



# **UNIVERSIDADE FEDERAL DE RORAIMA**

## **Introdução ao VHDL**

**Prof. Herbert Oliveira Rocha**



# **UNIVERSIDADE FEDERAL DE RORAIMA**

## **Introdução ao VHDL**

Baseado nas aulas do Prof. Dr. Mauricio Figueiredo and  
Prof. Dr. Raimundo Barreto - UFAM

**Prof. Herbert Oliveira Rocha**

# Visão Geral

- VHDL (VHSIC – **Very High Speed Integrated Circuit Hardware Description Language**) é uma linguagem de descrição de hardware amplamente utilizada no desenvolvimento dos sistemas digitais atuais.
- Criada pelo depto. De Defesa americano em 1980.
- Primeira linguagem de descrição de hardware **padronizada pelo IEEE (1076 e 1164)**.
- Ferramentas:
  - Intel Quartus (síntese e simulação gráfica)
  - Xilinx (síntese e simulação gráfica)
  - Outras
- Referências:
  - Eletrônica Digital Moderna e VHDL, Pedroni.
  - VHDL Programming by examples, Perry
  - VHDL Handbook, Hardi

# Visão Geral

- Very high speed integrated circuits Hardware Description Language (VHDL)
- Permite a descrição da estrutura de um projeto, ou seja, como é decomposto em sub-projetos e como estes são interconectados
- Permite a especificação da função do projeto usando formas familiares de linguagens de programação
- Permite um projeto ser simulado antes de ser manufaturado de tal forma que projetistas possam rapidamente comparar alternativas e testar sem o atraso e custo da prototipação em hardware

# Visão Geral

- Três partes:
  - Bibliotecas/pacotes: Componentes acessórios básicos e reuso.
  - Entity: Especifica interfaces
  - Arquitetura: Código, propriamente dito
- Entity:
  - Duas seções:
  - PORT: para entradas e saídas do circuito. Modos:
    - IN - entrada
    - INOUT – bidirecional
    - BUFFER – saída que pode ser usada internamente
  - Tipos de sinais: BIT, BIT\_VECTOR, STD\_LOGIC, STD\_LOGIC\_VECTOR, BOOLEAN, INTEGER, etc.
  - GENERIC: Constantes genéricas, globais.

