DIGITAL LOGIC DESIGN

PRACTICAL

VHDL PROGRAMMING

Name: Anil Kumar

ROLL\_NO. : 2020UCD2101

**VHDL CODES TO IMPLEMENT:**

1. OR GATE

2. X-OR GATE

3. FULL ADDER

4. 2\_BIT\_COMPARATOR

5. 16X1 MUX USING 2X1 MUX

**EACH TOPIC CONTAINS:**

— DESIGN.VHD CODE

— TESTBENCH.VHD CODE

— EPWAVE WAVEFORM

— BLOCK DIAGRAM representation

Exercise 1

Write vhdl code and testbench of

4:16 decoder using 3:8 decoder.

-- Code your design here

library IEEE;

use IEEE.std\_logic\_1164.all;

entity three\_eight\_decode is

port(

    enable: in Bit;

    i: in Bit\_vector(2 downto 0);

    o: out Bit\_vector(7 downto 0));

end three\_eight\_decode;

architecture ted of three\_eight\_decode is

begin

    process(i) is

    begin

        if enable = '1' then

          case i is

              when b"000" => o <= b"00000001";

              when b"001" => o <= b"00000010";

              when b"010" => o <= b"00000100";

              when b"011" => o <= b"00001000";

              when b"100" => o <= b"00010000";

              when b"101" => o <= b"00100000";

              when b"110" => o <= b"01000000";

              when b"111" => o <= b"10000000";

              when others => o <= b"00000000";

          end case;

        else

            o <= b"00000000";

        end if;

    end process;

end ted;

entity four\_sixteen\_decode is

port(

    msb\_input: in Bit;

    input: in Bit\_vector(2 downto 0);

    output1: out Bit\_vector(7 downto 0);

    output2: out Bit\_vector(7 downto 0));

end four\_sixteen\_decode;

architecture fsd of four\_sixteen\_decode is

component three\_eight\_decode is

port(

    enable: in Bit;

    i: in Bit\_vector(2 downto 0);

    o: out Bit\_vector(7 downto 0));

end component;

begin

    --if msb\_input = '0' then

    ted1: three\_eight\_decode port map ((not msb\_input), input, output1);

    ted2: three\_eight\_decode port map (msb\_input, input, output2);

end fsd;

-- Code your testbench here

library IEEE;

use IEEE.std\_logic\_1164.all;

entity decoder4x16\_tb is

end decoder4x16\_tb;

architecture be of decoder4x16\_tb is

component four\_sixteen\_decode is

port(

msb\_input: in Bit;

input: in Bit\_vector(2 downto 0);

output1: out Bit\_vector(7 downto 0);

output2: out Bit\_vector(7 downto 0));

end component;

signal msb: bit;

signal Inp: Bit\_vector(2 downto 0);

signal Outp1: Bit\_vector(7 downto 0);

signal Outp2: Bit\_vector(7 downto 0);

begin

DUT: four\_sixteen\_decode port map(msb,Inp,Outp1,Outp2);

process

    begin

    msb<='0';

    Inp<=b"000";

    wait for 20ns;

    assert(Outp1="10000000" and Outp2="00000000") report "Fail 0" severity error;

    msb<='0';

    Inp<=b"001";

    wait for 20ns;

    assert(Outp1="01000000" and Outp2="00000000") report "Fail 0" severity error;

    msb<='1';

    Inp<=b"000";

    wait for 20ns;

    assert(Outp1="00000000" and Outp2="10000000") report "Fail 0" severity error;

    msb<='1';

    Inp<=b"001";

    wait for 20ns;

    assert(Outp1="00000000" and Outp2="01000000") report "Fail 0" severity error;

    --clear input

    msb<='0';

    Inp<=b"000";

    assert false report "Test done." severity note;

    wait;

end process;

end be;

OUTPUT

BLOCK DIAGRAM

Waveform

Exercise 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;

BLOCK DIAGRAM

Waveform

To develop a VHDL code for a full adder.

Exercise 3

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" severity 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" severity 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" severity 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" severity 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" severity 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" severity 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" severity 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" severity error;

    -- Clear inputs

    a\_in <= '0';

    b\_in <= '0';

    c\_in <= '0';

    assert false report "Test done." severity note;

    wait;

  end process;

end tb;

BLOCK DIAGRAM

Waveform

To develop a VHDL code for a 2-bit Comparator

Exercise 4

-- 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)) and 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;

BLOCK DIAGRAM

Waveform

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

Exercise 5

-- 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

BLOCK DIAGRAM