DEPEL - Departamento
de Engenharia Elétrica

– Tópicos Especiais 2 –
Capítulo 3 – Introdução ao VHDL, sintaxe básica, tipo de dados e atribuições

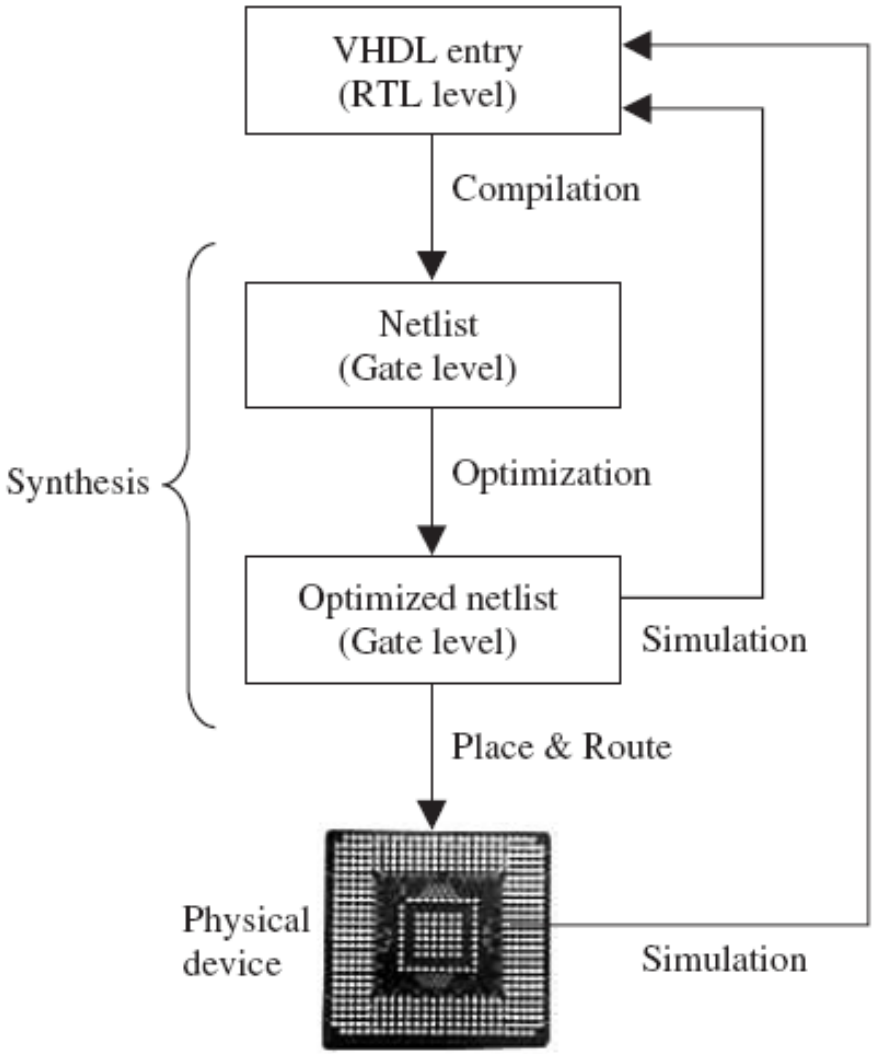
Prof. Alan Petrônio Pinheiro - 2011

Introdução

- VHDL é uma linguagem de descrição de hardware
  - *Hardware Description Language*
- Descreve operação e comportamento do circuito
  - passível de implementação em circuito físico
- Formulado pelo *United States Department of Defense* (198X)
  - IEEE 1076 e 1164
- Também usado em síntese e simulação
- Aplicação: PLD e ASICs

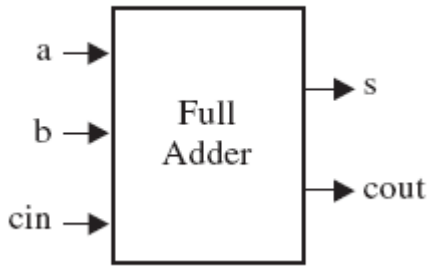
- Característica:
  - altamente paralelo: diferente entre linguagens de computadores (código x programa)
  - classe: linguagem de especificação ou modelamento (não linguagem de computador)
  - fortemente tipada
  - não é sensitive case
  - portabilidade (*e.g.*,PLD, VLSI)
- Netlist
  - Expressa a conectividade de um circuito a partir de uma hierarquia de blocos
- Por que não C/C++ ?

