



Circuitos Digitais II - 6882

André Barbosa Verona
Nardênio Almeida Martins

Universidade Estadual de Maringá
Departamento de Informática

Bacharelado em Ciência da Computação

Aula de Hoje

- **Revisão da aula anterior**
 - Comandos de Repetição
 - Comando *FOR LOOP*
 - Comando *WHILE LOOP*
 - Comando *NEXT e EXIT*
- **Processos**
- **Pacotes**
- **Atrasos**

Revisão

- **Comandos de Repetição**
 - **Comando *FOR LOOP***
 - **Comando *WHILE LOOP***
 - **Comando *NEXT e EXIT***

VHDL - Comandos de Repetição

Comandos de Repetição

- Comandos de Repetição permitem executar repetidamente uma sequência de instruções
- Em VHDL há 2 esquemas de repetição:
 - FOR LOOP
 - WHILE LOOP

VHDL - Comandos de Repetição

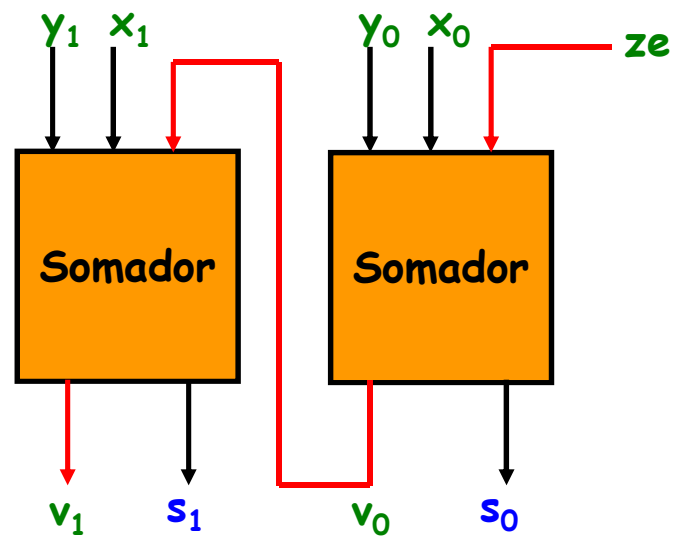
FOR LOOP

- Permite a repetição de instruções uma quantidade de vezes preestabelecida
- Restrição de uso dentro de procedimentos, funções e processos
- Um contador vai sendo incrementado ou decrementado a cada iteração até atingir um valor limite.
- Contador não pode ser alterado com operações de atribuição

Sintaxe

```
FOR contador IN valor_inicial TO|DOWNTO valor_final LOOP  
    comandos  
    ...  
END LOOP;
```

Somador de 2 bits



VHDL - Comandos de Repetição

FOR LOOP

- Exercício 02:
- Implemente um código para o circuito somador de 2 bits usando FOR LOOP.
- Use os seguintes nomes para as entradas e saídas:
 - x,y: para os dados;
 - ze para a entrada do vem-1;
 - v vai-1 interno;
 - zs para a saída do vai-1 final;
 - s para a saída da soma.

VHDL - Comandos de Repetição

Solução

• Exercício 02:

```
ENTITY som_2bits IS
  GENERIC (n          : INTEGER := 2);           -- numero de bits
  PORT   (x, y        : IN  BIT_VECTOR (n-1 DOWNT0 0); -- entradas do somador
          ze          : IN  BIT;                -- vem um
          s           : OUT BIT_VECTOR (n-1 DOWNT0 0); -- saida
          zs          : OUT BIT);                -- vai um
END som_2bits;

ARCHITECTURE logica OF som_2bits IS
BEGIN
  PROCESS (x, y, ze)
    VARIABLE v : BIT_VECTOR (n DOWNT0 0); -- vai um interno
    BEGIN
      v(0) := ze;
      FOR i IN 0 TO n-1 LOOP
        s(i) <= x(i) XOR y(i) XOR v(i);
        v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
      END LOOP;
      zs <= v(n);
    END PROCESS;
END logica;
```


