CPE 462
VHDL: Simulation and Synthesis

Topic #04 - b) Types and arrays

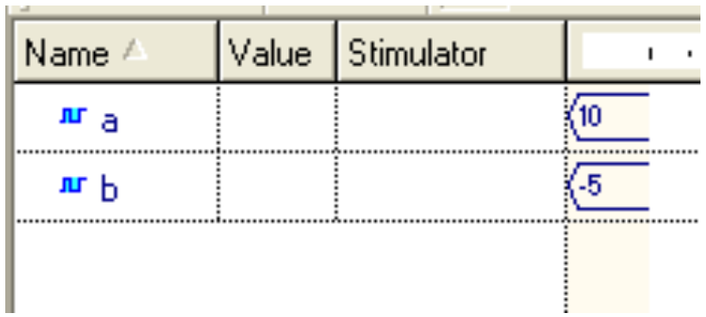
Types

User defined data-types (integer)

- In VHDL we can define our own data types amongst integers and enumerations
- For example, I want an integer that can only take values from -32 to 32

```
type my_integer is range -32 to 32
```

Full source code example on how you would use it...




```
entity test is
end entity;

architecture myarch of test is
    type my_integer is range -32 to 32;
    signal a : my_integer;
    signal b : my_integer;
begin

    a <= 10;
    b <= -5;

end architecture;
```

This will not compile

```
entity test is
end entity;

architecture myarch of test is
    type my_integer is range -32 to 32;
    signal a : my_integer;
    signal b : my_integer;
begin

    a <= 10;
    b <= -50;

end architecture;
```

```
acom -work test $DSN/src/main.vhd
# Compile...
# File: c:\cpe462\test\test\src\main.vhd
# Compile Entity "test"
# Compile Architecture "myarch" of Entity "test"
# Error: COMP96_0368: main.vhd : (10, 7): Value -50 out of range.
# Compile failure 1 Errors 0 Warnings  Analysis time :  0.0 [s]
>
```

This will also not compile

```
1   entity test is
2   end entity;
3
4   architecture myarch of test is
5       type myrange is range 0 to 10;
6       signal a : myrange := 10;
7       signal b : myrange := 5;
8       signal c : myrange;
9   begin
10
11      c <= a AND b;
12
13  end architecture;
14
15
```

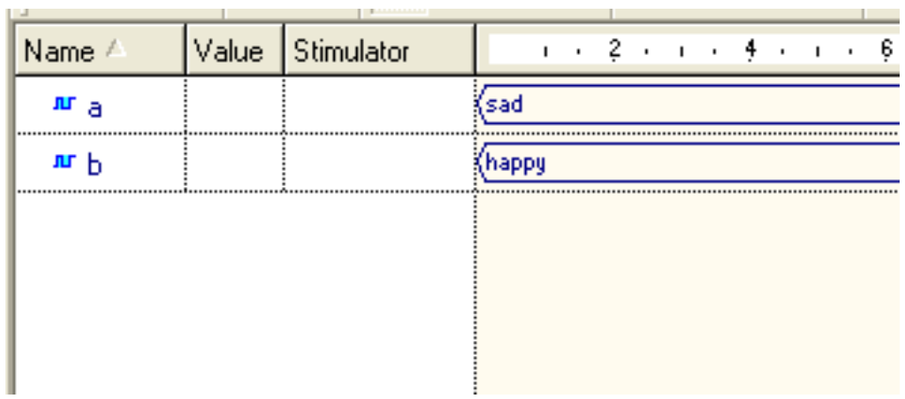
You can't have logic operations on integer user defined data-types

User defined data-types (enumerated)

- For example, I want an enumeration of the mood of my robot
- The robot can only be "confused" , "sad" or "happy"

```
type robot_mood is (confused, sad, happy)
```

Full source code example on how you would use it...




```
entity test is
end entity;

architecture myarch of test is
    type robot_mood is (confused, happy, sad);
    signal a : robot_mood;
    signal b : robot_mood;
begin

    a <= sad;
    b <= happy;

end architecture;
```

Why do we care about enumerated data-types?

- It may make your code a lot more readable.

  For example:

- Useful to describe finite state machines:

  - TYPE state IS (idle, forward, back, stop);

- Useful to describe colors:

  - TYPE color IS (red, green, blue);

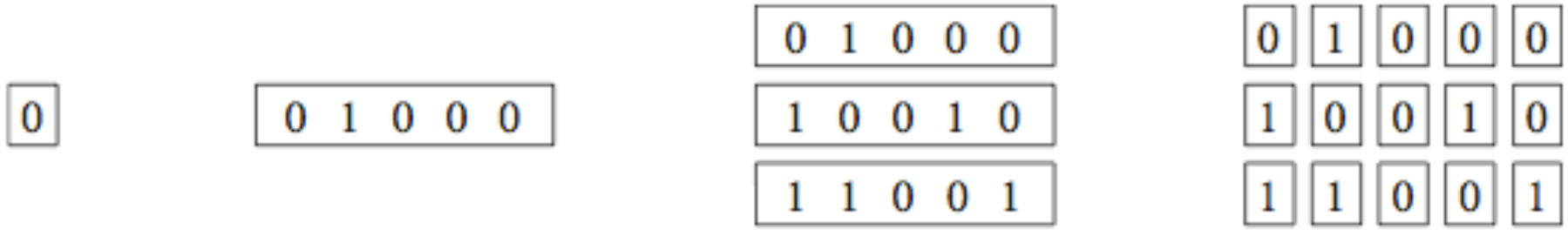
You can't perform logic operations with enumerations

```
1   entity test is
2   end entity;
3
4   architecture myarch of test is
5       type color is (red, green, blue);
6       signal colorA : color;
7       signal colorB : color;
8       signal mixedColor : color;
9   begin
10
11      mixedColor <= colorA AND colorB;
12
13  end architecture;
14
15
```

