Capítulo 3

Portas lógicas II

Neste capítulo, vamos trabalhar portas lógicas mais complexas, que na verdade são combinações das portas lógicas mais simples, estudadas no capítulo anterior. Portanto, é importante que você já esteja familiarizado com as portas lógicas básicas e com o modo de escrever expressões booleanas que descrevem como unir portas lógicas para obter o comportamento desejado.

Para obter esses resultados, empregamos as tabelas-verdade e as utilizamos para escrever as equações, e, por consequência, conseguimos desenhar os circuitos lógicos de que precisamos para resolver as situações de projeto com a técnica digital.

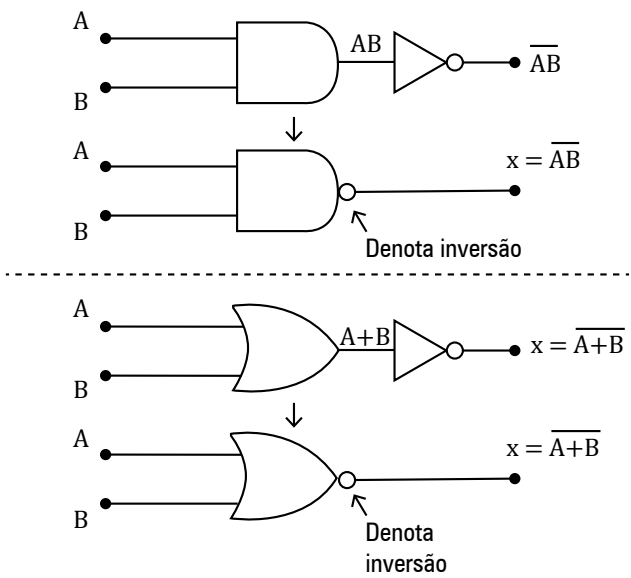
Além de trabalhar as portas lógicas mais complexas, vamos começar a entender como podemos fazer somas usando números binários, e por que isso é tão importante.

1 Portas lógicas NAND, NOR, XOR, XNOR

As primeiras portas lógicas complexas que vamos trabalhar neste capítulo são as portas NAND e NOR. A porta lógica NAND também pode ser chamada de "não AND" ou "AND negado". Já a porta lógica NOR também pode ser referida como "não OR" ou "OR negado".

Do ponto de vista da matemática de Boole (IDOETA; CAPUANO, 1999), a porta NAND trata apenas de negar, ou inverter, a saída de uma porta lógica AND, enquanto a NOR trata de inverter a saída de uma porta lógica OR, o que pode ser visualizado na figura a seguir.

Figura 1 — Processo de contração da notação das portas lógicas



Podemos observar que, tanto na porta NAND como na porta NOR, temos uma pequena bolinha que denota que a porta está invertida. Isso se aplica a qualquer representação de porta lógica, incluindo circuitos lógicos mais complexos.

Além disso, podemos descrever seu comportamento pelas suas tabelas-verdade, a seguir.

Tabela 1 – Tabela-verdade 1

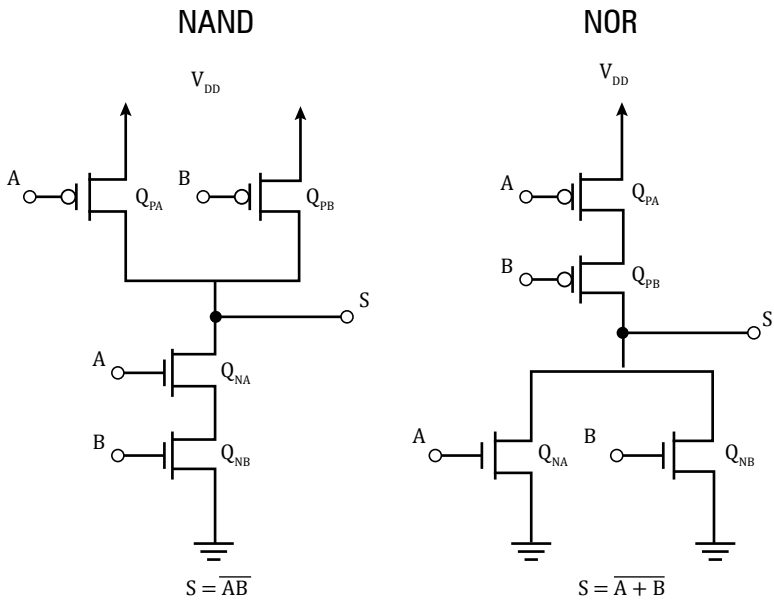
		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Tabela 2 – Tabela-verdade 2

