SERVIÇO PÚBLICO FEDERAL
UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ - UNIFESSPA
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS - IGE
FACULDADE DE COMPUTAÇÃO E ENG. ELÉTRICA – FACEEL
CURSO ENGENHARIA DA COMPUTAÇÃO

Microeletrônica

T-2018

Prof. José Carlos Da Silva
jcdsilv@hotmail.com
jose-carlos.silva@unifesspa.edu.br
whatsApp: 94-981431852

Maio/2022

Conteúdo

- Introdução a VHDL;
- Estruturas do código VHDL;
- Bibliotecas e pacotes fundamentais;
- Tipos de dados predefinidos;
- Objetos (Constant, Signal, Variable, File);
- Tipos de dados definidos pelo usuário
- Operadores;
- Atributos;
- Código concorrente versus sequencial;
- Código concorrente (WHEN, SELECT, GENERATE);;
- Código sequencial (PROCESS, IF, CASE, LOOP, WAIT)
- Instruções auxiliares (ASSERT, ALIAS);
- Pacotes (PACKAGE);
- Componentes (COMPONENT);
- Funções (FUNCTION);
- Procedimentos (Procedure);
- Introdução a ferramenta de síntese e simulação quartus II;
- Exercícios (Atividades e trabalhos).

SEMÂNTICA

Identificadores

- **Usados como referência a todos os objetos declarados**

- REGRAS:

  - Comentários deve-se utilizar “--”;
    - Ex: x <= a NAND b - - x=[a . b]`

  - Primeiro caractere deve ser uma letra (Obrigatório);
    - Ex: Teste, Teste123

  - Não são CASE-SENSITIVE (maiúsculas/minúsculas);
    - Ex: Teste = teste = TESTE

  - Não é possível o uso de palavras reservadas com outras finalidades;
    - Ex: mux, and

SEMÂNTICA

Identificadores

- São permitidos apenas letras, números e underscore (_);
- Último caractere não pode ser underscore;
  - Ex: Teste_
- Não são permitidos 2 underscores em seqüência;
  - Ex: Teste__projeto
- Nomes com underscore são diferentes de nome sem underscore.
  - Ex: teste_projeto ≠ testeprojeto

SEMÂNTICA

LITERAIS

- **Valores de dados específicos usados como parâmetros de objetos ou dentro de expressões.**

- Não representam tipos específicos:
  - Ex: '1' pode representar um bit ou um caractere.

- São válidos dependendo do tipo:
  - Ex: '$' é válido como um caractere mas não como bit.

SEMÂNTICA

LITERAIS

- **Podem ser representados pelas seguintes categorias:**
  - **Character Literals:** um caracter ASCII (‘a’, ’z’).
  - **String Literals:** seqüência de caracteres ASCII (“texto”).
  - **Bit String Literals:** formas especiais de string literals para representar valores das bases binária, octal e hexadecimal.
    - B”100100”
    - O”446”
    - X”A0F4B51”

Válidos para bit_vector e std_logic_vector

SEMÂNTICA

LITERAIS

- **Numeric Literals:** Integer Literals (Ex: 1) e Real Literals (Ex: 1.1)
  - Números reais não são sintetizáveis.

- **Based Literals:** idêntico a **numeric literals**, mas utilizando bases binária, octal e hexadecimal.
  - Ex: 2#101#, 16#FC9#, 2#1.

- **Physical Literals:** grandeza física. Contém parte numérica e unidade.
  - Podem representar tempo, velocidade, distância etc
  - Não são sintetizáveis (tempo é simulável)
    - Ex: 300s, 40m

SEMÂNTICA

TIPOS DE DADOS (TYPE)

- VHDL é uma linguagem fortemente tipada;
- Pouca conversão é feita automaticamente;
- Cada tipo tem um conjunto de operações válidas;
- Cada tipo tem um conjunto de valores definidos;
- Os tipos podem ser estendidos pelo usuário.

SEMÂNTICA

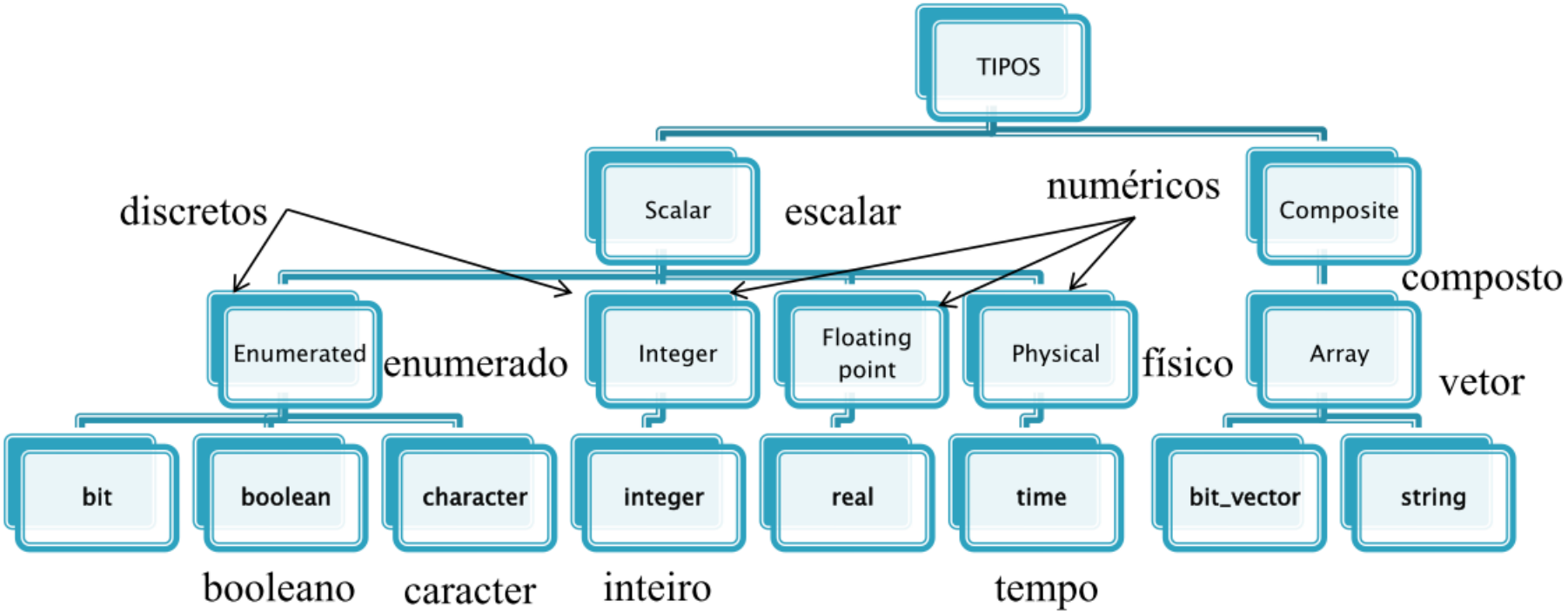
TIPOS DE DADOS (TYPE)