Arrays

Structure of arrays

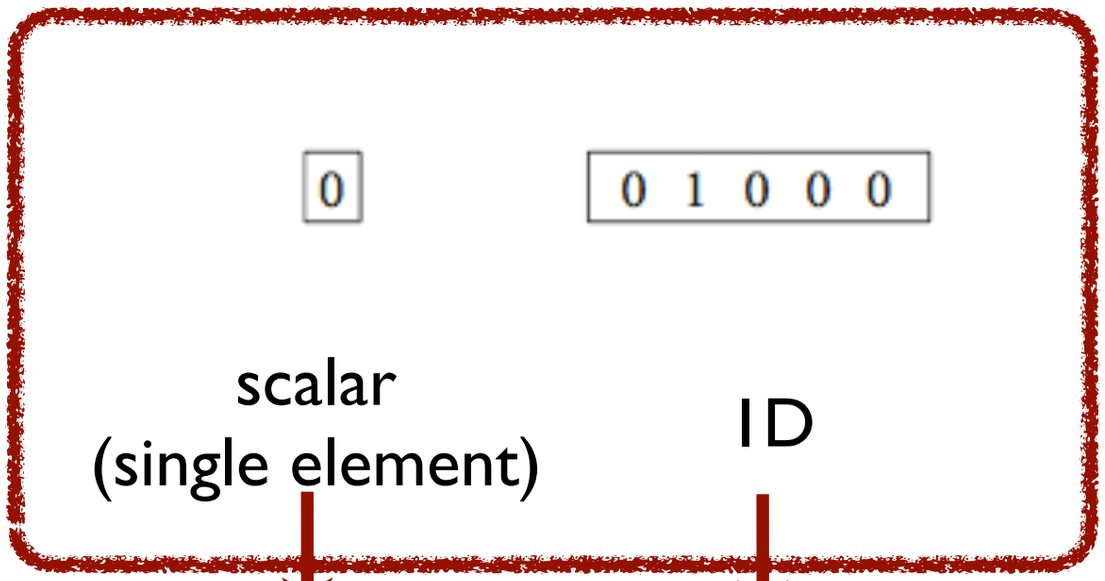
- Arrays are collections of objects of the same type
- They can be one-dimensional (1D), two-dimensional (2D), or one-dimensional-by-one-dimensional (1Dx1D)

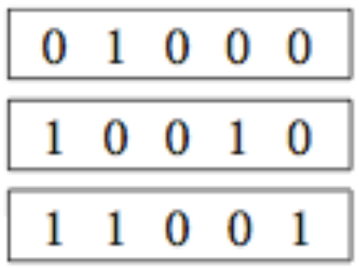


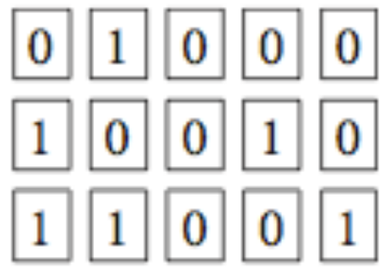
scalar (single element) | 1D | 1D x 1D | 2D

Structure of pre-defined data types

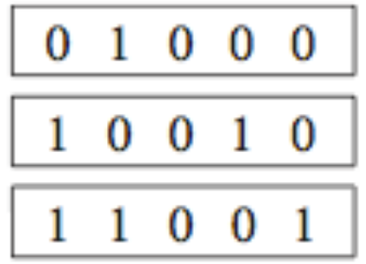
The pre-defined VHDL data types include only the scalar (single bit) and vector (one-dimensional array of bits) categories.







Creating a 1D x 1D array



- 1D x 1D arrays are useful to store groups of bits (e.g. a ROM)
- In order to create an array we need two steps

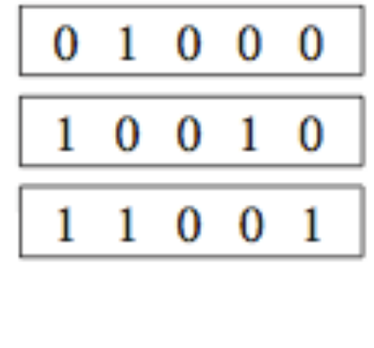
To specify a new array type:

```
TYPE type_name IS ARRAY (specification) OF data_type;
```

