



Circuitos Digitais II - 6882

Paulo Roberto de Oliveira

Universidade Estadual de Maringá
Departamento de Informática

Bacharelado em Ciência da Computação

Aula de Hoje

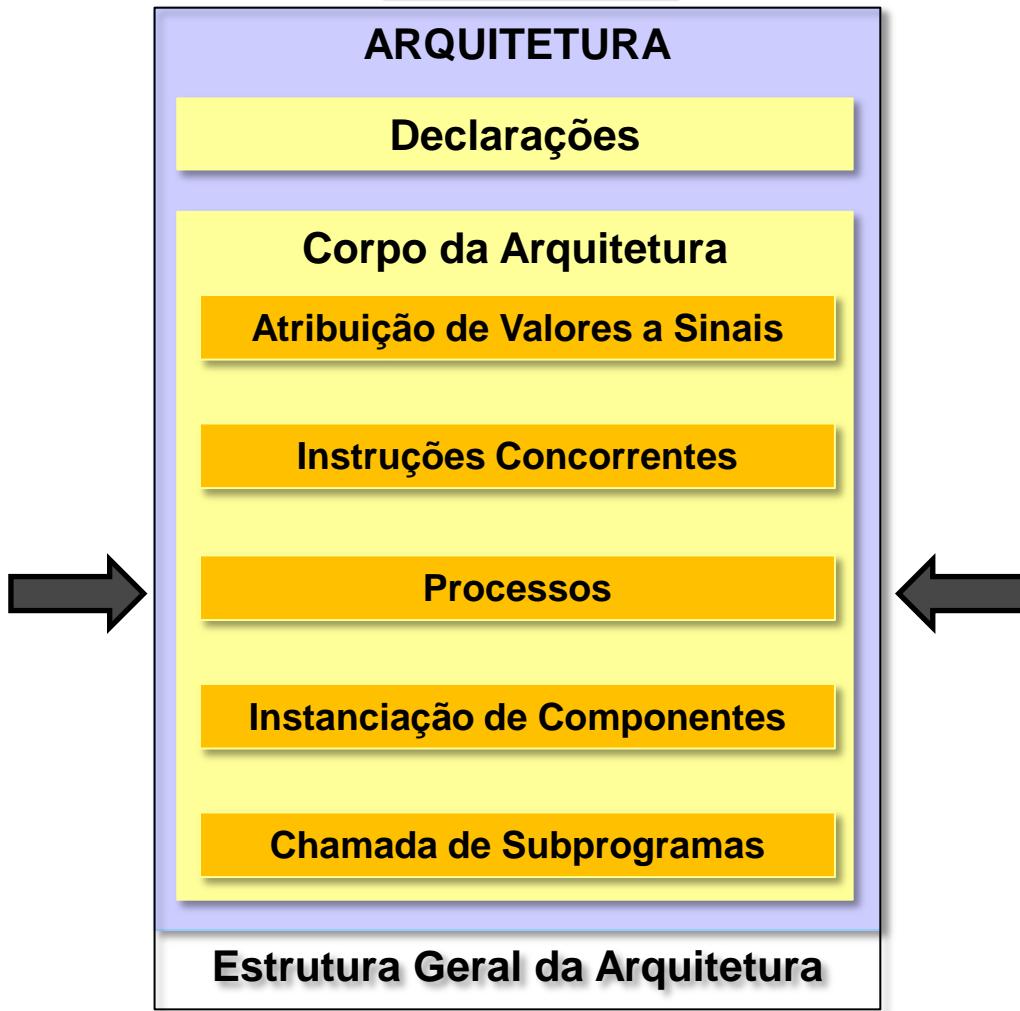
- **Revisão da aula anterior**
 - Processos
 - Pacotes
 - Atrasos
- **Subprogramação**
 - Funções
 - Procedimentos

Revisão

- **Processos**
- **Pacotes**
- **Atrasos**

VHDL - Processos

Processos



VHDL - Processos

Processos

- Um processo (process) define uma estrutura independente de processamento sequencial representativa do comportamento de uma parte do projeto.

PROCESS (lista de sensibilidades)

-- Parte declaratória

BEGIN

-- Corpo do processo

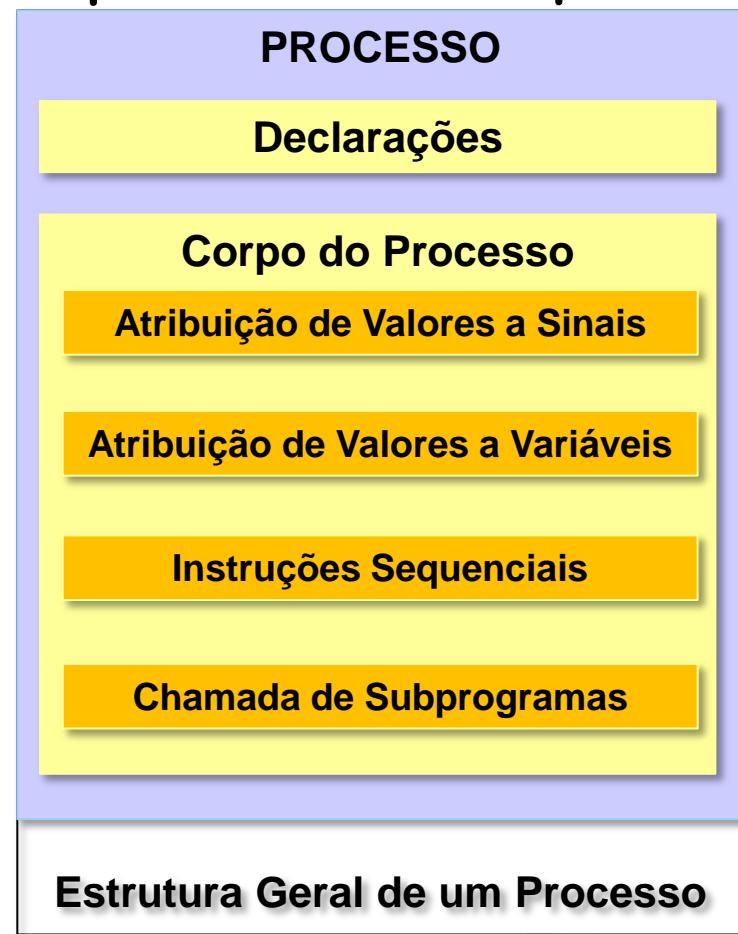
END PROCESS:

- A parte declaratória do processo pode conter:
 - Declaração de objetos (variáveis e constantes);
 - Declaração de tipos e subtipos de dados;
 - Declaração de subprogramas.