- São divididos em 4 classes:
  - Tipos escalares (representam um único valor)
  - Tipos compostos (representam uma coleção de valores);
  - *Tipos de acessos (similares a ponteiros);
  - *Tipos de arquivo (referencia objetos que contém uma sequência de valores).

*** Não são sintetizáveis**

SEMÂNTICA

TIPOS DE DADOS (TYPE)



SEMÂNTICA
TIPOS ESCALARES

- **Tipos enumerados: tipos já definidos pela norma:**
  - Bit
  - Boolean
  - Integer
  - Real
  - Physical
  - STD_LOGIC

SEMÂNTICA

TIPOS ESCALARES

TIPO PREDEFINIDO	VALOR	EXEMPLO
BIT	um, zero	1, 0
BOOLEAN	Verdadeiro, falso	TRUE, FALSE
CHARACTER	Caracteres ASCII	a, b, c, A, B, C, ?, (
INTEGER	$-2^{31}-1 \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
NATURAL	$0 \leq X \leq 2^{31}-1$	123, 8#173#, 16#7B#, 2#11_11_011#
POSITIVE	$1 \leq X \leq 2^{31}-1$	
REAL	$-3.65*10^{47} \leq x \leq +3.65*10^{47}$	1.23, 1.23E+2, 16#7.B#E+1
TIME	ps=10^3 fs ns=10^3 ps us=10^3 ns ms=10^3 us sec=10^3 ms min=60sec hr=60min	1 us, 100 ps, 1 fs
"NATURAL" e "POSITIVE" são subtipos de "INTEGER"		

SEMÂNTICA

TIPOS ESCALARES

- **Tipos enumerados: permite criar novos tipos.**
  - Útil para máquina de estados (FSM)
    - Ex: type estado is (inicio, espera, calculo, final);
  - Os tipos criados podem ser declarados
    - Ex: signal estado_atual : estado
  - Outros Exemplos (user defined types):
    - type dedos is range 0 to 10;
    - type pos_neg is range -1 to 1;

SEMÂNTICA

TIPOS COMPOSTO

- **ARRAY: Coleção de elementos do mesmo tipo**

- Ordem crescente ou decrescente
  - type word is array (7 downto 0) of bit;
  - type word is array (0 to 7) of bit;

- Indexação por parêntesis
  - signal palavra: word := “01111111”;
  - signal aux: bit;
  - aux <= palavra(0);

SEMÂNTICA

TIPOS “ARRAY” NO PACOTE PADRÃO

TIPO PREDEFINIDO	VALOR	EXEMPLO
BIT_VECTOR	1, 0	“1010”, B”10_10”, O”12”, X”A”
STRING	Tipo character	“texto”, ““incluindo_aspas””

Signal a: Bit_Vector (0 TO 7) := “10110011”

1	0	1	1	0	0	1	1
a(0)	a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	a(7)

Constant c: String (1 TO 9) := “Alo mundo”

A	l	o		m	u	n	d	o
c(1)	c(2)	c(3)	c(4)	c(5)	c(6)	c(7)	c(8)	c(9)

SEMÂNTICA

TIPOS “ARRAY” NO PACOTE PADRÃO (EXEMPLO)

```
a(4) <= b(4);                          -- 1 elemento <= 1 elemento
a(0 to 3) <= b(2 DOWNTO 0);            -- parte do vetor <= parte do vetor
b(4) <= '0';                           -- 1 elemento <= 1 valor
b(2 DOWNTO 0) <= "0010";               -- parte do vetor <= valor
c <= ('0', '0', '0', '1', '0');        -- valor 00010 agregado notação
                                       -- posicional
d <= (1 => '1', OTHERS => '0'); -- valor 00010 agregado notação
                                       -- posicional
```

- **OTHERS**: forma genérica de se atribuir valores aos demais bits não especificados anteriormente

SEMÂNTICA
TIPOS COMPOSTO

- **Records: coleção de elementos de tipos diferentes (Variável especial que contente outras variáveis).**

  - Semelhante a struct em C

  - Exemplos:

    type instruction **is record**
    Mnemonico: **string;**
    Codigo: **bit_vector(3 downto 0);**
    Ciclos: **integer;**
    end record;

    signal instrucao **:instruction;**
    instrucao.Mnemonico:“registrador”
    instrucao.codigo: “0001”
    Instrucao.ciclos : ‘3’

SEMÂNTICA

EXPRESSÕES

- **Realizam operações sobre objetos do mesmo tipo;**

- **Operações lógicas:** and, or, nand, nor, xor, xnor e not; Para vetores, são efetuados bit a bit;

- **Operações relacionais:** igual (=), diferente (/=), menor que (<), menor ou igual (<=), maior que (>), maior ou igual (>=) – o resultado é sempre do tipo boolean;

- **Operações numéricas:** soma (+), subtração (-), negação (- unário), multiplicação (*), divisão (/), módulo (mod), remanescente (rem), expoente (**) e valor absoluto (abs) são aplicados somente para integer e real e para o tipo time, embora existam bibliotecas específicas para aritmética;

