Islamic University of Technology

Department of Electrical and Electronic Engineering (EEE)

Digital Electronics and Pulse Techniques(EEE 4484)

Lab-5

Necessary concepts which will be required to perform the examples and exercises:

- ⇨ `Signal`(s) are like physical wires.
- ⇨ `std_logic_vector` is just a vector (or, array/collection) of `std_logic` elements.
- ⇨ `others => '0'` is used to set/initialize all the elements of an array to `'0'`
- ⇨ `clk'event and clk = '1'` is used to indicate the *rising edge* of the clock signal. There is an alternative statement

```
if (rising_edge = clk) then
        …
        …
```

- ⇨ `X"000000"` refers to hexadecimal in VHDL and that consists of 24 bits.

Example-1

4 to 1 Multiplexer:

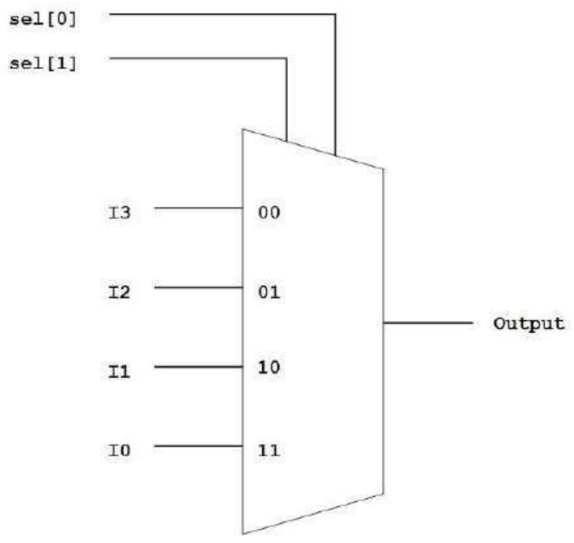


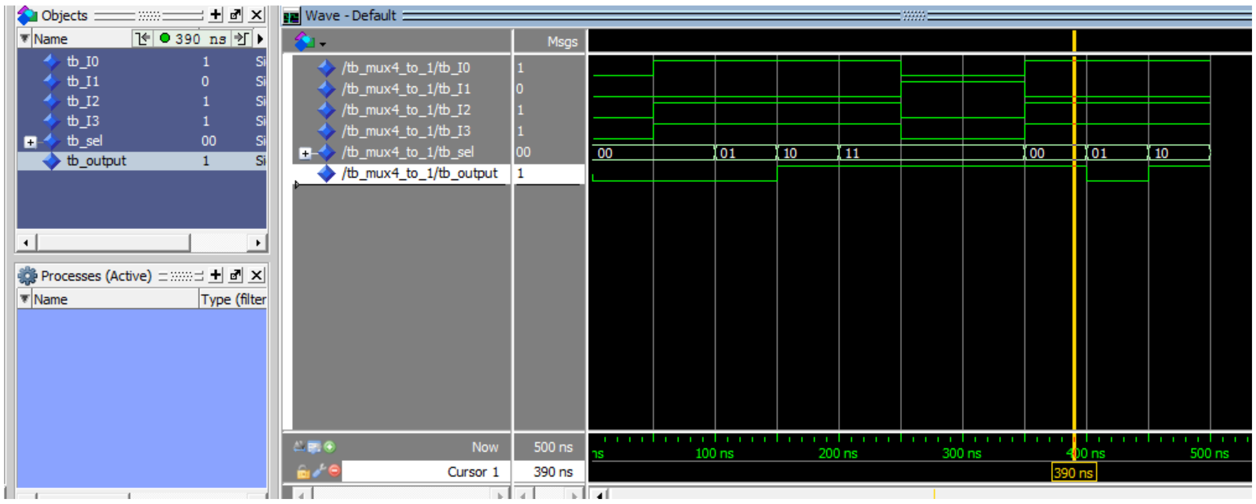
Figure 1. 4 to 1 Multiplexer

Source code:

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_to_1 is

  port (

        I0  : in     std_logic;
        I1  : in     std_logic;
        I2  : in     std_logic;
        I3  : in     std_logic;
        sel : in     std_logic_vector (1 downto 0);
        output : out  std_logic

);

end entity;




architecture behavioral of mux4_to_1 is

begin

        process(sel)        -- process and sensitivity list
        begin

                case sel is

                 when "00" => output <= I0;
                 when "01" => output <= I1;
                 when "10" => output <= I2;
                 when others => output <= I3;

                end case;
        end process;

end architecture behavioral;
```

Test-bench code:

```
library ieee;
use ieee.std_logic_1164.all;


entity tb_mux4_to_1 is
end tb_mux4_to_1;


architecture behavioral of tb_mux4_to_1 is

  component mux4_to_1
    port (
    I0  : in     std_logic;
    I1  : in     std_logic;
    I2  : in     std_logic;
    I3  : in     std_logic;

    sel : in     std_logic_vector (1 downto 0);
    output : out  std_logic
);

  end component;

  signal tb_I0, tb_I1, tb_I2, tb_I3 : std_logic := '0';
  signal tb_sel : std_logic_vector (1 downto 0)  := (others => '0');
  signal tb_output : std_logic;

begin

  uut: mux4_to_1 port map (
    I0 => tb_I0,
    I1 => tb_I1,
    I2 => tb_I2,
    I3 => tb_I3,
    sel => tb_sel,
    output => tb_output
    );

  -- stimulus process

  stim_process: process
  begin
    wait for 50 ns;

    tb_sel <= "00";
    tb_I0 <= '1'; tb_I1 <= '0'; tb_I2 <= '1'; tb_I3 <= '1';

    wait for 50 ns;
    tb_sel <= "01";

    wait for 50 ns;
    tb_sel <= "10";

    wait for 50 ns;
    tb_sel <= "11";

    wait for 50 ns;
    tb_I0 <= '0'; tb_I1 <= '1'; tb_I2 <= '0'; tb_I3 <= '0';
    tb_sel <= "11";

    wait for 50 ns;

  end process;
end architecture behavioral;
```



Name	Value
tb_I0	1
tb_I1	0
tb_I2	1
tb_I3	1
tb_sel	00
tb_output	1

Signal	Msgs
/tb_mux4_to_1/tb_I0	1
/tb_mux4_to_1/tb_I1	0
/tb_mux4_to_1/tb_I2	1
/tb_mux4_to_1/tb_I3	1
/tb_mux4_to_1/tb_sel	00
/tb_mux4_to_1/tb_output	1

Now 500 ns
Cursor 1 390 ns

Figure 2. Simulation result (timing diagram) of the multiplexer testbench

Example-2

D Flip-Flop:

In general, we define a *synchronous sequential circuit*, or just *sequential circuit* as a circuit with m inputs, n outputs, and a distinguished *clock* input.

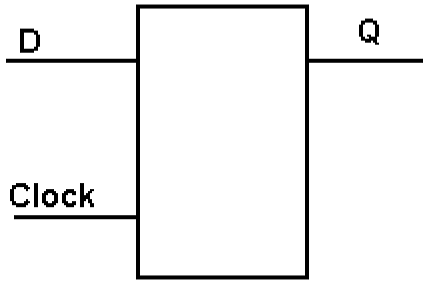


Figure 3. D Flip-Flop

Source code:

