Introdução à VHDL

Prof. MSc. André Macário Barros

18/03/2019

Agenda

- O que é a VHDL
- partes básicas de um código VHDL (ENTITY, ARCHITECTURE)
- tipo std_logic
- Identificadores, regras léxicas
- biblioteca ieee, pacote std_logic.1164
- comentários, operadores básicos para o pacote std_logic.1164
- conceito de paralelismo da VHDL
- uso de sinais em VHDL
- 4 exemplos, 5 exercícios

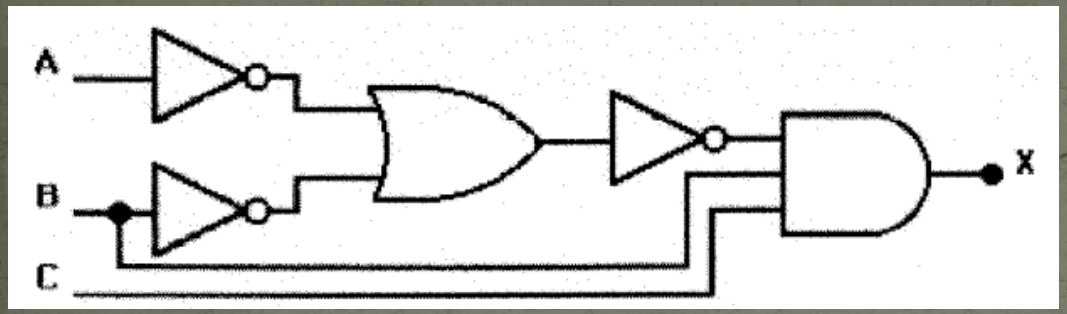
O que é a VHDL

- [V]HSIC (*Very High Speed Integrated Circuit*) [H]ardware [D]escription [L]anguage
  - programa do DoD (*Department of Defense*) dos EUA que, na década de 80, promoveu avanços na área de circuitos integrados. Uma de suas criações foi a VHDL.

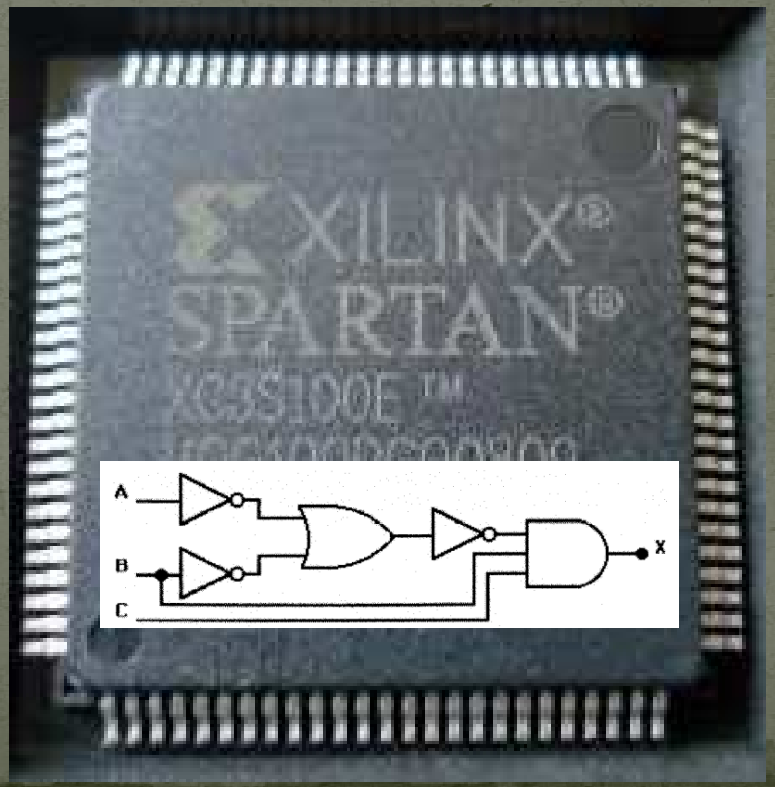
O que é a VHDL

- VHDL (*VHSIC Hardware Description Language*) é uma linguagem que permite:
  - a descrição da estrutura de um sistema digital (como pode ser decomposto em subsistemas e como estes podem ser interconectados)
  - especificar o que um sistema faz através de uma forma parecida com a que é usada nas linguagens de programação
  - que um projeto seja simulado antes de ser implementado em um dispositivo
  - que detalhes de subsistemas possam ser abstraídos, assegurando ao projetista focar-se em aspectos mais estratégicos do sistema como um todo

O que é a VHDL – Figura 1



VHDL



O Código VHDL – Partes Constituintes

- Um código VHDL é composto por três seções (partes) básicas:
  - bibliotecas;
  - ENTITY; e
  - ARCHITECTURE

