Stelvio Barboza

Conceitos de **computação II**

Senac

Dados Internacionais de Catalogação na Publicação (CIP)
(Jeane Passos de Souza - CRB 8ª/6189)

Barboza, Stelvio
Conceitos de computação II / Stelvio Barboza. – São Paulo : Editora Senac São Paulo, 2020. (Série Universitária)

Bibliografia.
e-ISBN 978-65-5536-223-7 (ePub/2020)
e-ISBN 978-65-5536-224-4 (PDF/2020)

1. Ciência da computação 2. Tecnologia da informação (TI) – Conceitos 3. Tecnologia da informação (TI) – Desenvolvimento 4. Tecnologia digital 5. Álgebra booleana I. Título. II. Série.

20-1174t CDD – 004
BISAC COM014000

Índice para catálogo sistemático
1. Ciência da computação 004

CONCEITOS DE COMPUTAÇÃO II

Stelvio Barboza

Administração Regional do Senac no Estado de São Paulo

Presidente do Conselho Regional
Abram Szajman

Diretor do Departamento Regional
Luiz Francisco de A. Salgado

Superintendente Universitário e de Desenvolvimento
Luiz Carlos Dourado

Editora Senac São Paulo

Conselho Editorial
Luiz Francisco de A. Salgado
Luiz Carlos Dourado
Darcio Sayad Maia
Lucila Mara Sbrana Sciotti
Jeane Passos de Souza

Gerente/Publisher
Jeane Passos de Souza (jpassos@sp.senac.br)

Coordenação Editorial/Prospecção
Luís Américo Tousi Botelho (luis.tbotelho@sp.senac.br)
Dolores Crisci Manzano (dolores.cmanzano@sp.senac.br)

Administrativo
grupoedsadministrativo@sp.senac.br

Comercial
comercial@editorasenacsp.com.br

Acompanhamento Pedagógico
Mônica Rodrigues dos Santos

Designer Educacional
Patrícia Pinheiro de Sant'Ana

Revisão Técnica
Marco Antonio Barreto

Preparação e Revisão de Texto
Juliana Ramos Gonçalves

Projeto Gráfico
Alexandre Lemes da Silva
Emília Corrêa Abreu

Capa
Antonio Carlos De Angelis

Editoração Eletrônica
Sidney Foot Gomes

Ilustrações
Sidney Foot Gomes

Imagens
Adobe Stock

E-pub
Ricardo Diana

Editora Senac São Paulo
Rua 24 de Maio, 208 – 3º andar
Centro – CEP 01041-000 – São Paulo – SP
Caixa Postal 1120 – CEP 01032-970 – São Paulo – SP
Tel. (11) 2187-4450 – Fax (11) 2187-4486
E-mail: editora@sp.senac.br
Home page: http://www.livrariasenac.com.br

Sumário

Capítulo 1

Introdução: circuitos lógicos

No mundo moderno, somos cercados por dispositivos eletrônicos, que usam tecnologias das mais variadas e são o resultado da evolução das técnicas de projeto e fabricação ao longo dos últimos séculos. Somos herdeiros, por exemplo, de pesquisas no campo da eletricidade e da matemática que remontam ao século XVIII.

Atualmente, em um mesmo dispositivo eletrônico, é muito comum que tenham sido usadas diversas técnicas de projeto e diversas tecnologias de fabricação dos componentes eletrônicos dos produtos que fazem a vida moderna ser o que é.

Dentre as técnicas de projeto utilizadas para desenvolver determinado produto eletrônico, é comum identificar claramente trechos do

circuito que utilizam técnicas de projeto analógico ou técnicas de projeto digital, e ainda trechos nos quais as duas técnicas se misturam – em geral, as interfaces entre os trechos de técnica analógica e trechos de técnica digital.

Resumidamente, os trechos de um produto eletrônico identificados como analógicos são aqueles em que o controle da tensão e do funcionamento do circuito se dá de forma contínua. Em geral, eles se baseiam nas propriedades físicas dos componentes eletrônicos envolvidos, como um regulador de tensão linear, que utiliza a curva de resposta de um diodo Zener reversamente polarizado para regular a tensão de saída do regulador de tensão.

Outro exemplo de circuito analógico são os amplificadores utilizados para condicionar o sinal de um microfone, sendo que o sinal captado é proporcional à pressão sonora percebida pelo microfone.

Por sua vez, os trechos do circuito que foram projetados utilizando a técnica digital são, em geral, trechos em que o controle do circuito se dá por sinais discretos de tensão elétrica (diferença de potencial em volts), ou seja, são trechos cujo funcionamento se dá por mudanças abruptas de tensão elétrica e que possuem apenas dois níveis de tensão elétrica válidos para o funcionamento do circuito. Mais adiante, vamos perceber que os dois níveis costumam identificar nível lógico 0 ou nível lógico 1, e por que isso é importante.

A tecnologia de fabricação dos componentes eletrônicos, em geral, determina os níveis de tensão elétrica que são válidos para identificar os níveis lógicos 0 e 1. Por exemplo, temos o padrão TTL, definido inicialmente na década de 1960, que tinha como tensões para os níveis lógicos 0 e 1, respectivamente, 0 V e 5 V.

Esse padrão foi sucedido por outros, que trabalham com níveis 0 V e 3,3 V ou 0 V e 1,8 V. Há, ainda, padrões que não identificam o nível lógico 0 como 0 V, como o padrão LVPECL, que identifica o nível 0 como 1,6

V e o nível lógico 1 como 2,4 V (HOLLAND, 2002), ou mesmo o padrão RS232 de comunicação serial, em que o nível lógico 0 corresponde a uma tensão entre 5 V e 15 V, e o nível lógico 1 corresponde a uma tensão entre -5 V e -15 V.

Não cabe, neste momento, discutir se determinado padrão é melhor ou pior que outro, ou por que foram inventados tantos padrões de tensão para representar os valores lógicos 0 e 1. O mais importante é perceber que os circuitos digitais trabalham com dois níveis distintos de tensão e que esses níveis correspondem aos níveis lógicos de que vamos tratar a seguir.

1 Sistemas binários e a relação com a álgebra de Boole

As técnicas de projeto digitais atualmente em uso tiveram sua base matemática inicialmente descrita na obra *An investigation of the laws of thouth on which are founded the mathematical theories of logic and probabilities*, escrita pelo matemático inglês George Boole e publicada em 1854.

Paralelamente a George Boole, Augustus De Morgan – em sua obra *Formal logic*, publicada em 1847, e em artigos subsequentes, publicados pela *Transactions of the Cambridge Philosophical Society* – criou as leis de De Morgan, que, em conjunto com as teorias de Boole, formam a base da matemática binária.

PARA SABER MAIS

Para saber mais sobre a trajetória e a importância de George Boole, busque o documentário *The genius of George Boole* (direção de Stephen Mizelas, Reino Unido, 2015, 59 min.).

Entretanto, apesar de as bases da matemática binária terem sido descritas no século XIX, seu uso prático em engenharia teve início apenas no século XX, quando o engenheiro norte-americano Claude Elwood Shannon aplicou as ideias e teorias de Boole na resolução de problemas relacionados ao projeto de sistemas de telefonia da época: era o início das centrais telefônicas automáticas. Como fruto desse trabalho, Shannon publicou o artigo "Symbolic analisys of relay and switching circuits", em 1938.

Nesse artigo pioneiro, ele descreve o uso prático das ideias de Boole, o modo de aplicar suas equações e como isso podia ser implementado no nível físico dos relês utilizados naquela época para a implementação dos circuitos de comutação e controle das centrais telefônicas automáticas, fazendo uma comparação entre os cálculos de lógica propostos por Boole e o seu uso prático.

Ainda não se tratava de um projeto de computador de uso geral, mas da implementação de uma técnica de projeto que anos mais tarde viria a tornar viável a construção de computadores de uso geral, ou seja, máquinas que poderiam ser programadas posteriormente à sua fabricação e cuja função seria dada pelo programa a ser carregado nela.

Essa técnica de projeto desenvolvida por Shannon viria a ser a base de toda a técnica de projeto digital no nível lógico, independentemente de como seria implementado o nível físico (componentes eletrônicos).

Em termos de implementação, podemos entender o trabalho de Shannon da seguinte forma: quando há tensão para acionar a bobina de um relê, isso corresponde a um nível lógico 1, ou verdadeiro, e quando não temos tensão para acionar a bobina de um relê, temos um nível lógico 0, ou falso.

Dessa forma, podemos perceber que, tendo apenas esses dois estados, a implementação física das equações de Boole se torna viável mesmo com uma técnica de fabricação bastante simples (a utilização de relês).

Hoje, os circuitos lógicos são feitos utilizando transistores, que possuem vantagens evidentes em relação às implementações originais de Shannon, como o tamanho e o tempo de transição de um estado a outro, sem contar o consumo de energia.

2 Representando números

Alguns conceitos, depois que os utilizamos por muito tempo, parecem muito simples, e não paramos muito para pensar neles. Um deles diz respeito à representação de números, para que eles possam ser úteis em nosso dia a dia.

A forma mais "natural" de representação numérica, que as pessoas que não são da área conhecem, é a forma decimal, em que há dez símbolos básicos para a representação de quantidades, além de uma lei de formação segundo a qual podemos combinar os símbolos a fim de representar qualquer quantidade de determinado item que estejamos contando.

A lei de formação do sistema decimal pode ser descrita como um polinômio, em que cada elemento corresponde a um multiplicador e a uma mantissa. Antes de partir para um exemplo, vamos lembrar uma das propriedades da potencialização:

$10^0 = 1$

$10^1 = 10$

$10^2 = 100$

$10^3 = 1.000$

$10^4 = 10.000$

$10^5 = 100.000$

Para ilustrar isso, observe a decomposição do número 1976.

Tabela 1 – Decomposição do número 1976

	UNIDADE	DEZENA	CENTENA	MILHAR
1976 =	(6 × 1) +	(7 × 10) +	(9 × 100) +	(1 × 1.000)

A decomposição apresentada pelo quadro é equivalente a:

$$1976 = (6 \times 10^0) + (7 \times 10^1) + (9 \times 10^2) + (1 \times 10^3)$$

Nessa expressão, cada número da representação decimal multiplica a base elevada ao número da posição.

Tabela 2 – Decompondo números na base 10

3	2	1	0
10^3	10^2	10^1	10^0
1.000	100	10	1
1 x 1.000	9 x 100	7 x 10	6 x 1
1	9	7	6

A partir da regra apresentada, podemos extrair o polinômio para a base 10:

$$(n)_{10} = n_i \times 10^i + n_{i-1} \times 10^{(i-1)} + n_{i-2} \times 10^{(i-2)} + \ldots + n_2 \times 10^2 + n_1 \times 10^1 + n_0 \times 10^0$$

Entretanto, a regra para a base 10 também se aplica a qualquer base de representação, basta termos símbolos suficientes que equivalham a todos os elementos da base que desejamos representar.

Assim, podemos criar o que é chamado de polinômio geral:

$$(n)_b = n_i \times b^i + n_{i-1} \times b^{(i-1)} + n_{i-2} \times b^{(i-2)} + \ldots + n_2 \times b^{(2)} + n_1 \times b^{(1)} + n_0 \times b^{(0)}$$

Nesse polinômio, b é a base em que queremos representar o valor n, e n_i são as decomposições parciais do número n na base b.

Podemos usar o polinômio geral para representar números em qualquer base e, por consequência, podemos usá-lo para fazer conversões de base.

3 Conversão da base 10 para a base 2

Antes de tratarmos da conversão da base 10 para a base 2, precisamos definir o que significa um bit. Um bit é a menor unidade de informação existente. Em um circuito lógico, representa se determinado circuito está ligado ou desligado; porém, em um sistema para representar um número, um bit simboliza a menor quantidade que é possível representar, no caso, zero ou uma unidade. Ou seja, sempre que estivermos falando sobre sistemas de representação binário, haverá apenas a possibilidade, em cada bit da representação, de ter ou não a quantidade que aquela posição representa. Isso ficará mais claro ao longo deste capítulo, quando definirmos a base 16.

No sistema binário, a cada bit, só temos duas possibilidades: ou a quantidade da posição existe, ou ela é nula. Para simplificar a representação, particularmente quando estamos fazendo os cálculos, é necessário utilizar símbolos que representem os elementos da base com a qual estamos trabalhando.

Os dois símbolos que temos para representar quantidades na base 2 são o 0 e o 1, em que 0 representa a ausência daquela quantidade e 1, a presença daquela quantidade.

A regra de formação dos números na base 2 é:

$$(n)_2 = n_i \times 2^i + n_{i-1} \times 2^{(i-1)} + n_{i-2} \times 2^{(i-2)} + \ldots + n_2 \times 2^{(2)} + n_1 \times 2^{(1)} + n_0 \times 2^{(0)}$$

Como exemplo, temos:

$$(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

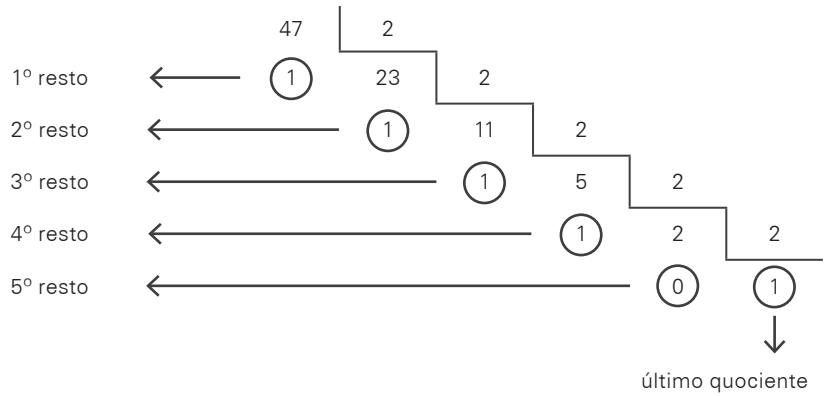
$$= 32 + 0 + 8 + 4 + 0 + 1$$

$$= (45)_{10}$$

E como fazemos para converter a base 10 para a base 2?