```
library ieee;
use ieee.std_logic_1164.all;


entity d_flipflop is
  port (
    clk : in   std_logic;
    D   : in   std_logic;
    Q   : out  std_logic
);
end entity;


architecture behavioral of d_flipflop is

begin
  process (clk, D)
  begin
    if (clk'event and clk = '1') then
      Q <= D;
    end if;
  end process;
end architecture behavioral;
```

Test-bench code:

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_d_flipflop is
end tb_d_flipflop;

architecture behavioral of tb_d_flipflop is

  component d_flipflop
     port (
        clk : in    std_logic;
        D   : in    std_logic;
        Q   : out    std_logic
);
  end component;

  signal clk     : std_logic;
  signal tb_D    : std_logic;
  signal tb_Q    : std_logic;

  constant clk_period : time := 100 ns; -- period of the clock

begin

  uut: d_flipflop port map(
    clk => clk,
    D => tb_D,
    Q => tb_Q
);
  -- clock process
  clk_process: process
  begin
    clk <= '0';
    wait for clk_period/2;

    clk <= '1';
    wait for clk_period/2;
  end process;


-- stimulus process
  stim_process: process
  begin
    wait for 50 ns;
    tb_D <= '0';

    wait for 50 ns;
    tb_D <= '1';

    wait for 50 ns;
    tb_D <= '0';

  end process;


end architecture behavioral;
```

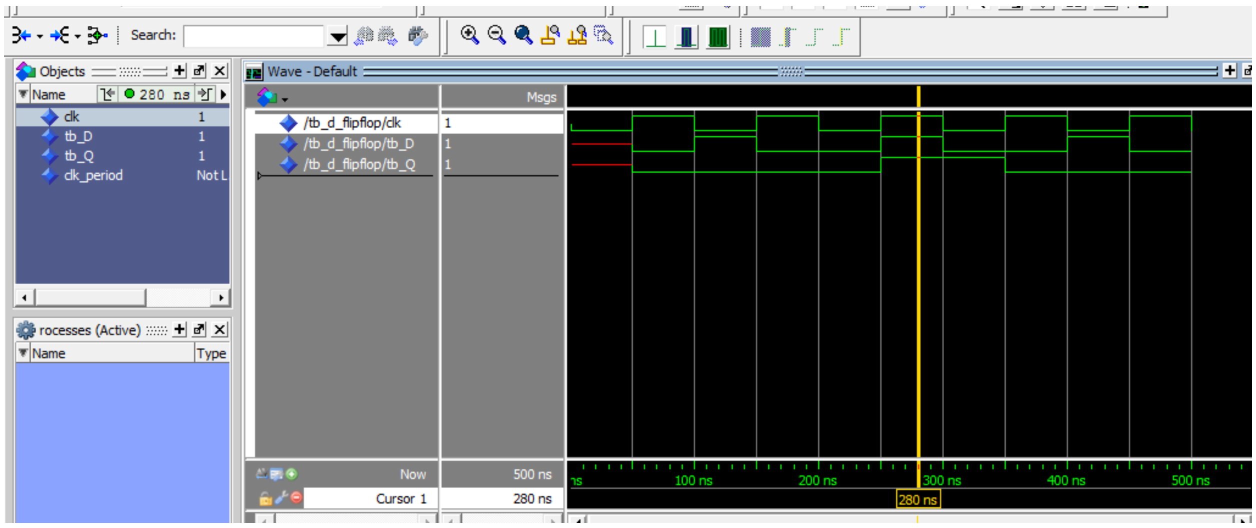
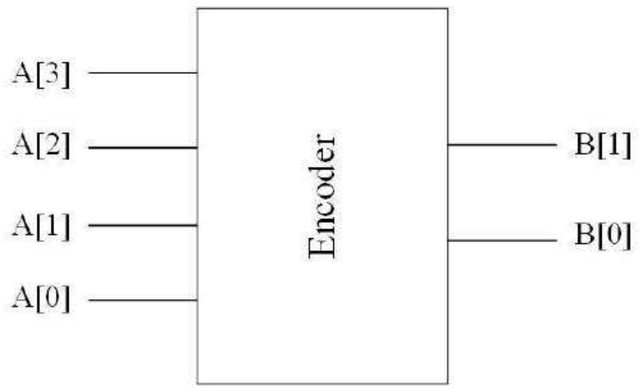


Figure 4. Simulation result (timing diagram) of the D-flipflop testbench

Example-3

A binary encoder has 2n input lines and n output lines, hence it encodes the information from 2n inputs into an n-bit code. From all the input lines, only one of an input line is activated at a time, and depending on the input line, it produces the n bit output code.

