

# Introdução à Linguagem VHDL

Período 2021/1

Roberto G. Pacheco

[pacheco@gta.ufrj.br](mailto:pacheco@gta.ufrj.br)

**Grupo de Teleinformática e Automação – GTA/UFRJ**  
**Programa de Engenharia Elétrica - PEE/COPPE/UFRJ**  
**Universidade Federal do Rio de Janeiro**

- Volnei A. Pedroni, "Circuit Design and Simulation with VHDL", The MIT Press, 2nd Ed.

- VHDL significa **VHSIC** (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage (**Linguagem de Descrição de Hardware**)
  - Linguagem de Descrição de Hardware vs Linguagem de Programação
  - Desenvolvida pelo Departamento de Defesa dos EUA na década de 80 e a primeira a ser padronizada pelo IEEE
  - FPGA (Field Programmable Field Array)
    - Uma placa composta de portas lógicas e elementos de memória (FFs) organizadas em formato matricial

- O código em VHDL permite descrever em software o comportamento e a estrutura do hardware de qualquer sistema digital
- Síntese
  - tradução do código em uma estrutura de hardware (**FPGA**) para implementar as funcionalidades descritas no código
    - Processo de transferir o código à FPGA
- Simulação
  - Processo de testar para garantir que o circuito sintetizado alcança as funcionalidades programadas no código
  - O simulador utilizado será o **Quartus II Lite**

# Projeto em VHDL

- Análise teórica
- Desenvolvimento do código em VHDL
- Simulação
- Síntese

- O código em VHDL não é **case-sensitive**
  - **VHDL não diferencia letras maiúsculas e minúsculas**
    - SignalIN é o mesmo que signalin
  - Geralmente, letras maiúsculas são reservadas para comandos reservados
    - AND, OR, XOR, LIBRARY, USE
- Comentários
  - Iniciados por "--"
  - Terminados pelo fim da linha, conseqüentemente, fim do comentário
  - Os relatórios **DEVEM** ser todos comentados
  - Boa prática disponibilizar os códigos desenvolvidos em um repositório, como github

- Os códigos em VHDL possuem comandos:
  - Concorrentes:
    - Executados todos de forma paralela
  - Sequenciais:
    - Executados em sequência, assemelhando-se mais a execução de linguagem de programação

- Atribuição de sinais
  - $A \leq B$ 
    - Cuidado com o pensamento de programação
    - $A \leq B; A \leq C; \quad \Rightarrow$  implica curto-circuito
    - São Fios e entradas físicas
- Comparação
  - ">", "=", "<"
- Operações Booleanas
  - AND, OR, NOT, XOR;



# Comandos básicos

- Declarações Concorrentes
  - a
- Declarações Sequenciais
  - a

# Estrutura Básica do Código em VHDL



- Declaração de Bibliotecas
- Entity
- Architecture

# Declaração de Bibliotecas

- Declaração de Bibliotecas
  - A declaração de biblioteca é uma lista de bibliotecas e pacotes que o compilador precisa para processar o projeto, ou seja, o circuito digital descrito no código
- Exemplo:
  - LIBRARY ieee;
  - USE ieee.std\_logic\_1164.all;

# Declaração de Bibliotecas

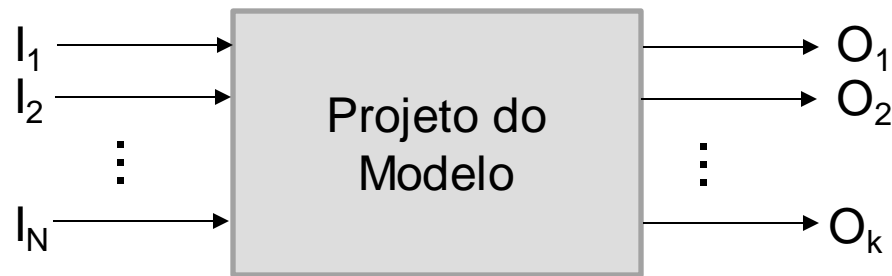
- Declaração de Bibliotecas
  - A declaração de biblioteca é uma lista de bibliotecas e pacotes que o compilador precisa para processar o projeto, ou seja, o circuito digital descrito no código
- Exemplo:
  - LIBRARY ieee;
  - USE ieee.std\_logic\_1164.all;

Note o ";" que indica  
término de comando

- Declaração de Bibliotecas
  - A declaração de biblioteca é uma lista de bibliotecas e pacotes que o compilador precisa para processar o projeto, ou seja, o circuito digital descrito no código
- Exemplo:
  - LIBRARY ieee;
  - USE ieee.std\_logic\_1164.all;
    - Geralmente essa biblioteca é importada para permitir utilizar tipos lógicos, como STD\_LOGIC

- Entity é uma lista de entradas e saídas do modelo do sistema digital projetado.
  - Descreve a interface de entradas e saídas de modelo

- Entity é uma lista de entradas e saídas do modelo do sistema digital projetado.
  - Descreve a interface de entradas e saídas de modelo



- Sintaxe

ENTITY entity\_name IS

    GENERIC (constant\_name: constant\_type:= constant\_value;  
            ...);

    PORT(port\_name : signal\_mode signal\_type;  
          ...)



- Sintaxe

**ENTITY** entity\_name IS

```
  GENERIC(constant_name: constant_type:= constant_value;  
          ...);
```

```
  PORT(port_name : signal_mode signal_type;  
        ...)
```

- **entity\_name**: nome de identificação da entity/modelo
- **GENERIC**: define uma constante genérica que pode ser usada e alterada em qualquer momento, inclusive dentro de PORT
  - GENERIC (number\_of\_bits: INTEGER := 16)
- **PORT**: declara a interface I/O

- Sintaxe de PORT

```
PORT(port_name : signal_mode signal_type;  
    ...)
```

- Cada sinal de entrada ou saída possui um modo (**IN, OUT, BUFFER, INOUT**) e um tipo de sinal (**BIT, BIT\_VECTOR, STD\_LOGIC, STD\_LOGIC\_VECTOR**)

# Modos do Sinal PORT

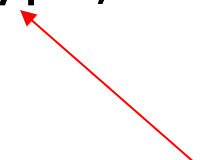
- **IN:** dados de **entrada** que fluem para **dentro** da **ENTIDADE**.
  - Não pode escrever nesses sinais
  - Ex.: clock, dados de entrada, dados de controle, entrada **unidirecionais** de dados
- **OUT:** dados de **saída** que fluem para **fora** da **ENTIDADE**
  - Os dados em modo **OUT** são usados quando não serão lidos pela ENTIDADE
- **BUFFER:** dados de **saída** que fluem para fora da **ENTIDADE**, mas também podem ser lidas. Na prática, esses dados são armazenadas em elemento de memória (**FF**)
  - Buffer pode ser utilizado para realimentação interna
  - Contudo, o BUFFER não pode ser usado como dados de entrada

# Modos do Sinal PORT

- **INOUT**: dados que podem fluir para **dentro e fora** da ENTIDADE
  - Evitem utilizar esse modo
    - Aumenta a complexidade do projeto
    - Reduz a interpretabilidade do código
    - Dificulta a detecção de erros
  - Devem ser utilizados apenas para **barramento bidirecionais**

- Sintaxe

```
ENTITY entity_name IS
  GENERIC (constant_name: constant_type:= constant_value;
           ...);
  PORT(port_name : signal_mode signal_type;
       ...)
```



# Tipos de sinais em VHDL

- BIT, BIT\_VECTOR:
  - Representa nível lógico de sinais de entrada ou saída
  - Valores: '0' ou '1'
  - Tipos **nativos** do VHDL, logo não necessita de declaração de bibliotecas
- STD\_LOGIC, STD\_LOGIC\_VECTOR:
  - Representa nível lógico de sinais de entrada ou saída
  - Valores: '0', '1', '-' (**don't case**), 'Z' (**alta impedância**)
  - Esse tipo de sinais necessita das seguintes bibliotecas:
    - LIBRARY ieee;
    - USE ieee.std\_logic\_1164.all;

# Tipos de sinais em VHDL

- BIT, BIT\_VECTOR:
  - Representa nível lógico de sinais de entrada ou saída
  - Valores: '0' ou '1'
  - Tipos **nativos** do VHDL, logo não necessita de declaração de bibliotecas
- STD\_LOGIC, STD\_LOGIC\_VECTOR:
  - Representa nível lógico de sinais de entrada ou saída
  - Valores: '0', '1', '-' (**don't case**), 'Z' (**alta impedância**)
  - Esse tipo de sinais necessita das seguintes bibliotecas:
    - LIBRARY ieee;
    - USE ieee.std\_logic\_1164.all;

# Tipos de sinais em VHDL

- **CONSTANT:**
  - Atribuição de valores constantes
  - Sintaxe: `constant_name: constant_type := constant_value`
    - Ex.: `n_bits: INTEGER := 16`
- **SIGNAL:**
  - Tipo de sinais que transmitem **dados de dentro para fora** da entidade, ou entre componentes internos do circuito, atuando como **fios**.
  - Funciona para **"armazenar variáveis temporárias"**
- **VARIABLE:**
  - Armazena valores na **parte sequencial** do VHDL



# Constant vs Generic

## Constant

- Sintaxe
  - `CONSTANT n: INTEGER := 8;`
- Atribuição de um valor constante
  - Define-se até na Architecture (corpo do projeto)

## Generic

- Sintaxe
  - `GENERIC(n: INTEGER :=8);`
- Atribuição de um valor constante semelhante a Constant
  - Define-se na entity
  - Pode ser alterado em qualquer momento
  - Pode ser alterado e mapeado como componente

# Constant vs Generic

## Constant

- Sintaxe
  - `CONSTANT n: INTEGER := 8;`
- Atribuição de um valor constante
  - Define-se até na Architecture (corpo do projeto)

## Generic

- Sintaxe
  - `GENERIC(n: INTEGER :=8);`
- Atribuição de um valor constante semelhante a Constant
  - Define-se na entity
  - Pode ser alterado em qualquer momento
  - **Pode ser alterado e mapeado como componente**

# Tipos de sinais em VHDL

Predefined data types	Library / Package of origin	Synthesizable values without restrictions
BIT, BIT_VECTOR	std / standard	'0', '1'
BOOLEAN	std / standard	TRUE, FALSE
INTEGER	std / standard	$-(2^{31}-1)$ to $+(2^{31}-1)$
NATURAL	std / standard	0 to $+(2^{31}-1)$
POSITIVE	std / standard	1 to $+(2^{31}-1)$
CHARACTER	std / standard	256-symbol alphabet (1 byte/symbol)
STRING	std / standard	Set of characters
REAL	std / standard	Little or no synthesis support
STD_(U)LOGIC, STD_(U)LOGIC_VECTOR	ieee / std_logic_1164	Input: '0' or 'L', '1' or 'H' Output: '0' or 'L', '1' or 'H', '-' or 'X', and 'Z'
UNSIGNED, SIGNED	ieee / numeric_std, ( ) / std_logic_arith	Same as STD_LOGIC