A forma mais simples é utilizando o resto por divisões sucessivas na referida base (IDOETA, 1999, p. 5). A técnica se resume a dividir o número em sua base original por 2, até obtermos o último quociente diferente de zero. Esse processo pode ser demonstrado de uma forma simplificada na figura 1, que representa a conversão do número $(47)_{10}$ para a base 2.

Figura 1 – Técnica das divisões sucessivas



Observe que, para compor o número na base para a qual queremos converter, estamos pegando o resto de cada divisão até o último quociente, sendo que o primeiro resto é o dígito menos significativo, e o último quociente, o dígito mais significativo.

No jargão da computação e da eletrônica digital, o bit menos significativo é chamado de LSB (least significant bit), e o bit mais significativo é o MSB (most significant bit). Para a representação do número binário, usamos o esquema apresentado na tabela 3.

Tabela 3 – Representação do número binário obtido na técnica de divisões sucessivas

ÚLTIMO QUOCIENTE	5° RESTO	4° RESTO	3° RESTO	2° RESTO	1° RESTO
1	0	1	1	1	1

A representação binária do número 47 decimal gera o número binário $(101111)_2$, que é equivalente a $(47)_{10}$.

Também podemos descrever essa operação por meio das equações a seguir.

$$47 \,/\, 2 = 23 \text{ com resto } 1$$

Podemos escrever isso da seguinte forma:

$$(23 \times 2) + 1 = 47 \text{ (Equação A)}$$

Também podemos representar 23 por uma divisão:

$$23 \,/\, 2 = 11 \text{ com resto } 1$$

Que, por sua vez, podemos representar por:

$$(11 \times 2) + 1 = 23 \text{ (Equação B)}$$

Substituindo 23 da equação A pela equação B, temos:

$$(11 \times 2 + 1) \times 2 + 1 = 47$$

Podemos reescrever isso de uma outra forma, mais adequada:

$$(11 \times 2) \times 2 + 1 \times 2 + 1 = 47$$

$$11 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47 \text{ (Equação C)}$$

Podemos repetir esse processo com o número 11:

$$11 \,/\, 2 = 5 \text{ com resto } 1$$

$$(2 \times 5) + 1 = 11 \text{ (Equação D)}$$

Substituindo 11 na equação C pela equação D, temos:

$$(11) \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$$

$$((5 \times 2) + 1) \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$$

$(5 \text{ x } 2) \times 2^2 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$5 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$ (Equação E)

Repetindo o processo, temos:

$5 / 2 = 2$ com resto 1

$(2 \times 2) + 1 = 5$

$(5) \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$((2 \times 2) + 1) \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$(2 \times 2) \times 2^3 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$(2^2) \times 2^3 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

$2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$ (Equação F)

Podemos representar a equação F de uma forma mais conveniente:

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 47$

Por sua vez, podemos colocar a equação acima em uma tabela, de modo que a representação na base 2 fique mais clara.

Tabela 4 – Representando um número binário na forma de uma tabela

2^5	2^4	2^3	2^2	2^1	2^0	101111 = 47
1	0	1	1	1	1	

O resultado apresentado na tabela 4, obtido por meio de equações, é equivalente ao resultado obtido por meio da primeira técnica de conversão, usando o resto da divisão por 2.

4 Convertendo para a base 16

A segunda base mais importante para a representação numérica, em termos de computação, é a base 16, que também é conhecida como hexadecimal (IDOETA, 1999, p. 19).

A base hexadecimal é composta pelos dez algarismos tradicionais da base decimal somados com mais seis símbolos, a fim de completar o número de símbolos necessário para representar, em um único algarismo, os 16 símbolos usados para a formação da base.

O conjunto de símbolos que formam a base hexadecimal e seus valores equivalentes na representação decimal estão representados na tabela 5.

Tabela 5 – Equivalência entre os números hexadecimais e decimais

HEXADECIMAL	DECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12

(*Cont.*)

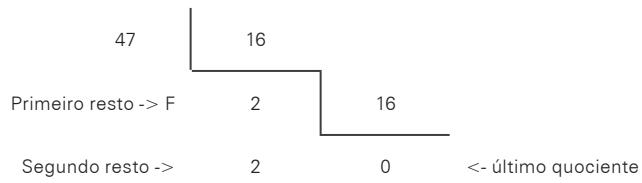
HEXADECIMAL	DECIMAL
D	13
E	14
F	15

Na tabela 5, observe que o valor 10 em decimal corresponde ao símbolo A em hexadecimal, que o valor 11 corresponde ao símbolo B, e assim sucessivamente, até o F, que corresponde à quantidade 15.

A regra de conversão da base 10 para a base 16 é a mesma da base 10 para a base 2, ou seja, fazem-se divisões sucessivas do número representado pelo número de símbolos da base de destino e usa-se o resto da divisão como os algarismos para representar o número.

Vamos tomar como exemplo o mesmo valor $(47)_{10}$, que utilizamos no exemplo de conversão da base 10 para a base 2.

Figura 2 – Divisões sucessivas na base 16



Na figura, podemos observar que o primeiro resto, que em decimal seria 15, na base hexadecimal é representado pela letra F.

Assim como no exemplo para a base 2, o elemento mais significativo é o último quociente, que no caso é 0, seguido pelo último resto, que no caso é o símbolo 2, e o menos significativo é o primeiro resto, que no caso corresponde ao símbolo F.

Tabela 6 – Representação do resultado das divisões sucessivas na base 16

ÚLTIMO QUOCIENTE	2° RESTO	1° RESTO
0	2	F

Utilizando equações para descrever a mesma operação apresentada, temos:

$$47 \,/\, 16 = 2 \text{ com resto F}$$

O que podemos escrever como:

$$(2 \times 16^1) + \mathrm{F} \times 16^0 = 47 \text{ (Equação A)}$$

Podemos representar 2 também por uma divisão:

$$2 \,/\, 16 = 0 \text{ com resto } 2$$

Que, por sua vez, podemos representar por:

$$(0 \times 16) + 2 = 2 \text{ (Equação B)}$$

Substituindo 2 da equação A pela equação B, temos:

$$(0 \times 16 + 2) \times 16^1 + \mathrm{F} \times 16^0 = 47$$

$$(0 \times 16 \times 16) + 2 \times 16^1 + \mathrm{F} \times 16^0 = 47$$

$$0 \times 16^2 + 2 \times 16^1 + \mathrm{F} \times 16^0 = 47$$

E podemos simplificar a equação acima por 02F.

5 Nibbles e bytes

Uma forma de entender a equivalência entre os números binários, hexadecimais e decimais é por meio da associação apresentada na tabela 7, a seguir.

Tabela 7 – Representação dos números de 0 a 15 em hexadecimal, binário e decimal

HEXADECIMAIS	BINÁRIOS	DECIMAIS
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

É possível notar que, no padrão hexadecimal de representação de números, cada símbolo na coluna “Hexadecimais” corresponde a 4 bits na coluna “Binários”. Isso tem um bom motivo: no início da computação, havia implementações em que um byte era composto de 6 bits para implementar os códigos BCD, pois com 6 bits — ou seja, 64 caracteres

— ficava viável representar o alfabeto alfanumérico completo e alguns caracteres especiais. Em impressoras e terminais teletipo (TTY), conectados por meio de interfaces seriais, existiam muitas implementações na comunicação de dados, em que 1 byte correspondia a 7 bits, e atualmente ainda se pode configurar uma interface RS232C para operar em 7 bits de dados.

A primeira codificação em que 1 byte foi implementado com 8 bits foi feita pela IBM, com a criação do código EBCDIC, em 1960. Como a IBM era a maior empresa do setor e as demais empresas queriam que seus equipamentos fossem compatíveis com as padronizações dela, padronizou-se que 1 byte seria formado por 8 bits. Posteriormente, em 1961, também surgiu o código ASCII, de 8 bits.

O fato de padronizar um byte com 8 bits fez com que os programadores e engenheiros de computadores da época buscassem uma forma mais fácil de representar os números e comandos em vez de usar sequências de 8 bits.

Como não temos 256 símbolos imprimíveis no teclado, buscou-se uma forma mais simples de representar um byte, mas que fosse uma tradução direta do código binário. Foi aí que surgiram as ideias para o padrão hexadecimal.

Se você observar, com dois conjuntos de 4 bits, temos um byte. Ou seja, com dois símbolos hexadecimais podemos formar um byte, sem qualquer sombra de dúvida de como isso pode ser representado no código binário.

Um byte de 8 bits pode representar de 0 a 255 em decimal ou de 00 a FF em hexadecimal. Para ilustrar como essa forma de agrupamento é importante e simplifica a representação numérica, vamos considerar um número binário com 24 bits:

0011 1111 0100 0101 1011 1010

Podemos representá-lo em hexadecimal:

3F 45BA

E podemos montar um pequeno quadro para ilustrar como a conversão é direta.

Tabela 8 – Representação de um número em binário e hexadecimal

0011	1111	0100	0101	1011	1010
3	F	4	5	B	A

Observe que todo o trabalho de conversão pode ser feito de 4 em 4 bits, que correspondem a apenas um símbolo em hexadecimal. Essa é a forma mais simples e direta de conversão entre as duas bases, com pouca ou nenhuma operação matemática, bastando consultar a tabela 7 (p. 20).

Obviamente, é possível aplicar o mesmo polinômio geral para a conversão da base 16 para a base 2, ou a potenciação para converter da base 2 para a base 16, porém, o método apresentado é muito mais direto.

Considerações finais

Neste capítulo, vimos como podemos usar elementos muito simples para representar números dentro de uma máquina e como podemos converter o mesmo número em diversas bases, tornando mais fácil a representação dos números e símbolos de que precisamos ao trabalharmos em um projeto digital ou escrevermos um programa de computador.

Essas são as ferramentas básicas para qualquer sistema que tenha componentes digitais, seja um sistema pequeno e simples ou um grande e complexo.

Referências

BOOLE, George. **An investigation of the laws of thouth on which are founded the mathematical theories of logic and probabilities**. Cambridge: Cambridge University Press, 2009. (Cambridge Library Collection – Mathematics.)

DE MORGAN, Augustus. **Formal logic**. Londres: Taylor & Walton, 1847. Disponível em: https://books.google.com.br/books?id=HscAAAAAMAAJ&hl=pt-BR&pg=PR5#v=onepage&q&f=false. Acesso em: 12 ago. 2020.

HOLLAND, Nick. **Interfacing between LVPECL, VML, CML, and LVDS levels**. Texas: Texas Instruments – Application Report, dez. 2002. Disponível em: https://www.ti.com/lit/an/slla120/slla120.pdf. Acesso em: 5 mar. 2020.

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

SHANNON, Claude E. Symbolic analisys of relay and switching circuits in electrical engineering. **Transactions American Institute of Electrical Engineers**, [*s. l.*], vol. 57, n. 12, p. 713-723, dez. 1938.

Capítulo 2

Portas lógicas I

Uma das bases da técnica de projeto digital é a percepção de que qualquer informação ou valor pode ser representado de forma binária, seja por meio de zeros e uns, de falsos e verdadeiros ou de ligados e desligados. Tudo do mundo externo ao circuito que vamos representar dentro do nosso projeto, usando as técnicas de projeto digital, será feito com valores binários. Isso se aplica mesmo a valores de variáveis tipicamente contínuas, como temperatura, pressão etc. — com as quais temos de usar as técnicas de conversão analógica para digital e conversão digital para analógica —; ainda assim, estamos falando de valores e informações que são representados usando valores binários.

Neste capítulo, vamos começar a entender como podemos manipular essas informações e esses valores para que eles sejam úteis dentro de um produto que foi feito com as técnicas de projeto digital. Vamos iniciar com as operações mais básicas, que são os blocos fundamentais para a construção física de qualquer dispositivo que utiliza as técnicas de projeto digital.

Qualquer operação para tratar os valores representados em binário, do ponto de vista físico, será implementada por meio dos circuitos mais básicos, que vamos descrever neste capítulo — e isso inclui todas as operações que qualquer software pode fazer. Por exemplo, para que um computador possa fazer a soma de dois números, a operação matemática será feita por um circuito digital, que é composto pelos circuitos básicos descritos a seguir.

1 Portas lógicas AND, OR, NOT

Na técnica de projeto digital, as portas lógicas são definidas como um circuito eletrônico que tem por função executar operações lógicas.

Atualmente, essas funções lógicas são tipicamente implementadas usando-se transistores em circuitos integrados, e podemos encontrá-los sob diversos tipos de encapsulamento. Podemos encontrar chips que implementam apenas algumas poucas portas lógicas, que são acessíveis diretamente pelos terminais, ou chips com muitos milhões de portas lógicas, que podem se apresentar na forma de processadores, microcontroladores, chips lógicos programáveis em campo etc.

Entretanto, esses circuitos podem ser implementados de diversas formas, inclusive com o uso de tecnologias que estão disponíveis desde a década de 1930; nesse caso, estamos falando dos relês como elemento ativo.

