Elementos de Lógica Digital II

Aula 2 – Introduction to VHDL

Prof. Vanderlei Bonato - vbonato@icmc.usp.br

Sumário

- History
- VHDL Structure
- Sequencial and Parallel Execution
- Signal and Variable
- Data Types
- Operations
- State Machine Example
- Exercises

Concepts

- **VHDL is the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language**
- **VHDL is an international standard specification language for describing digital hardware used by industry worldwide**
- **VHDL enables hardware modeling from the gate to system level**
- **VHDL provides a mechanism for digital design and reusable design documentation**

History of VHDL

- **Launched in 1980**
- **Aggressive effort to advance state of the art**
- **Object was to achieve significant gains in VLSI technology**
- **Need for common descriptive language**
- **In July 1983, a team of Intermetrics, IBM and Texas Instruments were awarded a contract to develop VHDL**

History of VHDL

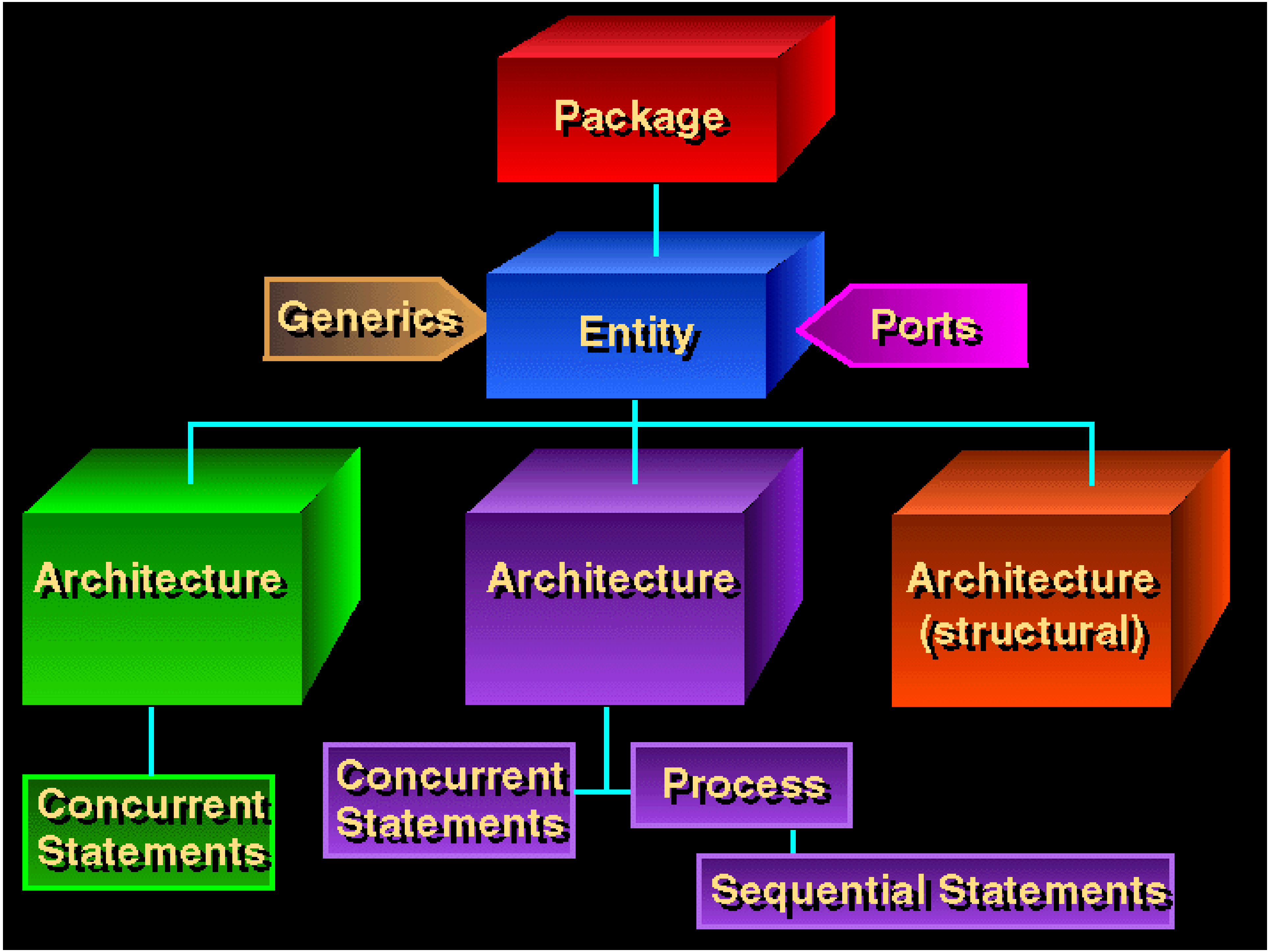
- **In August 1985, the final version of the language under government contract was released: VHDL Version 7.2**
- **In December 1987, VHDL became IEEE Standard 1076-1987 and in 1988 an ANSI approved standard**
- **In September 1993, VHDL was restandardized to clarify and enhance the language (IEEE Standard 1076-1993)**
- **Since then there has been many other VHDL standard revision**

