U.P.V. E.H.U.

Basic programming using VHDL

U.P.V. E.H.U.

Index

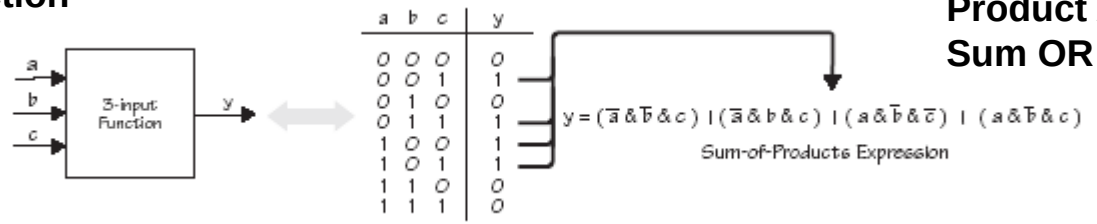
1:

Introduction

U.P.V. E.H.U.

Standard wired logic

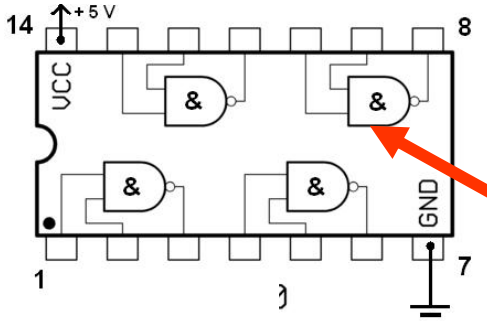
Digital logic function



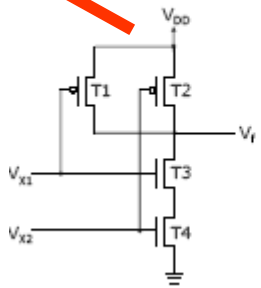
Product AND (&)
Sum OR (|)

"Black box" with I / O **True table** **Sum of products**

Simplification



CMOS NAND gate



X1	X2	T1	T2	T3	T4	f
0	0	On	On	Off	Off	1
0	1	On	Off	Off	On	1
1	0	Off	On	On	Off	1
1	1	Off	Off	On	On	0

Based on transistors

Logic chips standard

Fixed logic!

Need for user-defined layouts

U.P.V. E.H.U.

Programmable logic (I)

- Programmable logic allows the user to define the function carried out by a circuit.

- Programmable logic is based on programmable devices, where you can define their function, unlike standard wired circuits, which have a fixed function.

To define the functionality of these programmable devices, we use Hardware Description Languages, HDL

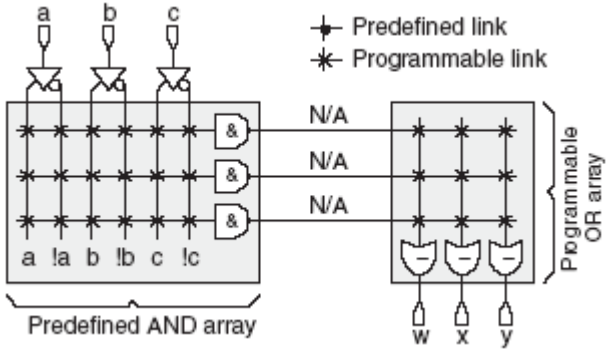
U.P.V. E.H.U.

Programmable logic (II)

- The HDL program is designed for the desired functionality.
- We simulate the program to verify proper operation (waveforms ...).
- We synthesize the program. A synthesis software translates the program into logic elements (gates ...), and then, it implements the circuit according to the programable technology under use.
- The result (file .bit ...) is uploaded in the programmable device.
- Different types of programmable devices:
    - PLD, Programmable Logic Device
    - ASIC, Application Specific Integrated Circuit (CUSTOM)
    - FPGA, Field Programmable Gate Array

Example of programmable logic device: PLD

- **Groups of ANDs and ORs linked by potential links:**
- **programmable links**
- **reconfigurable links**
- **different types of PLDs, eg: SUM OF PRODUCTS:**