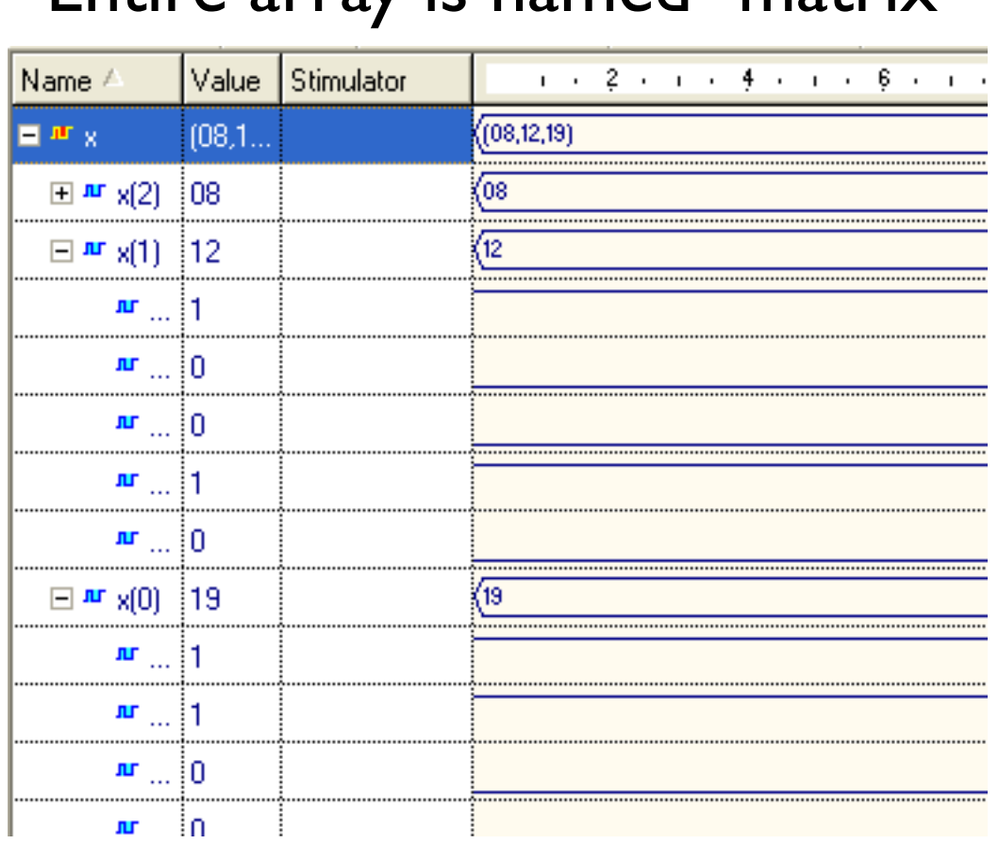
To make use of the new array type:

```
SIGNAL signal_name: type_name [:= initial_value];
```

1D x 1D array example

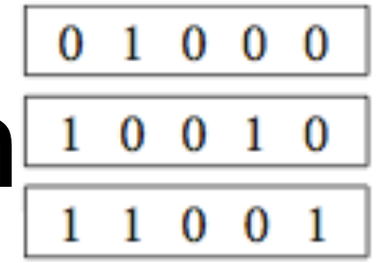


- We want array with four vectors, each of size eight bits
- Each vector is named “row”
- Entire array is named “matrix”




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

entity test is
end entity;

architecture myarch of test is

-- 1D array
type row is array (4 downto 0) of std_logic;

-- 1Dx1D array
type matrix is array (2 downto 0) of row;

-- 1Dx1D signal
signal x: matrix;

begin

    x(0) <= "11001";
    x(1) <= "10010";
    x(2) <= "01000";

end architecture;
```

1D x 1D Array simplification

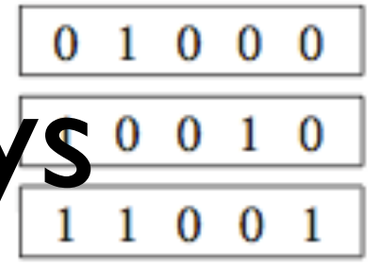



```
library ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix; -- 1Dx1D signal
begin

    x(0) <= "11001";
    x(1) <= "10010";
    x(2) <= "01000";

end architecture;
```

number of rows

number of elements

Another way to initialize arrays

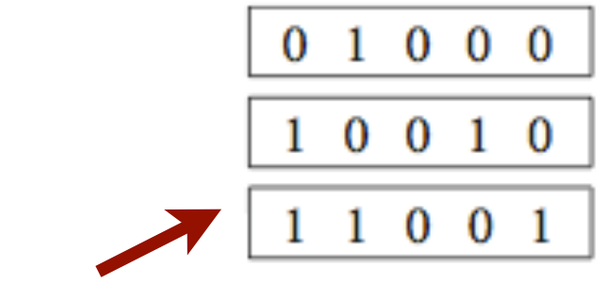



```
library ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( "01000", "10010", "11001");
begin

    --x(0) <= "11001";
    --x(1) <= "10010";
    --x(2) <= "01000";

end architecture;
```

this is how you do it...

all this is commented out!

Even another way

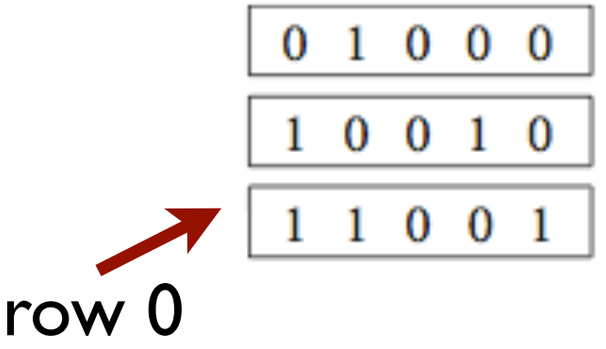



```
library ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( ('0','1','0','0','0'), "10010", "11001");
begin

    --x(0) <= "11001";
    --x(1) <= "10010";
    --x(2) <= "01000";

end architecture;
```

this is how you do it...

all this is commented out!

Constant array

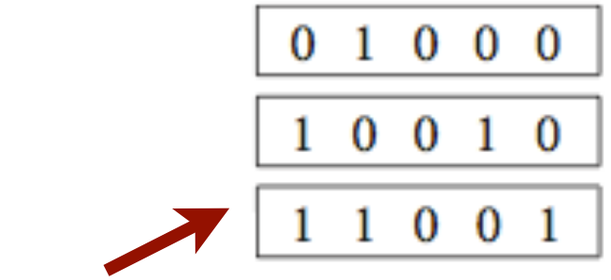



```
library ieee;
use ieee.std_logic_1164.all;

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
constant x: matrix := ( "01000", "10010", "11001");
begin

    --x(0) <= "11001";
    --x(1) <= "10010";
    --x(2) <= "01000";

end architecture;
```

if you want unmodifiable data in your array, replace **signal** with **constant**

all this is commented out!

Lets get some data




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

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( "01000", "10010", "11001");

signal y : std_logic_vector(4 downto 0);

begin

    y <= x(1);

end architecture;
```

creating a new signal bus

assigning one matrix row to to this new bus

Now with input stimulus!