How about Quartus II 9.1

- **The Quartus II software supports a subset of the constructs defined by the IEEE Std 1076-1987 and IEEE Std 1076-1993**
  - **It supports only those constructs that are relevant to logic synthesis**
- **The Quartus II software also supports packages defined in IEEE Std 1076.3-1997**
- **The Quartus II software contains support for VHDL 2008, IEEE Std 1076-2008**

Why Use VHDL?

- **Provides technology independence**
- **Describes a wide variety of digital hardware**
- **Eases communication through standard language**
- **Allows for better design management**
- **Provides a flexible design language**


Package
Generics
Entity
Ports
Architecture
Architecture
Architecture (structural)
Concurrent Statements
Concurrent Statements
Process
Sequential Statements

Sample VHDL Design Process

- **Problem: design a single bit half adder with carry and enable**
- **Specifications**
  - **Passes results only on enable high**
  - **Passes zero on enable low**
  - **Result gets x plus y**
  - **Carry gets any carry of x plus y**

Entity Declaration

- **An entity declaration describes the interface of the component**
- **PORT clause indicates input and output ports**
- **An entity can be thought of as a symbol for a component**
- **Generics may be added for readability, maintenance and configuration**

Entity Declaration

```
ENTITY half_adder IS
     PORT (x, y, enable: IN bit;
               carry, result: OUT bit);

END half_adder;
```