		OR	NOR
A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Há duas características que tornam essas portas lógicas muito interessantes. A primeira é que suas implementações no mundo da microeletrônica são das mais eficientes em termos de área ocupada. A seguir, como exemplo, temos uma implementação usando tecnologia CMOS das duas portas, cada uma ocupando a área de apenas quatro transistores.

Figura 2 – Esquema elétrico de portas lógicas CMOS



Não vamos nos aprofundar nas técnicas construtivas da microeletrônica, porém, é possível perceber o quanto é simples a implementação dessas portas lógicas.

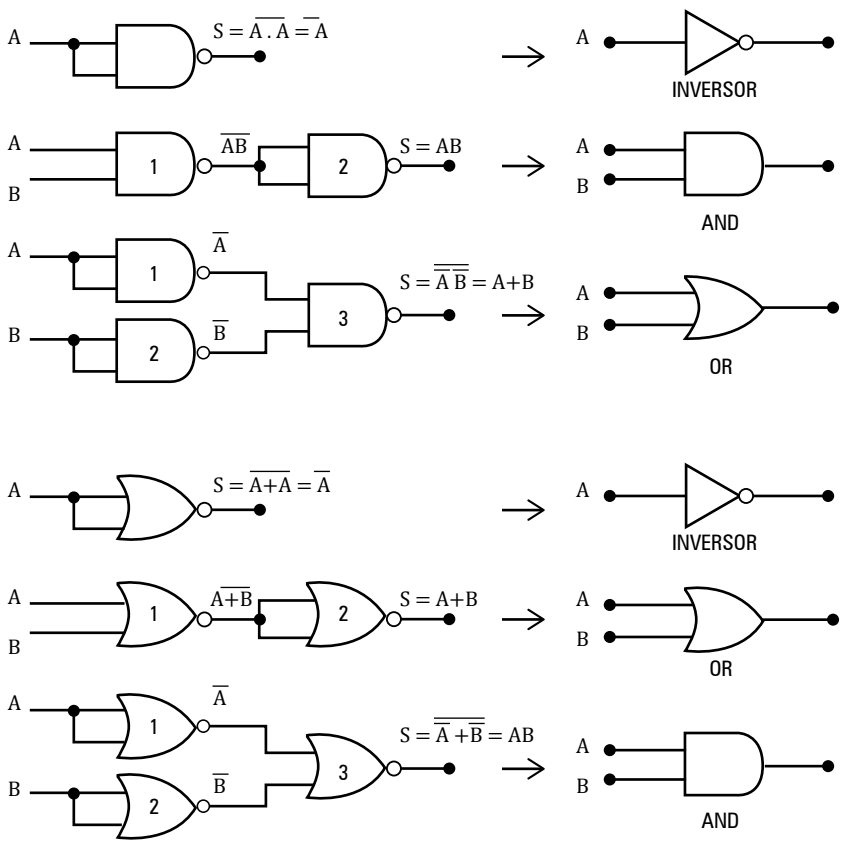
A segunda característica que torna essas portas lógicas muito interessantes é que elas são universais, ou seja, é possível implementar qualquer expressão booleana apenas combinando portas NAND ou apenas combinando portas NOR.

Essa propriedade, entre outras vantagens, facilita – e muito – a implementação de algoritmos que transformam as expressões lógicas em circuitos eletrônicos, permitindo uma enorme automação no processo de desenvolvimento de circuitos integrados bastante complexos. Ou seja, com um desenho físico de uma porta NAND, que vai ser muito eficiente em termos de área, é possível implementar qualquer função lógica usando um algoritmo embutido em um software.