# Exemplos – Contador de 4 bits

```
ENTITY counter_4 IS PORT (  
    clk, reset, load_counter:    IN BIT;  
    data:                        IN BIT_VECTOR(3 DOWNT0 0);  
    count_zero:                  OUT BIT;  
    counter:                      BUFFER BIT_VECTOR(3 DOWNT0 0)  
);  
END counter_4;
```

# Exemplos – Contador de 4 bits

```
ENTITY counter_4 IS PORT (  
    clk, reset, load_counter: IN BIT;  
    data: IN BIT_VECTOR(3 DOWNT0 0);  
    count_zero: OUT BIT;  
    counter: BUFFER BIT_VECTOR(3 DOWNT0 0)  
);  
END counter_4;
```

A última não precisa do ";"

# Exemplos – Contador de 4 bits

```
ENTITY counter_4 IS PORT (  
    clk, reset, load_counter:    IN BIT;  
    data:                        IN BIT_VECTOR(3 DOWNT0 0);  
    count_zero:                  OUT BIT;  
    counter:                      BUFFER BIT_VECTOR(3 DOWNT0 0)  
);  
END counter_4;
```



END entity\_name;

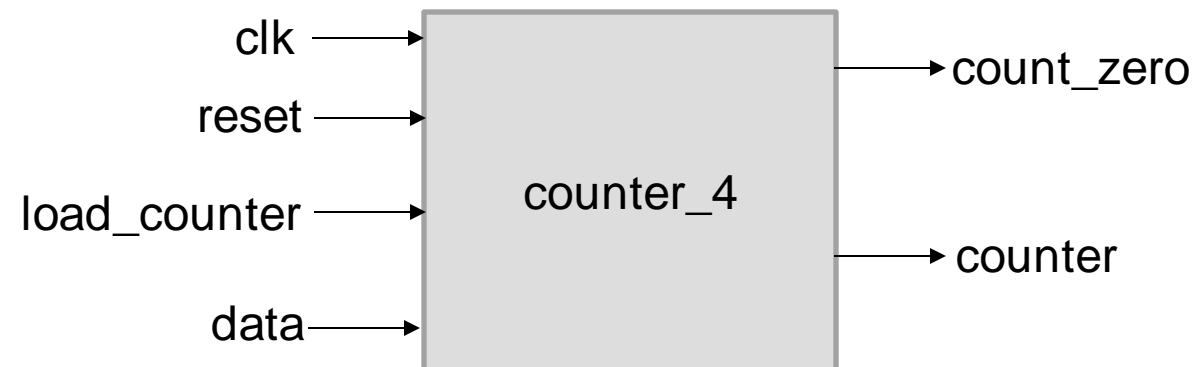
# Exemplos – Contador de 4 bits

```
ENTITY counter_4 IS PORT (  
    clk, reset, load_counter:    IN BIT;  
    data:                        IN BIT_VECTOR(3 DOWNTO 0);  
    count_zero:                  OUT BIT;  
    counter:                      BUFFER BIT_VECTOR(3 DOWNTO 0)  
);  
END counter_4;
```

**CUIDADO:** não misture BIT com STD\_LOGIC ou BIT\_VECTOR com STD\_LOGIC\_VECTOR

# Exemplos – Contador de 4 bits

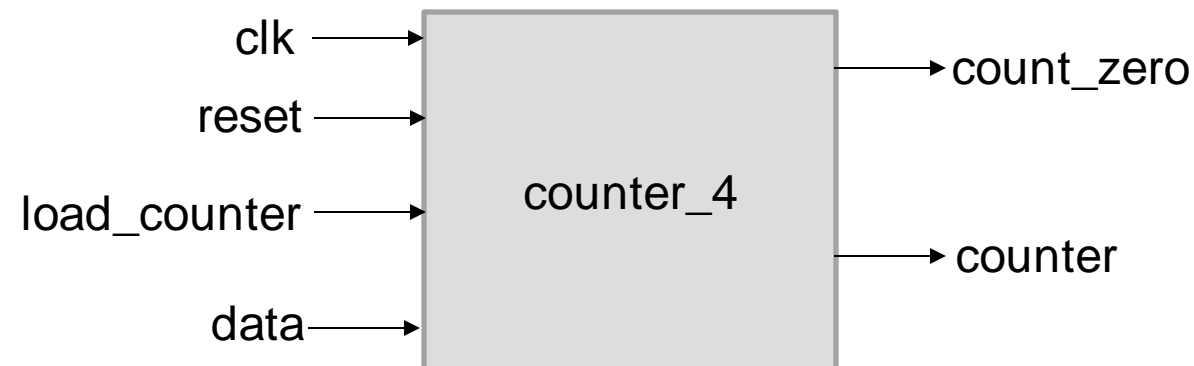
```
ENTITY counter_4 IS PORT (  
    clk, reset, load_counter:    IN BIT;  
    data:                        IN BIT_VECTOR(3 DOWNT0 0);  
    count_zero:                  OUT BIT;  
    counter:                     BUFFER BIT_VECTOR(3 DOWNT0 0)  
);  
END counter_4;
```



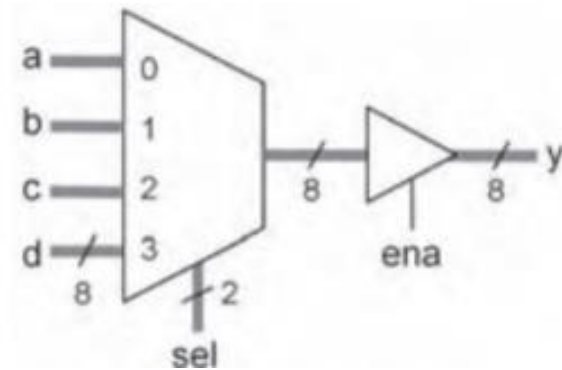


# Exemplos – Contador de 4 bits

```
ENTITY counter_4 IS PORT (  
  clk, reset, load_counter:  IN BIT;  
  data:                       IN BIT_VECTOR(3 DOWNT0 0);  
  count_zero:                 OUT BIT;  
  counter:                   BUFFER BIT_VECTOR(3 DOWNT0 0)  
);  
END counter_4;
```



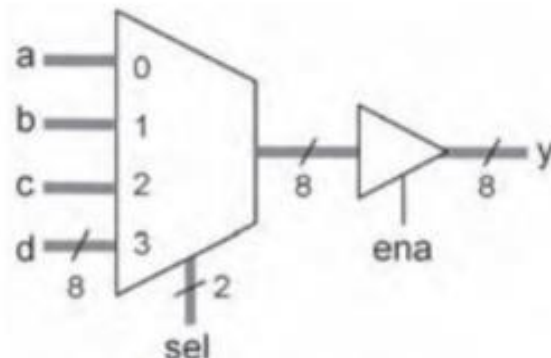
# Exemplos – Buffered MUX 8 bits



ena	sel	y
0	X	"ZZZZZZZZ"
1	0	a
	1	b
	2	c
	3	d

# Exemplos – Buffered MUX 8 bits

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY buffered_mux IS PORT (  
    a, b, c, d:          IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
    sel:                IN STD_LOGIC_VECTOR(1 DOWNTO 0);  
    ena:                INT BIT;  
    y:                  OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  
);  
END buffered_mux;
```

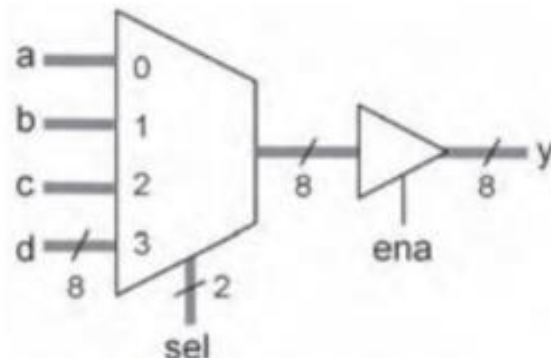


ena	sel	y
0	X	"ZZZZZZZZ"
1	0	a
	1	b
	2	c
	3	d

# Exemplos – Buffered MUX 8 bits

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY buffered_mux IS PORT (  
    a, b, c, d:          IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
    sel:                IN STD_LOGIC_VECTOR(1 DOWNTO 0);  
    ena:                INT BIT;  
    y:                  OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  
);  
END buffered_mux;
```

Por que não usar BIT\_VECTOR?

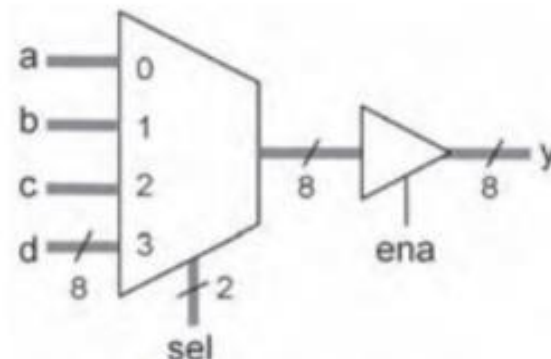


ena	sel	y
0	X	"ZZZZZZZZ"
1	0	a
	1	b
	2	c
	3	d

# Exemplos – Buffered MUX 8 bits

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY buffered_mux IS PORT (  
    a, b, c, d:          IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
    sel:                IN STD_LOGIC_VECTOR(1 DOWNTO 0);  
    ena:                INT BIT;  
    y:                  OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  
);  
END buffered_mux;
```

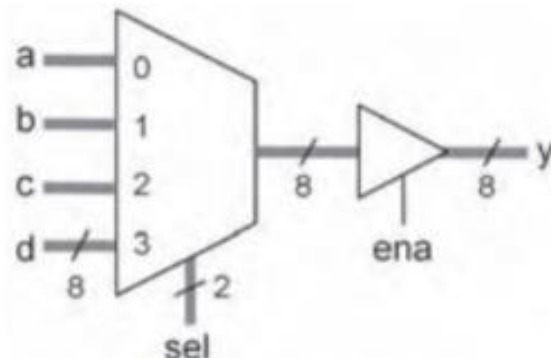
Note a necessidade da alta impedância na tabela-verdade



ena	sel	y
0	X	"ZZZZZZZZ"
1	0	a
	1	b
	2	c
	3	d

# Exemplos – Buffered MUX 8 bits

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
ENTITY buffered_mux IS PORT (  
    a, b, c, d:          IN STD_LOGIC_VECTOR (7 DOWNTO 0);  
    sel:                IN INTEGER RANGE 0 TO 3;  
    ena:                INT BIT;  
    y:                  OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  
);  
END buffered_mux;
```



ena	sel	y
0	X	"ZZZZZZZZ"
1	0	a
	1	b
	2	c
	3	d

- BIT\_VECTOR:
  - Declaração:
    - bit\_signal\_vector: BIT\_VECTOR (max\_index DOWNT0 0);
    - bit\_vector1, bit\_vector2 : BIT\_VECTOR (3 DOWNT0 0);
    - bit\_standalone: BIT;
- Formas de atribuição
  - bit\_vector1(0) <= '1';
  - bit\_vector1(0) <= bit\_standalone;
  - bit\_vector1(0) <= bit\_vector2(3);
  - bit\_vector1 <= "0101"