VHDL - Comandos de Repetição

WHILE LOOP

- Permite a repetição de instruções se uma condição for verdadeira
- A iteração termina se a condição for falsa
- Restrição de uso dentro de procedimentos, funções e processos

Sintaxe

```
WHILE condição LOOP
    comandos
    ...
END LOOP;
```

VHDL - Comandos de Repetição

WHILE LOOP

- Exercício 04:
- Implemente um código para o circuito somador de 2 bits usando WHILE LOOP.
- Use os seguintes nomes para as entradas e saídas:
 - x,y: para os dados;
 - ze para a entrada do vem-1;
 - v vai-1 interno;
 - zs para a saída do vai-1 final;
 - s para a saída da soma.

VHDL - Comandos de Repetição

Solução

• Exercício 04:

```
ENTITY som_2bits IS
  GENERIC (n          : INTEGER := 2);           -- numero de bits
  PORT  (x, y          : IN  BIT_VECTOR (n-1 DOWNT0 0); -- entradas do somador
         ze            : IN  BIT;                -- vem um
         s             : OUT BIT_VECTOR (n-1 DOWNT0 0); -- saida
         zs            : OUT BIT);               -- vai um
END som_2bits;

ARCHITECTURE teste OF som_2bits IS
BEGIN
  PROCESS (x, y, ze)
    VARIABLE i      : INTEGER ;
    VARIABLE v      : BIT_VECTOR (n DOWNT0 0); -- vai um interno
  BEGIN
    i := 0; -- deve ser atualizado a cada iteracao
    v(0) := ze;
    WHILE i <= n-1 LOOP -- executado enquanto verdadeiro
      s(i) <= x(i) XOR y(i) XOR v(i);
      v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
      i := i+1;
    END LOOP;
    zs <= v(n);
  END PROCESS;
END teste;
```

VHDL – Comandos de Repetição

NEXT e EXIT

- Permitem alterar a sequência das operações executadas em um comando LOOP.
- EXIT interrompe a execução do laço de repetição e força o código a prosseguir para o comando posterior ao LOOP.
- NEXT causa um salto para o final do laço de repetição e segue para a próxima iteração do comando LOOP.

VHDL - Comandos de Repetição

NEXT e EXIT

Sintaxe NEXT

NEXT;	--pula para a proxima iteracao
NEXT WHEN condicao_1;	--pula para a proxima iteracao caso --condicao_1 seja verdadeira
NEXT loop_1 WHEN condicao_3;	--pula para a loop_1
abc: NEXT WHEN condicao_2;	--idem, rotulo abc opcional

VHDL - Comandos de Repetição

NEXT e EXIT

Sintaxe EXIT

```
EXIT;                                --termina a iteracao
EXIT WHEN condicao_1;                 --termina a iteracao caso
                                     --condicao_1 seja verdadeira
EXIT loop_1 WHEN condicao_3;          --termina a iteracao e salta para
                                     --loop_1
abc: EXIT WHEN condicao_2;             --idem, rotulo abc opcional
```

VHDL - Comandos de Repetição

NEXT

- Exemplo:

```
process (flag)
variable a, b : integer := 0 ;
begin
    a := 0 ; b := 3 ;
    for i in 0 to 7 loop
        b := b + 1 ;
        if i = 5 then next;
    end if ;
    a := a + b ;
    end loop;
end process;
```

VHDL - Comandos de Repetição

EXIT

- Exemplo:

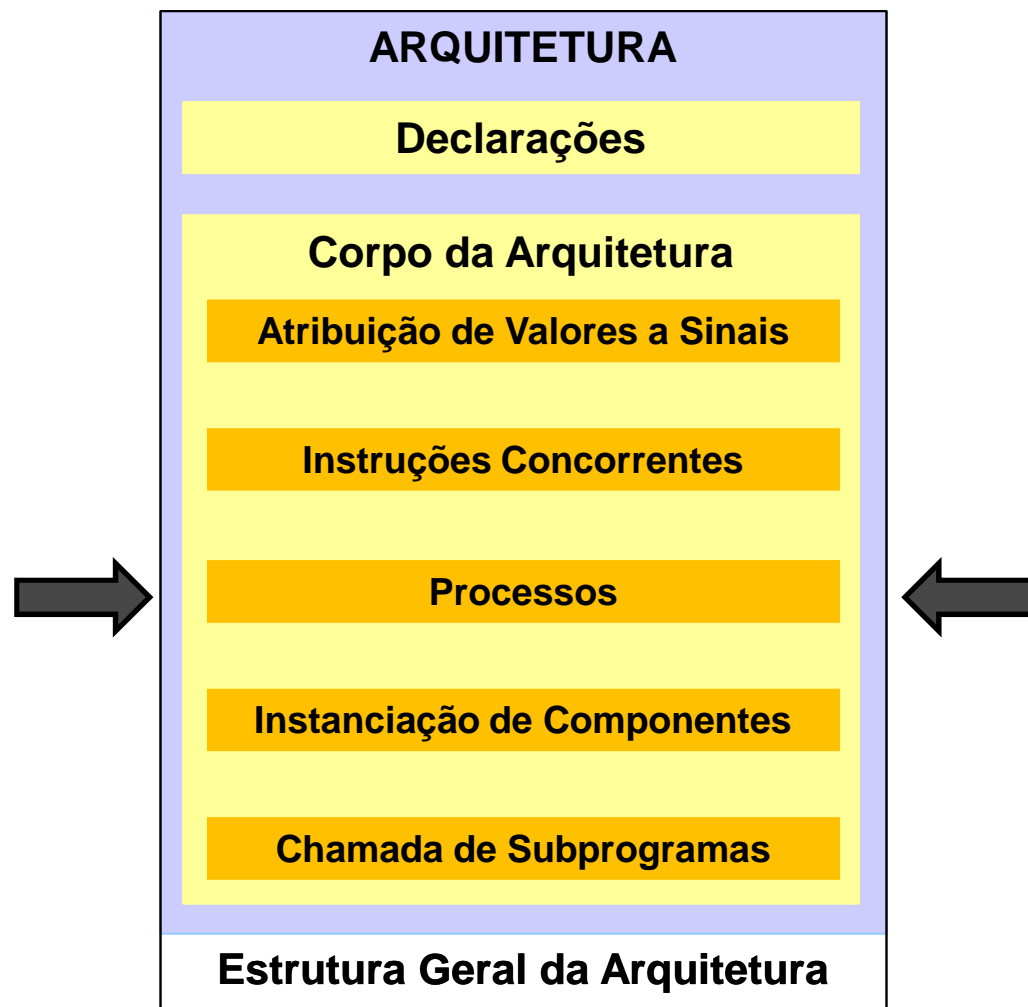
```
process (flag)
variable sum, cnt : integer := 0 ;
begin
    sum := 0; cnt := 0;
    loop
        cnt := cnt + 1 ;
        sum := sum + cnt ;
        exit when sum > 100 ;
    end loop;
end process;
```


Aula de Hoje

- **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;