SEMÂNTICA

EXPRESSÕES

- **Operações de concatenação:**
  - Cria um novo vetor a partir de dois vetores já existentes.
  - Ex:
    - Dado1: bit_vector(7 downto 0);
    - Dado2: bit_vector(7 downto 0);
    - Dado_Resultante: bit_vector(7 downto 0)
    - Dado1 := “01011011”;
    - Dado2 := “11010010”;
    - Dado_Resultante := (Dado1(7 downto 6) & Dado2(5 downto 2) & Dado1(1 downto 0));
    - Dado_Resultante = “01010011”

SEMÂNTICA

EXPRESSÕES

- Operações lógicas: and, or, nand, nor, xor, not
- Operações relacionais: =, /=, <, <=, >, >=
- Operações aritméticas: - (unária), abs
- Operações aritméticas: +, -
- Operações aritméticas: *, /
- Operações aritméticas: mod, rem, **
- Concatenação: &

Menor

Prioridade

Maior

SEMÂNTICA

OBJETOS DE DADOS

- **Usados para representar e armazenar dados;**
  - Três tipos básicos: constantes, sinais e variáveis;
  - Cada objeto possui um tipo de dados específico e um conjunto de possíveis valores;
  - Objetos de dados de tipos diferentes não podem ser atribuídos um ao outro.
    - Ex: somar 101(Bit) e 011(Std_logic).

SEMÂNTICA

Constantes

- **Assumem apenas um valor em todo o código.**

- Declaração:
  - constant <identificador>: <tipo> := <valor>
    - Ex: constant errado : boolean := False;
    - Ex: constant parte_ram : bit_vector(3 downto 0) := 1110;
    - Podem ser declaradas em qualquer parte do código

- Constante com valor global:
    - Package (uso mais freqüente)

- Somente no contexto em que foi declarada:
    - entity, architecture, process, function

SEMÂNTICA

Sinais

- **Representam ligações entre elementos;**
- Comunicação de módulos em uma estrutura
- Temporizados
- Declaração:
  - signal <identificador>: <tipo> [<= valor];
  - Podem ser declaradas:
    - Globalmente:
      - Package
    - Internamente:
      - architecture (mais utilizado)

SEMÂNTICA

Variáveis

- **Utilizados para armazenar valores intermediários entre expressões;**
- Atribuição imediata;
- Declaração:
  - variable <identificador>: <tipo> [:= valor];
- Podem ser declaradas apenas em processos (variáveis locais);
- Podem corresponder a registradores (processos sem temporização) ou não (processos com temporização);

SEMÂNTICA

Atribuição a Sinais e Variáveis

- Quando utiliza-se sinal, a atribuição ocorre no final do processo, enquanto que a atribuição na variável ocorre simultaneamente.

- Diferença entre as atribuições:
  <= (atribuição de sinal)
  := (atribuição de variável)

- Dentro de um processo (atribuições sequenciais) um sinal só pode ter atribuído um valor de cada vez.

Architecture (Arquitetura)

- **Descrição comportamental:**
  - Descreve o que o sistema deve fazer de forma abstrata;

- **Descrição por fluxo de dados (data-flow):**
  - Descreve o que o sistema deve fazer utilizando expressões lógicas.

- **Descrição estrutural:**

  - Descreve como é o hardware em termos de interconexão de componentes.

Comandos concorrentes e paralelos

- **Comandos paralelos:** operam simultaneamente, geralmente, sem comunicação entre si;

- **Comandos concorrentes:** operam ao mesmo tempo com uma cooperação implícita na comunicação entre eles.

Natureza Concorrentes

- Usados para imitar a natureza concorrente e paralela do hardware.
  - Um comando do tipo:
    - a <= b XOR c
- Será executado apenas quando os sinais da direita mudam o valor e não por ordem de aparecimento na descrição ou no fluxo de controle.

Natureza Concorrentes

Exemplo: ou-exclusivo (XOR)

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

ENTITY xor2 IS
PORT (  a, b : IN STD_LOGIC;
        s : OUT STD_LOGIC);
END xor2;
```

```
ARCHITECTURE behavior OF xor2 IS
SIGNAL int1, int2 : STD_LOGIC;
BEGIN
  int1 <= NOT a AND b;
  int2 <= a AND NOT b;
  s <= int1 OR int2;
END behavior;
```

s <= int1 OR int2 só será executado se alguma modificação de a ou b provocarem alguma modificação em um ou ambos os sinais int1 e int2. A ordem de aparecimento na descrição em nada modifica o resultado.

Natureza Concorrentes

- **WHEN-ELSE**
  - Atribuição condicional de sinais.
- **WITH-SELECT**
  - Atribuição de sinal com escolha.
- **PROCESS**
  - Concorrente entre si e Sequencial na estrutura do código.

Comandos Concorrentes

- **WHEN-ELSE**
  - Atribuição de sinal condicional

```
LABEL:
<sinal> <=      <expressão> WHEN <expressão_booleana> ELSE
                <expressão> WHEN <expressão_booleana> ELSE
                <expressão>
```

Comandos Concorrentes

- **WHEN-ELSE**
  - Atribuição de sinal condicional

```
ENTITY teste IS
PORT (a, b, c : IN integer;
      z : OUT integer);
END teste;

ARCHITECTURE behavior OF teste IS
SIGNAL x : integer;
BEGIN
  x <= 3;
  z <= a WHEN (x > 3) ELSE
       b WHEN (x < 3) ELSE
       c;
END behavior;
```

Comandos Concorrentes

- **WHEN-ELSE**
  - As opções de escolha são definidas por expressões que retornam um valor booleano.
  - Como WHEN-ELSE define uma prioridade na ordem das opções, o circuito equivalente corresponde a uma cadeia de seletores.