Unprogrammed state

Groups of ANDs and ORs

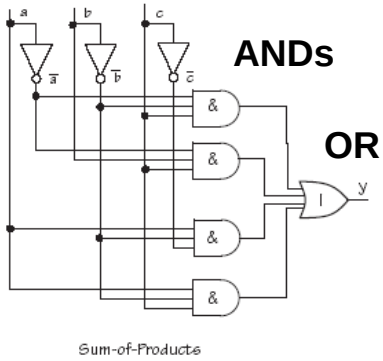
$y = (a \,\&\, b \,\&\, c)$

$x = (a \,\&\, b \,\&\, c) \mid (\bar{b} \,\&\, \bar{c})$

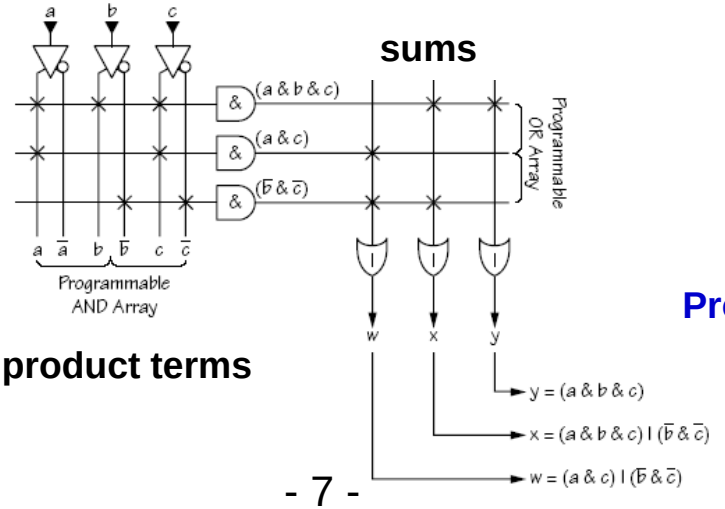
$w = (a \,\&\, c) \mid (\bar{b} \,\&\, \bar{c})$

logical function

Inputs



Sum of products



product terms

Programed PLD

U.P.V. E.H.U.

Programmable logic devices

Limited complexity: thousands of doors		**Higher complexity: millions of doors**
Cheap	**Intermediate solution**	**Very expensive for small amounts (it is necessary to manufacture a *mask* for the programming). Manufactured in a Custom way (Custom, eg. for very high speed, low consumption...)**
Easy of design		**Long & complex**
<u>RE</u>programmable		**<u>DO NOT RE</u>programmable, high risk of failures of design**

U.P.V. E.H.U.

Introduction to VHDL language

- **V**HSIC: (**V**ery High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage

- An standard industrial language of IEEE for the description of hardware.

- A high level language for the simulation, and synthesis and implementation of programmable circuits.

U.P.V. E.H.U.

History of VHDL

- 1980: The Department of Defense of US created a project to develop a new standard for the description of hardware within the VHSIC (Very High Speed Integrated Circuit) program.
- 1982: Accepted by Intermetrics, IBM and Texas.
- 1985: Version 7.2, public.
- 1987: The IEEE Institute confirms the standard as 1076 (VHDL-87).
- 1993: The VHDL language was revised and expanded as standard 1076'93 (VHDL-93)

U.P.V. E.H.U.

2:

Basic structure of digital circuit design with VHDL

U.P.V. E.H.U.

VHDL: design units

Entity

- Defines the external interface of the model

LOGICAL SYMBOL

Architecture

- Defines the internal functionality of the model

CIRCUIT LAYOUT

Configuration

- It is used to match an architecture with an entity

ENTITY <- -> Architecture

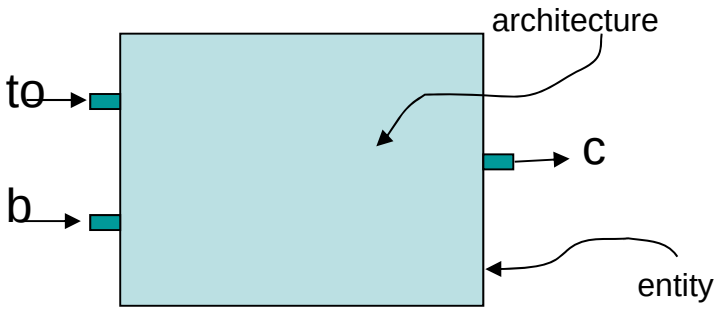
Library

- Saved set of components

PACKAGES (STATEMENT and BODY)

Entity and architecture: 1st level of abstraction

Abstraction: black box



Interface: inputs and outputs (ports)

Entity and architecture

- A hardware unit is displayed as a "black box"

  - The interface of the black box is completely defined.

  - The inside part is hidden.

- In VHDL the black box is called entity.