- BIT\_VECTOR:

- Declaração:

- bit\_signal\_vector: BIT\_VECTOR (max\_index DOWNT0 0);
    - bit\_vector1, bit\_vector2: BIT\_VECTOR (3 DOWNT0 0);
    - bit\_standalone: BIT;

- Formas de atribuição

- bit\_vector1(0) <= '1';
  - bit\_vector1(0) <= bit\_standalone;
  - bit\_vector1(0) <= bit\_vector2(3);
  - bit\_vector1 <= "0101"

Aspas simples para atribuir 1 bit

Aspas dupla para atribuir conjunto de bits



- Operadores de Atribuição ("`<=`", "`:=`")
  - "`<=`" usado para atribuir valores à sinais
  - "`:=`" usado para atribuir valores à constantes, variables
- Operadores de concatenação ("`&`", "`,`")
  - `k: CONSTANT BIT_VECTOR(1 TO 4) := "1100";`
  - `x <= ('Z', k(2 TO 3), "11111");`
    - `x <= 'Z1011111'`
  - `x <= 'Z' & k(2 TO 3) & "11111";`
    - `x <= 'Z1011111'`



- Operadores de deslocamento (**SLL, SRL, SLA, SRA, ROL, ROR**)
  - **SLL (shift left logical)**: Data are shifted to the left with '0's in the empty positions
  - **SRL (shift right logical)**: Data are shifted to the right with '0's in the empty positions
  - **SLA (shift left arithmetic)**: Data are shifted to the left with the rightmost bit in the empty positions
  - **SRA (shift right arithmetic)**: Data are shifted to the right with the leftmost bit in the empty positions
  - **ROL (rotate left)**: Circular shift to the left
  - **ROR (rotate right)**: Circular shift to the right

- Operadores de deslocamento (**SLL, SRL, SLA, SRA, ROL, ROR**)

- Exemplos:

- `a <= "11001";`
- `x <= a SLL 2;`
- `y <= a SLA 2;`
- `z <= a SLL -3;`

-- result: `x <= "00100"`

--result: `y <= "00111"`

--result: `z <= "00011"`

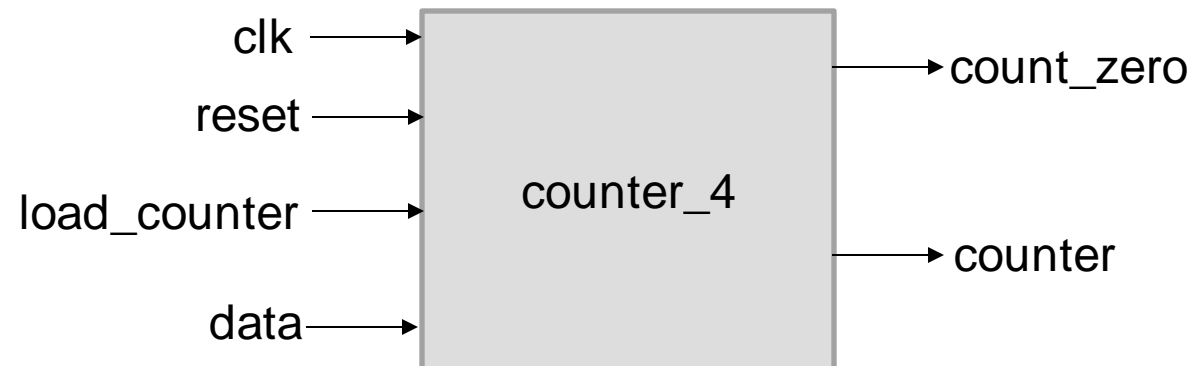
- Operadores de Comparação (=, /=, >, <, >=, <=)

- Exemplos:

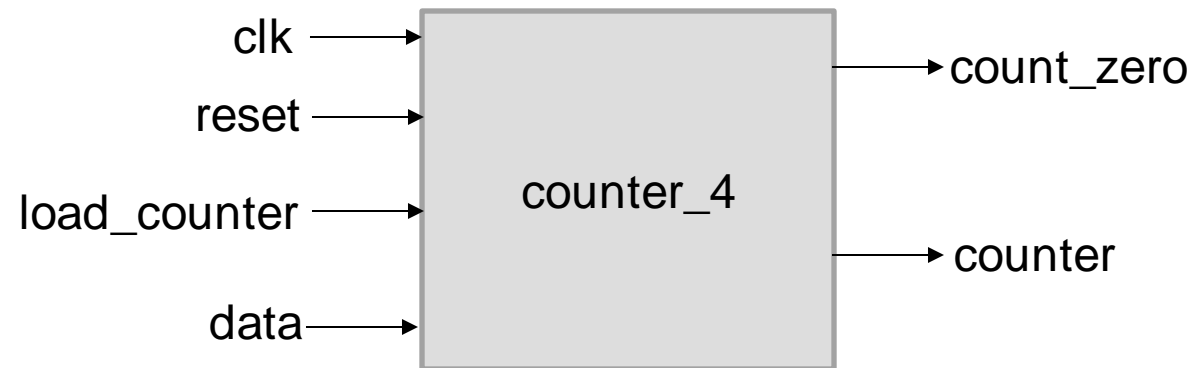
- IF a >= b THEN x <= '1';

Note "<=" pode significar atribuição de sinal, mas também pode significar comparação, dentro do contexto de IF THEN

- Descreve o comportamento e as funcionalidades do sistema
- Enquanto a Entity descreve a interface



- Descreve o comportamento e as funcionalidades do sistema
- Enquanto a Entity descreve a interface
  - Architecture descreve o funcionamento do interior da entidade



# Exemplo Completo – Maioria de 3

- Exercício:
  - Projete um circuito que apresenta saída ativa em nível alto, quando a **maioria** das suas três entradas estiverem em nível alto.



# Exemplo Completo – Maioria de 3

- Exercício:
  - Projete um circuito que apresenta saída ativa em nível alto, quando a **maioria** das suas três entradas estiverem em nível alto.
    - Entradas:
      - a, b, c
    - Saída
      - y
  - Após fazer o projeto de circuito combinacional, obtém-se:
    - $y \leq (a \text{ AND } b) \text{ OR } (a \text{ AND } c) \text{ OR } (b \text{ AND } c);$

# Exemplo Completo – Maioria de 3

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
Entity majority_3 IS PORT(
    a, b, c : IN STD_LOGIC;
    y:      OUT STD_LOGIC
);
end majority_3;
ARCHITECTURE arq_maj_3 OF majority_3 IS
BEGIN
    y <= (a AND b) OR (a AND c) OR (b AND c);
END arq_maj_3;
```

Declaração de bibliotecas

Bloco de Entity

Bloco de Architecture

# Exemplo Completo – Half Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY half_adder IS PORT (
    a, b:      IN  STD_LOGIC;
    sum, carry_out:  OUT STD_LOGIC
);
end half_adder;
ARCHITECTURE HA of half_adder IS
BEGIN
    sum <= a XOR b;
    carry_out <= a AND b;
END HA;
```