0	1	0	0	0
1	0	0	1	0
1	1	0	0	1

row 0

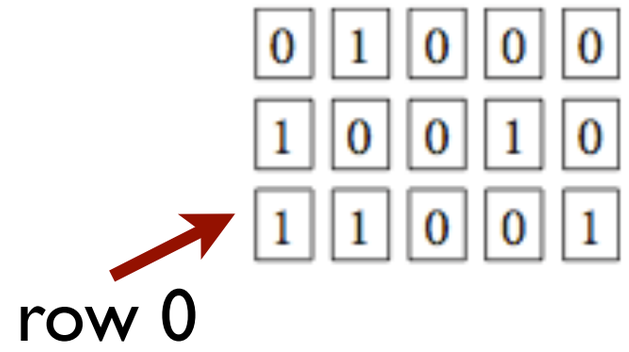
```
library ieee;
use ieee.std_logic_1164.all;

entity test is
    port (a      : in std_logic_vector(1 downto 0);
          output : out std_logic_vector(4 downto 0));
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( "01000", "10010", "11001");

begin

    process(a)
    begin
        if (a="00") then output <= x(0);
        elsif (a="01") then output <= x(1);
        elsif (a="10") then output <= x(2);
        else output <= "ZZZZZ";
        end if;
    end process;

end architecture;
```

We haven't studied in detail "processes" and "if statements"... but this is an example!

I could have added a logic gates forming a MUX instead

Name	Value	Stimulator	20	40 ns
a	0	Binary Counter	0 1 2 3	
a(1)	0			
a(0)	0			
output	19		19 12 08 ZZ	
output(4)	**1**			
output(3)	1			
output(2)	0			
output(1)	0			
output(0)	1			

2D Array



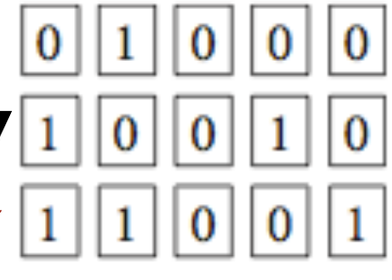
- Its construction is not based on vectors, but rather entirely on scalars

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

entity test is
end entity;

architecture myarch of test is
type matrix2D is array (0 TO 2, 4 DOWNTO 0) of std_logic;
signal x : matrix2D := (
('1','1','0','0','1'),
('1','0','0','1','0'),
('0','1','0','0','0')
);
begin

end architecture;
```

# of cells

# of rows

Extracting data on 2D Array




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

entity test is
end entity;

architecture myarch of test is
type matrix2D is array (0 TO 2, 4 DOWNTO 0) of std_logic;
signal x : matrix2D := (
('1','1','0','0','1'),
('1','0','0','1','0'),
('0','1','0','0','0')
);

signal y : std_logic;

begin

    y <= x(2,4);

end architecture;
```

creating a new scalar

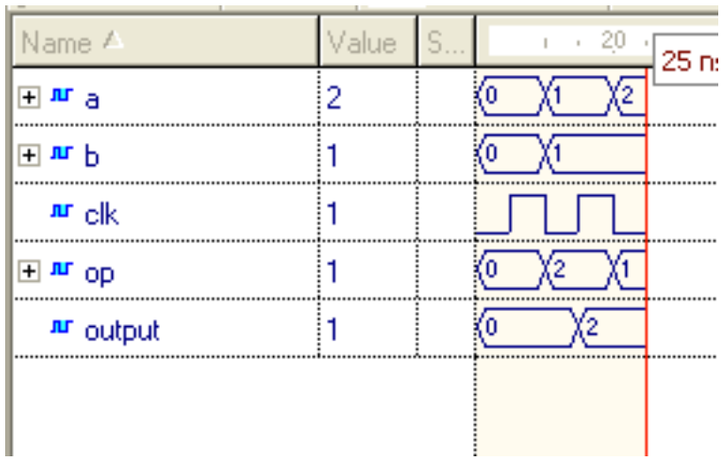
(row, column)