```
s0 <=     i0 OR i1    WHEN na = 8 OR nb = 2    ELSE
          i1          WHEN na = 3 OR nb = 5    ELSE
          i0 AND i1 WHEN na = 7                ELSE
          i0;
```

Comandos Concorrentes

- **WHEN-ELSE**

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

ENTITY Mux_1b IS
PORT (
   D0, D1, Sinal : IN  std_logic;
   Saida         : OUT std_logic
 );
END Mux_1b;

ARCHITECTURE behavior_we OF Mux_1b IS
BEGIN
   Saida <=   D0 WHEN Sinal = '0' ELSE
              D1 WHEN Sinal = '1';
END behavior;
```

Comandos Concorrentes

- **WHEN-ELSE**

DECODIFICADOR 3x8

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

ENTITY decod3to8 IS
PORT (
    endereco  : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    Saida     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)      );
END decod3to8;
ARCHITECTURE behav OF decod3to8 IS
BEGIN
    Saida <= "00000001" WHEN  endereco = "000" ELSE
             "00000010" WHEN  endereco = "001" ELSE
             "00000100" WHEN  endereco = "010" ELSE
             "00001000" WHEN  endereco = "011" ELSE
             "00010000" WHEN  endereco = "100" ELSE
             "00100000" WHEN  endereco = "101" ELSE
             "01000000" WHEN  endereco = "110" ELSE
             "10000000" WHEN  endereco = "111";
END behav;
```

Comandos Concorrentes

- **WITH-SELECT**
  - **Atribuição de sinal selecionada**

```
WITH <expressão> SELECT
  <sinal> <= <expressão_a> WHEN <condicao_1>,
             <expressão_b> WHEN <condicao_2>,
             <expressão_c> WHEN <condicao_3> [|
                                <condicao_4>],
             <expressão_d> WHEN OTHERS;
```

Comandos Concorrentes

- **WITH-SELECT**
- As condições são mutuamente exclusivas;
- Não contém uma prioridade como WHEN-ELSE;
- As condições podem ser agrupas através do delimitador “|”, equivalente a “OU”;
- “TO” e “DOWNTO” podem ser empregadas para faixa de valores;
- “OTHERS” é válida, como última alternativa.

Comandos Concorrentes

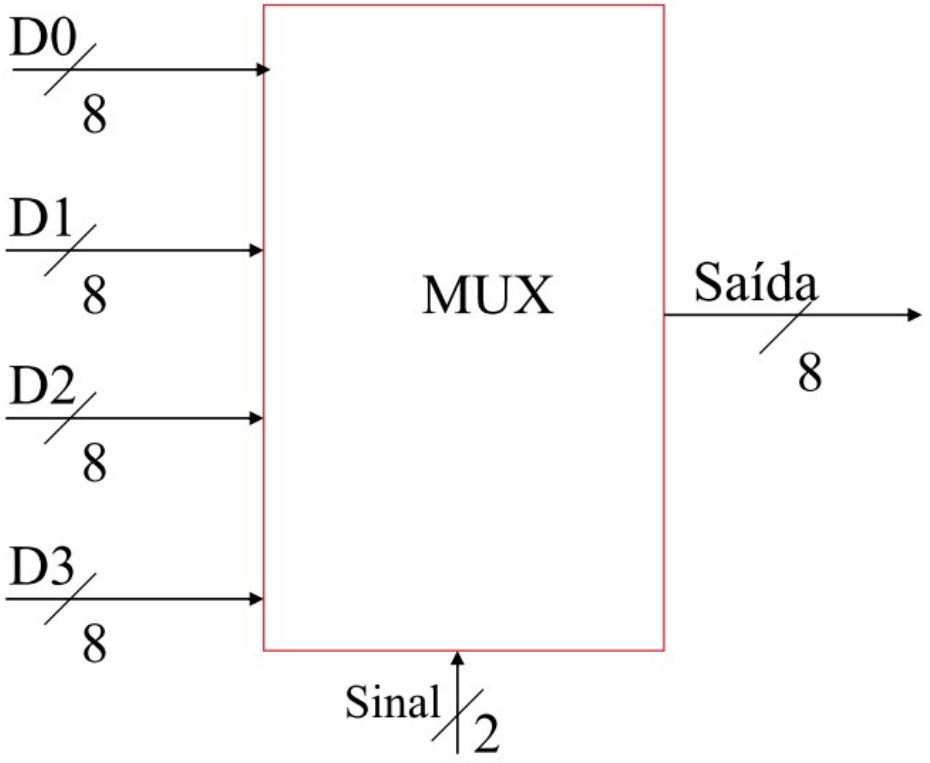
- **WITH-SELECT**
  - **Exemplos:**

```
WITH s0 SELECT -- s0 tipo CHARACTER
        x0 <=   i0 AND i1 WHEN 'a',
                i0 OR i1   WHEN 'b' | 'c',
                i0 XOR i1 WHEN 'd' TO 'g',
                i0            WHEN 'x' DOWNTO 'k',
                i1            WHEN OTHERS;
```

```
WITH b1 AND b0 SELECT -- b1 e b0 tipo BIT
        x1 <=   i0 WHEN '0',
                i1 WHEN '1';
```

Comandos Concorrentes

- **WITH-SELECT**
  - **Exemplos: Multiplexador de 8 bits com 4 entradas**



Comandos Concorrentes

- **WITH-SELECT**
  - **Exemplos: Multiplexador de 8 bits com 4 entradas**

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

ENTITY Mux_8b IS
PORT (
    D0, D1, D2, D3  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    Sinal           : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
    Saida           : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  );
END Mux_8b;

ARCHITECTURE behavior OF Mux_8b IS
BEGIN
    WITH Sinal SELECT
            Saida <=   D0 WHEN "00",
                       D1 WHEN "01",
                       D2 WHEN "10",
                       D3 WHEN OTHERS;
END behavior;
```

Comandos Concorrentes

- **PROCESS (PROCESSOS)**
- **Compostos de:**
  - Parte declarativa
  - Parte de comandos sequenciais
- **Estrutura:**

```
nome: PROCESS (lista de sensibilidade)
  -- declarações de tipos, constantes,variáveis
  -- não é permitido declaração de sinais
  BEGIN
       -- comandos seqüenciais
END PROCESS nome;
```

Comandos Concorrentes

- **PROCESS (PROCESSOS)**
- Processos são concorrentes entre si;
- Dentro de um processo, a execução é sequencial;
- O processo é executado até o último comando e suspenso até que ocorra um evento em sua lista de sensibilidade;
- A lista de sensibilidade é composta por sinais e entradas;
- Um evento em qualquer desses sinais dispara o processo.

