Relatório: Atividade 1

Filipe Augusto Parreira Almeida

RA: 2320622

30 de março de 2024

Conteúdo

1 Resumo

Está sendo relatado a simulação de portas lógicas por meio do *software* ***Quartus II*** da Intel, utilizando a linguagem **VHDL**. Neste projeto há duas entradas, A e B, e oito saídas, onde cada saída é o resultado de uma porta lógica. As portas lógicas utilizadas são **NOT** (uma para cada entrada), **AND**, **OR**, **NOR**, **NAND**, **XOR** e **XNOR**. Logo, o intuito é entender o funcionamento de cada porta lógica, conseguindo relacionar o resultado da simulação com a tabela-verdade de cada uma.

2 Introdução

2.1 Portas Lógicas e Álgebra Binária

Circuitos lógicos ou portas lógicas são dispositivos que operam de acordo com a álgebra booleana. A álgebra booleana foi inicialmente introduzida em 1854, através da obra ***"Investigação das Leis do Pensamento"*** escrita pelo matemático ***George Boole*** (TOCCI; WIDMER; MOSS, 2011), nela é expressado um modo de tomada de decisões a partir de verdadeiro ou falso. Eletronicamente, valores verdadeiros ou falsos, mais precisamente o conjunto numérico binário, em que é representado estados de **ALTO** ou **BAIXO**, são detectados por meio de faixas de tensão onde o 0 (BAIXO) está geralmente entre as faixas 0V a 0.8V, e 1 está entre as faixas 2V a 5V.

Com o auxílio da álgebra binária podemos descrever o funcionamento das portas lógicas através das **tabelas-verdade**, onde é representada todas as entradas possíveis para uma certa quantidade de bits. Vale ressaltar que para uma quantidade de bits muito grande (mais precisamente em uma ordem de 2^n, onde n é a quantidade de bits de entrada) utiliza-se outros métodos de análise, porém estes métodos estão fora do escopo deste trabalho. Segue o exemplo de uma tabela-verdade para entrada de 4 bits:

Figura 1: Exemplo Tabela-Verdade.

A	B	C	D	x
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Fonte: (TOCCI; WIDMER; MOSS, 2011)

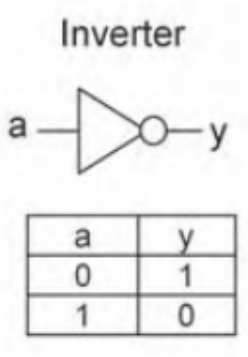
2.2 Funcionamento das Portas Lógicas

Dentre as portas lógicas, existem 3 portas lógicas fundamentais: a porta **AND**, **OR** e a **NOT**. Elas originam a maioria das portas lógicas. As utilizadas neste trabalho são:

- Inversor (NOT): Inverte o sinal de entrada.

$$y = a'$$

Figura 2: Inversor



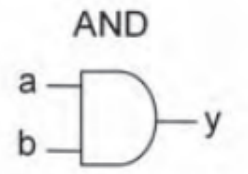
a	y
0	1
1	0

Fonte: (PEDRONI, 2010)

- AND: Executa a multiplicação lógica.

$$y = a * b$$

Figura 3: Porta AND



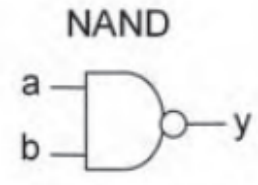
a b	y
0 0	0
0 1	0
1 0	0
1 1	1

Fonte: (PEDRONI, 2010)

- NAND: Produz multiplicação lógica invertida, funciona como uma porta AND porem com um inversor na saída.

$$y = (a * b)'$$

Figura 4: Porta NAND



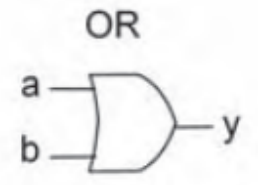
a b	y
0 0	1
0 1	1
1 0	1
1 1	0

Fonte: (PEDRONI, 2010)

- OR: Executa adição lógica.