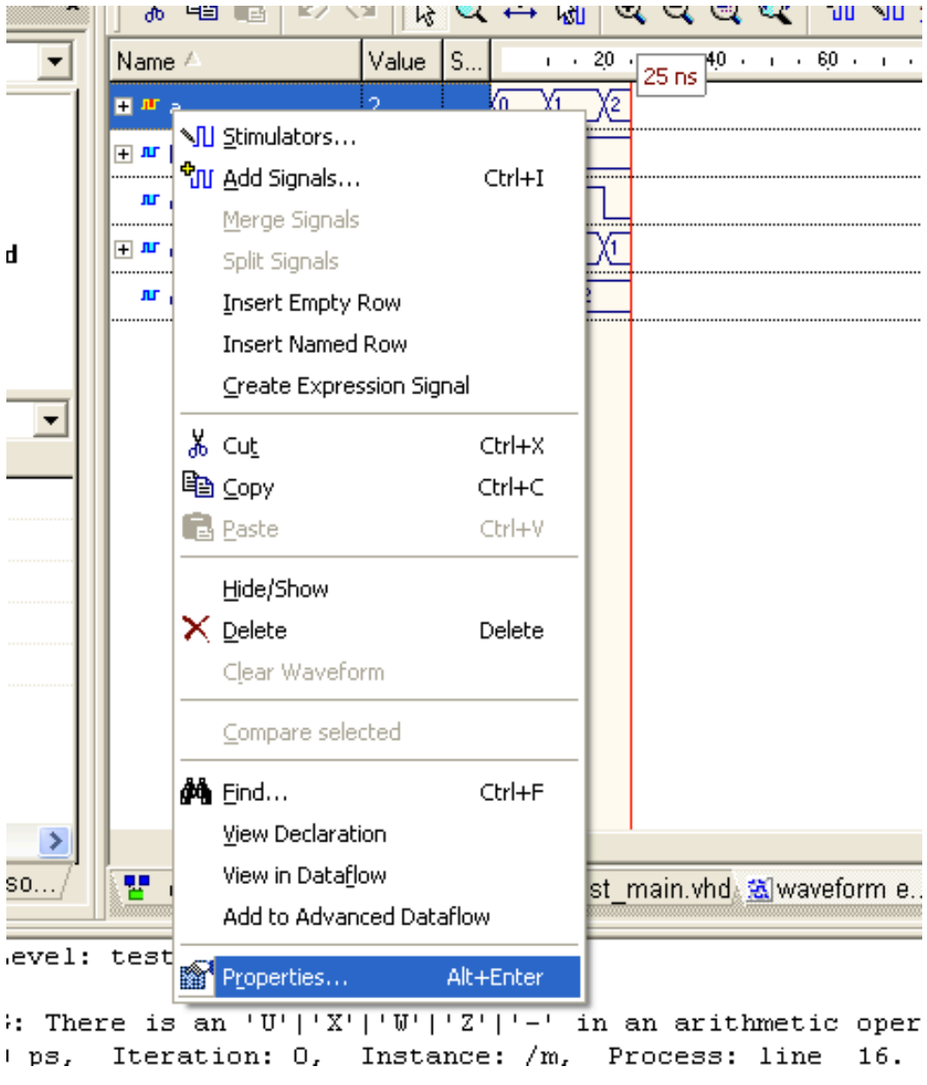
Name	Value	S...
x	6648	
x(0,4)	1	
x(0,3)	1	
x(0,2)	0	
x(0,1)	0	
x(0,0)	1	
x(1,4)	1	
x(1,3)	0	
x(1,2)	0	
x(1,1)	1	
x(1,0)	0	
x(2,4)	0	
x(2,3)	1	
x(2,2)	0	
x(2,1)	0	
x(2,0)	0	
y	0	

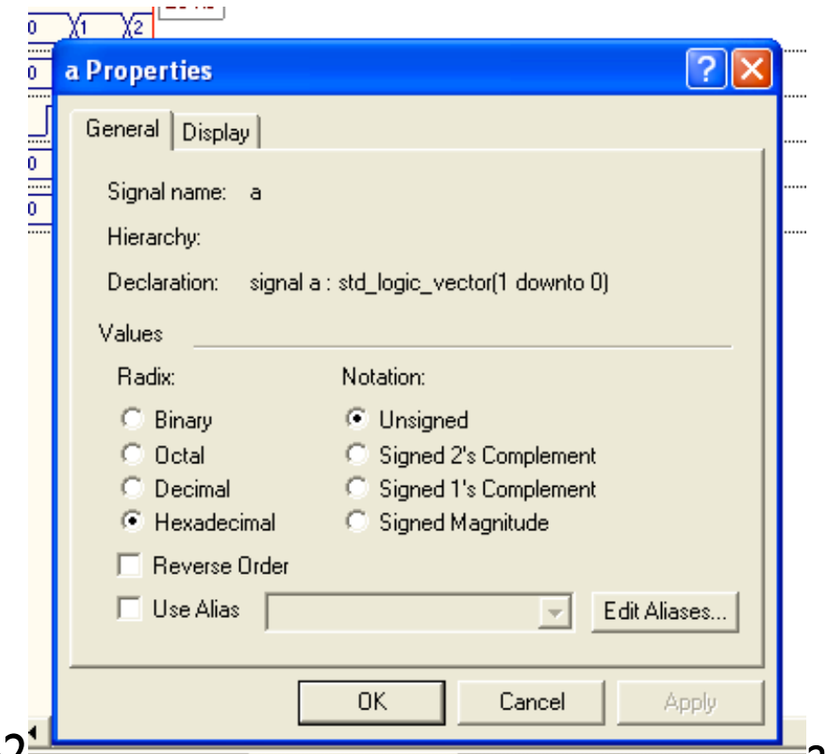
Neat Active HDL feature

Nuno Alves (nalves@wne.edu), College of Engineering

Change signal display types







Signed / unsigned data types

What for?

