



Tutorial 2: VHDL Design and Simulation

Example Solution

1 Introduction

The following are example solutions for Tutorial 2. Provided are the VHDL code for the designs, schematics and simulation results, including VHDL code for the test benches. The questions are shown in italics.

The aim of this tutorial is to provide you with experience in VHDL design and simulation by implementing various building blocks which are used for microprocessor interfacing. These include a decoder and a latch. All of these designs will first be simulated using Altium Designer before downloading to the NanoBoard for implementation. The DIP switches and TEST/RESET button will be used as inputs and the LEDs will be used as outputs. As an additional task, the bi-directional octal register described in lectures can be simulated.

2 Decoder

a) Create a new FPGA project and design a three-to-eight decoder, which each of the outputs high for a particular combination of the input. Connect three of the DIP switches to the input and connect the outputs to the LEDs.

A decoder is designed to only have one of its outputs asserted based on the input combination. The truth table for a three to eight decoder is shown in Table 1. In this case the output is active high, that is, the decoded value has logic level 1

Input			Output							
a_2	a_1	a_0	q_7	q_6	q_5	q_4	q_3	q_2	q_1	q_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Table 1: Truth table for a three to eight decoder

while all other outputs are at logic level 0. This could be referred to as *positive logic*. An alternative would be to have the decoded output at logic level 0 while all the other outputs are at logic level 1. This could be referred to as *negative logic*.

A schematic of the decoder as it would appear in a schematic diagram is shown in Figure 1. This forms the basis for the

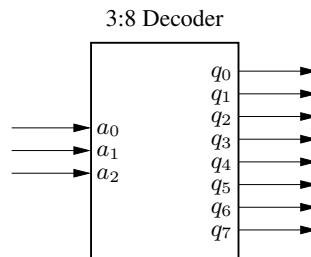


Figure 1: Schematic of a three to eight decoder

entity declaration as all the inputs and outputs are shown.

There are a number of ways that the decoder can be described. Two possible ways are shown starting on Page 7. The template was deleted to make the code more compact. The first example used *ieee.std_logic_unsigned.all* which allows the input to be compared to an unsigned number. The second example used *ieee.numeric_std.all* which includes the type casting function. Either way is fine. It was suggested in lectures just to include both of these libraries.

The schematic used entered in the Altium software should be similar to that shown in Figure 2. Note that the DIP switch

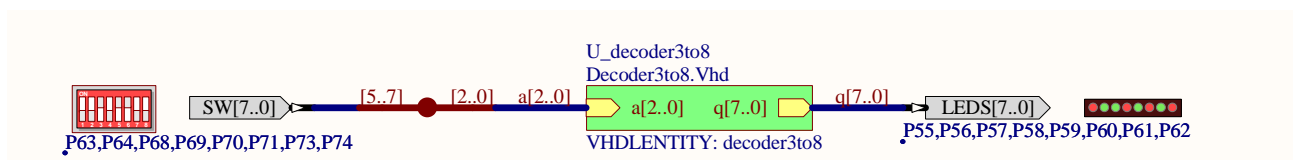


Figure 2: Schematic of the three to eight decoder showing the DIP switch inputs and the LED outputs

inputs are flipped around and the last three are used to make it easier to operate.

In each of the above examples, the *entity* declaration is the same, only the *architecture* changes.

b) Simulate the design by creating a VHDL test bench and using the Altium Designer Simulator. Test all input combinations and verify correct operation. If the simulation does not provide the expected results, modify the design and repeat the simulation.