Pensando em circuitos muito grandes e complexos, como processadores, temos uma forma de automatizar ou "compilar" silício graças a essa propriedade das portas NAND e NOR.

Para entender como funciona a universalidade das portas NAND e NOR, confira os circuitos a seguir, nos quais há a transformação das funções lógicas básicas para seus circuitos lógicos equivalentes, implementados apenas com as funções NAND e NOR (IDOETA; CAPUANO, 1999).

Figura 3 — Demonstrando a universalidade das portas NAND e NOR

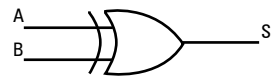


Podemos concluir que geramos todas as portas lógicas básicas a partir de portas NAND e NOR – e isso vale para a implementação de qualquer circuito combinacional, o que simplifica muito a sua implementação nas atuais técnicas de microeletrônica.

Agora, vamos tratar de duas portas lógicas complexas com muitas aplicações, que são importantes no dia a dia de quem projeta equipamentos usando a técnica digital. No caso, estamos falando das portas lógicas XOR e XNOR, ou, respectivamente, "OU exclusivo" e "não OU exclusivo", sendo que esta última é a negação da porta XOR.

A função XOR tem como símbolo na representação algébrica (+) ou ⊕. Assim, uma operação XOR de duas entradas, A e B, seria representada como S = A (+) B ou S = A ⊕ B. A leitura da expressão será sempre A XOR B, e sua representação em termos de circuitos lógicos será conforme a figura a seguir.

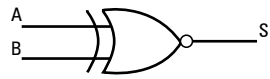
Figura 4 — Representação gráfica de uma porta XOR



Fonte: adaptado de Idoeta e Capuano, 1999.

Já a função XNOR é representada basicamente com a negação da porta XOR, como na figura a seguir.

Figura 5 — Representação gráfica de uma porta XNOR



Fonte: adaptado de Idoeta e Capuano, 1999.

Para representar a operação booleana, podemos simbolizar o XNOR como (*) ou ⊙. Assim, uma operação XNOR de duas entradas, A e B, seria representada como: S = A (*) B ou S = A ⊙ B.

Na função XOR, a saída pode ser entendida como verdadeira quando apenas uma das entradas for verdadeira (tiver valor 1), ou seja, se mais de uma entrada for verdadeira, a saída assume valor falso. Isso fica fácil de entender considerando-se as tabelas-verdade a seguir. Na tabela-verdade 3 há duas entradas, e na tabela-verdade 4, três entradas. Além disso, ambas apresentam a saída para as funções XOR e XNOR, deixando bem claro como ambas as funções lógicas funcionam.

Tabela 3 – Tabela-verdade 3

A	B	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Fonte: adaptado de Idoeta e Capuano, 1999.

Tabela 4 – Tabela-verdade 4

A	B	C	XOR	XNOR
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

(*Cont.*)

A	B	C	XOR	XNOR
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Fonte: adaptado de Idoeta e Capuano, 1999.

2 Revisão: soma de binários

Agora que já exploramos todas as principais portas lógicas utilizadas em circuitos combinacionais, vamos abstrair um pouco. Antes de partirmos para a implementação de circuitos digitais que fazem somas ou subtrações, vamos entender o que é uma soma quando estamos falando de números representados na base 2, ou seja, na matemática de Boole.

Partindo de algo mais familiar, vamos explorar um pouco como funciona a soma na base 10. Para isso, comecemos com um exemplo simples:

$$(2)_{10} \ +$$

$$\underline{(5)_{10}}$$

$$(7)_{10}$$

Parece simples – e é! –, mas vamos dar um passo adiante, somando os números 15 e 7 na base 10. A linha acima dos números corresponde ao "vai 1", ou seja, quando a soma de dois números de um dígito supera o maior símbolo que existe na representação de uma determinada base:

$$(1\ 0) = (\text{nesta linha, temos o "vai 1"})$$

$$(15)_{10} \ +$$

$\underline{(07)_{10}}$

$(22)_{10}$

A operação apresentada ilustra a situação de quando uma operação de soma gera um resultado para o qual não há símbolos que o representem em apenas um algarismo, exigindo que a operação seja representada com o recurso do "vai 1". Isso vale para qualquer base de representação numérica, ou seja, podemos aplicar o mesmo princípio à base 16, à base 8 e à base 2.

