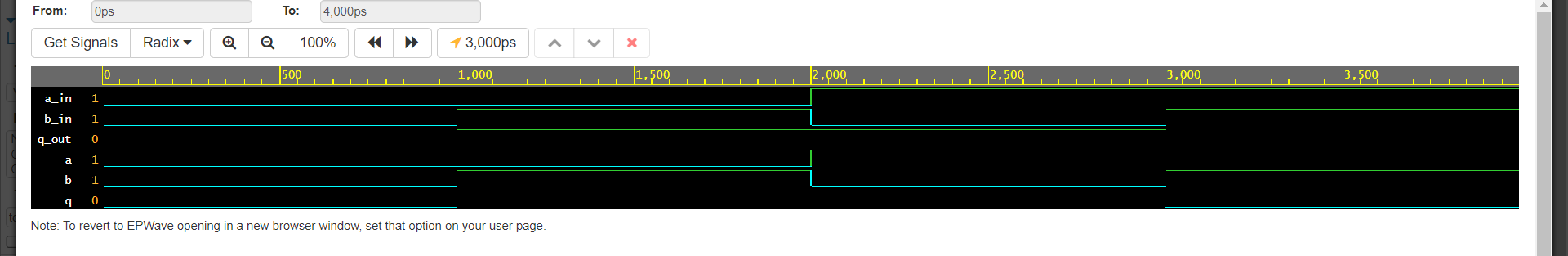
end behavioral;

**OUTPUT:**

****

****

****

****

**XNOR GATE:**

**Testbench Code:**

-- Testbench for XNOR gate

library IEEE;

use IEEE.std\_logic\_1164.all;

entity testbench is

-- empty

end testbench;

architecture tb of testbench is

-- DUT component

component xnor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end component;

signal a\_in, b\_in, q\_out: std\_logic;

begin

-- Connect DUT

DUT: xnor\_gate port map(a\_in, b\_in, q\_out);

process

begin

a\_in <= '0';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='1') report "Fail 0/0" severity error;

a\_in <= '0';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='0') report "Fail 0/1" severity error;

a\_in <= '1';

b\_in <= '0';

wait for 1 ns;

assert(q\_out='0') report "Fail 1/X" severity error;

a\_in <= '1';

b\_in <= '1';

wait for 1 ns;

assert(q\_out='1') report "Fail 1/1" severity error;

-- Clear inputs

a\_in <= '0';

b\_in <= '0';

assert false report "Test done." severity note;

wait;

end process;

end tb;

**Design Code:**

-- Simple XOR gate design

library IEEE;

use IEEE.std\_logic\_1164.all;

entity xnor\_gate is

port(

a: in std\_logic;

b: in std\_logic;

q: out std\_logic);

end xnor\_gate;

architecture behavioral of xnor\_gate is

begin

process(a, b) is

begin

if a = b then q<= '1';

    else q <='0';

    end if;

end process;

end behavioral;

**OUTPUT:**

