UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

SISTEMAS DIGITAIS

Linguagem de Descrição de Hardware
VHDL

Prof. Fernanda Gusmão de Lima Kastensmidt
fglima@inf.ufrgs.br

Linguagem de Descrição de Hardware

- Hardware Description Language (HDL) = "Programming"-language para modelar hardware digital
- **VHDL = VHSIC Hardware Description Language**
- **VHSIC = Very High Speed Integrated Circuit Hardware description, simulation, and synthesis**

Descreve o hardware em diferentes níveis:

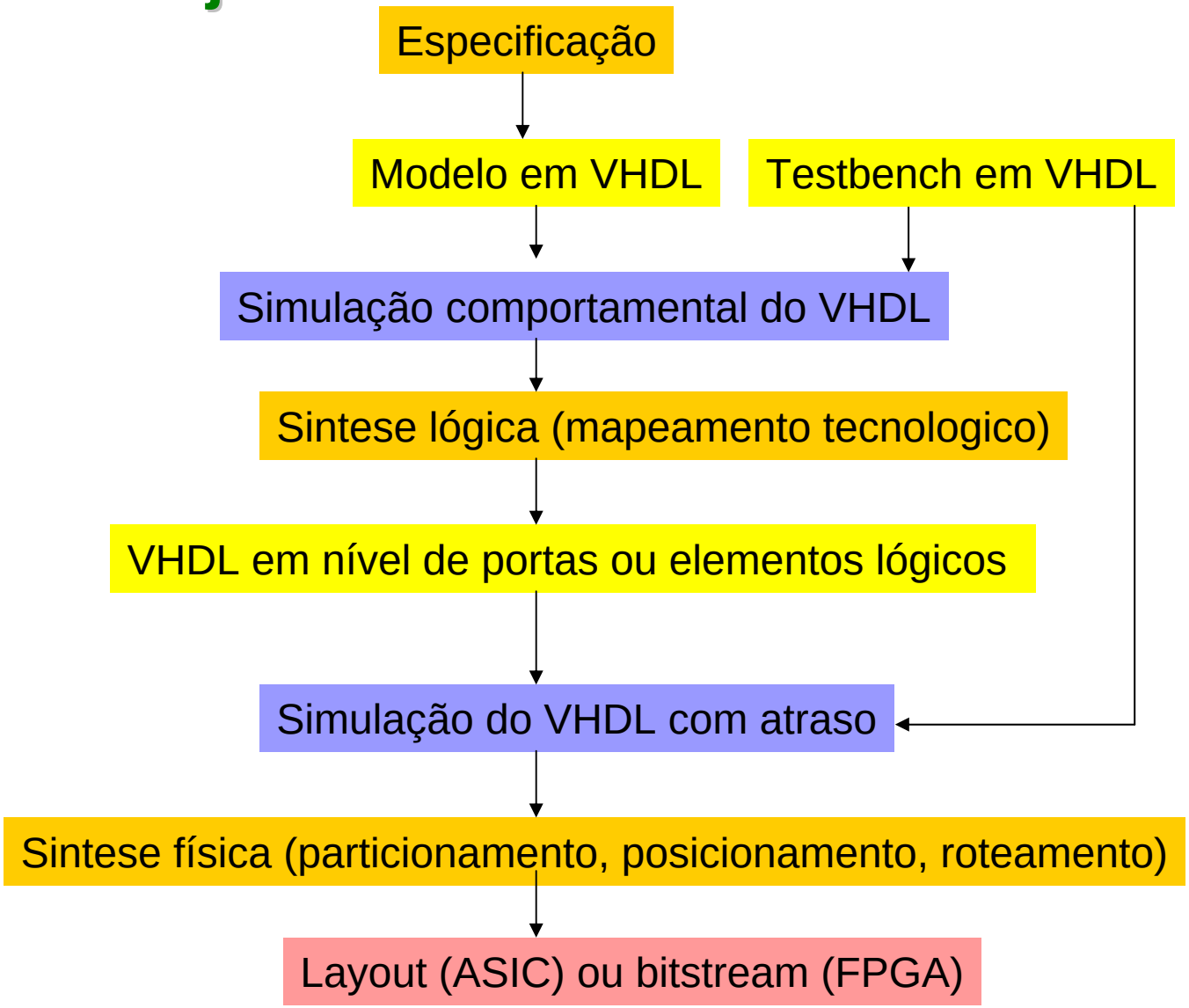
- **comportamental, equação lógica, estrutural**
- **Metodologia de projeto Top-down**
- **Independe de tecnologia**

VHDL

- VHDL foi desenvolvido primeiramente pelo Departamento de Defesa Americano: the American Department of Defense (DoD).
- Em 1987, VHDL foi padronizado pelo Instituto Americano de Engenharia Eletro e Eletrônica: American Institute of Electrical and Electronics Engineers (IEEE) em 1993.

  International Standards
  - **IEEE Std 1076-1987**
  - **IEEE Std 1076-1993**
- VHDL para circuitos análogicos e mistos é chamado de VHDL-AMS (**a**nalogue- **m** ixed- **s** ignal) e é um subconjunto do VHDL.
  - **Analogue- and mixed-signal extension: VHDL-AMS**
  - **IEEE Std 1076.1-1999**

Fluxo de Projeto



Sintaxe da linguagem VHDL

Geral

- Não é sensivel a maiusculas e minusculas
- Comentario de linha inteira usando dois traços (--) no começo da linha
- Comandos são terminados por ponto e virgula (;)
- Delimitador de elementos de uma lista: virgula (,)
- Assinalamento de sinais com flecha (<=)
- Nomes devem começar com letras

Elementos Estruturais

- **Entity :** entidade que define a interface
- **Architecture :** implementação da arquitetura que pode ser comportamental ou estrutural
- **Configuration :** usado para configurar a simulação ou sintese, fazendo a conexão entre a arquitetura e a entidade.
- **Process :** controlado por um evento, os processos são todos concorrentes.
- **Package :** projeto modular, define os tipos de dados a e as constantes a serem usadas no projeto.
- **Library :** usada para funções especificas de aritmetica, lógica e outras.