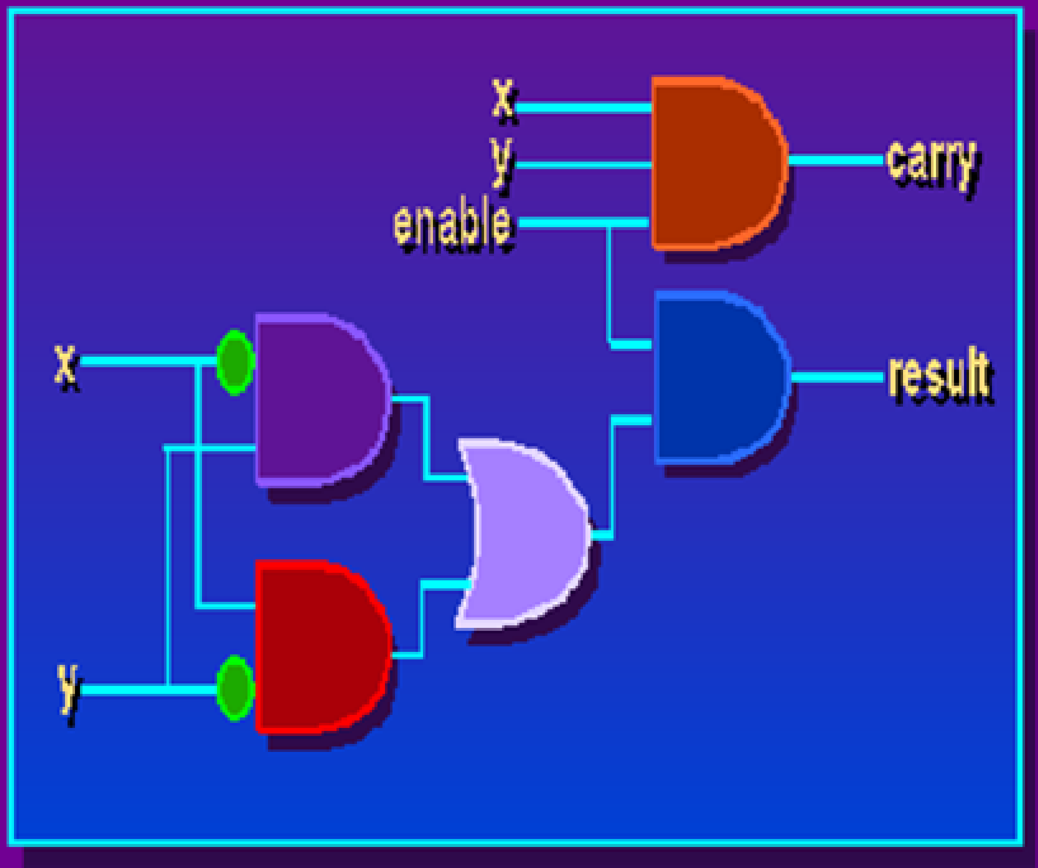
Architecture Declaration

- **Architecture declarations describe the operation of the component**

- **Many architectures may exist for one entity, but only one may be active at a time**

- **An architecture is similar to a schematic of the component**

```
ARCHITECTURE behavior1 OF
half_adder IS BEGIN
   PROCESS (enable, x, y)
   BEGIN
   IF (enable = '1') THEN
      result <= x XOR y;
      carry <= x AND y;
   ELSE
      carry <= '0'
      result <= '0';
   END PROCESS;
END behavior1;
```



Packages and Libraries

- **User defined constructs declared inside architectures and entities are not visible to other entities**
  - **Subprograms, user defined data types, and constants can not be shared**
- **Packages and libraries provide the ability to reuse constructs in multiple entities and architectures**

Sequential and Concurrent Statements

- **VHDL provides two different types of execution: sequential and concurrent**
- **Different types of execution are useful for modeling of real hardware**
  - **Supports various levels of abstraction**
- **Sequential statements view hardware from a "programmer" approach**
- **Concurrent statements are order-independent and asynchronous**

Sequential Statements

- **Sequential statements run in top to bottom order**
- **Sequential execution most often found in behavioral descriptions**
- **Statements inside PROCESS execute sequentially**

Concurrent Statements

- **All concurrent statements occur simultaneously**
- **How are concurrent statements processed?**
- **Simulator time does not advance until all concurrent statements are processed**
- **Some concurrent statements**
  - **Block, process, assert, signal assignment, procedure call, component instantiation**

VHDL Processes

- Assignments executed sequentially
- Sequential statements
  - {Signal, variable} assignments
  - Flow control
    - if <condition> then <statements> else <statements> end if;
    - for <range> loop <statements> end loop;
      while <condition> loop <statements> end loop;
    - case <condition> is when <value> => <statements>;
      when <value> => <statements>;
      when others => <statements>;
      end case;
  - Wait on <signal> until <expression> for <time>;
  - Assert <condition> report <string> severity <level>;

VHDL Processes

- A VHDL process statement is used for all behavioral descriptions

```
[process_label :] PROCESS

[(sensitivity_list)]


  process_declarations


BEGIN


  process_statements


END PROCESS [process_label];
```

Process Example - Carry Bit

```
Carry: PROCESS(A, B, Cin)

 BEGIN

  IF (A = '1' and B = '1') THEN

      Cout <= '1';

  ELSIF (A = '1' and Cin = '1') THEN

      Cout <= '1';

  ELSIF (B = '1' and Cin = '1') THEN

      Cout <= '1';

  ELSE

      Cout <= '0';

  END IF;

END PROCESS Carry;
```

A Design Example - 2-bit Counter

```
ENTITY count2 IS
    PORT (clock : IN BIT;
          q1, q0: OUT BIT);
END count2;

ARCHITECTURE behavior OF count2 IS

BEGIN
    count_up: PROCESS (clock)
        VARIABLE count_value: NATURAL := 0;

    BEGIN
        IF clock='1' THEN
            count_value := (count_value+1) MOD 4;
            q0 <= bit'val(count_value MOD 2);
            q1 <= bit'val(count_value/2);
        END IF;
    END PROCESS count_up;
END behavior;
```

