# 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 Condicionais**
    - Comando *WHEN ELSE*
    - Comando *IF THEN ELSE*
    - Comando *CASE WHEN*
- **Comandos de Repetição**
  - **Comando *FOR LOOP***
  - **Comando *WHILE LOOP***
  - **Comando *NEXT* e *EXIT***

# Revisão

- Comandos Condicionais
  - Comando *WHEN ELSE*
  - Comando *IF THEN ELSE*
  - Comando *CASE WHEN*

# VHDL – Comandos Condicionais

## Comandos Condicionais

- Comandos Condicionais permitem alterar o fluxo de execução do código
- Em VHDL há 3 comandos condicionais:
  - WHEN ELSE
  - IF THEN ELSE
  - CASE WHEN

# VHDL – Comandos Condicionais

## WHEN ELSE

- É um comando concorrente
- Restrição de uso dentro de procedimentos, funções e processos
- Transfere o valor de uma expressão para um sinal destino caso uma determinada condição seja satisfeita

### Sintaxe

```
sinal_destino <= expressao_1 WHEN condiçao_1 ELSE
                 expressao_2 WHEN condiçao_2 ELSE
                 expressao_3 WHEN condiçao_3 ELSE
                 expressao_4;
```

- Nota: O comando condicional WHEN ELSE é útil para expressar funções lógicas em forma de tabela verdade

# VHDL – Comandos Condicionais

## WHEN ELSE

- Exemplo 01:
- Implemente a função XNOR com o comando condicional WHEN ELSE

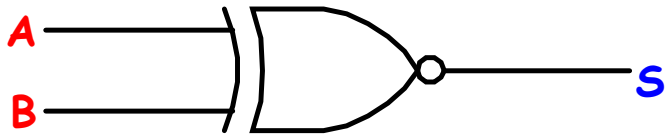
**TV da Porta XNOR**

Entradas | Saída

| A | B | S |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Função XNOR Representação: $S = \overline{A \oplus B} = A \odot B$

**Símbolo da Porta XNOR**




**Condições**

A variável de saída S será igual a 1 quando as variáveis de entrada A e B forem iguais, senão a variável de saída S será igual a 0.

# VHDL – Comandos Condicionais

## WHEN ELSE

- Exemplo 01:
- Implemente a função XNOR com o comando condicional WHEN ELSE

**TV da Porta XNOR**

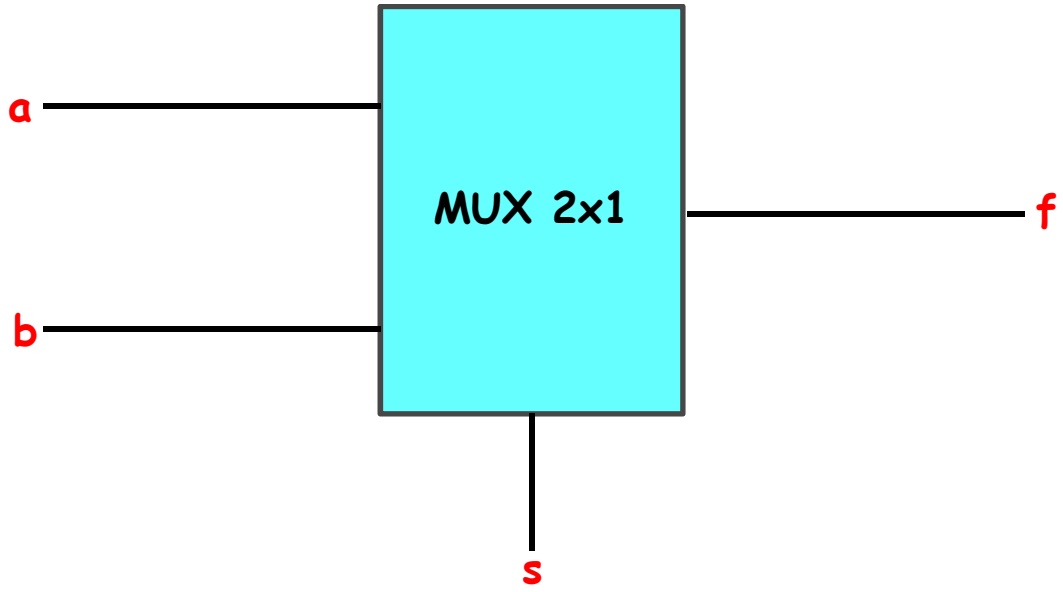
| Entradas | | Saída |
|---|---|---|
| A | B | S |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