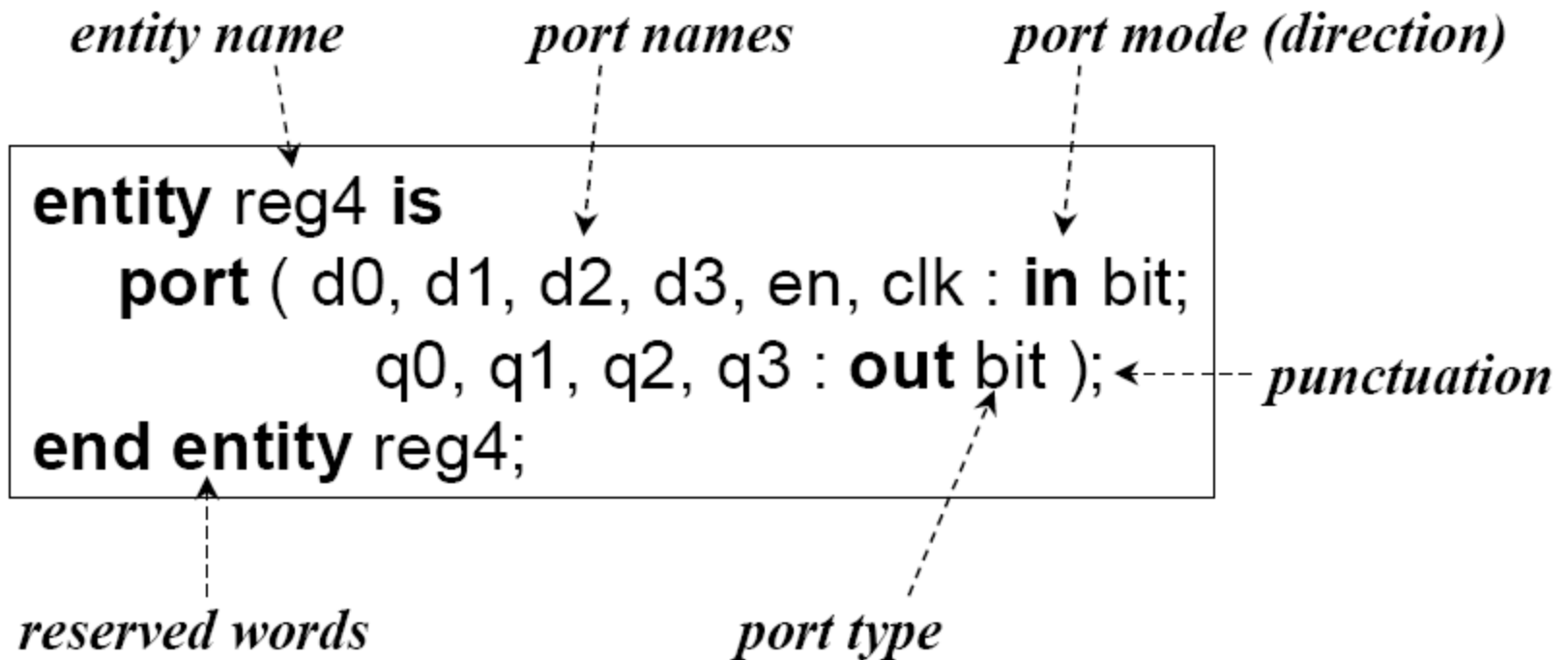
# Visão Geral

## Arquitetura

- Tem uma parte declarativa usada normalmente para definir como `TYPE`, `SIGNAL`, `CONSTANT`, `COMPONENT` e `FUNCTION`.
- Tem uma parte de código, que pode conter código:
  - Concorrente: Circuitos combinacionais
  - Sequencial: Circuitos Sequenciais ou combinacionais
- Todas as declarações de um código são concorrentes, a menos que seja declarada dentro de um processo (`PROCESS`). Um processo é concorrente com o restante do programa.

# Modelando

- Descreve as portas de entrada e saída de um módulo



# Modelando

- **Corpo da Arquitetura**
  - Descreve a implementação de uma entidade
- **Arquitetura Comportamental**
  - Descreve o algoritmo executado pelo módulo
  - Contém instruções cada uma contendo instruções sequenciais, atribuição de sinais e instrução wait.



# Modelando

```
architecture behav of reg4 is
begin
    storage : process is
        variable stored_d0, stored_d1, stored_d2, stored_d3 : bit;
    begin
        if en = '1' and clk = '1' then
            stored_d0 := d0;
            stored_d1 := d1;
            stored_d2 := d2;
            stored_d3 := d3;
        end if;
        q0 <= stored_d0 after 5 ns;
        q1 <= stored_d1 after 5 ns;
        q2 <= stored_d2 after 5 ns;
        q3 <= stored_d3 after 5 ns;
        wait on d0, d1, d2, d3, en, clk;
    end process storage;
end architecture behav;
```

# Modelando

## Arquitetura Estrutural

- Implementa os módulos como composição de sub-sistemas
- Contém
  - *Declaração de sinais para interconexões internas*
    - As portas das entidade são tratadas como sinais
  - Instâncias de Componentes
    - instâncias de entidades/arquiteturas previamente declaradas
  - *Port maps em instâncias de componentes*
    - Conecta os sinais em portas de componentes
  - *Instruções wait*