Um relê é um dispositivo eletromecânico composto por uma alavanca metálica com uma mola e um eletroímã. A alavanca metálica liga dois (ou mais) contatos elétricos, que podem ser conectados ou desconectados pela ação do eletroímã. Para que isso fique mais claro, vamos explicar como funciona um relê normalmente fechado. Quando não há corrente elétrica passando pelo eletroímã, os contatos elétricos desse dispositivo ficam conectados, ou seja, trata-se de uma chave elétrica fechada; por isso, como o normal do relê é operar sem corrente elétrica na bobina, ele é um dispositivo normalmente fechado. No entanto, quando passamos corrente elétrica pela bobina do relê, a alavanca é acionada e o contato elétrico é desfeito, ou seja, a chave elétrica do nosso relê se torna aberta.

Em suma, um relê normalmente aberto opera no inverso de um relê normalmente fechado, ou seja, os contatos do relê se encontram na forma de uma chave elétrica aberta quando não há corrente elétrica passando pela bobina, e os contatos são fechados quando há corrente elétrica pela bobina.

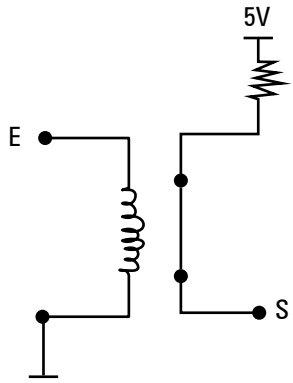
Fundamentalmente, qualquer circuito eletrônico que implemente uma das funções lógicas básicas pode ser considerado uma porta lógica, não importa como tenha sido construído do ponto de vista físico. Isso tornou a técnica de projeto digital perene em meio à tremenda evolução tecnológica das últimas décadas.

IMPORTANTE

Por mais que a tecnologia de fabricação dos dispositivos tenha evoluído, as equações e a matemática que eles implementam são as mesmas.

Para ilustrar isso de uma forma bastante simples, na figura 1, a seguir, temos uma porta NOT implementada com um relê normalmente fechado.

Figura 1 — Porta lógica a relê



Analisando esse circuito, identificamos o terminal E como entrada e o terminal S como saída. Dado que a tensão para o acionamento da bobina é de 5 V, quando esse nível de tensão for aplicado no terminal E, a chave interna do relê será aberta, e o terminal S ficará sem alimentação da fonte, o que vai levar a sua tensão para 0 V (em uma aplicação real, o terminal S estaria ligado na bobina de outro relê, o que levaria o terminal para 0 V). Se aplicarmos uma tensão de 0 V na entrada E do circuito, a bobina deixará de ser acionada, e a tensão no terminal S será de 5 V.

Isso nos leva à nossa primeira tabela-verdade, expressa na tabela 1, a seguir.

Tabela 1 – Tabela-verdade 1

E	S
5 V	0 V
0 V	5 V

Entretanto, para simplificar e padronizar a notação dos projetos digitais, se convencionou que os símbolos a serem usados são 0 e 1, ou falso e verdadeiro, respectivamente.

Assim, é comum considerarmos que 0 V corresponde a falso (ou desligado) e 5 V corresponde a verdadeiro (ou ligado), porém, como já exposto no capítulo anterior, cada tecnologia de fabricação adota uma padronização diferente de tensões para representar 0 e 1 (falso e verdadeiro).

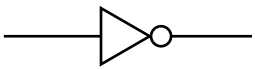
Remodelando nossa tabela-verdade para utilizar a simbologia tradicional, chegamos à tabela 2.

Tabela 2 – Tabela-verdade 2

E	S
1	0
0	1

A tabela 2 é a primeira forma de representação da função lógica de negação (NOT). Além dessa tabela-verdade, podemos representar a porta lógica NOT pelo símbolo usado em esquemas elétricos, apresentado na figura 2.

Figura 2 – Porta NOT (negação)



Outra forma de representar a função lógica da negação é por meio da representação algébrica:

$$S = \overline{E}$$

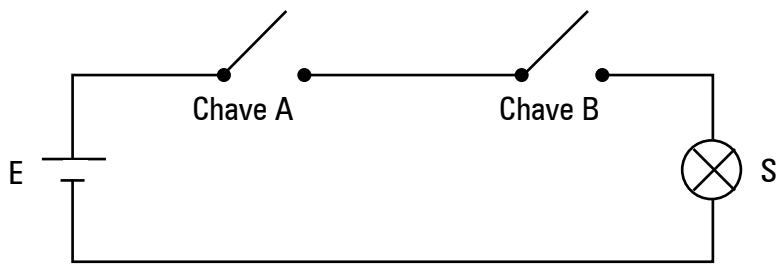
Note que, nessa representação, a negação é simbolizada por uma linha sobre a expressão que ela está negando. Mais adiante, vai ficar

mais claro como podemos usar essa notação para fazer as simplificações das expressões lógicas.

A segunda porta lógica básica que vamos estudar é a porta lógica AND ("e"). Podemos descrever a função lógica AND como uma multiplicação de duas ou mais variáveis booleanas, em que todas as entradas da porta lógica devem estar acionadas para que a saída seja verdadeira.

O circuito mais simples que podemos usar para representar a função lógica AND seria o de chaves em série, em que as chaves são as entradas. Quando a chave está fechada, ela representa 1 (verdadeiro), e quando está aberta, representa 0 (falso). Já a saída pode ser representada pela presença ou pela ausência de tensão para acionar uma lâmpada, por exemplo, como na figura 3, a seguir.

Figura 3 – Implementando uma porta AND



No circuito da figura 3, podemos notar que a lâmpada só ficará acesa se a chave A e a chave B estiverem acionadas (fechadas), ou seja, em estado verdadeiro.

Podemos descrever esse circuito por meio de uma tabela-verdade, conforme expresso na tabela 3.

Tabela 3 – Tabela-verdade 3

ENTRADAS		SAÍDA
Chave A	**Chave B**	**S**
Aberta	Aberta	Apagada
Aberta	Fechada	Apagada
Fechada	Aberta	Apagada
Fechada	Fechada	Acesa

Se adotarmos que fechado corresponde a 1 (verdadeiro) e aberto corresponde a 0 (falso), bem como que o estado "aceso" equivale a verdadeiro e "apagado", a falso, podemos refazer a tabela de forma mais padronizada, conforme a tabela 4.

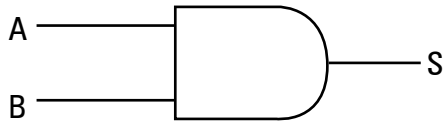
Tabela 4 – Tabela-verdade 4

ENTRADAS		SAÍDA
Chave A	**Chave B**	**S**
0	0	0
0	1	0
1	0	0
1	1	1

Ou seja, a função lógica AND só gera uma saída verdadeira quando todas as entradas são verdadeiras, independentemente do número de entradas da porta.

A forma gráfica de representar a função lógica AND de duas entradas é a da figura 4, a seguir.

Figura 4 – Representação gráfica de uma porta AND



O número de entradas, em termos teóricos, é ilimitado, porém, na prática, existe um limite, em parte por questões econômicas e em parte em decorrência da tecnologia que se usa para implementar a porta lógica. Entretanto, se uma função exigir muitas entradas, pode-se facilmente combinar várias portas AND menores, que são equivalentes a uma maior. Esse tópico será aprofundado mais adiante.

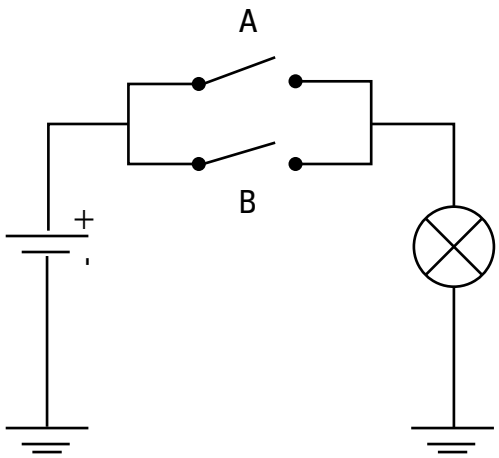
Do ponto de vista de expressão algébrica, a operação AND é representada por um ponto (.) entre duas variáveis booleanas, como exemplificado na expressão a seguir:

$$S = A \,.\, B$$

A terceira porta lógica que vamos analisar é a porta lógica OR ("ou"). Podemos descrevê-la como uma soma de duas ou mais variáveis booleanas, em que, se qualquer uma das entradas for verdadeira, a saída será verdadeira.

Como forma de ilustrar essa função lógica, vamos analisar o circuito da figura 5.

Figura 5 – Construindo uma porta OR



No circuito da figura 5, podemos notar que a lâmpada ficará acesa se qualquer uma das chaves estiver acionada. Ou seja, se a chave A ou a chave B estiverem acionadas (fechadas), a lâmpada ficará acesa.

Podemos descrever esse circuito por meio de uma tabela-verdade, conforme a tabela 5, a seguir.

Tabela 5 – Tabela-verdade 5

ENTRADAS		SAÍDA
Chave A	**Chave B**	**S**
Aberta	Aberta	Apagada
Aberta	Fechada	Acesa
Fechada	Aberta	Acesa
Fechada	Fechada	Acesa

Se adotarmos que fechado corresponde a 1 (verdadeiro) e aberto corresponde a 0 (falso), bem como que o estado "aceso" é equivalente a verdadeiro e "apagado", a falso, podemos refazer a tabela de uma forma mais padronizada, conforme a tabela 6.

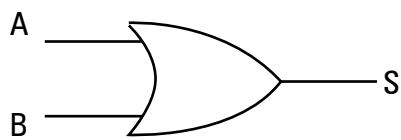
Tabela 6 – Tabela-verdade 6

ENTRADAS		SAÍDA
Chave A	**Chave B**	**S**
0	0	0
0	1	1
1	0	1
1	1	1

Ou seja, a função lógica OR gera uma saída verdadeira quando qualquer uma das entradas estiver verdadeira, independentemente do número de entradas da porta.

A forma gráfica de representar a função lógica OR de duas entradas é a da figura 6, a seguir.

Figura 6 – Representação gráfica de uma porta OR



Novamente, o número de entradas, em termos teóricos, é ilimitado, porém, na prática, os limites são os mesmos observados para a porta AND. Além disso, similarmente, se uma função exigir muitas entradas,

pode-se facilmente combinar várias portas OR menores, que são equivalentes a uma maior.

Do ponto de vista da expressão algébrica, a operação OR é representada por um sinal de soma + entre duas variáveis booleanas, como exemplificado na expressão:

$$S = A + B$$

2 Tabelas-verdade

Como você já deve ter percebido, uma das formas de descrever as portas lógicas simples é por meio de suas tabelas-verdade. As tabelas-verdade, entretanto, podem descrever quaisquer circuitos digitais combinacionais. Elas consistem em uma coluna que simboliza a saída do circuito combinacional e outra coluna que remete a cada variável booleana do circuito.

O número de linhas desse tipo de tabela depende do número de combinações possíveis das variáveis booleanas de entrada. Assim, se tivermos um circuito com três entradas, teremos 2^3 combinações, e se tivermos quatro entradas, teremos 2^4 combinações.

O procedimento básico para usar as tabelas-verdade consiste em gerar uma tabela com todas as combinações possíveis de entradas e, em seguida, anotar, para cada combinação, se o valor da saída é falso ou verdadeiro.

Um primeiro exemplo de circuito com três entradas é mostrado na tabela 7.

Tabela 7 – Tabela-verdade 7

A	B	C	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Na tabela-verdade apresentada, podemos ler que a saída será verdadeira em quatro situações:

- quando A = 0, B = 0 e C = 1;
- quando A = 0, B = 1 e C = 0;
- quando A = 1, B = 0 e C = 0; e
- quando A = 1, B = 0 e C = 1.

No próximo subcapítulo, vamos explorar como transformamos a tabela-verdade em expressões lógicas e em circuitos lógicos.

Quanto à transformação de uma lista de requisitos de funcionalidade em um circuito lógico, uma das formas mais fáceis de fazê-lo é por meio de tabelas-verdade. Basicamente, a cada linha da lista de requisitos, anotamos se a saída do circuito combinatório deve ser falsa ou verdadeira.

Como exemplo, vamos analisar o texto a seguir, que é uma maneira informal de descrever o que queremos que um circuito combinatório que vamos projetar faça.

> "Estamos projetando um forno de micro-ondas e precisamos de um circuito que proteja o proprietário de acidentes.
>
> Se o proprietário colocasse a mão dentro do forno ligado, haveria um acidente, portanto, o forno não pode ligar se a porta estiver aberta. Além disso, precisamos que a luz interna fique acesa quando a porta estiver aberta. Também precisamos que a lâmpada fique ligada quando o forno estiver ligado, e que ela fique desligada se a porta estiver fechada e o forno, desligado.
>
> Quando o proprietário pressionar o botão 'ligar', se a porta estiver fechada, o forno vai ligar. Quando o proprietário pressionar 'ligar', se a porta estiver aberta, o forno não vai ligar."

Nesse texto, fizemos uma descrição parcial do projeto de um forno de micro-ondas. Agora, vamos transformar o texto em variáveis booleanas, começando pelas entradas:

- temos o botão para acionar o forno, que vamos chamar de A;
- por simplicidade, vamos adotar que o botão A, quando pressionado, fica ligado; para ser desligado, ele precisa ser pressionando para o outro lado;
- vamos adotar que, quando quisermos o forno ligado, a variável A estará em verdadeiro (1); quando quisermos o forno desligado, a variável estará em falso (0);
- temos o sensor da porta, que vamos chamar de P; quando a porta estiver aberta, estará em verdadeiro, e quando estiver fechada, estará em falso.

Agora, vamos definir as saídas:

- a primeira saída é a lâmpada, que vamos chamar de L; quando L estiver em verdadeiro, a lâmpada estará ligada, e quando estiver em falso, a lâmpada estará desligada;
- a segunda saída é o magnétron – o elemento que aquece os alimentos dentro do forno –, que vamos chamar de M; quando M estiver verdadeiro, o forno estará aquecendo, e quando M estiver em falso, o forno estará desligado.