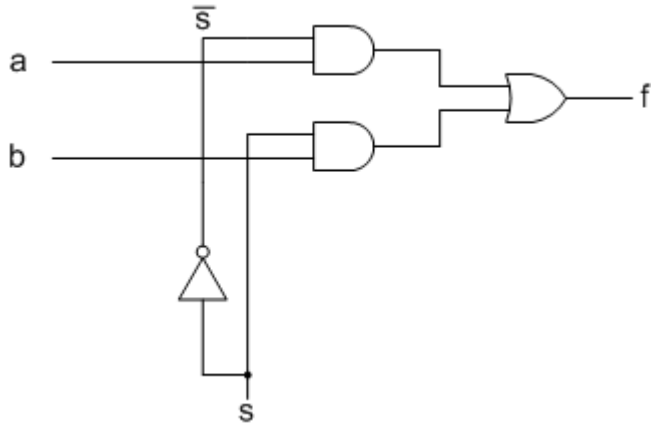
```
LIBRARY ieee;                -- Funcao xnor usando comando when else
USE ieee.std_logic_1164.all;
ENTITY xnor_cc_when IS
            PORT (a, b : IN BIT;
                        s  : OUT BIT);
END xnor_cc_when;
ARCHITECTURE condicional OF xnor_cc_when IS
BEGIN
     s <= '1' WHEN (a=b) ELSE '0';
END condicional;
```

# VHDL – Comandos Condicionais

## WHEN ELSE

- Exemplo 02: Implemente um multiplexador de duas entradas de dados (a e b), uma entrada de seleção (s) e uma única saída de dados (f). O seu funcionamento basear-se-á na escolha da entrada de seleção que determinará qual das entradas de dados aparecerá na saída de dados.



# VHDL – Comandos Condicionais

**Solução**

```
LIBRARY ieee;          -- Multiplexador 2 x 1 usando comando when else
USE ieee.std_logic_1164.all;
ENTITY mux_cc_when IS
     PORT (a, b : IN BIT;
                s  : IN BIT;
                f  : OUT BIT);
END mux_cc_when;
ARCHITECTURE condicional OF mux_cc_when IS
BEGIN
    f <= a WHEN s = '0' ELSE b;
END condicional;
```

# VHDL – Comandos Condicionais

## IF THEN ELSE

- É um comando sequencial
- Utilizado na descrição comportamental de componentes → Utilizado em procedimentos, funções e processos.
- Transfere o valor de uma expressão para um sinal destino caso uma determinada condição seja satisfeita
- Sintaxe →

```
IF condicao_1 THEN
        comando_sequencial;
ELSIF condicao_2 THEN                    -- Clausula ELSIF opcional
        comando_sequencial;
ELSIF condicao_3 THEN
        comando_sequencial;
ELSE                                     -- Clausula ELSE opcional
        comando_sequencial;
END IF;
```

# VHDL – Comandos Condicionais

## IF THEN ELSE

- Aninhar vários níveis de contruções IF THEN ELSE.
- Sintaxe →

```
IF condicao_1 THEN
        IF condicao_2 THEN
                comando_sequencial;
        ELSE
                comando_sequencial;
        END IF;
ELSE
        IF condicao_3 THEN
                comando_sequencial;
        ELSE
                comando_sequencial;
        END IF;
END IF;
```

# VHDL – Comandos Condicionais

## IF THEN ELSE

- Exemplo 03:
- Implemente a função XNOR com o comando condicional IF THEN ELSE

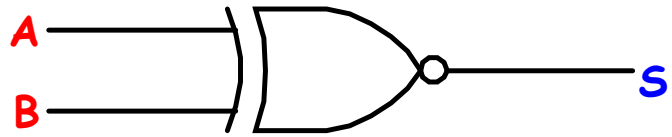
**TV da Porta XNOR**

Entradas | Saída

| A | B | S |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Função XNOR Representação: $S = \overline{A \oplus B} = A \odot B$

**Símbolo da Porta XNOR**




**Condições**

Se as variáveis de entrada A e B forem iguais então a variável de saída S será igual a 1, senão a variável de saída S será igual a 0.