Estrutura do VHDL

```
library library_name;                    6
use library_name.tipo;

entity name is
port(pino1 : in tipo;                    2
     pino2 : in tipo;                    1
     pino3 : out tipo);
name end;

architecture name_tipo of name is
<declaração de componentes>              5
<declaração de sinais>                   3
begin

process(sinais)                          7
begin

end process;

process(clk)                             8
begin

end process;

<outros comandos>                        4
<instanciação de componentes;

end;
```

Bibliotecas IEEE para VHDL

Library IEEE

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_textio.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_bit.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.math_real.all;
use IEEE.math_complex.all;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
```

- The package std_logic_1164 provides enhanced signal types
- Types defined include:
- std_ulogic
- std_ulogic_vector
- std_logic
- std_logic_vector

library **IEEE;**
use **IEEE.std_logic_textio.**all**;**

- The package textio provides user input/output.

- Types defined include:
- line
- text
- side
- width
- Functions defined include: readline, read, writeline, write, endline

library **IEEE;**
use **IEEE.std_logic_arith.**all**;**

- std_logic_arith_syn.vhd defines types signed and unsigned and has arithmetic functions that operate on signal types signed and unsigned and std_logic_vector and std_ulogic_vector, but adding A to B of std_logic_vector type, needs unsigned(A) + unsigned(B).

- std_logic_arith_ex.vhd has arithmetic functions that operate on signal types std_logic_vector and std_ulogic_vector

```
library IEEE;
use IEEE.numeric_bit.all;
 use IEEE.numeric_std.all
```

- The package numeric_bit provides numerical computation Types defined include: unsigned signed arrays of type bit for signals

- The package numeric_std provides numerical computation Types defined include: unsigned signed arrays of type std_logic for signals

```
library IEEE;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all
```

- The package std_logic_signed provides signed numerical computation on type std_logic_vector

- The package std_logic_unsigned provides unsigned numerical computation on type std_logic_vector

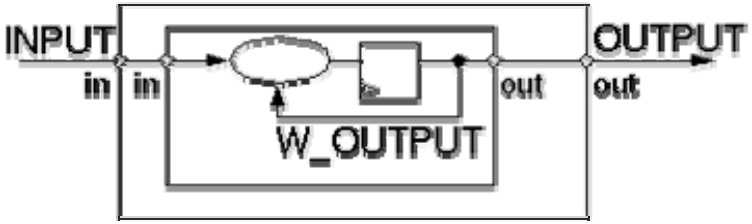
```
library IEEE;
 use IEEE.math_real.all;
use IEEE.math_complex.all;
```

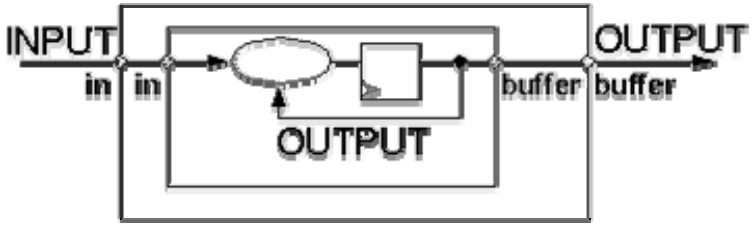
- The package math_real provides numerical computation on type real

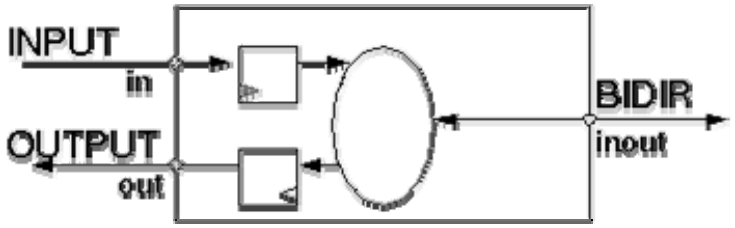
- The package math_complex provides numerical computation Types defined include: complex, complex_vector, complex_polar

Modos dos pinos da Entity e de Sinais Internos

Modos dos pinos da entidade







in:

Valor do sinal do pino é read-only

out:

Valor do sinal do pino é write-only

buffer:

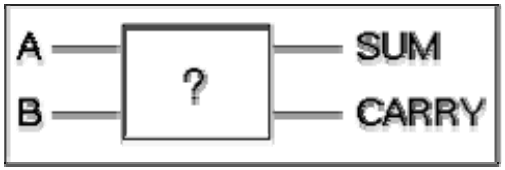
O sinal pode ser tambem lido da saída

inout:

Pino bidirecional

Exemplos de entidades

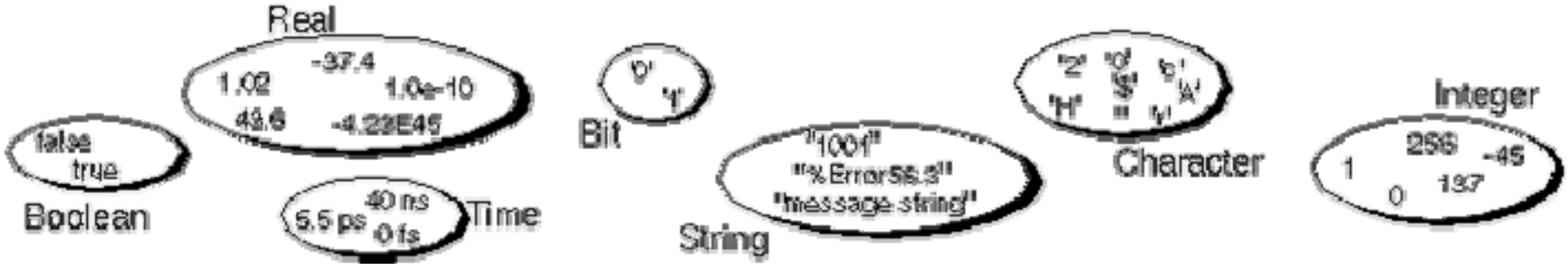
```
entity HALFADDER is
   port(
      A, B:              in   std_logic;
      SUM, CARRY: out std_logic);
end HALFADDER;
-- VHDL'93: end entity HALFADDER
```




```
entity ADDER is
    port(
      A, B:              in      std_logic_vector(3 downto 0;
      SUM:               out    std_logic_vector(3 downto 0);
      CARRY:             out    std_logic );
end ADDER;
```

Tipos de dados de pinos e sinais e declaração de sinais

Standard Data Types




```
package STANDARD is
   type BOOLEAN is (FALSE,TRUE);
   type BIT is (`0`,`1`);
   type CHARACTER is (-- ascii set);
   type INTEGER is range
                        -- implementation defined
   type REAL is range
                        --  implementation defined
   -- BIT_VECTOR, STRING, TIME
end STANDARD;
```

Tipo de dado: TIME

- **Uso:**
  - **testbenches**
  - **Atraso de portas**
- **Unidades de tempo disponíveis:**
  - **fs,**
  - **ps,**
  - **ns,**
  - **us,**
  - **ms,**
  - **sec,**
  - **min,**
  - **hr**

```
architecture EXAMPLE of TIME_TYPE is
  signal CLK : bit := `0';
  constant PERIOD : time := 50 ns;

begin
  process
  begin
    wait for 50 ns;
    . . .
    wait for  PERIOD ;
    . . .
    wait for 5 *  PERIOD ;
    . . .
    wait for  PERIOD  * 5.5;
  end process;

. . .