- This type is used for arithmetic functions!
- An UNSIGNED value is a number never lower than zero
  - "0101" represents the decimal 5
  - "1101" signifies 13
- A SIGNED value can be positive or negative (in 2's complement)
  - "0101" represents the decimal 5
  - "1101" signifies -3

2's Complement

- Suppose we're working with 8 bit quantities
- We want to find -28 in two's complement.
  1. First we write out 28 in binary form : 00011100
  2. Then we invert each digits : 11100011
  3. Then we add 1 : 11100100
- That is how one would write -28 in 8 bit binary

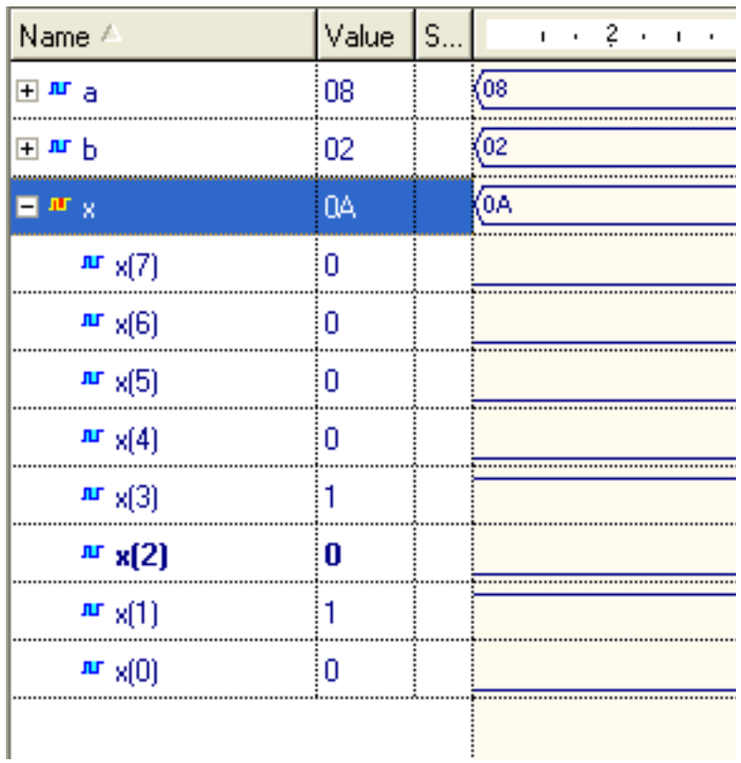
How to use?

- You need to add the std_logic_arith library to your program
- Syntax is as follows:
  - SIGNAL x: SIGNED (7 DOWNTO 0);
  - SIGNAL y: UNSIGNED (0 TO 3);

Arithmetic operators!

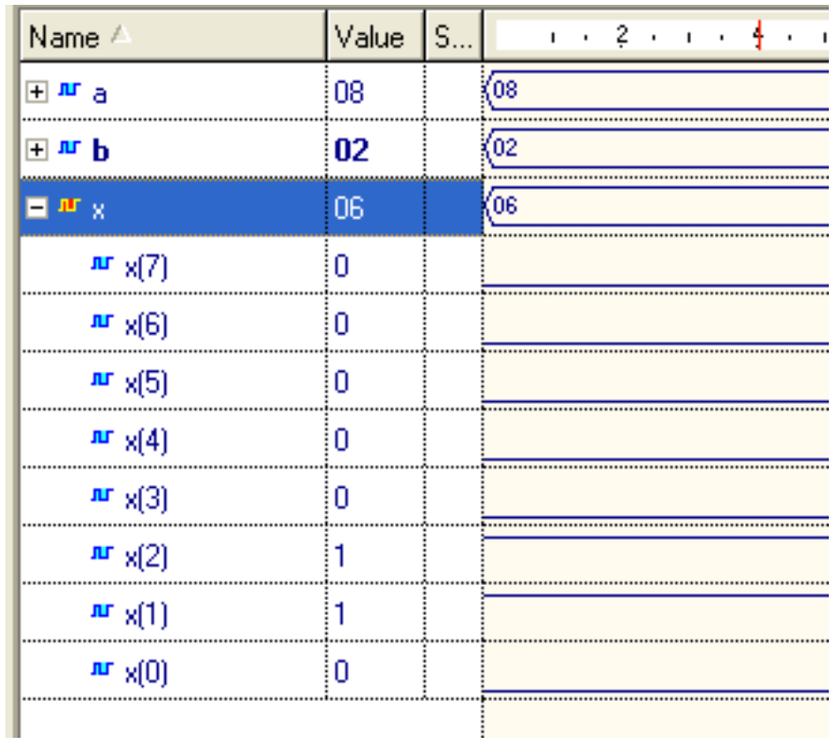
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation
MOD	Modulus
REM	Remainder
ABS	Absolute value