# VHDL – Comandos Condicionais

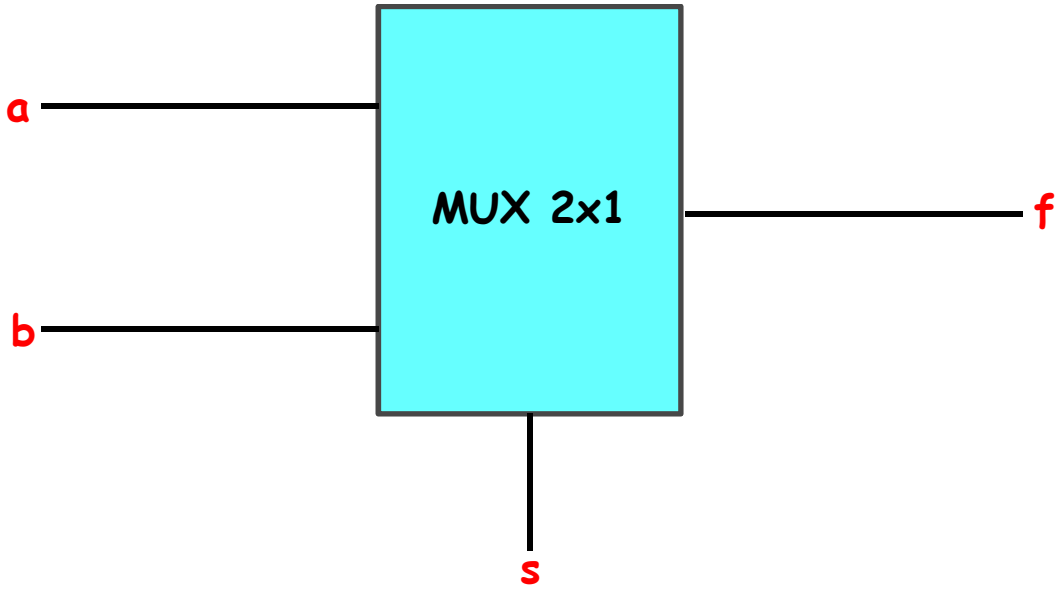
**Solução**

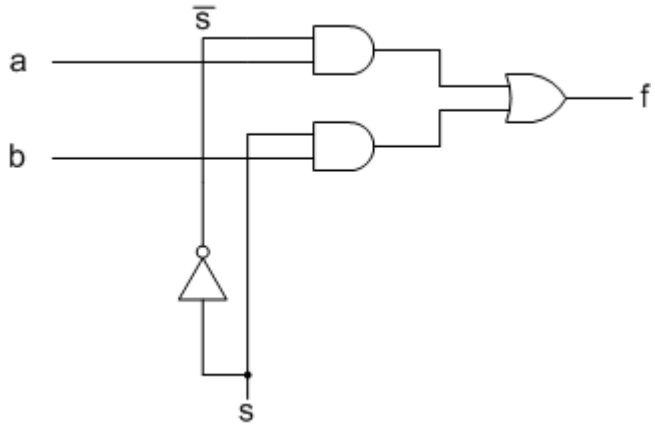
```
LIBRARY ieee;          -- Funcao xnor usando comando if then else end if
USE ieee.std_logic_1164.all;
ENTITY xnor_cc_if IS
          PORT (a, b : IN BIT;
                    s  : OUT BIT);
END xnor_cc_if;
ARCHITECTURE condicional OF xnor_cc_if IS
BEGIN
      PROCESS (a, b)
      BEGIN
            IF (a=b) THEN s <= '1';
            ELSE
               s <= '0';
            END IF;
     END PROCESS;
END condicional;
```

# VHDL – Comandos Condicionais

## IF THEN ELSE

- Exemplo 04: Implemente um multiplexador de duas entradas de dados (a e b), uma entrada de seleção (s) e uma única saída de dados (f). O seu funcionamento basear-se-á na escolha da entrada de seleção que determinará qual das entradas de dados aparecerá na saída de dados.



# VHDL – Comandos Condicionais

**Solução**

```
LIBRARY ieee;                                   -- Multiplexador 2 x 1 usando comando if then else
USE ieee.std_logic_1164.all;
ENTITY mux_cc_if IS
     PORT (a, b : IN BIT;
                s  : IN BIT;
                f  : OUT BIT);
END mux_cc_if;
ARCHITECTURE condicional OF mux_cc_if IS
BEGIN
    PROCESS (a, b, s)
    BEGIN
          IF s='0' THEN f<=a;
          ELSE f<=b;
          END IF;
     END PROCESS;
END condicional;
```

# VHDL – Comandos Condicionais

## CASE WHEN

- É um comando sequencial com uso dentro de procedimentos, funções e processos.
- Permite a definição de várias condições em um componente.
- Neste comando, as comparações sempre são feitas em torno de um único objeto ou expressão, e será o valor desse objeto ou determinada condição que indicará quais comandos serão executados.
- Sintaxe:

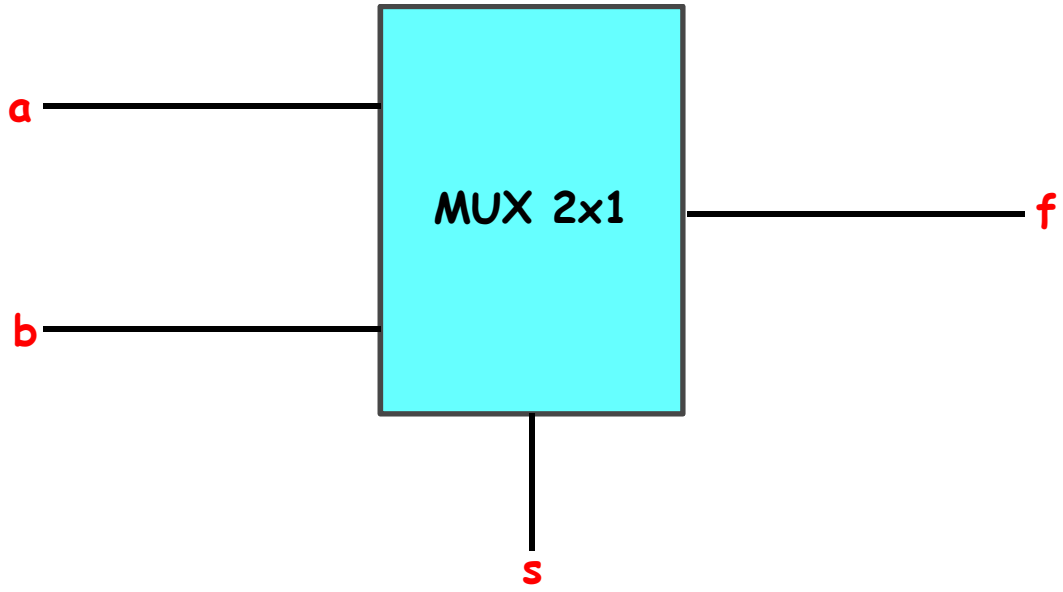
```
CASE expressao_de_escolha IS                          -- expressao_de_escolha =
   WHEN condicao_1                  => comando_a;             -- condicao_1
   WHEN condicao_2                  => comando_b; comando_c;-- condicao_2
   WHEN condicao_3 | condicao_4     => comando_d;             -- condicao_3 ou condicao_4
   WHEN condicao_5 TO condicao_9    => comando_d;             -- condicao_5 ate condicao_9
   WHEN OTHERS                      => comando_e; comando_f;-- condicoes restantes
END CASE;
```

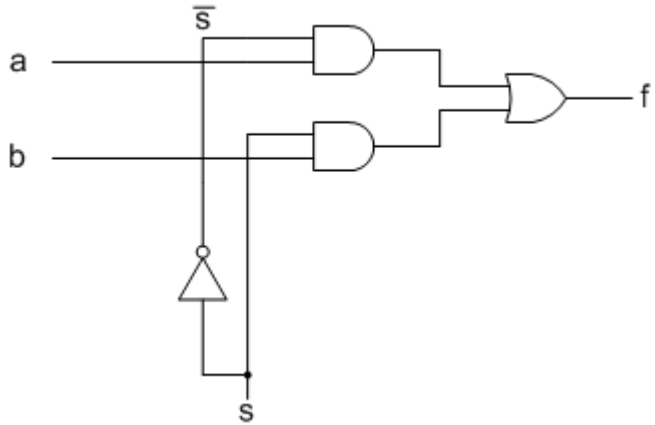
- NOTA: O delimitador | equivale a uma operação OU entre as condições de escolha. As palavras reservadas TO e DOWNTO servem para delimitar uma faixa de condições. A palavra reservada OTHERS na última condição serve para agrupar as condições não-relacionadas na lista.

# VHDL – Comandos Condicionais

## CASE WHEN

- Exemplo 05: Implemente um multiplexador de duas entradas de dados (a e b), uma entrada de seleção (s) e uma única saída de dados (f). O seu funcionamento basear-se-á na escolha da entrada de seleção que determinará qual das entradas de dados aparecerá na saída de dados.



# VHDL – Comandos Condicionais

```
LIBRARY ieee;          -- Multiplexador 2 x 1 usando comando case when
USE ieee.std_logic_1164.all;
ENTITY mux_cc_case IS
    PORT (a, b : IN BIT;
          s  : IN BIT;
          f  : OUT BIT);
END mux_cc_case;
ARCHITECTURE condicional OF mux_cc_case IS
BEGIN
        PROCESS (a, b, s)
        BEGIN
                CASE s IS
                        WHEN '0' => f <= a;
                        WHEN '1' => f <= b;
                END CASE;
        END PROCESS;
END condicional;
```

# VHDL – Comandos Condicionais

## WITH SELECT WHEN

- É um comando concorrente.
- Transfere um valor a um sinal de destino segundo uma relação de opções.
- Todas as condições de seleção devem ser consideradas e elas devem ser mutuamente exclusivas.
- A lista de opções nesta construção não contém uma prioridade.