  -- concurrent signal assignment
  CLK <= not CLK after 0.025 us;
  -- or with constant time
  -- CLK <= not CLK after PERIOD/2;
end EXAMPLE;
```

Tipo de dados: Bit

- **O tipo de dado deve estar especificado no:**
  - **Port**
  - **Declaração dos sinais**
- **Os tipos devem ser compativeis no assinalamento.**

```
entity FULLADDER is
   port(A, B, CARRY_IN: in   bit;
        SUM, CARRY:      out bit);
end FULLADDER;

architecture MIX of FULLADDER is

begin
   SUM <= A xor B xor CARRY_IN;
   CARRY <= (A and B) or (A and CARRY_IN)
or (B and CARRY_IN);

end MIX;
```

Problemas com o tipo: BIT

type BIT is ('0', '1')

- **Valores `0` e `1`, apenas**
  - **Valor padrão `0`**
- **Para simulação e sintese é preciso outros valores como:**
  - **Não inicializado**
  - **Alta impedancia**
  - **Não definido**
  - **Não interessa (X)**
  - **Diferentes correntes**

Tipo Binário com multiplos valores: STD_LOGIC

- **IEEE-standard**
- **9 valores padronizados pela IEEE**
- **standard IEEE 1164 (STD_LOGIC_1164)**

IEEE Standard Logic Type

```
type STD_ULOGIC is (
                `U`,   -- uninitialized
                `X`,   -- strong 0 or 1 (= unknown)
                `0`,   -- strong 0
                `1`,   -- strong 1
                `Z`,   -- high impedance
                `W`,   -- weak 0 or 1 (= unknown)
                `L`,   -- weak 0
                `H`,   -- weak 1
                 `-`,   -- don`t care);
```

Exemplo de declaração de sinais

signal <nome> **:** tipo **:=** inicialização;

```
Library ieee;
Use ieee.std_logic_1164.all;

entity FULLADDER is
   port(A, B, CARRY_IN: in   std_logic;
        SUM, CARRY:     out std_logic);
end FULLADDER;

architecture MIX of FULLADDER is
signal x : std_logic;
begin
   x <= A xor B;
   SUM <= x xor CARRY_IN;
   CARRY <= (A and B) or (A and CARRY_IN)
or (B and CARRY_IN);

end MIX;
```

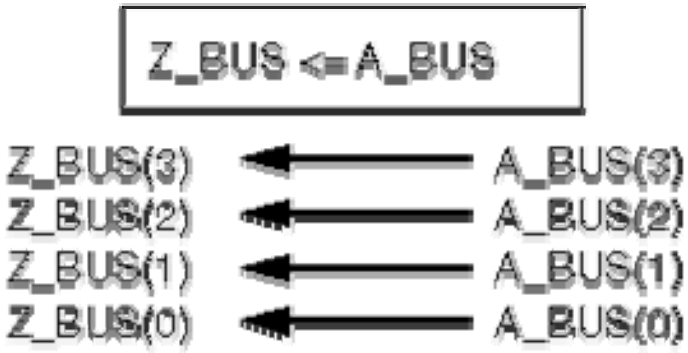
Tipo de dados: Std_logic

- **O tipo de dado deve estar especificado no:**
  - **port**
  - **Declaração dos sinais**
- **Os tipos devem ser compativeis no assinalamento.**

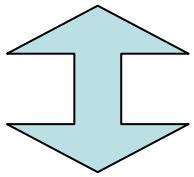
```
Library ieee;
Use ieee.std_logic_1164.all;

entity FULLADDER is
   port(A, B, CARRY_IN: in   std_logic;
        SUM, CARRY:      out std_logic);
end FULLADDER;

architecture MIX of FULLADDER is

begin
   SUM <= A xor B xor CARRY_IN;
   CARRY <= (A and B) or (A and CARRY_IN)
or (B and CARRY_IN);

end MIX;
```

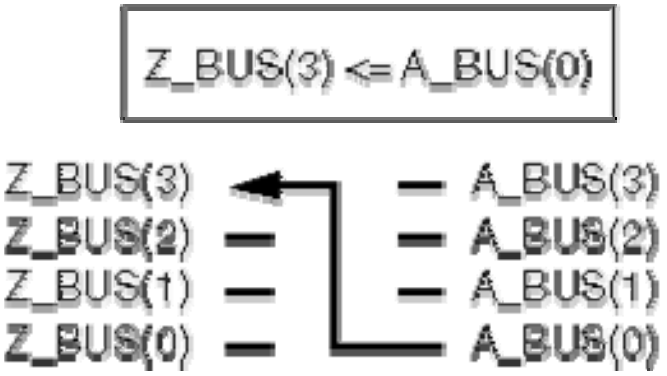
Definição de Array

- **Coleção de sinais do mesmo tipo**
- **Array pre-definido**
  - **bit_vector (array of bit)**
  - **string (array of character)**
- **A definição do tamanho do array é dada na declaração do sinal ou do pino.**




```
signal A_BUS, Z_BUS : bit_vector (3 downto 0);
```

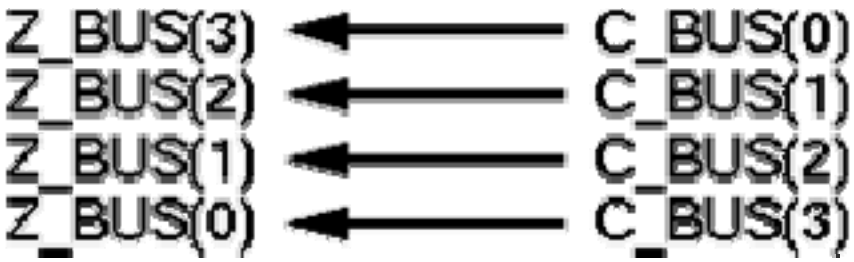




signal A_BUS, Z_BUS : std_logic_vector(3 downto 0);

Assignments with Array Types

```
architecture EXAMPLE of ARRAYS is
   signal Z_BUS : std_logic_vector (3 downto 0);
   signal C_BUS : std_logic_vector (0 to 3);
begin
   Z_BUS <= C_BUS;
end EXAMPLE;
```



Cuidado ao usar (X downto 0) ou (0 to X)

Tipo Integer, Signed e Unsigned

- Os numeros inteiros podem variar de $-2^{31} + 1$ to $+2^{31} - 1$
- (-2147483647 to +2147483647).
- A representação padrão é em decimal.
- Para utilizar uma outra representação é preciso mostrar explicitamente:

binary	2#...#
Octal	8#...#
Hexadecimal	16#...#

Tipos de assinalamento para ‘std_logic'

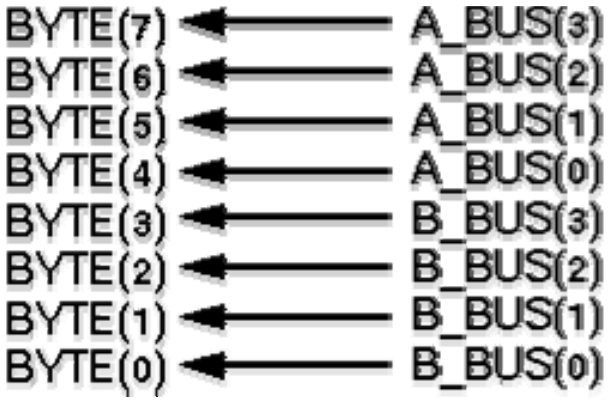
```
architecture EXAMPLE of ASSIGNMENT is

   signal Z_BUS :    std_logic_vector (3 downto 0);
   signal BIG_BUS : std_logic_vector (15 downto 0);

begin

   -- legal assignments:
   Z_BUS(3) <= `1`;
   Z_BUS    <= ``1100``;