Comandos Concorrentes

- **REGIÕES SEQUENCIAIS**
  - Seguem a ordem de aparecimento no código;
  - Comandos sequências só podem ser usados dentro de processos (process);
  - Processos podem ser usados para modelar os diferentes componentes de um sistema;
  - Processos são concorrentes entre si;

Comandos Sequenciais

- IF-THEN-ELSE
- CASE
- NULL
- FOR
- WHILE (* não sintetizável)

Comandos Sequenciais

- **IF-THEN-ELSE:**

```
IF <condição1> THEN
  <comandos>;
ELSIF <condição2> THEN
  <comandos>;
ELSE
  IF <condição3> THEN
      <comandos>;
  END IF;
END IF;
```

Comandos Sequenciais

- **IF-THEN-ELSE:**

  Exemplo:

```
IF na = 3 THEN
    s0 <= i0 OR i1;
    s1 <= i3;
ELSIF na = 8 OR y = '1' THEN
    s0 <= i1;
    s1 <= i4;
ELSE
    s0 <= i0 AND i1;
    s1 <= i5;
END IF;
```

Comandos Sequenciais

- **IF-THEN-ELSE:**

  Exemplo: Multiplexador

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

ENTITY mux2to1 IS
PORT (D0, D1, X :  IN STD_LOGIC;
          S:              OUT STD_LOGIC
           );
END mux2to1;
```

```
ARCHITECTURE behavior OF mux2to1 IS
BEGIN;
PROCESS (D0, D1, X)
BEGIN
      IF X = '0' THEN
          S <= D0;
      ELSE
          S <= D1;
      END IF;
END PROCESS;
END behavior;
```

Comandos Sequenciais

- **CASE**
- **Seleciona a execução que ocorrerá de uma lista de alternativas;**
- **Equivale ao comando WITH-SELECT.**

```
CASE <seleção> IS
   WHEN <condicao1> =>
        <comandos>;
   WHEN <condicao2> TO <condicao3> =>
        <comandos>;
   WHEN <condicao4> | <condicao5> =>
        <comandos>;
   WHEN others =>
        <comandos>;
END CASE;
```

Comandos Sequenciais

- **CASE:**
- **EXEMPLO**

```
CASE na IS
   WHEN 3           =>
            s0 <= i0 OR i1;
            s1 <= i3;
   WHEN 7 TO 12 =>
            s0 <= i1;
            s1 <= i4;
   WHEN OTHERS  =>
            s0 <= i0 AND i1;
            s1 <= i5;
END CASE;
```

Comandos Sequenciais

- **CASE:**

```
PROCESS (sel, en)
BEGIN
y <= "11111111";
IF (en = '1') THEN
    CASE sel IS
        WHEN "000" => y(0) <= '0';
        WHEN "001" => y(1) <= '0';
        WHEN "010" => y(2) <= '0';
        WHEN "011" => y(3) <= '0';
        WHEN "100" => y(4) <= '0';
        WHEN "101" => y(5) <= '0';
        WHEN "110" => y(6) <= '0';
        WHEN "111" => y(7) <= '0';
    END CASE;
END IF;
END PROCESS;
```

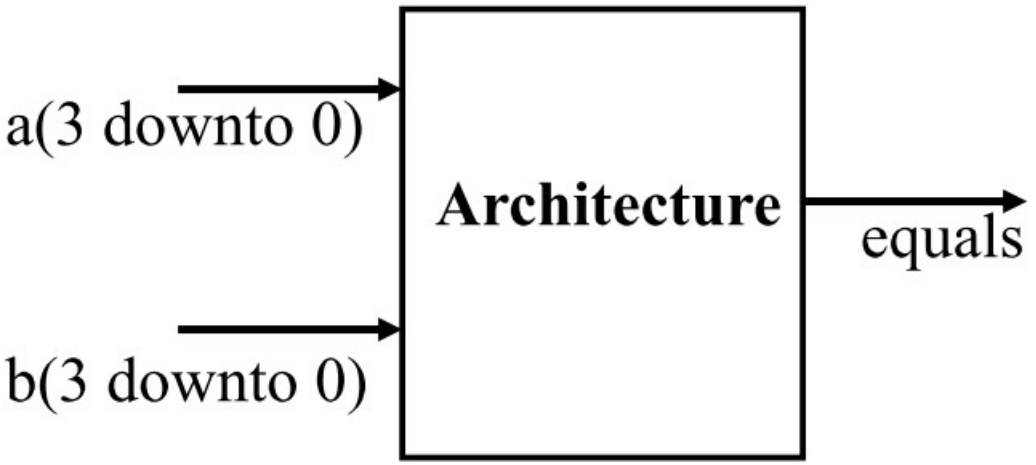
- Ao utilizar processos, um erro comum é esquecer de atribuir a uma saída um valor default. Todas as saídas devem ter valores defaults.
- Neste exemplo, se não fosse atribuído a y o valor default "11111111", nenhum valor seria atribuído a y caso en = 0.

Comandos Sequenciais

- **NULL**
- Não realiza nenhuma operação;
- A execução é passada para o próximo comando;
- Especialmente útil na construção CASE WHEN que precisa cobrir todos os valores da expressão de escolha mas para alguns valores não deve ser feito nada em um dado projeto

EXEMPLOS

Comparador de 4 bits



se a = b então
Equals = 1
senão
Equals = 0

EXEMPLOS

Comparador de 4 bits

Descrição Comportamental

```
-- comparador de 4 bits
entity comp4 is
    port (     a, b: in bit_vector (3 downto 0);
               equals: out bit       );
end comp4;

architecture comport of comp4 is
begin
   comp: process (a,b) -- lista de sensibilidade
   begin
        if a = b then
               equals  < = '1' ;
        else
               equals < = 'O' ;
        end if;
   end process comp;
end comport;
```

EXEMPLOS

Comparador de 4 bits

Descrição por Data-Flow