  - The **ENTITY describes the I / O** of the design

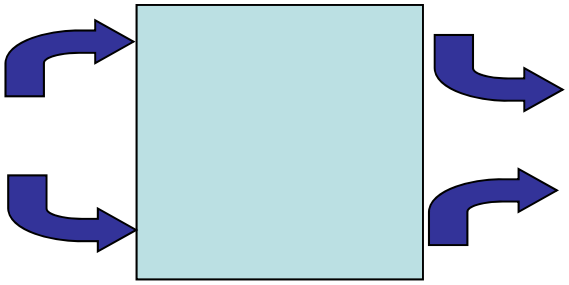
- To describe its operation an implementation called architecture is associated

  - The **ARCHITECTURE describes the content** of the design.

U.P.V. E.H.U.

PORTS of an entity

Interface of the device



Ports: inputs and outputs

Ports = communication channels

Each of the possible connections is called a PORT and comprises:

- A **name**, which must be unique within the entity.
- Some characteristics such as:

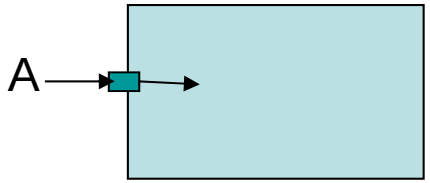
  - the direction of the data flow, input, output, which is known as **MODE** of the port.

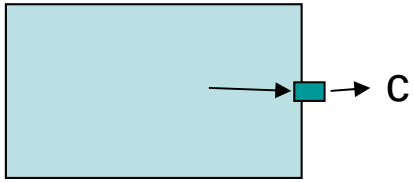
  - the values that the port can take, '0', '1', 'Z', etc, these possible values depend on what is called **TYPE** of signal.

U.P.V. E.H.U.

PORTS: Modes of a port

Mode of ports



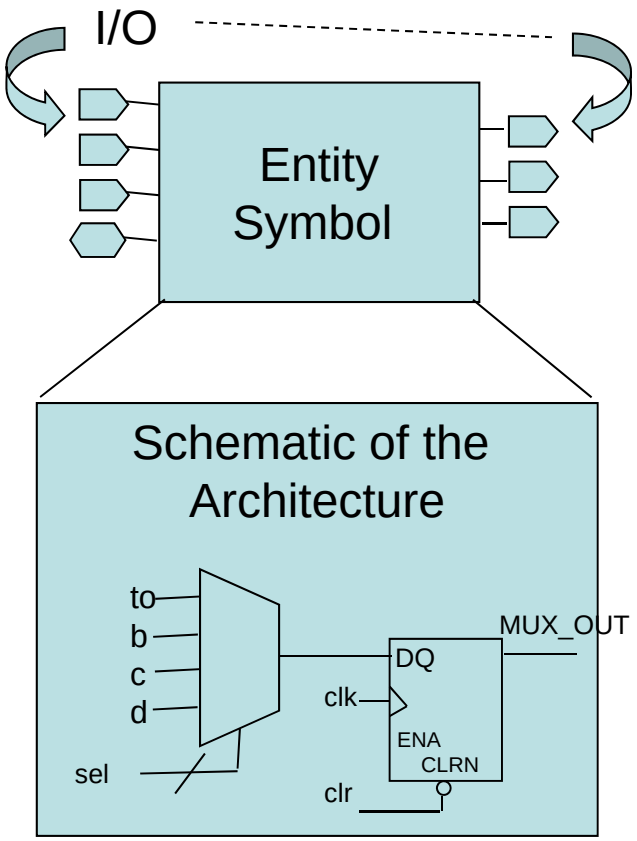


It indicates the direction and whether the port can be read or written within the entity.

- **IN** A signal entering the entity. The signal can be read but not written.
- **OUT** A signal that goes out of the entity. The signal can not be read within the entity.
- OTHERS…

U.P.V. E.H.U.

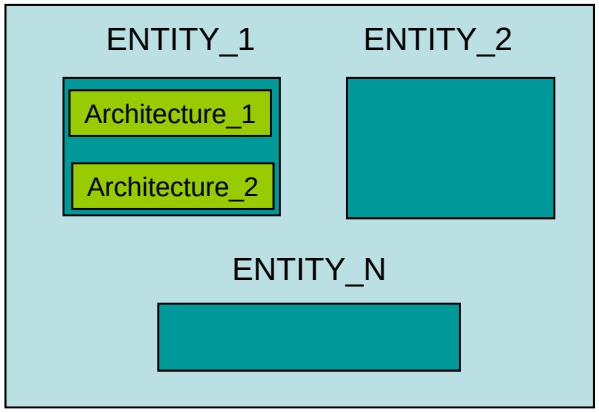
Structure of a VHDL design




```
ENTITY IS entity_name
        port declaration
END entity_name;


my_arq_name OF ARCHITECTURE IS my_entity_name
        Declaration of:
        constants
        internal signals
        components
BEGIN
        signal assignments
        processes
        component instances
END my_arq_name;
```

U.P.V. E.H.U.