Como temos duas variáveis de entrada e quatro possibilidades de entrada, isso vai nos dar a tabela 8, a seguir.

Tabela 8 – Tabela-verdade 8

A	P	L	M
Não acionado	Fechado	Desligado	Desligado
Não acionado	Aberto	Ligado	Desligado
Acionado	Fechado	Ligado	Ligado
Acionado	Aberto	Ligado	Desligado

Observe que a tabela consegue expressar, de forma muito resumida, o que precisamos que o circuito faça.

Como temos duas saídas, podemos fazer duas tabelas separadas para cada saída, ou podemos resumir as informações em uma única tabela, a fim de facilitar o entendimento.

IMPORTANTE

Muitas vezes, é possível aproveitar circuitos lógicos em mais de uma saída.

Trocando as palavras pelos significados que demos para elas, em termos de verdadeiro e falso, temos a tabela 9.

Tabela 9 – Tabela-verdade 9

A	P	L	M
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	0

3 Expressões booleanas

Podemos entender as expressões booleanas como uma forma algébrica de representar os circuitos digitais. Desse modo, temos uma ferramenta para manipular e otimizar os circuitos.

As operações básicas da álgebra booleana são fundamentalmente as mesmas que as portas lógicas já apresentadas fazem, ou seja, as das operações lógicas AND, OR e NOT.

Como já destacamos anteriormente, as operações lógicas podem ser representadas por:

- AND ⟶ A AND B = A . B = B . A = B AND A
- OR ⟶ A OR B = A + B = B + A = B OR A
- NOT ⟶ NOT A = A = $\overline{A}$

Assim, temos:

$0 . 0 = 0$

$1 . 1 = 1$

$0 . 1 = 1 . 0 = 0$

$0 + 0 = 0$

$1 + 1 = 1$

$0 + 1 = 1 + 0 = 1$

Não $1 = 1' = \overline{1} = 0$

Não $0 = 0' = \overline{0} = 1$

Isso nos leva aos principais postulados de Boole (IDOETA; CAPUANO, 1999), conforme descrição a seguir.

Considere X, Y e Z variáveis lógicas distintas.

$0*X = 0$

$1*X = X$

$X*X = X$

$\overline{X}*X = 0$

$0 + X = X$

$1 + X = 1$

$X + X = X$

$\overline{X} + X = 1$

$\overline{\overline{X}} = X$

Comutativas:

$X + Y = Y + X$

$X*Y = Y*X$

Associativas:

$$X + (Y + Z) = (X + Y) + Z$$

$$X*(Y*Z) = (X*Y)*Z$$

Distributivas:

$$X*(Y + Z) = (X*Y) + (X*Z)$$

Veja que esses postulados são bastante semelhantes à álgebra tradicional, porém, têm suas particularidades.

Além desses postulados, é muito importante ter em mente e treinar o uso dos dois teoremas de De Morgan (IDOETA; CAPUANO, 1999):

- 1° Teorema: $\overline{A} + \overline{B} = \overline{A \cdot B}$
- 2° Teorema: $\overline{A} \cdot \overline{B} = \overline{A + B}$

Podemos demonstrá-los por meio da tabela 10, a seguir.

Tabela 10 – Tabela-verdade 10

1° TEOREMA			
A	**B**	**Termo 1**	**Termo 2**
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

2° TEOREMA			
A	**B**	**Termo 1**	**Termo 2**
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

3.1 Usando as expressões booleanas

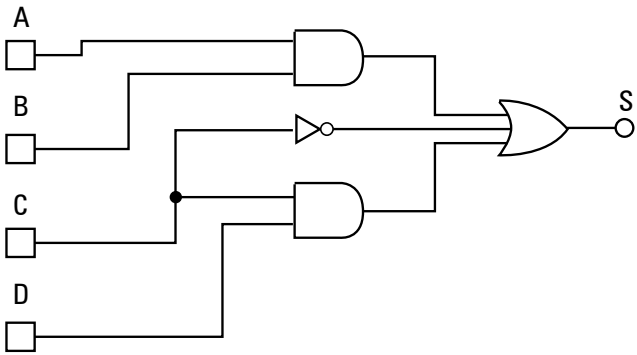
O uso das expressões booleanas exige treino e certamente não é algo simples à primeira vista. Por isso, vamos usar as expressões e

suas representações em termos de circuitos lógicos, assim conseguiremos mostrar o uso real dessa ferramenta de projeto digital.

O primeiro exemplo pode ser conferido na figura 7, cuja equação algébrica é:

$$S = (A \,.\, B) + \overline{C} + (C \,.\, D)$$

Figura 7 – Primeiro exemplo de expressão booleana



3.2 Criando expressões booleanas a partir de tabelas-verdade

Até este ponto do capítulo, trabalhamos para gerar as tabelas-verdade a partir das expressões booleanas, porém, em geral, o que ocorre é o processo inverso: a partir da descrição do problema a ser resolvido geramos a tabela-verdade, e da tabela-verdade geramos as expressões booleanas.

O processo de conversão, entretanto, é bastante simples, embora possa ser trabalhoso, dependendo do número de variáveis de entrada que o circuito possui.

Basicamente, o processo consiste, primeiro, em escolher se vamos trabalhar com lógica positiva ou lógica negativa.

Essa escolha se dá pela observação, por parte da pessoa que está fazendo o projeto, da quantidade de combinações em que a saída é verdadeira e da quantidade de combinações em que a saída é falsa.

Uma vez feita a escolha, identificamos os casos em que apenas uma a situação escolhida ocorre, e escrevemos a expressão lógica correspondente àquela combinação.

Como exemplo, vamos analisar a tabela 11.

Tabela 11 – Tabela-verdade 11

LINHA	A	B	S
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

Para fins didáticos, vamos analisar os casos em que a saída é verdadeira, embora seja muito mais vantajoso fazer o processo com a situação em que a saída é falsa.

A primeira situação em que a saída é verdadeira está na linha 0, cuja expressão lógica seria:

$$S = \overline{A} \cdot \overline{B}$$

A segunda situação ocorre na linha 2, que nos dá a expressão:

$$S = A \cdot \overline{B}$$

E a última situação ocorre na linha 3:

$$S = A \cdot B$$

Se fizermos um OU entre as três expressões obtidas até agora, vamos obter a expressão que descreve a tabela 11:

$$S = (\overline{A} \,.\, \overline{B}) + (A \,.\, \overline{B}) + (A \,.\, B)$$

Podemos notar que, independentemente do número de entradas, é possível obter as expressões booleanas que descrevem a tabela-verdade, e manipulando as expressões podemos simplificar qualquer circuito ao seu mínimo para a implementação.

Considerações finais

Neste capítulo, conhecemos as portas lógicas básicas, detalhamos o comportamento de cada uma delas – expressando-os na forma de equações – e começamos a entender como podemos combinar as portas lógicas para que elas tenham o comportamento desejado. Conferimos, também, que a matemática por trás dessas portas lógicas não depende de como elas estão sendo implementadas, e que essa mesma matemática pode ser utilizada com circuitos baseados em transistores, relês ou outras formas de implementação.

Referências

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

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.

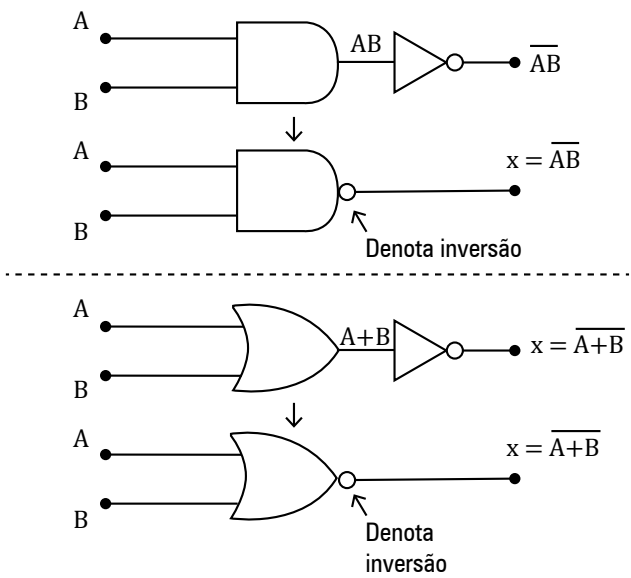
Além de trabalhar as portas lógicas mais complexas, vamos começar a entender como podemos fazer somas usando números binários, e por que isso é tão importante.

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

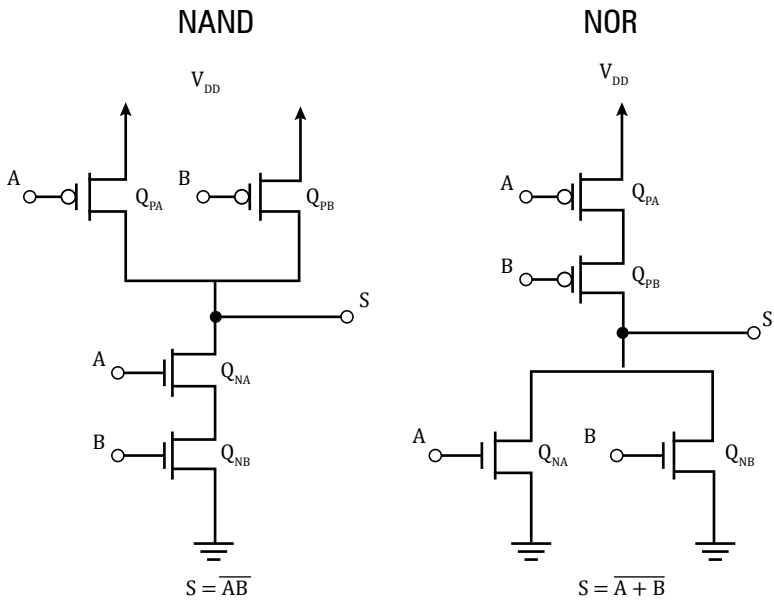
		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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.

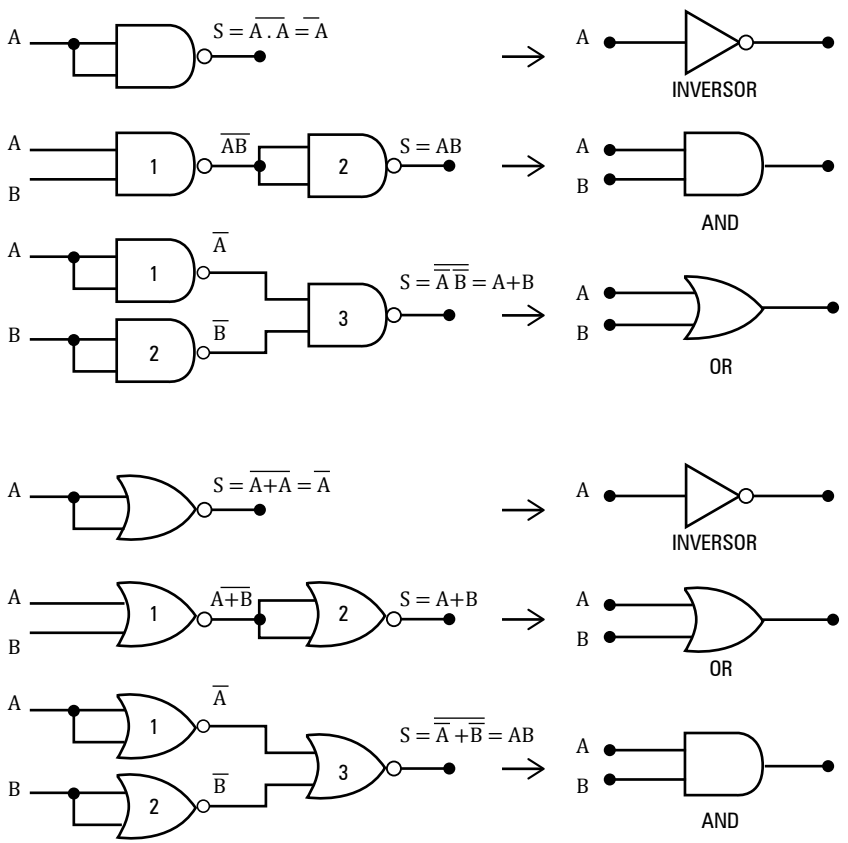
A segunda característica que torna essas portas lógicas muito interessantes é que elas são universais, ou seja, é possível implementar qualquer expressão booleana apenas combinando portas NAND ou apenas combinando portas NOR.

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

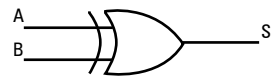


Podemos concluir que geramos todas as portas lógicas básicas a partir de portas NAND e NOR – e isso vale para a implementação de qualquer circuito combinacional, o que simplifica muito a sua implementação nas atuais técnicas de microeletrônica.

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.

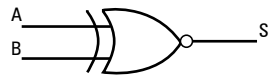
Figura 4 — Representação gráfica de uma porta XOR



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.

Capítulo 4

Circuitos combinacionais I

Neste capítulo, vamos construir um dos blocos de circuitos digitais mais importantes em termos de processamento de informações com as técnicas de projeto digital. Além disso, vamos começar a fazer operações matemáticas usando essas técnicas.

Antes de mais nada, é necessário relembrar que é muito comum um sistema digital processar informações recebidas. Quando falamos em processar informações, estamos nos referindo a tratar um dado coletado de um sensor, por exemplo, e executar uma ação em função do valor obtido na leitura desse dado.