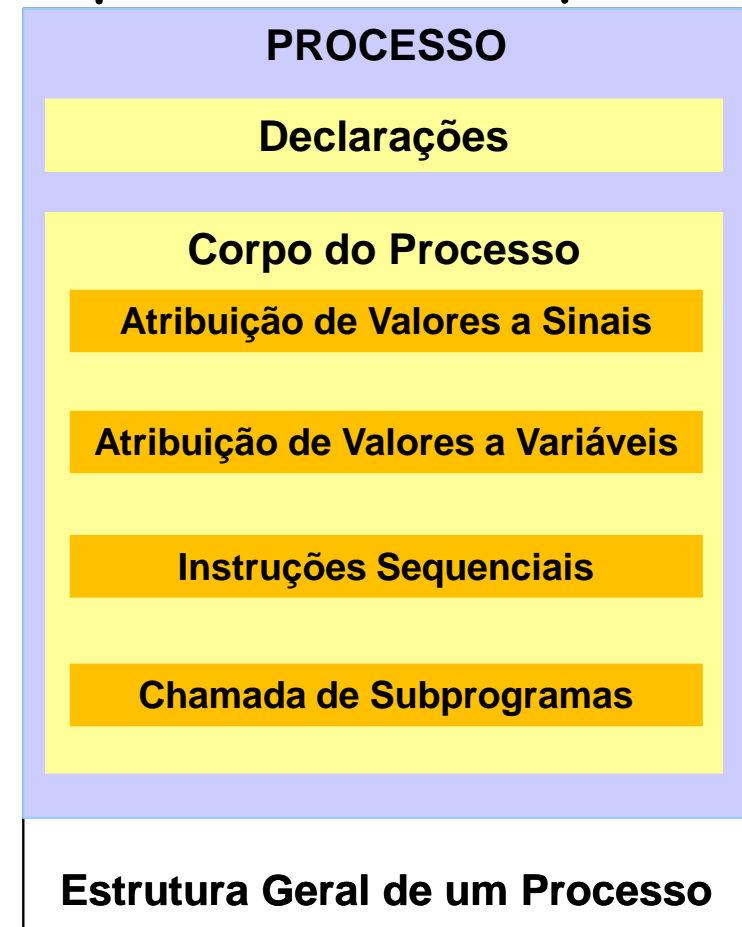
- A parte declaratória do processo pode conter:
 - Declaração de objetos (variáveis e constantes);
 - Declaração de tipos e subtipos de dados;
 - Declaração de subprogramas.

VHDL - Processos

Processos

• O corpo do processo é uma estrutura de processamento sequencial que pode conter:

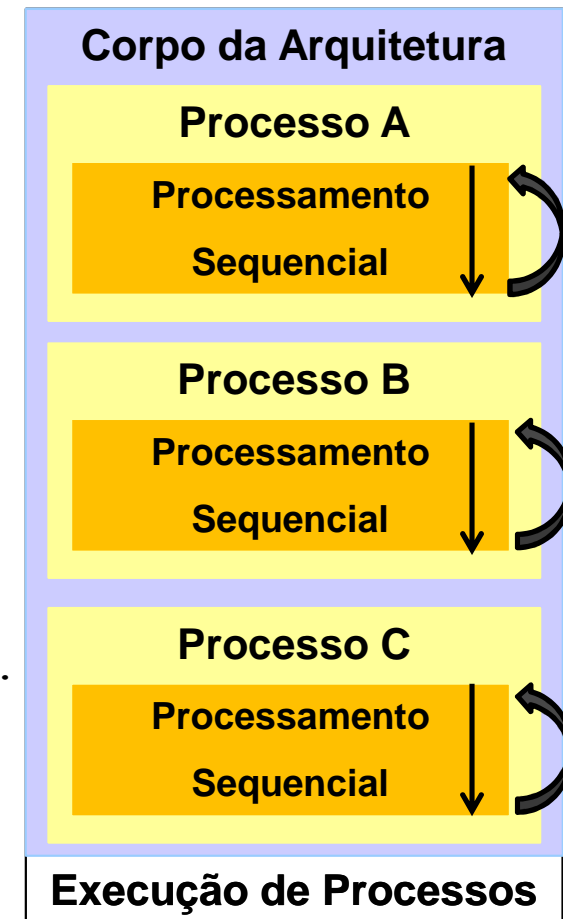
- Atribuição de valores a sinais;
- Atribuição de valores a variáveis;
- Instruções sequenciais;
- Chamada de subprogramas (funções e procedimentos).