   Z_BUS    <=  b`` 1100 ``;
   Z_BUS    <=  x`` c ``;
   Z_BUS    <=  X`` C ``;
   BIG_BUS  <=  B` `0000 _ 0001_0010_0011 ``;

end EXAMPLE;
```

Simples para um digito

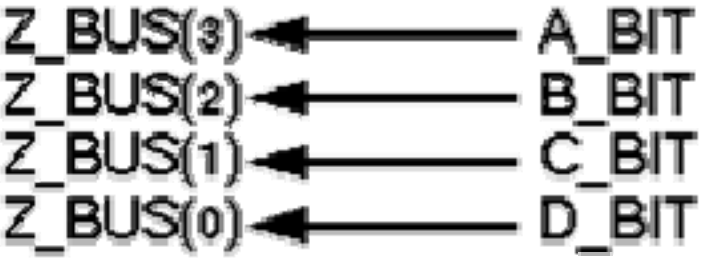
Aspas duplas para multiplos digitos

Concatenação (&)

```
architecture EXAMPLE_1 of CONCATENATION is
   signal BYTE              : std_logic_vector (7 downto 0);
   signal A_BUS, B_BUS : std_logic_vector (3 downto 0);
begin

   BYTE   <= A_BUS & B_BUS;

end EXAMPLE;
```




```
architecture EXAMPLE_2 of CONCATENATION is
   signal Z_BUS : std_logic_vector (3 downto 0);
   signal A_BIT, B_BIT, C_BIT, D_BIT : bit;
begin

   Z_BUS <= A_BIT & B_BIT & C_BIT & D_BIT;

end EXAMPLE;
```



Lado direito do assinalamento

Agregação

```
architecture EXAMPLE of AGGREGATES is
   signal BYTE :  std_logic_vector (7 downto 0);
   signal Z_BUS : std_logic_vector (3 downto 0);
   signal A_BIT, B_BIT, C_BIT, D_BIT : std_logic;
begin
   Z_BUS <= ( A_BIT, B_BIT, C_BIT, D_BIT ) ;
    ( A_BIT, B_BIT, C_BIT, D_BIT ) <= std_logic'(``1011``);
    ( A_BIT, B_BIT, C_BIT, D_BIT ) <= BYTE(3 downto 0);

   BYTE <= ( 7 => `1`, 5 downto 1 => `1`, 6 => B_BIT, others => `0`) ;
end EXAMPLE;
```

Utilizando Porções de Arrays

```
architecture EXAMPLE of SLICES is
   signal BYTE : std_logic_vector (7 downto 0);
   signal A_BUS, Z_BUS : std_logic_vector (3 downto 0);
   signal A_BIT : bit;
begin
   BYTE (5 downto 2) <= A_BUS;
   --BYTE (5 downto 0) <= A_BUS;              -- wrong

   Z_BUS (1 downto 0) <= `0` & A_BIT;
   Z_BUS                    <= BYTE (6 downto 3);
   --Z_BUS (0 to 1)       <= `0` & B_BIT;      -- wrong

   A_BIT   <= A_BUS (0);
end EXAMPLE;
```

Tipos Enumerados

```
architecture EXAMPLE of ENUMERATION is

   type T_STATE is (RESET, START, EXECUTE, FINISH);

  signal CURRENT_STATE, NEXT_STATE : T_STATE ;
  signal TWO_BIT_VEC : std_logic_vector(1 downto 0);

begin

  -- valid signal assignments
  NEXT_STATE    <= CURRENT_STATE;
  CURRENT_STATE <= RESET;

  -- invalid signal assignments
  CURRENT_STATE <= "00";
  CURRENT_STATE <= TWO_BIT_VEC;

end EXAMPLE;
```

Multidimensional Arrays

```
architecture EXAMPLE of ARRAY is

    type  INTEGER_VECTOR  is
        array (1 to 8) of integer;

   -- 1 --
    type  MATRIX_A  is
        array(1 to 3)  of  INTEGER_VECTOR ;
   -- 2 --
   type  MATRIX_B  is
        array(1 to 4, 1 to 8)  of  integer ;

    signal MATRIX_3x8 : MATRIX_A;
    signal MATRIX_4x8 : MATIRX_B;

begin

    MATRIX_3x8(3)(5)  <= 10;  --array of array

    MATRIX_4x8(4, 5)   <= 17;  -- 2 dim array

end EXAMPLE;
```

2 possibilities
- **array of array**
- **multidimensional array**

- **Different referencing**

- **Barely supported by synthesis tools**

```
architecture EXAMPLE of AGGREGATE is

    type INTEGER_VECTOR is
        array (1 to 8) of integer;
    type MATRIX_A is
        array(1 to 3) of INTEGER_VECTOR;

    type MATRIX_B is
        array(1 to 4, 1 to 8) of integer;

   signal MATRIX3x8 : MATRIX_A;
   signal MATRIX4x8 : MATIRX_B;
   signal VEC0, VEC1,
          VEC2, VEC3  :  INTEGER_VECTOR;
begin

   MATRIX3x8 <= (VEC0, VEC1, VEC2);
   MATRIX4x8 <= (VEC0, VEC1, VEC2, VEC3);

   MATRIX3x8 <= (others => VEC3);
   MATRIX4x8 <= (others => VEC3);

   MATRIX3x8 <= (others => (others => 5));
   MATRIX4x8 <= (others => (others => 5));

end EXAMPLE;
```

Type Conversion

```
architecture EXAMPLE of CONVERSION is
     type MY_BYTE is array (7 downto 0) of std_logic;

    signal VECTOR:       std_logic_vector(7 downto 0);
    signal SOME_BITS:  bit_vector(7 downto 0);
    signal BYTE:            MY_BYTE;

begin

    SOME_BITS <= VECTOR;                          -- wrong
    SOME_BITS <= Convert_to_Bit( VECTOR ) ;