VHDL - Processos

Processos

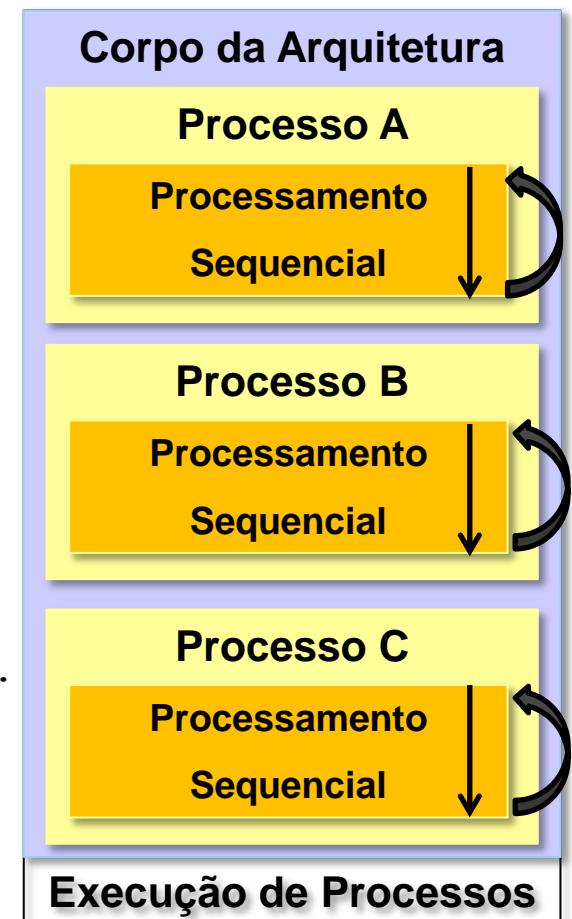
- O corpo do processo é uma estrutura de processamento sequencial que pode conter:
 - Atribuição de valores a sinais;
 - Atribuição de valores a variáveis;
 - Instruções sequenciais;
 - Chamada de subprogramas (funções e procedimentos).



VHDL - Processos

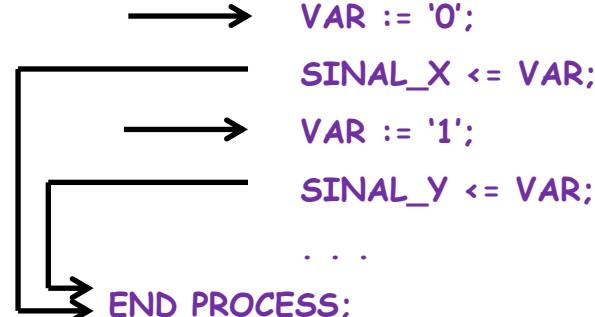
Execução de Processos

- Os processos são descritos dentro do corpo da arquitetura, podendo uma arquitetura conter diversos processos.
- Todos os processos descritos numa arquitetura são concorrentes entre si, sendo executados paralelamente com as restantes sentenças da arquitetura.
- Cada processo é executado repetidamente num ciclo infinito.
- É no entanto possível controlar a execução de cada processo, através de uma lista de sensibilidades ou através de instruções que permitem suspender a sua execução.
- As sentenças incluídas dentro de um processo são executadas de forma sequencial, pela ordem em que aparecem no programa.
- Todos os sinais tratados dentro de um processo são globais, sendo visíveis em toda a arquitetura, no entanto as variáveis tratadas (e declaradas) dentro de cada processo são apenas visíveis dentro dos respectivos processos.



VHDL - Processos

Atualização de sinais e variáveis

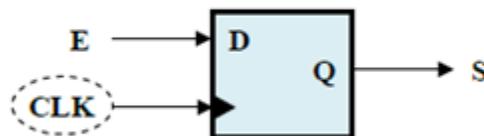
- Embora dentro de um processo se possam efetuar atribuições de valores a variáveis e sinais, a atualização destes dois tipos de objetos é processada de forma diferente:
 - As variáveis são atualizadas no mesmo instante Ex.: **PROCESS**
BEGIN
 
 VAR := '0';
 SIGNAL_X <= VAR;
 VAR := '1';
 SIGNAL_Y <= VAR;
 ...
END PROCESS;
 - Os sinais são atualizados apenas no final do processo, sendo os seus valores modificados uma única vez (por cada ciclo de execução).
 - Se num processo existirem Ex.: **PROCESS**
BEGIN
 X <= A OR B; -- atribuição ignorada
 Y <= X;
 X <= A AND C; -- atribuição que anula a precedente
 Z <= X; -- resultado: Z = Y = X = A AND C
END PROCESS;

VHDL - Processos

Listas de sensibilidade

- Um processo pode opcionalmente conter uma lista de sensibilidades, que consiste numa lista de sinais ao qual o processo é sensível.
- Esta lista de sensibilidades estabelece quando é que o processo deve ser reavaliado (executado).

Ex: Processo síncrono



PROCESS(CLK)

BEGIN

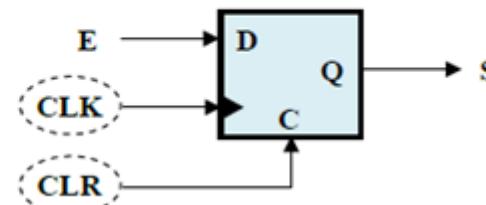
IF (CLK'EVENT) AND (CLK='1')

THEN S <= E;

END IF;

END PROCESS;

Ex: Processo síncrono com clear assíncrono



PROCESS(CLK,CLR)

BEGIN

IF CLR = '1'

THEN S <= '0';

ELSIF (CLK'EVENT) AND (CLK = '1')

THEN S <= E;

END IF;

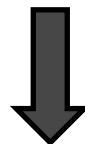
END PROCESS;

VHDL - Processos

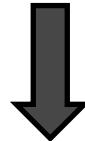
Processamento sequencial versus processamento concorrente

Arquitetura

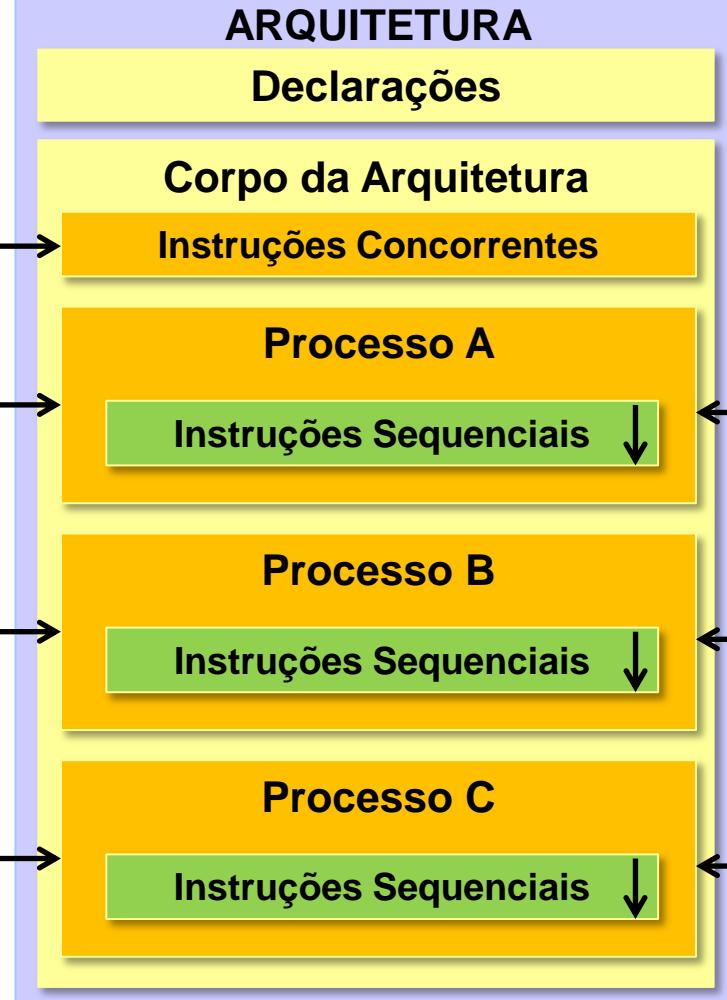
Estrutura de processamento concorrente.
A ordem das instruções não afeta os resultados de simulação e de síntese.



