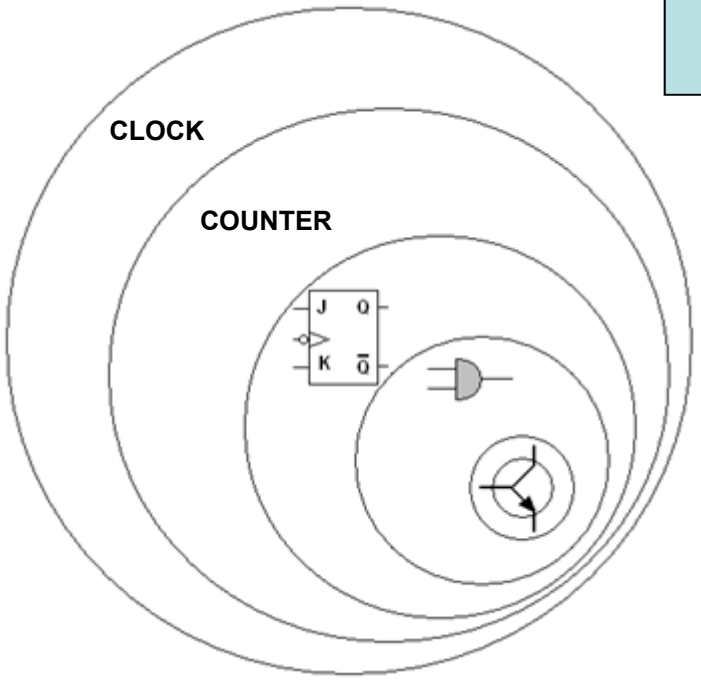
WHAT IS A SYSTEM?

Performs a function based on internal and / or external stimuli

Interconnection "modules" that can be elementary or not.

Described as **Hierarchical** form in layers, from the most elementary functions



Several complexity levels

Modelling with diferente abstraction levels

Differential Equations

GENERIC SYSTEM...

INPUTS
OUTPUTS
TRANSFORMATION

The Transformation defines the behaviour...

A system can be decomposed in sub-systems or MODULES

Digital System as a particular case of a system.

Is in this context that Hardware description languages appear

VHDL – VHSIC @HDL

At present the software tools are an essential component in the design of **digital systems.**

- Improve productivity
- Ability to predict the final behavior

VHDL – VHSIC @HDL

VHSIC-HDL
Very High Speed Integrated Circuit Hardware Description Language

Developed in the mid 80s by **DoD** and **IEEE**

VHDL

It began as a language for modeling and documentation.
At present there are synthesis tools based on VHDL.

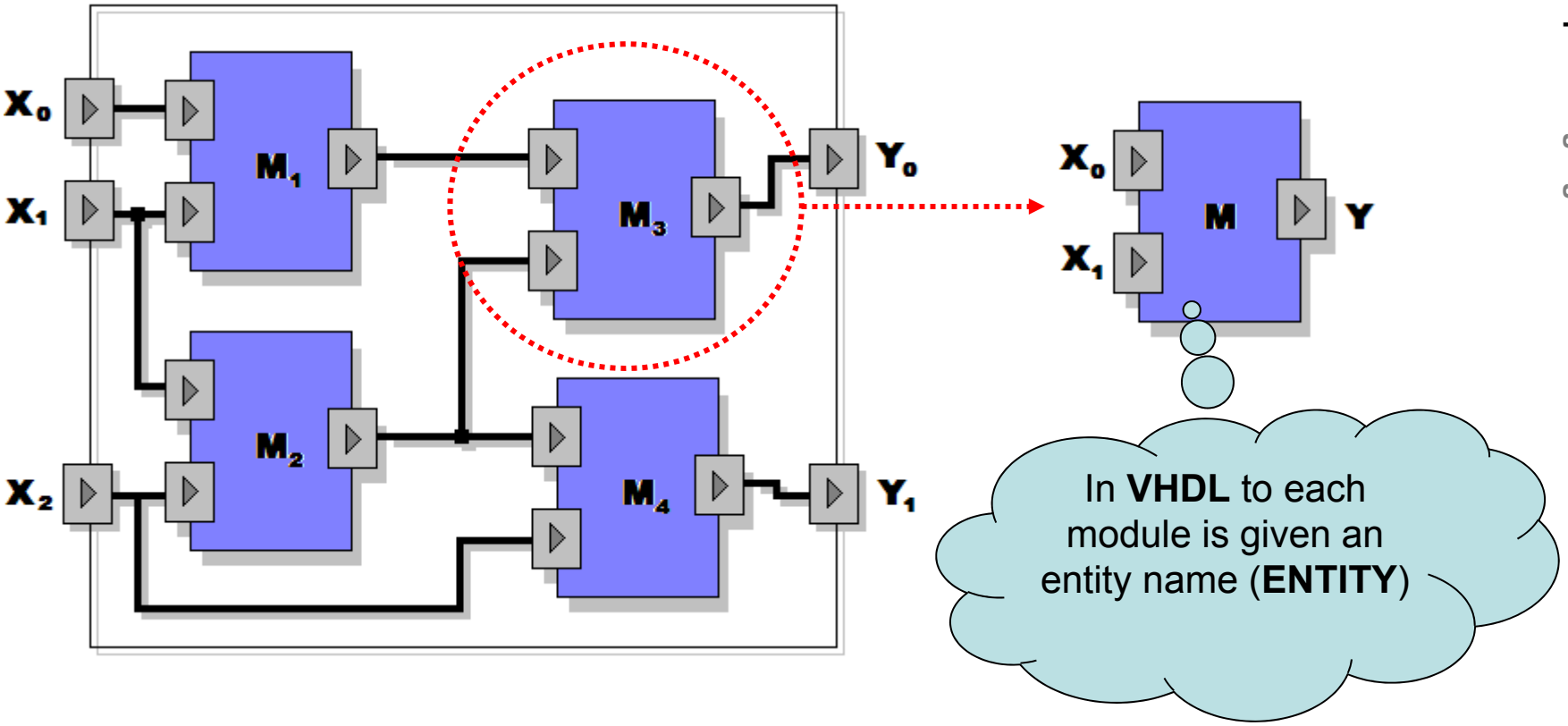
- Very Descriptive language
- Reminiscent of Pascal and Ada
- CASE INSENSITIVE

- Hierarchical decomposition of the project
- Each element has an **interface** and **behaviour**
- The behaviour can be specified by an algorithm or structure
- Timing, competition and sequencing can be modeled
- The logical operations can be simulated

- At present only set to digital systems
- Some efforts to extend for analog and hybrid systems