$$y = a + b$$

Figura 5: Porta OR



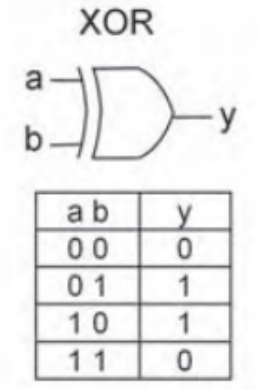
a b	y
0 0	0
0 1	1
1 0	1
1 1	1

Fonte: (PEDRONI, 2010)

- NOR: Produz adição lógica invertida, funciona como uma porta OR porem com um inversor na saída.

$$y = (a + b)'$$

Figura 6: Porta XOR



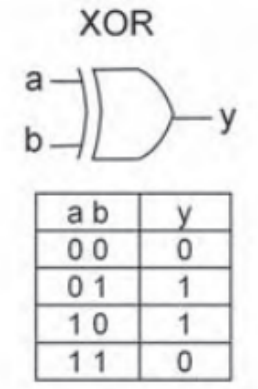
a b	y
0 0	0
0 1	1
1 0	1
1 1	0

Fonte: (PEDRONI, 2010)

- XOR (OR Exclusivo): Saída 1 quando o número de entradas altas é ímpar.

$$y = a \oplus b$$

Figura 7: Porta XOR



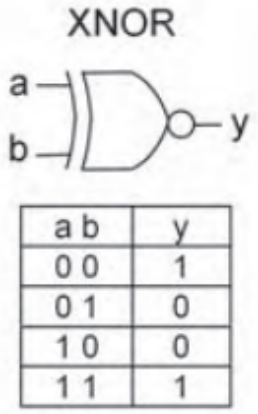
a b	y
0 0	0
0 1	1
1 0	1
1 1	0

Fonte: (PEDRONI, 2010)

- XNOR: Saída 1 quando o número de entradas altas é par.

$$y = (a \oplus b)'$$

Figura 8: Porta XNOR



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

Fonte: (PEDRONI, 2010)

3 Implementação

Todas as portas lógicas foram implementadas utilizando a **linguagem de descrição de hardware VHDL**. Ela é uma linguagem que independe de tecnologia e fabricante, em que o código descreve o comportamento desejado, onde o compilador gerará um circuito correspondente (PEDRONI, 2010). Uma de suas aplicações é a síntese de circuitos digitais em **CPLDs** e **FPGAs**, porém também é possível realizar a geração de máscaras de fabricação de CI's digitais integrados (ASICs). Ela surgiu como uma iniciativa financiada pelo Departamento de Defesa dos Estados Unidos em 1980, e foi a primeira linguagem de descrição de hardware padronizada pelo ***IEEE***.

Sua estrutura de código é dividido em **três** partes, **Declaração de Bibliotecas/Pacotes**, **Entidade** e **Arquitetura**; cada uma delas é apresentada a seguir (vale ressaltar que a linguagem VHDL não é case sensitive, ou seja, ela não diferencia o que é letra maiúscula do que é minúscula; porém, por convenção e para adotar um aspecto mais legível ao código, decidi que todas as palavras reservadas da linguagem serão escritas com letras maiúsculas):

- Declaração de Bibliotecas/Pacotes

  Para a declaração de bibliotecas, utiliza-se duas palavras reservadas, ***LIBRARY*** e ***USE***. ***LIBRARY*** é utilizada para especificar quais bibliotecas serão utilizadas, e ***USE*** é para especificar os pacotes destas bibliotecas que serão utilizados. Na implementação foi utilizada a biblioteca da ***IEEE*** e utilizado o pacote ***STD_LOGIC_1164***, que é utilizado para definir os valores do tipo lógico que serão os tipos utilizados nas entradas e saídas.

  Segue o trecho de código que representa a declaração de bibliotecas/pacotes:

```
1 -- Declara  o de Bibliotecas/Pacotes
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
```

Código 1: Declaração de Bibliotecas/Pacotes

- Entidade

  A entidade é onde se define as **entradas, saídas e também constantes**. Ela pode ser dividida em duas seções de código chamadas de ***PORT*** e ***GENERIC***. Em ***PORT*** é onde se define as portas de entrada e saída do circuito. ***GENERIC*** é onde se define as constantes genéricas, verdadeiramente globais, ou seja, pode-se utilizá-las em qualquer seção do código.