# Modelando

- Processo
- Algoritmo(Processo):
  - Sequencial
- Lista de Sensibilidade:
  - Sinais de entrada
- Região declarativa:
  - Região entre o fim da lista de sensibilidade e a palavra chave begin.
  - Usada para declarar variáveis ou constantes dentro do processo.
- Campo de atribuições:
  - Campo entre a chave begin e end ALG;

```
entity compare is
    port ( A, B: in bit;
           C : out bit);
end compare;
architecture ALG of compare is
begin
    process(A,B)
    begin
        if(A = B) then
            C <= '1';
        else
            C <= '0';
        end if;
    end process;
end ALG;
```

# Bibliotecas Básicas

Algumas comuns:

- LIBRARY ieee;
- USE ieee.std\_logic\_1164.all; (Op lógicos e conversão de tipos)
- USE ieee.std\_logic\_arith.all; (Tipos e ops aritméticos)
- etc.

# Tipos

- **BIT**: assume valores de 0 ou 1
  - SIGNAL valor: bit;
- **BOOLEAN**: usado em teste de decisão, assume os valores true ou false.
  - SIGNAL teste: boolean;
- **INTEGER**: assume um valor inteiro entre -2.147.483.647 e +2.147.483.67.
  - SIGNAL valor: integer range 0 to 10;
- **REAL**: representa um número de ponto flutuante.
- Outros

# Tipos

- STD\_LOGIC: podem assumir dentre outros valores os seguintes: não inicializado(U), alta impedância(Z), nível lógico alto(1) e nível lógico baixo(0).
  - ***SIGNAL valor: std\_logic;***
- Vetores: são conjuntos de sinais previamente definidos, exemplos:
  - ***SIGNAL vetor: std\_logic\_vector(10 DOWNT0 0);***
  - ***SIGNAL vetor: std\_logic\_vector(0 TO 10);***
  - ***SIGNAL vetor: bit\_vector(10 DOWNT0 0);***
  - ***SIGNAL vetor: bit\_vector(0 TO 10);***

# Operadores

- Lógicos: and, or, xor, not, nand
  - valor := a xor b;
- Relacionais: =, />, <, >, <=, >=
  - IF (a /= b) THEN;
  - A <= B;
  - **obs.: cuidado com o operador <=, pois ele também é um operador de atribuição.**
- Deslocamento: sll, srl, sla, rol, ror
  - sinal\_a <= x ror 3; (circular)
  - z <= x sll 1;

# Operadores

- Adição: +, -, &
  - valor := a + b;
  - valor := "00" & "11"; (concatenação: "0011")
- Sinal: +, -
  - A <= -B;
- Multiplicação e divisão: \*, /, mod, rem
  - a\_div\_b := a / b;
- Diversos: \*\*, abs, MOD
  - sinal\_abs <= abs( -3); (absoluto)
  - valor := b\*\*2; (exp)



# Decisão

If then else:

```
IF (condição) THEN código;
```

```
Else código;
```

```
END IF;
```

Case When:

```
Case estados is
```

```
    when estado1 => código;
```

```
    when estado2 => código;
```

```
    when others => código;
```

```
END CASE;
```

# Decisão

```
- SIGNAL sinalA: STD_LOGIC_VECTOR(1 DOWNTO 0);  
  CASE sinalA IS  
    WHEN "00" => código;  
    WHEN "01" => código;  
    WHEN OTHERS => pode ter ou não operações;  
  END CASE;
```

- With select:

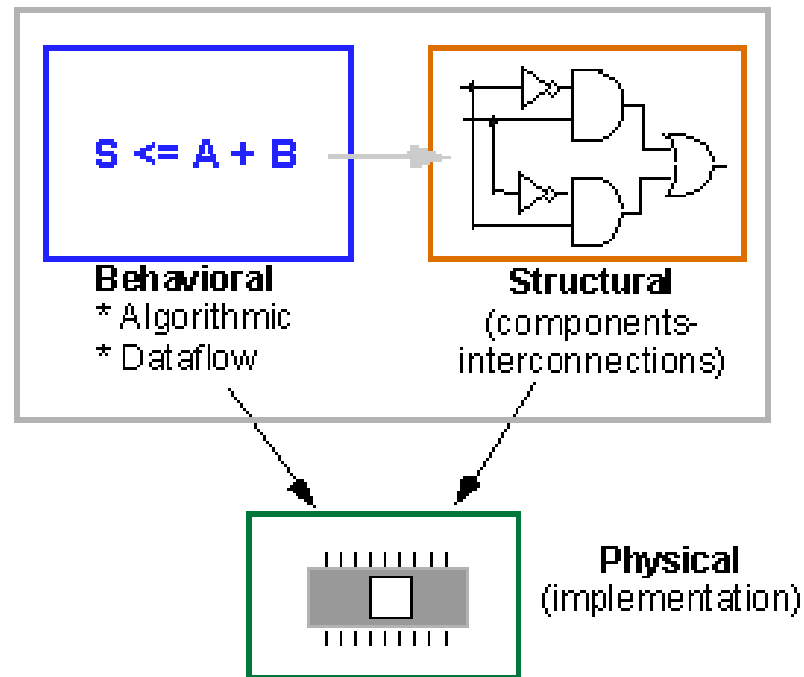
```
WITH expressao_escolha SELECT  
sinal <= expressao_a WHEN condicao1,  
        expressao_b WHEN condicao2,  
        expressao_c WHEN condicao3|condicao4,  
        expressao_d WHEN OTHERS;
```


# Loops

- Múltiplas instâncias da mesma atribuição




```
FOR i IN xrange LOOP  
    x(i) <= a(M-i) AND b(i)  
END LOOP
```

# Praticando!!!






PRODUCTS   SUPPORT   SOLUTIONS   DEVELOPERS   PARTNERS

  USA (ENGLISH)    Search Intel.com

Intel® FPGA Development Tools

# Intel® Quartus® Prime Software Suite

The Intuitive High-Performance Design Environment.



Overview

Features & Download

What's New

Videos

Documentation and Support

## Features and Downloads

To compare different Intel® Quartus® Prime Editions, please visit the [getting started](#) page.

### Intel® Quartus® Prime Pro Edition

The Intel® Quartus® Prime Pro Edition Software is optimized to support the advanced features in next-generation FPGAs and SoCs with the Intel® Agilex™, Intel® Stratix® 10, Intel® Arria® 10, and Intel® Cyclone® 10 GX device families.

[Download now \(paid license required\) →](#)

### Intel® Quartus® Prime Standard Edition

The Intel® Quartus® Prime Standard Edition Software includes extensive support for earlier device families in addition to the Intel® Cyclone® 10 LP device family.

[Download now \(paid license required\) →](#)

### Intel® Quartus® Prime Lite Edition

The Intel® Quartus® Prime Lite Edition software provides an ideal entry point to high-volume device families and is available as a free download with no license file required.

[Download now \(free, no license required\) →](#)

[Home](#) > [Downloads](#) > Quartus Prime Lite Edition

## Download Center for FPGAs

Design Software

Embedded Software

Archives

Licensing

Programming Software

Drivers

Board System Design

Board Layout and Test

Legacy Software

### Quartus Prime Lite Edition

Release date: November, 2020

Latest Release: v20.1.1

Select edition: Lite ▼

Select release: 20.1.1 ▼


Operating System   Windows  Linux


**Intel® Quartus® Prime**


Design Software

# Online Course

- <https://www.intel.com/content/www/us/en/programmable/support/training/course/ohdl1110.html>
- [https://www.vhdl-online.de/vhdl\\_workshop/start](https://www.vhdl-online.de/vhdl_workshop/start)

 VHDL-Online   On-Site Training   Contact   Legal notice   Privacy

Search 

Log In 

[[vhdl\_workshop:start]]