A test bench for the decoder is shown on Page 8. The bulk of this code is generated automatically by Altium Designer, only the stimulus needs to be added. The result of the simulation is shown in Figure 3. In this simulation, all the input combinations are tested to show correct operation. Note that the extra label was added to the schematic to view the input

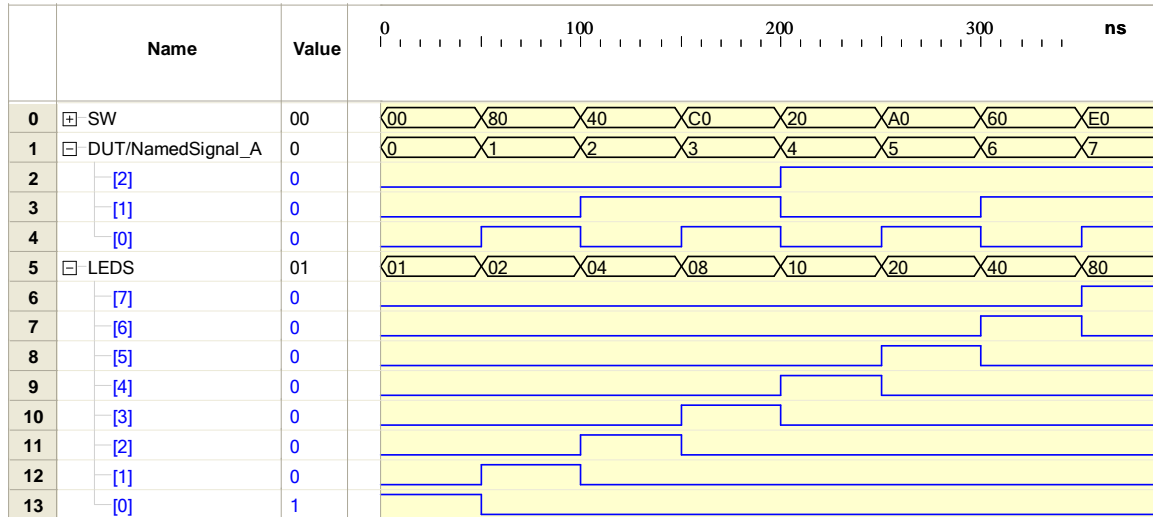


Figure 3: Simulation of the three to eight decoder

in a meaningful way.

c) Download the design to the NanoBoard and verify that the design operates as predicted in the simulation.

Need to configure for use with the FPGA on the NanoBoard are done in the previous tutorial. This should be a straight forward step and will confirm that the design operates as simulated.

d) If the decoder is used to decode the top three bits of an eight-bit address space, what are the addresses corresponding to each of the decoder outputs?

The total number of addresses for an eight bit address space is $2^8 = 256$. If the top three bits were decoded, the address range corresponding to that output would start from the top three bit combination with the lower five bits set to zero and finish at the top three bit combination with the lower five bits set to one. Note that the output of the decoder does not change with changes on the lower five bits. This is completely obvious as none of the lower five bits are inputs to the decoder.

The easiest way to describe the answer is to write an address map. This is shown in Table 2. In this table, the address bits

Output $q_7 \rightarrow q_0$	Address Bits								Range	
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Start	Finish
q_0	0	0	0	X	X	X	X	X	00	1f
q_1	0	0	1	X	X	X	X	X	20	3f
q_2	0	1	0	X	X	X	X	X	40	5f
q_3	0	1	1	X	X	X	X	X	60	7f
q_4	1	0	0	X	X	X	X	X	80	9f
q_5	1	0	1	X	X	X	X	X	a0	bf
q_6	1	1	0	X	X	X	X	X	c0	df
q_7	1	1	1	X	X	X	X	X	e0	ff

Table 2: Address map for decoding the top three bits in an eight bit address space

are labelled A_7 down to A_0 . The top three bits are fed into the decoder to generate the decoded output. The X's are used to show that this bit can take on either value. The address range as a start and finish address is shown in the last column. The number of addresses that each output decodes is $2^5 = 32$.

3 Latch

- a) Create a new FPGA project and design a positive edge-triggered three-input latch. Connect three of the DIP switches to the input and connect the outputs to three of the LEDs. Use the TEST/RESET button as the clock input. Do not worry about debouncing the clock input.

A schematic for the latch is shown in Figure 4.