Depending on the number of input lines, digital or binary encoders produce the output codes in the form of 2 or 3 or 4 bit codes.



(a)

Input				**Output**	
A[3]	A[2]	A[1]	A[0]	B[1]	B[0]
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(b)

Figure 5. 4 to 2 Encoder (a) Block Diagram (b) Truth Table

Source code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity encoder is
 port(
               A: in STD_LOGIC_VECTOR(3 downto 0);
               B: out STD_LOGIC_VECTOR(1 downto 0)
 );
end encoder;

architecture behavioral of encoder is
begin

process(a)
begin
        if (A = "1000") then
                B <= "00";
        elsif (A = "0100") then
                B <= "01";
        elsif (A = "0010") then
```

```
                B <= "10";
        elsif (A = "0001") then
                B <= "11";
        else
                B <= "ZZ";
        end if;
end process;

end architecture behavioral;
```

Test-bench code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb_encoder IS
END tb_encoder;

ARCHITECTURE behavioral OF tb_encoder IS

-- Component Declaration for the Unit Under Test (UUT)

COMPONENT encoder
PORT(
        A : IN std_logic_vector(3 downto 0);
        B : OUT std_logic_vector(1 downto 0)
);
END COMPONENT;

--Inputs
signal tb_A : std_logic_vector(3 downto 0) := (others => '0');
--Outputs
signal tb_B : std_logic_vector(1 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: encoder PORT MAP (
        A => tb_A,
        B => tb_B
);

-- Stimulus process
stim_proc: process
begin
-- hold reset state for 100 ns.
        wait for 100 ns;
        tb_A <= "0000";

        wait for 100 ns;
        tb_A <= "0001";

        wait for 100 ns;
        tb_A <= "0010";

        wait for 100 ns;
        tb_A <= "0100";

        wait for 100 ns;
        tb_A <= "1000";

        wait;
end process;

END architecture behavioral;
```

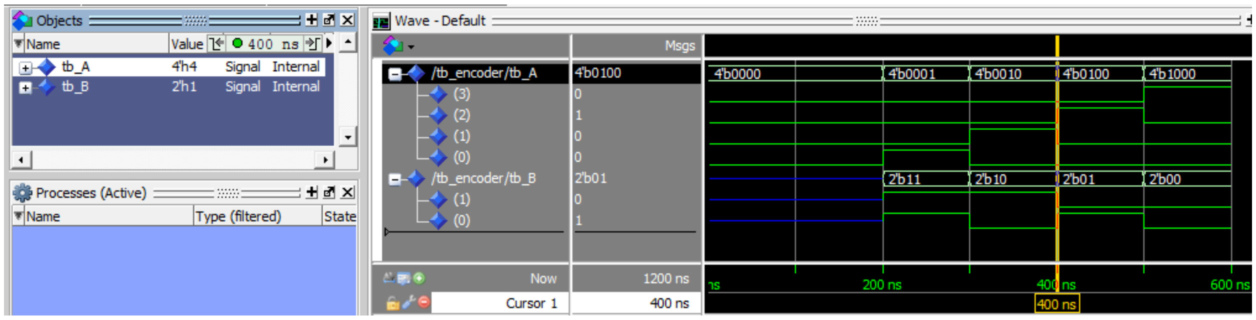


Figure 6. Simulation result (timing diagram) of the 4 to 2 Encoder testbench

Home tasks (Implement in VHDL. Compile and simulate with Modelsim)

1. Design a full-adder circuit. Then simulate and verify with testbench.
2. Design a 1 to 8 demultiplexer. Then simulate and verify with testbench.
3. Design a 4-bit Arithmetic Logic Unit (ALU) which can perform addition, subtraction, AND, OR operation between two numbers. Try to simulate with testbench.
4. Design a synchronous 5-bit shift register (use a input clock for synchronization). Then simulate and verify with testbench providing different combinations.