Name: Anil Kumar

ROLL_NO. : 2020UCD2101

DIGITAL LOGIC DESIGN PRACTICAL

VHDL PROGRAMMING

Excercise 1

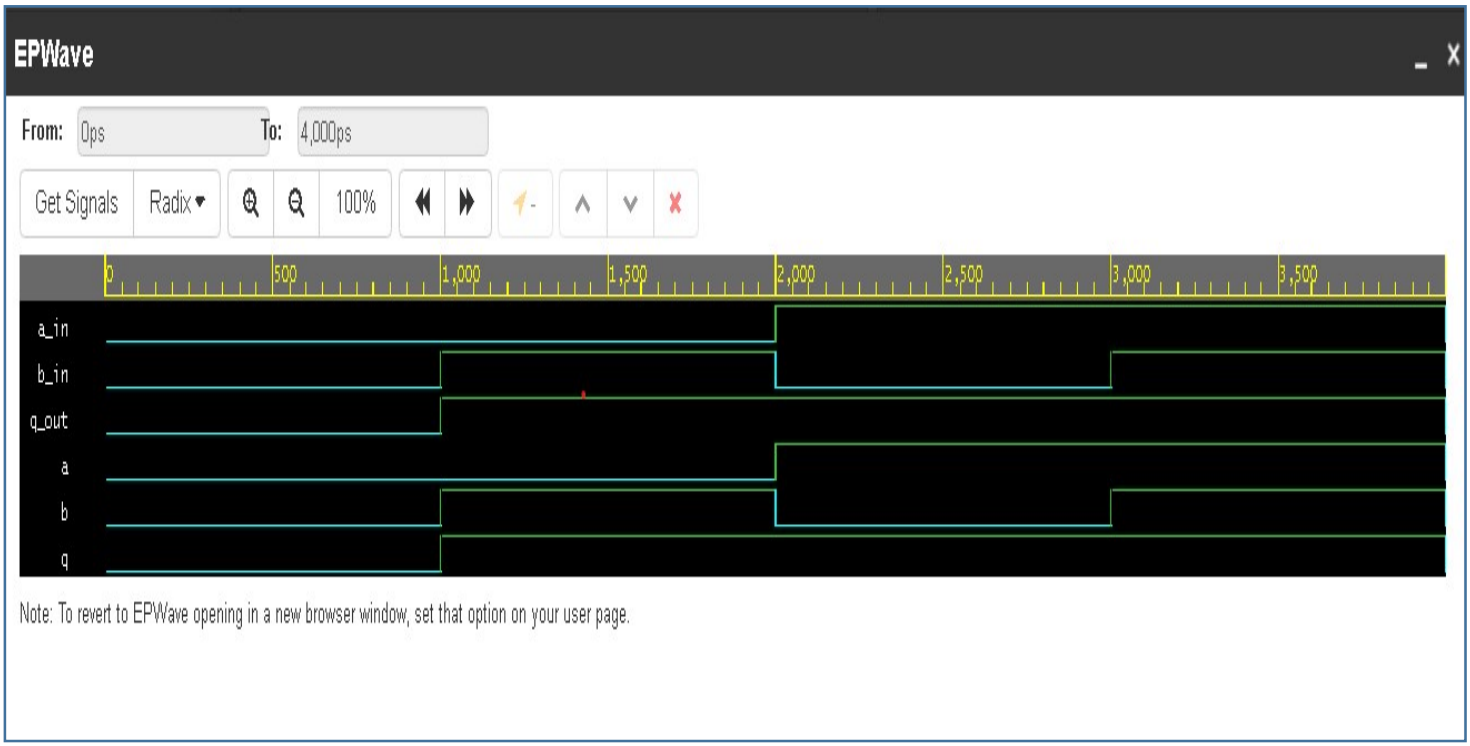
Write a VHDL code for or_gate

```
-- Code your design here
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 rtl of or_gate is
begin
  process(a,b) is
  begin
    q<=a or b;
  end process;
end rtl;
```

```
-- Code your testbench here
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 1/0" 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;
```

Waveform


EPWave
From:
0ps
To:
4,000ps
Get Signals
Radix
100%
0
500
1,000
1,500
2,000
2,500
3,000
3,500
a_in
b_in
q_out
a
b
q
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Excercise 2

