

Practice Exercises

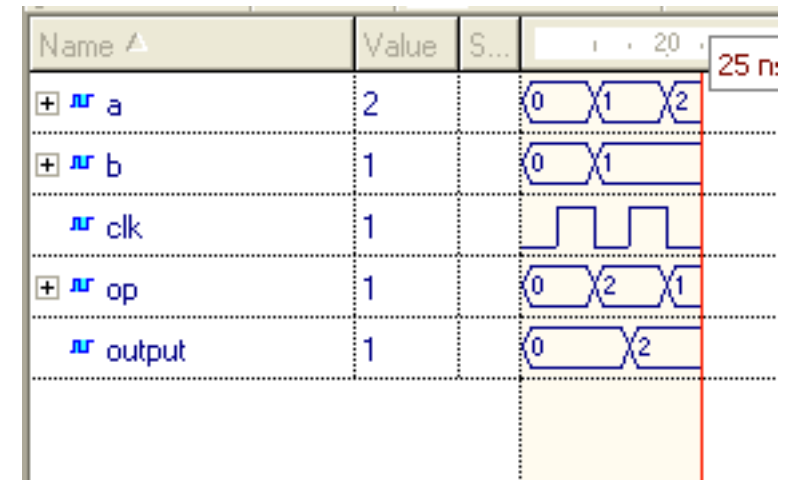
Topic #04 - b) Types and arrays

Exercise #1 - Arithmetic

- Create a circuit in VHDL that has in 3 inputs (**a**, **b** and **op**) and a single output (**output**). You may add a **CLK** signal if you want.
 - **a** , **b** and **op** are 2 bit-buses of type **std_logic_vector**
 - **output** is of type **integer**
 - If **op** has the value “10” then add **a** and **b** and send the results to output
 - If **op** has the value “01” then subtract **a** from **b** and send the results to **output**
 - For any other **op** value, **output** should be 0
- • Test your work with a test-bench!

Warnings on exercise #1

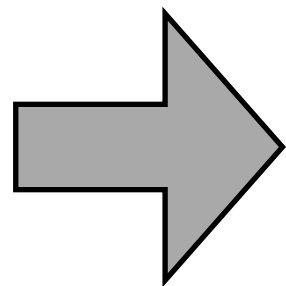
- While we have studied **processes**, you have seen them in homework assignments and it's easier to get things done with them.
- Inside **processes** everything is done sequentially, that means your output will probably not be displayed exactly when your stimulus is triggered.
- Bottom line... you may want to add a clock to make things easier



Name	Value	S...	20	25 ns
a	2		0 1 2	
b	1		0 1	
clk	1			
op	1		0 2 1	
output	1		0 2	

```
entity exercisel is
  port (a,b,op : in std_logic_vector(1 downto 0);
        clk : in bit;
        output : out integer
  );
```

Main program



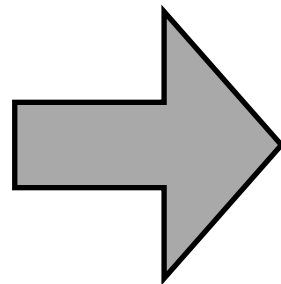
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

entity exercisel is
    port (a,b,op : in std_logic_vector(1 downto 0);
          clk : in bit;
          output : out integer
    );
end entity;

architecture myarch of exercisel is
    signal oo : std_logic_vector (1 downto 0):="00";
begin
    process (clk)
    begin
        if (op="10") then
            oo <= a + b;
        elsif (op="01") then
            oo <= a - b;
        else
            oo <= "00";
        end if;
        output<=conv_integer(oo);
    end process;

end architecture;
```

Test-bench



```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- extra package included
USE ieee.std_logic_unsigned.all;

entity testexercisel is
end;

architecture bench of testexercisel is
    component exercisel
        port (a,b,op : in std_logic_vector(1 downto 0);
              clk : in bit;
              output : out integer
              );
    end component;

    signal a,b,op : std_logic_vector (1 downto 0);
    signal clk : bit;
    signal output : integer;