Etapas


VHDL entry
(RTL level)
Compilation
Netlist
(Gate level)
Synthesis
Optimization
Optimized netlist
(Gate level)
Simulation
Place & Route
Physical
device
Simulation

UFSJ

Conversão VHDL para circuito

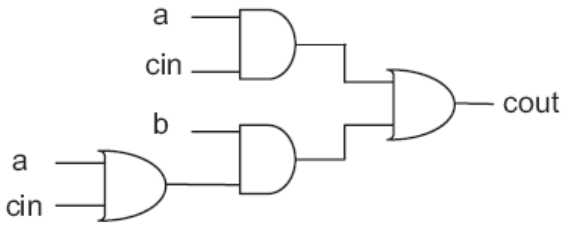


a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

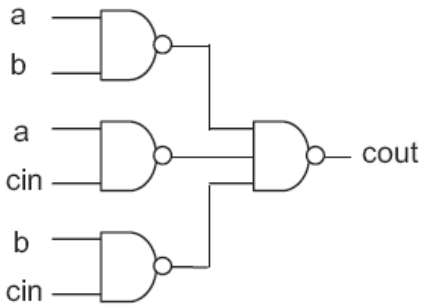
$$s = a \oplus b \oplus cin$$
$$cout = a.b + a.cin + b.cin$$

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
       s, cout: OUT BIT);
END full_adder;
-------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```

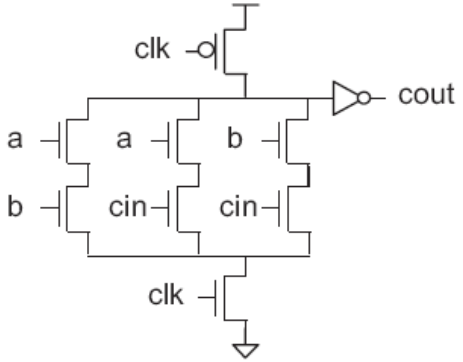
(a)



(b)



(c)

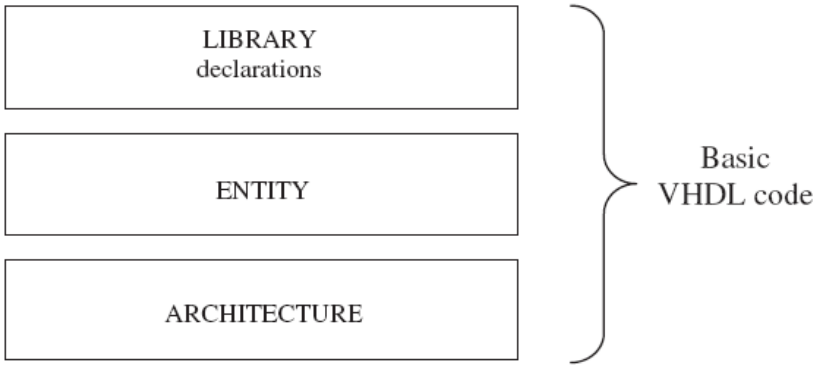


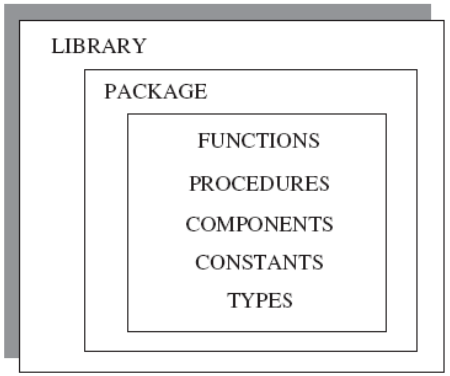
(d)

UFSJ

- Código VHDL é composto por, pelo menos, 3 seções:
  - **LIBRARY**: lista de bibliotecas
    - Exemplo: ieee, std, work, etc.
  - **ENTITY**: especifica pinos I/O do circuito
  - **ARCHITECTURE**: contém o código VHDL propriamente descrito e descreve como o circuito deve comportar-se.