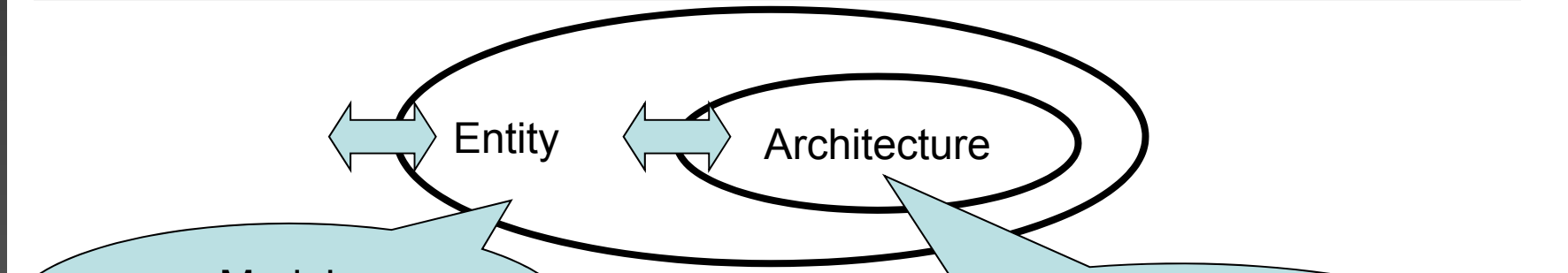
ENTITY vs. ARCHITECTURE

System Decomposed in MODULES



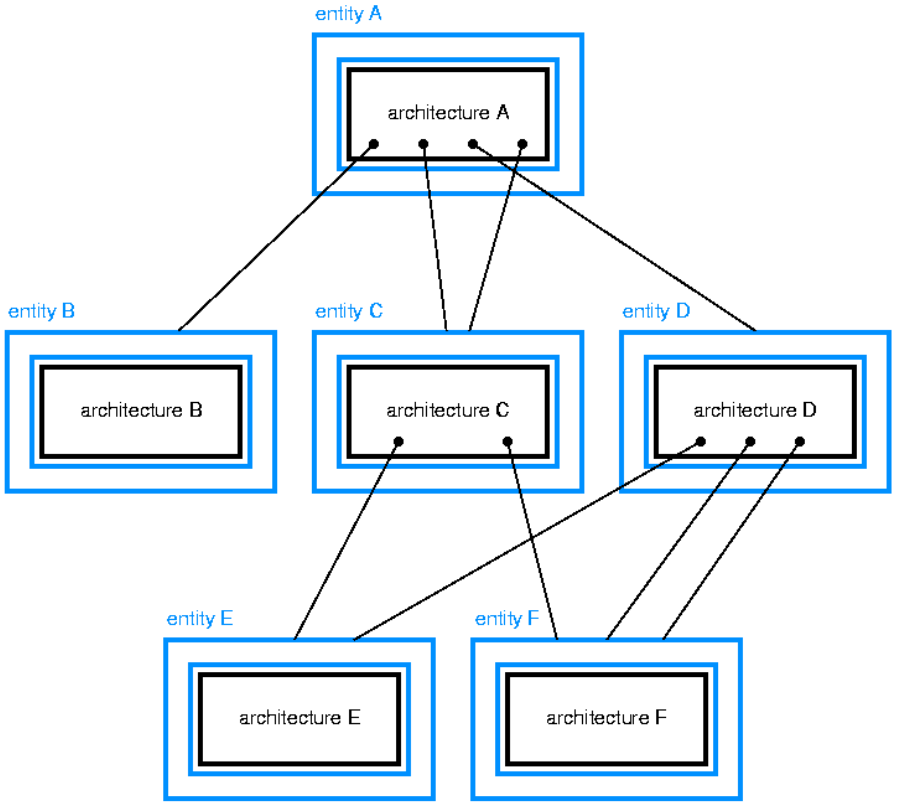
- The entities inputs and outputs are designated as ports (**PORT**).
- Each sub-module is an entity instance.
- Each instance is an entity.
- Entities are linked by signals (**SIGNAL**).

ENTITY vs. ARCHITECTURE



Module inputs/outputs declaration

Detailed description of the module internal structure and its behaviour



- Na architecture can use several entities
- Several architectures can use the same entity

Draw Hierarchical Structure

VHDL project components

PACKAGE
ENTITY
ARCHITECTURE
CONFIGURATION

Package: constants, libraries;

Entity: pinos de entrada e saída;

Architecture: project implementation;

Configuration: defines the architectures that will be used.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
```

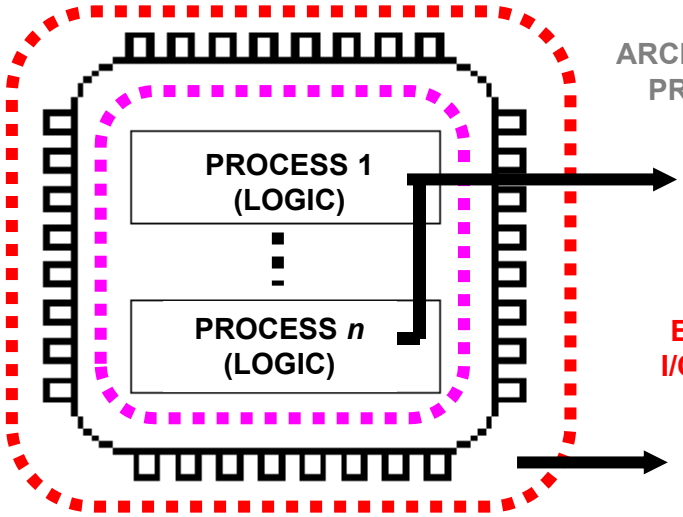
PACKAGE (BIBLIOTECAS)

```
ENTITY exemplo IS
PORT (

        < descrição dos pinos de entrada e saída >

    );
END exemplo;
```

ENTITY (PINOS DE I/O)

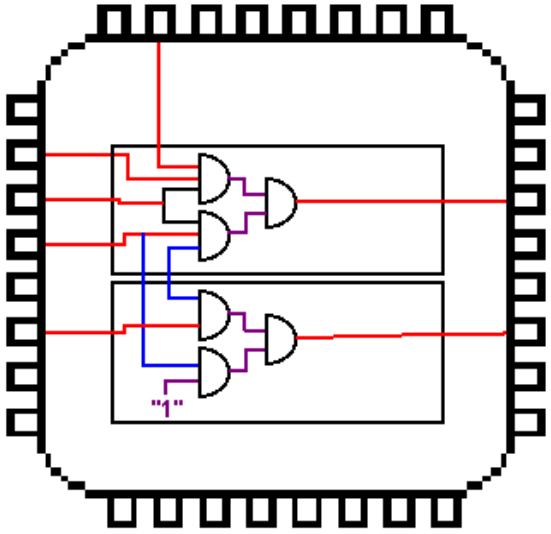
```
ARCHITECTURE teste OF exemplo IS
BEGIN
      PROCESS( <pinos de entrada e signal> )

           BEGIN

               < descrição do circuito integrado >

           END PROCESS;
END teste;
```

ARCHITECTURE (ARQUITETURA)



ARCHITECTURE
PROCESSES

ENTITY
I/O PINS



INPUT AND OUTPUT PINS
(I/O – INPUT/OUTPUT

SIGNAL

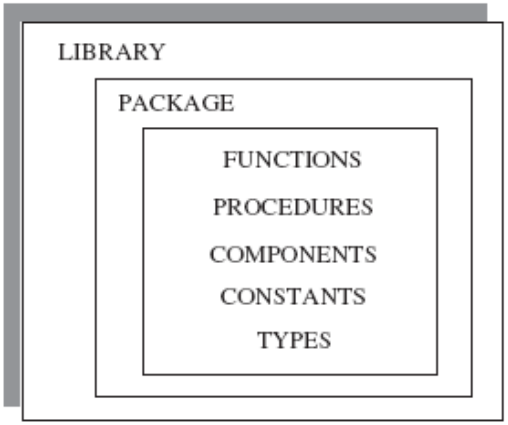
VARIABLE

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

entity placa is
port  (
        a : in bit_vector( 6 downto 0);
        b : out bit_vector( 7 downto 0)
      );
end placa;

architecture TTL of placa is
signal pino_1 bit;
signal pino_2 bit;
signal pino_3 bit;
signal pino_4 bit;
signal pino_5 bit;

begin
        CI_X : process( a )
        begin
                <descrição do processo>
        end process CI_Y;

        CI_Y : process( a )
        begin
                <descrição do processo>
        end process CI_Z;

end TTL;
```

Package

The package (library) has a collection of elements including data types description.