Dentro da arquitetura todas as instruções são executadas concorrentemente entre si.



No corpo da arquitetura os sinais são atualizados no mesmo instante da sua atribuição.



Processo

Estrutura de processamento sequencial.
A ordem das instruções afeta os resultados de simulação e de síntese.



Dentro dos processos as instruções são executadas sequencialmente.

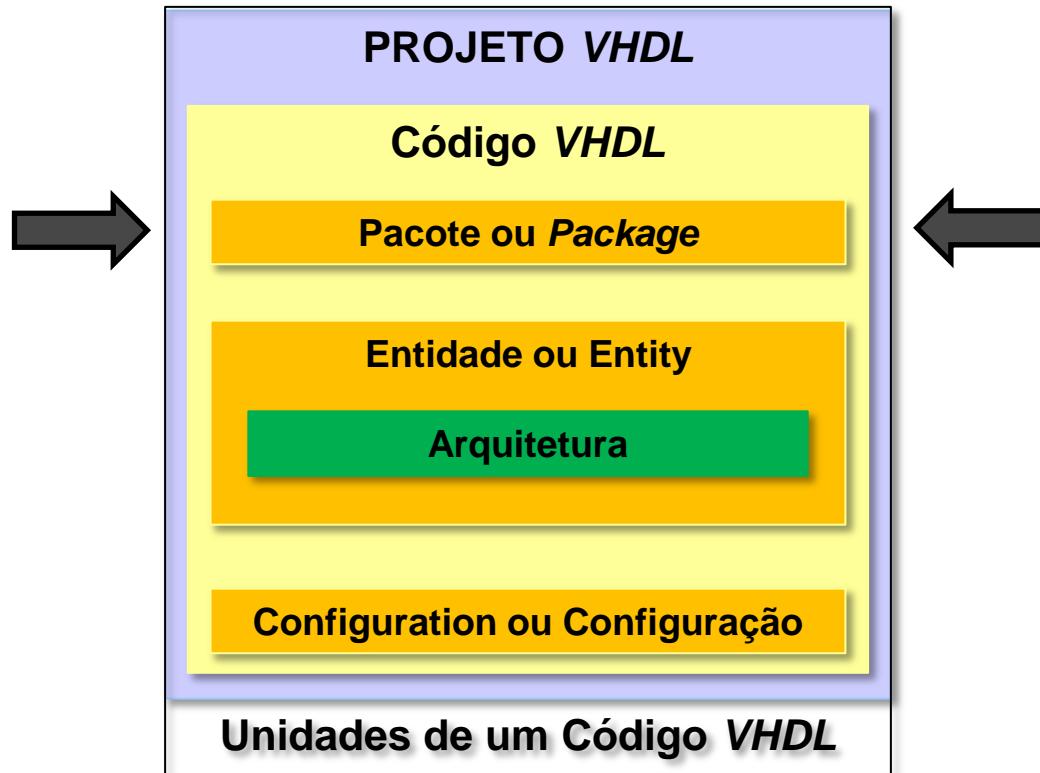


Dentro dos processos os sinais não são atualizados no mesmo instante da sua atribuição, mas apenas no final dos processos.

VHDL - Pacotes

Pacotes ou packages

Package: Unidade opcional que consiste numa biblioteca utilizada para criar definições partilhadas, utilizáveis em outros códigos ou projetos.



VHDL - Pacotes

Pacotes ou packages

- Esta unidade permite a declaração de um conjunto de definições que podem ser partilhadas por vários projetos VHDL.
- A unidade package é composta por uma parte declaratória mandatária (package declaration) e por um corpo opcional (package body).

```
PACKAGE nome_package IS  
    -- Parte declaratória  
END nome_package;
```

```
PACKAGE BODY nome_package IS  
    -- Corpo do package  
END nome_package;
```



VHDL - Pacotes

Pacotes ou packages

- A parte declaratória do *package* pode conter:
 - Declaração de objetos (sinais e constantes);
 - Declaração de componentes;
 - Declaração de tipos e subtipos de dados;
 - Declaração de subprogramas (funções e procedimentos).
- No corpo do *package* são definidos os subprogramas declarados na parte declaratória do *package*.

VHDL - Pacotes

Pacotes ou packages

- Nota 01: As unidades package são geralmente armazenadas em bibliotecas.
- Nota 02: Na norma VHDL estão pré-definidos um conjunto de packages agrupados na biblioteca IEEE:
 - Package Standard;
 - Package Textio;
 - Package Std_Logic_1164.
- Nota 03: Para que um projeto possa utilizar as definições declaradas num package é necessária a sua inclusão no projeto através das diretivas LIBRARY e USE.
LIBRARY nome_biblioteca;
USE nome_biblioteca . nome_package . item;
- Ex:

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

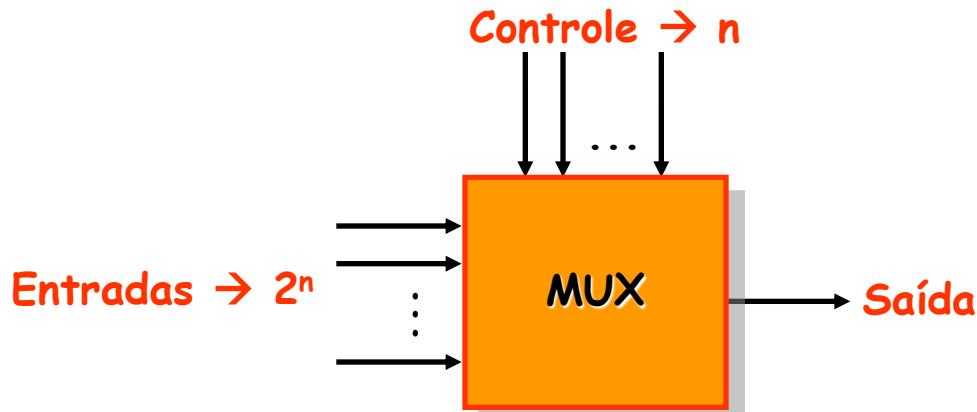
VHDL - Pacotes

Exemplo:

Multiplexador

Multiplexador ou Seletor de Dados: É um circuito lógico que tem diversas entradas e apenas uma saída. MUX seleciona uma única entrada para transmitir para a saída.

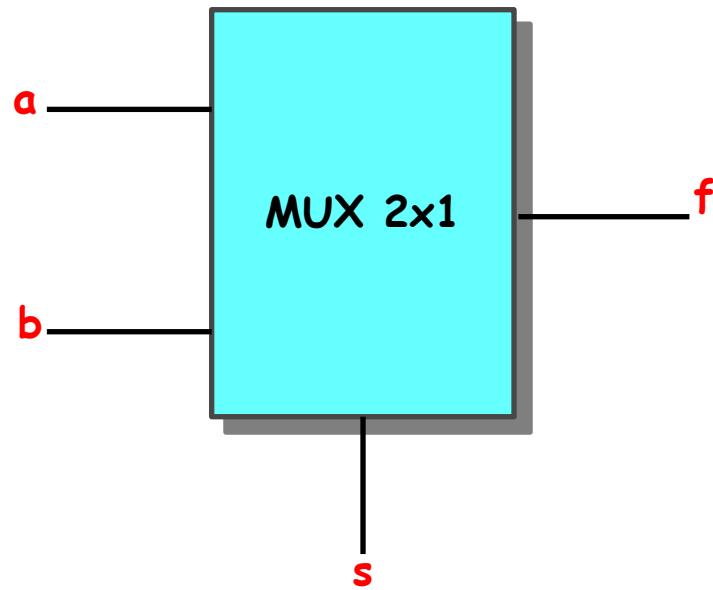
Entradas de Controle: permitem selecionar a entrada a ser transmitida.