Architecture



- The architecture describes what is inside the entity, ie, the contents of the black box.
- An entity can be described with different models of architecture: structural / RTL / behavioral.

```
ARCHITECTURE my_arq_1 of my_entity_name is

   declarations

begin

   architecture instructions

end my_arq_1;
```

U.P.V. E.H.U.

Libraries

`LIBRARY <name>`

name is a symbolic name defined by the compiler.

`USE lib_name.pack_name.object;`

ALL is a reserved word.

```
Library IEEE;
USE
   ieee.std_logic_1164.all;
```

A library contains a package or a collection of packages (a way of storing information).

Libraries with resources:

- Standard Package
- Packages developed by IEEE
- Packages developed by integrated circuit manufacturers
- Packages developed by the user

Working Library: WORK

Library where the current design unit is compiled.

3:

Basic features of VHDL

U.P.V. E.H.U.

Type of data

- A TYPE is the definition of the possible values that an object can take

- VHDL is a strongly typed language:
  - Objects are always assigned a type when declared
  - Assignments can only be made between objects of the same type

U.P.V. E.H.U.

Predefined basic types

Types IEEE-1076

- BIT: only the values '0' or '1‘ are allowed. For digital signals.
- BIT_VECTOR: A one-dimension array (vector) of bits. To model buses.
- INTEGER: Integer type. Used as an index value in loops, constants or generic values.
- BOOLEAN: Logical type. It can take the values 'TRUE' or 'FALSE'
- REAL: For floating point numbers.
- ENUMERATED: Enumeration. User-defined values set. For example: TYPE my_state IS (start, slow, fast).
- STD_LOGIC / STD_LOGIC_VECTOR

U.P.V. E.H.U.

Type STD_LOGIC

- The two BIT values are not enough to model all the states of a real digital signal.
- The package `IEEE.standar_logic_1164` defines the type `std_logic`, which represents all the possible states of a real signal:

0	Output of a gate with low logic level
1	Output of a gate with high logic level
L	0 weak, pull-down resistor
H	1 weak, pull-up resistor
X	Unknown strong, short-circuited output
W	Unknown weak, short-circuited output
U	Uninitialized
Z	High impedance
-	No matter, used as a wildcard for synthesis and implementation

U.P.V. E.H.U.

Type STD_LOGIC_VECTOR

- To describe buses we use `std_logic_vector`, which it is an array of `std_logic`.
- The types `std_logic` and `std_logic_vector` are industry standards.
- All the values are valid in a VHDL simulator, however only '0', '1', 'Z', 'L', ‘H' and '- 'are recognized for synthesis and implementation.
- Notation:
  - 1 bit: single quote (‘)
  - Multiple bits: double quotes (“)

U.P.V. E.H.U.

Operators defined in VHDL

- **Logic**
  and
  or, nor
  XOR, XNOR

- **Relational**
  = equal
  /= Different
  < Lower than
  <= Lower than or equal
  > Greater than
  > Greater than or equal

- **Miscellaneous**
  abs absolute value
  ** exponentiation
  not negation (unit)

- **Addition**
  + add
  - subtraction
  & Concatenation of vectors

- **Multiplicative**
  * multiplication
  / division
  rem rest
  mod modulus

- **Sign** (Unit)
  +, -

- **Shift** (Bit_vector)
  sll, srl, sla, sra, rol, ror

U.P.V. E.H.U.

More on operators

- Not all operators are defined for all data types
- In particular, for `std_logic / std_logic_vector` we must get them from the standard libraries:
  - `std_logic_signed`
  - `std_logic_unsigned`
  - `std_logic_arith`
- Example of using the concatenation operator:

```
signal a: std_logic_vector (3 downto 0);
signal b: std_logic_vector (3 downto 0);
signal c: std_logic_vector (7 downto 0);
a <= "1100";
b <= "1001";
c <= a&b; - c = “11001001"
```

U.P.V. E.H.U.

Objects available in VHDL

Constant: It is an association of a name to a fixed value.

```
constant pi: real: = 3.1416;
```

Variable: Is a temporal and local storage of data visible only within a process. Its value is immediately changed by an assignment (: =).