  No escopo do trabalho está sendo utilizado **somente** a seção ***PORT***, que é onde são definidas as entradas **A** e **B**, e as saídas, um ***OUT*** para **cada** tipo de circuito lógico. Todas as entradas e saídas são definidas do tipo ***STD_LOGIC***, o que quer dizer que elas podem assumir valores descritos na tabela a seguir, onde é relacionado a maioria dos tipos de dados existentes em **VHDL**:

Figura 9: Tabela Tipo de Dados 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

Fonte: (TOCCI; WIDMER; MOSS, 2011)

Porém, no código, são atribuídos somente valores **booleanos** no formato de *bit* para cada porta, onde neste caso poderia também ser utilizado o tipo padrão ***BIT*** para definir as portas.

Da estrutura de implementação o nome da entidade é definido como **Atividade1**, logo em seguida é chamado o comando ***PORT***, abrindo a subseção de entidade, e então definido as portas de **entrada** (linha **3**) e as portas de **saída** (linha **4** e **5**). Na linha **7** foi declarado o **fim da entidade**.

Segue o trecho de código onde é definida a entidade:

```
1 -- ENTIDADE
2 ENTITY Atividade1 IS PORT (
3     a, b: IN STD_LOGIC;
4     outNOTa, outNOTb, outOR, outAND,
5         outNAND, outNOR, outXNOR, ourXOR: OUT STD_LOGIC
6 );
7 END Atividade1;
```

Código 2: Declaração da Entidade

- Arquitetura

  A arquitetura é a seção de código onde escreve-se o "código" propriamente dito, ela pode ser dividida em **duas** parte, parte **declarativa** e parte de **código**; a parte declarativa é onde se permite realizar diversos tipos de declaração, como, de tipo(***TYPE***), sinal(***SIGNAL***), constante(***CONSTANT***), componente(***COMPONENT***) e funções(***FUNCTION***) (PEDRONI, 2010). A parte do código contém todo o código VHDL, onde, dependendo do tipo de circuito, o código pode ser concorrente ou sequencial.

  No geral a parte do código é executada de forma **concorrente** (utilizada para **circuitos combinacionais**), caso seja necessário uma execução **sequencial** (circuitos **tanto combinacionais quanto sequencias**) é necessário que forçar este tipo de execução, utilizando um processo (***PROCESS***), que passa a se tornar uma subdivisão de código dentro da parte de código da entidade. No presente trabalho é utilizado **execução sequencial** portanto **não** há uso de processos.

Na implementação, o nome da arquitetura é ***arq_ AT1***, logo em seguida é definido que a arquitetura é referente à entidade ***Atividade1***, e da linha **4** a **11** está a seção de código que é composto basicamente por atribuições (“<=”) aos valores de saída.

Segue o trecho de código referente à arquitetura:

```
1  -- ARQUITETURA
2  ARCHITECTURE arq_AT1 OF Atividade1 IS
3      BEGIN
4          outNOTa <= NOT a;
5          outNOTb <= NOT b;
6          outOR <= a OR b;
7          outAND <= a AND b;
8          outNAND <= a NAND b;
9          outNOR <= a NOR b;
10         outXNOR <= a XNOR b;
11         outXOR <= a XOR b;
12 END arq_AT1;
```

Código 3: Declaração da Arquitetura

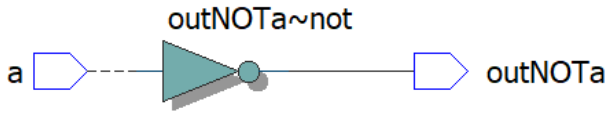
4 Conclusão

É apresentado os resultados obtidos com a execução de código e a simulação dos sinais **A** e **B** como valores de ***clock*** padronizados, sendo **A** a **metade** da duração que **B**, ou seja, **A** com ***clocks*** de duração **50 ns** e **B** com ***clocks*** de duração **100 ns**. Os resultados serão apresentados para cada tipo de porta lógica separadamente e posteriormente um conjunto de todas as saídas, sendo apresentado a representação em forma de onda e a gráfica do circuito.