VHDL - Pacotes

Exemplo:

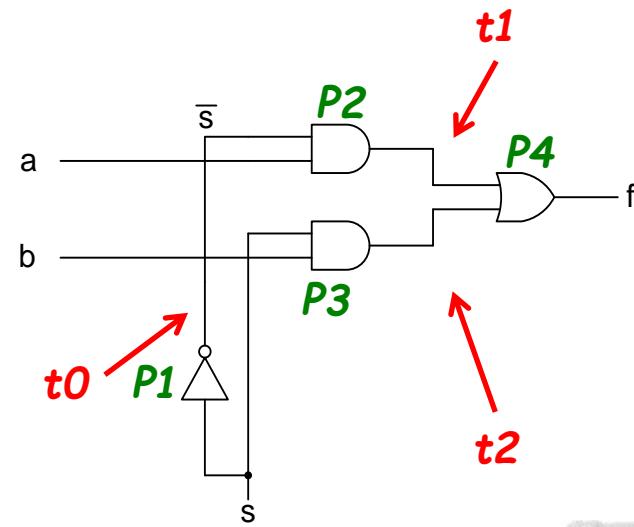
Multiplexador 2 X 1



$$f = \bar{s} \cdot a + s \cdot b$$

TV

s	f
0	a
1	b



VHDL - Pacotes

Exemplo:

- Passo 01: Criação do componente not_1

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY not_1 IS  
    PORT (x : IN BIT;  
          z : OUT BIT);  
END not_1;  
  
ARCHITECTURE logica1 OF not_1 IS  
BEGIN  
    z <= NOT x;  
END logica1;
```

VHDL - Pacotes

Exemplo:

- Passo 02: Criação do componente and_2

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY and_2 IS  
    PORT (x, y : IN BIT;  
          z : OUT BIT);  
END and_2;  
  
ARCHITECTURE logica2 OF and_2 IS  
BEGIN  
    z <= x AND y;  
END logica2;
```

VHDL - Pacotes

Exemplo:

- Passo 03: Criação do componente or_2

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY or_2 IS  
    PORT (x, y : IN BIT;  
          z : OUT BIT);  
END or_2;  
  
ARCHITECTURE logica3 OF or_2 IS  
BEGIN  
    z <= x OR y;  
END logica3;
```

VHDL - Pacotes

Exemplo:

○ Passo 04: Criação do pacote

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
PACKAGE mux2to1_package IS  
  
    COMPONENT or_2  
        PORT(x : IN BIT;  
              y : IN BIT;  
              z : OUT BIT);  
    END COMPONENT;  
  
    COMPONENT and_2  
        PORT(x : IN BIT;  
              y : IN BIT;  
              z : OUT BIT);  
    END COMPONENT;  
  
    COMPONENT not_1  
        PORT(x : IN BIT;  
              z : OUT BIT);  
    END COMPONENT;  
  
END mux2to1_package;
```

VHDL - Pacotes

Exemplo:

○ Passo 05: Código em VHDL → Arquitetura Estrutural

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;
-- USE work.all;
USE work.mux2to1_package.all ;

ENTITY mux2to1 IS
    PORT (a, b : IN BIT;
          s : IN BIT;
          f : OUT BIT);
END mux2to1;
-- continuacao
```

```
ARCHITECTURE estrutural OF mux2to1 IS
SIGNAL t0, t1, t2 : BIT;
BEGIN
    P1: not_1 PORT MAP (s, t0);
    P2: and_2 PORT MAP (t0, a, t1);
    P3: and_2 PORT MAP (s, b, t2);
    P4: or_2 PORT MAP (t1, t2, f);
END estrutural;
```

VHDL - Atrasos

ATRASOS

- Hardwares reais apresentam atrasos:
 - Atrasos de Propagação:
 - São os atrasos das portas lógicas. Correspondem ao tempo que as portas lógicas necessitam para responder às mudanças das entradas.
 - Atrasos de Transporte:
 - São associados com o atraso de tempo ao longo de fios de conexões.

VHDL - Atrasos

ATRASOS

- VHDL permite especificar esses atrasos com os comandos:
 - AFTER
 - TRANSPORT AFTER

VHDL - Atrasos

ATRASOS

○ Exemplo:

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY not_1 IS  
    PORT (x : IN BIT;  
          z : OUT BIT);  
END not_1;  
  
ARCHITECTURE logica OF not_1 IS  
BEGIN  
    z <= NOT x AFTER 5 ns;  
END logica;
```

Porta NOT com atraso
de propagação de 5 ns

VHDL - Atrasos

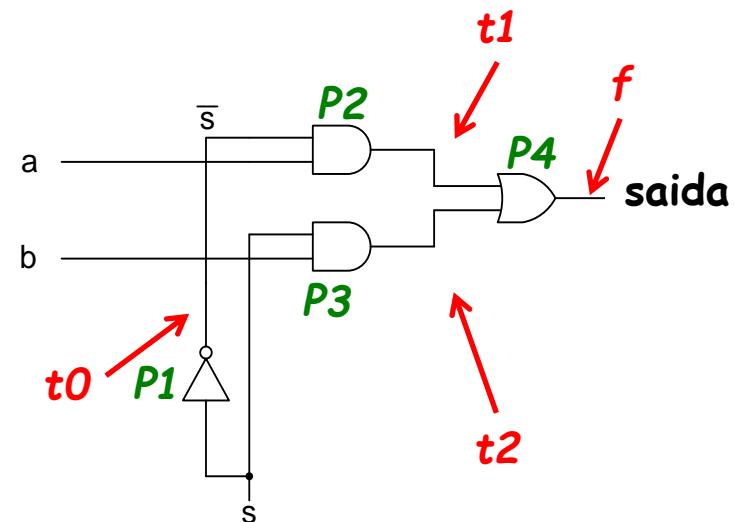
ATRASOS

○ Exemplo MUX 2x1

```
ENTITY mux2to1 IS
    PORT (a, b : IN BIT;
          s : IN BIT;
          saida : OUT BIT);
END mux2to1;
```

-- continuacao

Multiplexador 2x1 com atraso
de transporte de 10 ps



VHDL - Atrasos

ATRASOS

○ Exemplo MUX 2x1

```
ARCHITECTURE estrutural OF mux2to1 IS
COMPONENT and_2
PORT (x,y: IN BIT;
      z: OUT BIT);
END COMPONENT;
COMPONENT or_2
PORT (x,y: IN BIT;
      z: OUT BIT);
END COMPONENT;
COMPONENT not_1
PORT (x: IN BIT; z: OUT BIT);
END COMPONENT;
```

```
SIGNAL t0, t1, t2, f : BIT;
BEGIN
  P1: not_1 PORT MAP (s, t0);
  P2: and_2 PORT MAP (t0, a, t1);
  P3: and_2 PORT MAP (s, b, t2);
  P4: or_2 PORT MAP (t1, t2, f);
  saida <= TRANSPORT (f) AFTER 10 ps;
END estrutural;
```