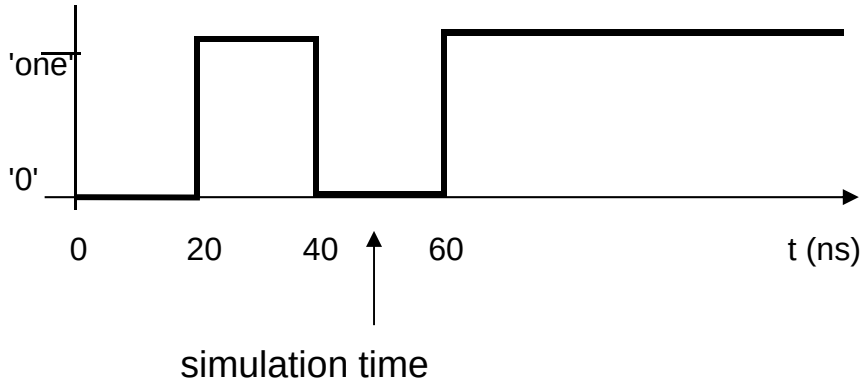
Signal: Is the modeling of a cable, which can connect ports (I / O) of entities and also connect processes within the architecture. It is a waveform that changes over time, so the assignment (<=) is not immediate, it is effective only when time goes on.

```
signal led: std_logic;

led <= '1';
```

U.P.V. E.H.U.

Concept of signal

```
signal alarm: bit: = '0'; --Initial value in testbench
          ...
alarm <= '1' after 20 ns, '0' after 40 ns,
         '1' after 60 ns;
```

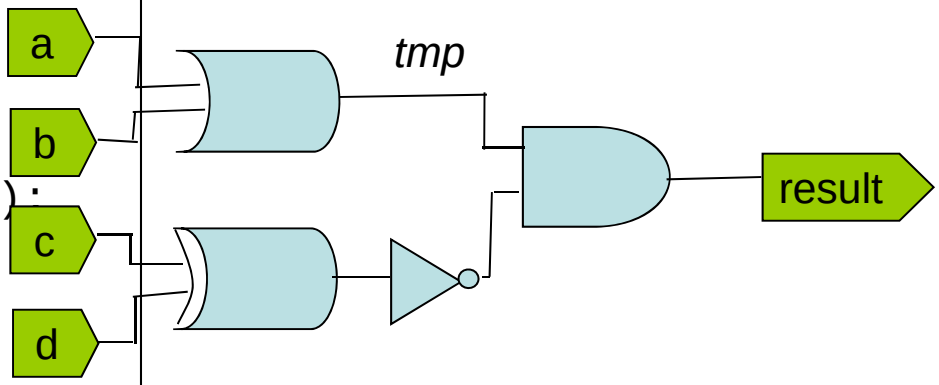


U.P.V. E.H.U.

Declaration of signals

Interconnection of components or processes

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


ENTITY my_design IS
PORT(a, b, c, d:  IN STD_LOGIC;
          result: OUT STD_LOGIC);
END my_design;


ARCHITECTURE logic OF my_design IS

signal tmp: STD_LOGIC;

BEGIN

tmp    <= a or b;
result <= (tmp and not (c xor d));

END logic;
```



`a, b, c, d` and `result` are default signals (ports), while `tmp` needs to be declared as it is an internal signal.

U.P.V. E.H.U.

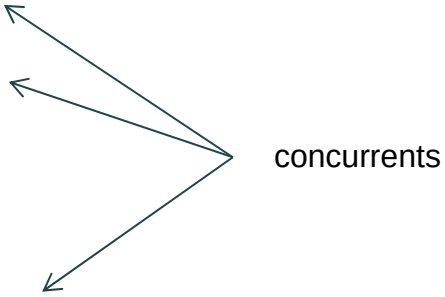
Basic elements of VHDL

Processes: Consist of instructions executed and evaluated within it sequentially. Several processes are evaluated concurrently, but once a process starts the execution of its sequential instructions time does not go on, No signals are assigned until the process ends

Concurrent instructions: the ones in the architecture outside any process, and they are evaluated concurrently.

Component Instances: Evaluated concurrently.

processes
concurrent instructions
    Boolean equations
    conditional assignments
Component Instances



U.P.V. E.H.U.

Concurrence in VHDL

```
architecture my_arch of my_ent is
begin
    c <= a and b;
    d <= c when led = '1' else 'Z';

    seq: process (clk, reset)
    begin
         .
         .
         .
    end process;