**Sintaxe:**

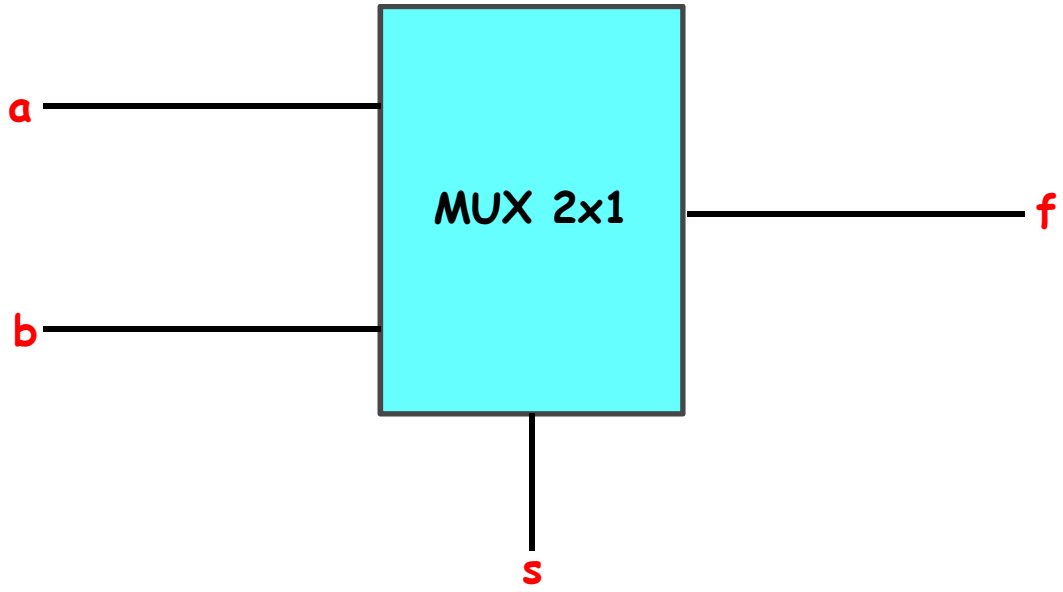
```
WITH expressao_de_escolha SELECT                    -- expressao_de_escolha =
  sinal_destino <= expressao_a WHEN condicao_1,     -- condicao_1
                   expressao_b WHEN condicao_2,     -- condicao_2
                   expressao_c WHEN condicao_3 | condicao_4, -- condicao_3 ou condicao_4
                   expressao_d WHEN condicao_5 TO condicao_9, -- condicao_5 ate condicao_9
                   expressao_e WHEN OTHERS;         -- condicoes restantes
```

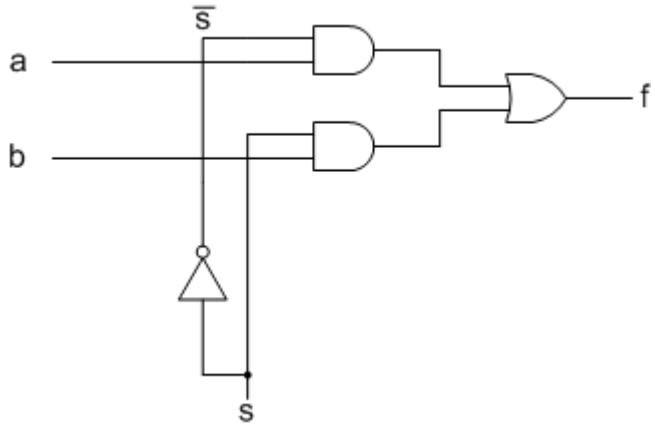
- NOTA: O delimitador | equivale a uma operação OU entre as condições de escolha. As palavras reservadas TO e DOWNTO servem para delimitar uma faixa de condições. A palavra reservada OTHERS na última condição serve para agrupar as condições não-relacionadas na lista.

# VHDL – Comandos Condicionais

## WITH SELECT WHEN

- **Exemplo 06:** Implemente um multiplexador de duas entradas de dados (a e b), uma entrada de seleção (s) e uma única saída de dados (f). O seu funcionamento basear-se-á na escolha da entrada de seleção que determinará qual das entradas de dados aparecerá na saída de dados.



# VHDL – Comandos Condicionais

```
LIBRARY ieee; -- Multiplexador 2 x 1 usando comando with select when
USE ieee.std_logic_1164.all;
ENTITY mux_cc_case IS
    PORT (a, b : IN BIT;
          s  : IN BIT;
          f  : OUT BIT);
END mux_cc_case;
ARCHITECTURE condicional OF mux_cc_case IS
BEGIN
        WITH s SELECT
                f <= a WHEN '0',
                     b WHEN '1';
END condicional;
```

# Aula de Hoje

- 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;
```

# VHDL – Comandos de Repetição

## FOR LOOP

- Exemplo:

```
--calcula o quadrado de 1 a 10 e armazena em i_squared
FOR i IN 1 TO 10 LOOP
        i_squared(i) := i * i;
END LOOP;
```

# VHDL – Comandos de Repetição

## FOR LOOP

- **Exercício 01:**
- **Implemente um código para um circuito gerador de paridade para 4 bits:**

  - ✓ **Se dígitos "1" no número binário for ímpar → Paridade ímpar → Paridade = 1**

  - ✓ **Se dígitos "1" no número binário for par → Paridade par → Paridade = 0**

# VHDL – Comandos de Repetição

**Solução**

- **Exercício 01:**

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


ENTITY gerador_paridade IS
PORT   (a: IN BIT_VECTOR (0 TO 3);
          paridade: OUT BIT);
END gerador_paridade;                    -- continua
```

# VHDL – Comandos de Repetição

**Solução**

- **Exercício 01:** Atributo: a'RANGE[n] → Faixa do vetor da dimensão "[n]"

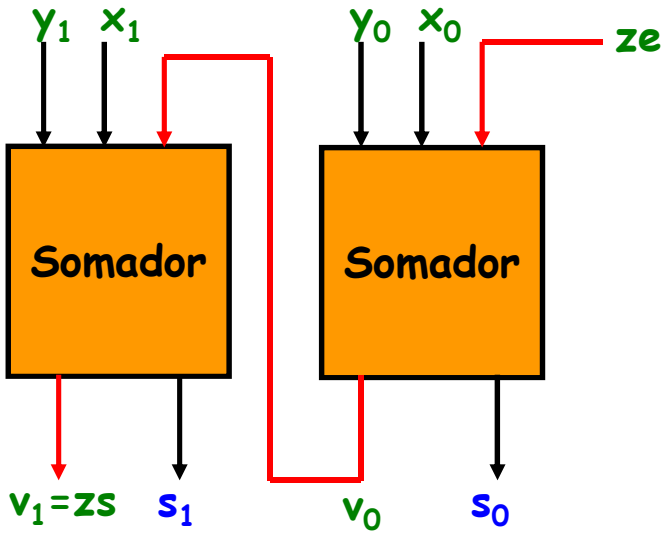
```
ARCHITECTURE logica OF gerador_paridade IS        -- continuação
BEGIN
        PROCESS (a)
        VARIABLE par_temp: BIT;
        BEGIN
        par_temp:='0';
                FOR i IN a'RANGE LOOP        -- a'RANGE → 0 TO 3
                        par_temp:=par_temp XOR a(i);
                END LOOP;
        paridade <= par_temp;
        END PROCESS;
END logica;
```

# 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.**
  - **Veja figura a seguir**

# Somador de 2 bits



# 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 DOWNTO 0);     -- entradas do somador
            ze      : IN  BIT;                           -- vem-1
            s       : OUT BIT_VECTOR (n-1 DOWNTO 0);     -- saida
            zs      : OUT BIT);                          -- vai-1
END som_2bits;

ARCHITECTURE logica OF som_2bits IS
BEGIN
            PROCESS (x, y, ze)
            VARIABLE v : BIT_VECTOR (n DOWNTO 0);  -- vai-1 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

## FOR LOOP

- **Exercício 03:**
- **Implemente um código para o circuito conversor do tipo bit_vector para o tipo integer (binário para inteiro) usando FOR LOOP.**
- **Use os seguintes nomes para as entradas e saídas:**
  - **e_bit para o vetor de bits (com 4 posições);**
  - **inteiro_for para a saída convertida para inteiro;**
  - **temp para a variável que armazena as conversões.**

# VHDL – Comandos de Repetição

## Solução

- **Exercício 03:** Atributo: e_bit'RANGE[n] → Faixa do vetor da dimensão "[n]"

```
ENTITY loop_f1 IS
  PORT (e_bit          : IN  BIT_VECTOR (3 DOWNTO 0); -- entrada tipo vetor de bits
        inteiro_for    : OUT INTEGER RANGE 0 TO 15);  -- saida convertida para inteiro
END loop_f1;

ARCHITECTURE teste OF loop_f1 IS
BEGIN
          PROCESS (e_bit)
          VARIABLE temp : INTEGER;
          BEGIN
                    temp := 0;
                    FOR i IN e_bit'RANGE LOOP  -- numero de iteracoes: tamanho do vetor entrada
                              IF e_bit(i) = '1' THEN temp := temp + 2**i;
                              END IF;
                    END LOOP;
                    inteiro_for <= temp;
          END PROCESS;
END teste;
```