Aula de Hoje

- Subprogramação
 - Funções
 - Procedimentos

Subprogramação

- Subprogramação simplifica o código possibilitando o reuso de funcionalidades implementadas e eliminando a repetição de trechos de códigos idênticos. Pode ser feita por meio de:
 - Funções
 - Procedimentos

Funções e Procedimentos

- **Funções e Procedimentos:**
 - Só podem receber comandos sequenciais
 - A declaração de um subprograma é opcional
 - O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - No corpo da arquitetura da entidade
 - No corpo de um subprograma
 - No corpo de um pacote

Funções e Procedimentos

- **Funções:**

- Retornam um único valor
- Todos os parâmetros são de entrada
- Invocadas por uma expressão → Exemplo: `s <= soma(a, b);`

Procedimentos:

- Retornam mais de um valor
- Podem ter parâmetros de entrada, saída e de entrada/saída
- Invocadas por um comando → Exemplo: `soma(a, b, s);`

Funções e Procedimentos

Estrutura de Função

```
FUNCTION nome_funcao [(lista_parametros)] RETURN tipodado  
IS  
    [declarações]  
BEGIN  
    :  
    Comandos  
    :  
END nome_funcao;
```

[Lista de parâmetros] => Nas funções essa lista indica os parâmetros de entrada

[Lista de parâmetros] => Os parâmetros podem ser constantes ou sinais, mas não variáveis.
Se o parâmetro for constante pode-se omitir a palavra CONSTANT.

Exemplo de parâmetros: (par1, par2: tipo_dado1; SIGNAL par3: tipo_dado2)

Funções e Procedimentos

Estrutura de Procedimento

```
PROCEDURE nome_procedimento [(lista_parametros)]  
IS  
[declarações]  
BEGIN  
:  
Comandos  
:  
END nome_procedimento;
```

[Lista de parâmetros] => Nos procedimentos essa lista indica os parâmetros de entrada e de saída. Pode-se usar constantes, sinais e variáveis.

Exemplo de parâmetros: (par1, par2: IN tipo_dado1; SIGNAL par3: OUT tipo_dado2)

Nos procedimentos deve-se declarar os modos de operação dos parâmetros: IN, OUT, INOUT.

Funções e Procedimentos

Exemplo 01: Função

- Função para converter números binários de 8 dígitos em inteiros

```
FUNCTION BinToDec (ent: BIT_VECTOR (7 DOWNTO 0))
    RETURN INTEGER
IS
    VARIABLE Result, i: INTEGER;
BEGIN
    Result := 0;
    i := 1;
    FOR j IN 0 TO 7 LOOP
        IF (ent (j) = '1') THEN
            Result := Result + i;
        END IF;
        i := i * 2;
    END LOOP;
    RETURN Result;
END BinToDec;
```

Funções e Procedimentos

Exemplo 01: Função

- Para invocar a função dentro de um componente é necessário associá-la a algum objeto:
- **Result:= BinToDec (ent);**
- “Result” é uma variável do tipo INTEGER e “ent” um BIT_VECTOR de 8 posições

Funções e Procedimentos

Exemplo 02: Função

- Caso a função necessitasse de mais parâmetros, na chamada eles deveriam ser separados por vírgulas:
- **m := maior (a,b);**

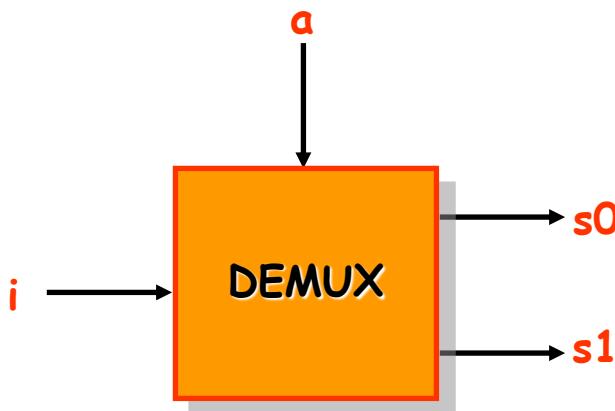
Exemplo de função para comparar 2 valores e retornar o maior deles

```
FUNCTION maior (a,b: INTEGER)
    RETURN INTEGER
IS
BEGIN
    IF a > b THEN RETURN (a);
    ELSE RETURN (b);
    END IF;
END maior;
```

Funções e Procedimentos

Exemplo: Procedimento

- Procedimento para implementar um circuito DEMUX 1x2



- Dado = i
- Controle = a
- Saídas = s0, s1

```
PROCEDURE Demux_1x2 (a, i: IN BIT;  
                      s0, s1: OUT BIT)  
IS  
BEGIN  
    IF a = '0' THEN s0 <= i;  
                  s1 <= '0';  
    ELSE           s0 <= '0';  
                  s1 <= i;  
    END IF;  
END Demux_1x2;
```

Para invocar o procedimento durante a execução de um código:

Demux_1x2 (controle, ent, saída0,saida1);

-- ou

Demux_1x2 (a, i, s0,s1);

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

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

ENTITY portasa IS
    PORT (a : IN STD_LOGIC;
          b : IN STD_LOGIC;
          c : OUT STD_LOGIC;
          d : OUT STD_LOGIC);
END portasa; --continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
ARCHITECTURE function_procedure OF portas IS
```

```
PROCEDURE and_2 (a:IN STD_LOGIC; b:IN STD_LOGIC; c:OUT STD_LOGIC) IS
BEGIN
    c := a AND b;
END;
```

```
FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS
BEGIN
    RETURN a OR b;
END; --continua
```

Funções e Procedimentos

- **O corpo de um subprograma pode ser inserido:**
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

BEGIN

PROCESS(a, b) -- regiao de codigo sequencial

VARIABLE av, bv, cv : STD_LOGIC;

BEGIN

c <= or_2(a, b);

av := a;

bv := b;

and_2(a => av, b => bv, c => cv);

d <= cv;

END PROCESS;

END function_procedure;

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

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

```
PACKAGE portasp_package IS  
    PROCEDURE and_2(a : IN STD_LOGIC; b : IN STD_LOGIC;  
                    c : OUT STD_LOGIC);  
    FUNCTION or_2(a, b : STD_LOGIC) RETURN STD_LOGIC;  
END portasp_package; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
PACKAGE BODY portasp_package IS
    PROCEDURE and_2(a : IN STD_LOGIC; b : IN STD_LOGIC;
                    c : OUT STD_LOGIC) IS
        BEGIN
            c := a AND b;
        END;
    FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS
        BEGIN
            RETURN a OR b;
        END;
END portasp_package; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
LIBRARY work;
```

```
USE work.portasp_package.all;
```

```
LIBRARY ieee;
```

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

```
ENTITY portasp IS
```

```
    PORT (a : IN STD_LOGIC;
```

```
          b : IN STD_LOGIC;
```

```
          c : OUT STD_LOGIC;
```

```
          d : OUT STD_LOGIC);
```

```
END portasp; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
ARCHITECTURE function_procedure OF portasp IS
BEGIN
    PROCESS(a, b)      -- regiao de codigo sequencial
    VARIABLE av, bv, cv : STD_LOGIC;
    BEGIN
        c <= or_2(a, b);
        av := a;
        bv := b;
        and_2(a => av, b => bv, c => cv);
        d <= cv;
    END PROCESS;
END function_procedure;
```

Funções e Procedimentos

- **O corpo de um subprograma pode ser inserido:**
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY portase IS  
    PORT (a : IN STD_LOGIC;  
          b : IN STD_LOGIC;  
          c : OUT STD_LOGIC;  
          d : OUT STD_LOGIC);  
  
PROCEDURE and_2 (a:IN STD_LOGIC; b:IN STD_LOGIC; c:OUT STD_LOGIC) IS  
    BEGIN  
        c := a AND b;  
    END; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS
```

```
    BEGIN
```

```
        RETURN a OR b;
```

```
    END;
```

```
END portase; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO SEQUENCIAL