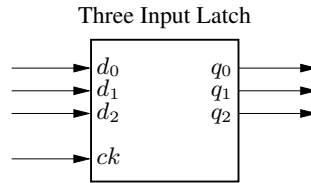


Figure 4: Schematic of a three input latch

In this case, a *process* statement with a sensitivity on the clock input *ck* is used. The VHDL code for this is shown on Page 10. This is a subset of the code that was presented in lectures for a bi-directional octal register. Note that the *d* input is not included in the sensitivity list.

The schematic used is similar to that for the decoder only that there is now an extra input for the TEST/RESET button. The actual schematic should be similar to that shown in Figure 5. Note that the DIP switch inputs was flipped around and

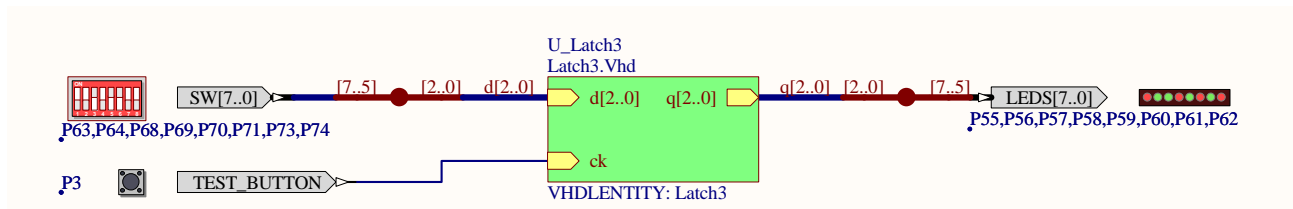


Figure 5: Schematic of the three input latch showing the DIP switch and TEST/RESET button inputs and the LED outputs the last three were used to make it easier to operate.

- b) Simulate the design by creating a VHDL test bench and using the Altium Designer Simulator. Verify correct operation and fix if necessary.

The test bench is shown on Page 10. The simulation of the design is shown in Figure 6. The edge on the clock at which

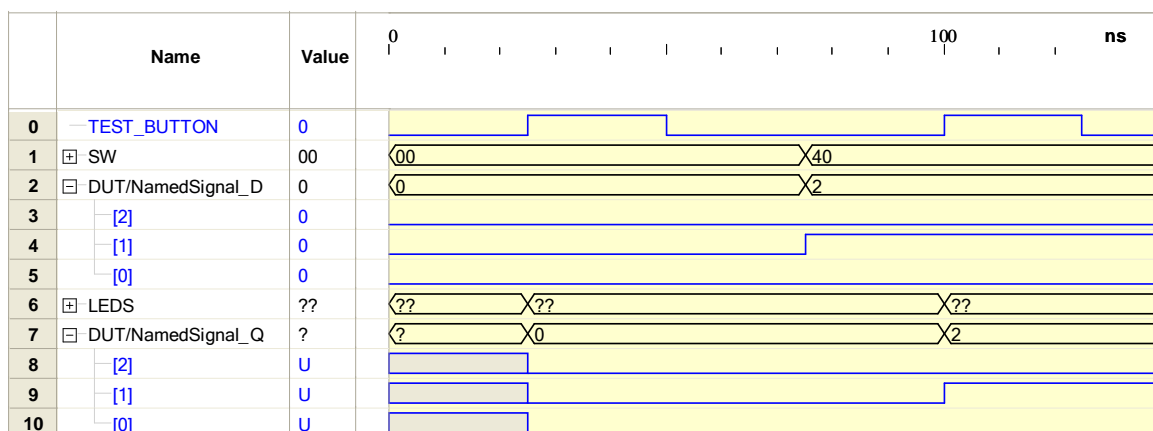


Figure 6: Simulation of the latch

the output is latched is determined by the conditional *if* statement which explicitly looks for a change in state of the clock input and what state it goes to. A good example of an elegant solution using VHDL.

c) Download the design to the NanoBoard and verify that the design operates as predicted in the simulation.

This should be straight forward.

d) Create a new FPGA project to include the latch connected to the decoder. Connect three of the DIP switches to the input of the latch and connect the outputs of the decoder to the LEDs. Simulate to verify correct operation and then download the design to the NanoBoard.