✓ VHDL Tutorial

✓ VHDL-AMS

VHDL Workshop

VHDL Reference

VHDL Glossary

VHDL Library

## VHDL Workshop: Camera

- LAB 1: A Multiplexer
- LAB 2: Extending the Multiplexer
- LAB 3: A 7-Segment Display Driver
- LAB 4: A Three Digit 7-Segment Display Driver
- LAB 5: A Decoder
- LAB 6: A Register
- LAB 7: A State Machine for the Display
- LAB 8: A Timer
- LAB 9: A BCD-Counter
- LAB 10: A State Machine for the Main Controller
- LAB 11: The Camera

### Introduction

This VHDL teaching program is divided into several exercises. By writing typical VHDL programs you learn how to use this hardware description language. In the beginning you are guided in a step by step manner, later on when you are used to the language and the tools you will solve the tasks on your own. We are referring to the VHDL'87 standard, because the VHDL'93 is not yet supported by all tool manufacturers.

### Structure of the Exercises

The tasks are divided into different sections:

- **Synopsis:** In this section you learn about the goal of the task and the function of the model within the whole design.
- **Behaviour:** This section tells you more about the function of the model.
- **Data Types:** In this section the types to be used in the current exercise are described.
- **To Do:** Here is the precise description of your job to solve the task.
- **Implementation:** Here you find a list of VHDL code which represents a possible implementation.

Table of Contents ▾

VHDL Workshop: Camera

Introduction

VHDL Working Environment

# Não tenho o Quartus! E agora?

<https://www.edaplayground.com>

The screenshot displays the EDA Playground web interface. On the left sidebar, there are sections for 'Languages & Libraries' (showing VHDL selected), 'Tools & Simulators' (showing Aldec Riviera Pro 2020.04 selected), and 'Examples' (showing 'using EDA Playground VHDL'). The main area is split into two panels: 'testbench.vhd' on the left and 'design.vhd' on the right. The 'testbench.vhd' panel contains a testbench for an OR gate, including component declarations, port mappings, and assertion checks. The 'design.vhd' panel contains the RTL logic for a simple OR gate.