```
ARCHITECTURE function_procedure OF portase IS
```

```
BEGIN
```

```
    PROCESS(a, b)                                -- regiao de codigo sequencial
```

```
        VARIABLE av, bv, cv : STD_LOGIC;
```

```
        BEGIN
```

```
            c <= or_2(a, b);
```

```
            av := a;
```

```
            bv := b;
```

```
            and_2(a => av, b => bv, c => cv);
```

```
            d <= cv;
```

```
        END PROCESS;
```

```
    END function_procedure;
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY portasac IS  
    PORT (a : IN STD_LOGIC;  
          b : IN STD_LOGIC;  
          c : OUT STD_LOGIC;  
          d : OUT STD_LOGIC);  
END portasac; --continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
ARCHITECTURE function_procedure OF portasac IS
```

```
PROCEDURE and_2 (a:IN STD_LOGIC; b:IN STD_LOGIC;  
                  SIGNAL c:OUT STD_LOGIC) IS  
  
BEGIN  
  
    c <= a AND b;  
  
END;
```

```
FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS  
  
BEGIN  
  
    RETURN a OR b;  
  
END; --continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo da arquitetura da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
BEGIN
```

```
c <= or_2(a, b);           -- regiao de codigo concorrente  
and_2(a, b, d);
```

```
END function_procedure;
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
PACKAGE portaspc_package IS  
    PROCEDURE and_2(a : IN STD_LOGIC; b : IN STD_LOGIC;  
                    SIGNAL c : OUT STD_LOGIC);  
    FUNCTION or_2(a, b : STD_LOGIC) RETURN STD_LOGIC;  
END portaspc_package; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
PACKAGE BODY portaspc_package IS
    PROCEDURE and_2(a : IN STD_LOGIC; b : IN STD_LOGIC;
                    SIGNAL c : OUT STD_LOGIC) IS
        BEGIN
            c <= a AND b;
        END;
    FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS
        BEGIN
            RETURN a OR b;
        END;
END portaspc_package; -- continua
```

Funções e Procedimentos

- **O corpo de um subprograma pode ser inserido:**
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
LIBRARY work;
```

```
USE work.portaspc_package.all;
```

```
LIBRARY ieee;
```

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

```
ENTITY portaspc IS
```

```
    PORT (a : IN STD_LOGIC;
```

```
          b : IN STD_LOGIC;
```

```
          c : OUT STD_LOGIC;
```

```
          d : OUT STD_LOGIC);
```

```
END portaspc; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - No corpo de um pacote
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
ARCHITECTURE function_procedure OF portaspc IS
BEGIN
    c <= or_2(a, b); -- regiao de codigo concorrente
    and_2(a, b, d);
END function_procedure;
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
LIBRARY ieee;
```

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

```
ENTITY portasec IS
```

```
    PORT (a : IN STD_LOGIC;  
          b : IN STD_LOGIC;  
          c : OUT STD_LOGIC;  
          d : OUT STD_LOGIC);
```

```
PROCEDURE and_2 (a:IN STD_LOGIC; b:IN STD_LOGIC;
```

```
                  SIGNAL c:OUT STD_LOGIC) IS
```

```
BEGIN
```

```
    c <= a AND b;
```

```
END; -- continua
```

Funções e Procedimentos

- O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
FUNCTION or_2 (a, b : STD_LOGIC) RETURN STD_LOGIC IS
```

```
    BEGIN
```

```
        RETURN a OR b;
```

```
    END;
```

```
END portasec; -- continua
```

Funções e Procedimentos

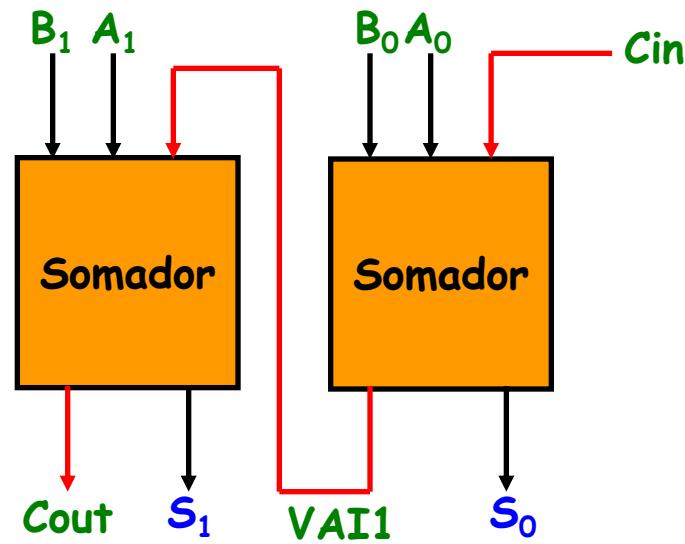
- O corpo de um subprograma pode ser inserido:
 - Na declaração da entidade
 - Função e procedimento invocados de uma região de CÓDIGO CONCORRENTE