begin
    a    <= "00", "01" after 10 ns , "10" after 20 ns;
    b    <= "00", "01" after 10 ns , "01" after 20 ns;
    op   <= "00", "10" after 10 ns , "01" after 20 ns;
    clk  <= '0' , '1' after 5 ns,'0' after 10 ns
          , '1' after 15 ns,'0' after 20 ns,'1' after 25 ns;

    m: exercisel port map (a,b,op,clk,output);

end bench;

```

Name ^	Value	S...	20	25 ns
<input checked="" type="checkbox"/> a	2		0 1 2	
<input checked="" type="checkbox"/> b	1		0 1	
<input checked="" type="checkbox"/> clk	1			
<input checked="" type="checkbox"/> op	1		0 2 1	
<input checked="" type="checkbox"/> output	1		0 2	

Exercise #2 - Why legal?

- Look at the following assignments.
- Why are they legal?

```
x(0) <= y(1)(2);
```

```
x(1) <= v(2)(3);
```

```
x(2) <= w(2,1);
```

```
y(1)(1) <= x(6);
```

```
y(2)(0) <= v(0)(0);
```

```
y(0)(0) <= w(3,3);
```

```
w(1,1) <= x(7);
```

```
w(3,0) <= v(0)(3);
```

```
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0);
```

```
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0);
```

```
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0);
```

```
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0);
```

where,

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
```

```
TYPE array1 IS ARRAY (0 TO 3) OF row;
```

```
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
```

```
SIGNAL x: row;
```

```
SIGNAL y: array1;
```

```
SIGNAL v: array2;
```

```
SIGNAL w: array3;
```

-- 1D array

-- 1Dx1D array

-- 1Dx1D

-- 2D array


```

----- Legal scalar assignments: -----
-- The scalar (single bit) assignments below are all legal,
-- because the "base" (scalar) type is STD_LOGIC for all signals
-- (x,y,v,w).
x(0) <= y(1)(2);      -- notice two pairs of parenthesis
                      -- (y is 1Dx1D)
x(1) <= v(2)(3);      -- two pairs of parenthesis (v is 1Dx1D)
x(2) <= w(2,1);       -- a single pair of parenthesis (w is 2D)
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
----- Vector assignments: -----
x <= y(0);            -- legal (same data types: ROW)
x <= v(1);            -- illegal (type mismatch: ROW x
                      -- STD_LOGIC_VECTOR)
x <= w(2);            -- illegal (w must have 2D index)
x <= w(2, 2 DOWNT0 0); -- illegal (type mismatch: ROW x
                      -- STD_LOGIC)
v(0) <= w(2, 2 DOWNT0 0); -- illegal (mismatch: STD_LOGIC_VECTOR
                      -- x STD_LOGIC)
v(0) <= w(2);         -- illegal (w must have 2D index)
y(1) <= v(3);         -- illegal (type mismatch: ROW x
                      -- STD_LOGIC_VECTOR)
y(1)(7 DOWNT0 3) <= x(4 DOWNT0 0); -- legal (same type,
                      -- same size)
v(1)(7 DOWNT0 3) <= v(2)(4 DOWNT0 0); -- legal (same type,
                      -- same size)
w(1, 5 DOWNT0 1) <= v(2)(4 DOWNT0 0); -- illegal (type mismatch)

```

Exercise #3 - Why illegal?

- Look at the following assignments.
- Why are they illegal?

```
x <= v(1);
```

```
x <= w(2);
```

```
x <= w(2, 2 DOWNT0 0);
```

```
v(0) <= w(2, 2 DOWNT0 0);
```

```
v(0) <= w(2);
```

```
y(1) <= v(3);
```

```
w(1, 5 DOWNT0 1) <= v(2)(4 DOWNT0 0);
```

where,

```
TYPE row IS ARRAY (7 DOWNT0 0) OF STD_LOGIC;
```

-- 1D array

```
TYPE array1 IS ARRAY (0 TO 3) OF row;
```

-- 1Dx1D array

```
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNT0 0);
```

-- 1Dx1D

```
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNT0 0) OF STD_LOGIC;
```