# 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

- Exemplo:

```
j := 0;
sum := 10;
WHILE j < 20 LOOP
        sum := sum * 2;
        j := j + 3;
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 DOWNTO 0);         -- entradas do somador
          ze          : IN  BIT;                               -- vem um
          s           : OUT BIT_VECTOR (n-1 DOWNTO 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 DOWNTO 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

## WHILE LOOP

- **Exercício 05:**
- **Implemente um código para o circuito conversor do tipo bit_vector para o tipo integer (binário para inteiro) usando WHILE LOOP.**
- **Use os seguintes nomes para as entradas e saídas:**
  - **e_bit para o vetor de bits (com 4 posições);**
  - **inteiro_for para a saída convertida para inteiro;**
  - **temp para a variável que armazena as conversões.**

# VHDL – Comandos de Repetição

**Solução**

**Exercício 05:**

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

ENTITY bit_to_int_while IS
  GENERIC (n            : INTEGER := 4);
  PORT    (e_bit        : IN  BIT_VECTOR (n-1 DOWNTO 0);      -- entrada tipo vetor de bits
           inteiro_while : OUT INTEGER RANGE 0 TO 2**n-1);     -- saida convertida para inteiro
END bit_to_int_while;                                         --continua
```

# VHDL – Comandos de Repetição

Solução

**Exercício 05:** Atributo: e_bit'LENGTH[n] → Número de elementos da dimensão "[n]" no vetor

```
ARCHITECTURE logica OF bit_to_int_while IS                    --continuacao
BEGIN
  PROCESS (e_bit)
  VARIABLE temp, i : INTEGER;
  BEGIN
    temp := 0;
    i := 0;
    WHILE (i /= e_bit'LENGTH) LOOP          -- numero de iteracoes: tamanho do vetor entrada
      IF e_bit(i) = '1' THEN temp := temp + 2**i;
      END IF;
      i := i + 1;
    END LOOP;
    inteiro_while <= temp;
  END PROCESS;
END logica;
```

# 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

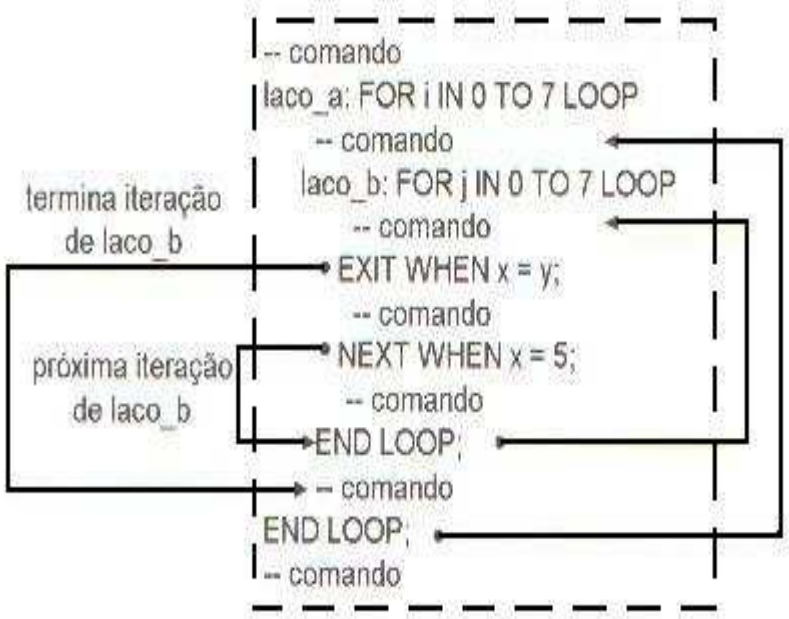
## EXIT

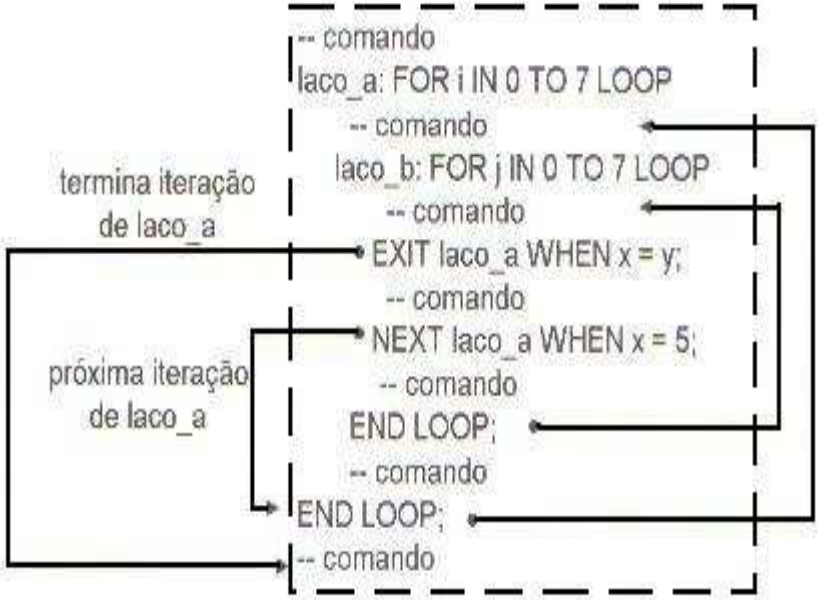
- Exemplo:

```
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;
```