Signals vs Variables

- **Variables**
  - **Used for local storage of data**
  - **Generally not available to multiple components and processes**
  - **All variable assignments take place immediately**
  - **Variables are more convenient than signals for the storage of data**
  - **Variables may be made global**
- **Signals**
  - **Used for communication between components**
  - **Signals can be seen as real, physical signals**
  - **Some delay must be incurred in a signal assignment**

Assignments

ARCHITECTURE test1 OF
test_mux IS
SIGNAL a : BIT := '1';
SIGNAL b : BIT := '0';
BEGIN
...more statements...
a <= b;
b <= a;
...more statements...
END test1;
	ARCHITECTURE test2 OF test_mux IS BEGIN
	PROCESS (result)
	VARIABLE a : BIT := '1';
	VARIABLE b : BIT := '0';
	BEGIN
	...more statements...
	a := b;
	b := a;
	...more statements...
	END PROCESS;
	END test2;

Signal x Variable Behaviour

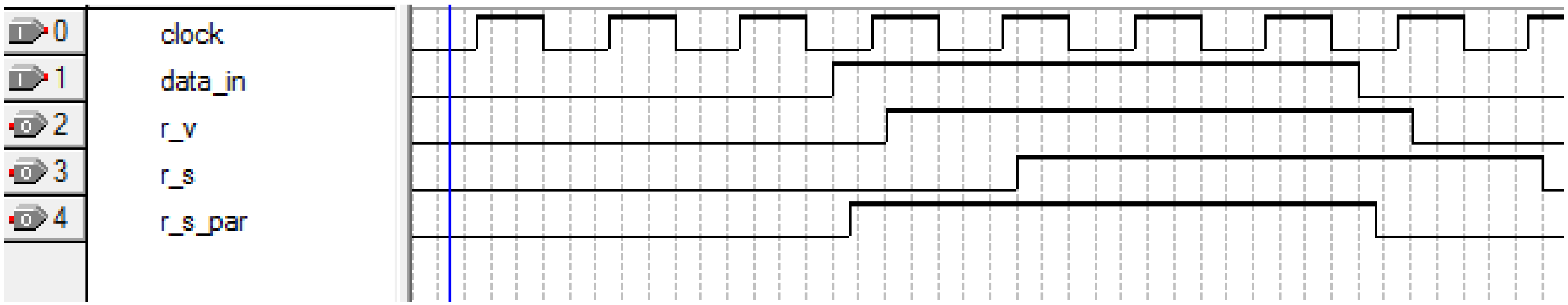
```
ENTITY aulavhdl IS
   PORT (clock, data_in : IN BIT;
        r_v, r_s, r_s_par: OUT BIT);
END aulavhdl;

ARCHITECTURE behavior OF aulavhdl IS
   signal a_s, a_s_par: BIT := '0';
BEGIN
   PROCESS (clock)
      variable a_v: BIT := '0';
   BEGIN
      IF clock='1' THEN
         a_v := data_in;
         r_v <= a_v;

         a_s <= data_in;
         r_s <= a_s;
      END IF;
   END PROCESS;
   a_s_par <= data_in;
   r_s_par <= a_s_par;
END behavior;
```

Signal x Variable Behaviour

- Percebam a diferença de comportamento do "signal" dentro e fora do processo!
- Quanto a "variable" não há surpresa, pois é utilizada somente dentro do processo



Data Types

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From −2,147,483,647 to +2,147,483,647
SIGNED	From −2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above

Dealing with Data Types

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;                -- 1D
                                                              -- array
TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;         -- 2D
                                                              -- array
TYPE mem2 IS ARRAY (0 TO 3) OF byte;                          -- 1Dx1D
                                                              -- array
TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7);      -- 1Dx1D
                                                              -- array
SIGNAL a: STD_LOGIC;                                 -- scalar signal
SIGNAL b: BIT;                                       -- scalar signal
SIGNAL x: byte;                                      -- 1D signal
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);             -- 1D signal
SIGNAL v: BIT_VECTOR (3 DOWNTO 0);                   -- 1D signal
SIGNAL z: STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);        -- 1D signal
SIGNAL w1: mem1;                                     -- 2D signal
SIGNAL w2: mem2;                                     -- 1Dx1D signal
SIGNAL w3: mem3;                                     -- 1Dx1D signal
```

Scalar Assignments