4.1 Representação Gráfica

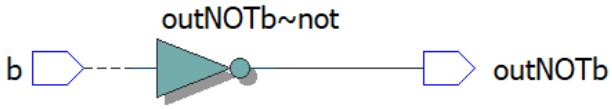
- Porta NOT

Figura 10: $y = A'$



Fonte: Autoria Própria

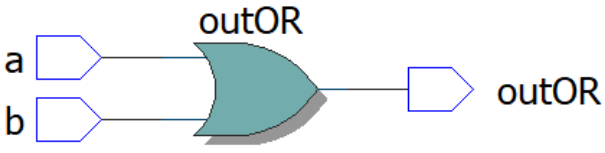
Figura 11: $y = B'$



Fonte: Autoria Própria

- Porta OR

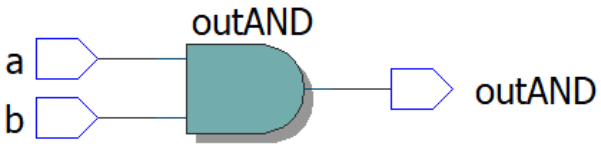
Figura 12: $y = A + B$



Fonte: Autoria Própria

- Porta AND

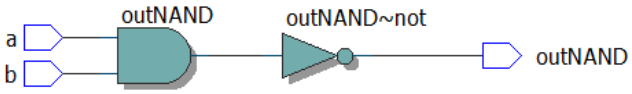
Figura 13: $y = A * B$



Fonte: Autoria Própria

- Porta NAND

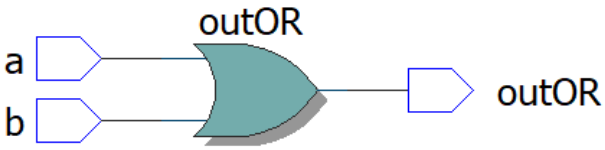
Figura 14: $y = (A * B)'$



Fonte: Autoria Própria

- Porta NOR

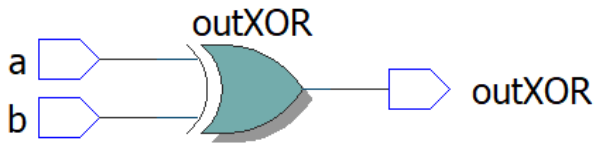
Figura 15: $y = (A + B)'$



Fonte: Autoria Própria

- Porta XOR

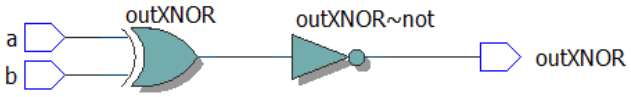
Figura 16: $y = A \oplus B$



Fonte: Autoria Própria

- Porta XNOR

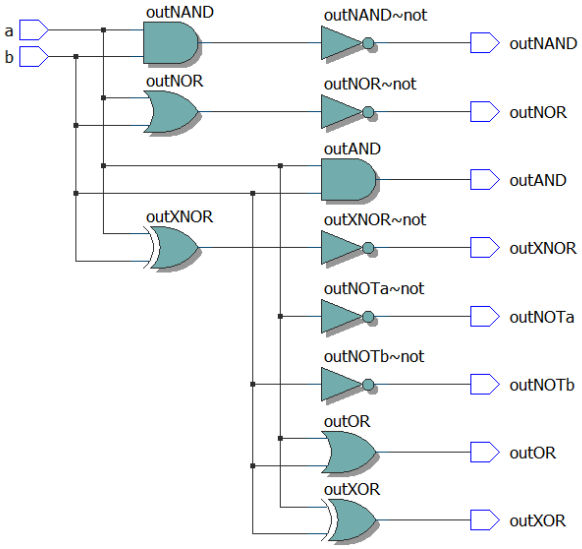
Figura 17: $y = (A \oplus B)'$



Fonte: Autoria Própria

- Circuito Completo

Figura 18: Representação gráfica Circuito Completo