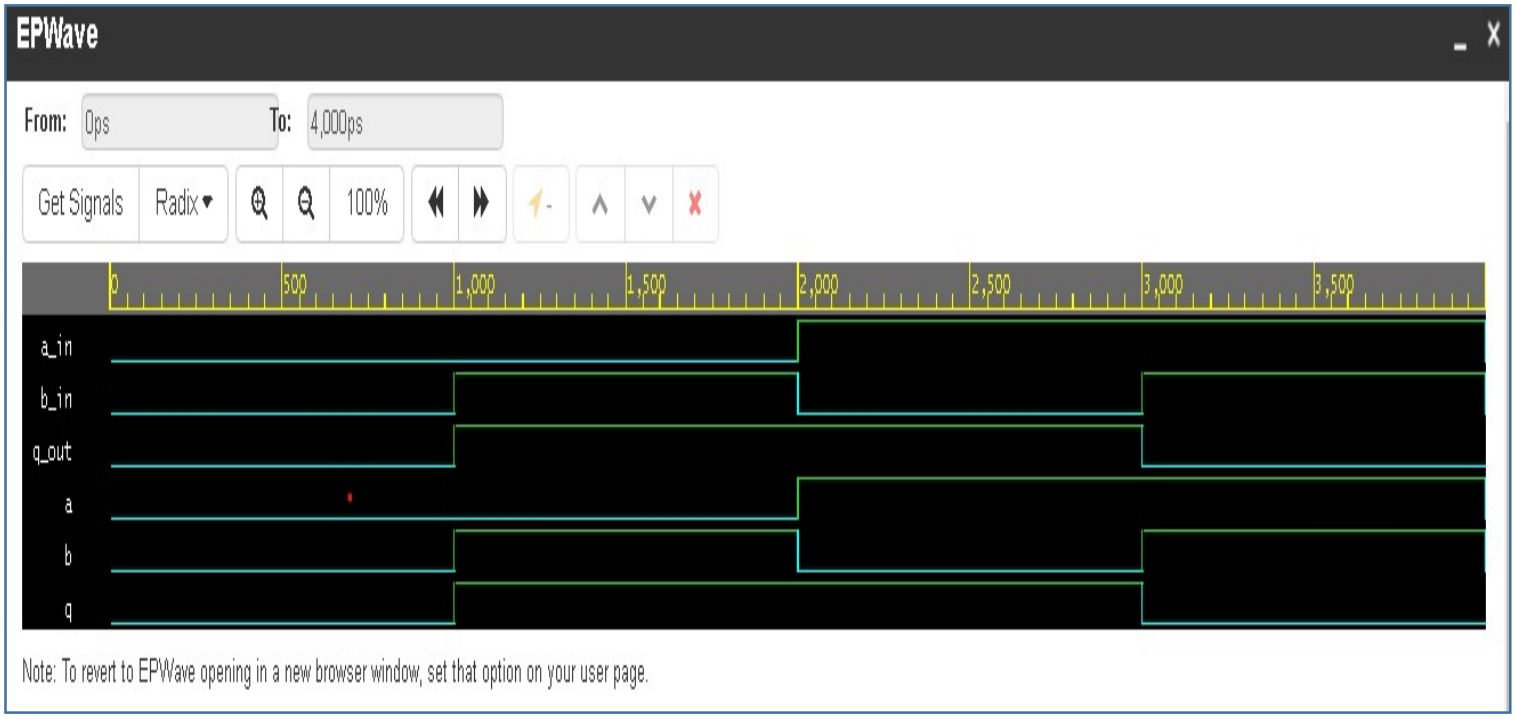
Write a VHDL code for xor_gate

```
-- Code your design here
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 rtl of xor_gate is
begin
  process(a,b) is
  begin
    q<=a xor b;
  end process;
end rtl;
```

```
-- Code your testbench here
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 1/0" 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;
```

Waveform


EPWave
From: 0ps
To: 4,000ps
Get Signals
Radix
100%
0
500
1,000
1,500
2,000
2,500
3,000
3,500
a_in
b_in
q_out
a
b
q
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Excercise 3