Lembrando que os números são representados por bits e bytes. Podemos representar um valor de 0 a 255 usando um byte de 8 bits, ou seja, se em uma determinada aplicação precisarmos representar a leitura de um sensor usando um byte, teremos 256 combinações. Se o número que precisamos representar for maior que 255, podemos aumentar a quantidade de bits; por exemplo, se representarmos esse número com 10 bits, ele pode ser até 1023, e se utilizarmos 16 bits, o número pode ser até 65535.

Além disso, podemos representar mais de um valor usando dois conjuntos de bits, cada um representando uma determinada grandeza que está sendo medida. E, já que temos mais de um valor, podemos fazer operações matemáticas com esses valores representados usando números binários.

1 Meio somador

A primeira operação matemática que vamos abordar é a soma de binários, que deve ser feita em acordo com a matemática de Boole (IDOETA; CAPUANO, 1999). Existem regras específicas para fazer essas operações, e os detalhes de como elas são feitas serão explanados ao longo do capítulo, mas elas já devem ser de seu conhecimento.

Aqui, vamos transformar aquelas operações em circuitos eletrônicos, que podemos implementar com portas lógicas ligadas entre si adequadamente. Porém, antes de fazer o circuito, vamos conferir a menor soma que podemos realizar usando números binários. Ao somarmos números de apenas 1 bit, as combinações que podemos ter são:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

Podemos notar que, em um dos casos, temos o carry, ou seja, a soma de 1 bit pode ter como resultado 2 bits, e não apenas 1.

A partir disso, podemos montar uma tabela-verdade da soma de dois números de apenas 1 bit, sendo o primeiro número chamado de A e o segundo número, de B.

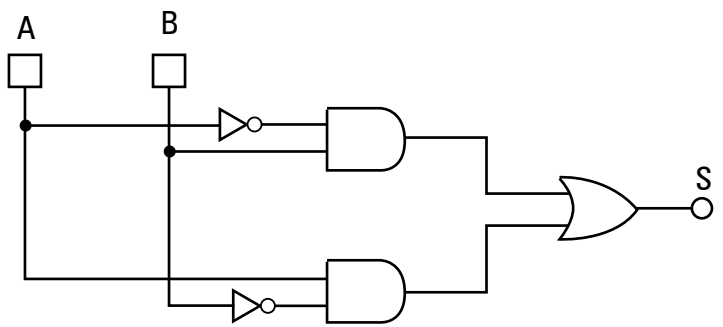
Tabela 1 — Somando dois números de 1 bit

A	B	S	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A partir da tabela-verdade apresentada, vamos criar uma equação booleana, começando pela equação que descreve a saída S:

$$S = (\overline{A} \,.\, B) + (A \,.\, \overline{B})$$

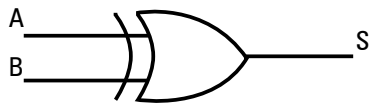
Figura 1 – Circuito gerado pela equação da saída S do meio somador



Fonte: adaptado de Idoeta e Capuano, 1999.

Porém, convém lembrar que essa combinação tem uma porta padrão já definida, que é uma porta XOR. Ou seja, S = A (+) B.

Figura 2 – S = A (+) B

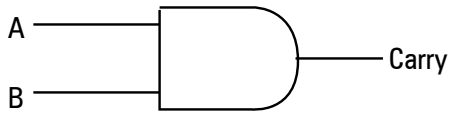


Fonte: adaptado de Idoeta e Capuano, 1999.

Agora, continuando a análise da tabela 1, podemos descrever a equação de carry como:

$$\text{Carry} = A \cdot B$$

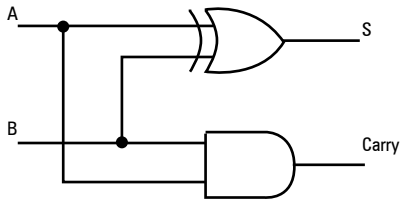
Figura 3 – Carry = A . B



Fonte: adaptado de Idoeta e Capuano, 1999.

Juntando no mesmo desenho as equações de S e de carry, temos a representação gráfica do meio somador, expressa pela figura 4.

Figura 4 — Meio somador



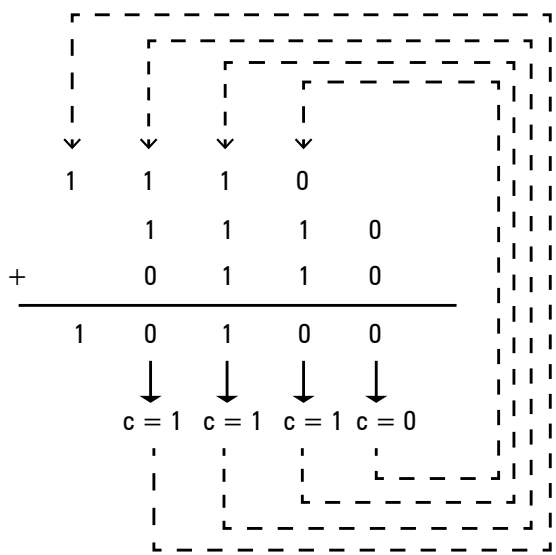
Fonte: adaptado de Idoeta e Capuano, 1999.

2 Somador completo

Conforme descrito, o meio somador possibilita fazer a soma de dois números binários de apenas um bit. Para fazermos a soma de números com múltiplos bits, precisamos de um circuito um pouco mais complexo.

Para compreender melhor o problema, vamos analisar uma soma de dois números de 4 bits, no caso, A = 1110 e B = 0110. Podemos representar essa soma de acordo com a figura 5, a seguir, em que C é o carry que descrevemos no meio somador.

Figura 5 – Somando dois números binários de 4 bits



Fonte: adaptado de Idoeta e Capuano, 1999.

Assim, para implementar um somador que considere o carry do bit anterior, temos a tabela-verdade a seguir.

Tabela 2 – Tabela-verdade do somador completo de 1 bit

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fonte: adaptado de Idoeta e Capuano, 1999.

Na tabela 2, descrevemos a operação de soma de apenas um dos bits somados na figura 5. Ou seja, essa tabela-verdade descreve cada uma das colunas (ou bits) da soma da figura 5. Nessa tabela, Ci é o carry de entrada, que é obtido do bit anterior – ou menos significativo em relação ao bit cuja a saída S estamos gerando –, e Co é o carry de saída, que será o carry de entrada do próximo bit cuja saída vamos gerar.

Explicado isso, vamos gerar as equações que descrevem S e Co:

$$S = (\overline{A} \,.\, \overline{B} \,.\, Ci) + (\overline{A} \,.\, B \,.\, \overline{Ci}) + (A \,.\, \overline{B} \,.\, \overline{Ci}) + (A \,.\, B \,.\, Ci)$$

$$Co = (\overline{A} \,.\, B \,.\, Ci) + (A \,.\, \overline{B} \,.\, Ci\,) + (A \,.\, B \,.\, \overline{Ci}\,) + (A \,.\, B \,.\, Ci)$$

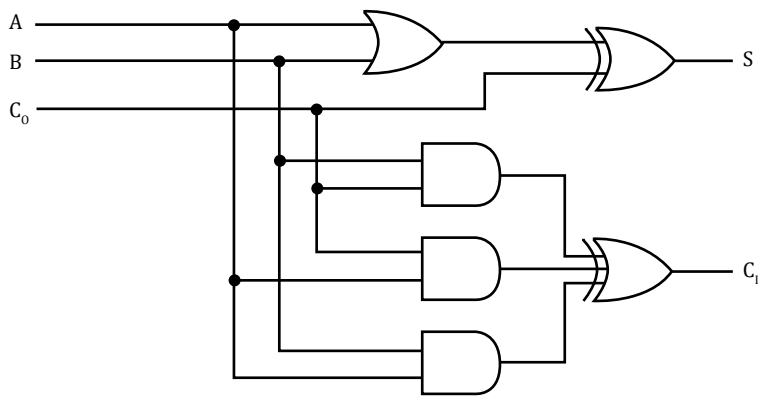
Simplificando essas equações, obtemos:

$$S = A\ (+)\ B\ (+)\ Ci$$

$$Co = (A \,.\, Ci) + (B \,.\, Ci) + (A \,.\, B)$$

Implementando as equações, obtemos o circuito a seguir (figura 6), que corresponde à implementação de qualquer uma das colunas do somador da figura 5 (p. 61).

Figura 6 – Somador completo



Fonte: adaptado de Idoeta e Capuano, 1999.

O somador completo que desenhamos até agora gera o resultado de apenas um bit, mas podemos usá-lo para somar quantos bits forem necessários.

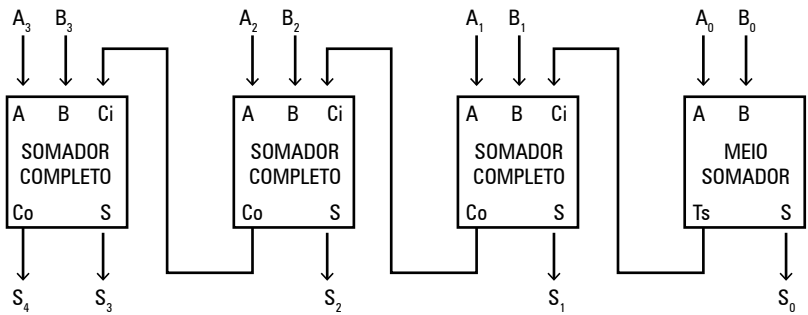
Para ilustrar isso, vamos implementar um somador de dois números de 4 bits, que vai gerar uma saída de 5 bits, sendo o primeiro número, A, desmembrado nos bits A_3, A_2, A_1 e A_0, e o segundo número, B, desmembrado em B_3, B_2, B_1 e B_0. A saída S, por sua vez, é desmembrada em S_4, S_3, S_2, S_1 e S_0. Esquematizando a soma, vamos ter o expresso pela figura 7, a seguir.

Figura 7 – Montando a soma de binários

	A_3	A_2	A_1	A_0
+	B_3	B_2	B_1	B_0
S_4	S_3	S_2	S_1	S_0

Usando o somador que descrevemos anteriormente, podemos implementar a soma descrita ligando a saída Co de um bit à entrada Ci do bit imediatamente mais significativo, à exceção dos bits A_0, B_0 e S_0, que são os menos significativos. Nesse caso, podemos usar o meio somador para implementar essa etapa do circuito, como mostrado na figura 8.

Figura 8 – Implementando um somador de 4 bits



Fonte: adaptado de Idoeta e Capuano, 1999.

Podemos generalizar esse circuito para somar números de quantos bits forem necessários para a nossa aplicação. Ou seja, esse circuito funciona como um bloco para a construção de somadores binários.

3 Outra forma de construir um somador completo

Outra forma de descrever um somador completo é utilizando dois meios somadores.

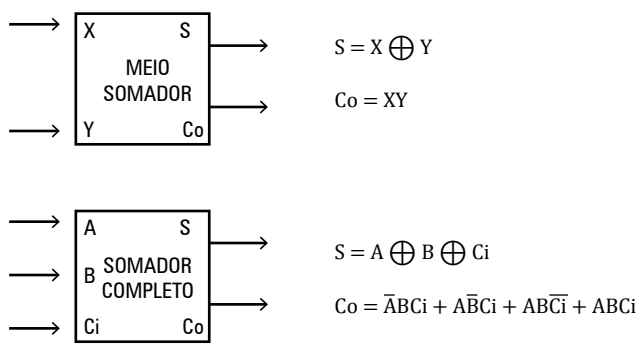
IMPORTANTE

A vantagem da técnica de dois meios somadores é que ela diminui o bloco que deve ser repetido diversas vezes no circuito em construção, bem como diminui a diversidade de blocos de que precisamos no nosso projeto.

Quando usamos a técnica de projeto digital, percebemos que quanto mais conseguimos dividir os problemas que estamos resolvendo, mais fácil o projeto como um todo se torna. Outro ponto importante é que, quando temos circuitos que se repetem muito, podemos gastar mais energia na sua otimização, melhorando a relação entre o esforço de projeto e o benefício que se obtém em termos de desempenho e redução de custos.

Para fazer a conversão de um somador completo na junção de dois meios somadores, vamos começar analisando as expressões que definem o comportamento de ambos os blocos, conforme apresentado na figura 9.

Figura 9 – Comparação entre as expressões lógicas do meio somador e do somador completo



Fonte: adaptado de Idoeta e Capuano, 1999.

Se reescrevermos a expressão de Co do somador completo, teremos:

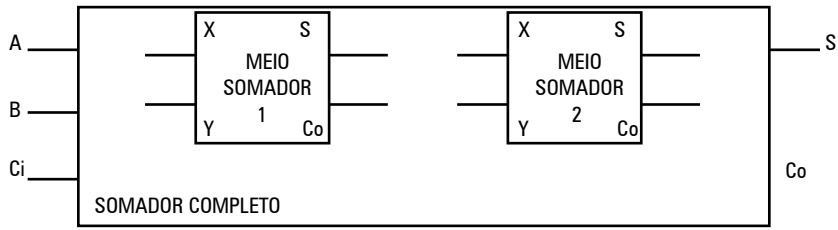
$$Co = Ci \,.\, (\overline{A} \,.\, B + A \,.\, \overline{B}) + A \,.\, B \,.\, (\overline{Ci} + Ci)$$

Podemos simplificá-la da seguinte forma:

$$Co = Ci \,.\, (\, A \,(+)\, B) + A \,.\, B \rightarrow \text{Expressão carry out}$$

Lembremos que a ideia é utilizar dois meio somadores para criar um somador completo. No início, vamos ter dois meio somadores: o meio somador 1 e o meio somador 2, como mostrado na figura 10, a seguir.