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 da sua atribuição, podendo os seus valores serem modificados diversas vezes durante a execução do processo.

- 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 várias atribuições de valores a um mesmo sinal, apenas a última atribuição terá efeito, sendo ignoradas todas as atribuições precedentes.

Ex.: PROCESS
 BEGIN
 → VAR := '0';
 → SINAL_X <= VAR;
 → VAR := '1';
 → SINAL_Y <= VAR;
 → ...
 END PROCESS;

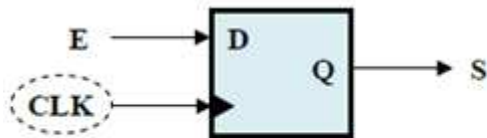
Ex.: PROCESS
 BEGIN
 X <= A OR B; -- atribuicao ignorada
 Y <= X;
 X <= A AND C; -- atribuicao que anula a precedente
 Z <= X; -- resultado: Z = Y = X = A AND C
 END PROCESS;

VHDL - Processos

Lista de sensibilidade

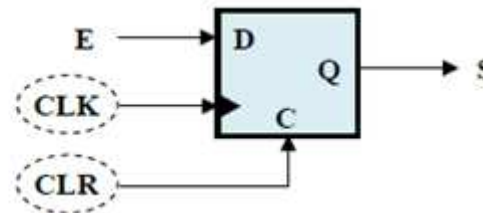
- Um processo pode opcionalmente conter uma lista de sensibilidades, que consiste numa lista de sinais ao qual o processo é sensível.
- Esta lista de sensibilidades estabelece quando é que o processo deve ser reavaliado (executado).

Ex: Processo síncrono



```
PROCESS(CLK)
BEGIN
    IF (CLK'EVENT) AND (CLK='1')
        THEN S <= E;
    END IF;
END PROCESS;
```