# VHDL – Comandos de Repetição

## NEXT e EXIT



# VHDL – Comandos de Repetição

## EXIT

- **Exercício 06:**
- **Implemente um código para o circuito conversor do tipo bit_vector para o tipo integer (binário para inteiro) usando EXIT num LOOP básico.**
- **Use os seguintes nomes para as entradas e saídas:**
  - **e_bit para o vetor de bits (com 4 posições);**
  - **inteiro_exit para a saída convertida para inteiro;**
  - **temp para a variável que armazena as conversões.**

# VHDL – Comandos de Repetição

## Solução

- **Exercício 06:** Atributo: e_bit'LENGTH[n] → Número de elementos da dimensão "[n]" no vetor

```
ENTITY loop_e1 IS
  GENERIC (n           : INTEGER := 4);
  PORT (e_bit          : IN  BIT_VECTOR (n-1 DOWNTO 0); -- entrada tipo vetor de bits
        inteiro_exit   : OUT INTEGER RANGE 0 TO 2**n-1);-- saida convertida para inteiro
END loop_e1;

ARCHITECTURE teste OF loop_e1 IS
BEGIN
  PROCESS (e_bit)
    VARIABLE temp, i : INTEGER;
  BEGIN
    temp := 0;
    i := 0;
    LOOP
      IF e_bit(i) = '1' THEN temp := temp + 2**i;
      END IF;
      i := i +1;
      EXIT WHEN i = e_bit'LENGTH;
    END LOOP;
    inteiro_exit <= temp;
  END PROCESS;
END teste;
```

# Tarefas para a Casa

**Implementar os projetos a seguir:**

- 2 variações do somador
- 1 variação para paridade

# Tarefas para a Casa

## Somador Variação 1

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;
ENTITY soma_cc_for2 IS
  GENERIC (n          : INTEGER  := 2);                          -- numero de bits
  PORT     (A, B      : IN  STD_LOGIC_VECTOR (n-1 DOWNTO 0); -- entradas do somador
                  cin : IN  STD_LOGIC;                           -- vem-1
                    S : OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0); -- saida
                 cout : OUT STD_LOGIC);                          -- vai-1
END soma_cc_for2;                                               --continua
```

# Tarefas para a Casa

## Somador Variação 1

```
architecture somador of soma_cc_for2 is                    --continuacao
begin
        process(A,B, cin)
        variable carry : STD_LOGIC;
        begin
                for w in 0 to 1 loop
                        if w=0 then
                                carry:=cin;
                        end if;
                        S(w) <= A(w) xor B(w) xor carry;
                        carry := (A(w) and B(w)) or (A(w) and carry) or (B(w) and carry);
                end loop;
                cout <= carry;
        end process;
end somador;
```

# Tarefas para a Casa

## Somador Variação 2

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;

entity soma_cc_for3 is
            port (a,b: in std_logic_vector(1 downto 0);
                        cin: in std_logic;
                        sum: out std_logic_vector(1 downto 0);
                       cout: out std_logic);
end soma_cc_for3;                                                    --continua
```

# Tarefas para a Casa

## Somador Variação 2

```
architecture behavior of soma_cc_for3 is                    --continuacao
signal c: std_logic_vector(2 downto 0);
begin
        process (a,b,cin,c)
        begin
                c(0) <= cin;
                for i in 0 to 1 loop
                        sum(i) <= a(i) xor b(i) xor c(i);
                        c(i+1) <= (a(i) and b(i)) or (c(i) and (a(i) or b(i)));
                end loop;
                cout <= c(2);
        end process;
end behavior;
```

# Tarefas para a Casa

## Gerador Variação 1

```
library IEEE;
use IEEE.Std_Logic_1164.all;
ENTITY paridade_for_1 IS
  PORT     (I            : IN  STD_LOGIC_VECTOR (0 TO 3);
            EVEN, ODD : OUT STD_LOGIC);
END paridade_for_1;                                          --continua
```

# Tarefas para a Casa

## Gerador Variação 1

- ✓ Convenção:
  - ➢ Paridade será ímpar quando ODD = 1 e EVEN = 0
  - ➢ Paridade será par quando ODD = 0 e EVEN = 1

```
ARCHITECTURE teste OF paridade_for_1 IS                    --continuacao
BEGIN
        PROCESS (I)
        VARIABLE p : STD_LOGIC;
        BEGIN
                p := I(0);
                FOR j IN 1 TO 3 LOOP
                        IF I(j) = '1' THEN p := NOT p; END IF;
                END LOOP;
                ODD  <= p;
                EVEN <= NOT p;
        END PROCESS;
END teste;
```

# Resumo da Aula de Hoje

**Tópicos mais importantes:**

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

# Próxima da Aula

- Atrasos
- Processos
- Pacotes