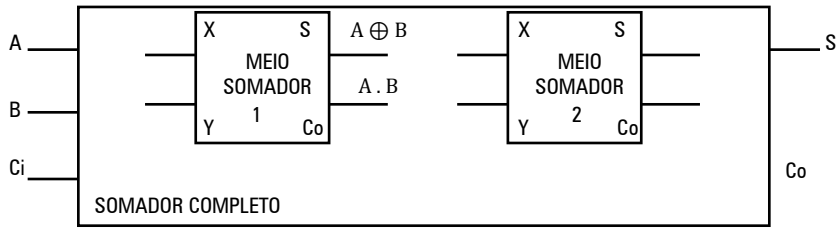
Figura 10 – Somador completo usando dois meios somadores



Fonte: adaptado de Idoeta e Capuano, 1999.

Se ligarmos, na entrada do meio somador 1, as entradas A e B a X e Y, respectivamente, teremos a situação expressa pela figura 11, a seguir.

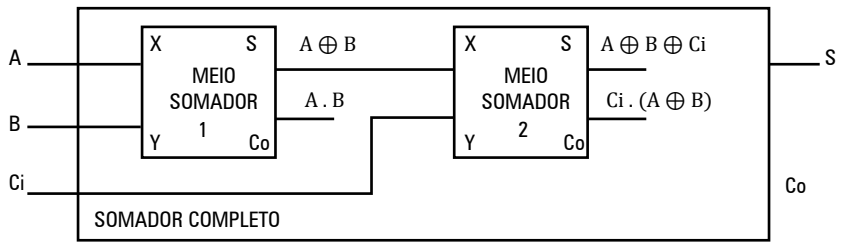
Figura 11 – Ligando as entradas A e B do somador completo no meio somador 1



Fonte: adaptado de Idoeta e Capuano, 1999.

Se ligarmos a saída S do meio somador 1 à entrada X do meio somador 2, e a entrada Ci do somador completo à entrada Y do meio somador 2, teremos, na saída S do meio somador 2, a exata expressão que descreve a saída do somador completo. Ou seja, já teremos metade do nosso somador completo definido.

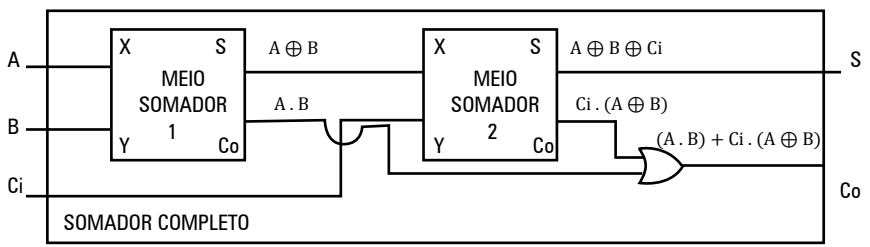
Figura 12 – Ligando o meio somador 2



Fonte: adaptado de Idoeta e Capuano, 1999.

Neste ponto, falta apenas definir a saída Co do somador completo; porém, ao observarmos a equação carry out definida anteriormente, podemos concluir que os dois termos da equação já estão presentes no circuito, nas saídas Co do meio somador 1 e na saída Co do meio somador 2. Para obtermos a equação carry out do somador completo, basta realizarmos a operação OR entre as saídas Co dos dois meio somadores, e teremos o nosso somador completo construído a partir de dois meios somadores, como podemos conferir na figura 13.

Figura 13 – Somador completo totalmente montado a partir de dois meios somadores



Fonte: adaptado de Idoeta e Capuano, 1999.

Considerações finais

Neste capítulo, pudemos usar a técnica digital para processar informações – no caso, fazer a soma de dois números binários. Essa técnica certamente tem muitas aplicações, seja na forma de um circuito discreto em uma aplicação muito específica, seja dentro de um processador, que precisa ter um circuito eletrônico para implementar a operação de soma de dois números binários.

Outro ponto importante trabalhado neste capítulo foi a exploração de várias formas de implementar a mesma função lógica, e como isso pode ser utilizado para que se tenha um melhor aproveitamento dos recursos físicos de projeto.

É importante compreender que essa é apenas uma das possibilidades de implementar uma soma binária e que existem diversas formas de obter o mesmo resultado, com diferenças entre área e velocidade de cálculo. Essas outras possibilidades não foram exploradas aqui, mas, caso haja interesse, é possível buscá-las em artigos científicos na área de projeto de circuitos digitais.

Referências

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

Capítulo 5

Circuitos combinacionais II

Dando continuidade à construção dos blocos básicos de processamento de informações com as técnicas de projeto digital, vamos trabalhar neste capítulo com a subtração de números binários.

Um ponto importante que será abordado é a introdução da representação de números negativos binários. Esse tópico é fundamental para que se compreenda, em contextos mais avançados e em linguagens de programação como C ou C++, por que temos de declarar se um inteiro é representado com ou sem sinal.

A compreensão desse tópico também é fundamental para que, em uma situação de projeto, se tenha conhecimento dos elementos

necessários à decisão de como representar os números, e ainda para que se entendam as consequências disso.

1 Representando números negativos

Conforme já abordado anteriormente, os números em um sistema digital são comumente representados por bits e bytes; quer dizer, podemos representar um valor de 0 a 255 usando um byte de 8 bits.

Entretanto, quando estamos realizando uma operação de subtração, é comum, e perfeitamente natural, que o resultado da operação seja um número negativo; por conta disso, é necessário estabelecer uma padronização para representar um número negativo.

Um ponto importante, neste caso, é pensar um pouco sobre qual a melhor forma de "avisar" aos circuitos eletrônicos, que farão as operações matemáticas, que um número negativo está sendo representado. Para isso, vamos adotar aqui a padronização descrita na técnica de complemento de 2 (IDOETA; CAPUANO, 1999), cujos fundamentos são descritos a seguir.

Uma das formas mais comuns de representar um número negativo é ter um bit para identificar se o número é positivo ou negativo. Vamos adotar que esse bit é o mais significativo. Assim, em um circuito em que representamos um número com 8 bits, o bit 7 será o nosso bit de sinal – lembrando que os bits de uma representação binária começam em 0, ou seja, um número de 8 bits vai ter seu bit menos significativo com índice 0 e seu bit mais significativo, com índice 7.

Outro ponto importante é considerar esta questão: qual valor desse bit indica que estamos representando números positivos e qual valor indica que estamos representando números negativos? Vamos adotar que, quando o bit de sinal for 0, estamos representando um número positivo, e quando o bit de sinal for 1, estamos representando um número negativo.

A primeira consequência de adotar um bit de sinal é que a representação será dividida em números positivos e negativos. Se considerarmos que zero é um número positivo, metade dos valores que podemos representar serão negativos, e a outra metade, positivos.

Para exemplificar isso, vamos usar um byte de 8 bits. Se esse byte representar apenas números positivos, será possível representar valores de 0 a 255. Entretanto, se esse byte também representar números negativos, serão representados valores de -128 até +127.

A forma mais prática e direta de inverter o sinal de um número binário, dentro da técnica de complemento de 2, é composta por duas operações booleanas. A primeira delas é a simples negação de todos os bits, ou seja, quando um número é positivo, um determinado bit é zero, ele se torna um e vice-versa. Isso também vale para o bit de sinal, ou seja, o bit mais significativo do número que estamos representando.

É importante notar que essa inversão deve ser feita quando já se sabe o número de bits com o qual o circuito como um todo trabalha. Ou seja, se um circuito é feito para trabalhar com 4 bits que representem uma grandeza, isso também vale para os números negativos. Sem isso, o número de bits seria infinito, o que não seria viável para um circuito implementado fisicamente.

A segunda operação booleana necessária para fazer troca de sinal, de positivo para negativo, é somar 1 ao resultado da primeira operação, usando as mesmas regras de soma do capítulo anterior.

PARA PENSAR

Você já parou para pensar em como representar números muito pequenos (como 0,00000001) ou muito grandes (como 100.000.000.000) usando os mesmos circuitos para representar e fazer as operações matemáticas? Para se aprofundar no assunto, faça uma pesquisa sobre como são representados os números em ponto flutuante.

Para tornar mais palpável o processo de inverter um número binário usando a técnica do complemento de 2, confira o exemplo da tabela 1. Nela, invertemos o número binário 0110 (6 em decimal).

Tabela 1 – Operação da operação de inversão de sinal

	Bit de sinal			
	Bit 3	**Bit 2**	**Bit 1**	**Bit 0**
Positivo	0	1	1	0
Complemento de 1	1	0	0	1
Soma do complemento de 2				1
Complemento de 2	1	0	1	0

Se o mesmo valor 6 fosse representando em 6 bits, a operação seria representada da forma expressa pela tabela 2.

Tabela 2 – Operação de inversão de sinal com um número representado em 6 bits

	Bit de sinal					
	Bit 5	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Positivo	0	0	0	1	1	0
Complemento de 1	1	1	1	0	0	1
Soma do complemento de 2						1
Complemento de 2	1	1	1	0	1	0

Observe que quem determina o número de bits de um número negativo é a pessoa que está fazendo o projeto do circuito eletrônico que vai manipular essas grandezas.

Note, também, que o maior número positivo que poderá ser representado tem um bit a menos do que o total de bits com que o circuito

trabalha, pois, se ultrapassarmos esse valor, vamos representar um valor negativo, e não um valor positivo.

Assim, no exemplo em que estamos representando o número 6 em 4 bits, o maior valor positivo que podemos representar em 4 bits seria o valor 7, que em binário corresponde a 0111. A mesma regra vale para qualquer quantidade de bits.

2 Meio subtrator

Dando continuidade às operações matemáticas utilizando as técnicas de projeto digital, vamos trabalhar a subtração de números binários, começando pela subtração mais simples possível, de apenas um bit.

Para entendermos bem, vamos começar com uma subtração na representação decimal. Ao subtrairmos B de A e guardarmos o resultado em S, podemos ter os resultados expressos pela tabela 3.

Tabela 3 – Subtração em decimal de 1 bit

A	B	S
0	0	0
0	1	-1
1	0	1
1	1	0

Porém, em binário, a forma de representar o negativo — ou o "empresta 1", ou ainda o borrow — deve ser um segundo bit. Como estamos usando a letra B para representar um dos números da operação, vamos representar o borrow com o T de transporte, em que Te é a entrada do transporte e Ts, a saída do transporte do circuito que estamos modelando. Assim, a representação da subtração da tabela 3 é apresentada em binário na tabela 4, a seguir.

Tabela 4 – Subtração de um bit com o "empresta 1" representado em T

A	B	S	Ts
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

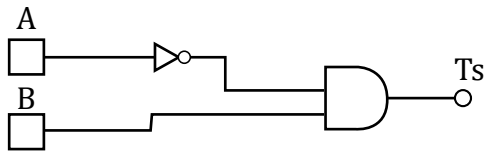
A tabela 4 é a tabela-verdade do meio subtrator. Se a observarmos, teremos a seguinte equação para descrever o comportamento que queremos do circuito:

$$S = (\overline{A} \,.\, B) + (A \,.\, \overline{B})$$

Essa equação corresponde exatamente a uma porta XOR.

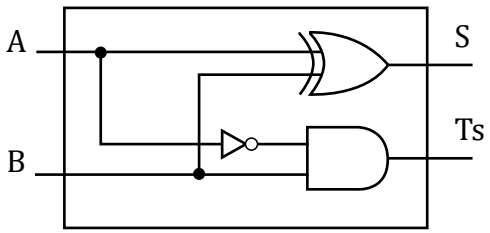
Continuando a análise da tabela 4, podemos descrever a equação de Ts como Ts = $\overline{A}$. B, a qual pode ser transformada no circuito a seguir, representado pela figura 1.

Figura 1 – Ts = $\overline{A}$. B



Ao unir os dois circuitos em um mesmo bloco, obtemos o circuito do meio subtrator, representado na figura 2.

Figura 2 — Meio subtrator



Fonte: adaptado de Idoeta e Capuano, 1999.

3 Subtrator completo

O meio subtrator, como o próprio nome já indica, não é capaz de fazer uma subtração completa, sendo necessário um circuito um pouco mais complexo para realizar essa tarefa.

Para evidenciar essa situação, vamos analisar uma subtração de dois números de 5 bits, no caso, A = 01100 e B = 00011. Podemos representar a subtração de acordo com a tabela 5, a seguir, em que Ts é o "empresta 1" que descrevemos no meio subtrator.

Tabela 5 – Subtraindo dois números binários de 5 bits com sinal

	Bit de sinal				
	Bit 4	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
A	0	1	1	0	0
B	0	0	0	1	1
"Empresta 1"			1	1	
S	0	1	0	0	1
	Ts = 0	Ts = 0	Ts = 0	Ts = 1	Ts = 1

Dessa forma, para implementarmos um subtrator que avalie o "empresta 1" do bit anterior e para fazermos a subtração de uma das colunas da subtração da tabela 5, temos a seguinte tabela-verdade.

Tabela 6 – Tabela-verdade do subtrator completo de 1 bit

A	B	Te	S	Ts
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Na tabela 6, Te é o "empresta 1" de entrada, obtido do bit anterior, e que é o menos significativo em relação ao bit cuja saída S estamos gerando, e Ts é o "empresta 1" de saída, que será o "empresta 1" de entrada do próximo bit cuja saída vamos gerar.