- **Library**
  - Conjunto de partes de códigos muito usados (re-uso)
  - Escritos na forma de FUNCTIONS, PROCEDURES, ou COMPONENTS e compilados para produzirem a biblioteca





- Declarações de biblioteca
  - ieee.std_logic_1164 (ieee library)
  - standard (std library)
  - work (work library)

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

- Exemplo

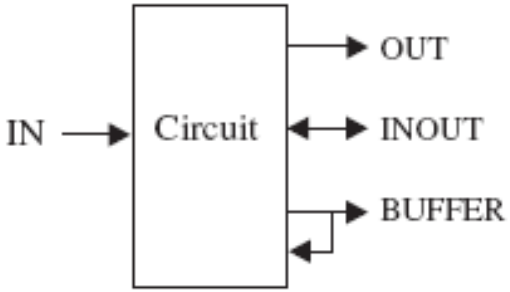
```
LIBRARY ieee;                       -- A semi-colon (;) indicates
USE ieee.std_logic_1164.all;        -- the end of a statement or

LIBRARY std;                        -- declaration, while a double
USE std.standard.all;               -- dash (--) indicates a comment.

LIBRARY work;
USE work.all;
```

- LIBRARY ieee mais empregadas:
  - std_logic_1164
  - std_logic_arith
  - std_logic_signed
  - std_logic_unsigned

- **ENTITY**

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

- Os modos dos sinais (pinos) podem ser:
  - **IN, OUT, INOUT, BUFFER**



- Tipo sinal:
  - **BIT, STD_LOGIC, INTEGER,** etc.

- Exemplo:

```
ENTITY nand_gate IS
   PORT (a, b : IN BIT;
         x : OUT BIT);
END nand_gate;
```

UFSJ

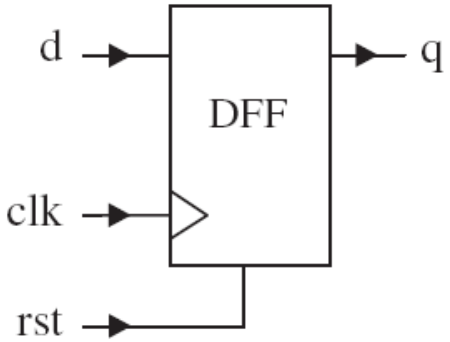
- ARCHITECTURE
  - Sintaxe:

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

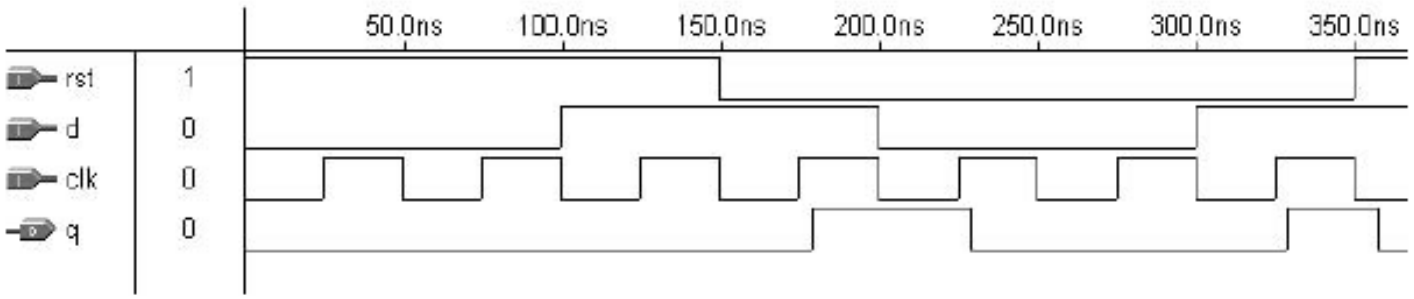
  - Declarações
    - Sinais e constantes
  - Exemplo

```
ARCHITECTURE myarch OF nand_gate IS
BEGIN
   x <= a NAND b;
END myarch;
```