-- 2D array

```
SIGNAL x: row;
```

```
SIGNAL y: array1;
```

```
SIGNAL v: array2;
```

```
SIGNAL w: array3;
```



```

----- Legal scalar assignments: -----
-- The scalar (single bit) assignments below are all legal,
-- because the "base" (scalar) type is STD_LOGIC for all signals
-- (x,y,v,w).
x(0) <= y(1)(2);      -- notice two pairs of parenthesis
                      -- (y is 1Dx1D)
x(1) <= v(2)(3);      -- two pairs of parenthesis (v is 1Dx1D)
x(2) <= w(2,1);       -- a single pair of parenthesis (w is 2D)
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
----- Vector assignments: -----
x <= y(0);            -- legal (same data types: ROW)
x <= v(1);            -- illegal (type mismatch: ROW x
                      -- STD_LOGIC_VECTOR)
x <= w(2);            -- illegal (w must have 2D index)
x <= w(2, 2 DOWNT0 0); -- illegal (type mismatch: ROW x
                      -- STD_LOGIC)
v(0) <= w(2, 2 DOWNT0 0); -- illegal (mismatch: STD_LOGIC_VECTOR
                      -- x STD_LOGIC)
v(0) <= w(2);         -- illegal (w must have 2D index)
y(1) <= v(3);         -- illegal (type mismatch: ROW x
                      -- STD_LOGIC_VECTOR)
y(1)(7 DOWNT0 3) <= x(4 DOWNT0 0); -- legal (same type,
                      -- same size)
v(1)(7 DOWNT0 3) <= v(2)(4 DOWNT0 0); -- legal (same type,
                      -- same size)
w(1, 5 DOWNT0 1) <= v(2)(4 DOWNT0 0); -- illegal (type mismatch)

```

Exercise #4 - Parity encoder

- A parity bit is a bit that is added to ensure that the number of bits with the value “one” in a set of bits is even or odd
- Parity bits are used as the simplest form of error detecting code
- If the number of “ones” is even the parity bit will be 0
- I challenge you to write a program that will find the parity bit for a 4-bit bus of type `std_logic_vector`
- Use the following entity:

```
entity parity_encoder is
    port (input_bus   : in std_logic_vector(3 downto 0);
          output_bus  : out std_logic_vector(4 downto 0)
        );
end entity;
```

Solution #4 - Parity encoder

See HW #5 solutions

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- extra package included
USE ieee.std_logic_unsigned.all;

entity test_parity_encoder is
end;

architecture bench of test_parity_encoder is
    component parity_encoder
        port (input_bus  : in std_logic_vector(3 downto 0);
              output_bus : out std_logic_vector(4 downto 0)
              );
    end component;

    signal input_bus : std_logic_vector(3 downto 0);
    signal output_bus : std_logic_vector(4 downto 0);