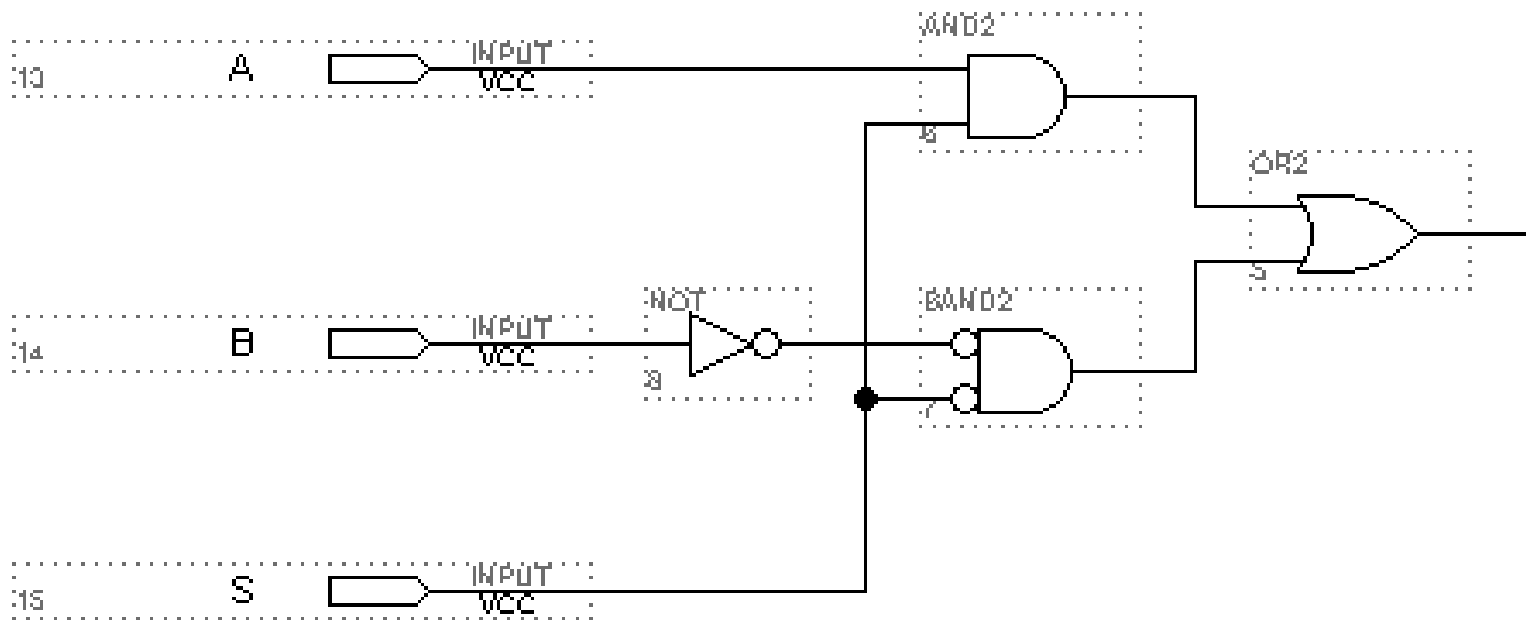
```
1 -- Testbench for OR gate
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4
5 entity testbench is
6   -- empty
7 end testbench;
8
9 architecture tb of testbench is
10
11   -- DUT component
12   component or_gate is
13   port(
14     a: in std_logic;
15     b: in std_logic;
16     q: out std_logic;
17   end component;
18
19   signal a_in, b_in, q_out: std_logic;
20
21   begin
22
23     -- Connect DUT
24     DUT: or_gate port map(a_in, b_in, q_out);
25
26     process
27     begin
28       a_in <= '0';
29       b_in <= '0';
30       wait for 1 ns;
31       assert(q_out='0') report "Fail 0/0" severity error;
32
33       a_in <= '0';
34       b_in <= '1';
35       wait for 1 ns;
36       assert(q_out='1') report "Fail 0/1" severity error;
37
38       a_in <= '1';
39       b_in <= 'X';
40       wait for 1 ns;
41       assert(q_out='1') report "Fail 1/X" severity error;
```

```
1 -- Simple OR gate design
2 library IEEE;
3 use IEEE.std_logic_1164.all;
4
5 entity or_gate is
6   port(
7     a: in std_logic;
8     b: in std_logic;
9     q: out std_logic;
10  end or_gate;
11
12 architecture rtl of or_gate is
13 begin
14   process(a, b) is
15   begin
16     q <= a or b;
17   end process;
18 end rtl;
```



# Exemplo 1

## Multiplexador 2x1



# Exemplo 1

## Multiplexador 2x1

```
ENTITY Multiplex IS
    PORT (
        A, B, S    : IN BIT;
        Saida      : OUT   BIT
    );
END Multiplex;
```

# Exemplo 1

## Multiplexador 2x1

```
ARCHITECTURE Objeto_1 OF Multiplex IS
BEGIN
    Saida <= (A AND S) OR (B AND NOT(S));
END Objeto_1;
```