    BYTE <= VECTOR;                               -- wrong
    BYTE <= MY_BYTE( VECTOR ) ;

end EXAMPLE;
```

Conversions

Operator	Argument (arg)	Result
conv_integer(arg)	std_logic std_logic_vect or signed unsigned	integer
conv_unsigned(arg, size: integer) size: number of bits of the final result	std_logic signed unsigned integer	unsigned
conv_signed(arg, size: integer) size: number of bits of the final result	std_logic signed unsigned integer	signed
conv_std_logic_vector(arg, size: integer) size: number of bits of the final result	std_logic signed unsigned integer	std_logic_vector

Subtypes

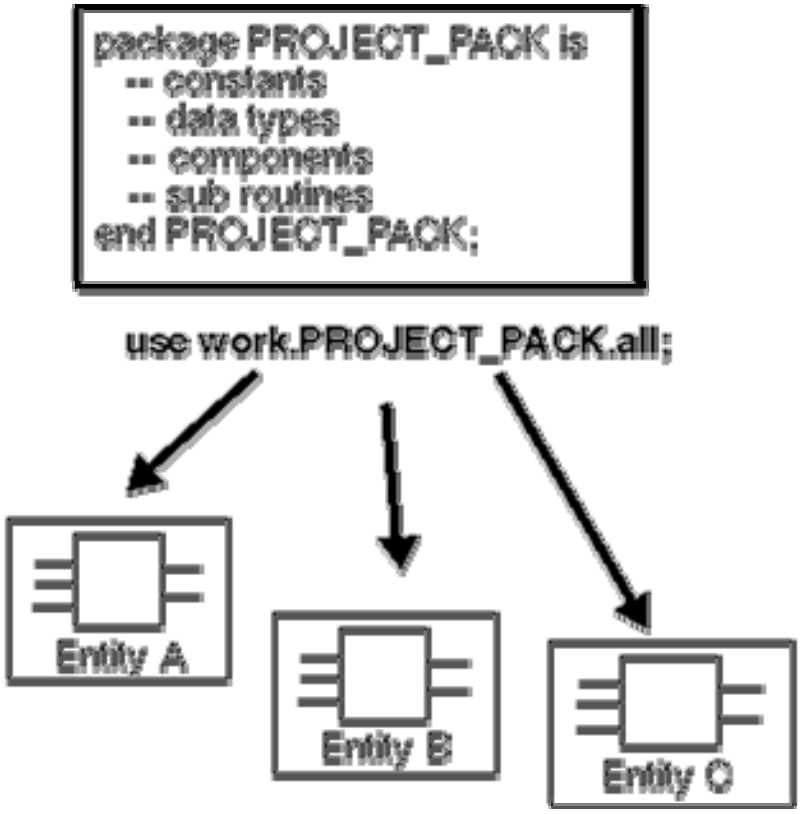
```
architecture EXAMPLE of SUBTYPES is
     type MY_WORD is array (15 downto 0) of std_logic;
     subtype SUB_WORD is std_logic_vector (15 downto 0);

    subtype MS_BYTE is integer range 15 downto 8;
    subtype LS_BYTE is integer range 7 downto 0;

    signal VECTOR:      std_logic_vector(15 downto 0);
    signal SOME_BITS: bit_vector(15 downto 0);
    signal WORD_1: MY_WORD;
    signal WORD_2: SUB_WORD;

begin
    SOME_BITS <= VECTOR;                        -- wrong
     SOME_BITS <= Convert_to_Bit(VECTOR);

    WORD_1 <= VECTOR;                           -- wrong
     WORD_1 <= MY_WORD(VECTOR);

     WORD_2 <= VECTOR;                          -- correct!

     WORD_2(LS_BYTE) <= "11110000";
end EXAMPLE;
```

Aliases

```
architecture EXAMPLE of ALIAS is
    signal DATA is bit_vector(9 downto 0);

    alias STARTBIT :   bit is DATA(9) ;
    alias MESSAGE:  bit_vector(6 downto 0) is DATA (8 downto 2);
    alias PARITY:       bit is DATA(1);
    alias STOPBIT:    bit is DATA(0);
    alias REVERSE:  bit_vector(1 to 10) is DATA;

    function calc_parity(data: bit_vector) return bit is
    . . .

begin
    STARTBIT       <= '0';
    MESSAGE       <= "1100011";
    PARITY          <= calc_parity(MESSAGE);
    REVERSE(10) <= '1';

end EXAMPLE;
```

Package

- Coleção de definições, tipos de dados e sub programas ou funções
- Referencia feita pelo grupo de projeto
- Vantagem:
  - Qualquer mudança é vista por todo o time rapidamente.
  - Mesmo tipo de dado e uso ("downto vs. to")
  - Funções para todosl

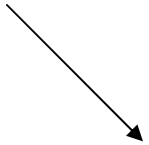


Declaração de sinais e constantes

architecture name_type **of** name_entity **is**

signal A, B, C : std_logic := ‘0’;
signal D : std_logic_vector(3 downto 0) := “0000”;
signal F: integer range 0 to 4;
constant G : integer := 9;

BEGIN



inicialização

Operadores e Comandos

Operadores

logical	not						
	and	or	nand	nor	xor	xnor	
relational		=	/=	<	<=	>=	>
shift	sll	srl	sla	sra	rol	ror	
arithmetic		+	-				
	*	/	mod	rem			
	**	abs					

Operadores e seus Argumentos

arg1 operator arg2 → result, operator: + , -	
arg1, arg2	**result**
both std_logic_vector std_logic_vector, std_logic (or reverse) std_logic_vector, integer (or reverse)	std_logic_vector
both unsigned unsigned, std_logic (or reverse) unsigned, integer (or reverse)	unsigned or std_logic_vector
both signed signed, std_logic (or reverse) signed, integer (or reverse) signed, unsigned (or reverse)	signed or std_logic_vector

Table 1: *Operators +,-*

Operadores e seus Argumentos

arg1 operator arg2 → result, operator: *	
arg1, arg2	**result**
both std_logic_vector	std_logic_vector
both unsigned	unsigned o std_logic_vector
both signed signed, unsigned (or reverse)	signed o std_logic_vector

Table 2: *Operators* *

Operadores e seus Argumentos

arg1 operator arg2 → result, operator: <,<=,>,>=,=,/=	
arg1, arg2	**result**
both std_logic_vector std_logic_vector, integer (or reverse)	boolean
both unsigned both signed unsigned, integer (or reverse) signed, integer (or reverse) signed, unsigned (or reverse)	boolean

Table 3: *Comparison Operators*

Operadores Lógicos

Prioridade

- **not (maior prioridade)**
- **and, or, nand, nor, xor, xnor (mesma prioridade)**

Pre-definidos para

- **bit, bit_vector**
- **boolean**

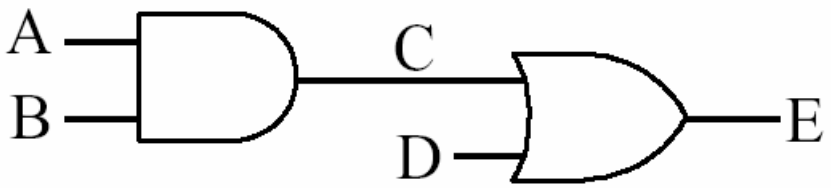
- **Para std_logic precisa usar a library ieee.std_logic_1164.all;**