Explicado isso, vamos gerar as equações que descrevem S e Co:

$$S = (\overline{A} \,.\, \overline{B} \,.\, Te) + (\overline{A} \,.\, B \,.\, \overline{Te}) + (A \,.\, \overline{B} \,.\, \overline{Te}) + (A \,.\, B \,.\, Te)$$

$$Ts = (\overline{A} \,.\, \overline{B} \,.\, Te) + (\overline{A} \,.\, B \,.\, \overline{Te}) + (\overline{A} \,.\, B \,.\, Te) + (A \,.\, B \,.\, Te)$$

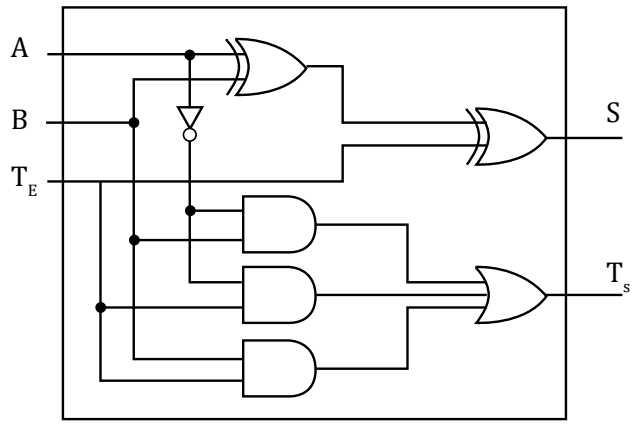
Ao simplificarmos essas equações, obtemos:

$$S = A\ (+)\ B\ (+)\ Te$$

$$Ts = (\overline{A} \,.\, B) + (\overline{A} \,.\, Te) + (B \,.\, Te)$$

Implementando essas equações, obtemos o circuito a seguir, que corresponde ao circuito que pode implementar qualquer uma das colunas da subtração binária da figura 3.

Figura 3 – Subtrator completo



Fonte: adaptado de Idoeta e Capuano, 1999.

O subtrator completo gera o resultado de apenas um bit, entretanto, a combinação de vários subtratores completos nos permite construir subtratores no tamanho necessário à aplicação.

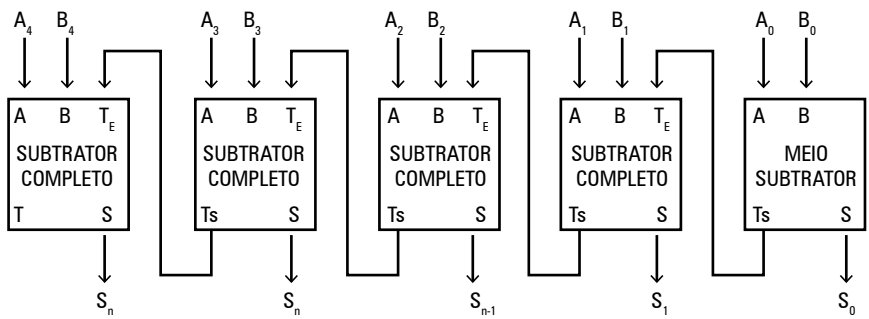
Para ilustrar isso, vamos implementar um somador de dois números de 5 bits, que vai gerar uma saída de 5 bits, sendo o primeiro número, A, desmembrado nos bits A_3, A_2, A_1 e A_0, e o segundo número, B, desmembrado em B_3, B_2, B_1 e B_0. A saída S, por sua vez, é desmembrada em S_4, S_3, S_2, S_1 e S_0. Esquematizando a soma, teremos o quadro expresso pela figura 4.

Figura 4 – Montando a subtração de binários

	A_4	A_3	A_2	A_1	A_0
-	B_4	B_3	B_2	B_1	B_0
	S_4	S_3	S_2	S_1	S_0

Usando o subtrator completo, podemos implementar a subtração da figura 4 ligando a saída Ts de um bit na entrada Te do bit imediatamente mais significativo, à exceção dos bits A_0, B_0 e S_0, que são os menos significativos. Nesse caso, podemos usar o meio somador, como expresso na figura 5.

Figura 5 — Implementando um subtrator de 5 bits



Fonte: adaptado de Idoeta e Capuano, 1999.

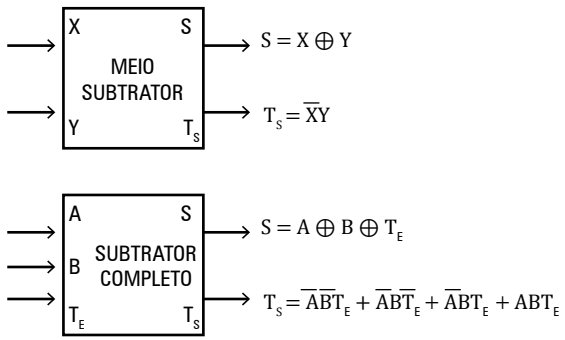
Além disso, os números negativos são representados usando a notação de complemento de 2, que discutimos no início deste capítulo.

4 Subtrator completo a partir do meio subtrator

Em muitas situações de projeto, temos uma boa vantagem ao quebrarmos o projeto de blocos muitas vezes repetidos no circuito que estamos construindo, principalmente em termos de otimização do projeto físico. Essa situação se aplica aos subtratores, visto que são blocos que se repetem pelo número de bits manipuláveis do circuito.

Ao analisarmos as expressões lógicas que descrevem o meio subtrator e o subtrator completo, percebemos que elas podem ser reescritas de uma forma diferente.

Figura 6 – Comparação entre as expressões lógicas do meio subtrator e do subtrator completo



Fonte: adaptado de Idoeta e Capuano, 1999.

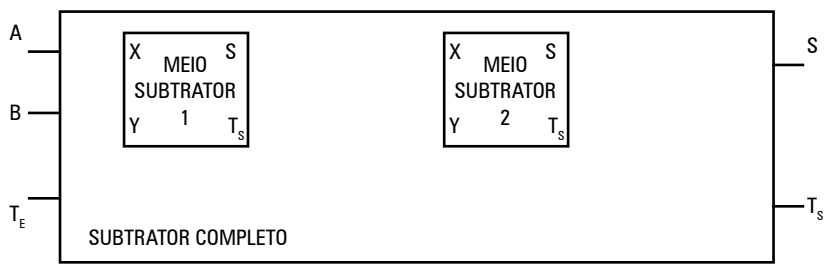
$$Ts = Te \,.\, (\overline{A} \,.\, \overline{B} + A \,.\, B) + \overline{A} \,.\, B \,.\, (\overline{Te} + Te)$$

Podemos simplificar essa equação da seguinte forma:

$$Ts = Te \,.\, (\overline{A\,(+)\,B}) + \overline{A} \,.\, B \rightarrow \text{Expressão “empresta 1”}$$

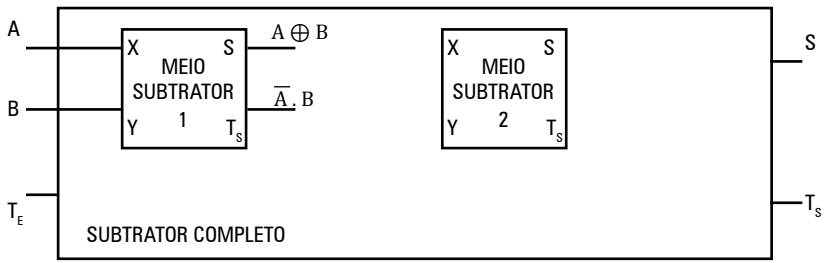
Vamos montar nosso subtrator completo usando dois subtratores, e faremos a ligação deles passo a passo.

Figura 7 – Subtrator completo usando dois meio subtratores



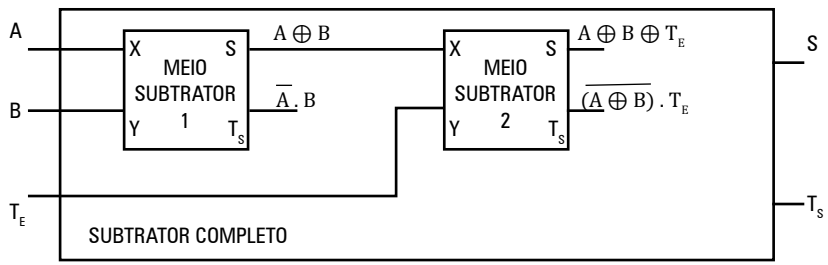
Ao ligarmos as entradas A e B do somador completo a X e Y do meio subtrator 1, respectivamente, teremos a situação expressa pela figura 8.

Figura 8 – Ligando as entradas A e B do subtrator completo no meio subtrator 1



Ao ligaremos a saída S do meio subtrator 1 à entrada X do meio subtrator 2, e a entrada Te do subtrator completo à entrada Y do meio subtrator 2, obteremos, na saída S do meio subtrator 2, a expressão que descreve a saída do subtrator completo.

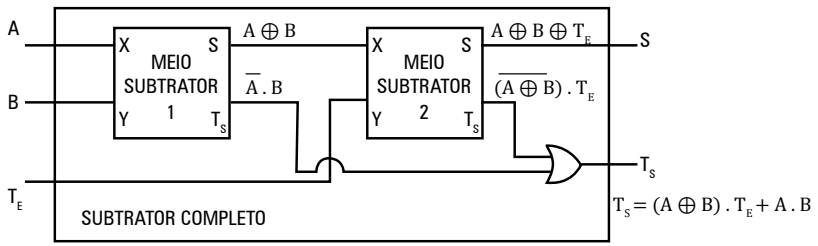
Figura 9 – Ligando o meio subtrator 2



Analisando a equação "empresta 1", concluímos que os dois termos da equação já estão presentes no circuito, na saída Ts do meio subtrator 1 e na saída Ts do meio subtrator 2.

Assim, a equação "empresta 1" do subtrator completo é obtida por meio de uma operação OR entre as saídas Ts dos dois meios subtratores, como podemos conferir na figura 10, a seguir.

Figura 10 – Subtrator completo totalmente montado a partir de dois meios subtratores



Considerações finais

Neste capítulo, pudemos conferir como são representados os binários negativos e como eles refletem no projeto dos circuitos digitais destinados a fazer as operações de subtração, as quais podem ter como resultado números negativos.

Além disso, tivemos um segundo exemplo de como implementar um circuito digital escalável, no qual se pode aumentar a quantidade de bits que ele é capaz de manipular apenas aumentando-se o número de blocos que se repetem no circuito. Junto com o circuito somador, esse é um dos circuitos combinacionais mais úteis na manipulação de dados.

Referências

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

Capítulo 6

Unidade lógica e aritmética (ULA)

Neste capítulo, vamos tratar de uma estrutura digital combinacional presente em todas as unidades centrais de processamento, mais comumente chamadas de processador ou CPU. Vamos abordar mais detidamente a unidade lógica e aritmética, a ULA — ou ALU, do inglês arithmetic logic unit (VAHID, 2008).

Esse elemento pode ser utilizado de diversas formas, porém, o mais comum é que seja parte de um processador — e ele costuma ser o elemento central de qualquer processador, pois, em muitas arquiteturas, é o que determina a velocidade com que as informações são processadas.

A ULA é a parte do processador que efetivamente faz o processamento da informação, ou seja, é na ULA que são feitas as operações matemáticas de que precisamos em um produto desenvolvido por meio das técnicas de projeto digital.

Uma ULA tem a complexidade necessária para que o produto que está sendo desenvolvido possa fazer o processamento adequado aos objetivos traçados. Assim, se em um determinado produto for necessária a operação de soma, essa operação deverá estar implementada como uma das funções da ULA. Entretanto, nem sempre é preciso implementar todas as operações matemáticas que serão necessárias, cabendo ao desenvolvedor fazer uma avaliação entre custo e desempenho.

PARA PENSAR

A velocidade de uma CPU é dada pelo tempo de propagação de seus circuitos combinacionais. Em um processador, é comum que o circuito combinacional com maior tempo de propagação seja a ULA. Pense: como o número de bits interfere no tempo de propagação da ULA e, por consequência, na frequência de operação da CPU?

Para exemplificar essa abordagem, vamos trabalhar a hipótese de que nosso produto precisa executar uma multiplicação como uma das etapas para tomar uma decisão no algoritmo. Essa operação matemática de multiplicação poderá ser implementada por meio de um circuito especialista dentro da ULA, ou ser implementada pelo deslocamento de bits e soma condicional. Mas como o desenvolvedor vai tomar essa decisão?

Ele terá diante de si duas opções. A primeira é usar mais elementos lógicos (seja na forma de circuitos integrados ou na forma de área de silício dentro de um circuito mais complexo) e ter um tempo de propagação dentro da ULA mais elevado, para todas as operações da ULA, e implementar o multiplicador. A segunda opção é usar mais ciclos de trabalho do processador para implementar a mesma operação, porém, com um tempo de propagação menor para todas as operações que a ULA vai realizar para executar o algoritmo do produto em questão.

Uma vez que estiver claro para o desenvolvedor o impacto de cada abordagem, ele verificará os requisitos de desempenho do produto – para que este atinja a qualidade que os usuários esperam dele – e tomará sua decisão. Assim, o projeto de uma ULA ou a escolha de qual modelo de ULA será utilizado tem efeito direto no projeto de todo o resto do produto.

1 Construção de uma ULA básica

Uma vez que já temos noção do que é uma ULA, vamos construir uma ULA básica — que, apesar de básica, já atenderá a diversos requisitos para uso em um sistema digital bastante complexo. Para isso, talvez o mais importante seja identificar quais são os requisitos de uma ULA básica, ou seja, o que uma ULA deve apresentar para que seja considerada uma ULA.

Como já comentamos na introdução, o papel de uma ULA é implementar todas as operações lógicas e aritméticas nas informações que serão tratadas pelo produto em desenvolvimento, e é por aí que começamos.

Para implementar a nossa ULA básica, selecionamos algumas das operações mais comuns em qualquer projeto digital:

- soma;
- subtração;
- operação lógica AND;
- operação lógica OR;
- operação lógica NOT; e
- operação lógica XOR.