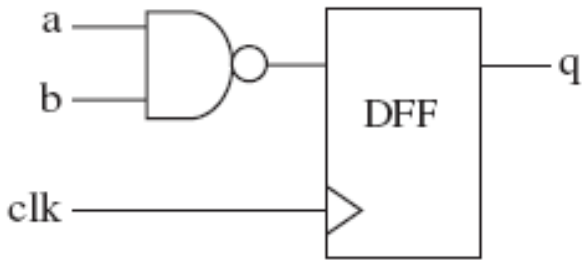
- Exemplos:




```
-------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------
ENTITY dff IS
   PORT ( d, clk, rst: IN STD_LOGIC;
          q: OUT STD_LOGIC);
END dff;
-------------------------------------
ARCHITECTURE behavior OF dff IS
BEGIN
   PROCESS (rst, clk)
   BEGIN
      IF (rst='1') THEN
         q <= '0';
      ELSIF (clk'EVENT AND clk='1') THEN
         q <= d;
      END IF;
   END PROCESS;
END behavior;
-------------------------------------
```



UFSJ




```
-------------------------------------
ENTITY example IS
   PORT ( a, b, clk: IN BIT;
          q: OUT BIT);
END example;
-------------------------------------
ARCHITECTURE example OF example IS
   SIGNAL temp : BIT;
BEGIN
   temp <= a NAND b;
   PROCESS (clk)
   BEGIN
      IF (clk'EVENT AND clk='1') THEN q<=temp;
      END IF;
   END PROCESS;
END example;
-------------------------------------
```

- Exemplo para implementação:

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY luzes IS
5      PORT ( CLOCK_50 : IN STD_LOGIC;
6             SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7             LEDR : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
8  END luzes;
9
10 ARCHITECTURE comportamento OF luzes IS
11 BEGIN
12     PROCESS (CLOCK_50)
13     BEGIN
14         IF(RISING_EDGE(CLOCK_50)) THEN
15             LEDR <= SW;
16         END IF;
17     END PROCESS;
18 END comportamento;
```

SW[0]	PIN_N25
SW[1]	PIN_N26
SW[2]	PIN_P25
SW[3]	PIN_AE14
SW[4]	PIN_AF14
SW[5]	PIN_AD13
SW[6]	PIN_AC13
SW[7]	PIN_C13
LEDR[0]	PIN_AE23
LEDR[1]	PIN_AF23
LEDR[2]	PIN_AB21
LEDR[3]	PIN_AC22
LEDR[4]	PIN_AD22
LEDR[5]	PIN_AD23
LEDR[6]	PIN_AD21
LEDR[7]	PIN_AC21
CLOCK_50	PIN_N2

Tipos de dados

- Pacotes com definições e tipos de dados:
  - Biblioteca STD, pacote standard
    - BIT, BOOLEAN, INTEGER, REAL
  - Biblioteca IEEE, pacote std_logic_1164
    - STD_LOGIC, STD_ULOGIC
  - Biblioteca IEEE, pacote std_logic_arith
    - Signed, unsigned
    - Funções de conversão como: conv_integer(p), conv_unsigned(p,b), conv_signed(p,b) e conv_std_logic_vector(p,b).
  - Biblioteca IEEE, pacotes std_logic_signed e std_logic_unsigned
    - Funções que permitem operações com dados STD_LOGIC_VECTOR do tipo SIGNED ou UNSIGNED, respectivamente.

- **BIT e BIT_VECTOR**
  - '0' ou '1'
  - Exemplos de declarações e atribuições:

```
SIGNAL x: BIT;
x <= '1';
```

  - x é declarado como um simples bit.
  - Aspas simples para bit

```
SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
y <= "0111";
```

  - y é um vetor de 4 bits, sendo o mais a esquerda o MSB.
  - MSB = 0 e aspas duplas são usadas para vetores.