```
entity LOGIC_OP is
   port (A, B, C, D : in   bit;
         Z1:          out bit;
         EQUAL :      out boolean);
end LOGIC_OP;

architecture EXAMPLE of LOGIC_OP is
begin

   Z1 <= A  and (B or (not C xor D)));

   EQUAL <= A xor B;                -- wrong

end EXAMPLE;
```

Operados Lógicos em Arrays

```
architecture EXAMPLE of LOGICAL_OP is
   signal A_BUS, B_BUS : std_logic_vector (3 downto 0);
   signal Z_BUS :        std_logic_vector (4 to 7);
begin
   Z_BUS <= A_BUS and B_BUS;
end EXAMPLE;
```

Comandos concorrentes



Comandos concorrentes:

C <= A **and** B **after** 5 ns;

E <= C **or** D **after** 5 ns;

Se um atrado não é especificado, um “delta” atraso é assumido

C <= A **and** B;

E <= C **or** D;

A ordem do comando concorrente não é importante

E <= C **or** D;

C <= A **and** B;

Esse comando executa repedidamente

CLK <= **not** CLK **after** 10 ns;

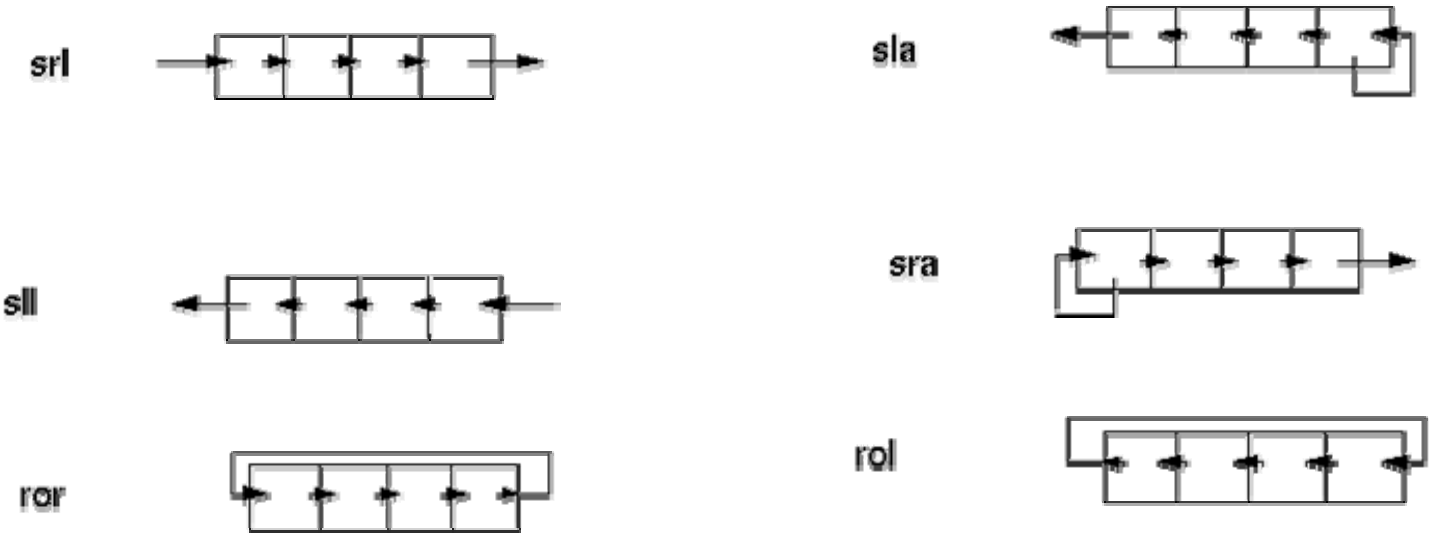
Este comando causa um erro na simulação

CLK <= **not** CLK;

Operadores de Deslocamento: só funcionam com bit_vector

```
signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);

'93

Z_BUS <= A_BUS sll 2;
Z_BUS <= B_BUS sra 1;
Z_BUS <= A_BUS ror 3;
```

← At the end, the first value of the type is used for filling up



Assinalamento condicional

Comandos concorrentes

```
TARGET <= VALUE_1 when CONDITION_1 else
          VALUE_2 when CONDITION_2 else
          . . .
          VALUE_n;
```

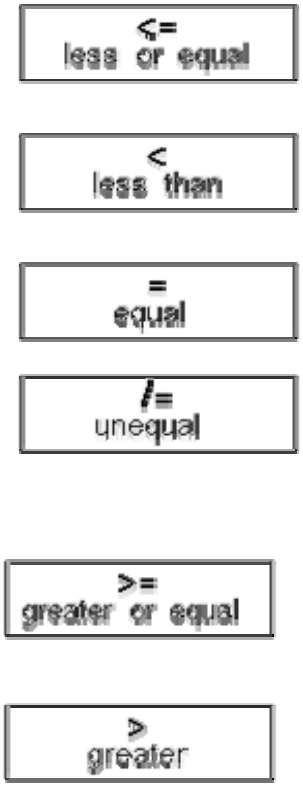
Assinalamento seletivo

Comandos concorrentes

```
with EXPRESSION select

   TARGET <= VALUE_1 when CHOICE_1,
             VALUE_2 when CHOICE_2 | CHOICE_3,
             VALUE_3 when CHOICE_4 to CHOICE_5,
             VALUE_n when others;
```

Operadores de relação

```
BEGIN

S_out <= “00” when A <=B else
         “11” when A=“1100” else
         “01”;