    Proc_A: process (sig_1, sig_2)

    begin

    end process;

end my_arch;
```

concurrent assignments

sequential instructions

optional label

List of sensitivity

declarative region of process

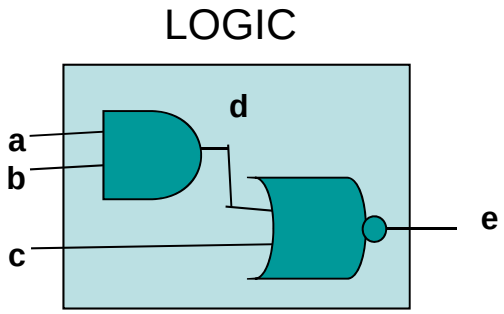
process body: sequential instructions

- VHDL supports the notion of "concurrent execution".
- The elements used to describe concurrency within an architecture are:
  - instances
  - concurrent instructions
  - processes
- The relative order of processes and concurrent assignments within an architecture is not relevant.
- Signals are used to control the activation of processes through the sensitivity list.

U.P.V. E.H.U.

Concurrent instructions

```
d <= a AND b;
e <= d NOR c;
```



concurrent assignments

- An instruction is executed only when any of its inputs changes.
- The assignment of the new values is performed once all the instructions whose inputs have changed are executed.
- If as a result of the new value, any of the inputs of another instruction changes, the instruction is executed.

U.P.V. E.H.U.

Processes

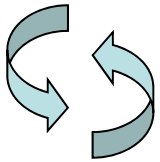
Processes

- A VHDL model can be considered as a set of processes executed in parallel.

- Every process must have a sensitivity list, which is a set of signals that activate the process when a change occurs in any of them.

```
my_process: process(A, B)
            instructions

begin
      instructions
      wait for ...

end process my_process;
```

(A, B), sensitivity list



U.P.V. E.H.U.

Sequential instructions: Processes

- Instructions within a process are evaluated sequentially.
- A process can be either *active* (AWAKE) or *inactive* (Asleep)
  - When a signal in the sensitivity list changes its value, the process proceeds to active
  - When all the instructions have been evaluated, the process becomes inactive
- Time goes on only when the process becomes inactive.

```
- Process
PROCESS (Sensitivity list)
         Constants declarations
         Variables declarations
         BEGIN
                  Sequential instruction 1
                  sequential instruction 2
                  ...
                  sequential instruction n
END PROCESS;
```

U.P.V. E.H.U.

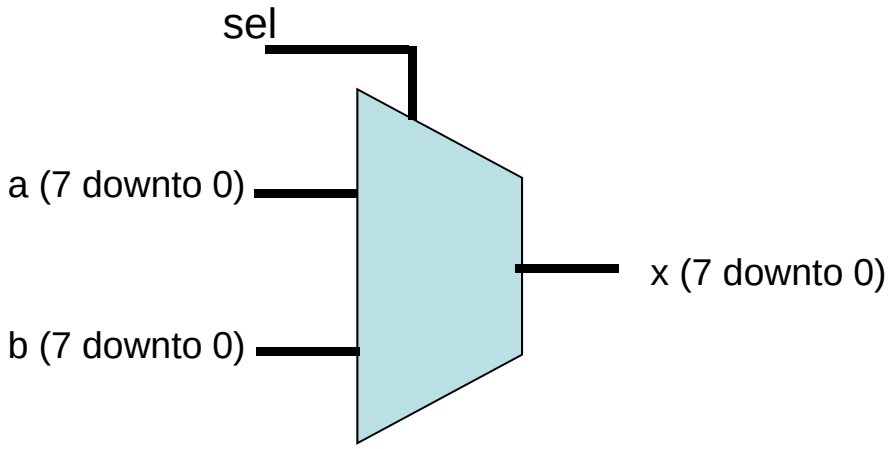
Signal assignment in processes

- Important!

  Assignments to signals within processes are nly executed when the process finishes.

- Why?:
  - Signals model physical connections, and therefore they have to take into account the value, but also the time
  - Time has to go on for a cable to change its value
  - Likewise, for a signal to change its value, time has to go on
  - Time only goes on when the process finishes

Sequential instructions: eg. MUX process



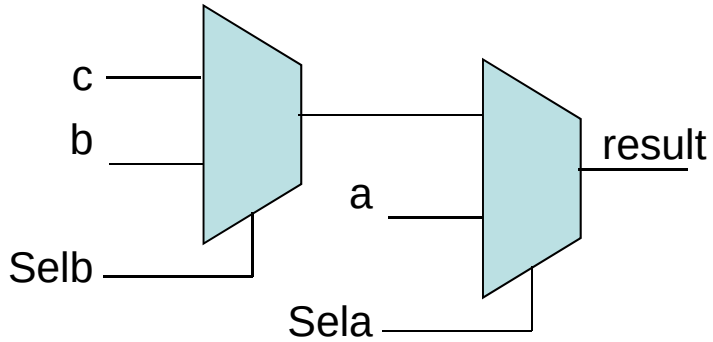