To develop a VHDL code for a full adder. using Structural modeling

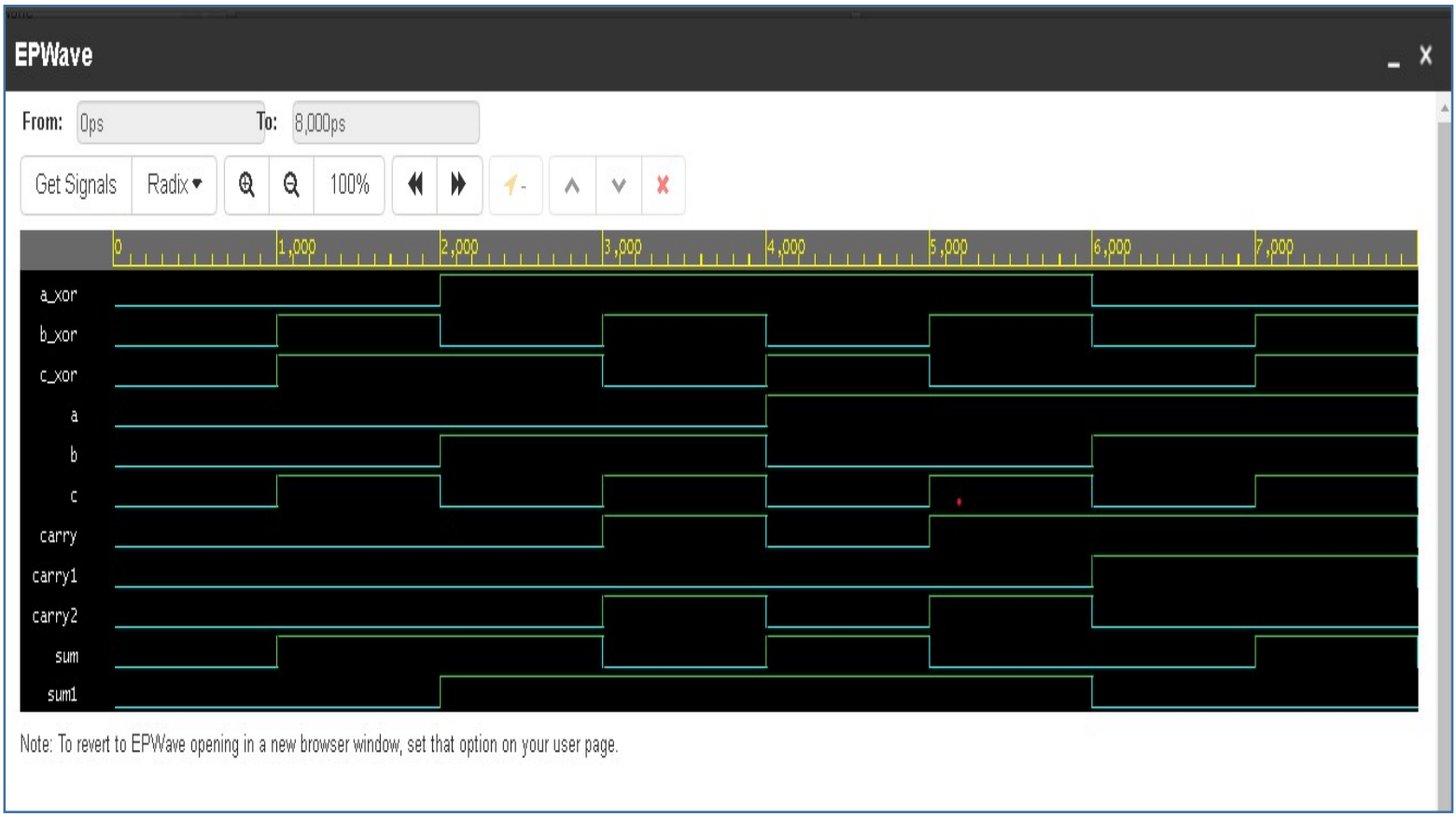
```
-- Simple Full_Adder design

library IEEE;
use IEEE.std_logic_1164.all;

entity andgate is
port(
    a_and: in Bit;
    b_and: in Bit;
    c_and: out Bit);
end andgate;

architecture str of andgate is
begin
    process(a_and, b_and) is
    begin
        c_and <= a_and and b_and;
    end process;
end str;

entity xorgate is
port(
    a_xor: in Bit;
    b_xor: in Bit;
    c_xor: out Bit);
end xorgate;

architecture xor_arc of xorgate is
begin
    process(a_xor, b_xor) is
    begin
        c_xor <= a_xor xor b_xor;
    end process;
end xor_arc;

entity half_adder is
port(
    a_ha: in Bit;
    b_ha: in Bit;
    y_ha: out Bit;
    z_ha: out Bit);
end half_adder;

architecture ha_str of half_adder is
component xorgate is
```

```
port(
    a_xor: in Bit;
    b_xor: in Bit;
    c_xor: out Bit);
end component;
component andgate is
port(
    a_and: in Bit;
    b_and: in Bit;
    c_and: out Bit);
end component;
begin
  XOR1: xorgate port map(a_ha, b_ha, z_ha);
  AND1: andgate port map(a_ha, b_ha, y_ha);
end ha_str;

entity full_adder is
port(
  a: in Bit;
  b: in Bit;
  c: in Bit;
  sum: out Bit;
  carry: out Bit);
end full_adder;

architecture rtl of full_adder is
signal sum1, carry1, carry2: Bit;
component half_adder is
port(
    a_ha: in Bit;
    b_ha: in Bit;
    y_ha: out Bit;
    z_ha: out Bit);
end component;
begin
  HA1: half_adder port map(a, b, carry1, sum1);
  HA2: half_adder port map(sum1, c, carry2, sum);
  process(carry1, carry2) is
  begin
    carry <= carry1 or carry2;
  end process;
end rtl;
```

```
-- Testbench for FullAdder

library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component full_adder is
port(
  a: in Bit;
  b: in Bit;
  c: in Bit;
  sum: out Bit;