```
-- comparador de 4 bits
entity comp4 is
    port (   a, b: in bit_vector (3 downto 0);
             equals: out bit          );
end comp4;

architecture fluxo of comp4 is
begin
   equals < = '1' when (a=b) else '0';
end fluxo;
```

EXEMPLOS

Comparador de 4 bits

Descrição Estrutural

```
-- comparador de 4 bits
entity comp4 is
    port (     a, b: in bit_vector (3 downto 0);
               equals: out bit       );
end comp4;

architecture estrut of comp4 is
signal x: bit_vector (3 downto 0);
component xnor is port (a, b: in bit;  equals: out bit );
end component;
begin
   U0: xnor port map (a(0), b(0), x(0));
   U1: xnor port map (a(1), b(1), x(1));
   U2: xnor port map (a(2), b(2), x(2));
   U3: xnor port map (a(3), b(3), x(3));
   U4: and4 port map (x(0), x(1), x(2), x(3), equals);
end estrut;
```

COMPONENTES

- É uma entidade de projeto empregada na arquitetura de uma outra entidade;

- A utilização desses elementos permite a interligação de múltiplas entidades de projeto, de modo a formar uma entidade mais complexa em um projeto hierárquico.

COMPONENTES

- Para a utilização é necessária a declaração do componente:
  - Empregando a palavra COMPONENT no lugar de ENTITY

```
COMPONENT nome_componente_x
        GENERIC (n      : tipo_n := valor;  -- lista de genéricos
        PORT ( sinal_a  : modo_a            tipo_sinal_a;-- lista de portas
               sinal_b  : modo_b            tipo_sinal_b;
               sinal_c  : modo_c            tipo_sinal_c);
END COMPONENT;
```

COMPONENTES

SOLICITAÇÃO DE COMPONENTES

- A solicitação é um comando concorrente que consiste em um rótulo, o nome do componente e o mapa de portas (PORT MAP);

```
-- rotulo   nome          mapa        lista
     x1:    componente_x              PORT MAP( a(1), b(1), c(1) );              -- posicional
     x2:    componente_x              PORT MAP( sianl_b => b(1), sinal_a => a(1),
                                                sinal_c => c(1) );              -- nomeada
     x3:    componente_y              PORT MAP( sinal_x => x(1), sinal_y => OPEN ) -- nomeada
```

COMPONENTES

SOLICITAÇÃO DE COMPONENTES

- **Posicional:**
  - A lista de sinais no mapa deve seguir a mesma ordem estabelecida na declaração do componente;

- **Nomeada:**
  - Uma nova sequência definida no mapa;

  - A palavra **OPEN** indica que o sinal do componente não é conectado a nenhum sinal da arquitetura

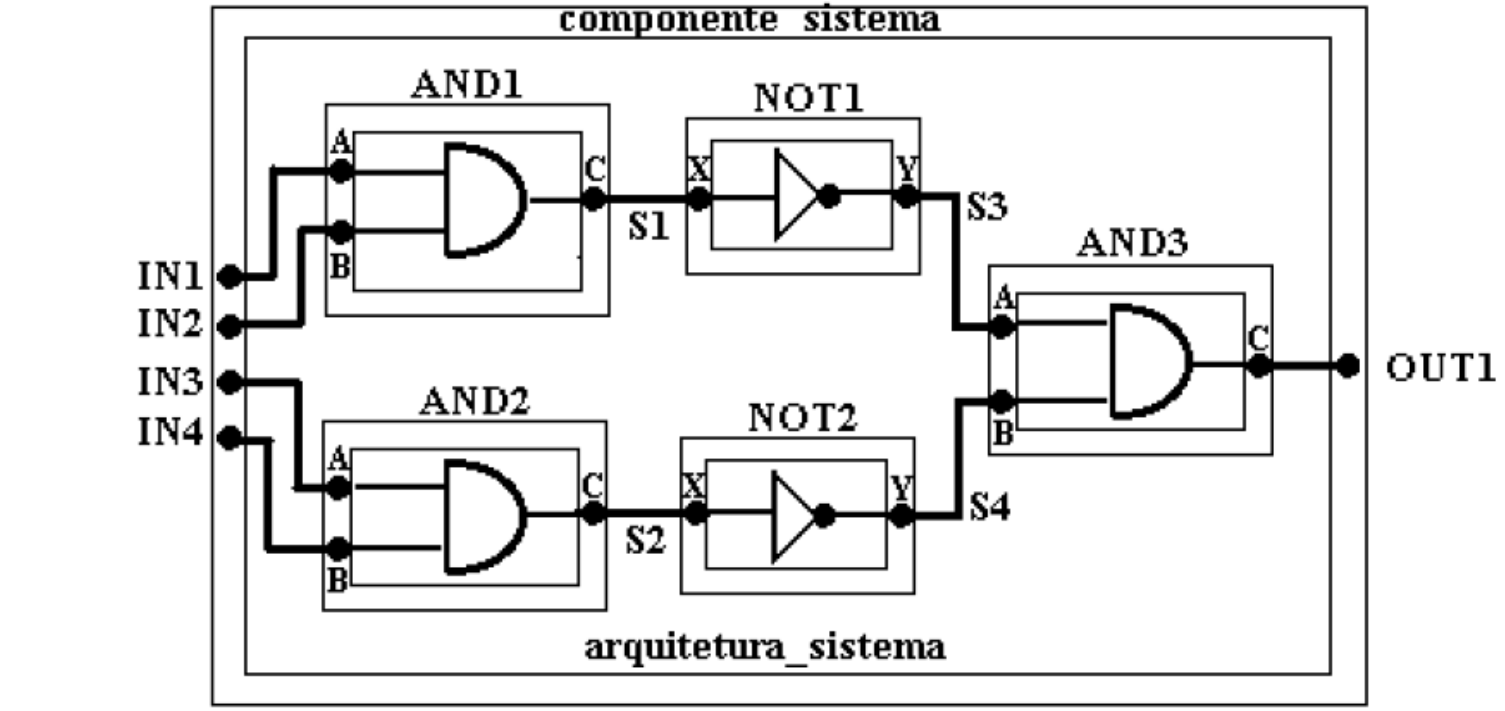
COMPONENTES

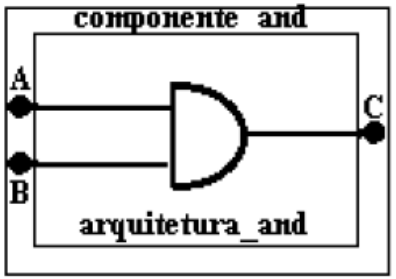
Especificando a Estrutura de um Sistema

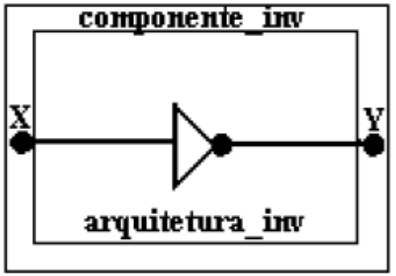
- O **component** é exatamente a descrição de um componente;
- O **port map** é um mapeamento deste componente em um sistema maior.

COMPONENTES

Especificando a Estrutura de um Sistema







COMPONENTES

Especificando a Estrutura de um Sistema

Programa 1