This is just a combination of the two. The schematic is shown in Figure 7. The simulation uses the same test bench

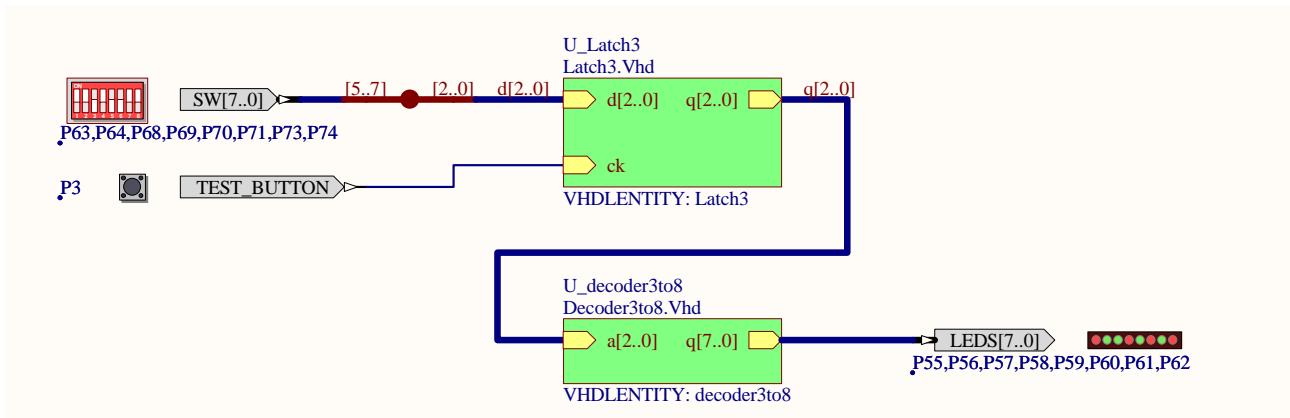


Figure 7: Schematic of the three input latch connected to the decoder

stimulus as that used for the latch. The simulation results is shown in Figure 8. Note that the latch output is undefined