Addition example



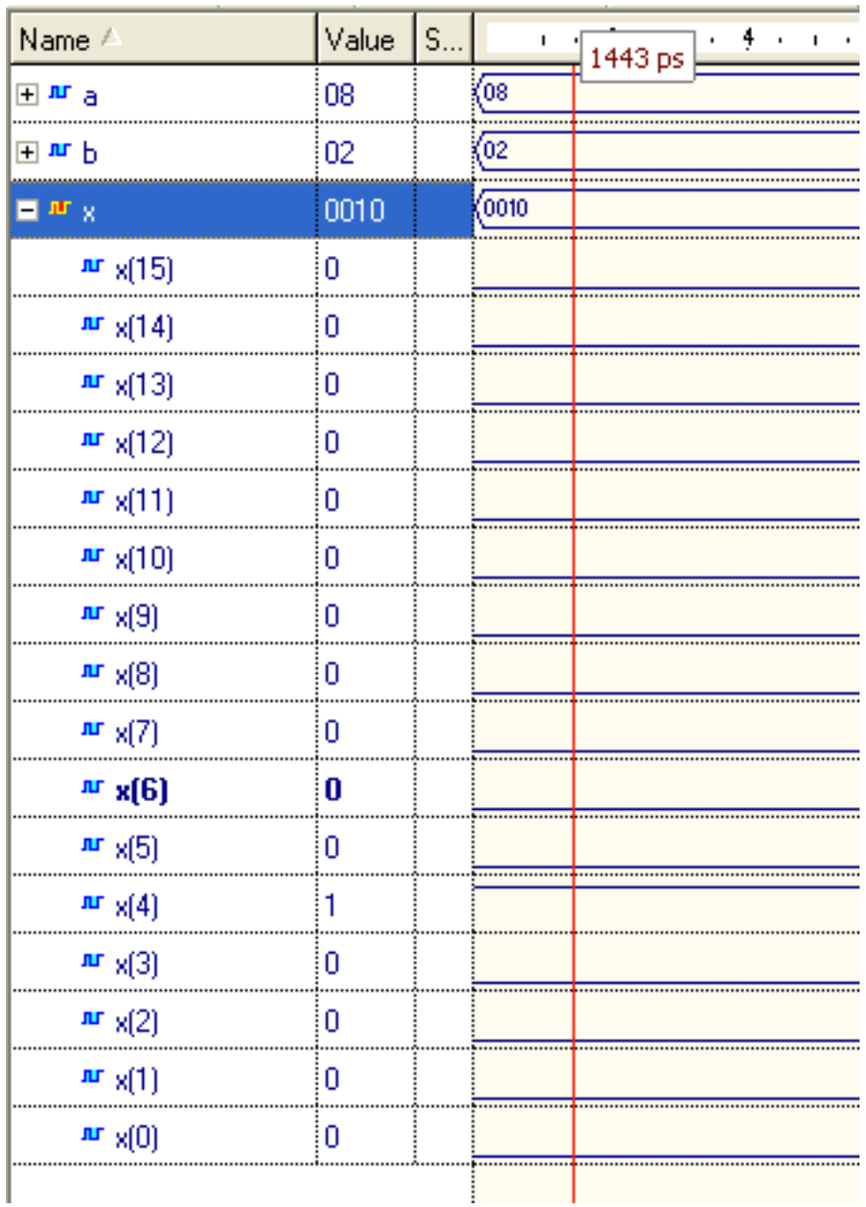

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal b: signed (7 downto 0);
signal x: signed (7 downto 0);

begin
     a <= x"08"; --hexadecimal
     --a <= "000001000"; --same as above
     b <= x"02";
     x <= a + b;
end architecture;
```

Subtraction example




```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal b: signed (7 downto 0);
signal x: signed (7 downto 0);

begin
    a <= x"08"; --hexadecimal
    --a <= "000001000"; --same as above
    b <= x"02";
    x <= a - b;
end architecture;
```

Multiplication example




```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal b: signed (7 downto 0);
signal x: signed (15 downto 0);

begin
    a <= x"08"; --hexadecimal
    --a <= "000001000"; --same as above
    b <= x"02";
    x <= a * b;
end architecture;
```

This is WRONG!

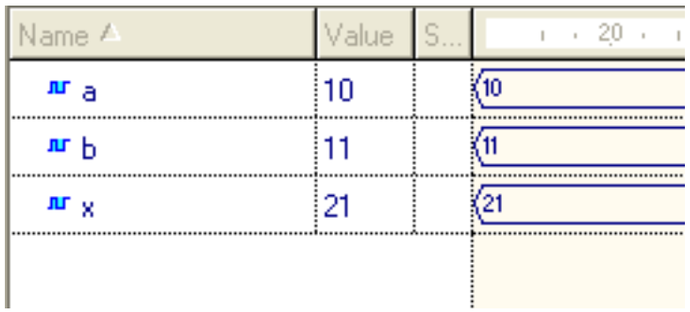
- With signed data-types I can ONLY perform arithmetic operations
- This means, no logic operations
- I need to do something else to make this work.

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal b: signed (7 downto 0);
signal x: signed (7 downto 0);

begin
    a <= x"08"; --hexadecimal
    --a <= "000001000"; --same as above
    b <= x"02";
    x <= a AND b;
end architecture;
```

More arithmetic

Nuno Alves (nalves@wne.edu), College of Engineering

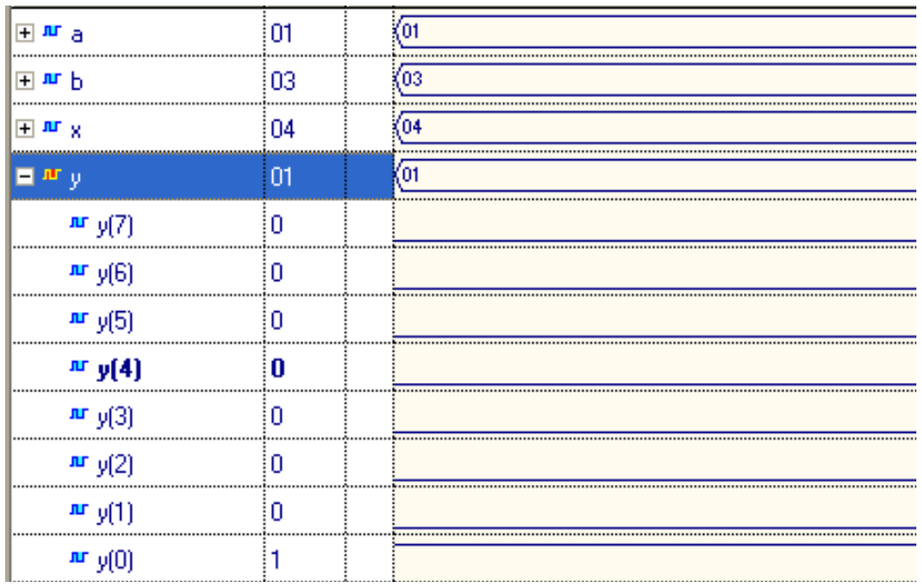
Arithmetic with integers



- You can do arithmetic with integers
- You **CAN'T** do logic operations with integers