```
ARCHITECTURE function_procedure OF portasec IS
BEGIN
    c <= or_2(a, b);      -- regiao de codigo concorrente
    and_2(a, b, d);
END function_procedure;
```

VHDL - Funções e Procedimentos

- Exercício:
- Implemente um código para o circuito somador de 2 bits usando Funções e Procedimentos.
- Use os seguintes nomes para as entradas e saídas:
 - A,B: para os dados;
 - Cin para a entrada do vem-1;
 - VAI1 para o vai-1 interno;
 - Cout para a saída do vai-1 final;
 - S para a saída da soma.
 - Veja figura a seguir

Somador Completo de 2 bits

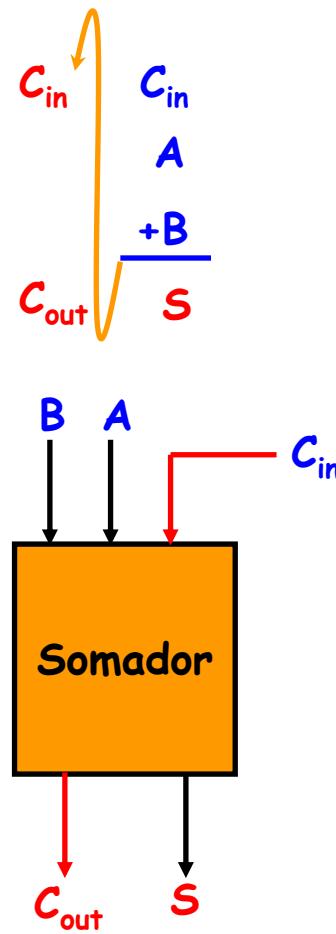


Somador Completo de 1 Bit

Entradas

Saidas

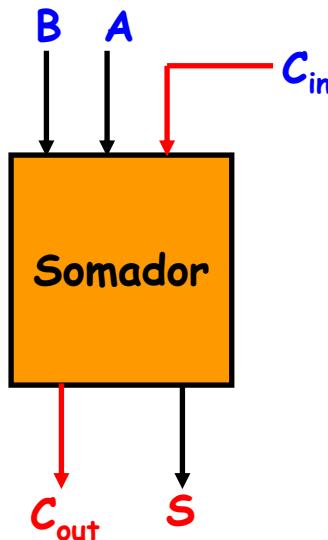
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



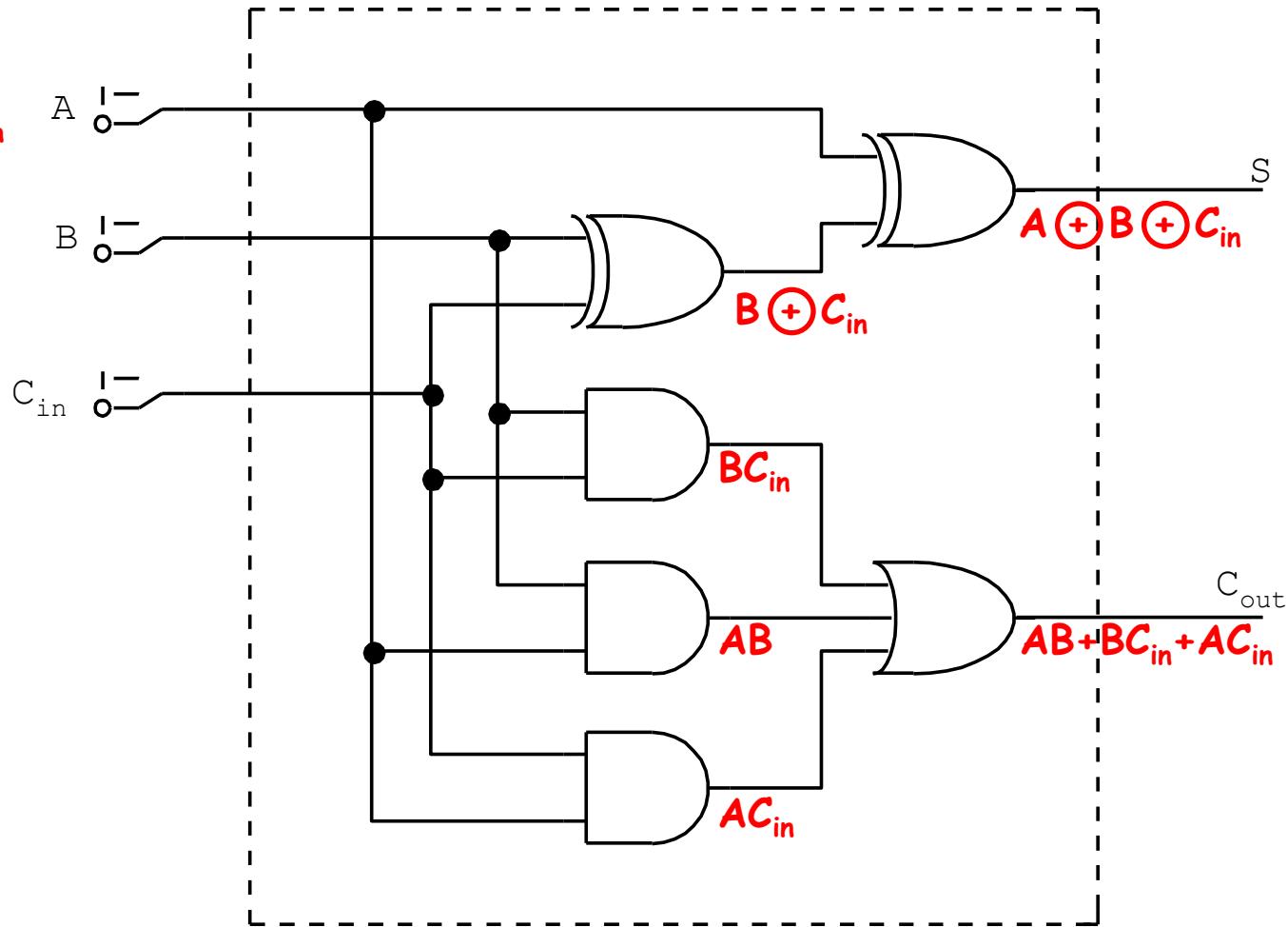
Somador Completo de 1 bit

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + BC_{in} + AC_{in}$$



Circuito Somador Completo de 1 bitt



VHDL - Funções e Procedimentos

Solução 01: Inserção no Corpo da Arquitetura da Entidade

```
LIBRARY ieee;
USE ieee.Std_Logic_1164.all;

ENTITY som_2_bits IS

    GENERIC (n: INTEGER := 2);                                -- numero de bits

    PORT      (A, B : IN BIT_VECTOR (n-1 DOWNTO 0);          -- entradas do somador
               Cin : IN BIT;                                     -- vem 1
               S : OUT BIT_VECTOR (n-1 DOWNTO 0);                -- soma
               Cout : OUT BIT);                                 -- vai 1

END som_2_bits;
```

VHDL - Funções e Procedimentos

Solução 01: Inserção no Corpo da Arquitetura da Entidade

```
ARCHITECTURE function_add OF som_2_bits IS

CONSTANT m : INTEGER := 2;

FUNCTION Soma(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT_VECTOR IS

VARIABLE VAI1 : BIT_VECTOR(m DOWNTO 0);
VARIABLE RES : BIT_VECTOR(m-1 DOWNTO 0);

BEGIN

VAI1(0) := VEM1;
loopSoma : FOR i IN 0 TO m-1 LOOP
    RES(i) := X(i) XOR Y(i) XOR VAI1(i);
    VAI1(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1(i)) OR (X(i) AND VAI1(i));
END LOOP loopSoma;
RETURN RES;                                -- Resultado de Soma

END Soma;
```

VHDL - Funções e Procedimentos

Solução 01: Inserção no Corpo da Arquitetura da Entidade

```
FUNCTION Couti(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT IS
    VARIABLE VAI1 : BIT_VECTOR(m DOWNTO 0);
BEGIN
    VAI1(0) := VEM1;
    loopVai1 : FOR i IN 0 TO m-1 LOOP
        VAI1(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1(i)) OR (X(i) AND VAI1(i));
    END LOOP loopVai1;
    RETURN VAI1(m);                                -- Resultado do Vai 1
END Couti;

BEGIN
    --regiao de codigo sequencial
    PROCESS (A, B, Cin)
        BEGIN
            S <= Soma(A, B, Cin);                --Soma
            Cout <= Couti(A, B, Cin);           -- Vai 1
        END PROCESS;
END function_add;
```

VHDL - Funções e Procedimentos

Solução 02: Inserção no Pacote

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

PACKAGE som_2_bits_package IS

    FUNCTION Soma(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT_VECTOR;

    FUNCTION Couti(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT;

END som_2_bits_package;
```

VHDL - Funções e Procedimentos

Solução 02: Inserção no Pacote

```
PACKAGE BODY som_2_bits_package IS

    CONSTANT m : INTEGER := 2;

    FUNCTION Soma(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT_VECTOR IS

        VARIABLE VAI1 : BIT_VECTOR(m DOWNTO 0);
        VARIABLE RES : BIT_VECTOR(m-1 DOWNTO 0);

    BEGIN

        VAI1(0) := VEM1;
        loopSoma: FOR i IN 0 TO m-1 LOOP
            RES(i) := X(i) XOR Y(i) XOR VAI1(i);
            VAI1(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1(i)) OR (X(i) AND VAI1(i));
        END LOOP loopSoma;
        RETURN RES;                                -- Resultado de Soma

    END Soma;

END som_2_bits_package;
```