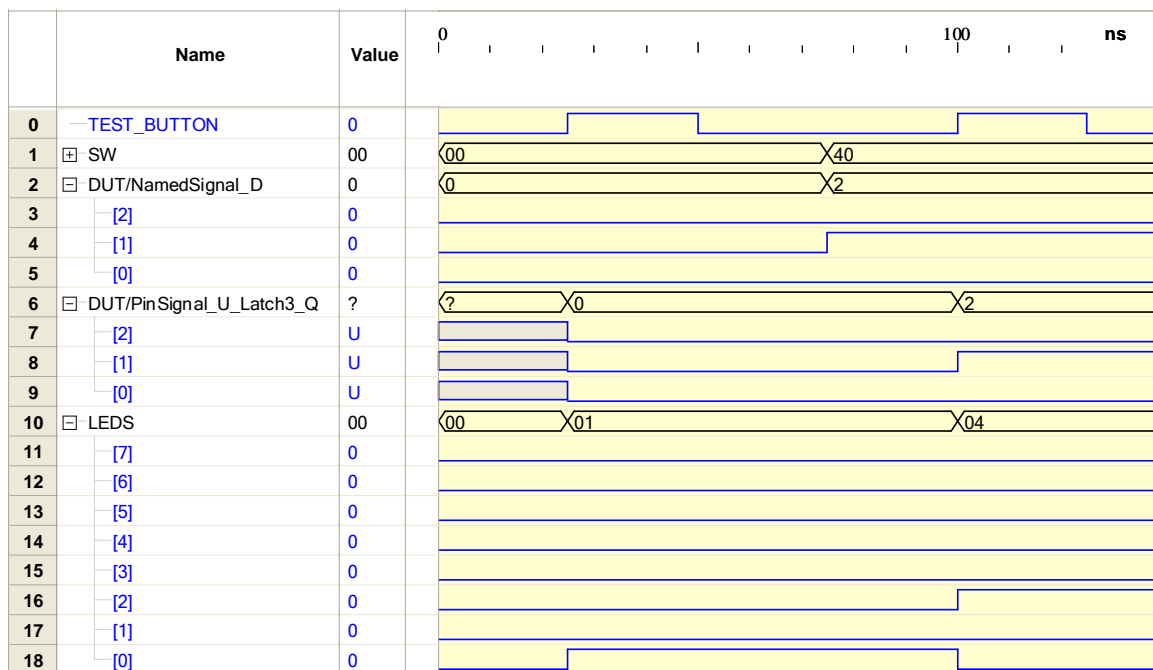


Figure 8: Simulation of the three input latch connected to the decoder

until the first positive edge of the clock input.

4 Bi-Directional Octal Register

This task is optional. Create a new FPGA project and implement the bi-directional octal register as described in lectures. Simulate the design for all possible combinations of direction and output-enable signals. Take care when changing the direction signal as the new output has to be driven to high impedance.

Schematically, the register and some internal detail is shown in Figure 9. This is a good example where a VHDL descrip-

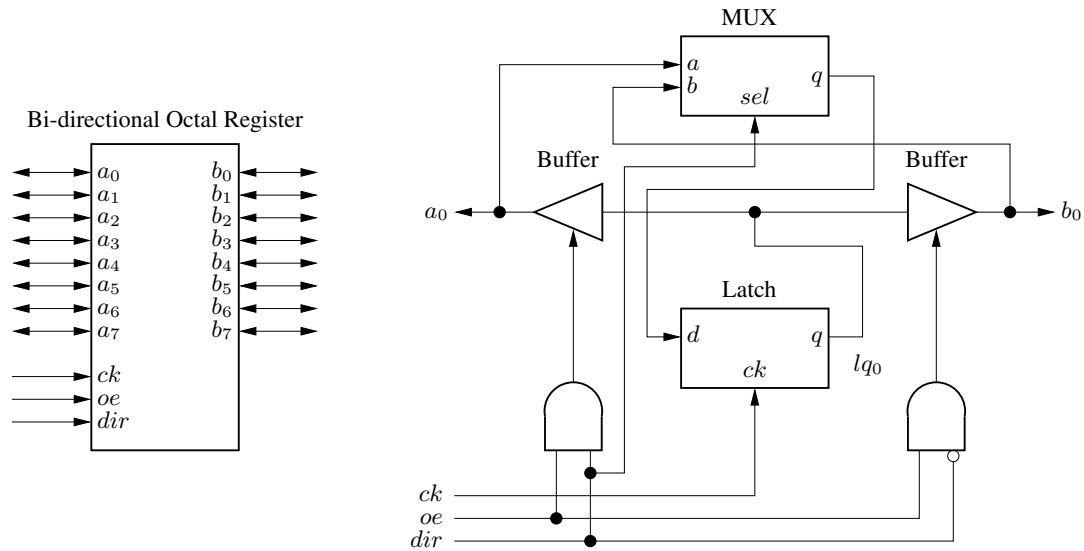


Figure 9: Schematic of the bi-directional octal register at left with some internal detail at right

tion is far more concise and simple than a schematic diagram.

The VHDL code is provided on the ELEC4605 web site to save having to type it in. It is also shown on Page 12.

The schematic from Altium Designer is shown in Figure 10.