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```



IEEE.std_logic_arith – Arithmetic functions

IEEE.std_logic_signed – Arithmetic functions with signal

IEEE.std_logic_unsigned – Arithmetic functions without signal

IEEE.std_logic_1164 – Logic operations

Entity

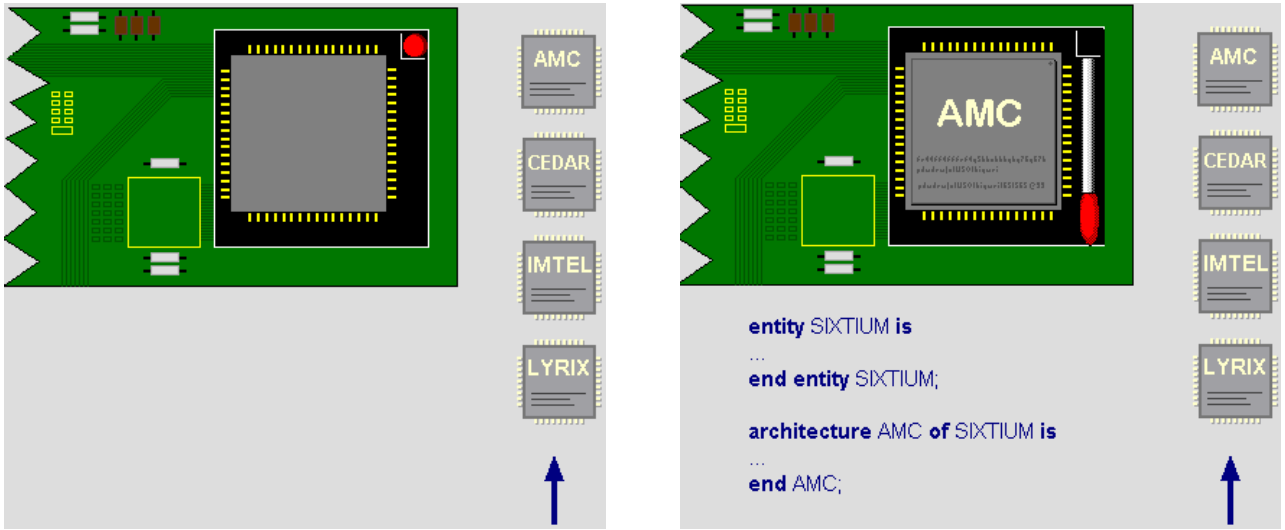
Description of system I/O interface with the plaque.

```
ENTITY <name> IS
PORT(
        control_signal: IN <type>;
        input:          IN <type>;
        output:         OUT <type>;
    );
END <name>;
```

IN, OUT, INOUT, BUFFER

<type> : bit,
bit_vector,
std_logic_vector,
real,
integer, etc.

Architecture




```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

An ARCHITECTURE has:

- DECLARATIVE part (optional) – where, for example, SIGNALS, CONSTANTS and VARIABLES, are declares

-Code – from BEGIN

Concurrent Architecture

The concurrent architecture is a more complex way to describe a system, generally has multiple processes within an architecture.

```
architecture SomeArch of SomeEnt is
begin
P1: process (A,B)
begin
  somestatement;
  somestatement;
  somestatement;
  somestatement;
end process P1;
P2: process (A,C)
begin
  somestatement;
  somestatement;
  somestatement;
end process P2;
P3: process (B)
begin
  somestatement;
  somestatement;
end process P3;
end architecture SomeArch;
```

```
architecture SomeArch of SomeEnt is
begin
P1: process (A,B)
begin
  somestatement;
  somestatement;
  somestatement;
  somestatement;
end process P1;
P2: process (A,C)
begin
  somestatement;
  somestatement;
  somestatement;
end process P2;
P3: process (B)
begin
  somestatement;
  somestatement;
end process P3;
end architecture SomeArch;
```

Data Types

BIT: assume values ‘0’ and ‘1’

LIBRARY **std**
PACKAGE **standard**

BIT_VECTOR: it’s a set of bits. Ex.: “010001”

BOOLEAN: assume values *TRUE* or *FALSE*

REAL: always with decimal point. Ex.: -3.2, 4.56, 6.0, -2.3E+2

INTEGER: Integer of 32 bit (de -2,147,483,647 a +2,147,483,647)

```
SIGNAL x: BIT; -- x is declared as an one bit signal.

SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
                -- y is a vector of 4 bit (the leftmost bit
          is the MSB)

SIGNAL w: BIT_VECTOR (0 TO 7);
                -- y is a vector of 8 bit (the rightmost bit
                   is the MSB).
```

Allocation of Values to Signs …

```
x <= '1'; --x is a 1 bit signal whose value is '1'

y <= "0111"; --y is a 4 bit signal whose value is "0111" (MSB='0')

w <= "01110001"; --w is a 8 bit signal whose value is "01110001"
                 (MSB='1').
```

Pelica to bit and quotation marks to vector!!!

LIBRARY **ieee**
PACKAGE **ieee.std_logic_1164**

STD_LOGIC and STD_LOGIC_VECTOR
('X', '0', '1', 'Z', 'W', 'L', 'H', '–')
STD_ULOGIC and STD_ULOGIC_VECTOR
('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '–')

```
'X'     Forcing Unknown (synthesizable unknown)
'0'     Forcing Low (synthesizable logic '1')
'1'     Forcing High (synthesizable logic '0')
'Z'     High impedance (synthesizable tri-state buffer)
'W'     Weak unknown
'L'     Weak low
'H'     Weak high
'–'     Don't care
'U'     Unresolved
```

```
x0 <= '0'; -- bit, std_logic, or std_ulogic with value '0'
x1 <= "00011111"; -- bit_vector, std_logic_vector,
-- std_ulogic_vector, signed, or unsigned
x2 <= "0001_1111"; -- underscore allowed to ease visualization
x3 <= "101111" -- binary representation of decimal 47
x4 <= B"101111" -- binary representation of decimal 47
x5 <= O"57" -- octal representation of decimal 47
x6 <= X"2F" -- hexadecimal representation of decimal 47
n <= 1200; -- integer
m <= 1_200; -- integer, underscore allowed
IF ready THEN... -- Boolean, executed if ready=TRUE
y <= 1.2E-5; -- real, not synthesizable
q <= d after 10 ns; -- physical, not synthesizable
```

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
a <= b(5); -- legal (same scalar type: BIT)
b(0) <= a; -- legal (same scalar type: BIT)
c <= d(5); -- legal (same scalar type: STD_LOGIC)
d(0) <= c; -- legal (same scalar type: STD_LOGIC)
a <= c; -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d; -- illegal (type mismatch: BIT_VECTOR x STD_LOGIC_VECTOR)
e <= b; -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d; -- illegal (type mismatch: INTEGER x STD_LOGIC_VECTOR)
```

Physical: represent a physical measure: voltage, capacitance, time
Pre-defined types: fs, ps, ns, um, ms, sec, min, hr.

Intervals: allow to determine an utilization interval withon a particular type.

range <lower_value> **to** <higher_value>
range <higher_value> **downto** <lower_value>

Array: in VHDL an array is defined as a collection of elements of the same type.

Operatos

- **Assignment operators :**

<= Used to assign a value to a SIGNAL.

:= Used to assign a value to a VARIABLE, CONSTANT, or GENERIC. Also used to establish the signals initial values

=> Used to assign values to individual elements of a vector or to OTHERS.

Operators (cont)

- **logical operators: and, or, nand, nor, xor, xnor and not**
- **numeric operators:** sum (**+**), subtraction (**-**), division (**/**), multiplication (*), module (**mod**), remaining (**rem** - ex.: 6 rem 4 = 2), exponent (**), absolut value (**abs**)
- **relational operators:** equal (**=**), different (**/=**), less than(**<**), less os equal (**<=**), bigger than (**>**), bigger or equal (**>=**)
- **displacement operators: sll** (shift left logical), **srl** (shift right logical), **sla** (shift left arithmetic), **sra** (shift right arithmetic), **rol** (rotate left logical), **ror** (rotate right logical)
- **concatenation operator:** consists in create a new vector from two existent vectors, for example:

```
data1 : bit_vector(0 to 7);                                    [01011011]
data2 : bit_vector(0 to 7);                                    [11010010]
new_data : bit_vector(0 to 7);
new_data <= (data1(0 to 1) & data2(2 to 5) & data1(6 to 7));   [01010011]
```

Sequential Commands

- **Variables assigment**

```
1  architecture topologia_arquitetura of teste is
2          signal A, B, J, G  : bit_vector(1 downto 0);
3          signal E, F: bit;
4  begin
5
6          process ( A, B, E, F, G, J)
7
8  variable C, D, H, Y : bit_vector(1 downto 0);
9  variable W          : bit_vector(3 downto 0);
10 variable Z          : bit_vector(7 to 0);
11 variable X          : bit;
12 variable DATA       : bit_vector(31 downto 0);
13
14         begin
15                 A <= "11", B <= "01", J <= "10", G <= "00'';
16                 E <= '0', F <= '1';
17                 C              := "01";
18                 X              := E nand F;
19                 Z(0 to 3)      := C & D;
20                 Z(4 to 0)      := (not A) & (A nor B);
21                 D              := ('1', '0');
22                 W              := (2 downto 1 => G, 3 => '1', others => '0');
23                 DATA           := (31 downto 28 => '1', others => '0');
24         end process;
25 end topologia_arquitetura;
```