O Código VHDL – Partes Constituintes

- A **ENTITY** é a seção do código VHDL destinada a descrever quais são os pinos de entrada, de saída ou de ambas que o sistema digital terá
- A **ARCHITECTURE** é a seção do código VHDL que descreve como que o sistema digital deverá se comportar ao processar as portas da ENTITY
- E a **seção das bibliotecas** é a parte do código VHDL destinada a incluir bibliotecas VHDL para proverem suporte aos tipos de dados e às funções a serem utilizadas nas seções da ENTITY e da ARCHITECTURE

Exemplo 1

- Desenvolva o código VHDL para o circuito digital da Figura 2

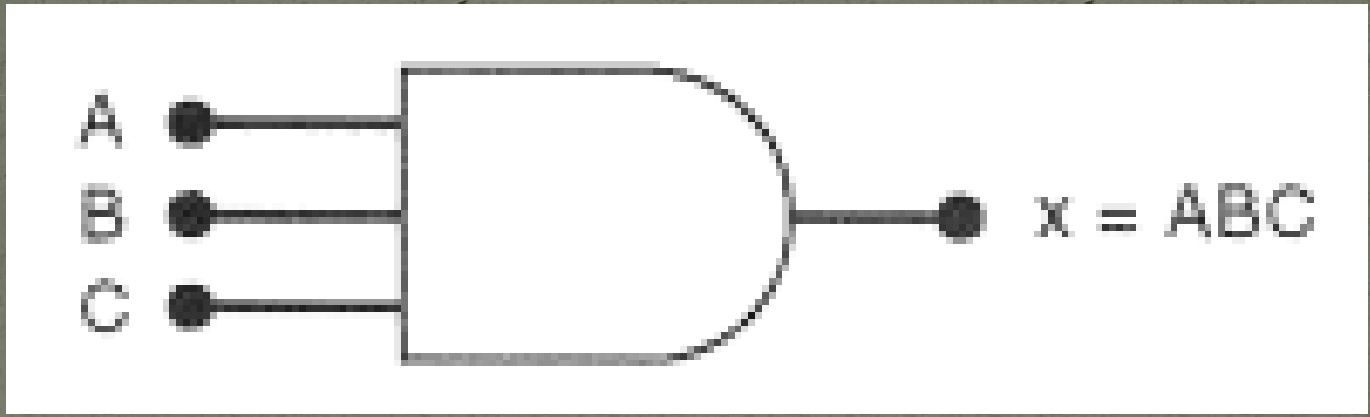
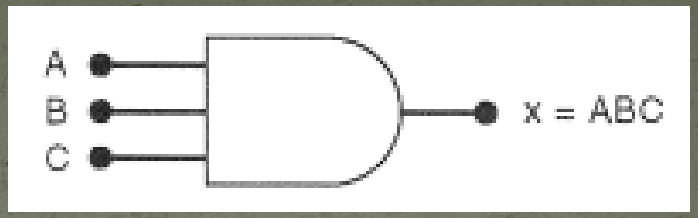


Figura 2 – Porta AND de três entradas.

Exemplo 1



- Solução:

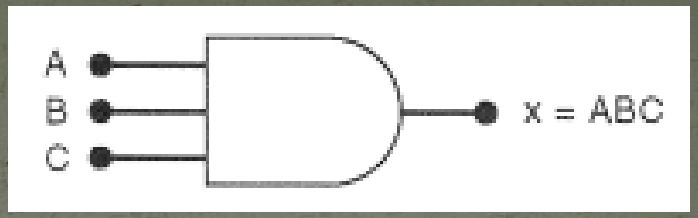
```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity porta_and is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end porta_and;

07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

bibliotecas

ENTITY

ARCHITECTURE

Exemplo 1

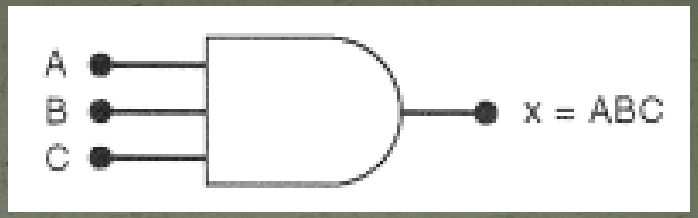


- Descrição:

```
01-library ieee;
02-use ieee.std_logic_1164.all;
```

- A biblioteca **`ieee`** e seu pacote **`std_logic`** são necessárias sempre que forem usadas portas ou sinais que recebam ou forneçam zeros ou uns lógicos, que é o caso das portas **`a`**, **`b`**, **`c`** e **`x`**. Tendo sido declaradas como **`std_logic`**, as entradas e saídas poderão assumir outros valores como 'alta impedância' (Z), don´t care (-), etc, como serão vistos ao longo desta disciplina.

Exemplo 1

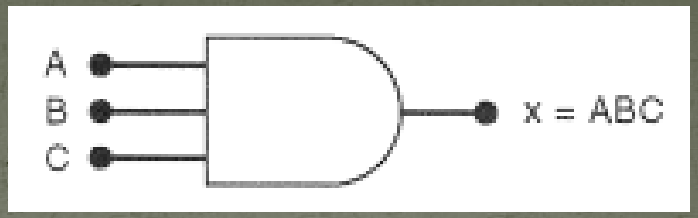


- Descrição:

```
03-entity porta_and is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end porta_and;
```

- Linhas 3 e 6: **`entity ... is ... end ...;`** são palavras-chave, necessárias à declaração de uma ENTITY

Exemplo 1

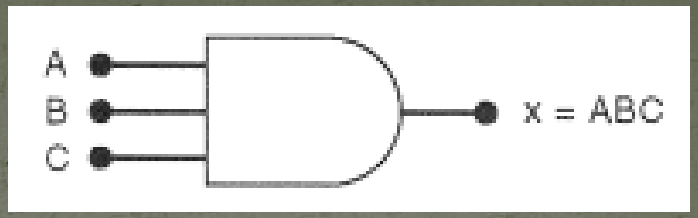


- Descrição:

```
03-entity porta_and is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end porta_and;
```

- Linha 3: **`porta_and`** é um identificador também necessário e escolhido pelo projetista. Identificadores podem ter até 26 números ou letras e ainda o caracter "_". Identificadores precisam começar com uma letra.

Exemplo 1

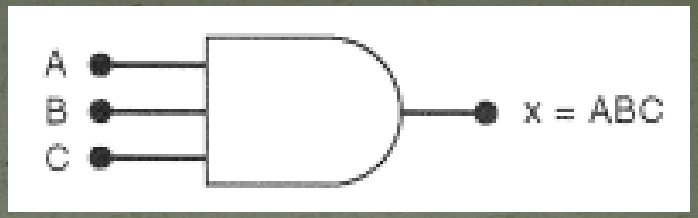


- Descrição:

```
03-entity porta_and is
04-port(a, b, c: in std_logic;
05-                x: out std_logic);
06-end porta_and;
```

- Linhas 4 e 5: **`port(...);`** é o escopo dentro do qual são definidas as entradas e saídas do sistema digital a ser descrito.

Exemplo 1

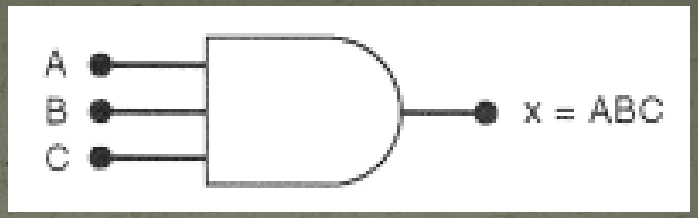


- Descrição:

```
03-entity porta_and is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end porta_and;
```

- Linhas 4 e 5: **a**, **b** e **c** são portas (pinos) do tipo **std_logic** de entrada (**in**) e **x** do mesmo tipo, porém de saída (**out**).

Exemplo 1

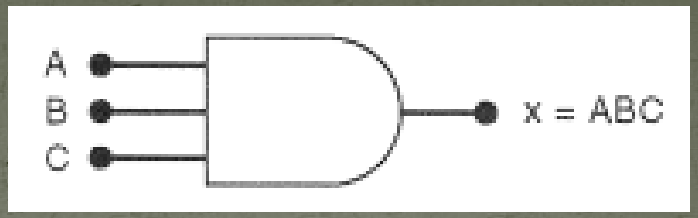


- Descrição:

```
03-entity porta_and is
04-port(a, b, c: in std_logic;
05-                x: out std_logic);
06-end porta_and;
```

- Linhas 4 e 5: **`std_logic`** é o tipo de dado atribuído aos pinos de entrada ou saída que passam dessa forma a serem capazes de receber e fornecer níveis lógicos, como ‘0’, ‘1’, ‘X’, etc.

Exemplo 1

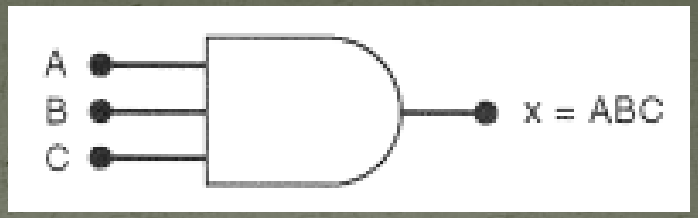


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linhas 7 e 10: **`architecture ... of ... is ... begin ... end ...;`** são as palavras-chave necessárias ao escopo dentro do qual o comportamento do circuito é descrito, isto é, o que o circuito deve fazer.

Exemplo 1

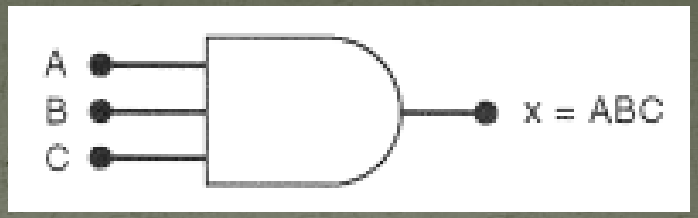


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linha 7: **`arq`** é também um identificador escolhido pelo projetista e **`porta_and`** precisa obrigatoriamente ser o nome do identificador que foi usado na declaração da ENTITY.

Exemplo 1

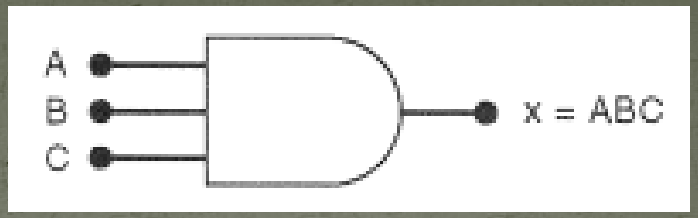


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linha 8: **--** é a sequência de caracteres que determina que, a partir daquele ponto, naquela linha, inicia-se um comentário dentro do código que o projetista necessita escrever.

Exemplo 1

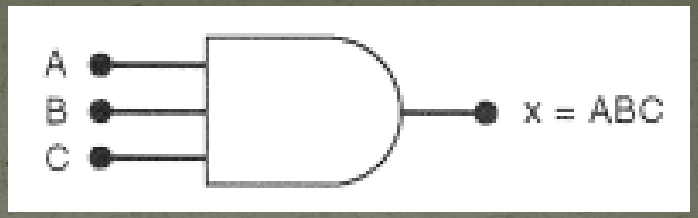


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linha 9: **`x <= a and b and c;`** É uma declaração (e não atribuição!) que faz as entradas **`a`**, **`b`** e **`c`** efetuarem entre si um **`and`** lógico e seu resultado ser direcionado à saída **`x`**. Declarações sempre terminam com um **`;`**

Exemplo 1

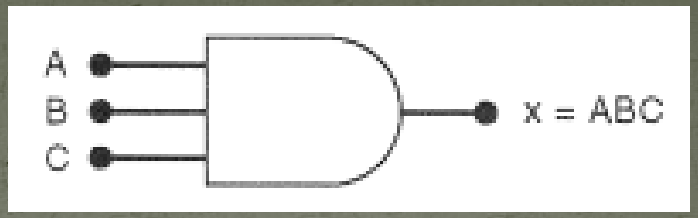


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linha 9: a operação **`and`** apresentada pode ser usada em sinais que pertençam ao pacote **`std_logic_1164`**. Esta e outras operações já estão pré-definidas em VHDL, como **`or`**, **`xor`** e **`not`**.

Exemplo 1

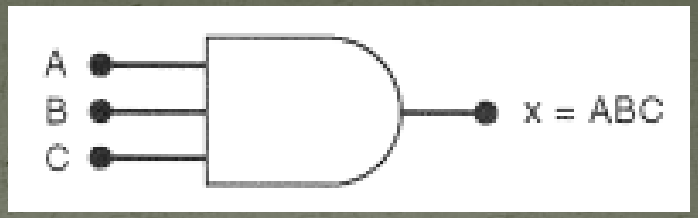


- Descrição:

```
07-architecture arq of porta_and is
08-begin --comentário qualquer
09-    x <= a and b and c;
10-end arq;
```

- Linha 10: **`arq`** usado nesta linha segue o nome do identificador que foi escolhido previamente na linha 7.

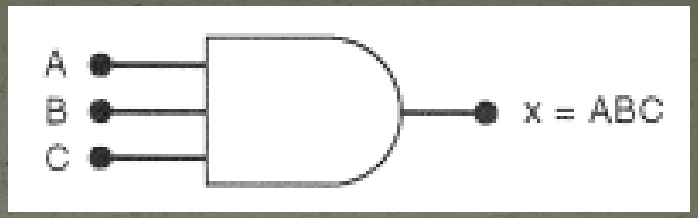
Exemplo 1



- Descrição:

- Linhas 1 a 10: VHDL não é *case-sensitive*, logo podem ser usadas maiúsculas, minúsculas, assim como as mesmas podem ser misturadas. Exemplos:
  - `ENTITY porta_and` ↔ `entity PORTA_AND`
  - `Entity Porta_And` ↔ `entity porta_and`

Exemplo 1



- Descrição:

- Linhas 1 a 10: entre os identificadores e palavras-chaves (não em seu interior), um espaço é igual a *n* espaços, também não importando para o processo de síntese (a ser explicado futuramente).
- Exemplo:
  - `entity porta_and`
  - é o mesmo que
  - `entity porta_and`

Exemplo 2

- Desenvolver o código VHDL para o circuito digital da Figura 3

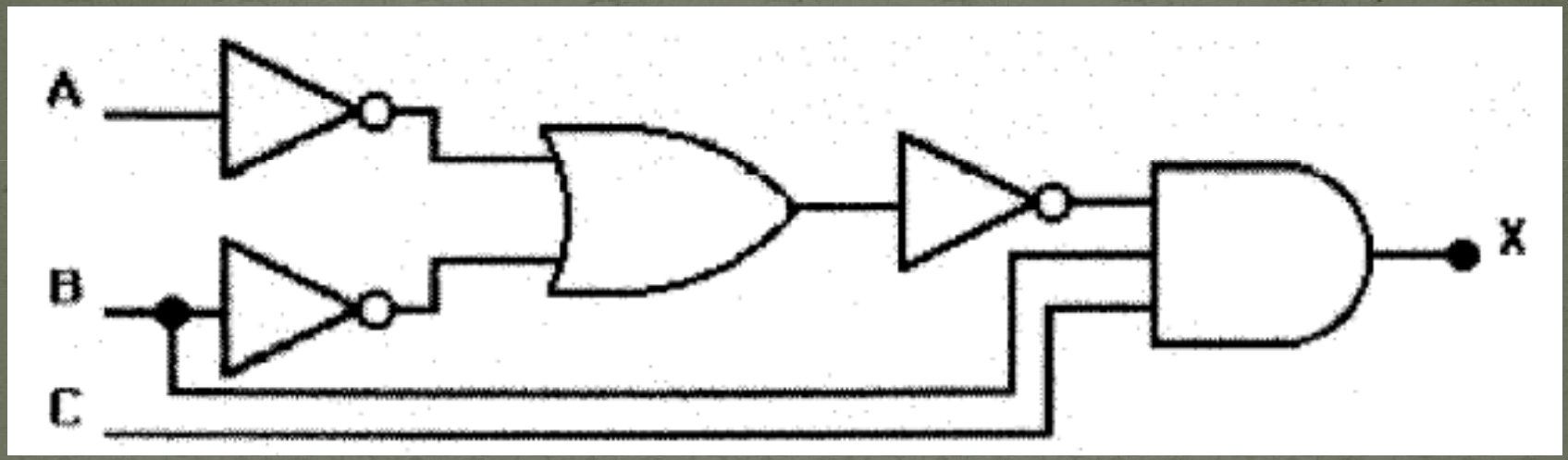
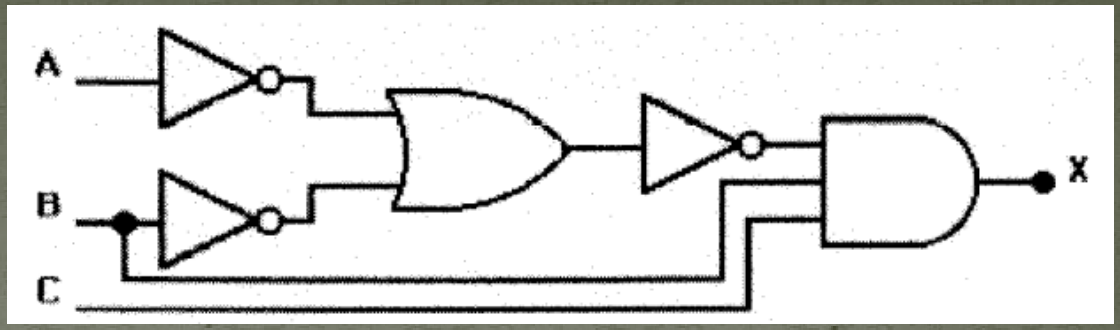


Figura 3 – Circuito combinacional básico.

Exemplo 2



- Solução:

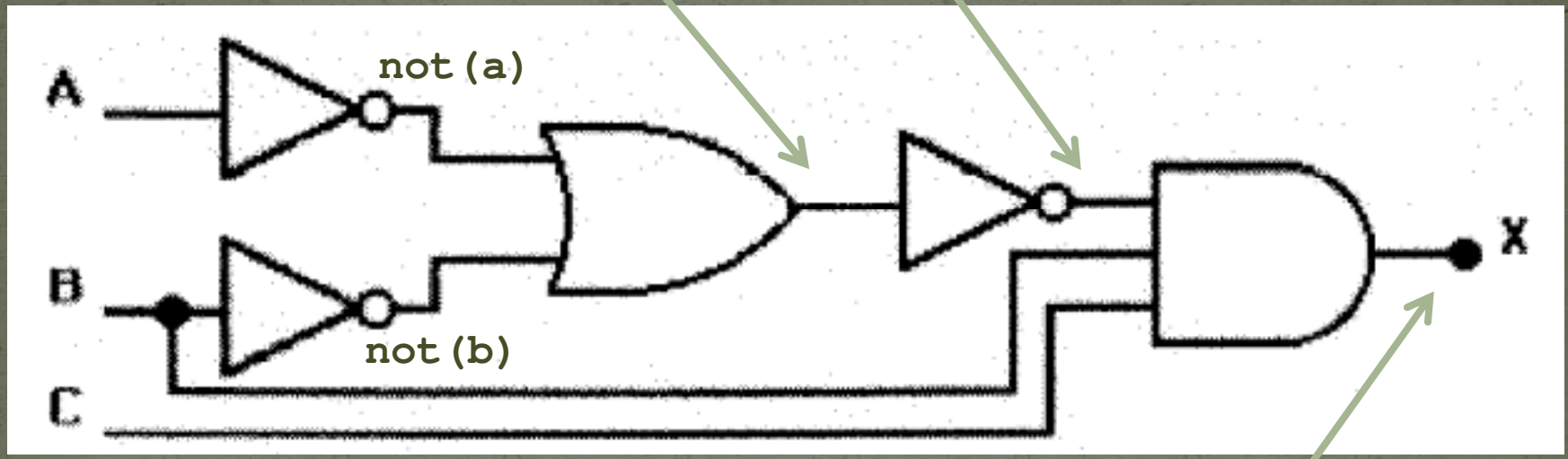
```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity ccto_basico is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end ccto_basico;