```
x(2) <= a;              -- same types (STD_LOGIC), correct indexing
y(0) <= x(0);           -- same types (STD_LOGIC), correct indexing
z(7) <= x(5);           -- same types (STD_LOGIC), correct indexing
b <= v(3);              -- same types (BIT), correct indexing
w1(0,0) <= x(3);        -- same types (STD LOGIC), correct indexing
w1(2,5) <= y(7);        -- same types (STD_LOGIC), correct indexing
w2(0)(0) <= x(2);       -- same types (STD_LOGIC), correct indexing
w2(2)(5) <= y(7);       -- same types (STD_LOGIC), correct indexing
w1(2,5) <= w2(3)(7);    -- same types (STD_LOGIC), correct indexing
```

Vector Assignments

```
x <= "11111110";
y <= ('1','1','1','1','1','1','0','Z');
z <= "11111" & "000";
x <= (OTHERS => '1');
y <= (7 =>'0', 1 =>'0', OTHERS => '1');
z <= y;
y(2 DOWNTO 0) <= z(6 DOWNTO 4);
w2(0)(7 DOWNTO 0) <= "11110000";
w3(2) <= y;
z <= w3(1);
z(5 DOWNTO 0) <= w3(1)(2 TO 7);
w3(1) <= "00000000";
w3(1) <= (OTHERS => '0');
w2 <= ((OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'));
w3 <= ("11111100", ('0','0','0','0','Z','Z','Z','Z',),
       (OTHERS=>'0'), (OTHERS=>'0'));
w1 <= ((OTHERS=>'Z'), "11110000" ,"11110000", (OTHERS=>'0'));
```

Ilegal Assignments

```
------- Illegal scalar assignments: --------------------
b <= a;               -- type mismatch (BIT x STD_LOGIC)
w1(0)(2) <= x(2);     -- index of w1 must be 2D
w2(2,0) <= a;         -- index of w2 must be 1Dx1D
```

```
------ Illegal array assignments: ----------------------
x <= y;                           -- type mismatch
y(5 TO 7) <= z(6 DOWNTO 0);       -- wrong direction of y
w1 <= (OTHERS => '1');            -- w1 is a 2D array
w1(0, 7 DOWNTO 0) <="11111111";   -- w1 is a 2D array
w2 <= (OTHERS => 'Z');            -- w2 is a 1Dx1D array
w2(0, 7 DOWNTO 0) <= "11110000";  -- index should be 1Dx1D
```

DOWNTO and TO

```
SIGNAL x: BIT;
-- x is declared as a one-digit signal of type BIT.

SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
-- y is a 4-bit vector, with the leftmost bit being the MSB.

SIGNAL w: BIT_VECTOR (0 TO 7);
-- w is an 8-bit vector, with the rightmost bit being the MSB.
```

```
x <= '1';
-- x is a single-bit signal (as specified above), whose value is
-- '1'. Notice that single quotes (' ') are used for a single bit.

y <= "0111";
-- y is a 4-bit signal (as specified above), whose value is "0111"
-- (MSB='0'). Notice that double quotes (" ") are used for
-- vectors.

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

Bit Levels

- BIT (and BIT_VECTOR): 2-level logic ('0', '1')

- STD_LOGIC (and STD_LOGIC_VECTOR): 8-valued logic system introduced in the IEEE 1164 standard.

'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	

Most of the std_logic are intended for simulation only!

ULOGIC

- STD_ULOGIC (STD_ULOGIC_VECTOR): 9-level logic system introduced in the IEEE 1164 standard ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '–').

- STD_LOGIC system described above is a *subtype* of STD_ULOGIC. The latter includes an extra logic value, 'U', which stands for unresolved. Thus, contrary to STD_LOGIC, conflicting logic levels are not automatically resolved here, so output wires should never be connected together directly. However, if two output wires are never supposed to be connected together, this logic system can be used to detect design errors.