Declaração de bibliotecas

Bloco de Entity

Bloco de Architecture

# Exemplo Completo – Half Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY half_adder IS PORT (
    a, b:      IN  STD_LOGIC;
    sum, carry_out:  OUT STD_LOGIC
);
end half_adder;
ARCHITECTURE HA of half_adder IS
BEGIN
    sum <= a XOR b;
    carry_out <= a AND b;
END HA;
```

Declaração de bibliotecas

Bloco de Entity

Bloco de Architecture

COMANDOS  
CONCORRENTES

# Exemplo Completo – Full Adder

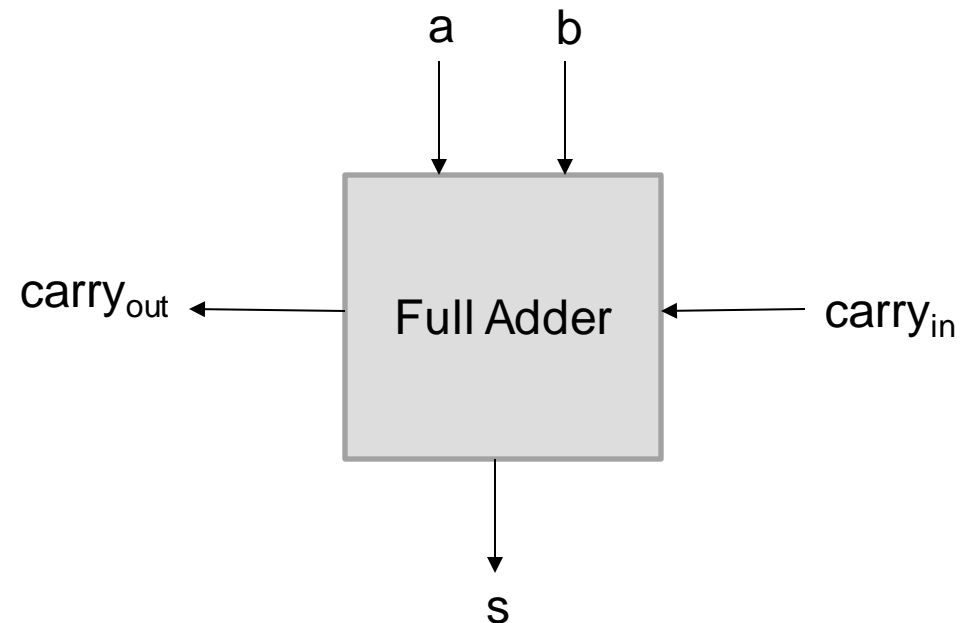
```
ENTITY full_adder IS
    PORT(a, b, carry_in:    IN  BIT;
         sum, carry_out:    OUT BIT;
END full_adder;
```

# Exemplo Completo – Full Adder

Projetar um circuito somador completo

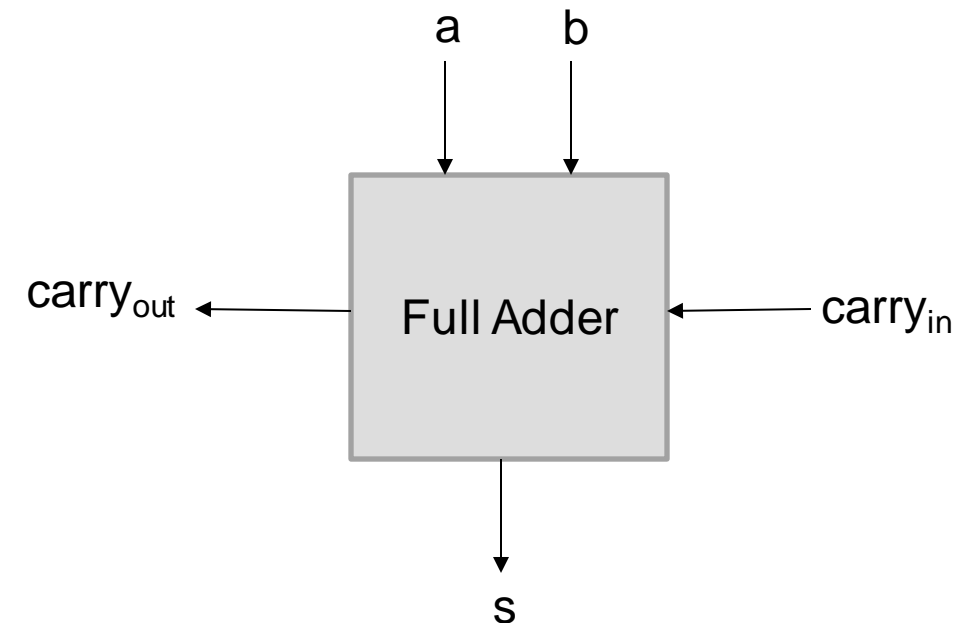
# Exemplo Completo – Full Adder

Projetar um circuito somador completo (full adder) de 1 bit



# Entity do Full Adder

```
ENTITY full_adder IS
    PORT(a, b, c_in:    IN BIT;
         s, c_out:     OUT BIT);
END full_adder;
```





# Tabela-Verdade do Full Adder

# Tipos de sinais em VHDL

A implementação da Architecture pode ser realizada de diversos modos conforme o desenvolvedor. Destaca-se três tipos:

- Fluxo de dados (dataflow)
- Estrutural (structural)
- Comportamental (behaviorial)

# Tipos de sinais em VHDL

A implementação da Architecture pode ser realizada de diversos modos conforme o desenvolvedor. Destaca-se três tipos:

- **Fluxo de dados (dataflow)**
- Estrutural (structural)
- Comportamental (behaviorial)

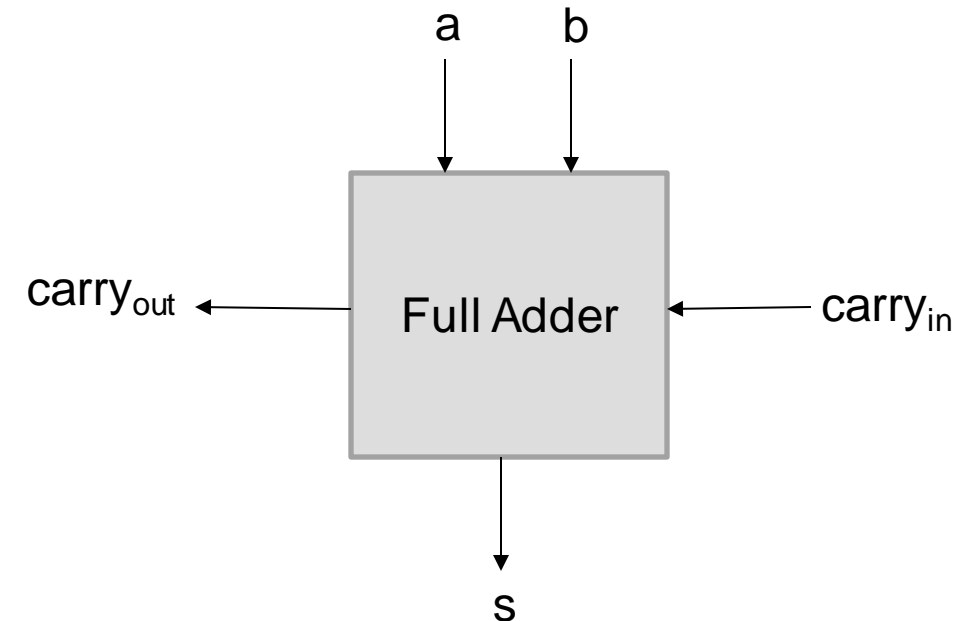
# Architecture do Full Adder: Dataflow

- Descreve o fluxo de dados do sistema
- As saídas são atribuídas diretamente, por meio de portas lógicas.
- Todas as expressões são **concorrentes** no tempo, ou seja, as atribuições ocorrem **simultaneamente**

# Architecture do Full Adder: Dataflow

```
ENTITY full_adder IS
    PORT(a, b, c_in:    IN BIT;
         s, c_out:     OUT BIT);
END full_adder;
```

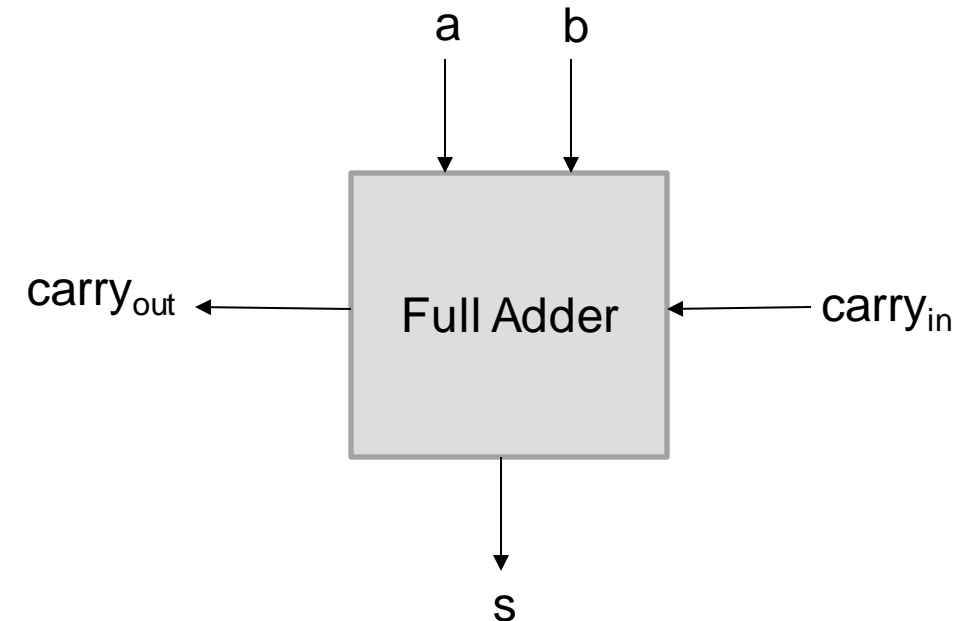
```
ARCHITECTURE dataflow OF full_adder IS
    SIGNAL x1, x2, x3, x4: BIT;
    BEGIN
        x1<= b XOR c_in;
        s<= a XOR x1;
        x2 <= a AND b;
        x3 <= a and c_in;
        x4 <= b AND c_in;
        c_out <= x2 OR x3 OR x4;
    END dataflow;
```



# Architecture do Full Adder: Dataflow

```
ENTITY full_adder IS
    PORT(a, b, c_in:    IN BIT;
         s, c_out:     OUT BIT);
END full_adder;

ARCHITECTURE dataflow OF full_adder IS
BEGIN
    s <= a XOR b XOR c_in;
    c_out <= (a AND b) OR (a AND c_in) OR (b AND c_in);
END dataflow;
```



# Architecture do Full Adder: Dataflow

Eliminando os sinais temporários SIGNAL

```
ENTITY full_adder IS
    PORT(a, b, c_in:    IN BIT;
         s, c_out:     OUT BIT);
END full_adder;

ARCHITECTURE dataflow OF full_adder IS
BEGIN
    s<= a XOR b XOR c_in;
    c_out <= (a AND b) OR (a AND c_in) OR (b AND c_in);
END dataflow;
```

# Architecture: Dataflow

Além disso, é possível descrever uma arquitetura dataflow usando comandos condicionais WHEN - ELSE

- **Sintaxe:**

- assignment WHEN conditions ELSE

- **Exemplo:**

```
x <= a WHEN sel=0 ELSE  
b WHEN sel=1 ELSE  
c;
```

← Caso default



Além disso, é possível descrever uma arquitetura dataflow usando comandos condicionais WHEN - ELSE

- **Sintaxe:**

- assignment WHEN conditions ELSE

- **Exemplo:**

```
x<= "00" WHEN (a=b)
ELSE "01" WHEN (a=c)      -- a !=b
ELSE "10" WHEN (a=b)     -- a != b and a!=c
ELSE "11";               --default case
```

# Exemplo

Projete um circuito que obtenha saída em nível alto quando as duas entradas forem iguais

# Exemplo

Projete um circuito que obtenha saída em nível alto quando as duas entradas forem iguais

- Tabela-Verdade:

a	b	
0	0	1
0	1	0
1	0	0
1	1	1

# Exemplo

Projete um circuito que obtenha saída em nível alto quando as duas entradas forem iguais  
Vamos usar a estrutura de condicional **WHEN-ELSE**

```
ENTITY detect_equal IS
    PORT(a, b:  IN BIT;
         y: OUT BIT);
END detect_equal;
ARCHITECTURE dataflow OF detect_equal IS
BEGIN
    y<= '1' WHEN a=b ELSE
        '0';
END dataflow;
```

- Tabela-Verdade:

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

# Exemplo

Projete um circuito que obtenha saída em nível alto quando as duas entradas forem iguais  
Vamos usar a estrutura de condicional **WHEN-ELSE**