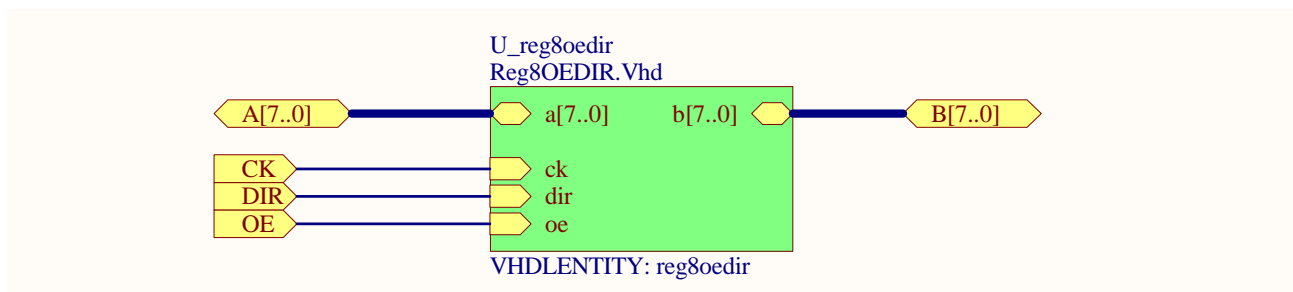


Figure 10: Schematic of the three input latch connected to the decoder

The simulation result is shown in Figure 11 given the test bench on Page 12. This is a good example to try changing what

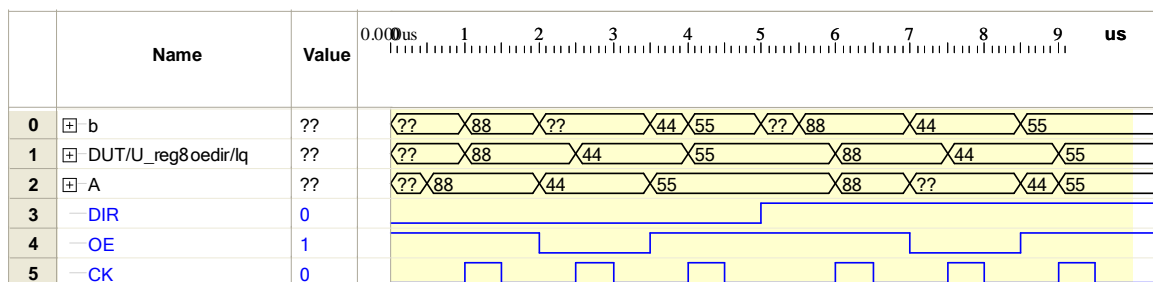


Figure 11: Simulation of the octal register with output enable and direction

signals appear in the process statement sensitivity list.

VHDL Code

Decoder - Example 1

```
— ELEC4605 Computer Engineering
— Decoder3to8.vhd
— Example 1: Using std_logic_unsigned package
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity decoder3to8 is
  port(a : in std_logic_vector(2 downto 0);
        q : out std_logic_vector(7 downto 0));
end decoder3to8;

architecture behaviour of decoder3to8 is
begin
  q(0) <= '1' when a=0 else '0';
  q(1) <= '1' when a=1 else '0';
  q(2) <= '1' when a=2 else '0';
  q(3) <= '1' when a=3 else '0';
  q(4) <= '1' when a=4 else '0';
  q(5) <= '1' when a=5 else '0';
  q(6) <= '1' when a=6 else '0';
  q(7) <= '1' when a=7 else '0';
end behaviour;
```

Decoder Example 2

```
— ELEC4605 Computer Engineering
— Decoder3to8.vhd
— Example 2: Using numeric_std package
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity decoder3to8 is
  port(a : in std_logic_vector(2 downto 0);
        q : out std_logic_vector(7 downto 0));
end decoder3to8;

architecture behaviour of decoder3to8 is
begin
  q(0) <= '1' when unsigned(a)=0 else '0';
  q(1) <= '1' when unsigned(a)=1 else '0';
  q(2) <= '1' when unsigned(a)=2 else '0';
  q(3) <= '1' when unsigned(a)=3 else '0';
  q(4) <= '1' when unsigned(a)=4 else '0';
  q(5) <= '1' when unsigned(a)=5 else '0';
  q(6) <= '1' when unsigned(a)=6 else '0';
  q(7) <= '1' when unsigned(a)=7 else '0';
end behaviour;
```