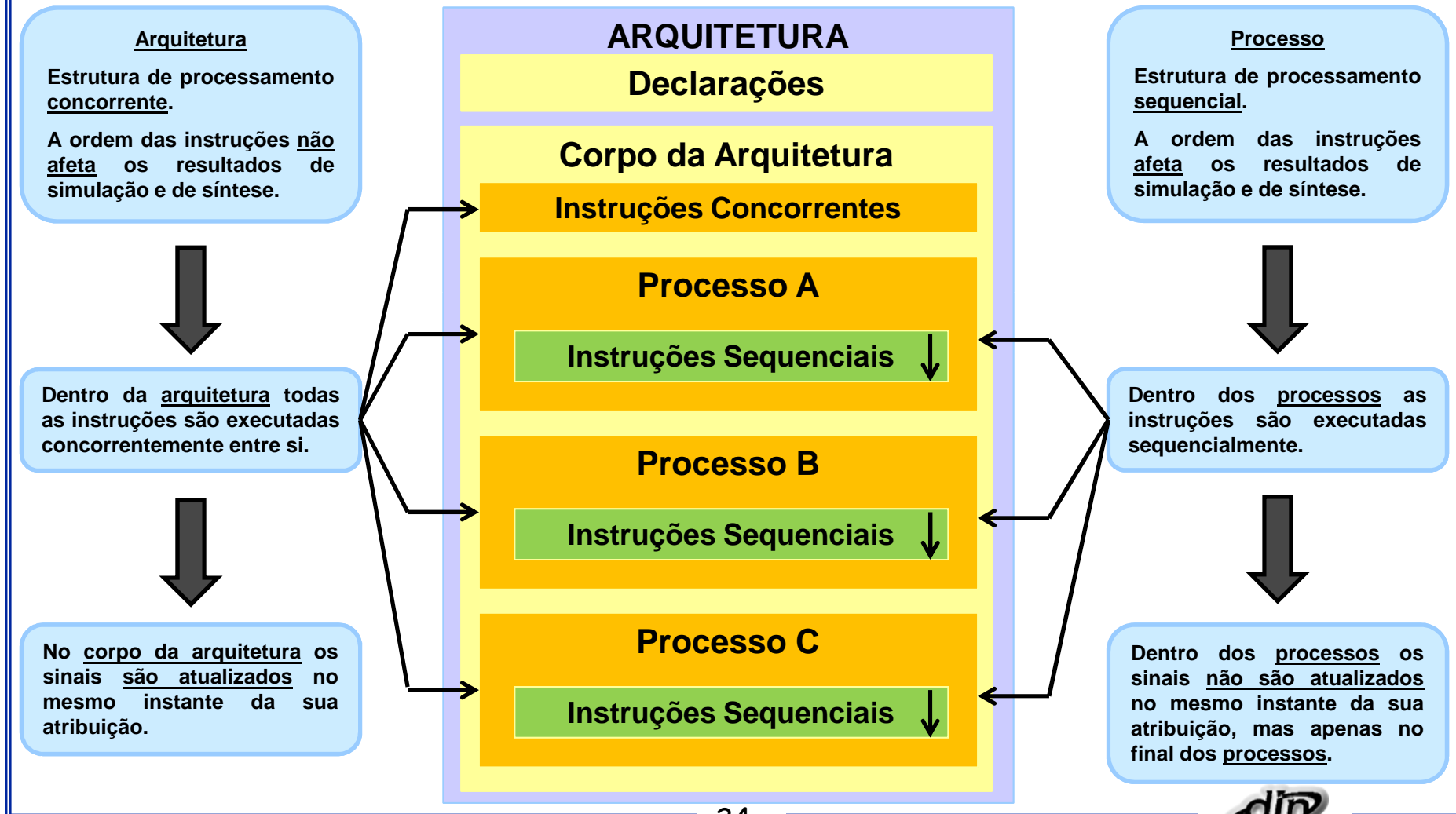
Ex: Processo síncrono com *clear* assíncrono



```
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

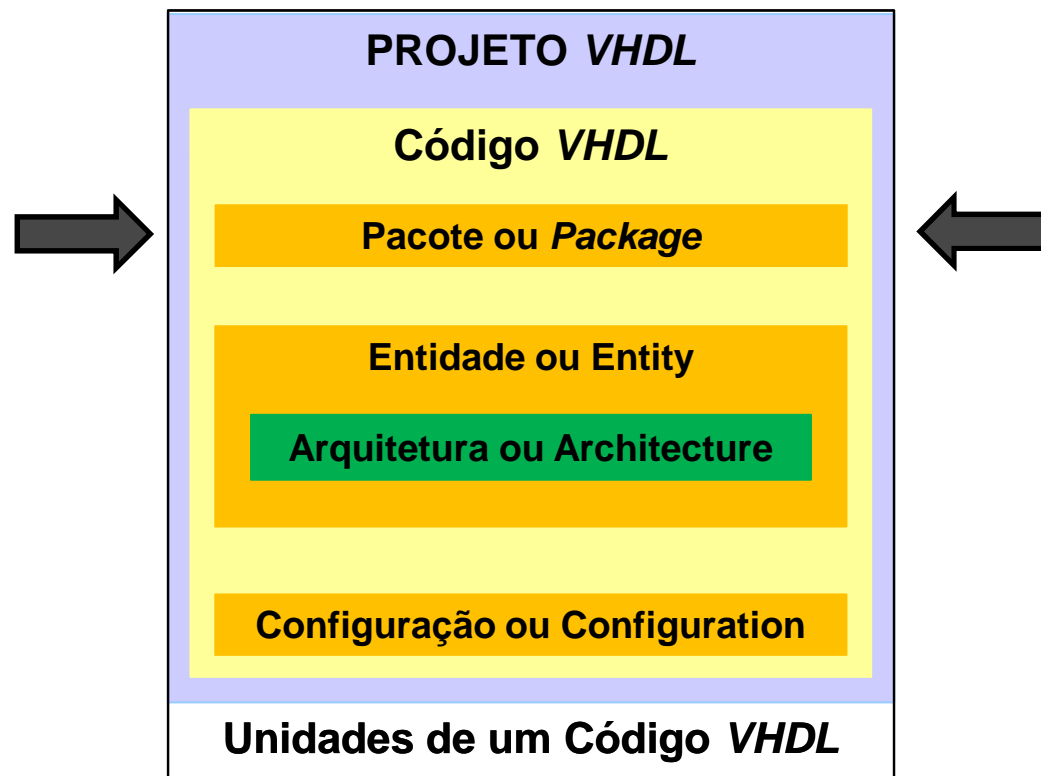
Processamento sequencial versus processamento concorrente



VHDL - Pacotes

Pacotes ou *packages*

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



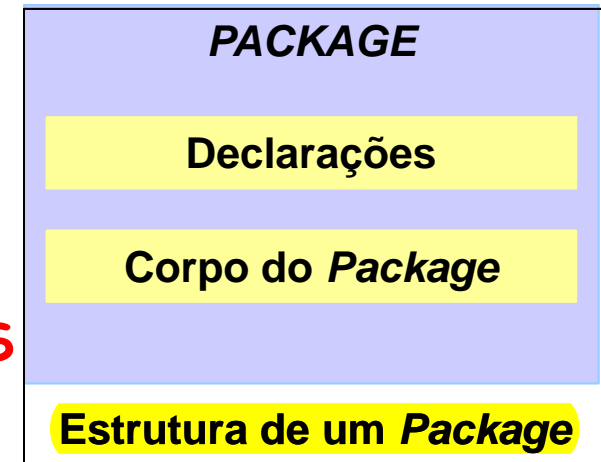
VHDL - Pacotes

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 OF 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*;