Na base 2, vamos começar nosso estudo de somadores por meio da análise da soma de dois números binários com apenas 1 bit cada. Observe que não se trata de uma operação OR, mas de uma soma:

$(0)_2 + (0)_2 = (0)_2$

$(0)_2 + (1)_2 = (1)_2$

$(1)_2 + (0)_2 = (1)_2$

$(1)_2 + (1)_2 = (10)_2$

Note que, para representar a situação em que a soma dá um resultado maior do que é possível representar em apenas um bit, tivemos de aumentar a representação do número de 1 bit para 2 bits. Esse bit mais significativo da soma de apenas 1 bit é equivalente ao "vai 1", o qual no jargão da técnica de projeto digital é chamado de "carry".

Observe que, se formos somar números com dois bits para representar o número, será necessário calcular o carry do resultado do bit menos significativo para poder fazer a soma do bit mais significativo.

Para ilustrar essa situação, vamos nos ater a apenas duas combinações das possibilidades de soma de dois números com dois bits, conforme as tabelas 5 e 6. Nos exemplos dados, podemos notar como o carry é utilizado.

Tabela 5 – Tabela de soma 1

Carry do bit mais significativo	Carry do bit menos significativo		
1	1		**Carry**
	0	1	**Número A**
	1	1	**Número B**
1	0	0	**Resultado**

Fonte: adaptado de Idoeta e Capuano, 1999.

Tabela 6 – Tabela de soma 2

Carry do bit mais significativo	Carry do bit menos significativo		
1	1		**Carry**
	1	1	**Número A**
	1	1	**Número B**
1	1	0	**Resultado**

Fonte: adaptado de Idoeta e Capuano, 1999.

Similar ao conceito de "vai 1" utilizado na soma, temos, na subtração, o conceito de "empresta 1", que no jargão dos circuitos digitais é chamado de "borrow".

IMPORTANTE

O borrow significa considerar um bit mais significativo na hora de fazer a operação de subtração entre dois números binários.

Para que o conceito fique mais claro, segue um exemplo de subtração de dois números de apenas 1 bit, em que todos os números estão na base 2.

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$0 - 1 = 1\ (\text{borrow} + 1)$

Note que, quando o bit do número do subtrator é maior do que o número subtraído, o borrow tem de ser representado. Isso fica mais claro no exemplo a seguir, com dois números representados em quatro bits.

Tabela 7 – Tabela de subtração 1

3	2	1	0	Índice	
1	1	0	0	$(12)_{10}$	**Subtraído**
1	0	0	1	$(9)_{10}$	**Subtrator**
0	1	1			**Borrow**
0	0	1	1	$(3)_{10}$	

Na tabela 7, notamos que no bit menos significativo (índice = 0) o subtrator é maior que o subtraído, o que gera um borrow no bit imediatamente mais significativo (índice = 1). A soma do subtrator (1) com o borrow (1) dá novamente 1, que é maior que o bit subtraído (1), gerando um novo borrow para o bit imediatamente mais significativo, que tem o índice 2.

A soma do borrow (2) com o bit subtrator (2) novamente dá 1, que tem o mesmo valor, e, nesse caso, o resultado dá zero, sem gerar um novo borrow para o bit mais significativo. Essa situação se mantém no bit mais significativo.

Considerações finais

Neste capítulo, abordamos circuitos mais avançados e com equivalências mais difíceis de serem intuídas. Além disso, começamos a trabalhar as técnicas para fazer contas usando circuitos lógicos, assunto que você vai explorar mais nos capítulos 4 e 5.

Referências

IDOETA, Ivan V.; CAPUANO, Francisco G. **Elementos de eletrônica digital**. São Paulo: Érica, 1999.