Decoder Test Bench

```

--- VHDL Testbench for Decoder
--- 2005 3 16 15 12 39
--- Created by "EditVHDL"
--- "Copyright (c) 2002 Altium Limited"

Library IEEE;
Use      IEEE.std_logic_1164.all;
Use      IEEE.std_logic_textio.all;
Use      STD.textio.all;

entity TestDecoder is
end TestDecoder;

architecture stimulus of TestDecoder is
    file RESULTS: TEXT open WRITE_MODE is "results.txt";
    procedure WRITE_RESULTS(
        LEDES: std_logic_vector(7 downto 0);
        SW: std_logic_vector(7 downto 0)
    ) is
        variable l_out : line;
    begin
        write(l_out, now, right, 15);
        write(l_out, LEDES, right, 9);
        write(l_out, SW, right, 9);
        writeline(RESULTS, l_out);
    end procedure;

    component Decoder
        port (
            LEDES: out std_logic_vector(7 downto 0);
            SW: in std_logic_vector(7 downto 0)
        );
    end component;

    signal LEDES: std_logic_vector(7 downto 0);
    signal SW: std_logic_vector(7 downto 0);

begin
    DUT: Decoder port map (
        LEDES => LEDES,
        SW => SW
    );

    STIMULUS0: process
    begin
        -- insert stimulus here
        SW <= b"00000000"; wait for 50ns;
        SW <= b"10000000"; wait for 50ns;
        SW <= b"01000000"; wait for 50ns;
        SW <= b"11000000"; wait for 50ns;
        SW <= b"00100000"; wait for 50ns;
        SW <= b"10100000"; wait for 50ns;
        SW <= b"01100000"; wait for 50ns;
    end process;
end architecture;

```

```
        SW <= b"11100000"; wait for 50ns;
        wait;
    end process;

    WRITE_RESULTS(
        LEADS,
        SW
    );
end architecture;
```

Latch

```

— ELEC4605 Computer Engineering
— Latch3.Vhd
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Latch3 is
  port(ck : in std_logic;
        d  : in std_logic_vector(2 downto 0);
        q  : out std_logic_vector(2 downto 0));
end;

architecture behaviour of Latch3 is
begin
  process (ck)
  begin
    if (ck='1' and ck'event) then
      q <= d;
    end if;
  end process;
end;

```

Latch Test Bench

```

— VHDL Testbench for Latch
— 2005 3 17 11 57 40
— Created by "EditVHDL"
— "Copyright (c) 2002 Altium Limited"

Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_textio.all;
Use STD.textio.all;

entity TestLatch is
end TestLatch;

architecture stimulus of TestLatch is
  file RESULTS: TEXT open WRITE_MODE is "results.txt";
  procedure WRITE_RESULTS(
    LEDS: std_logic_vector(7 downto 0);
    SW: std_logic_vector(7 downto 0);
    TEST_BUTTON: std_logic
  ) is
    variable l_out : line;
  begin
    write(l_out, now, right, 15);
    write(l_out, LEDS, right, 9);

```

```

        write(l_out , SW, right , 9);
        write(l_out , TEST.BUTTON, right , 2);
        writeline(RESULTS, l_out);
    end procedure;

    component Latch
        port (
            LEDES: out std_logic_vector(7 downto 0);
            SW: in std_logic_vector(7 downto 0);
            TEST.BUTTON: in std_logic
        );
    end component;

    signal LEDES: std_logic_vector(7 downto 0);
    signal SW: std_logic_vector(7 downto 0);
    signal TEST.BUTTON: std_logic;

begin
    DUT:Latch port map (
        LEDES => LEDES,
        SW => SW,
        TEST.BUTTON => TEST.BUTTON
    );

    STIMULUS0: process
    begin
        -- insert stimulus here
        TEST.BUTTON <= '0';
        -- Latch b"000"
        SW <= b"00000000"; wait for 25 ns;
        TEST.BUTTON <= '1'; wait for 25 ns;
        TEST.BUTTON <= '0'; wait for 25 ns;
        -- Latch b"010"
        SW <= b"01000000"; wait for 25 ns;
        TEST.BUTTON <= '1'; wait for 25 ns;
        TEST.BUTTON <= '0'; wait for 25 ns;
        wait;
    end process;

    WRITE_RESULTS(
        LEDES,
        SW,
        TEST.BUTTON
    );

end architecture;