```
SIGNAL w: BIT_VECTOR (0 TO 7);
w <= "01110001";
```

  - w é um vetor de 8 bits com o mais a direita sendo o MSB.
  - MSB = 1

- **STD_LOGIC e STD_LOGIC_VECTOR**
  - ‘X’ Forcing Unknown; ‘0’ Forcing Low; ‘1’ Forcing High; ‘Z’ High impedance; ‘W’ Weak unknown; ‘L’ Weak low; ‘H’ Weak high; ‘–’ Don’t care
  - Exemplos:

```
SIGNAL x: STD_LOGIC;
```

- x é declarado como tipo simples

```
SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001";
```

- y é um vetor de 4 bits, como o bit mais a esquerda sendo MSB e valor inicial "0001".

- **BOOLEAN**
  - Verdadeiro e falso.
- **INTEGER**
  - Inteiro 32-bits (-2,147,483,647 +2,147,483,647).
- **NATURAL**
  - Inteiros não-negativos (0 a +2,147,483,647)
- **REAL**
  - Números reais (-1.0E38 A +1.0E38) (*não sintetizável*)
- **SIGNED** e **UNSIGNED**
  - Tem semelhança com `STD_LOGIC_VECTOR`, contudo permitem operações aritmétcas que são tipicas de dados inteiros

- Exemplos:

```
x0 <= '0';
```

  - bit, std_logic ou std_ulogic

```
x1 <= "00011111";
```

  - bit_vector, std_logic_vector, std_ulogic_vector, signed, ou unsigned

```
x3 <= "101111"
```

  - Representação binária de 47

```
x4 <= B"101111"
```

  - Representação binária de 47

```
x5 <= O"57"
```

  - Representação octal de 47

```
x6 <= X"2F"
```

  - Representação hexadecimal de 47

```
n <= 1200;
```

  - inteiro

```
y <= 1.2E-5;
```

  - Valor tipo real (valor *não sintetizável*)

- Exemplos de características de tipagem:

```
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);`
- Operação legal

`b(0) <= a;`
- Operação legal

`c <= d(5);`
- Operação legal

`d(0) <= c;`
- Operação legal

`a <= c;`
- Operação ilegal (tipos diferentes: BIT x STD_LOGIC)

`b <= d;`
- Operação ilegal (tipos diferentes: BIT_VECTOR x STD_LOGIC_VECTOR)

`e <= b;`
- Operação ilegal (tipos diferentes: INTEGER x BIT_VECTOR)

`e <= d;`
- Operação ilegal (tipos diferentes: INTEGER x STD_LOGIC_VECTOR)

- Tipagem de dados definida pelo usuário

```
TYPE meu_tipo IS RANGE -32 TO 32;
TYPE contagem_atleticana IS RANGE 0 TO 24;
```

– Enumerações:

```
TYPE my_logic IS ('0', '1', 'Z');
```

  - Usuário definiu um subgrupo do std_logic

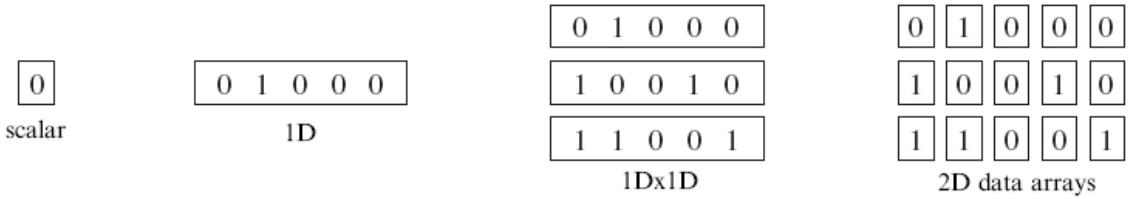
```
TYPE estado IS (idle, forward, backward,
  stop);
```

  - Tipo enumerado

```
TYPE cor IS (verme, verde, azul, branco);
```

  - Tipo enumerado

- Matrizes/vetores
  - 1D/2D ... Demais geralmente não-sintetizáveis



  - Tipos de dados aceitos:
    - Escalares: `BIT, STD_LOGIC, STD_ULOGIC e BOOLEAN`
    - Vetores: `BIT_VECTOR, STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, INTEGER, SIGNED e UNSIGNED`
  - Instanciação:

```
TYPE nome_tipo IS ARRAY (tamanho) OF tipo_dado;
SIGNAL nome_sinal: nome_tipo [:= inicializacao];
```

– Exemplo matriz 1Dx1D:

```
TYPE linha IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
-- 1D array
TYPE matriz IS ARRAY (0 TO 3) OF linha;
-- 1Dx1D array
SIGNAL x: matriz;
-- 1Dx1D signal
```

– Exemplo alternativo matriz 1Dx1D:

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

- Exemplo matriz 2D:

```
TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF
  STD_LOGIC;
```

- Inicialização:

```
:="0001";                                    -- 1D
:=('0','0','0','1')                          -- 1D
:=(('0','1','1','1'),('1','1','1','0')); -- 1Dx1D ou 2D
```

- Atribuições e acesso à vetores:

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;-- 1D
TYPE array1 IS ARRAY (0 TO 3) OF row;        -- 1Dx1D
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

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

```
-- Exemplos de atribuicoes de escalares: todas legais
-- pois a base eh a mesma: STD_LOGIC
x(0) <= y(1)(2); -- 2 pares parenteses pq y eh 1Dx1D
x(1) <= v(2)(3); -- 2 pares parenteses pq v eh 1Dx1D
x(2) <= w(2,1); -- 1 par parenteses pq w eh 2D
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
```

```
-- Exemplos de atribuicoes de vetores
x <= y(0); -- valido(mesmo tipo dado: ROW)
x <= v(1); -- invalido (ROW x STD_LOGIC_VECTOR)
x <= w(2); -- invalido (w deve ser indexado como 2D)
x <= w(2, 2 DOWNTO 0); -- invalido (ROW x STD_LOGIC)
v(0) <= w(2, 2 DOWNTO 0); --invalido(STD_LOGIC_VECTOR x
   STD_LOGIC)
v(0) <= w(2); -- invalido(w deve ser indexado como 2D)
y(1) <= v(3); -- invalido(ROW x STD_LOGIC_VECTOR)
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0); -- valido (mesmo tipo,
   mesmo tamanho)
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0); -- valido(mesmo tipo,
   mesmo tamanho)
w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0); -- invalido(tipos
   diferentes)
```

- Especificar portas (pinos) como array de vetores
  - Não é permitido TYPE em ENTITY
  - Solução: usar PACKAGE
  - Ex.:

```
------- Package: -------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------
PACKAGE my_data_types IS
   TYPE vector_array IS ARRAY (NATURAL RANGE <>) OF
      STD_LOGIC_VECTOR(7 DOWNTO 0);
END my_data_types;
-----------------------------------------

------- Main code: ------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_data_types.all;     -- user-defined package
---------------------------
ENTITY mux IS
   PORT (inp: IN VECTOR_ARRAY (0 TO 3);
   ... );
END mux;
   ... ;
-----------------------------------------
```

- Tipos SIGNED e UNSIGNED

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

  - Necessidade do pacote `std_logic_arith`
  - Similar ao STD_LOGIC_VECTOR
    - Usado para operações aritméticas
    - Não permitido para operações lógicas
  - Legalidade de operações :

```
SIGNAL a: SIGNED (7 DOWNTO 0);
SIGNAL b: SIGNED (7 DOWNTO 0);
SIGNAL x: SIGNED (7 DOWNTO 0);
v <= a + b; -- valida
w <= a AND b; -- invalida
```

- Operações aritméticas com STD_LOGIC_VECTOR
  - Pacotes IEEE: `std_logic_signed` e `std_logic_unsigned`
  - Exemplo:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
SIGNAL a: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: STD_LOGIC_VECTOR (7 DOWNTO 0);
v <= a + b; -- valida
w <= a AND b; -- valida
```

- Exercícios: Implementar o circuito de um somador usando dados de 4bits com: (a) sinal e (b) STD_LOGIC_VECTOR

- Conversão de dados
  - Conversão de tipos relacionados
    - std_logic_1164
  - Exemplo:

```
TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
y <= 2*x + 5; -- invalido
y <= long(2*x + 5); -- OK, resultado
  convertido em long
```

- Funções de conversão presentes no **`std_logic_arith`**
  - **`conv_integer(p)`**: converte o parâmetro p do tipo INTEGER, UNSIGNED, SIGNED, ou STD_ULOGIC para um valor INTEGER.
  - **`conv_unsigned(p, b)`**: converte o parâmetro p do tipo INTEGER, UNSIGNED, SIGNED, ou STD_ULOGIC para um UNSIGNED com b bits.
  - **`conv_signed(p, b)`**: converte o parâmetro p do tipo INTEGER, UNSIGNED, SIGNED, ou STD_ULOGIC para um SIGNED com b bits.
  - **`conv_std_logic_vector(p, b)`**: converte o parâmetro p do tipo INTEGER, UNSIGNED, SIGNED, ou STD_LOGIC para STD_LOGIC_VECTOR com b bits.

- Exemplos de conversão de dados:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
SIGNAL a: UNSIGNED (7 DOWNTO 0);
SIGNAL b: UNSIGNED (7 DOWNTO 0);
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8); --a+b eh
  convertido para um vetor de 8 bits tipo
  STD_LOGIC_VECTOR e depois atribuido a y.
```