END;
```



Importante

- Comandos seletivos são puramente combinacionais e por isso, todas as possibilidades devem estar cobertas no comando.

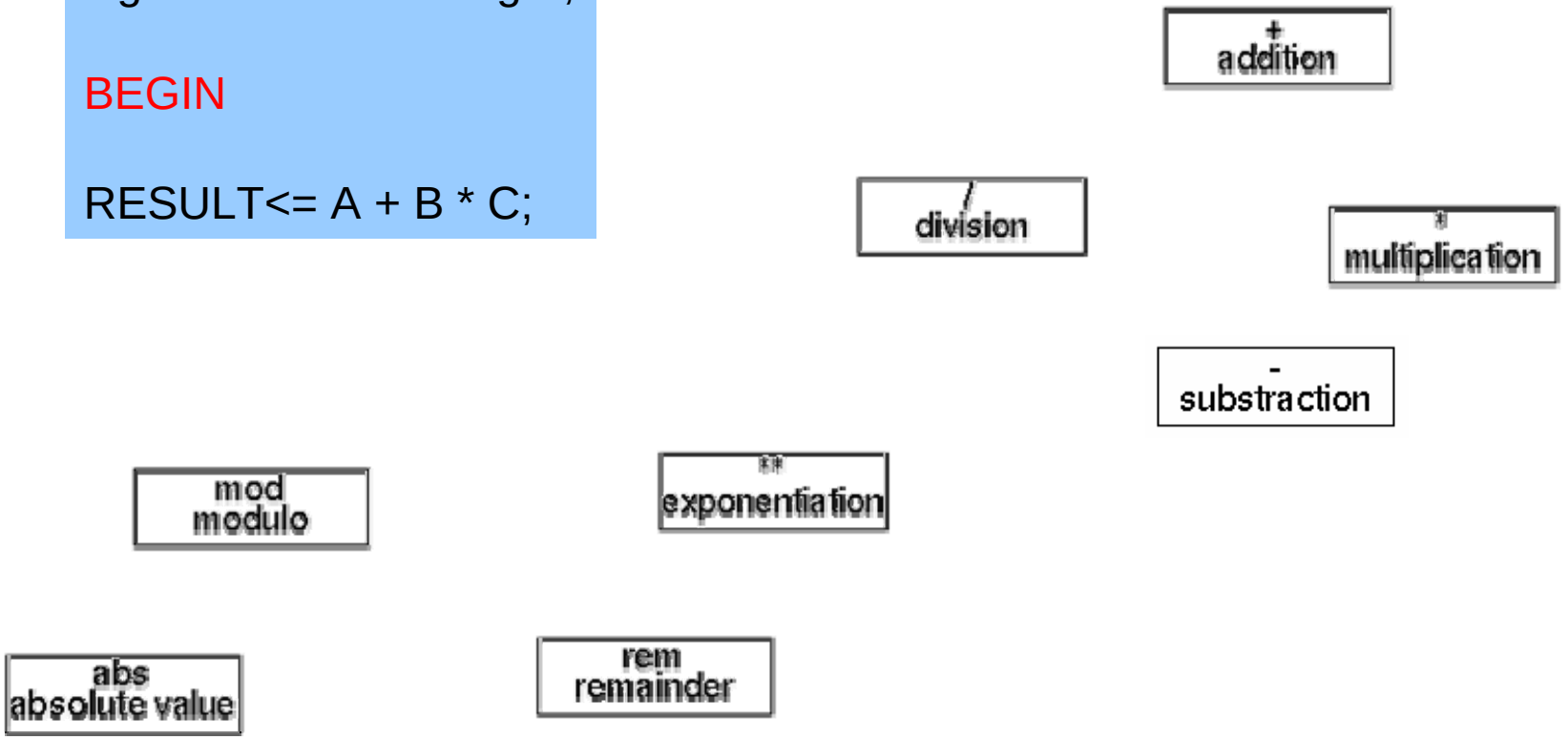
```
TARGET <= VALUE_1 when CONDITION_1 else
          VALUE_2 when CONDITION_2 else
          VALUE_n;
```

```
with EXPRESSION select

   TARGET <= VALUE_1 when CHOICE_1,
             VALUE_2 when CHOICE_2 | CHOICE_3,
             VALUE_3 when CHOICE_4 to CHOICE_5,
             VALUE_n when others;
```

- Caso isso não aconteça, surgirão latches e flip-flops indesejados na sintese o que irá comprometer o desempenho do circuito.

Operadores Aritméticos

```
signal A, B, C:    integer;
signal RESULT: integer;

BEGIN

RESULT<= A + B * C;
```



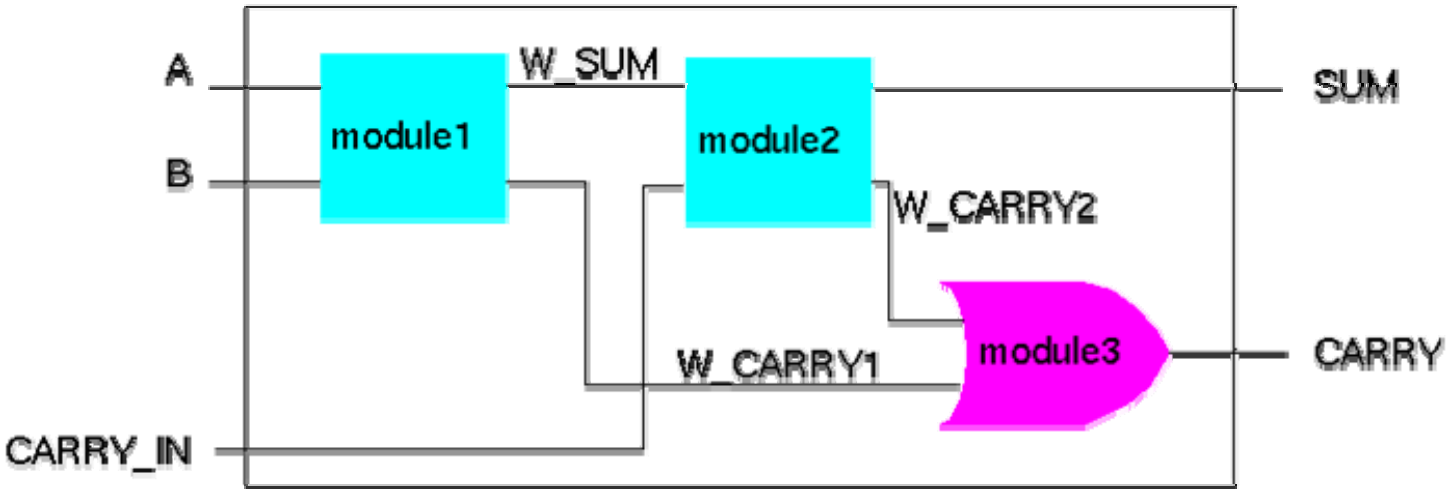
Uso da biblioteca ieee.std_logic_arith.all;

Declaração e Instanciação de Componentes

Declaração de componente

```
entity FULLADDER is
   port (A,B, CARRY_IN: in  std_logic;
         SUM, CARRY:    out  std_logic);
end FULLADDER;

architecture STRUCT of FULLADDER is
   signal W_SUM, W_CARRY1, W_CARRY2 :  std_logic;

    component  HALFADDER
      port (A, B :              in    std_logic;
            SUM, CARRY : out  std_logic);
    end component;

    component  ORGATE
      port (A, B : in    std_logic;
            RES : out  std_logic);
    end component;

begin
. . .
```

Modelo Hierarquico



Full adder: 2 halfadders + 1 OR-gate

Instanciação de Componente