07-architecture arq of ccto_basico is
08-begin
09-   --colocar aqui a função lógica
10-end arq;
```

Mesmas entradas e mesmas saídas. Porém, a linha 9 deve receber uma extensa função lógica

Exemplo 2

```
not(not(a) or not(b))
```

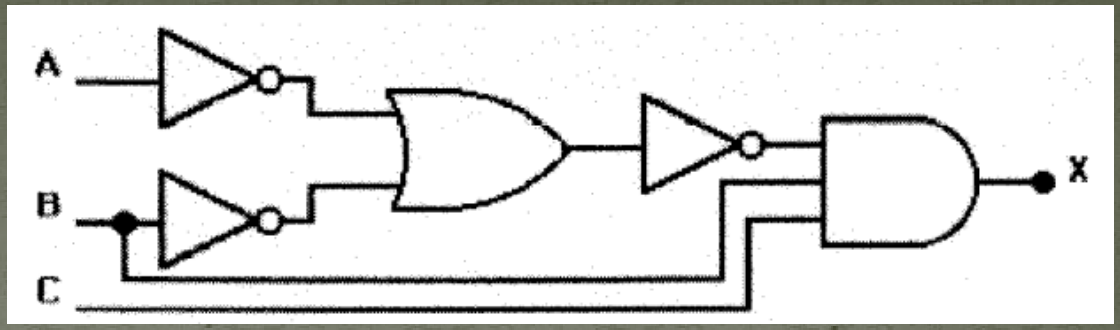
```
not(a) or not(b)
```