  carry: out Bit);
end component;

signal a_in, b_in, c_in, sum_out, carry_out: Bit;

begin

  -- Connect DUT
  DUT: full_adder port map(a_in, b_in, c_in, sum_out, carry_out);
  process
  begin
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    wait for 1 ns;
    assert(sum_out='0' and carry_out='0') report "Fail 0 0 0/0 0" sever
ity error;

    a_in <= '0';
    b_in <= '0';
    c_in <= '1';
    wait for 1 ns;
    assert(sum_out='1' and carry_out='0') report "Fail 0 0 1/1 0" sever
ity error;

    a_in <= '0';
    b_in <= '1';
    c_in <= '0';
    wait for 1 ns;
```

```
    assert(sum_out='1' and carry_out='0') report "Fail 0 1 0/1 0" sever
ity error;
    a_in <= '0';
    b_in <= '1';
    c_in <= '1';
    wait for 1 ns;
    assert(sum_out='0' and carry_out='1') report "Fail 0 1 1/0 1" sever
ity error;

    a_in <= '1';
    b_in <= '0';
    c_in <= '0';
    wait for 1 ns;
    assert(sum_out='1' and carry_out='0') report "Fail 1 0 0/1 0" sever
ity error;

    a_in <= '1';
    b_in <= '0';
    c_in <= '1';
    wait for 1 ns;
    assert(sum_out='0' and carry_out='1') report "Fail 1 0 1/0 1" sever
ity error;

    a_in <= '1';
    b_in <= '1';
    c_in <= '0';
    wait for 1 ns;
    assert(sum_out='0' and carry_out='1') report "Fail 1 1 0/0 1" sever
ity error;
    a_in <= '1';
    b_in <= '1';
    c_in <= '1';
    wait for 1 ns;
    assert(sum_out='1' and carry_out='1') report "Fail 1 1 1/1 1" sever
ity error;

    -- Clear inputs
    a_in <= '0';
    b_in <= '0';
    c_in <= '0';
    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

Waveform


EPWave
From: 0ps
To: 8,000ps
Get Signals
Radix
100%
0
1,000
2,000
3,000
4,000
5,000
6,000
7,000
a_xor
b_xor
c_xor
a
b
c
carry
carry1
carry2
sum
sum1
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Excercise 4

To develop a VHDL code for a 2-bit Comparator