**Método 1**

# Exemplo 1

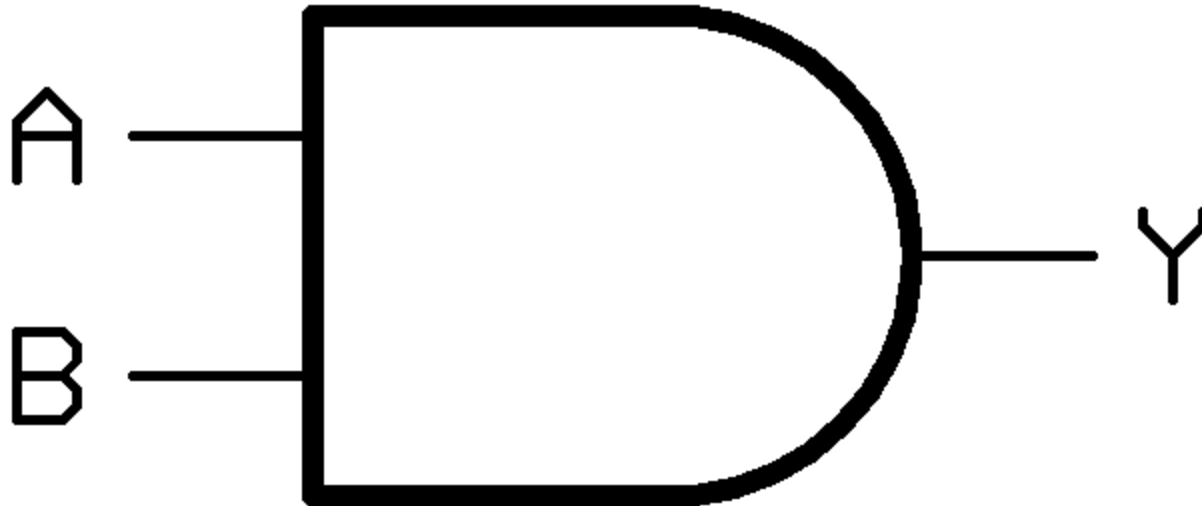
## Multiplexador 2x1

```
ARCHITECTURE Objeto_1 OF Multiplex IS
BEGIN
    Process (A,B,S) is
    Begin
        If S='1' Then Saida <= A;
        Else Saida <=B;
        End If;
    End Process;
END Objeto_1;
```

**Método 2**

## Exemplo 2

And



## Exemplo 2

### And

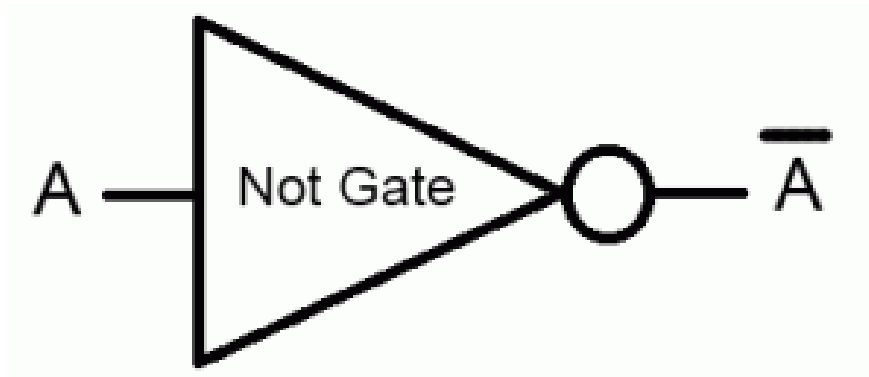
```
Library ieee;
use ieee.std_logic_1164.all;

Entity QAnd is
Port(
    A, B : IN STD_LOGIC;
    Result : OUT STD_LOGIC
);
End QAnd;

Architecture behavior of QAnd is
BEGIN
    Result <= A and B;
END behavior;
```