```
ENTITY detect_equal IS
    PORT(a, b:  IN BIT;
         y: OUT BIT);
END detect_equal;
ARCHITECTURE dataflow OF detect_equal IS
BEGIN
    y<= a XNOR b;
END dataflow;
```

- Tabela-Verdade:

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

# Tipos de sinais em VHDL

A implementação da Architecture pode ser realizada de diversos modos conforme o desenvolvedor. Destaca-se três tipos:

- Fluxo de dados (dataflow)
- **Estrutural (structural)**
- Comportamental (behaviorial)

# Arquitetura Estrutural

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Exemplo: Detector de Overflow:

O **Overflow** ocorre sempre que o resultado da adição está fora da faixa de números inteiros representáveis. Quando ocorre, o resultado está **INCORRETO**, de forma que é necessário detectar essa situação de overflow



Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Exemplo: Detector de Overflow:

Considere  $z = x + y$ , onde  $x = x_{n-1} \dots x_0$ ,  $y = y_{n-1} \dots y_0$  que resulta em  $z = z_{n-1} \dots z_0$

$$v = x_{n-1}'y_{n-1}'z_{n-1} + x_{n-1}y_{n-1}z_{n-1}'$$

Onde  $v=1$  significa que ocorre overflow e  $v=0$  indica que não ocorreu overflow.

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Exemplo: Detector de Overflow:

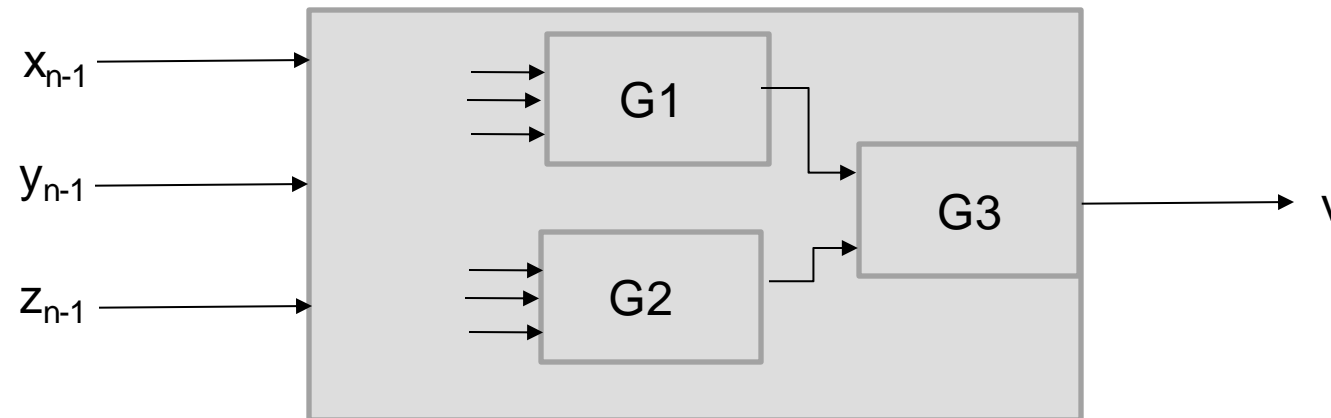
$$V = \underbrace{X_{n-1} Y_{n-1} Z_{n-1}}_{G1} + \underbrace{X_{n-1} Y_{n-1} Z_{n+1}}_{G2}$$

$\underbrace{\hspace{10em}}_{G3}$

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Exemplo: Detector de Overflow:

$$v = x_{n-1}'y_{n-1}'z_{n-1} + x_{n-1}y_{n-1}z_{n-1}$$



Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

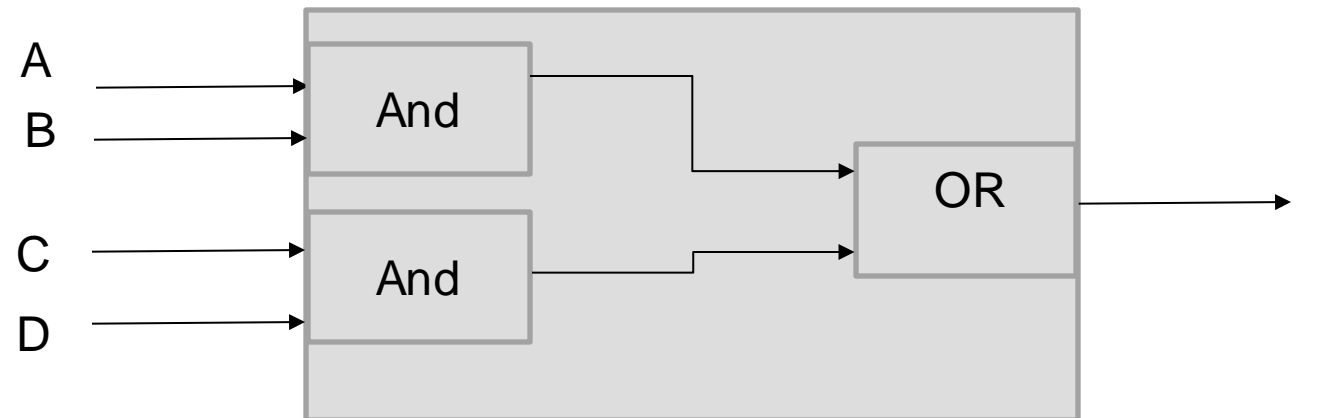
Exemplo:

$$f = A.B + C.D$$

# Arquitetura Estrutural

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Exemplo:  $f = A.B + C.D$



Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Sintaxe:

A sintaxe é marcada pela presença de dois componentes:

Esse tipo de arquitetura descreve os **componentes** do sistema digital e **suas interconexões**. Nessa arquitetura, a atribuição de sinais é realizada a partir do **mapeamento de entradas e saídas** de cada componente

Sintaxe:

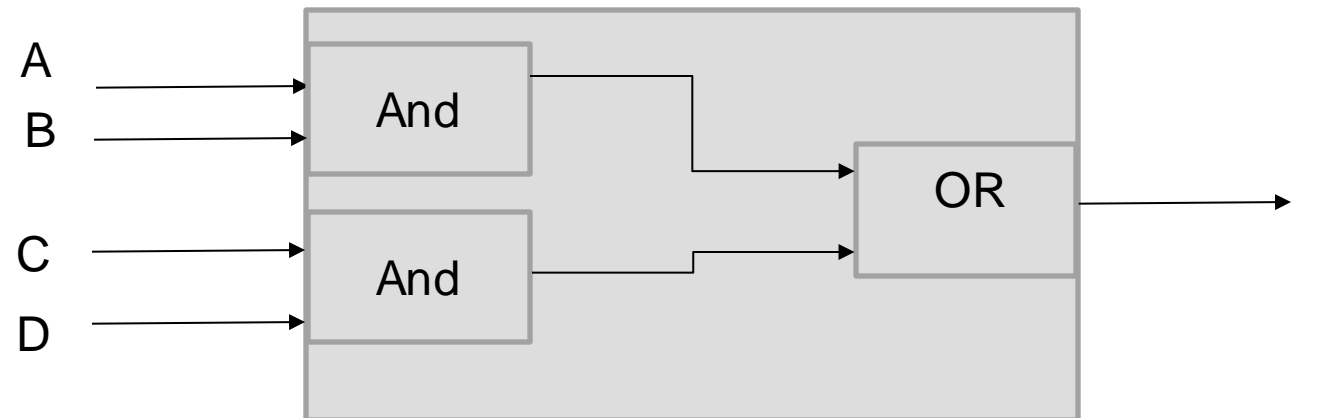
A sintaxe é marcada pela presença de dois componentes:

**Component:** descreve a interface do componente, ou seja, funciona como a ENTITY na construção de componente

**Port map:** faz o mapeamento dos sinais de entrada e saída.

# Arquitetura Estrutural

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural





# Arquitetura Estrutural

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY and_gate IS
```

```
  Port(a, b: in bit;
```

```
        s: out bit);
```

```
End and_gate;
```

```
Architecture dataflow of and_gate IS
```

```
Begin
```

```
  s <= a AND b;
```

```
End dataflow;
```

# Arquitetura Estrutural

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY and_gate IS
```

```
    Port(a, b: in bit;
```

```
          s: out bit);
```

```
End and_gate;
```

```
Architecture dataflow of and_gate IS
```

```
Begin
```

```
    s <= a AND b;
```

```
End dataflow;
```

Geralmente, esses  
**COMPONENTES** ficam  
em outros arquivos  
VHDL dentro do mesmo  
projeto

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY or_gate IS
```

```
  Port(a, b: in bit;
```

```
        s: out bit);
```

```
End or_gate;
```

```
Architecture dataflow of or_gate IS
```

```
Begin
```

```
  s <= a OR b;
```

```
End dataflow;
```

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY exemplo1 IS
```

```
  Port (a, b, c, d: in bit;
```

```
        f: out bit);
```

```
end exemplo1;
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY exemplo1 IS
```

```
  Port (a, b, c, d: in bit;
```

```
        f: out bit);
```

```
end exemplo1;
```

```
-- Vamos criar os componentes
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

```
ENTITY exemplo1 IS
```

```
  Port (a, b, c, d: in bit;
```

```
        f: out bit);
```

```
end exemplo1;
```

```
Component and_gate
```

```
  Port(a,b: in bit;
```

```
        s: out bit);
```

```
End and_gate;
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Component or\_gate

Port(a,b: in bit;

s: out bit);

End or\_gate;



# Arquitetura Estrutural

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

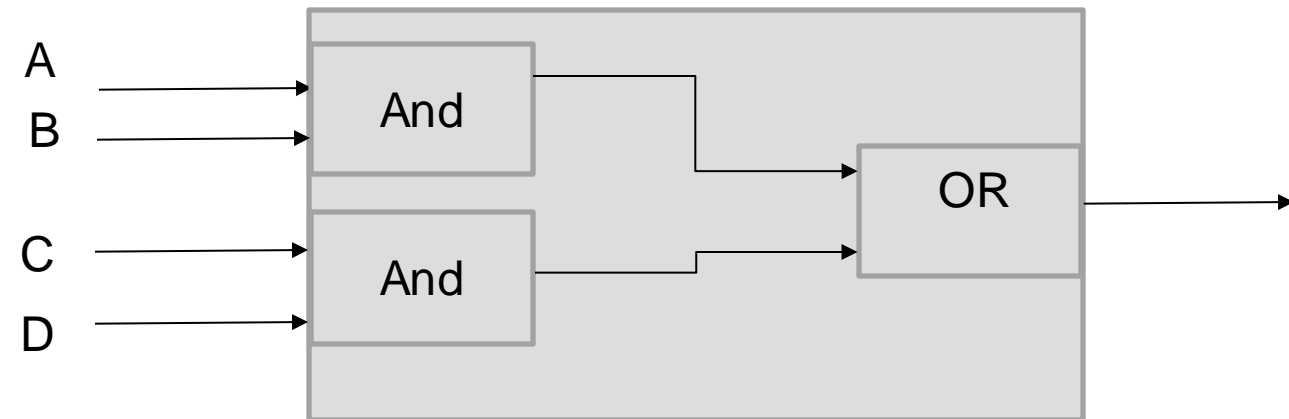
Architecture structural of exemplo1 is