```
-- Code your design here

library IEEE;
use IEEE.std_logic_1164.all;

entity comparator2Bit is
port(
    a: in Bit_vector;
    b: in Bit_vector;
    x: out Bit;
    y: out Bit;
    z: out Bit);
end comparator2Bit;
```

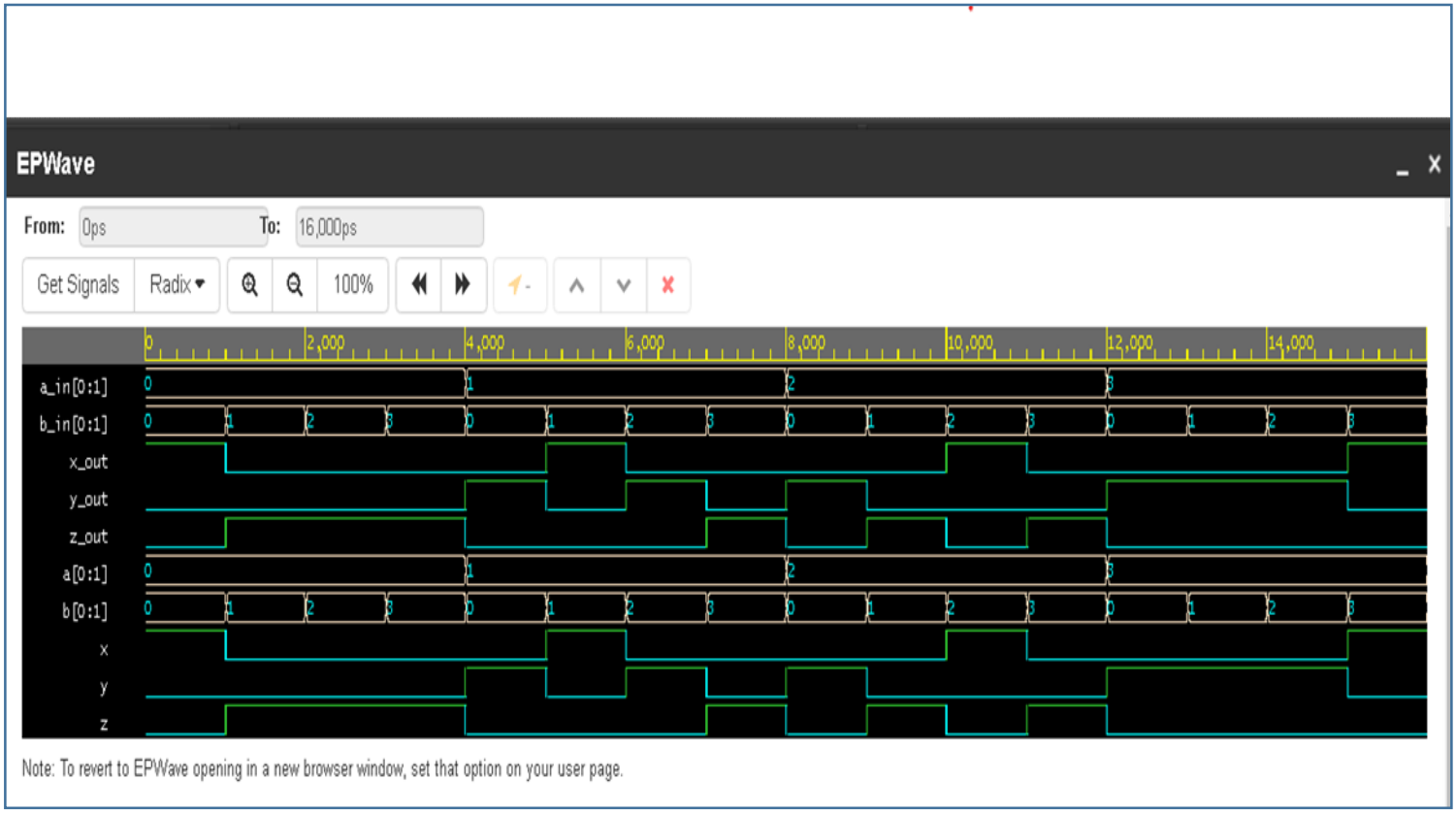
```
architecture str of comparator2Bit is
begin
    process(a, b) is
    begin
        x<=((a(1) xnor b(1)) and (a(0) xnor b(0)));

        y<=((a(1) and (not b(1)))) or ((a(1) xnor b(1)) and (a(0) and (
not b(0))));

        z<=((not a(1)) and b(1)) or ((a(1) xnor b(1)) and ((not a(0)) a
nd b(0)));