- Mais exemplos de atribuições:

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

```
-- Atribuicoes PERMITIDAS de escalares: mesmo tipo
-- e indexacao correta
x(2) <= a;
y(0) <= x(0);
z(7) <= x(5);
b <= v(3);
w1(0,0) <= x(3);
w1(2,5) <= y(7);
w2(0)(0) <= x(2);
w2(2)(5) <= y(7);
w1(2,5) <= w2(3)(7);
-- Atribuicoes ILEGAIS de escalres
b <= a; -- tipos diferentes (BIT x STD_LOGIC)
w1(0)(2) <= x(2); -- o indice de w1 deve ser 2D
w2(2,0) <= a; -- o indice de w2 deve ser 1Dx1D
```

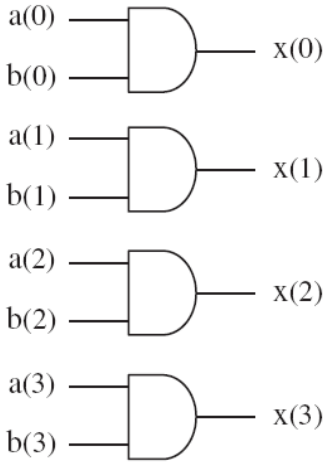
```
-- atribuicoes PERMITIDAS de vetores
x <= "11111110";
y <= ('1','1','1','1','1','1','0','Z');
z <= y;
y(2 DOWNTO 0) <= z(6 DOWNTO 4);
w2(0)(7 DOWNTO 0) <= "11110000";
w3(2) <= y;
z <= w3(1);
z(5 DOWNTO 0) <= w3(1)(2 TO 7);
w3(1) <= "00000000";
w1 <= ("11111100", "11110000" ,"10110000",
  "10011100");
```

```
--atribuicoes de matrizes ilegais
x <= y; -- tipos diferentes
w1(0, 7 DOWNTO 0) <="11111111"; -- w1 eh 2D array
w2 <= (OTHERS => 'Z'); -- w2 eh 1Dx1D array
w2(0, 7 DOWNTO 0) <= "11110000"; -- indice deve
  ser 1Dx1D
-- Exemplo de inicializacao de array
FOR i IN 0 TO 3 LOOP
  FOR j IN 7 DOWNTO 0 LOOP
      x(j) <= '0';
      y(j) <= '0';
      z(j) <= '0';
      w1(i,j) <= '0';
      w2(i)(j) <= '0';
      w3(i)(j) <= '0';
  END LOOP;
END LOOP;
```

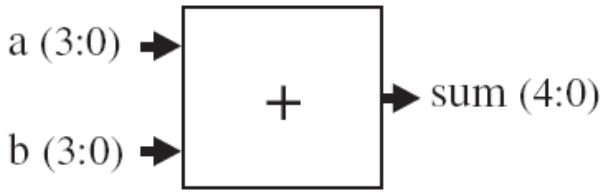
- Mais exemplos:

```
-----------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT;
         x: OUT BIT);
END and2;
-----------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------
```


```
-----------------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT_VECTOR (0 TO 3);
         x: OUT BIT_VECTOR (0 TO 3));
END and2;
-----------------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------------
```



UFSJ




```
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  USE ieee.std_logic_arith.all;
5  ---------------------------------------
6  ENTITY adder2 IS
7     PORT ( a, b : IN SIGNED (3 DOWNTO 0);
8            sum : OUT INTEGER RANGE -16 TO 15);
9  END adder2;
10 ---------------------------------------
11 ARCHITECTURE adder2 OF adder2 IS
12 BEGIN
13    sum <= CONV_INTEGER(a + b);
14 END adder2;
15 ---------------------------------------
```