```
-------------------------------------------------------
-- Arquivo componente_inv.vhd
-- Modelo do inversor
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_inv is
port(
        x : in std_logic;
        y : out std_logic
        );
end componente_inv;

architecture arquitetura_inv of compo-
nente_inv is

        begin
                y <= not x;
end arquitetura_inv;
```

Programa 2

```
-------------------------------------------------------
-- Arquivo componente_and.vhd
-- Modelo da porta AND
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_and is
port(
        a : in std_logic;
        b : in std_logic;
        c : out std_logic
        );
end componente_and;

architecture arquitetura_and of com-
ponente_and is

        begin
                c <= a and b;
end arquitetura_and;
```

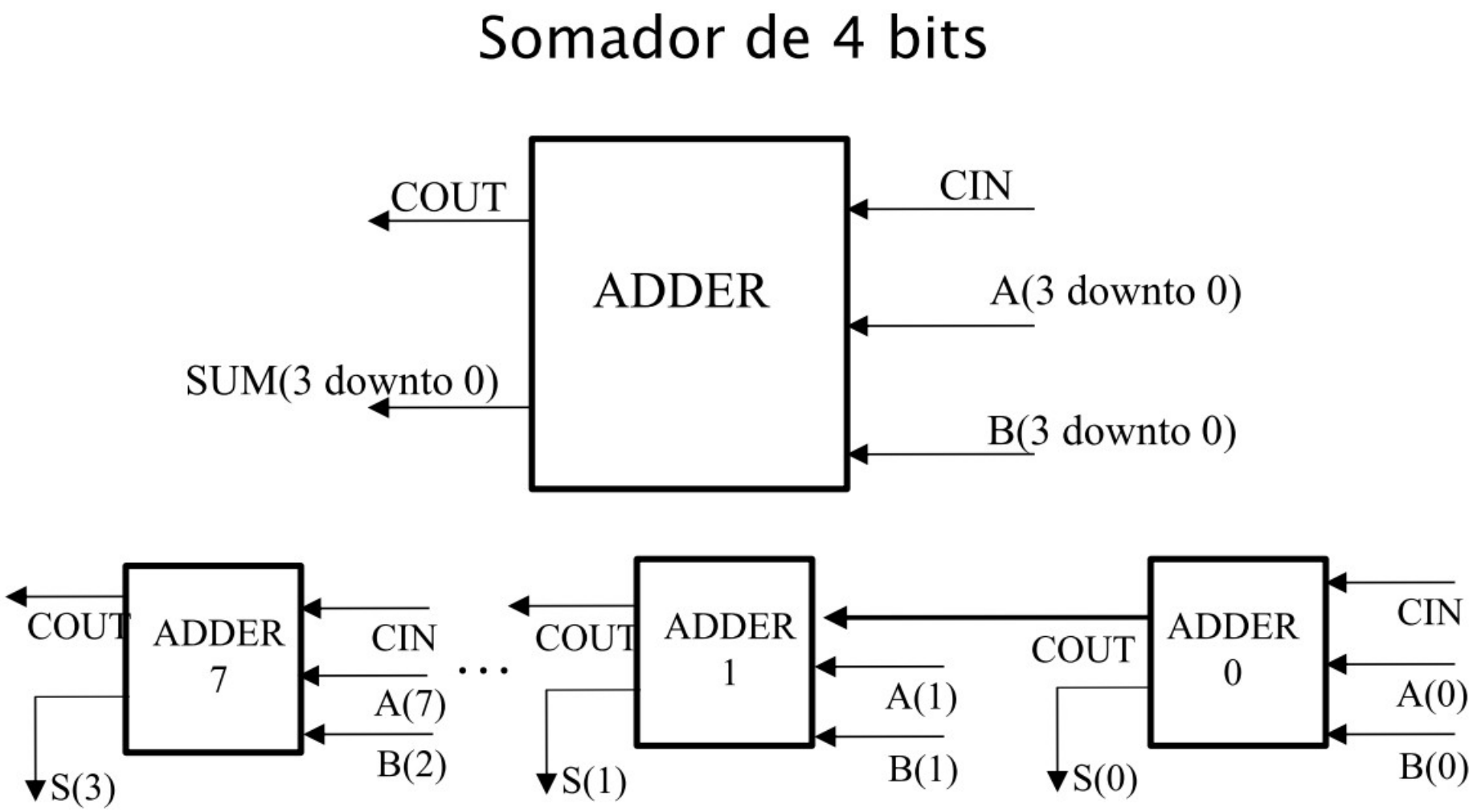
COMPONENTES

Especificando a Estrutura de um Sistema

Programa 3