SIGNED and UNSIGNED

- Their syntax similar to STD_LOGIC_VECTOR
- SIGNED and UNSIGNED are intended mainly for arithmetic operations
- Logic operations are not allowed

```
SIGNAL x: SIGNED (7 DOWNTO 0);
SIGNAL y: UNSIGNED (0 TO 3);
```

Data Conversion

- **VHDL does not allow direct operations between data of different types**
- **Conversions are necessary**
- **Several data conversion functions can be found in the std_logic_arith package of IEEE library**

std_logic_arith Conversion Functions

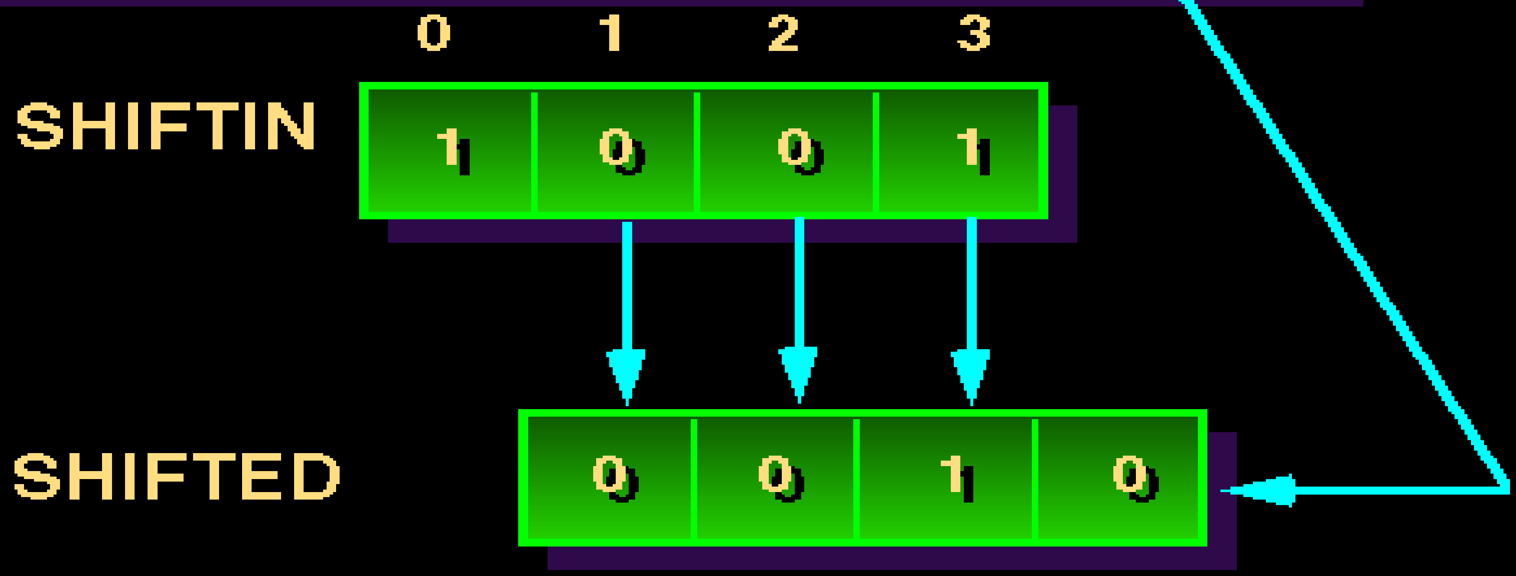
- conv_integer(p) : Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an INTEGER value. Notice that STD_LOGIC_VECTOR is not included.
- conv_unsigned(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an UNSIGNED value with size b bits.
- conv_signed(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to a SIGNED value with size b bits.
- conv_std_logic_vector(p, b): Converts a parameter p of type INTEGER, UNSIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size b bits.

Operators

Operator type	Operators	Data types
Assignment	<=, :=, =>	Any
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR
Arithmetic	+, −, *, /, ** (mod, rem, abs)[♦]	INTEGER, SIGNED, UNSIGNED
Comparison	=, /=, <, >, <=, >=	All above
Shift	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenation	&, (,,,)	Same as for logical operators, plus SIGNED and UNSIGNED

The concatentation operator &

```
VARIABLE shifted, shiftin : BIT_VECTOR (0 TO 3) :
....
shifted := shiftin (1 TO 3) & '0';
```



