

# Laboratório 1 – Portas Lógicas

## Objetivos:

1. Treinar os conceitos de portas lógicas, circuitos lógicos, funções booleanas e tabelas da verdade;
2. Provar algumas das propriedades da álgebra booleana;
3. Treinar a programação em VHDL de entidades básicas.

## Introdução Teórica:

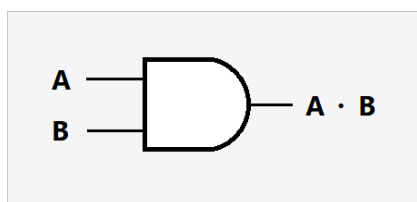
Em operações booleanas, temos apenas dois estados distintos da informação:

- O estado 0 (zero) e
- O estado 1 (um)

Comumente, atribui-se o estado 0 para representar as informações: “Desligado, falso, chave aberta, ausência de tensão etc. O estado 1 representa as informações opostas: ligado, chave fechada, presença de tensão, verdadeiro etc.

As portas lógicas que operam esses estados são as portas E, OU e a porta não, que serão descritas a seguir:

### Porta Lógica: “E (AND)”



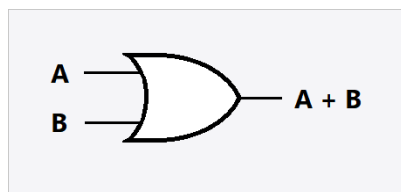
A função lógica AND produz 1 quando **ambos** os operandos são 1. Desse modo, o resultado de  $S = A \text{ and } B$  ou ainda  $S = A \cdot B$  só é 1 quando  $A = 1$  e  $B = 1$ , caso contrário o resultado dessa operação é 0.

Chama-se **tabela verdade**, ou ainda, tabela da verdade, um mapa, onde se representa todas as possibilidades das portas lógicas e seus respectivos resultados. Na tabela, iremos encontrar a descrição como a porta funciona. A tabela 1, mostra a tabela verdade para operação  $S = A \cdot B$ , que se lê, S é igual A e B.

Tabela 1: Tabela Verdade da Porta **AND**

| A | B | $S = A \cdot B$ |
|---|---|-----------------|
| 0 | 0 | 0               |
| 0 | 1 | 0               |
| 1 | 0 | 0               |
| 1 | 1 | 1               |

## Porta Lógica: “OU (OR)”



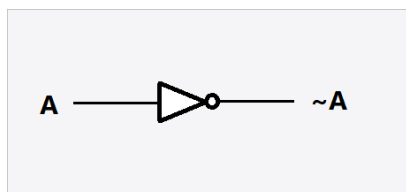
A função lógica OR produz 1 quando **qualquer** um dos operandos tem valor 1. Desse modo, o resultado de  $S = A \text{ or } B$  ou ainda  $S = A + B$  é 1 quando  $A = 1$  ou  $B = 1$ , caso contrário o resultado dessa operação é 0.

A tabela 2, mostra a tabela verdade para operação  $S = A + B$ , que se lê, S é igual A **ou** B.

Tabela 2: Tabela Verdade da Porta **OR**

| A | B | $S = A + B$ |
|---|---|-------------|
| 0 | 0 | 0           |
| 0 | 1 | 1           |
| 1 | 0 | 1           |
| 1 | 1 | 1           |

## Porta Lógica: “Não (NOT)”



A função lógica NOT produz 1 quando o valor de  $A = 0$  e produz 0 quando o valor de  $A = 1$ . Desse modo, a porta **NOT** funciona como um inversor. O resultado de  $S = \text{not}(A)$  ou ainda  $S = A'$  é 1 quando  $A = 0$ , e 0 quando  $A = 1$ .

A tabela 3, mostra a tabela verdade para operação  $S = A'$ , que se lê, S é igual **não** A.

Tabela 3: Tabela Verdade da Porta **NOT**

| A | $S = A'$ |
|---|----------|
| 0 | 1        |
| 1 | 0        |

### Postulados:

A seguir serão apresentados os postulados de identidade para as operações de Adição (Ou), Multiplicação (E) e Negação (Não)

#### Identidade:

1.  $A + 0 = A$
2.  $A \cdot 1 = A$
3.  $(A')' = A$

### Propriedades das Operações Booleanas:

A lógica booleana tem algumas propriedades que são respeitadas pelas portas AND, OR e NOT, essas propriedades são extremamente úteis na simplificação de circuitos e de expressões. Tal como na matemática comum, valem na Álgebra de Boole, propriedades como: comutativa, associativa e distributiva.

#### Comutativa:

Esta propriedade é válida tanto na adição, como na multiplicação:

1. Adição:  $A + B = B + A$ ;
2. Multiplicação:  $A \cdot B = B \cdot A$

#### Associativa:

Esta propriedade é válida tanto na adição, como na multiplicação:

3. Adição:  $A + (B + C) = (A + B) + C$ ;
4. Multiplicação:  $A \cdot (B \cdot C) = (A \cdot B) \cdot C$

#### Distributiva:

$$A \cdot (B + C) = A \cdot B + A \cdot C;$$
$$A + (B \cdot C) = (A + B) \cdot (A + C);$$

### Teoremas de De Morgan

Os teoremas de De Morgan são muito empregados na prática, em simplificações de expressões booleanas e, ainda, no desenvolvimento de circuitos digitais.