```
-------------------------------------------------------
-- Arquivo componente_sistema.vhd
-------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity componente_sistema is
port(
        in1 : in std_logic;
        in2 : in std_logic;
        in3 : in std_logic;
        in4 : in std_logic;
        out1 : out std_logic
        );
end componente_sistema;
architecture arquitetura_sistema of componente_sistema is

component componente_and
                port( a: in std_logic; b : in std_logicbit; c : out std_logic);
end component;

component componente_inv
                port( x: in std_logic; y : out std_logic);
end component;

signal s1, s2, s3, s4 : std_logic;

        begin
                and1 : componente_and port map (a => in1, b => in2, c => s1);
                and2 : componente_and port map (a => in3, b => in4, c => s2);
                and3 : componente_and port map (a => s3, b => s4, c => ut1);
                inv1 : componente_inv port map (x => s1, y => s3);
                inv2 : componente_inv port map (x => s2, y => s4);
end arquitetura_sistema;
```

EXEMPLOS


Somador de 4 bits
COUT
CIN
ADDER
A(3 downto 0)
SUM(3 downto 0)
B(3 downto 0)
COUT
ADDER 7
CIN
A(7)
B(2)
S(3)
COUT
ADDER 1
A(1)
B(1)
S(1)
COUT
ADDER 0
CIN
A(0)
B(0)
S(0)

EXEMPLOS

Somador de 4 bits

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

ENTITY adder_4bits IS
  PORT (
         cin: IN std_logic;
         a: IN std_logic_vector(3 DOWNTO 0);
         b: IN std_logic_vector(3 DOWNTO 0);
         sum: OUT std_logic_vector(3 DOWNTO 0);
         cout: OUT std_logic
    );
END adder_4bits;
```

EXEMPLOS

```
BEGIN
  FULL_ADDER_0: full_adder
    PORT MAP (cin => cin, a => a(0), b => b(0), sum =>
              sum(0), cout => c(0));
  FULL_ADDER_1: full_adder
    PORT MAP (cin => c(0), a => a(1), b => b(1), sum =>
              sum(1), cout => c(1));
  FULL_ADDER_2: full_adder
    PORT MAP (cin => c(1), a => a(2), b => b(2), sum =>
              sum(2), cout => c(2));
  FULL_ADDER_3: full_adder
    PORT MAP (cin => c(2), a => a(3), b => b(3), sum =>
              sum(3), cout => cout);
END structure;
```

EXEMPLOS

FLIP-FLOP D

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dff_logic IS
PORT (d, clk : IN BIT;
      q : OUT BIT);
END dff_logic;

ARCHITECTURE behavior OF dff_logic IS
BEGIN
  PROCESS(clk)
  BEGIN
      IF (clk'event AND clk = '1') THEN
            q <= d;
      END IF;
  END PROCESS;
END behavior;
```

EXEMPLOS

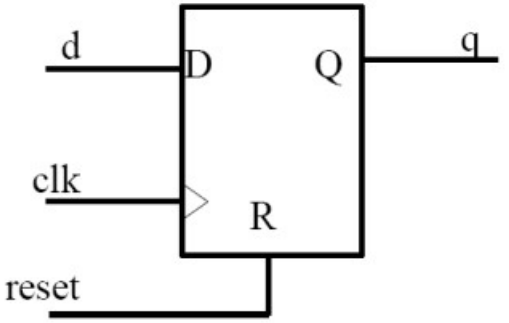
FLIP-FLOP D

- A condição “clk” event é ativada quando ocorre variação no valor de clk;
- A utilização de “clk” event em conjunto com a lista de sensibilidade é redundante, mas necessária devido ao fato de que algumas ferramentas de síntese ignoram a lista de sensibilidade;
- Uma mudança do clock pode ocorrer de “0” para “1“ ou de “1” para “0”, desse modo é necessária a condição adicional clk = “1’ ou clk = “0”;
- Em STD_LOGIC, usa-se rising_edge para transição de subida de clock e falling_edge para transição de descida.

EXEMPLOS

Circuito Assíncrono - RESET

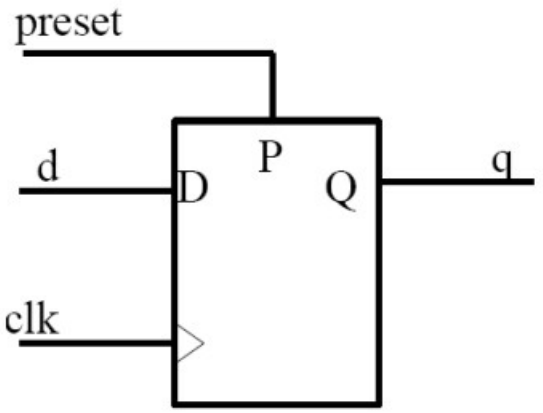
```
ARCHITECTURE behavior OF dff_logic IS
BEGIN
  PROCESS(clk, reset)
  BEGIN
      IF reset = '1' THEN
            q <= (others => '0');
      ELSIF RISING_EDGE(clk) THEN
            q <= d;
      END IF;
  END PROCESS;
END behavior;
```



EXEMPLOS

Circuito Assíncrono - PRESET

```
ARCHITECTURE behavior OF dff_logic IS
BEGIN
  PROCESS(clk, preset)
  BEGIN
      IF preset = '1' THEN
            q <= (others => '1');
      ELSIF RISING_EDGE(clk) THEN
            q <= d;
      END IF;
  END PROCESS;
END behavior;
```



Referencias

- ALTERA. DE2 Development and education board user manual. 2008. Version 1.42.
- ALTERA. Quartus II Introduction Using VHDL Design. 2008.
- MENEZES, M.P.; SATO, L.M.; MIDORIKAWA, E.T. Projeto de Circuitos com Quartus II 9.1.
- Apostila de Laboratório Digital. Departamento de Engenharia de Computação e Sistemas Digitais,
- Escola Politécnica da USP. Edição de 2011.
- TOCCI, R. J.; WIDMER, N.S.; MOSS, G.L. Sistemas Digitais: Princípios e Aplicações.
- Prentice-Hall, 11 ed., 2011.
- WAKERLY, John F. Digital Design Principles & Practices. 4th edition, Prentice Hall, 2006.
- A. C. Leonardo; S.S. Ivan. Minicurso tópicos em vhdl. 2010.