```
architecture STRUCT of FULLADDER is
  component HALFADDER
    port (A, B :              in    std_logic;
          SUM, CARRY : out  std_logic);
  end component;
  component ORGATE
    port (A, B : in   std_logic;
          RES : out  std_logic);
  end component;

  signal W_SUM, W_CARRY1, W_CARRY2: std_logic;

begin

    MODULE1: HALFADDER
     port map( A, B, W_SUM, W_CARRY1 );

    MODULE2: HALFADDER
     port map ( W_SUM, CARRY_IN,
                SUM, W_CARRY2 );

    MODULE3: ORGATE
     port map ( W_CARRY2, W_CARRY1, CARRY );

end STRUCT;
```

Assinalamento dos sinais (ports) pela ordem dos pinos na interface do componente

Instanciação de Componente: associação de nomes dos ports

```
entity FULLADDER is
   port (A,B, CARRY_IN: in    std_logic;
         SUM, CARRY:    out  std_logic);
end FULLADDER;

architecture STRUCT of FULLADDER is

   component HALFADDER
      port (A, B :              in  std_logic;
            SUM, CARRY : out  std_logic);
   end component;
. . .
   signal W_SUM, W_CARRY1, W_CARRY2 :  std_logic;

begin

    MODULE1: HALFADDER
                port map (  A       => A,
                            SUM     => W_SUM,
                            B       => B,
                            CARRY   => W_CARRY1 );
   . . .
end STRUCT;
```

Assinalamento explicito

Uso de configuração

- **Seleciona a arquitetura para a entidade topo (top-level entity)**
- **Gera a hierarquia de simulação ou sintese**

```
entity FULLADDER is
. . .
end FULLADDER;

architecture STRUCT of FULLADDER is
. . .
end STRUCT;

architecture SCHEME of FULLADDER is
. . .
end SCHEME;

configuration CFG_FULLADDER of FULLADDER is
    for STRUCT   -- select architecture STRUCT
      -- use default configuration rules
    end for;
end configuration CFG_FULLADDER ;
```

Configuração: Examplo (1)

```
entity A is
   port(A, B:        in   std_logic;
          SUM, CARRY: out std_logic);
end A;

architecture RTL of A is
. . .



entity B is
   port(U,V: in    std_logic;
          X,Y: out std_logic);
end B;

architecture GATE of B is
. . .
```

```
entity FULLADDER is
   port(A, B, CARRY_IN: in    std_logic;
          SUM, CARRY:      out std_logic);
end FULLADDER;
architecture STRUCT of FULLADDER is

   component HALFADDER
      port(A, B:               in    std_logic;
             SUM, CARRY: out std_logic);
   . . .

   signal W_SUM, W_CARRY1, W_CARRY2: std_logic;

begin
   MODULE1: HALFADDER
      port map (A, B, W_SUM, W_CARRY1);

   MODULE2: HALFADDER
      port map(W_SUM, CARRY_IN, SUM, W_CARRY2);
   . . .

end STRUCT;
```

Configuração: Examplo (2)

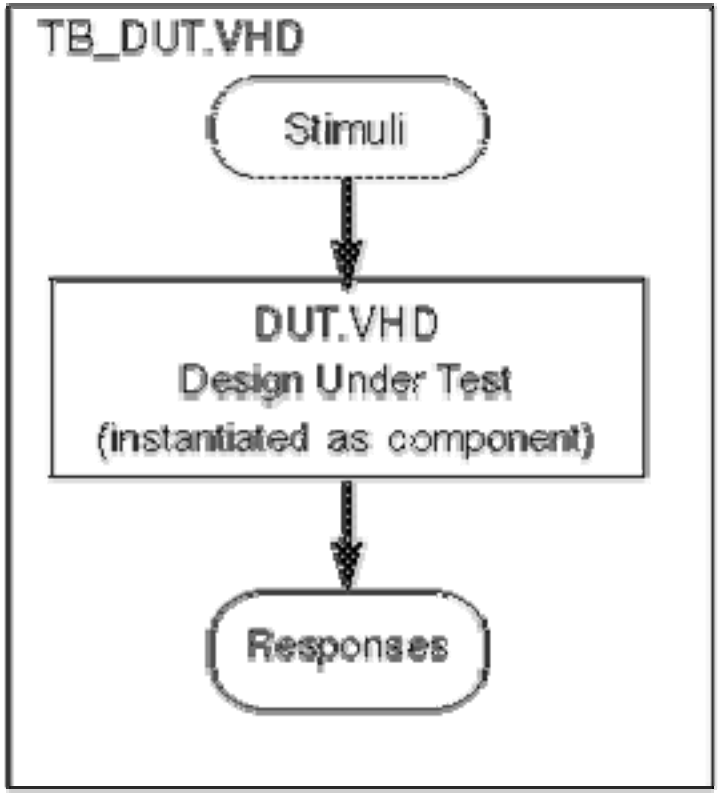
```
entity A is
   port (A, B:              in    std_logic;
         SUM, CARRY: out std_logic);
end A;

architecture RTL of A is
. . .



entity B is
   port (U,V: in    std_logic;
         X,Y: out std_logic);
end B;

architecture GATE of B is
. . .
```

```
configuration CFG_FULLADDER of FULLADDER is
   for STRUCT
      for MODULE2: HALFADDER
         use entity work.B(GATE);
         port map ( U => A,
                    V => B,
                    X => SUM,
                    Y => CARRY );
      end for;

      for others : HALFADDER
         use entity work.A(RTL);
      end for;
   end for;
end CFG_FULLADDER;
```

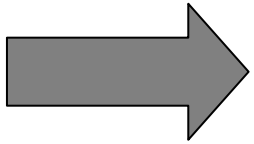
Testbench

Testbench

- VHDL code para a arquitetura topo
- Pode não ter sinais de interface
- O projeto DUT deve ser instanciado no testbench
- Comandos para o estimulo de sinais
- Testbenches simples: a analise da resposta por inspeção das formas de onda.
- Testbenches sofisticados podem ocupar mais de 50% do projeto.



Example of Testbench



Design Under Test (DUT)

```
entity ADDER IS
port (A,B :              in   bit;
CARRY,SUM : out bit);
end ADDER;

architecture RTL of ADDER is
begin
SUM <= A xor B;
CARRY <= A and B;
end RTL;
```

```
entity TB_ADDER IS
end TB_ADDER;

architecture TEST of TB_ADDER is
  component ADDER
port (A, B:              in   bit;
CARRY, SUM: out bit);
  end component;
  signal A_I, B_I, CARRY_I, SUM_I : bit;
begin
  DUT: ADDER port map (A_I, B_I, CARRY_I, SUM_I);

  STIMULUS: process
  begin
    A_I <= ´1´; B_I <= ´0´;
    wait for 10 ns;
    A_I <= ´1´; B_I <= ´1´;
    wait for 10 ns;
    -- and so on ...
  end process STIMULUS;
end TEST;
```