```
not(not(a) or not(b)) and b and c
```

```
x <= not(not(a) or not(b)) and b and c;
```

Exemplo 2



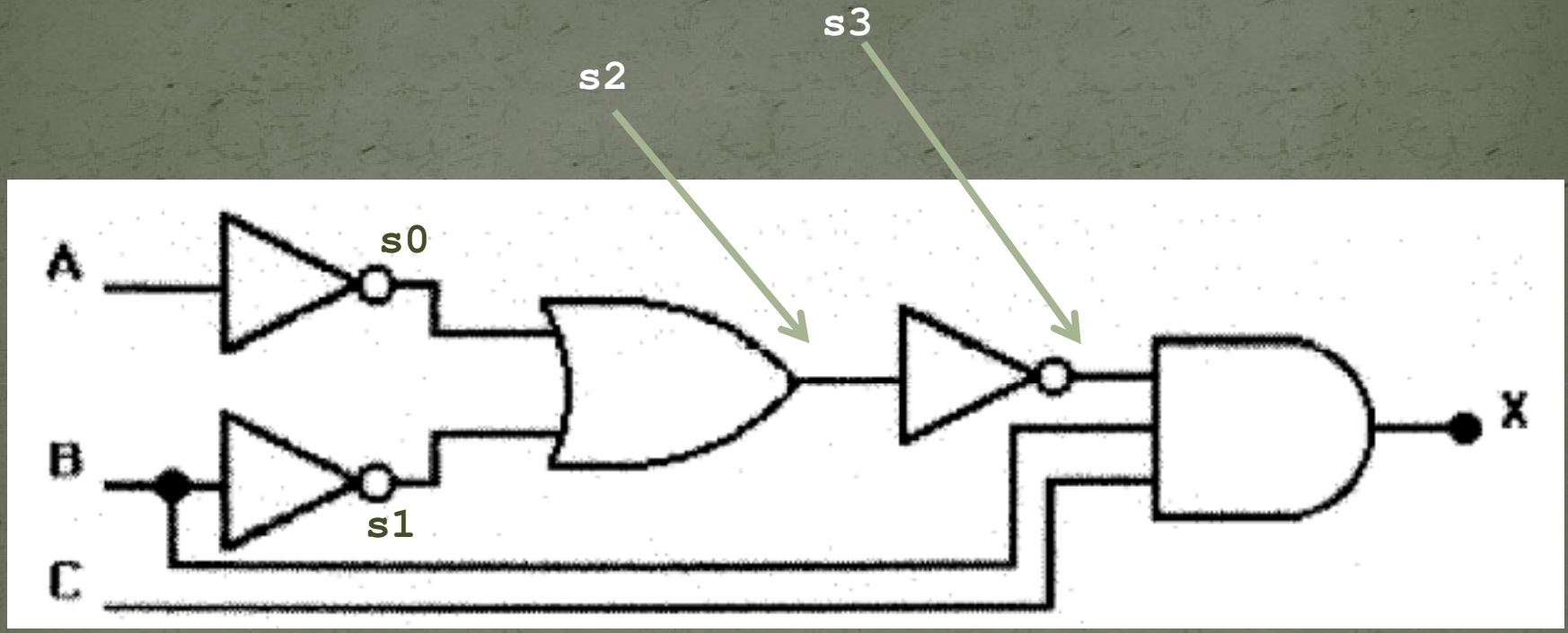
- Solução:

```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity ccto_basico is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end ccto_basico;

07-architecture arq of ccto_basico is
08-begin
09-    x <= not(not(a) or not(b)) and b and c;
10-end arq;
```

Exemplo 3 – sinais e paralelismo

- Reescrever o código VHDL do Exemplo 2 utilizando sinais
- Solução:
  - A função lógica escrita no Exemplo 2 é extensa e pode causar confusão
  - Será apresentada uma solução alternativa, utilizando sinais para a interconexão das saídas internas do sistema digital.
  - Os sinais **`s0`** ao **`s3`** da figura a seguir podem aqui ser entendidos como fios utilizados para interligar dois pontos em um proto-board
  - Acompanhe...

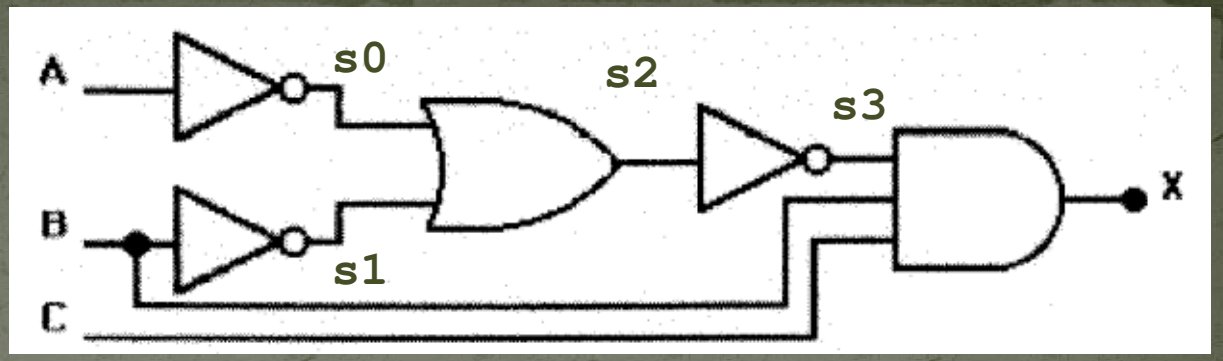
Exemplo 3 – sinais