- **Signals assignment**

```
1  architecture topologia_arquitetura of teste is
2          signal A, B, C : bit;
3          signal D: integer;
4  begin
5
6          process ( A, B, C, D)
7
8  variable L, M, N, Q bit;
9
10         begin
11                 A <= '0', '1' after 20ns, '0' after 40ns
12                 B <= not L;
13                 C <= ((L andM) xor (N nand Q));
14                 D <= 3, 5 after 20ns, 7 after 40ns, 9 after 60ns;
16
17         end process;
18 end topologia_arquitetura;
```

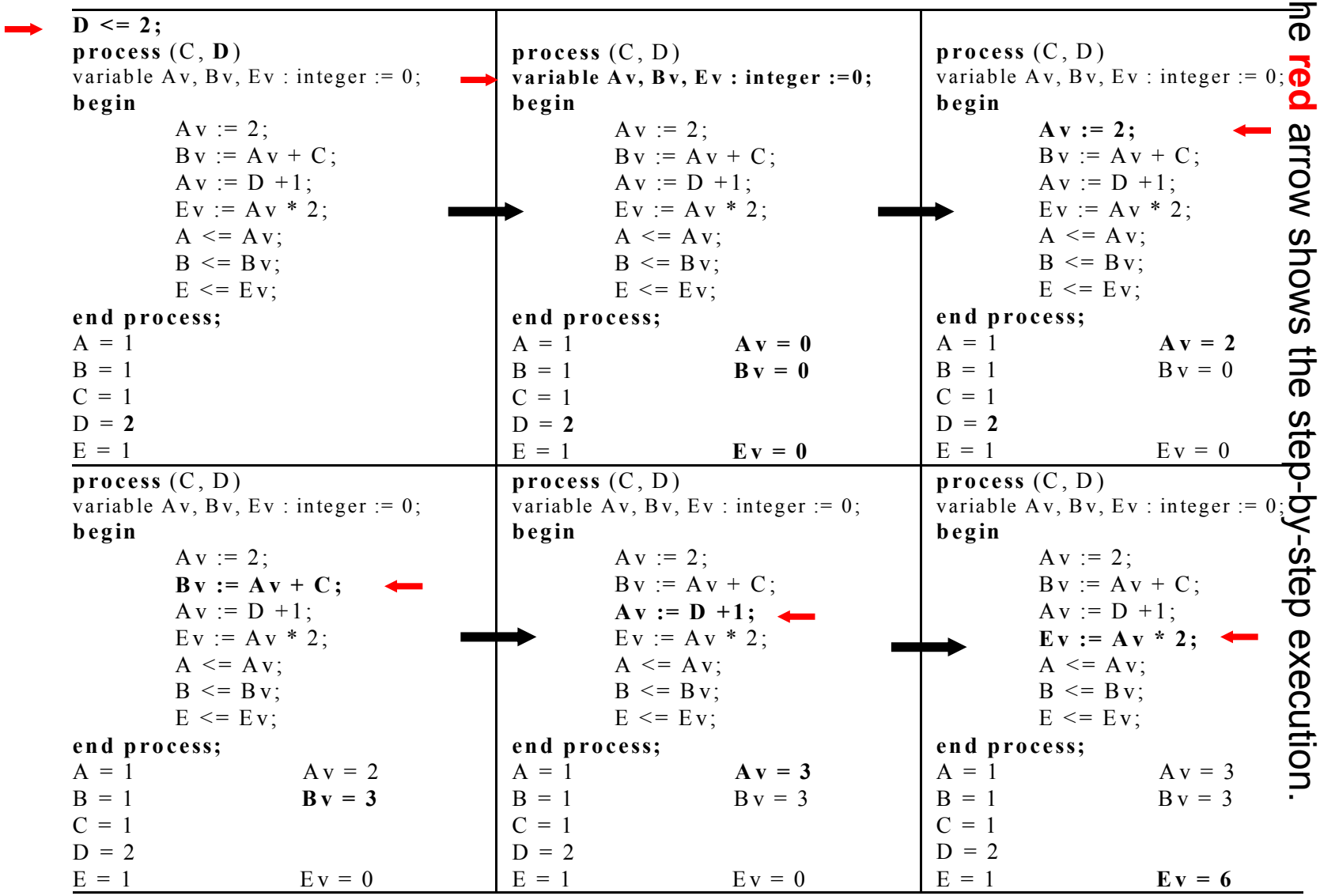
- **Difference between SIGNAL and VARIABLE**

- When using SIGNAL, allocation occurs at the end of the process, whereas the allocation VARIABLE occurs simultaneously.
- Over the next two acetates, it shows the difference between these assignments.

  <= (signal assignment)

  := (variable assigment)

The red arrow shows the step-by-step execution.



The red arrow shows the step-by-step execution.

D <= 2;			
process (C, **D**) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;** A = 1 B = 1 C = 1 D = 1 E = 1	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;** A = 1 B = 1 C = 1 **D = 2** E = 1	**process** (C, D) **begin** **A <= 2**; B <= A + C; A <= D +1; E <= A * 2; **end process;** A = 1 A <= 2 B = 1 C = 1 D = 2 E = 1	**process** (C, D) **begin** A <= 2; **B <= A + C** A <= D +1; E <= A * 2; **end process;** A = 1 A <= 2 B = 1 B <= A + (C = 1 D = 2 E = 1
process (C, D) **begin** A <= 2; B <= A + C; **A <= D +1;** E <= A * 2; **end process;** A = 1 A <= D + 1 B = 1 B <= A + C C = 1 D = 2 E = 1	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; **E <= A * 2;** **end process;** A = 1 A <= D + 1 B = 1 B <= A + C C = 1 D = 2 E = 1 E <= A * 2;	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;** A = 1 A <= 3 B = 1 B <= 2 C = 1 D = 2 E = 1 E <= 2;	**process** (C, D) **begin** A <= 2; B <= A + C A <= D +1; E <= A * 2; **end process;** A = **3** B = **2** **e não 3** C = 1 D = 2 E = **2** **e não 6**

The difference between the values is due to assignment form…

Data attributes

'LOW: Returns the lower index of the vector
'HIGH: Returns the upper index of the vector
'LEFT: Returns the index leftmost
'RIGHT: Returns the index of the rightmost
'LENGTH: Returns the size of the vector
'RANGE: Returns the range of the vector
'REVERSE_RANGE: Returns vector range in reverse order

Example:

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,
d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).

Signals attributes

'EVENT: Returns TRUE when an event occurs
'STABLE: Returns TRUE when any event occurs
'ACTIVE: Returns TRUE if the signal is '1'
'QUIET <time>: Returns TRUE if no event tooks place in time
'LAST_EVENT: Returns the time spent since the last event
'LAST_ACTIVE: Returns the time taken from the signal= '1'
'LAST_VALUE: Returns the value of the signal since the last event.