The exponentiation operator **

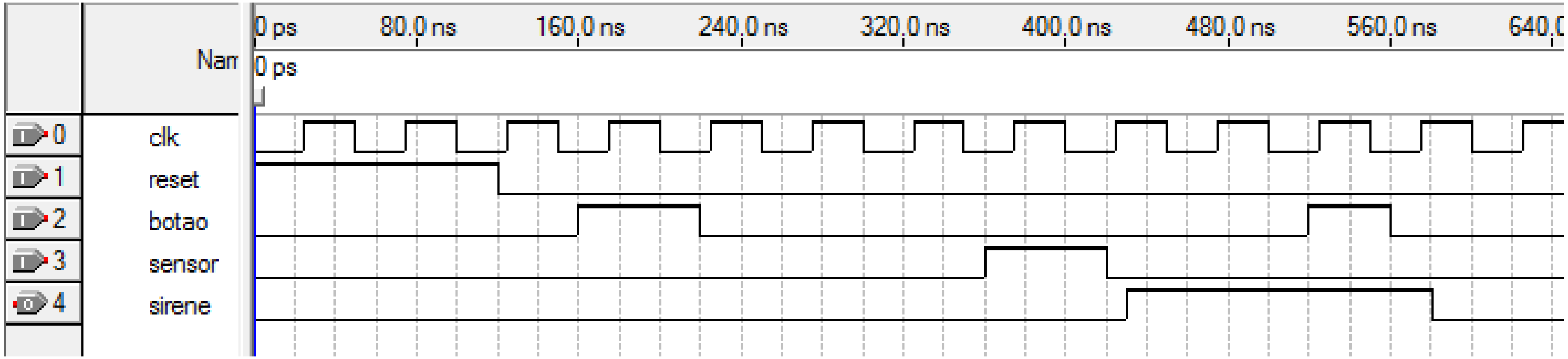
```
x := 5**5   -- 5^5, OK
y := 0.5**3 -- 0.5^3, OK
x := 4**0.5 -- 4^0.5, bad
y := 0.5**(-2) -- 0.5^(-2), OK
```

State Machine – Alarm Example

```vhdl
ENTITY alarme IS
    PORT (clk,reset : IN BIT;
          sensor, botao: IN BIT;
          sirene: OUT BIT);
END alarme;
ARCHITECTURE behavior OF alarme IS
 TYPE estados IS (desligado, ativado, disparado);
 SIGNAL estado: estados;
BEGIN
    PROCESS (clk)
    BEGIN
        if (reset = '1') then
            estado <= desligado;
            sirene <= '0';
        elsif (clk'event) and (clk = '1') then
            case estado is
                WHEN desligado =>
                    sirene <= '0';
                    if (botao = '1') then
                        estado <= ativado;
                    end if;
                WHEN ativado =>
                    if (sensor = '1') then
                        estado <= disparado;
                    end if;
                WHEN disparado =>
                    sirene <= '1';
                    if (botao = '1') then
                        estado <= desligado;
                    end if;
            end case;
        end if;
    END PROCESS;
END behavior;
```

Alarm State Machine Waveform

- **Compare the VHDL source code and identify what is the difference from the alarm state machine seen in the first class**



Exercises 1

- **Deixar o alarme com o mesmo comportamento da maquina de estados visto na aula anterior**

- **Adicionar um temporizador no alarme para que o mesmo toque somente se o sensor ficar acionado por mais de 5 segundos e desligue o alarme (ir para o estado inicial) se o sensor ficar desacionado por mais de 20 segundos**

Exercises 2

- **Implemente um elevador para 4 andares com as seguintes entradas e saídas:**
  - **Entrada:**
    - **um botão de chamada externo**
    - **4 botoes internos para indicar o andar**
    - **sensor de fechamento da porta**
    - **sensor de presença na porta**
    - **um sensor de presenca de elevador por andar**
  - **Saída**
    - **Motor elevador (liga/desliga)**
    - **Direção_elevador (sobe/desce)**
    - **Porta porta (liga/desliga)**
    - **Direcao_porta (abre/fecha)**

Tips

- **The ENTITY name and the file name must the same**
- **Physical and time data types are not synthesizable to FPGAs**
  - **ohm, kohm**
  - **fs, ps, ns, um, ms, min, hr**

And more ...

- **Function**
  - **Produce a single return value**
  - **Requires a RETURN statement**
- **Procedure**
  - **Produce many output values**
  - **Do not require a RETURN statement**
- **Testbench**
  - **Generate stimulus for simulation**
  - **Compare output responses with expected values**

References

- **Pedroni, Volnei A. Circuit Design with VHDL, MIT Press, 2004**

- **DARPA/Tri-Services RASSP Program**
  - **http://www.vhdl.org/rassp/**