```
S0 <= not(a);
S1 <= not(b);
S2 <= s0 or s1;
S3 <= not(s2);
```

```
x <= s3 and b and c;
```

Exemplo 3 sinais



- Solução:

```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity ccto_basico is
04-port(a, b, c: in std_logic;
05-             x: out std_logic);
06-end ccto_basico;

07-architecture arq of ccto_basico is
08-signal s0, s1, s2, s3: std_logic;
09-begin
09-   s0 <= not(a); s1 <= not(b); s2 <= s0 or s1;
10-   s3 <= not(s2); x <= s3 and b and c;
11-end arq;
```

Exemplo 3 – paralelismo

- A linha 09 pode ser trocada com a linha 10, pois VHDL é uma linguagem essencialmente **concorrente** por ser uma linguagem de **construção de *hardware***.
- VHDL **não é** uma linguagem de programação, onde as instruções ocorrem de forma sequencial, uma após a outra (a não ser que seja explicitamente designado através da declaração PROCESS).
- Portanto, as três declarações da linha 09 e as duas da linha 10 estão acontecendo ao mesmo tempo no circuito.

```
09-   s0 <= not(a); s1 <= not(b); s2 <= s0 or s1;
10-   s3 <= not(s2); x <= s3 and b and c;
```

↔

```
10-   s3 <= not(s2); x <= s3 and b and c;
09-   s0 <= not(a); s1 <= not(b); s2 <= s0 or s1;
```

Exemplo 3 – precedência

- IMPORTANTE: exceto **`not`**, as operações **`and`**, **`or`**, **`nand`**, **`nor`**, **`xor`** e **`xnor`** têm a mesma precedência
- Portanto, cuidado ao descrever funções lógicas da eletrônica digital clássica, por exemplo $y = ab + cd$, . Em VHDL deve-se usar parênteses, caso se queira que a função se comporte da forma esperada:
  - `y <= (a and b) or (c and d);`
  - `--resulta '1' para abcd='1100' e '1110'`
  - ao invés de
  - `y <= a and b or c and d;`
  - `--resulta '0' para abcd='1100' e '1110'`

Exemplo 4

- Desenvolver um código VHDL para um circuito comparador de 1 *bit*.
- Solução: inicialmente, vamos imaginar o circuito em termos de entradas (`a` e `b`) e saídas (`y`). Veja a Figura 4:

Exemplo 4

- De posse do formato do circuito, estabeleçamos a tabela-verdade que um comparador deve cumprir:

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

Exemplo 4

- A função lógica que satisfaz à tabela-verdade pode ser obtida diretamente dos mintermos, não sendo necessário Karnaugh:
  - $y = a'b' + ab$

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

Exemplo 4

- Solução ($y = a'b' + ab$):

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

```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity comparador is
04-port(a, b: in std_logic;
05-        y: out std_logic);
06-end comparador;

07-architecture arq_comp of comparador is
08-begin
09-   y <= (not(a) and not(b)) or (a and b);
10-end arq_comp;
```

Exercício 1

- Desenvolva um código em VHDL para o circuito proposto no Exemplo 4. Porém utilize sinais.

Exercício 2

- O circuito do Exemplo 4 pode ser escrito através de uma função lógica menos extensa e que não tem relação com o uso de sinais. Encontre esta função.
- Dica: é um conceito que vem das propriedades básicas das funções lógicas (conceitual – Eletrônica Digital).

Exercício 3

- Desenvolva o circuito digital e o código VHDL correspondente para um circuito GT (*greater than*) de dois *bits*. Utilizar funções lógicas mínimas. Atentar para a ordenação MSB e LSB.

Exercício 4

- Desenvolva o circuito digital e o código VHDL correspondente para um circuito que faça a conversão binário → *gray* de três *bits*. Utilizar funções lógicas mínimas. Atentar para a ordenação MSB e LSB.

Exercício 5

- Desenvolva o circuito digital e o código VHDL correspondente para um circuito multiplexador 4 × 1. Utilizar funções lógicas mínimas. Atentar para a ordenação MSB e LSB.
- Dica 4 × 1 significa 4 entradas de dados para uma saída. Para que isso aconteça, duas entradas de endereçamento precisam governar qual entrada deverá ser destinada à saída.

Referências

- CHU, Pong P. **FPGA Prototyping by VHDL examples**. New Jersey (NJ): John Wiley & Sons, 2008. Há 8 exemplares disponíveis na biblioteca. Número de chamada: 621.395 C559f
- PEDRONI, Volnei. **Feletrônica Digital Moderna e VHDL**. Rio de Janeiro: Elsevier, 2010. Há 12 exemplares disponíveis na biblioteca. Número de chamada: 621.395 C559f