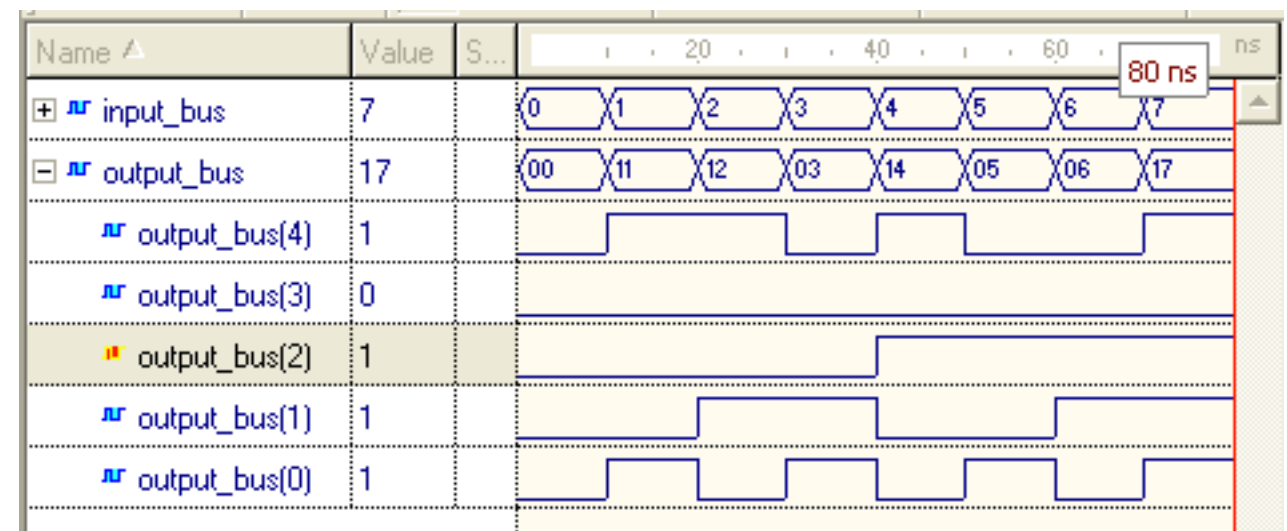
```

← Test-bench

```

begin
    input_bus <= "0000", -- even
    "0001" after 10 ns, -- odd
    "0010" after 20 ns, -- odd
    "0011" after 30 ns, -- even
    "0100" after 40 ns, -- odd
    "0101" after 50 ns, -- even
    "0110" after 60 ns, -- even
    "0111" after 70 ns; -- odd

```



```

    m: parity_encoder port map (input_bus, output_bus);

```

```

end bench;

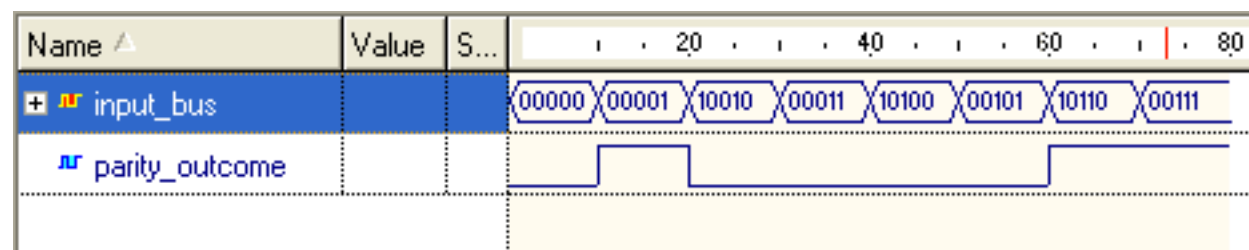
```

Exercise #5 - Parity decoder

- Create another VHDL program that will tell if the parity of a 4bit bus is correct or not.
- Use the test-bench on the course website and the following entity:

```
entity parity_decoder is
    port (input_bus  : in std_logic_vector(4 downto 0);
          parity_outcome : out std_logic
    );
end entity;
```

- So you can check your answer, here is the output after 80ns



Solution #5 - Parity decoder

See HW #5 solutions


```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- extra package included
USE ieee.std_logic_unsigned.all;

entity test_parity_decoder is
end;

architecture bench of test_parity_decoder is
    component parity_decoder
        port (input_bus  : in std_logic_vector(4 downto 0);
              parity_outcome : out std_logic
            );
    end component;

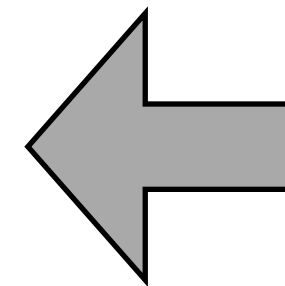
    signal input_bus : std_logic_vector(4 downto 0);
    signal parity_outcome : std_logic;

begin
    input_bus  <= "00000", -- even (correct parity)
    "00001" after 10 ns, -- odd (incorrect parity)
    "10010" after 20 ns, -- odd (correct parity)
    "00011" after 30 ns, -- even (correct parity)
    "10100" after 40 ns, -- odd (correct parity)
    "00101" after 50 ns, -- even (correct parity)
    "10110" after 60 ns, -- even (incorrect parity)
    "00111" after 70 ns; -- odd (incorrect parity)

    m: parity_decoder port map (input_bus,
    parity_outcome);

end bench;

```



Test-bench