```
SIGNAL x1, x2: bit;
```

```
BEGIN
```

```
-- Mapeamento dos componentes criados
```

```
End structural;
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Architecture structural of exemplo1 is

```
SIGNAL x1, x2: bit;
```

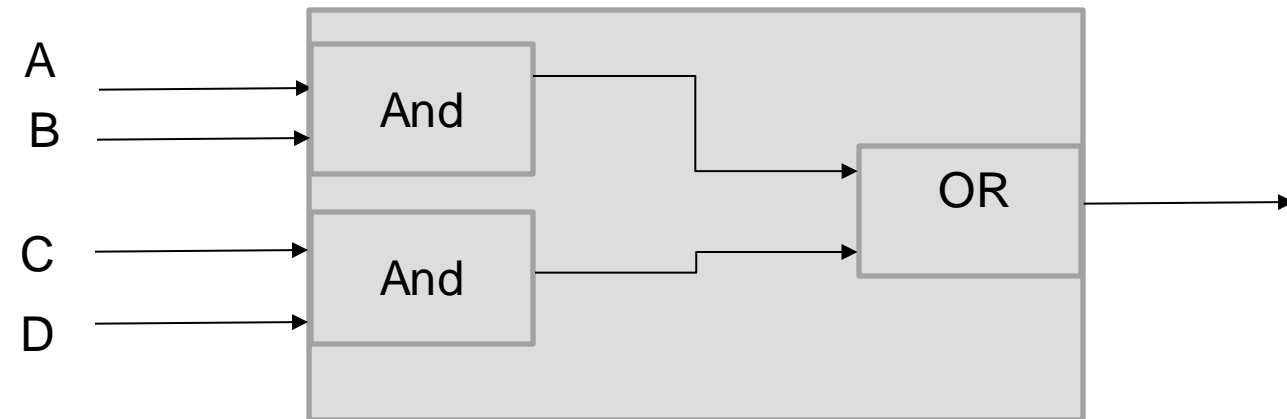
```
BEGIN
```

```
And_0: PORT MAP(a, b, x1);
```

```
And_1: PORT MAP(c, d, x2);
```

```
OR_0: PORT MAP(x1, x2, f);
```

```
End structural;
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Architecture structural of exemplo1 is

```
SIGNAL x1, x2: bit;
```

```
BEGIN
```

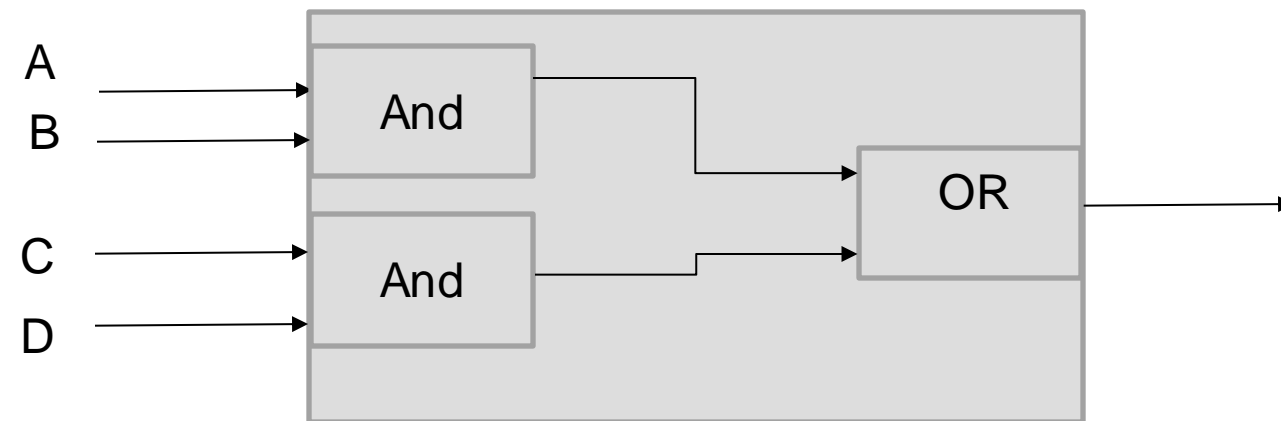
```
And_0: PORT MAP(a, b, x1);
```

```
And_1: PORT MAP(c, d, x2);
```

```
OR_0: PORT MAP(x1, x2, f);
```

Label

```
End structural;
```



# Arquitetura Estrutural

Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Architecture structural of exemplo1 is

```
SIGNAL x1, x2: bit;
```

```
BEGIN
```

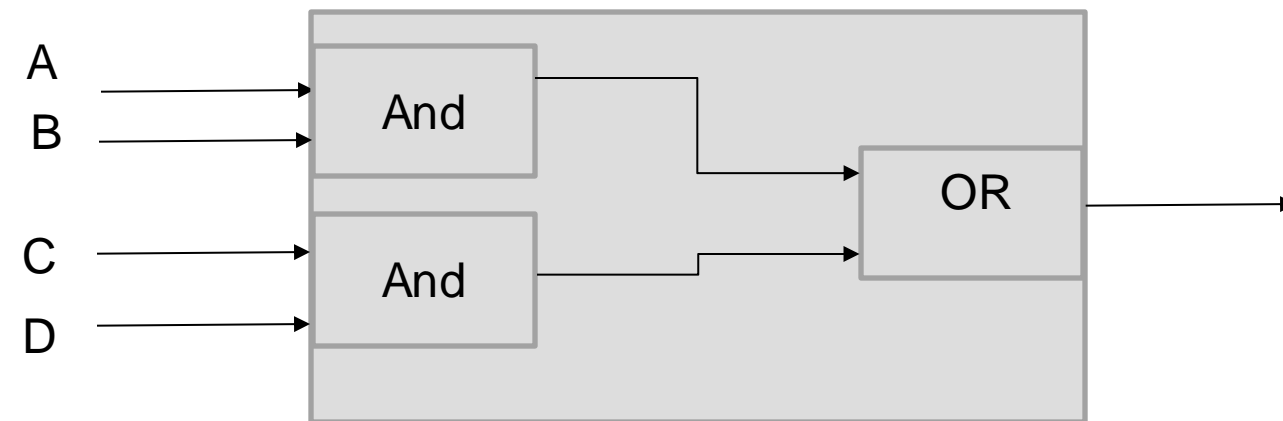
```
And_0: PORT MAP(a, b, x1);
```

```
And_1: PORT MAP(c, d, x2);
```

```
OR_0: PORT MAP(x1, x2, f);
```

Label identifica o  
componentes

```
End structural;
```



Exemplo: Faça o exemplo  $f = A.B + C.D$  conforme a arquitetura estrutural

Primeiro vamos criar os componentes:

Architecture structural of exemplo1 is

```
SIGNAL x1, x2: bit;
```

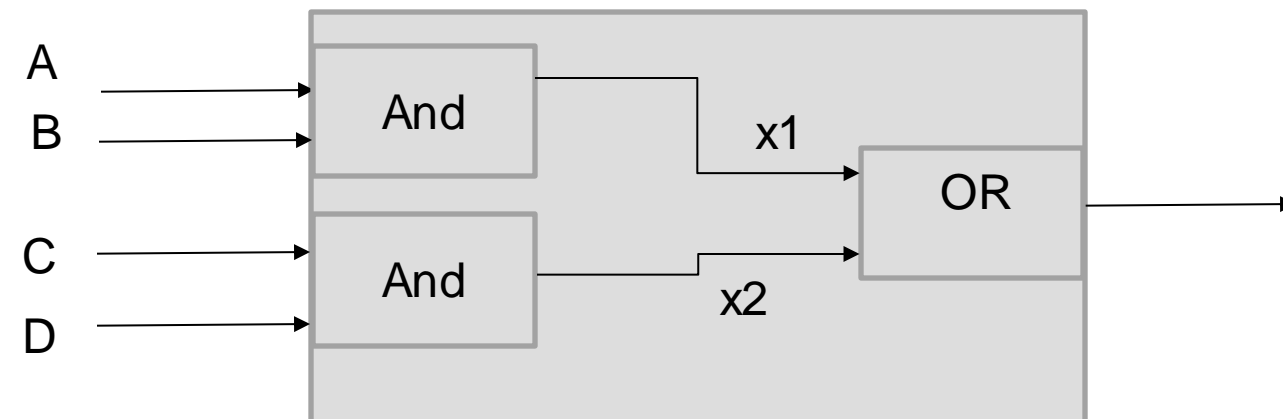
```
BEGIN
```

```
And_0: PORT MAP(a, b, x1);
```

```
And_1: PORT MAP(c, d, x2);
```

```
OR_0: PORT MAP(x1, x2, f);
```

Sinais temporários



```
End structural;
```

Exercício: Faça o exemplo do detector de overflow usando architecture structural

# Estruturas Concorrentes

Neste tópico, apresenta-se estruturas **de condição e de repetição** em códigos concorrentes.

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

**Assignment WHEN condition ELSE**

**Assignment WHEN condition ELSE**

**...;**

Exemplos:

```
x<= a when sel=0 else  
  b when sel=1 else  
  c;
```



Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

Assignment WHEN condition ELSE

Assignment WHEN condition ELSE

...;

Exemplos:

Condição



```
x <= a when sel=0 else  
  b when sel=1 else  
  c;
```

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

**Assignment WHEN condition ELSE**

**Assignment WHEN condition ELSE**

**...;**

Exemplos:

1° verifica se sel=0.

x <= a when sel=0 else  
b when sel=1 else  
c;

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

Assignment WHEN condition ELSE

Assignment WHEN condition ELSE

...;

Exemplos:

```
x <= a when sel=0 else  
  b when sel=1 else  
  c;
```

1° verifica se sel=0.  
Caso não, verifica se  
sel=1

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

Assignment WHEN condition ELSE

Assignment WHEN condition ELSE

...;

Exemplos:

```
x <= a when sel=0 else  
  b when sel=1 else  
  c;
```

↙ Caso não, então a saída é sempre c

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

Assignment WHEN condition ELSE

Assignment WHEN condition ELSE

...;

Exemplos:

```
x <= a when sel=0 else  
  b when sel=1 else  
  c;
```

Portanto, a saída default  
c ocorre quando sel!=0 e  
sel!=1

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

```
Assignment WHEN condition ELSE  
Assignment WHEN condition ELSE  
...;
```

Sintaxe (WITH-SELECT-WHEN)

```
WITH identifier SELECT  
assignment when condition else  
assignment when condition else  
...;
```

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

```
Assignment WHEN condition ELSE  
Assignment WHEN condition ELSE  
...;
```

Sintaxe (WITH-SELECT-WHEN)

```
WITH identifier SELECT  
assignment when condition else  
assignment when condition else  
...;
```

Nesse caso a condition é o valor que variável  
identifier pode assumir

Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

```
Assignment WHEN condition ELSE  
Assignment WHEN condition ELSE  
...;
```

Sintaxe (WITH-SELECT-WHEN)

```
WITH identifier SELECT  
assignment when condition else  
...;
```

Exemplo:

```
WITH sel SELECT  
X <= a WHEN 0 else  
b WHEN 1 else  
c WHEN OTHERS;
```



Estruturas **de condição** em códigos concorrentes.

- The WHEN statement: atua como uma estrutura de condição, tendo duas formas possíveis. Vamos mostrar as sintaxes em exemplos de multiplexadores.

Sintaxe (WHEN-ELSE)

```
Assignment WHEN condition ELSE  
Assignment WHEN condition ELSE  
...;
```

Sintaxe (WITH-SELECT-WHEN)

```
WITH identifier SELECT  
assignment when condition else  
...;
```

Exemplo:

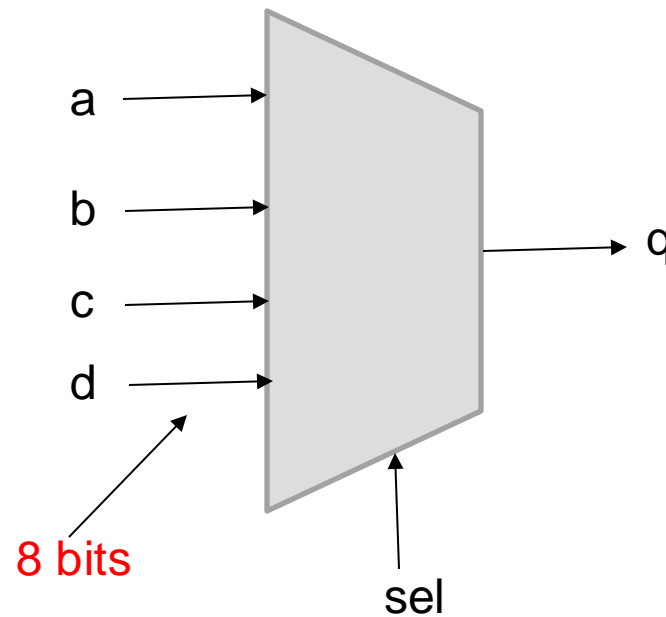
```
WITH sel SELECT  
X <= a WHEN 0 else  
b WHEN 1 else  
c WHEN OTHERS;
```

# Estruturas Concorrentes

Exemplo: MUX 4:1 COM VETORES DE 8 BITS

# Estruturas Concorrentes

Exemplo: MUX 4:1 COM VETORES DE 8 BITS



Exemplo: MUX 4:1 COM VETORES DE 8 BITS

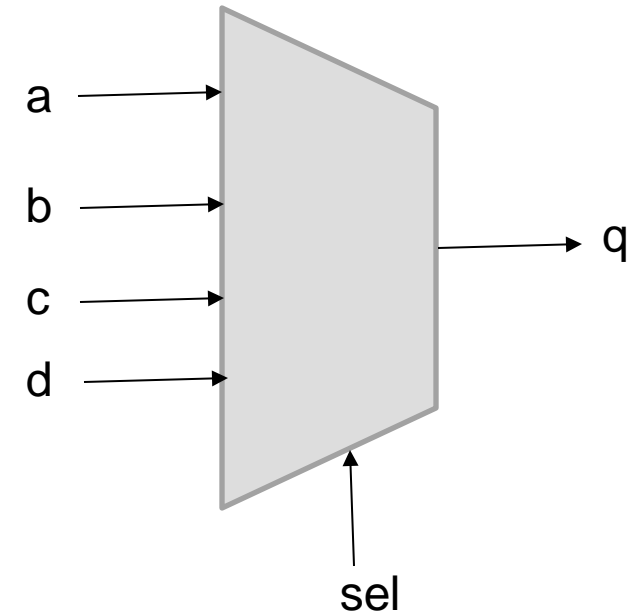
```
library ieee;
```

```
use ieee.std_logic_1164.all
```

```
Entity mux_4_1 is
```

```
    PORT (a, b, c, d:    in std_logic_vector(7 downto 0);  
          sel:          in std_logic_vector(1 downto 0);  
          q:            out std_logic_vector(7 downto 0));
```

```
End mux_4_1;
```



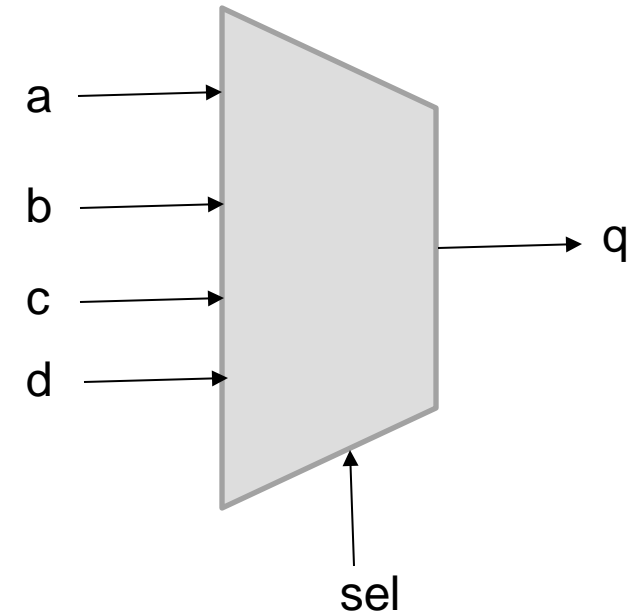
Exemplo: MUX 4:1 COM VETORES DE 8 BITS

Architecture dataflow of mux\_4\_1 is

BEGIN

```
q <= b when (sel="01") else  
  c when(sel="10") else  
  d when(sel="11") else  
  a;                --default case
```

END dataflow;



# Estruturas Concorrentes

The **GENERATE** statement: isso equivale a uma estrutura de repetição, ou seja, a um LOOP.

Vamos ver a sintaxe para implementar um loop em código concorrente.

The **GENERATE** statement: isso equivale a uma estrutura de repetição, ou seja, a um LOOP.

- Sintaxe:
  - **Label: for identifier in range GENERATE**  
**Concurrent assignments;**  
**End GENERATE [label];**
    - Label: identifica a estrutura
    - Identifier é a variável que percorre o range
    - Range especifica um certo intervalo

# Estruturas Concorrentes

Diferenças de FOR-GENERATE para Estrutura For em linguagem de programação.



ARCHITECTURE structural of teste is

COMPONENT func

PORT(a0, a1: int std\_logic;

Y: out std\_logic);

End COMPONENT;

BEGIN

G1: for i in (N-1) Downto 0 GENERATE

func\_i: PORT MAP(a0=>sig1(i), a1=>sig2(i), y=>z(i));

End GENERATE G1;

END structural;

# Repetição de Código para Components

ARCHITECTURE structural of teste is

COMPONENT func

```
PORT(a0, a1: int std_logic;
```

```
Y: out std_logic);
```

```
End COMPONENT;
```

```
BEGIN
```

```
G1: for i in (N-1) Downto 0 GENERATE
```

```
func_i: PORT MAP(a0=>sig1(i), a1=>sig2(i), y=>z(i));
```

```
End GENERATE G1;
```

```
END structural;
```

Cria uma  
repetição de  
múltiplos  
componentes

The **GENERATE** statement: isso equivale a uma estrutura de repetição, ou seja, a um LOOP.

- Exemplo:

```
G1: for i in 0 to M GENERATE
```

```
    b(i) <= a(M-i);
```

```
End Generate G1;
```

# Arquitetura Comportamental

Arquitetura utilizada na descrição de **sistemas sequenciais**. Essa arquitetura inaugura o **mundo sequencial** em VHDL

Elemento principal: **PROCESS**

Sintaxe do elemento PROCESS:

```
process_name: PROCESS (sensitivity_list1, ...)
```

```
BEGIN
```

```
-- comando do process
```

```
END PROCESS process_name;
```

**process\_name:** atua como o label identificando o PROCESS

Sintaxe do elemento PROCESS:

```
process_name: PROCESS (sensitivity_list1, ...)
```

```
BEGIN
```

```
-- comando do process
```

```
END PROCESS process_name;
```

**process\_name:** atua como o label identificando o PROCESS

**PROCESS:** palavra reservada para indicar início da parte comportamental

Sintaxe do elemento PROCESS:

```
process_name: PROCESS (sensitivity_list1, ...)
```

```
BEGIN
```

```
-- comando do process
```

```
END PROCESS process_name;
```

**process\_name:** atua como o label identificando o PROCESS

**PROCESS:** palavra reservada para indicar início da aprte comportamental

**Lista de sensibilidade:** lista de sensibilidade que indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída.

Sintaxe do elemento PROCESS:

```
process_name: PROCESS (sensitivity_list1, ...)
```

```
BEGIN
```

```
-- comando do process
```

```
END PROCESS process_name;
```

**process\_name:** atua como o label identificando o PROCESS

**PROCESS:** palavra reservada para indicar início da aprte comportamental

**Lista de sensibilidade:** lista de sensibilidade que indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída. **Em outras palavras, quais sinais fazem com que os comandos do PROCESS se movimentem, sem alterem.**



# Arquitetura Comportamental

Sempre uma variável da lista de sensibilidade é alterada, todo o código do PROCESS é executado

Sintaxe do elemento PROCESS:

```
process_name: PROCESS (sensitivity_list1, ...)  
BEGIN  
-- comando do process  
END PROCESS process_name;
```

Os comandos dentro Process são  
**Sequenciais**

**process\_name:** atua como o label identificando o PROCESS

**PROCESS:** palavra reservada para indicar início da aprte comportamental

**Lista de sensibilidade:** lista de sensibilidade que indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída. **Em outras palavras, quais sinais fazem com que os comandos do PROCESS se movimentem, sem alterem.**

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade 8 bits

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

A saída do codificador de prioridade mostra o código do binário mais prioritário. Considere prioridade como o seguinte:

Seja  $x = x_7 \dots x_0$  seja a entrada do codificador de prioridade.

Se  $i > j$ , logo a entrada  $x_i$  é mais prioritária que  $x_j$

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

A saída do codificador de prioridade mostra o código do binário mais prioritário. Considere prioridade como o seguinte:

Seja  $x = x_7 \dots x_0$  seja a entrada do codificador de prioridade.

Se  $i > j$ , logo a entrada  $x_i$  é mais prioritária que  $x_j$

Exemplo: Se  $x_1 = x_5 = 1$  e todos os demais igual a 0. Logo o codificador de prioridade emitirá a saída sendo o binário de 5

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

```
use ieee;  
use ieee.std_logic_1164.all  
ENTITY priority_coder IS  
    Port(x0, x1, x2, x3, x4, x5, x6, x7: in std_logic;  
        prio_out: out std_logic_vector(7 DOWNTO 0)  
    );  
End priority_coder;
```

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

    p\_code: process (x0, x1, x2, x3, x4, x5, x6, x7)

    Begin

        if (x7=1) then prio\_out="111";

        Elsif(x6=1) then prio\_out="110";

        ...

        ...

        else prio\_out="000";

    End process p\_code;

End behavioral;



# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

p\_code: process (x0, x1, x2, x3, x4, x5, x6, x7)

Begin

if (x7=1) then prio\_out="111";

Elsif(x6=1) then prio\_out="110";

...

...

else prio\_out="000";

End process p\_code;

End behavioral;

Há Begin para  
architecture e outro  
begin pro process

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

```
p_code: process (x0, x1, x2, x3, x4, x5, x6, x7)
```

```
Begin
```

```
if (x7=1) then prio_out="111";
```

```
Elsif(x6=1) then prio_out="110";
```

```
...
```

```
...
```

```
else prio_out="000";
```

```
End process p_code;
```

```
End behavioral;
```

";" para cada fim de if-  
elseif statement

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

p\_code: process (x0, x1, x2, x3, x4, x5, x6, x7)

Begin

if (x7=1) then prio\_out="111";

Elsif(x6=1) then prio\_out="110";

...

...

else prio\_out="000";

End process p\_code;

End behavioral;

Lista de Sensibilidade



# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

```
p_code: process (x0, x1, x2, x3, x4, x5, x6, x7)
```

```
Begin
```

```
  if (x7=1) then prio_out="111";
```

```
  Elsif(x6=1) then prio_out="110";
```

```
  ...
```

```
  ...
```

```
  else prio_out="000";
```

```
  End if;
```

```
End process p_code;
```

```
End behavioral;
```

End if



# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

```
p_code: process (x0, x1, x2, x3, x4, x5, x6, x7)
```

```
Begin
```

```
  if (x7=1) then prio_out="111";
```

```
  Elsif(x6=1) then prio_out="110";
```

```
  ...
```

```
  ...
```

```
  else prio_out="000";
```

```
  End if;
```

```
End process p_code;
```

```
End behavioral;
```

Comandos Sequencial

# Arquitetura Comportamental

Exemplo: Descreva, em VHDL, um codificador de prioridade com 8 bits

Architecture behavioral of priority\_coder is

Begin

```
p_code: process (x0, x1, x2, x3, x4, x5, x6, x7)
```

```
Begin
```

```
  if (x7=1) then prio_out="111";
```

```
  Elsif(x6=1) then prio_out="110";
```

```
  ...
```

```
  ...
```

```
  else prio_out="000";
```

```
End if;
```

```
End process p_code;
```

```
End behavioral;
```

If-elseif-else statement

```
If (condition) then assignments;  
Elseif (condition) then assignments;  
Else assignment;  
End if;
```

# Arquitetura Comportamental

Exemplo: Identificador de igualdade de 4 bits usando Process

# Arquitetura Comportamental

Exemplo: Identificador de igualdade de 4 bits usando Process

O modelo recebe dois sinais com 4 bits cada, compara-os e emite saída 1, se forem iguais e zero, caso contrário.



# Arquitetura Comportamental

Exemplo: Identificador de igualdade de 4 bits usando Process

O modelo recebe dois sinais com 4 bits cada, compara-os e emite saída 1, se forem iguais e zero, caso contrário.

# Arquitetura Comportamental

Exemplo: Identificador de igualdade de 4 bits usando Process

```
Entity equal_identifier IS
```

```
    Port (
```

```
    );
```

```
End equal_identifier;
```

# Estruturas Sequenciais

Neste tópico, apresenta-se estruturas **de condição e de repetição** em códigos sequenciais.

- The IF statement
- The WAIT statement
- The CASE statement
- The LOOP statement

Só são permitidas dentro da estrutura de PROCESS, pois são comandos SEQUENCIAIS

Neste tópico, apresenta-se estruturas **de condição e de repetição** em códigos sequenciais.

- The IF statement: estrutura de decisão
- Sintaxe

```
IF condition THEN
    assignments;
ELSIF condition THEN
    assignments;
ELSE
    assignments;
End IF;
```

Neste tópico, apresenta-se estruturas **de condição e de repetição** em códigos sequenciais.

- The IF statement: estrutura de decisão
- Sintaxe

```
IF condition THEN
    assignments;
ELSIF condition THEN
    assignments;
ELSE
    assignments;
End IF;
```

Neste tópico, apresenta-se estruturas **de condição** em códigos sequenciais.

- The Case statement: estrutura de decisão
- Sintaxe

CASE identifier IS

WHEN value => assignments;

WHEN value => assignments;

...;

END CASE;

Exemplo:

Case sel is

When 0 => a;

When 1=> b;

When OTHERS => c;

End Case;

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**For - LOOP**):

```
[label] FOR identifier in range LOOP  
    -- sequential assignments;  
END LOOP [label];
```

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**For - LOOP**):  
← **Opcional**

```
[label] FOR identifier in range LOOP  
    -- sequential assignments;  
END LOOP [label];
```

Exemplo:

```
FOR i in x'RANGE LOOP  
    x(i) <= a(M-i) AND b(i);  
End LOOP;
```



Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**For - LOOP**):  
Opcional

←  
[label] FOR identifier in range LOOP  
    -- sequential assignments;  
END LOOP [label];

Exemplo:

FOR i in x'RANGE LOOP  
    x(i) <= a(M-i) AND b(i);  
End LOOP;

x'RANGE  
percorre  
todos os  
bits de x

# Estruturas Sequenciais

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**For - LOOP**):  
Opcional

←  
[label] FOR identifier in range LOOP  
    -- sequential assignments;  
END LOOP [label];

Exemplo:

Python:  
range(0, len(x))

FOR i in x'RANGE LOOP  
    x(i) <= a(M-i) AND b(i);  
End LOOP;

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**While - LOOP**):

← **Opcional**

```
[label] WHILE condition LOOP
  -- sequential assignments;
END LOOP [label];
```

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**While - LOOP**):  
← **Opcional**

```
[label] WHILE condition LOOP
  -- sequential assignments;
END LOOP [label];
```

Exemplo:

```
i:=0;
WHILE i<=M LOOP
  x(i) <= a(M-i) AND b(i);
  i := i + 1;
End LOOP;
```

Python:  
range(0, len(x))



# Estruturas Sequenciais

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**LOOP with EXIT**): essa estrutura faz um loop com uma condição de SAÍDA

Neste tópico, apresenta-se estruturas **de repetição** em códigos sequenciais.

The **LOOP** statement: isso permite criação de **múltiplas assignments** por meio de uma **estrutura de repetição**

Sintaxe (**LOOP with EXIT**):

```
For identifier in range LOOP  
    [label:] EXIT [loop_label] WHEN condition;  
    assignments;  
End LOOP;
```

# Estruturas Sequenciais

Exemplo: Conte o número de zeros presentes em uma sequencia de 8 bits



# Estruturas Sequenciais

Exemplo: Conte o número de zeros presentes em uma sequência de 8 bits

Entity counting\_zeros\_8 IS

Generic(N: Integer := 8);

Port(x: in bit\_vector(N-1 downto 0);

y: out INTEGER RANGE 0 TO N

);

End counting\_zeros\_8;

Exemplo: Conte o número de zeros presentes em uma sequencia de 8 bits

ARCHITECTURE behavioral of counting\_zeros\_8 IS

BEGIN

    Process(x)

        VARIABLE temp: INTEGER 0 RANGE N;

        Begin

            temp := 0;

            For i in x'RANGE LOOP

                IF (x(i) = '0') then

                    temp:=temp+1;

                End IF;

            y <= temp;

        End Process;

END behavioral;

# Estruturas Sequenciais

Exemplo: Conte o número de zeros presentes, em sequência, em uma sequência de 8 bits

# Estruturas Sequenciais

Exemplo: Conte o número de zeros presentes em uma sequencia de 8 bits

ARCHITECTURE behavioral of counting\_zeros\_8 IS

BEGIN

    Process(x)

        VARIABLE temp: INTEGER 0 RANGE N;

        Begin

            temp := 0;

            For i in x'RANGE LOOP

                EXIT WHEN x(i) = '1';

                temp := temp + 1;

            End IF;

            y<= temp;

        End Process;

END behavioral;

# Signal vs Variable

## Signal

- Declarada na Architecture

## Variable

- Declarada dentro do PROCESS

# Signal vs Variable

## Signal

- Declarada na Architecture
- Escopo global

## Variable

- Declarada dentro do PROCESS
- Escopo local

# Signal vs Variable

## Signal

- Declarada na Architecture
- Escopo global
- Novo valor é considerado após a conclusão do PROCESS

## Variable

- Declarada dentro do PROCESS
- Escopo local
- Novo valor já está disponível imediatamente dentro do PROCESS

# Signal vs Variable

## Signal

- Declarada na Architecture
- Escopo global
- Novo valor é considerado após a conclusão do PROCESS
- Atribuição: **<=**

## Variable

- Declarada dentro do PROCESS
- Escopo local
- Novo valor já está disponível imediatamente dentro do PROCESS
- Atribuição: **:=**



# Exercício

- Construa um contador de 4 bits
- Construa um gerador de paridade

# Dúvidas

- E-mail de contato: [pacheco@gta.ufrj.br](mailto:pacheco@gta.ufrj.br)