Example:

```
IF (clk'EVENT AND clk='1')...
IF (NOT clk'STABLE AND clk='1')...
WAIT UNTIL (clk'EVENT AND clk='1');
IF RISING_EDGE(clk)... - called to a function
```

Concurrent Mode vs. Sequential Mode

VHDL	CONCURRENT	WHEN
		GENERATE
		AND, NOT, etc.
		BLOCK
	SEQUENTIAL	PROCESS
		FUNCTION
		PROCEDURE

Combinatorial Logic Circuits
=> Concurrent Code

Sequential Logic Cirtcuits
=> Sequential code

CONCURRENT: WHEN

WHEN / ELSE:

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

WITH / SELECT / WHEN:

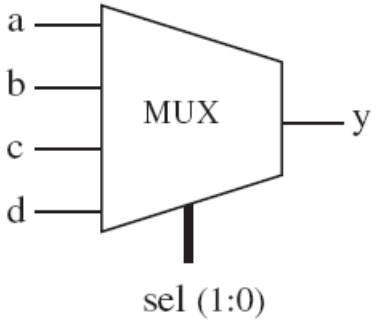
```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
   y: OUT STD_LOGIC);
END mux;
---------------------------------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
   y: OUT STD_LOGIC);
END mux;
---------------------------------------
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN "00",
            b WHEN "01",
            c WHEN "10",
            d WHEN OTHERS;
END mux2;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN INTEGER RANGE 0 TO 3;
   y: OUT STD_LOGIC);
END mux;
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN 0,
            b WHEN 1,
            c WHEN 2,
            d WHEN 3;
END mux2;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
      sel: IN INTEGER RANGE 0 TO 3;
      y: OUT STD_LOGIC);
END mux;
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel=0 ELSE
         b WHEN sel=1 ELSE
         c WHEN sel=2 ELSE
         d;
END mux1;
```



SEQUENTIAL: PROCESS

```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

> A PROCESS should be installed in the main code.

> It is executed when a signal, in its sensibilities list.

> WAIT condition is fulfilled.

The VARIABLES are optionals as well as the lables usage !!

Example: Counter Mod10

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY counter IS
   PORT (clk : IN STD_LOGIC;
         digit : OUT INTEGER RANGE 0 TO 9);
END counter;
-----------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   count: PROCESS(clk)
      VARIABLE temp : INTEGER RANGE 0 TO 10;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := temp + 1;
         IF (temp=10) THEN temp := 0;
         END IF;
      END IF;
      digit <= temp;
   END PROCESS count;
END counter;
```

Flow Changes Commands

- Command **WAIT**

- This command is intended to cause a suspension of the process declared or procedure.
- The wait command can be used in four different ways, they are:

1	**wait until** <conditional>**;**	**wait until** CLK'**event and** CLK = '1';
2	**wait on** <signal_list>**;**	**wait on** a, b;
3	**wait for** <time>**;**	**wait for** 10ns;
4	**wait;**	**wait;**

- Command **IF-THEN-ELSE**

MODELO	**exemplo 1:**	**exemplo 2:**

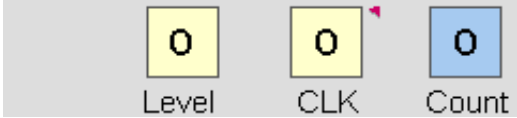
if condição_1 **then**
<comandos>
elsif condição_2 **then**
<comandos>
else
<comandos>
end if;
	if (A = '0'**) then**
	B <= "00";
	else
	B <= "11";
	end if;
		if (CLK**'event and** CLK ='1'**) then**
		FF <= '0';
		elsif (CLK**'event and** CLK ='0'**) then**
		FF <= '1';
		elsif (J = '1'**) and (**K='1'**) then**
		FM <= '1';
		end if;

- Command **LOOP FOR - WHILE**

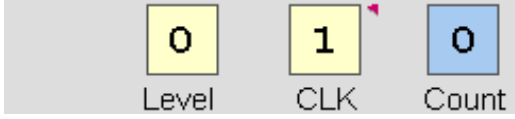
<label opcional>: **for** <parâmetros> **in** <valor_final> **loop**
<seqüência de comandos>
end loop <label opcional>**;**

1	**process (A)**
2	
3	**begin**
4	Z <= “0000”;
5	**for** i **in** 0 **to** 3 **loop**
6	**if**(A = i) **then**
7	Z(i) <= ‘1’;
8	**end if;**
9	**end loop;**
10	**end process;**

```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```



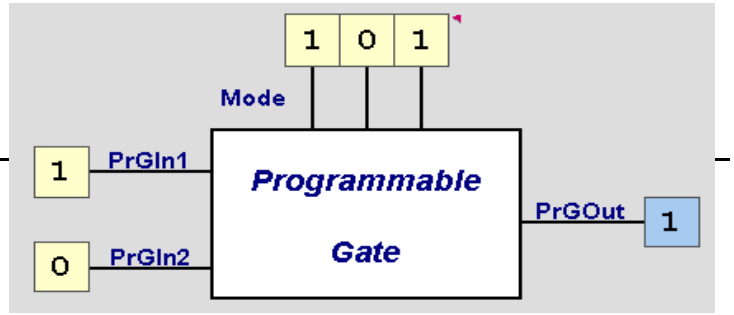

```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```



```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```


```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```

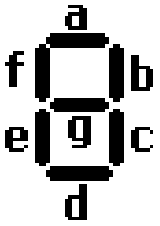
- Command **CASE**




```
porta_programável : process (Mode, PrGIn1, PrGIn2)
        begin
                case Mode is
                        when “000”      => PrGOut   <=   PrGIn1 and PrGIn2;
                        when “001”      => PrGOut   <=   PrGIn1 or PrGIn2;
                        when “010”      => PrGOut   <=   PrGIn1 nand PrGIn2;
                        when “011”      => PrGOut   <=   PrGIn1 nor PrGIn2;
                        when “100”      => PrGOut   <=   not PrGIn1;
                        when “101”      => PrGOut   <=   not PrGIn2;
                        when  others    => PrGOut   <=   ‘0’
                end case;
end process porta_programavel;
```

Coder BCD to seven-segments.

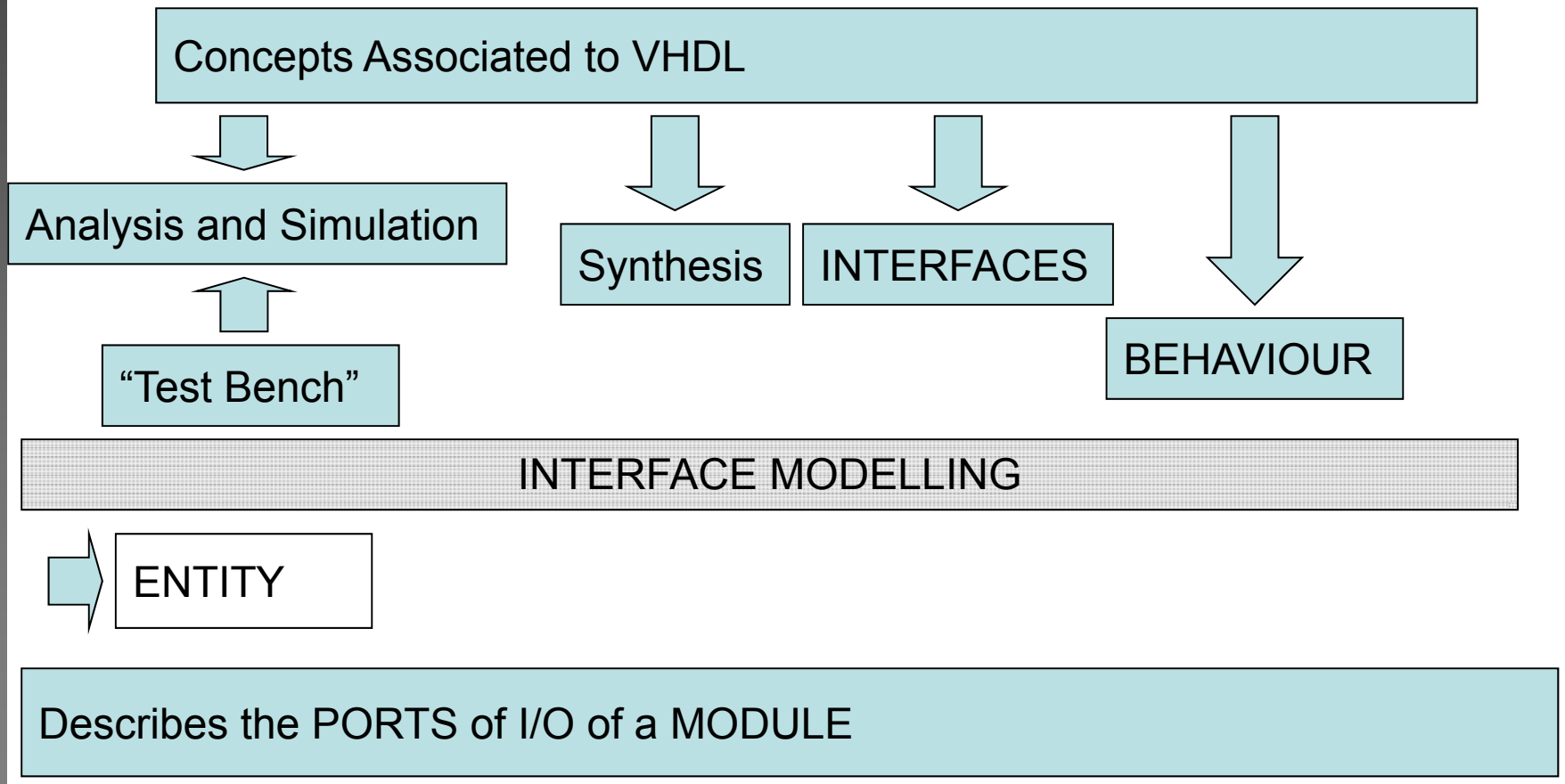
```
case code is
        when “0000” =>
                digito <=“0111111”;
        when “0001” =>
                digito <=“0000110”;
        when “0010” =>
                digito <=“1011011”;
        when “0011” =>
                digito <=“1001111”;
         .....
end case;
```