    end process;
end str;
```

```
-- Code your testbench here

library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is

-- DUT component
component comparator2Bit is
port(
    a: in Bit_vector;
    b: in Bit_vector;
    x: out Bit;
    y: out Bit;
    z: out Bit);
end component;

signal a_in,b_in : Bit_vector(0 to 1);
signal x_out ,y_out ,z_out : Bit;

begin

  -- Connect DUT
  DUT: comparator2Bit port map(a_in, b_in, x_out, y_out, z_out);

  process
  begin
    a_in <= b"00";
    b_in <= b"00";
    wait for 1 ns;
    assert(x_out = '1' and y_out = '0' and z_out = '0') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"00";
    b_in <= b"01";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '0' and z_out = '1') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"00";
    b_in <= b"10";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '0' and z_out = '1') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"00";
    b_in <= b"11";
```

```
    wait for 1 ns;
    assert(x_out = '0' and y_out = '0' and z_out = '1') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"01";
    b_in <= b"00";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '1' and z_out = '0') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"01";
    b_in <= b"01";
    wait for 1 ns;
    assert(x_out = '1' and y_out = '0' and z_out = '0') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"01";
    b_in <= b"10";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '0' and z_out = '1') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"01";
    b_in <= b"11";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '0' and z_out = '1') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"10";
    b_in <= b"00";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '1' and z_out = '0') report "Fail 0
0 0/0 0" severity error;
    a_in <= b"10";
    b_in <= b"01";
    wait for 1 ns;
    assert(x_out = '0' and y_out = '1' and z_out = '0')
report "Fail 0 0 0/0 0" severity error;
    a_in<=b"10";
    b_in<=b"10";
    wait for 1ns;
    assert(x_out='1' and y_out ='0' and z_out='0')    report "Fail 0 0
0/0 0" severity error;
    a_in<=b"10";
    b_in<=b"11";
    wait for 1ns;
    assert(x_out='0' and y_out ='0' and z_out='1')    report "Fail 0 0
0/0 0" severity error;
    a_in<=b"11";
    b_in<=b"00";
    wait for 1ns;
    assert(x_out='0' and y_out ='1' and z_out='0')    report "Fail 0 0
0/0 0" severity error;
```

```
    a_in<=b"11";
    b_in<=b"01";
    wait for 1ns;
    assert(x_out='0' and y_out ='1' and z_out='0')    report "Fail 0 0
0/0 0" severity error;
    a_in<=b"11";
    b_in<=b"10";
    wait for 1ns;
    assert(x_out='0' and y_out ='1' and z_out='0')    report "Fail 0 0
0/0 0" severity error;
    a_in<=b"11";
    b_in<=b"11";
    wait for 1ns;
    assert(x_out='1' and y_out ='0' and z_out='0')    report "Fail 0 0
0/0 0" severity error;

    -- Clear inputs
    a_in <= b"00";
    b_in <= b"00";
    assert false report "Test done." severity note;
    wait;
  end process;
end tb;
```

Waveform


EPWave
From: 0ps
To: 16,000ps
Get Signals
Radix
100%
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
a_in[0:1]
b_in[0:1]
x_out
y_out
z_out
a[0:1]
b[0:1]
x
y
z
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Excercise 5

Write a VHDL code for 16:1 mux using 2:1 mux

```
-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;


-- and gate
entity andgate is
port(
    a_and :in Bit;
    b_and :in Bit;
    c_and : out Bit);
end andgate;

architecture atr of andgate is
begin
    process(a_and,b_and) is
    begin
        c_and<=a_and and b_and;
    end process;
end atr;

-- orgate
entity orgate is
port(
    a_or :in Bit;
    b_or :in Bit;
    c_or : out Bit);
end orgate;

architecture otr of orgate is
begin
    process(a_or,b_or) is
    begin
        c_or <= a_or or b_or;
    end process;
end otr;
```

```
-- not gate
```

```
entity notgate is
port(
    a_not :in Bit;
    b_not :out Bit);
end notgate;
```

```
architecture ntr of notgate is
begin
    process(a_not) is
    begin
        b_not<=(not a_not) ;
    end process;
end ntr;
```