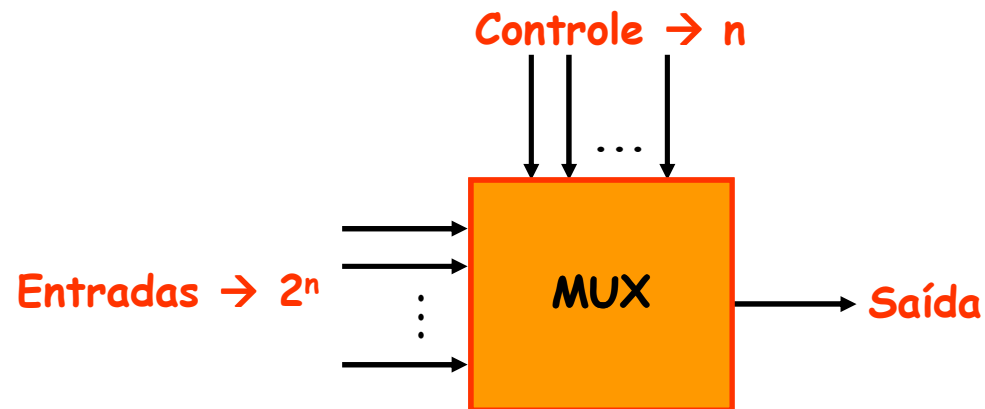
VHDL - Pacotes

Exemplo:

Multiplexador

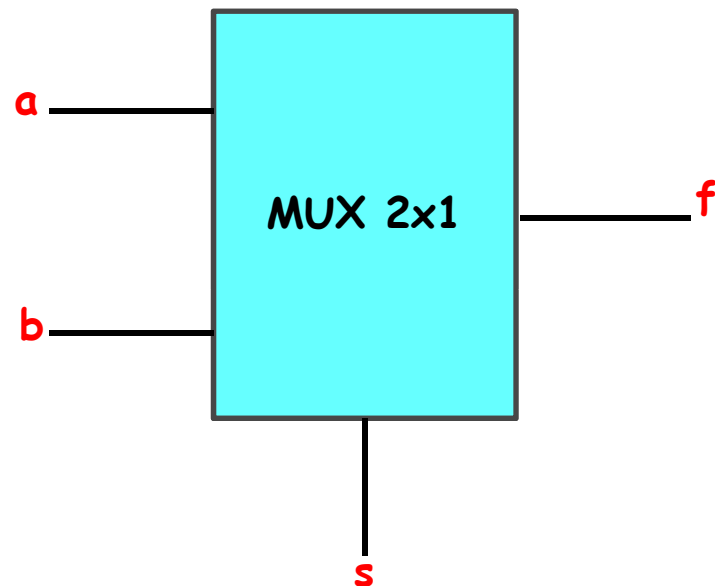
Multiplexador ou Seletor de Dados: É um circuito lógico que tem diversas entradas e apenas uma saída. MUX seleciona uma única entrada para transmitir para a saída.

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



VHDL - Pacotes

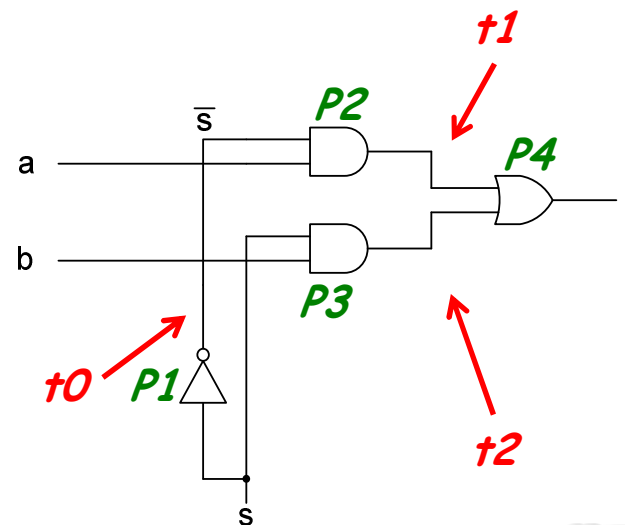
Exemplo: Multiplexador 2 X 1



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

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 and_2  