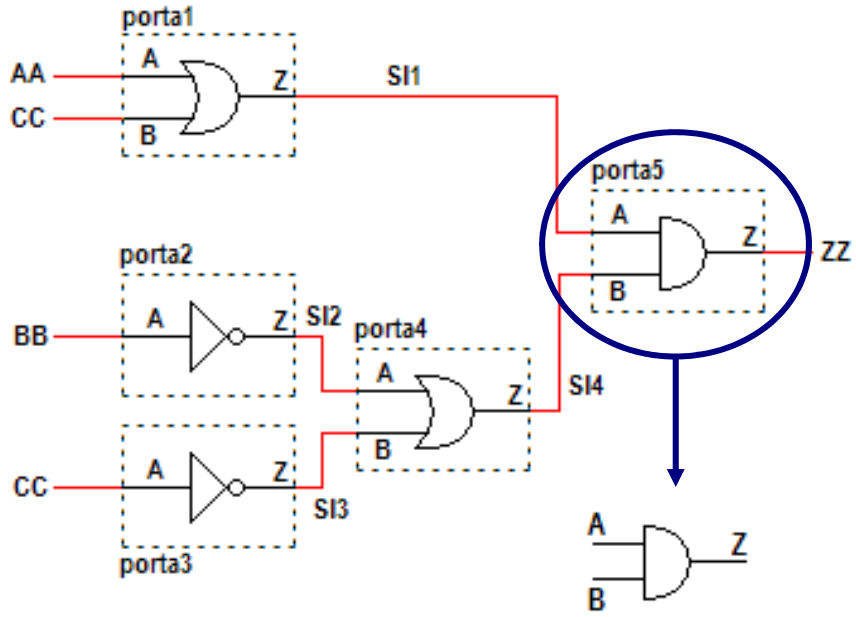


ULA (Unidade Lógica Aritmética, Arithmetic Logic Unit)

```
if (mode = 1) then
        case command is
                when “000” =>
                        answer <= oper1 and oper2;
                when “001” =>
                        answer <= oper1 or oper2;
                when “010” =>
                        answer <= oper1 nor oper2;
                .....
        end case;
else
```

- Modules can be described in several ways
- Depending on the application can not describe internally the accurately form but only their behavior.
- VHDL allows that the behavior be defined in the form of an executable program.





entity name

signal type

```
ENTITY porta_and IS
        PORT(A,B: IN BIT;Z: OUT BIT);
END porta_and
```

ports names

signal direction

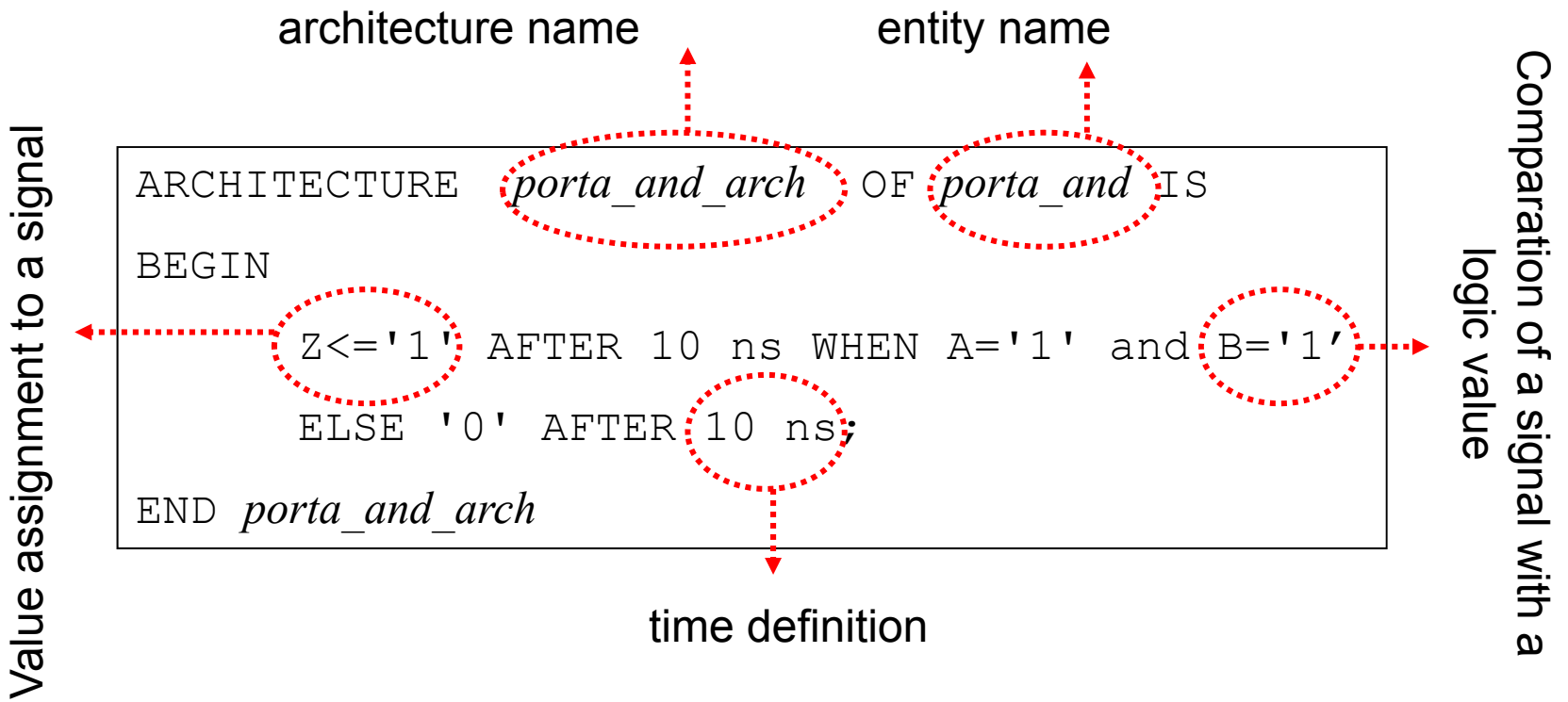
Identifies the declaration end

The capitalized words are reserved.