VHDL - Funções e Procedimentos

Solução 02: Inserção no Pacote

```
FUNCTION Couti(X : BIT_VECTOR; Y : BIT_VECTOR; VEM1 : BIT) RETURN BIT IS
    VARIABLE VAI1 : BIT_VECTOR(m DOWNTO 0);
BEGIN
    VAI1(0) := VEM1;
    loopVai1: FOR i IN 0 TO m-1 LOOP
        VAI1(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1(i)) OR (X(i) AND VAI1(i));
    END LOOP loopVai1;
    RETURN VAI1(m);                                -- Resultado do Vai 1
END Couti;

END som_2_bits_package;
```

VHDL - Funções e Procedimentos

Solução 02: Inserção no Pacote

```
LIBRARY ieee;
USE ieee.Std_Logic_1164.all;

-- LIBRARY work;
USE work.som_2_bits_package.all;

ENTITY som_2_bits_p IS

  GENERIC (n      : INTEGER  := 2);                      -- numero de bits

  PORT    (A, B   : IN BIT_VECTOR (n-1 DOWNTO 0);        -- entradas do somador
          Cin   : IN BIT;                                -- vem 1
          S     : OUT BIT_VECTOR (n-1 DOWNTO 0);          -- soma
          Cout  : OUT BIT);                             -- vai 1

END som_2_bits_p;
```

VHDL - Funções e Procedimentos

Solução 02: Inserção no Pacote

```
ARCHITECTURE function_add OF som_2_bits_p IS
BEGIN
    --regiao de codigo sequencial
    PROCESS (A, B, Cin)
        BEGIN
            S <= Soma (A, B, Cin);          --Soma
            Cout <= Couti(A, B, Cin);      -- Vai 1
        END PROCESS;
    END function_add;
```

VHDL - Funções e Procedimentos

Solução 03: Inserção no Corpo da Declaração da Entidade

```
LIBRARY ieee;
USE ieee.Std_Logic_1164.all;

ENTITY som_2_bits_e IS

  GENERIC (n      : INTEGER  := 2);                      -- numero de bits

  PORT   (A, B    : IN  BIT_VECTOR (n-1 DOWNTO 0);       -- entradas do somador
          Cin    : IN  BIT;                                -- vem 1
          S      : OUT BIT_VECTOR (n-1 DOWNTO 0);           -- soma
          Cout   : OUT BIT);                              -- vai 1
```

VHDL - Funções e Procedimentos

Solução 03: Inserção no Corpo da Declaração da Entidade

```
CONSTANT m : INTEGER := 2;

PROCEDURE Soma(X:IN BIT_VECTOR(m-1 DOWNTO 0);Y:IN BIT_VECTOR(m-1 DOWNTO 0);
               VEM1:IN BIT; SIGNAL RES:OUT BIT_VECTOR(m-1 DOWNTO 0);
               SIGNAL VAI1:OUT BIT) IS

   VARIABLE VAI1i : BIT_VECTOR(m DOWNTO 0);

BEGIN

   VAI1i(0) := VEM1;
   loopSoma: FOR i IN 0 TO m-1 LOOP
      RES(i) <= X(i) XOR Y(i) XOR VAI1i(i);
      VAI1i(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1i(i)) OR (X(i) AND VAI1i(i));
   END LOOP loopSoma;
   VAI1 <= VAI1i(m);

END Soma;

END som_2_bits_e;
```

VHDL - Funções e Procedimentos

Solução 03: Inserção no Corpo da Declaração da Entidade

```
ARCHITECTURE procedure_add OF som_2_bits_e IS
BEGIN
    Soma(A, B, Cin, S, Cout);          --regiao de codigo concorrente
                                         -- Soma e Vai 1
END procedure_add;
```

VHDL - Funções e Procedimentos

Solução 04: Inserção no Pacote

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

PACKAGE som_2_bits_package_p IS

    PROCEDURE Soma(X : IN BIT_VECTOR; Y : IN BIT_VECTOR;
                    VEM1 : IN BIT; SIGNAL RES : OUT BIT_VECTOR;
                    SIGNAL VAI1 : OUT BIT);

END som_2_bits_package_p;
```

VHDL - Funções e Procedimentos

Solução 04: Inserção no Pacote

```
PACKAGE BODY som_2_bits_package_p IS

    CONSTANT m : INTEGER := 2;

    PROCEDURE Soma(X:IN BIT_VECTOR(m-1 DOWNTO 0); Y:IN BIT_VECTOR(m-1 DOWNTO 0);
                    VEM1:IN BIT; SIGNAL RES:OUT BIT_VECTOR(m-1 DOWNTO 0);
                    SIGNAL VAI1:OUT BIT) IS

        VARIABLE VAI1i : BIT_VECTOR(m DOWNTO 0);

        BEGIN

            VAI1i(0) := VEM1;
            loopSoma: FOR i IN 0 TO m-1 LOOP
                RES(i) <= X(i) XOR Y(i) XOR VAI1i(i);
                VAI1i(i+1) := (X(i) AND Y(i)) OR (Y(i) AND VAI1i(i)) OR (X(i) AND VAI1i(i));
            END LOOP loopSoma;
            VAI1 <= VAI1i(m);

        END Soma;

    END som_2_bits_package_p;
```

VHDL - Funções e Procedimentos

Solução 04: Inserção no Pacote

```
LIBRARY ieee;
USE ieee.Std_Logic_1164.all;

-- LIBRARY work;
USE work.som_2_bits_package_pp.all;

ENTITY som_2_bits_pp IS
    GENERIC (n      : INTEGER  := 2);                      -- numero de bits
    PORT      (A, B      : IN BIT_VECTOR (n-1 DOWNTO 0);   -- entradas do somador
              Cin      : IN BIT;                            -- vem 1
              S        : OUT BIT_VECTOR (n-1 DOWNTO 0);     -- soma
              Cout     : OUT BIT);                          -- vai 1

END som_2_bits_pp;

ARCHITECTURE procedure_add OF som_2_bits_pp IS
BEGIN
    Soma(A, B, Cin, S, Cout);                         --regiao de codigo concorrente
                                                       -- Soma e Vai 1

END procedure_add;
```

Resumo da Aula de Hoje

Tópicos mais importantes:

- **Funções**
- **Procedimentos**

Previsão de Datas para as Avaliações

Previsão

- Provas:
 - 1^a. avaliação - Conteúdo → Itens 01 a 11 → **27/07/2015**
 - 2^a. avaliação - Conteúdo → Itens 12 a 14 → **31/08/2015**
- Trabalho - Data da Entrega:
 - 1^a. Subavaliação - Projetos - Parte I → **03/08/2015**
 - 2^a. Subavaliação - Projetos - Parte II → **09/09/2015**
 - 3^a. Subavaliação - Práticas de Laboratório → **09/09/2015**
- Avaliação Final: **28/09/2015**

Próxima da Aula

- **Projetos**
 - **Projetos de Circuitos Combinacionais em VHDL**
 - **Projetos de Circuitos Sequenciais em VHDL**