```
-- 2*1 MUX
```

```
entity Mux_2x1 is
port(
    a_mux1 :in Bit;
    b_mux1 :in Bit;
    s_mux1 :in Bit;
    x_mux1 : out Bit);
end Mux_2x1;

architecture Mutr of Mux_2x1 is
signal sum1,sum2,t_not: Bit;
component notgate is
port(
    a_not :in Bit;
    b_not :out Bit);
end component;

component andgate is
port(
    a_and :in Bit;
    b_and :in Bit;
    c_and : out Bit);
end component;
component orgate is
port(
    a_or :in Bit;
    b_or :in Bit;
    c_or : out Bit);
end component;

begin
    and1: andgate port map(a_mux1,s_mux1,sum1);
    not1: notgate port map(s_mux1,t_not);
    and2: andgate port map(t_not,b_mux1,sum2);
    or1: orgate port map(sum1,sum2,x_mux1);
end Mutr;
```

```
-- 16*1 Mux
entity Mux_16x1 is
port(
```

```
    I: in Bit_vector;
    s: in Bit_vector;
    out1 : out Bit);
end Mux_16x1;

architecture mtr of Mux_16x1 is
signal opt1: Bit_vector(0 to 7);
signal opt2: Bit_vector(0 to 3);
signal opt3: Bit_vector(0 to 1);
component Mux_2x1 is
port(
    a_mux1 :in Bit;
    b_mux1 :in Bit;
    s_mux1 :in Bit;
    x_mux1 : out Bit);
end component;
begin
    mux1: Mux_2x1 port map(I(0),I(1),s(0),opt1(0));
    mux2: Mux_2x1 port map(I(2),I(3),s(0),opt1(1));
    mux3: Mux_2x1 port map(I(4),I(5),s(0),opt1(2));
    mux4: Mux_2x1 port map(I(6),I(7),s(0),opt1(3));
    mux5: Mux_2x1 port map(I(8),I(9),s(0),opt1(4));
    mux6: Mux_2x1 port map(I(10),I(11),s(0),opt1(5));
    mux7: Mux_2x1 port map(I(12),I(13),s(0),opt1(6));
    mux8: Mux_2x1 port map(I(14),I(15),s(0),opt1(7));

    mux9: Mux_2x1 port map(opt1(0),opt1(1),s(1),opt2(0));
    mux10: Mux_2x1 port map(opt1(2),opt1(3),s(1),opt2(1));
    mux11: Mux_2x1 port map(opt1(4),opt1(5),s(1),opt2(2));
    mux12: Mux_2x1 port map(opt1(6),opt1(7),s(1),opt2(3));

    mux13: Mux_2x1 port map(opt2(0),opt2(1),s(2),opt3(0));
    mux14: Mux_2x1 port map(opt2(2),opt2(3),s(2),opt3(1));

    mux15: Mux_2x1 port map(opt3(0),opt3(1),s(3),out1);
end mtr;
```

```
-- Code your testbench here
library IEEE;
use IEEE.std_logic_1164.all;

entity testbench is
-- empty
end testbench;

architecture tb of testbench is
component Mux_16x1 is
port(
    I: in Bit_vector;
    s: in Bit_vector;
    out1 : out Bit);
end component;

signal I1: Bit_vector(0 to 15);
signal s1: Bit_vector(0 to 3);
signal opt: Bit;

begin
DUT: Mux_16x1 port map(I1,s1,opt);

process
    begin
    I1 <= b"0000000000000000";
    s1 <= b"0000";
    wait for 1 ns;
    assert(opt='0') report "Fail 0" severity error;

    I1 <= b"0000000000000001";
    s1 <= b"0000";
    wait for 1 ns;
    assert(opt='1') report "Fail 0" severity error;

    I1 <= b"0000000000000010";
    s1 <= b"1000";
    wait for 1 ns;
    assert(opt='1') report "Fail 0" severity error;

    I1 <= b"0000000000000100";
    s1 <= b"0100";
    wait for 1 ns;
    assert(opt='1') report "Fail 0" severity error;

    I1 <= b"0000000000001000";
    s1 <= b"1100";
    wait for 1 ns;
```

```
    assert(opt='1') report "Fail 0" severity error;

    I1 <= b"0000000000010000";
    s1 <= b"0010";
    wait for 1 ns;
    assert(opt='1') report "Fail 0" severity error;

    --clear inputs
    I1 <= b"0000000000000000";
    s1 <= b"0000";

    assert false report "Test done." severity note;
    wait;
    end process;
end tb;
```

Waveform