BEHAVIOURS MODELLING

ARCHITECTURE

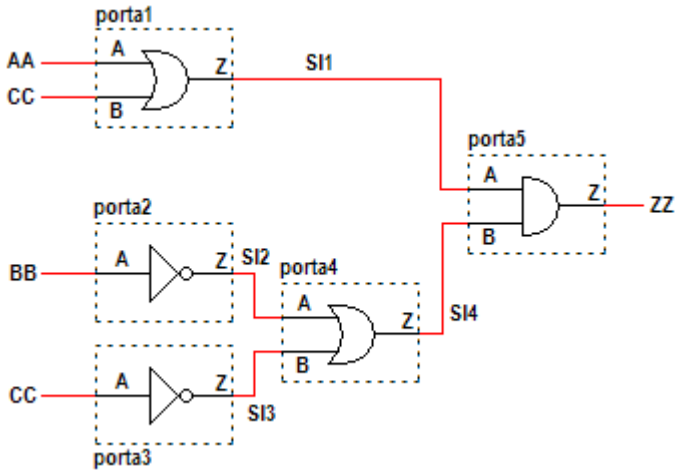
Describe the algorithm executed by the MODULE
It is divided into two main parts
- Declarative Part
- Descriptive Part



```
ENTITY mAND IS
GENERIC(tp:TIME:=10 ns); -- tp - propagation time
PORT(A,B: in BIT; Z: out BIT);
END ENTITY mAND;
ARCHITECTURE mAND_arch of mAND IS
BEGIN
Z<='1' AFTER tp WHEN (A AND B)='1' ELSE '0' AFTER tp;
END mAND_arch;
```

```
ENTITY mOR IS
GENERIC(tp:TIME:=10 ns); -- tp - propagation time
PORT(A,B: in BIT; Z: out BIT);
END ENTITY mOR;
ARCHITECTURE mOR_arch of mOR IS
BEGIN
Z<='1' AFTER tp WHEN (A OR B)='1' ELSE '0' AFTER tp;
END mOR_arch;
```

```
ENTITY mNOT IS
GENERIC(tp:TIME:=10 ns); -- tp - propagation time
PORT(A: in BIT; Z: out BIT);
END ENTITY mNOT;
ARCHITECTURE mNOT_arch of mNOT IS
BEGIN
Z<='1' AFTER tp WHEN (NOT A)='1' ELSE '0' AFTER tp;
END mNOT_arch;
```



COMPONENT SIMULATION

TEST BENCH

Definition of system components

Identifies the doors of the components
Simulation parameters are passed as GENERIC

Creates 8 boolean type signals

Comments after two hyphens followed

Create a process (no list of sensitivities) within the architecture

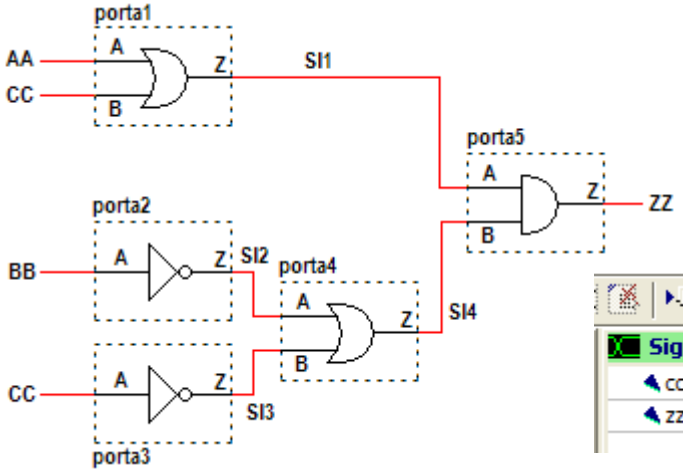
Note how the allocation of signals is made ...

Awaits 100 nanoseconds ....

```
ENTITY testbench IS
END testbench;
ARCHITECTURE testbench_arch OF testbench IS
COMPONENT mAND
          PORT(A,B: in BIT; Z: out BIT);
END COMPONENT;
COMPONENT mOR
          PORT(A,B: in BIT; Z: out BIT);
END COMPONENT;
COMPONENT mNOT
          PORT(A: in BIT; Z: out BIT);
END COMPONENT;
SIGNAL AA,BB,CC,SI1,SI2,SI3,SI4,ZZ:BIT;
BEGIN
AA<='0';
BB<='1';
-- Instantiate componentes
          porta1: mOR PORT MAP (A=>AA,B=>CC,Z=>SI1);
          porta2: mNOT PORT MAP (A=>BB,Z=>SI2);
          porta3: mNOT PORT MAP (A=>CC,Z=>SI3);
          porta4: mOR PORT MAP (A=>SI2,B=>SI3,Z=>SI4);
          porta5: mAND PORT MAP (A=>SI1,B=>SI4,Z=>ZZ);
teste:PROCESS
          BEGIN
          CC<='0';
          WAIT FOR 100 ns;
          CC<='1';
          WAIT FOR 100 ns;
          CC<='0';
          WAIT FOR 100 ns;
END PROCESS teste;
END testbench_arch;
```

Note the figure of the previous acetate ...

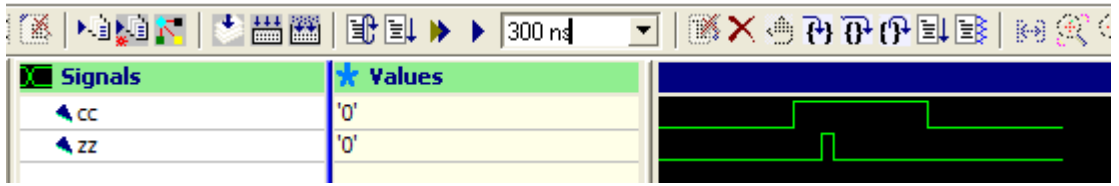
SIMULATION RESULTS



AA<=0, BB<=1

CC=0->1

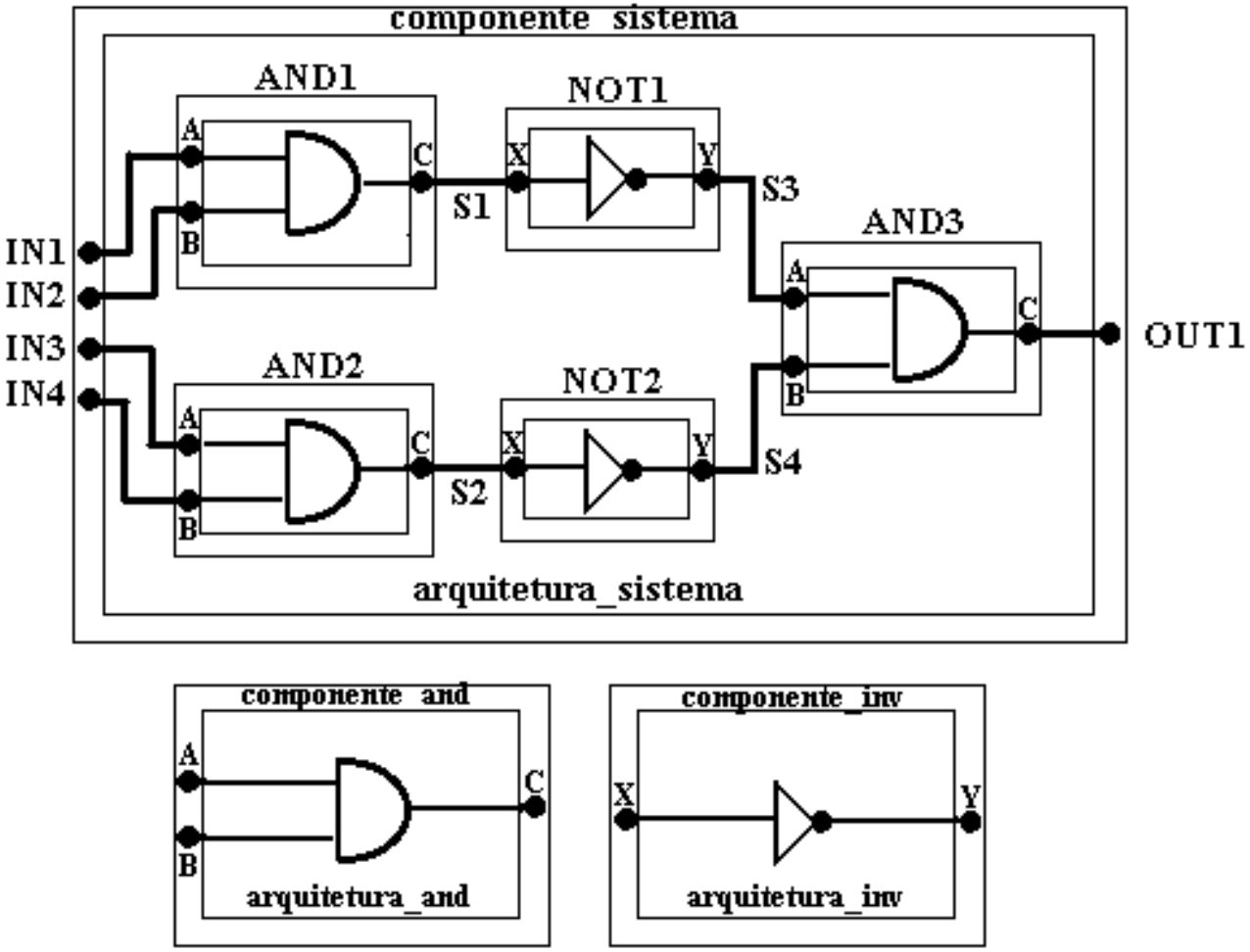
CC \ AA BB	00	01	11	10
0	0	0		
1		0	0	



A more compact alternative version ...

```
ENTITY testbench IS
END testbench;
ARCHITECTURE testbench_arch OF testbench IS
SIGNAL A,B,C,Z,nC,nB,PSI,FI : BIT;
BEGIN
A<='0';
B<='1';
C<='0','1' AFTER 100 ns,'0' AFTER 200 ns;
         nC<=NOT C AFTER 10 ns;
         nB<=NOT B AFTER 10 ns;
         FI<=A OR C AFTER 10 ns;
         PSI<=nB OR nC AFTER 10 ns;
         Z<=FI AND PSI AFTER 10 ns;
END testbench_arch;
```