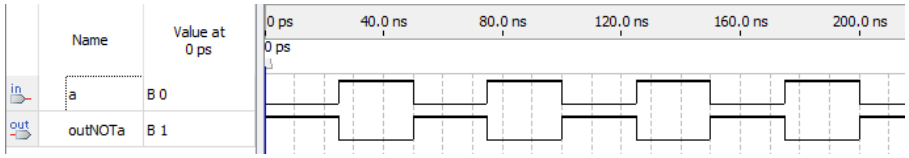


Fonte: Autoria Própria

4.2 Forma de Onda

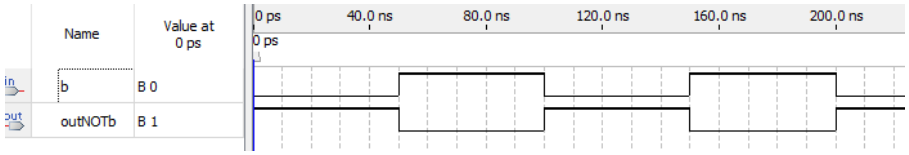
- Porta NOT

Figura 19: $y = A'$



Fonte: Autoria Própria

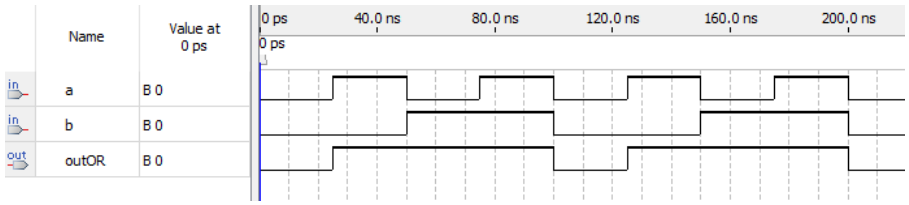
Figura 20: $y = B'$



Fonte: Autoria Própria

- Porta OR

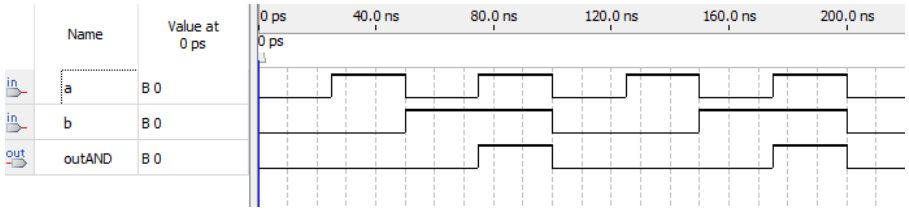
Figura 21: $y = A + B$



Fonte: Autoria Própria

- Porta AND

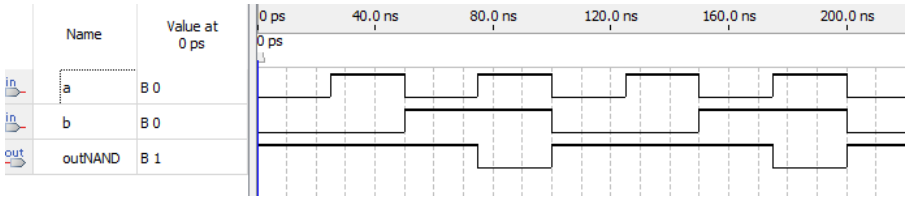
Figura 22: $y = A * B$



Fonte: Autoria Própria

- Porta NAND

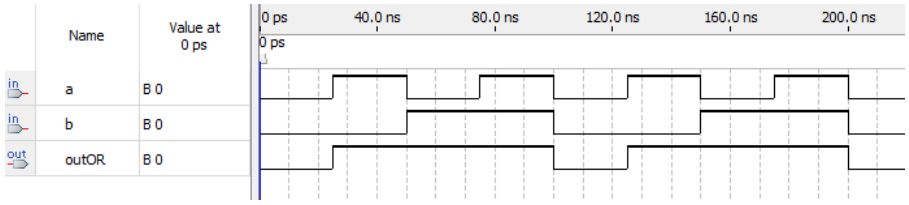
Figura 23: $y = (A * B)'$



Fonte: Autoria Própria

- Porta NOR

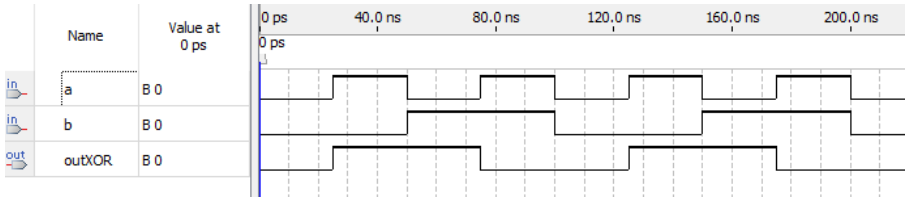
Figura 24: $y = (A + B)'$



Fonte: Autoria Própria

- Porta XOR

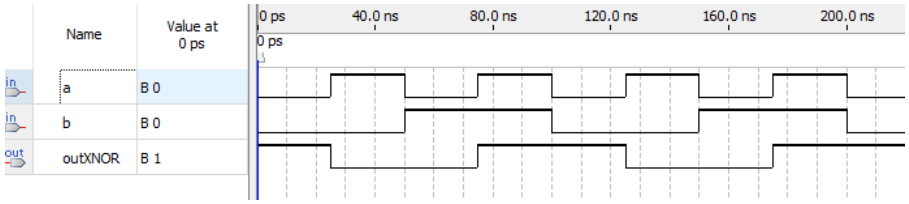
Figura 25: $y = A \oplus B$



Fonte: Autoria Própria

- Porta XNOR

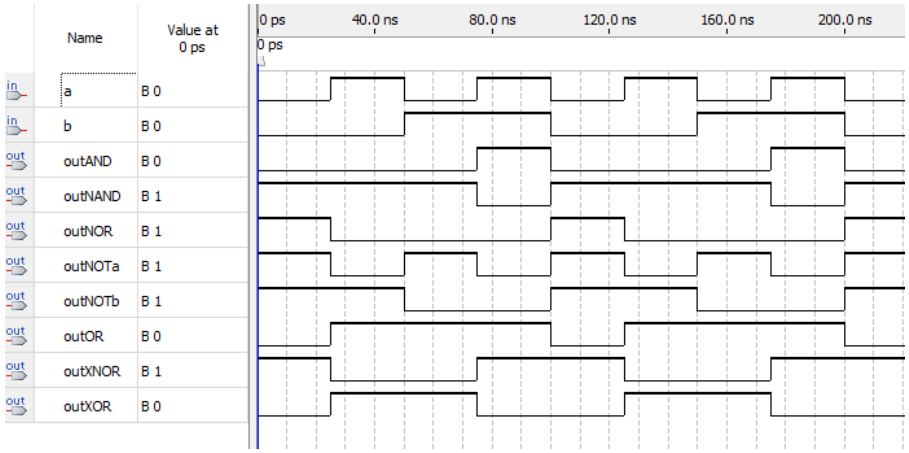
Figura 26: $y = (A \oplus B)'$



Fonte: Autoria Própria

- Circuito Completo

Figura 27: Forma de Onda Circuito Completo



Fonte: Autoria Própria

Analisando os resultados gráficos podemos concluir que todas as saídas representadas são compostas por **quatro portas lógicas primordiais**: ***OR***, ***AND***, ***NOT*** e ***XOR***. Onde a **junção** delas em ordem específicas produz cada saída apresentada. Na análise de ondas temos que **o resultado foi o esperado**, considerando as **tabelas-verdade** de cada circuito lógico, possibilitando assim a visualização e comprovação simulada do funcionamento de cada circuito.

Referências

PEDRONI, Volnei A. **Eletrônica digital moderna e VHDL**. Rio de Janeiro, RJ: Elsevier, 2010. P. 619. ISBN 9788535234657.

TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. **Sistemas digitais: princípios e aplicações**. 11. ed. São Paulo, SP: Pearson Prentice Hall, 2011. P. xxii, 817. ISBN 9788576059226.