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a : integer := 10;
signal b : integer := 11;
signal x : integer;

begin

    x <= a + b;

end architecture;
```

Arithmetic with STD_LOGIC_VECTOR



- You can still do logic!
- You can do arithmetic with STD_LOGIC_VECTOR but you must add and extra library:
  - `USE ieee.std_logic_unsigned.all;`

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

entity test is
end entity;

architecture myarch of test is
SIGNAL a: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
begin

     a <= "00000001";
     b <= "00000011";
     x <= a + b;
     y <= a AND b;


end architecture;
```

Data conversion

Converting between data-types

- VHDL does not allow direct operations (arithmetic, logical, etc.) between data of different types
- However, it is often necessary to convert data from one type to another

- There are special functions that allow us to convert from one type to another
- I can convert from certain data-types into
  - integers
  - signed / unsigned
  - std_logic_vector

conv_integer(p)

conv_integer(p) : Converts a parameter p of type UNSIGNED, SIGNED, or STD_ULOGIC to an INTEGER value

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a: signed (7 downto 0);
signal x: integer;

begin

    a <= "00000011";
    x <= conv_integer (a);

end architecture;
```

conv_unsigned(p, b)

conv_unsigned(p, b) :
Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an UNSIGNED value with size b bits.

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a : integer;
signal x: unsigned (7 downto 0);

begin

    a <= 10;
    x <= conv_unsigned (a,8);

end architecture;
```

conv_signed(p, b)

conv_signed(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to a SIGNED value with size b bits

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;

architecture myarch of test is
signal a : integer;
signal x: signed (7 downto 0);

begin

    a <= 10;
    x <= conv_signed (a,8);

end architecture;
```

conv_std_logic_vector(p, b)

conv_std_logic_vector(p, b):
Converts a parameter p of type INTEGER, UN- SIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size b bits.

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
-- extra package necessary
use ieee.std_logic_arith.all;

entity test is
end entity;


architecture myarch of test is
signal a : integer;
signal x: std_logic_vector (7 downto
0);

begin

    a <= 10;
    x <= conv_std_logic_vector (a,8);

end architecture;
```

Summary

Important to remember!

- You can only perform arithmetic operations with signed/ unsigned, integers and std_logic_vector types (with extra libraries)
- You can only perform logic operations with std_logic (or closely related types)
- You can use certain functions to convert from one data-type into another

Arithmetic **and** logic operations

- You can only perform arithmetic **and** logic operations with std_logic_vector types.
- Make sure you add all the following libraries:

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;
```

What is the difference between these two codes?

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

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( "01000", "10010", "11001");
begin

end architecture;
```

Values are assigned only at the beginning (time=0)!

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

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix;
begin

    x(0) <= "11001";
    x(1) <= "10010";
    x(2) <= "01000";

end architecture;
```

Values are constantly being assigned (and in parallel)!

Why is this illegal?

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

entity test is
   port (signal x: in bit);
end entity;

architecture myarch of test is
begin

   x <= '0';

end architecture;
```

Extremely important!

I can't assign values to the circuit inputs!

Practice Exercises

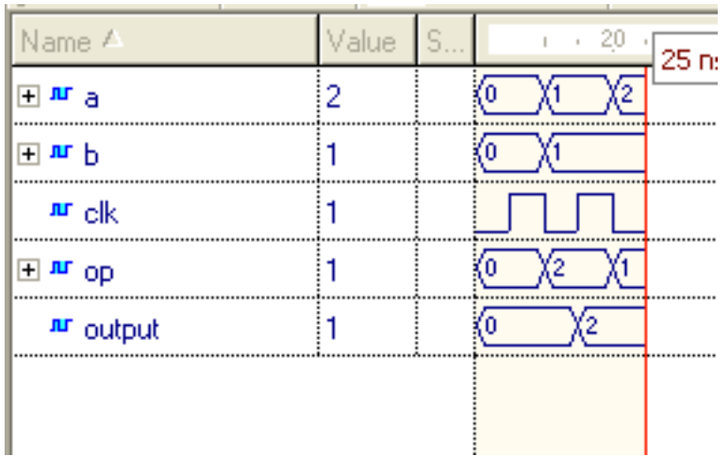
Exercise #1 - Arithmetic

- 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 its 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

```
entity exercise1 is
    port (a,b,op : in std_logic_vector(1 downto 0);
          clk : in bit;
          output : out integer
    );
```

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;
                                                    -- 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 DOWNTO 0);
                                                    -- 1Dx1D
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
                                                    -- 2D array

SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
```

Exercise #3 - Why illegal?

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

```
x <= v(1);

x <= w(2);
x <= w(2, 2 DOWNTO 0);

v(0) <= w(2, 2 DOWNTO 0);

v(0) <= w(2);
y(1) <= v(3);
w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0);
```

where,

```
TYPE row IS ARRAY (7 DOWNTO 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 DOWNTO 0);
                                                    -- 1Dx1D
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
                                                    -- 2D array

SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
```

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 test-bench on the course website and 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;
```

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