Other example:



Programa 1

```
-------------------------------------------------------
-- Arquivo componente_inv.vhd
-- Modelo do inversor
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_inv is
port(
        x : in bit;
        y : out bit
        );
end componente_inv;

architecture arquitetura_inv of
componente_inv is

        begin
                y <= not x;
end arquitetura_inv;
```

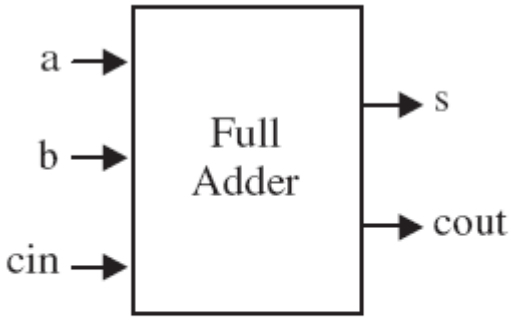
Programa 2

```
-------------------------------------------------------
-- Arquivo componente_and.vhd
-- Modelo da porta AND
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_and is
port(
        a : in bit;
        b : in bit;
        c : out bit
        );
end componente_and;

architecture arquitetura_and of componete_and is

        begin
                c <= a and b;
end arquitetura_and;
```

Programa 3

```
-----------------------------------------------------
-- Arquivo componente_sistema.vhd
-- Modelo da porta AND
-----------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_sistema is
port(
        in1 : in bit;
        in2 : in bit;
        in3 : in bit;
        in4 : in bit;
        out1 : out bit
        );
end componente_sistema;

architecture arquitetura_sistema of componente_sistema is

component componente_and
                port( a: in bit; b : in bit; c : out bit);
end component;

component componente_inv
                port( x: in bit; y : out bit);
end component;

signal s1, s2, s3, s4 : bit;

        begin
                and1 : componente_and port map (a => in1, b => in2, c => s1);
                and2 : componente_and port map (a => in3, b => in4, c => s2);
                and3 : componente_and port map (a => s3, b => s4, c => ut1);
                inv1 : componente_inv port map (x => s1, y => s3);
                inv2 : componente_inv port map (x => s2, y => s4);
end arquitetura_sistema;
```

Example: **Full Adder**



a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

- Modelling from logic gates

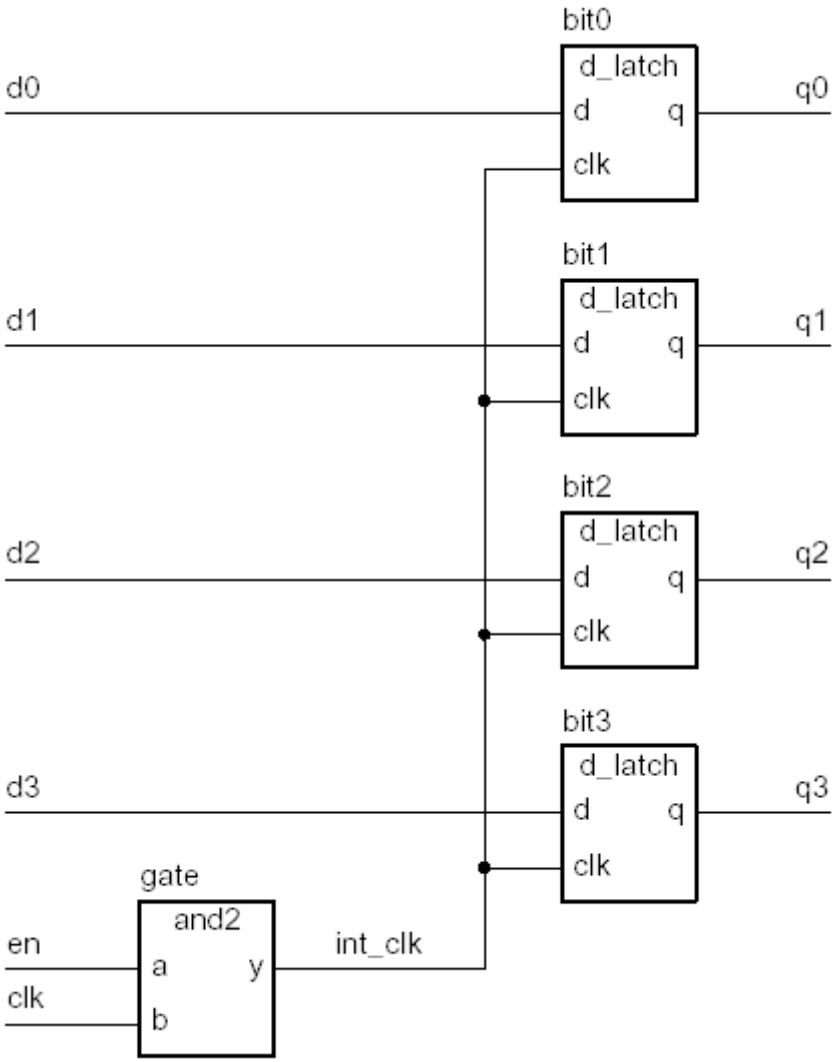
- Modelling behaviour

Examples: **4 bits Register**

TO IMPLEMENT ANY SYNCHRONOUS CIRCUIT IS NECESSARY FORCE VHDL TO BE SEQUENTIAL AND NOT CONCURRENT. HOW?

***process_name*: PROCESS** (*sensibilities_list*)
BEGIN
(…)
END PROCESS *process_name*

Example: **4 bits Register (cont)**



- Modelling behaviour Latch D (td=2ns)
- Modelling behaviour gate (td=2ns)
- Modelling register
- Testing Register

Reserved Words in VHDL

From VHDL 87:	ENTITY	OPEN	WAIT
	EXIT	OR	WHEN
ABS	FILE	OTHERS	WHILE
ACCESS	FOR	OUT	WITH
AFTER	FUNCTION	PACKAGE	XOR
ALIAS	GENERATE	PORT	
ALL	GENERIC	PROCEDURE	*From VHDL 93:*
AND	GUARDED	PROCESS	
ARCHITECTURE	IF	RANGE	GROUP
ARRAY	IN	RECORD	IMPURE
ASSERT	INOUT	REGISTER	INERTIAL
ATTRIBUTE	IS	REM	LITERAL
BEGIN	LABEL	REPORT	POSTPONED
BLOCK	LIBRARY	RETURN	PURE
BODY	LINKAGE	SELECT	REJECT
BUFFER	LOOP	SEVERITY	ROL
BUS	MAP	SIGNAL	ROR
CASE	MOD	SUBTYPE	SHARED
COMPONENT	NAND	THEN	SLA
CONFIGURATION	NEW	TO	SLL
CONSTANT	NEXT	TRANSPORT	SRA
DISCONNECT	NOR	TYPE	SRL
DOWNTO	NOT	UNITS	UNAFFECTED
ELSE	NULL	UNTIL	XNOR
ELSIF	OF	USE	
END	ON	VARIABLE	

END

Ref. **LINGUAGEM DE DESCRIÇÃO DE HARDWARE, Prof. Anderson Royes Terroso**