```

Bi-directional Octal Register

```

— ELEC4605 Computer Engineering
— Reg8OEDIR.Vhd
— Peter Stepien
— 2005

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity reg8oedir is
  port(ck, oe, dir : in std_logic;
        a          : inout std_logic_vector(7 downto 0);
        b          : inout std_logic_vector(7 downto 0));
end;

architecture behaviour of reg8oedir is
  signal lq : std_logic_vector(7 downto 0);
begin

  Latch: process (ck)
  begin
    if (ck='1' and ck'event) then
      if dir='0' then
        lq <= a;
      elsif dir='1' then
        lq <= b;
      end if;
    end if;
  end process;

  — Output_Enable
  b <= lq when ((oe='1') and (dir='0')) else "ZZZZZZZZ";
  a <= lq when ((oe='1') and (dir='1')) else "ZZZZZZZZ";

end;

```

Bi-directional Octal Register Test Bench

```

— VHDL Testbench for BiDirRegister
— 2005 3 17 13 50 46
— Created by "EditVHDL"
— "Copyright (c) 2002 Altium Limited"

```

```

Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_textio.all;
Use STD.textio.all;

```

```

entity TestBiDirRegister is
end TestBiDirRegister;

```

```

architecture stimulus of TestBiDirRegister is
  file RESULTS: TEXT open WRITE_MODE is "results.txt";
  procedure WRITE_RESULTS(
    A: std_logic_vector(7 downto 0);
    B: std_logic_vector(7 downto 0);
    CK: std_logic;
    DIR: std_logic;
    OE: std_logic
  ) is
    variable l_out : line;
  begin
    write(l_out, now, right, 15);
    write(l_out, A, right, 9);
    write(l_out, B, right, 9);
    write(l_out, CK, right, 2);
    write(l_out, DIR, right, 2);
    write(l_out, OE, right, 2);
    writeline(RESULTS, l_out);
  end procedure;

  component BiDirRegister
    port (
      A: inout std_logic_vector(7 downto 0);
      B: inout std_logic_vector(7 downto 0);
      CK: in std_logic;
      DIR: in std_logic;
      OE: in std_logic
    );
  end component;

  signal A: std_logic_vector(7 downto 0);
  signal B: std_logic_vector(7 downto 0);
  signal CK: std_logic;
  signal DIR: std_logic;
  signal OE: std_logic;

begin
  DUT: BiDirRegister port map (
    A => A,
    B => B,
    CK => CK,
    DIR => DIR,
    OE => OE
  );

  STIMULUS0: process
  begin
    — insert stimulus here
    dir <= '0';
    oe <= '1';
    ck <= '0';
    a <= "ZZZZZZZZ";
    b <= "ZZZZZZZZ";
    wait for 500ns;

    — Latch a value
    a <= b"10001000"; wait for 500ns;
    ck <= '1'; wait for 500ns;
    ck <= '0'; wait for 500ns;

    — Latch another value but disable output first

```

```
oe <= '0';
a <= b"01000100"; wait for 500ns;
ck <= '1'; wait for 500ns;
ck <= '0'; wait for 500ns;
oe <= '1';

-- Latch another value
a <= b"01010101"; wait for 500ns;
ck <= '1'; wait for 500ns;
ck <= '0'; wait for 500ns;

-- Change direction
dir <= '1';
a <= "ZZZZZZZZ";
b <= "ZZZZZZZZ";
wait for 500ns;

-- Latch a value
b <= b"10001000"; wait for 500ns;
ck <= '1'; wait for 500ns;
ck <= '0'; wait for 500ns;

-- Latch another value but disable output first
oe <= '0';
b <= b"01000100"; wait for 500ns;
ck <= '1'; wait for 500ns;
ck <= '0'; wait for 500ns;
oe <= '1';

-- Latch another value
b <= b"01010101"; wait for 500ns;
ck <= '1'; wait for 500ns;
ck <= '0'; wait for 500ns;

    wait;
    end process;

    WRITE_RESULTS(
        A,
        B,
        CK,
        DIR,
        OE
    );

end architecture;
```