```
mux: PROCESS (a, b, sel)
BEGIN
      IF sel = '0' THEN
         x <= a;
      ELSE
         x <= b;
      END IF;
END PROCESS mux;
```

Sequential instructions: IF-THEN-ELSE

```
PROCESS(Sela, Selb, a, b, c)
BEGIN
     IF Sela = '1' THEN
        result <= a;
     ELSIF Selb = '1' THEN
        result <= b;
     ELSE
        result <= c;
     END IF;
END PROCESS;
```

IF-THEN-ELSE:

```
IF <Condition1> THEN
            {instructions}
ELSIF <Condition2> THEN
            {instructions}
              .
              .
              .
ELSE
            {instructions}
END IF;
```

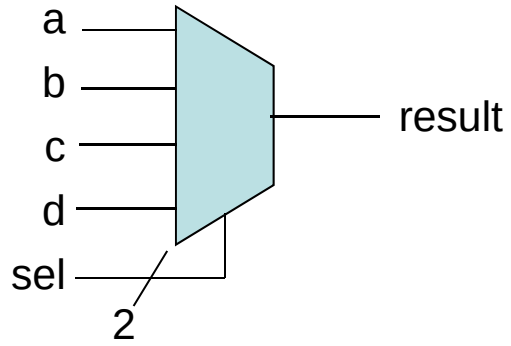


Priority: Conditions are evaluated in order from top to bottom.

The first condition TRUE causes the associated instructions to be evaluated. If all conditions are false associated instructions to ELSE are evaluated

Sequential instructions: CASE

```
PROCESS(sel, a, b, c, d)
BEGIN
        CASE sel IS
                WHEN  "00" =>
                   result <= a;
                WHEN  "01" =>
                   result <= b;
                WHEN  "10" =>
                   result <= c;
                WHEN OTHERS =>
                   result <= d;
        END CASE;
END PROCESS;
```



CASE:

```
CASE <Expression> IS
        WHEN <Condition1> =>
                {instructions}
        WHEN <Condition2> =>
                {instructions}
                .
                .
                .
        WHEN OTHERS =>
                {instructions}
END CASE;
```

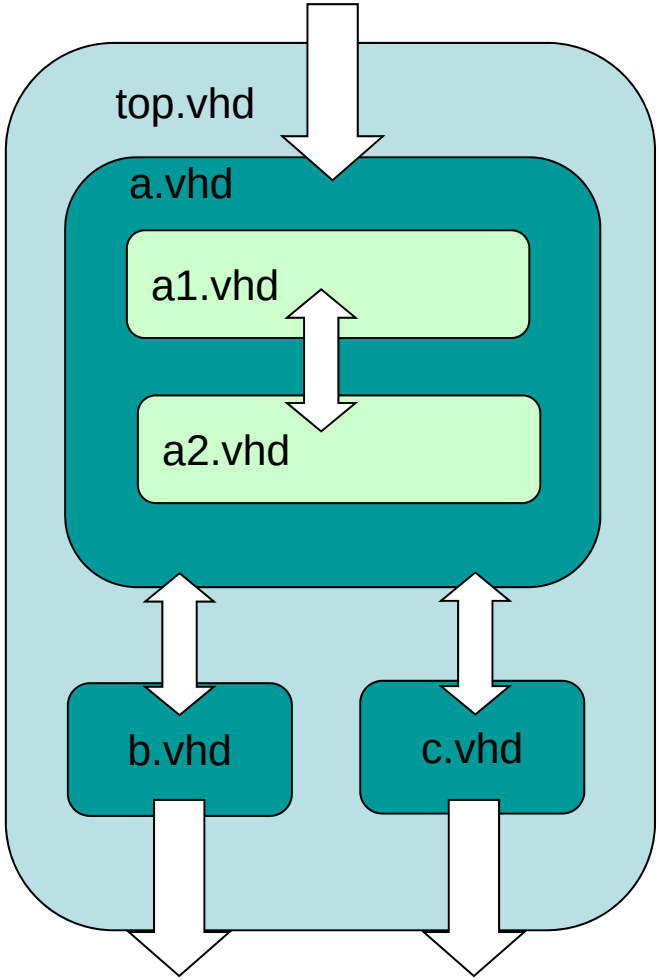
Conditions are evaluated once, no priority.

The OTHERS clause is used to assess unspecified cases.

U.P.V. E.H.U.

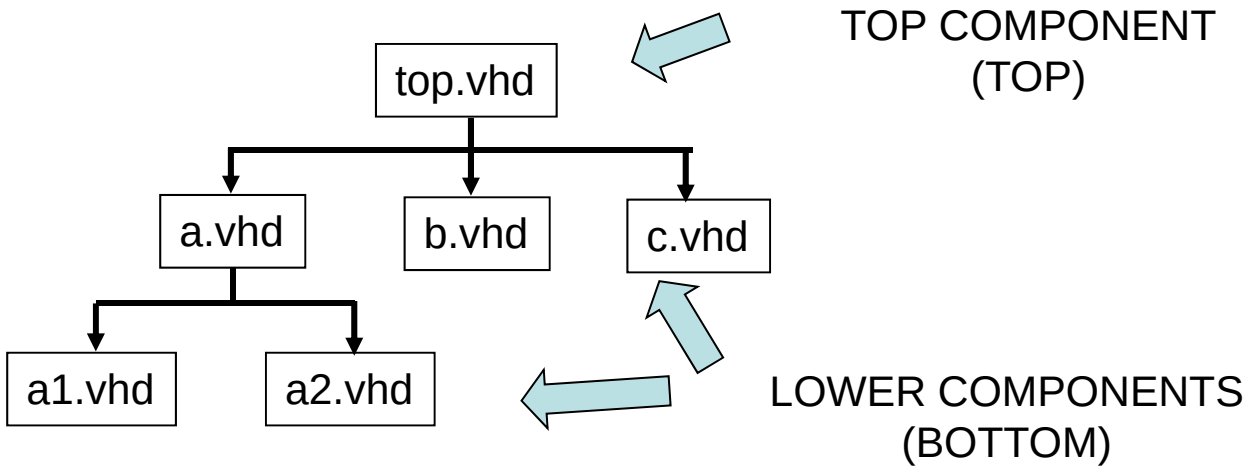
Hierarchical design

- Small components are used as elements of larger components,
- It is essential for code reusability
- It generates more readable and more portable designs



U.P.V. E.H.U.

TREE of the hierarchy



Each component of the hierarchy is a VHDL file, and has its own:

- Entity (logical symbol)
- Architecture (design of the circuit)

U.P.V. E.H.U.

Declaration of components

- Before you can use a component, you must declare it
  - Specify ports (PORT)

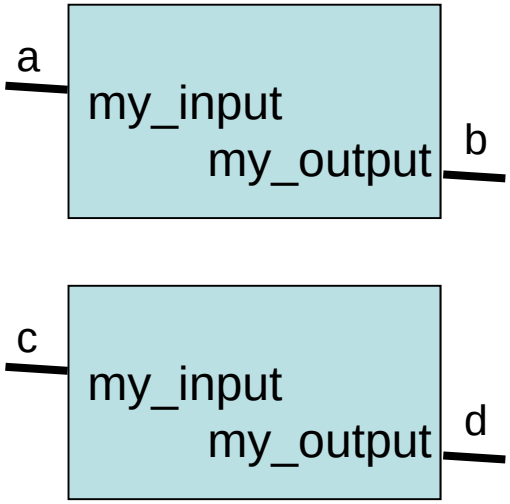
- Once the component is instantiated, the ports of the instance are connected to the circuit signals using PORT MAP

U.P.V. E.H.U.

How to instantiate a component

```
ENTITY top_ent IS

   PORT ();

END top_ent;
```

```
ARCHITECTURE my_hier OF top_ent IS
signal a, b, c, d: std_logic;
    COMPONENT my_comp PORT
      (my_input: IN std_logic;
       my_output: OUT std_logic);
    END COMPONENT;
begin
    U1: my_comp PORT MAP
        (my_input => a,
        my_output => b);

    U2: my_comp PORT MAP
        (my_input => c,
        my_output => d);

end my_hier;
```



U.P.V. E.H.U.

Configuration

A *configuration* is **a unit of design**

They are used to:

- Associate an entity with an architecture

Extensively used in simulation environments

- It provides a quick and flexible way to **try out different alternatives for the design (different architectures)**

Limited or not supported in synthesis environments and implementation

```
CONFIGURATION my_config OF my_ent IS
   FOR my_arqui

   END FOR;
END my_conf;
```