São duas leis:

- **1ª Lei de De Morgan:**

$$(A \cdot B)' = A' + B'$$

- **2ª Lei de De Morgan**

$$(A + B)' = A' \cdot B'$$

---

## **Projeto em VHDL:**

A estrutura de um projeto em VHDL consiste em três partes:

|                         |
|-------------------------|
| Library<br>declarations |
| ENTITY                  |
| ARCHITECTURE            |

### **Librarys:**

A primeira parte do código deve conter uma lista com todas as bibliotecas e pacotes que o compilador necessitará para processar o projeto. As bibliotecas std (standard) e work são disponibilizadas automaticamente pela IDE.

A biblioteca STD contém definições para os tipos de dados básicos: BIT, BOOLEAN, BIT\_VECTOR, INT. Outra biblioteca bastante utilizada é a biblioteca **ieee.std\_logic\_1164.all**, que especifica os tipos STD\_LOGIC e STD\_LOGIC\_VECTOR.

É feita da seguinte forma:

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

## ENTITY:

Todo componente VHDL tem que ser definido como uma entidade (entity), o que nada mais é do que uma representação formal de uma simples porta lógica até um sistema lógico completo. Na declaração de uma entidade, descreve-se o conjunto de entradas e saídas que constituem o projeto, é equivalente ao símbolo de um bloco em captura esquemática. As entradas podem ser dos seguintes tipos:

- **BIT**: Assume valores lógicos 0 ou 1;
- **BIT\_VECTOR**: Define um vetor de BITS;
- **INT**: Inteiros

Com a inclusão da biblioteca IEEE:

- **STD\_LOGIC**: Pode assumir os valores mostrados na tabela a seguir;
- **STD\_LOGIC\_VECTOR**: Vetor de STD\_LOGIC que pode assumir os valores mostrados na tabela a seguir:

| Valor  | Estado                            |
|--------|-----------------------------------|
| U      | Não inicializado                  |
| X      | Desconhecido forte                |
| 0 ou 1 | Nível Baixo ou Nível Alto         |
| W      | Desconhecido fraco                |
| Z      | Alta Impedância (circuito aberto) |
| L ou H | 0 ou 1 fracos                     |
| -      | Não importa                       |

O modo de operação dos pinos podem ser:

- **IN**: Configura o pino como entrada;
- **OUT**: Configura o pino como saída;
- **INOUT**: Configura o pino como entrada e saída (Bidirecional)
- **BUFFER**: Pino de saída que pode ser lido internamente pelo código.

A declaração pode ser dos pinos tem que ser da forma:

<NOME\_DO\_PINO> : <MODO\_DE\_OPERAÇÃO> <TIPO\_DO\_PINO>;

**Exemplo:**

A : IN BIT;

A declaração da entidade segue a seguinte sintaxe:

```
ENTITY <nome_da_entidade> IS
    PORT( <nome_do_pino> : <modo> <tipo>
        );
END <nome_da_entidade>;
```

#### Exemplo:

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

Note que a declaração da última variável não tem o “;” do final.

## ARCHITECTURE

A declaração da arquitetura (“architecture”) descreve o comportamento da entidade, define o seu funcionamento interno, isto é, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos. Para tal, utiliza-se uma série de comandos de operação.

A declaração de uma arquitetura pode conter comandos concorrentes ou sequenciais. Sua organização pode conter declaração de sinais, constante, componentes, operadores lógicos, etc., assim como comandos (ex: BEGIN, END). VHDL permite ter mais de uma architecture para a mesma entidade. Uma Arquitetura consiste em duas partes: a seção de declaração da arquitetura e o corpo da arquitetura.

A arquitetura contém o código propriamente dito. Sua sintaxe é mostrada a seguir:

```
ARCHITECTURE nome_da_arquitetura OF nome_entidade IS
BEGIN
<CODIGO>
END nome_da_arquitetura;
```

#### EXEMPLO:

```
ARCHITECTURE comportamento OF PortaAnd IS
BEGIN
S <= A and B;
END comportamento;
```

### Resumindo:

A implementação de uma porta lógica AND, em VHDL é:

|  |
|--|
| <pre>LIBRARY ieee;<br/>USE ieee.std_logic_1164.all;</pre>  |
| <pre>ENTITY PortaAnd IS<br/>    PORT( A, B : IN BIT;<br/>          S : OUT BIT);<br/>END PortaAnd;</pre>       |
| <pre>ARCHITECTURE comportamento OF PortaAnd IS<br/>BEGIN<br/>    S &lt;= A and B;<br/>END comportamento;</pre> |

A atribuição na linguagem VHDL é feita pelo símbolo: "<=";

Obs:

O VHDL não é case sensitive, ou seja, não é sensível para minúsculo e maiúsculo.

Obs2.:

Comentários em VHDL são feitos com dois traços (--).

---

### Hora de Praticar:

1. Desenvolva a porta OR em VHDL e simule utilizando o Quartus;
2. Faça o teste da porta AND em VHDL e simule utilizando o Quartus;
3. Desenvolva um código para que seja possível testar **TODAS** as propriedades, postulados e lei de De Morgan mostradas na primeira seção deste roteiro.

Dica:

Faça um código com três pinos de entradas e vários pinos de saída, no qual, cada igualdade será um pino de saída.

Ex:

Para a propriedade de identidade:

Entradas:

A;

Saída:

S1  $\leq$  A or '0'; -- A + 0

S2  $\leq$  A and '1'; -- A · 1

S3  $\leq$  not(not(A)) -- (A')'

4. Entregue um relatório contendo/descrevendo a execução dos itens 1, 2 e 3.
-