        PORT(x : IN BIT;  
             y : IN BIT;  
             z : OUT BIT);  
    END COMPONENT;  
  
    -- continuacao
```

```
    COMPONENT or_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

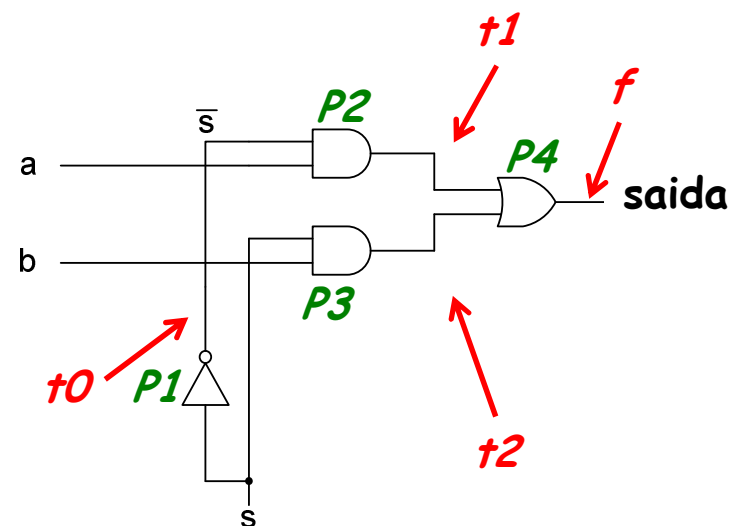
VHDL - Atrasos

ATRASOS

○ Exemplo MUX 2x1

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


Resumo da Aula de Hoje

Tópicos mais importantes:

- Processos
- Pacotes
- Atrasos

Próxima da Aula

- **Subprogramação**
 - Funções
 - Procedimentos