## Exemplo 3

Not



## Exemplo 3

### Not

```
library ieee;
use ieee.std_logic_1164.all;

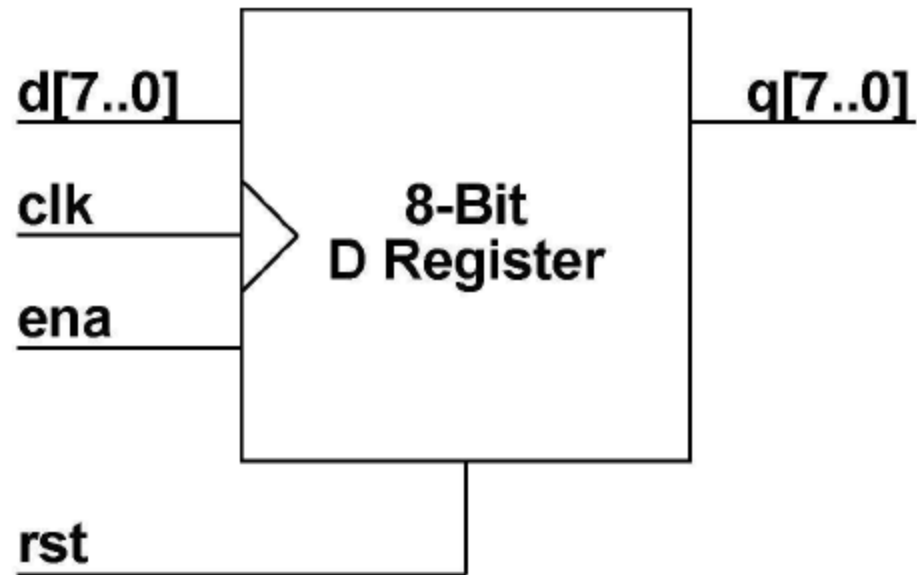
entity notGate is
    port(
        inPort    : in std_logic;
        outPort   : out std_logic
    );
end notGate;

architecture func of notGate is
begin
    outPort <= not inPort;
end func;
```



## Exemplo 4

### Registrador 8-bits Sequencial



## Exemplo 4

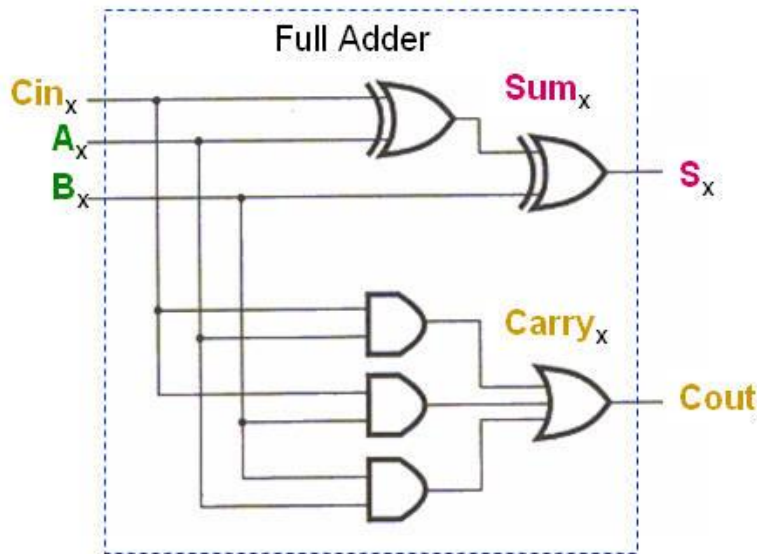
### Registrador 8-bits Sequencial

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dregister IS
    PORT( rst, clk, ena      : IN      std_logic;
          d                  : IN      std_logic_vector(7 DOWNTO 0);
          q                  : OUT     std_logic_vector(7 DOWNTO 0) );
END dregister;

ARCHITECTURE synthesis1 OF dregister IS
BEGIN
    PROCESS (rst, clk)
    BEGIN
        IF (rst = '1') THEN
            q <= X"00";
        ELSIF (clk'EVENT) AND (clk = '1') THEN
            IF (ena = '1') THEN
                q <= d;
            END IF;
        END IF;
    END PROCESS;
END synthesis1;
```

## Tente outros como:



$Cin$	1	1	1	1	0	0	0	0
$A$	0	1	0	1	1	1	0	0
$B$	0	0	1	1	1	0	1	0
$Cout$	0	1	1	1	1	0	1	0
$S$	0	1	0	0	1	0	1	0

## Banco de Registradores