Para implementar todas as funções citadas, vamos usar alguns blocos básicos, mais simples, de lógica combinacional. Em seguida, vamos juntar esses blocos para formar a nossa ULA básica, a qual

pode ser implementada com 2, 4 ou 8 bits, ou ainda com mais bits de dados.

A nossa ULA terá como entradas de dados as entradas A e B, e como saída de dados a saída S. A função lógica a ser executada pela ULA será definida pelo seletor F. Na forma como vamos implementar a nossa ULA, ela pode ser feita com um barramento de 2 bits ou com um barramento de mais bits de dados — lembrando que o número de bits de dados determina o quão rápido os dados poderão ser manipulados pela ULA em questão.

2 Componentes de uma ULA básica

Uma das formas mais comuns de resolver um problema complexo é dividi-lo em problemas menores, os quais, por sua vez, podem ser novamente divididos, até que sejam simples e fáceis de resolver. Dessa forma, vamos começar a dividir nosso problema pelo número de bits que temos em nossa ULA. Ou seja, em vez de tentar descrever a nossa ULA com todas as entradas e saídas de uma única vez, vamos trabalhar um bit de cada vez; como todos os bits são iguais, basta ligarmos os bits de forma adequada e teremos nossa ULA básica formada.

É importante salientar que não vamos entrar em detalhes sobre como cada bloco é construído internamente, como cada porta lógica é otimizada etc. A ideia, neste capítulo, é construir uma visão sistêmica de como se constrói um circuito mais complexo a partir de elementos mais simples.

3 Operações lógicas

Dentre as operações que selecionamos para implementar em nossa ULA básica, podemos destacar:

- operação lógica AND;

- operação lógica OR;
- operação lógica NOT; e
- operação lógica XOR.

Com exceção da operação NOT, as demais são a saída binária entre cada bit das entradas A e B. Ou seja, no circuito que vamos criar para fazer cada bit de nossa ULA, vamos ter uma porta AND, uma porta OR e uma porta XOR – cada uma com duas entradas, uma ligada na entrada A e outra na entrada B –, além de uma porta NOT, que vamos ligar apenas na entrada A. Vamos ligar as saídas posteriormente.

4 Operações aritméticas

Além das operações lógicas, há duas operações aritméticas que vamos implementar na nossa ULA: soma e subtração. Para executá-las, vamos utilizar os circuitos somador completo e subtrator completo. Ambos os blocos possuem as entradas A, B e Te, além das saídas S e Ts.

No caso do somador, Te e Ts correspondem ao sinal de carry, que é o famoso "vai 1", sendo Te o "vai 1" de entrada e Ts, o "vai 1" de saída, para o próximo bit do somador.

No caso do subtratator, Te e Ts correspondem ao sinal de borrow, que é o "empresta 1" do circuito de subtração, sendo que o Te é o "empresta 1" vindo do bit menos significativo e que o Ts vai ligado ao Te do próximo bit mais significativo.

Não vamos, aqui, entrar em detalhes sobre a construção desse bloco, já que os capítulos 4 e 5 são dedicados a eles.

5 Multiplexador (MUX)

Agora que já descrevemos os circuitos que efetivamente realizam as operações, precisamos de uma forma de selecionar, na saída de cada bit, qual das operações será refletida na saída.

O circuito que implementa essa função lógica é chamado de multiplexador, ou MUX. Resumidamente, sua função é, dado um código de seleção, refletir na saída uma das entradas.

Para implementar o nosso MUX, precisamos selecionar uma de seis opções de operações disponíveis. Isso vai nos exigir ao menos 3 bits de seleção. Porém, com 3 bits de seleção, temos disponíveis oito possibilidades.

Como forma de aproveitar melhor os recursos disponíveis, podemos acrescentar mais duas operações lógicas:

- reset (em que zeramos todos os bits da saída); e
- cópia A (em que copiamos a entrada A na saída).

Como forma de simplificar a representação da nossa tabela-verdade, vamos fazer uma ligeira modificação na representação tradicional de tabelas-verdade.

Tabela 1 – Tabela-verdade do MUX

F2	F1	F0	S
0	0	0	E0
0	0	1	E1
0	1	0	E2
0	1	1	E3
1	0	0	E4
1	0	1	E5
1	1	0	E6
1	1	1	E7

Nessa tabela, podemos notar que, de acordo com o código colocado nas entradas de seleção F0, F1 e F2, estamos selecionando qual das entradas de dados entre E0 e E7 será refletida na saída. Assim, podemos usar o MUX para selecionar qual das operações lógicas vamos ter na saída da nossa ULA simplificada.

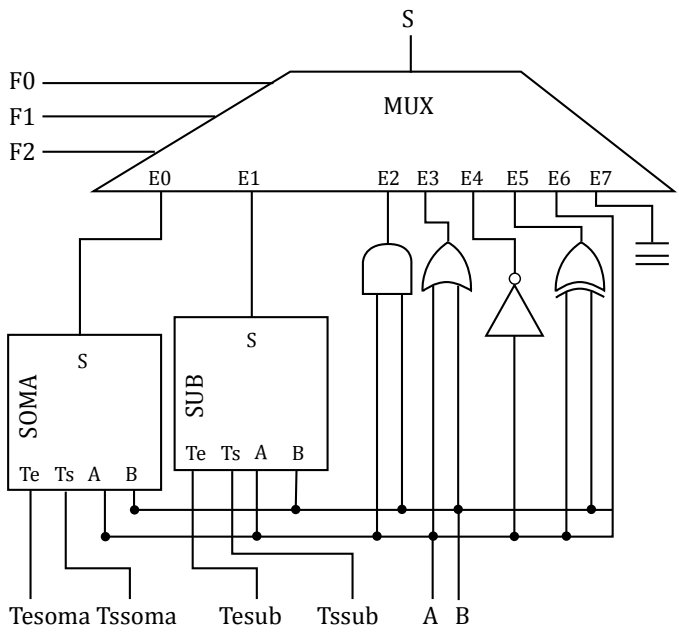
Para que fique mais claro, vamos trocar as entradas do MUX da tabela 1 pelas operações que pretendemos realizar com essas seleções, conforme expresso pela tabela 2.

Tabela 2 – Tabela de funções da nossa ULA

F2	F1	F0	S
0	0	0	Soma
0	0	1	Subtração
0	1	0	AND
0	1	1	OR
1	0	0	NOT
1	0	1	XOR
1	1	0	A
1	1	1	RESET

Dessa forma, podemos juntar todos os circuitos que já foram discutidos neste capítulo, e teremos uma representação da nossa ULA básica na forma de circuito digital, conforme expresso pela figura 1.

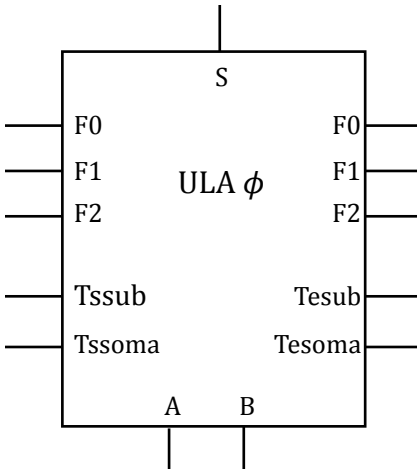
Figura 1 — Diagrama em blocos de um bit da ULA básica



No circuito da figura 1, vamos ter quatro entradas — Tesoma, Tesub, A e B — e três saídas — Tssoma, Tssub e S —, além das entradas F0, F1 e F2, que selecionam qual função da ULA está ativa. Ligamos essas entradas e saídas internamente no nosso bit da ULA, conforme descrito na figura 1.

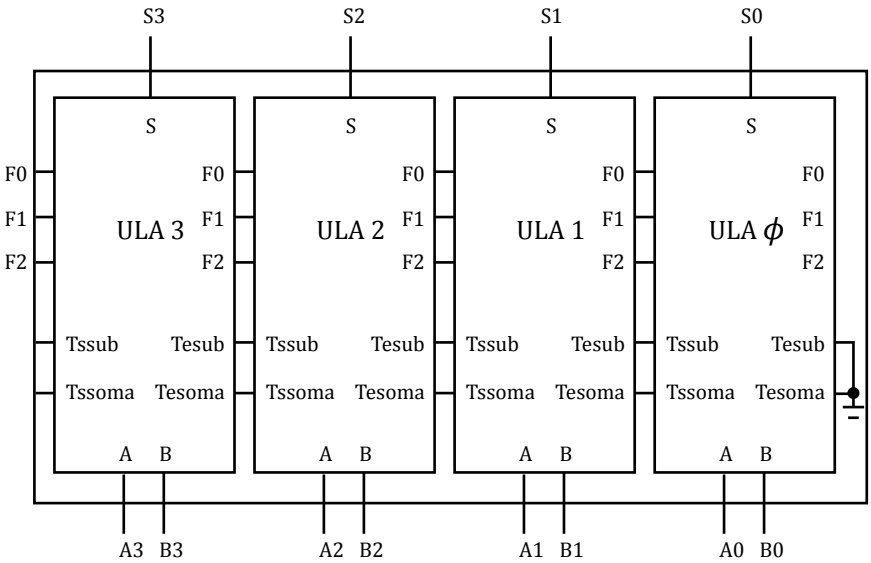
Podemos simplificar a representação de cada bit da nossa ULA com o símbolo da figura 2, a seguir.

Figura 2 — Símbolo de 1 bit da nossa ULA básica



Se juntarmos dois circuitos de 1 bit da nossa ULA, teremos uma ULA de 2 bits. Por outro lado, se juntarmos quatro circuitos, como na figura 3, a seguir, teremos uma ULA de 4 bits.

Figura 3 — ULA de 4 bits



Assim, implementamos uma ULA que pode ser ampliada, a fim de executar as operações descritas neste capítulo, para quantos bits forem necessários.

Considerações finais

A descrição que fizemos da nossa ULA básica certamente não é a forma mais eficiente de implementar esse circuito; porém, ilustra bem o que é uma ULA e como podemos implementar esse elemento, que é de fundamental importância para a implantação de circuitos digitais mais complexos. Além disso, a ULA é um dos componentes básicos dos quais precisamos para implementar computadores e quaisquer outros produtos que sejam desenvolvidos utilizando as técnicas de projeto digital, e que tenham de manipular informações de forma bastante compacta.

Referências

VAHID, Frank. **Sistemas digitais**: projeto, otimização e HDLs. Tradução: Anatólio Laschuk. Porto Alegre: Artmed, 2008.

Capítulo 7

Flip-flops

Neste capítulo, vamos conhecer e analisar os elementos fundamentais à construção de produtos desenvolvidos usando a técnica digital, em particular. São os elementos fundamentais do que chamamos de circuitos sequenciais.

Os circuitos sequenciais têm por característica o fato de que a saída não depende exclusivamente do estado vigente das entradas, diferentemente do que ocorre com os circuitos combinacionais. Eles são chamados de sequenciais porque o valor da saída depende de uma sequência de acionamentos das entradas do circuito; ou seja, não basta olhar como o circuito está no momento presente, é preciso analisar como ele estava antes.

Para que se possa ter essa propriedade, os elementos fundamentais são os elementos de memória, que são aqueles que guardam a informação do estado anterior do circuito. Os elementos de memória podem ser fabricados de diversas formas, com maior ou menor quantidade de elementos lógicos, ou mesmo de técnicas de fabricação digital.

Os elementos de memória que vamos analisar neste capítulo são comumente chamados de flip-flop. Normalmente, são utilizados para a construção de sistemas digitais bastante complexos.

Atualmente, os elementos de armazenamento em massa não utilizam flip-flops como elementos de armazenamento; porém, toda a lógica sequencial que faz esses elementos de armazenamento em massa funcionarem usa flip-flops.

Quando nos referimos aos elementos de armazenamento em massa, estamos considerando as memórias RAM de altíssima capacidade, na casa de gigabytes dentro do mesmo chip, ou as memórias SSD, que guardam as informações armazenadas por um longo tempo, mesmo sem energia.

PARA SABER MAIS

Pesquise como são construídos os sistemas de armazenamento em massa, como as memórias SSD e as memórias RAM, e procure entender suas diferenças e a importância de cada item na construção de um sistema real.

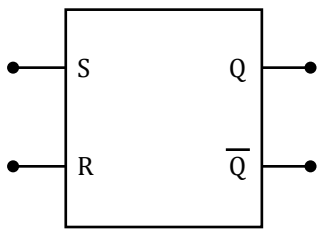
Os flip-flops são os elementos que permitem os comportamentos complexos observáveis nos diversos produtos eletrônicos que manipulamos no dia a dia.

1 Flip-flop RS básico

Os flip-flops do tipo RS básico são os elementos de memória mais simples que podemos usar nos circuitos sequenciais. Tipicamente, quando usamos esse dispositivo como elemento de memória, cada flip-flop corresponde a um bit.

Do ponto de vista externo, o flip-flop RS tem dois pinos de entrada e dois pinos de saída, conforme pode ser conferido na figura 1.

Figura 1 – Flip-flop RS Básico



Fonte: adaptado de Idoeta e Capuano, 1999.

Os pinos de entrada são S e R, e os pinos de saída, Q e $\overline{Q}$. Q e $\overline{Q}$ são sempre complementares, ou seja, se Q = 1, $\overline{Q}$ = 0, e se Q = 0, $\overline{Q}$ = 1. Assim, temos apenas duas possibilidades de estado. S é uma abreviação de set, e R é uma abreviação de reset, o que já indica a função desses pinos.

A capacidade de armazenamento do flip-flop RS básico é decorrente de sua característica biestável, ou seja, ele tem a capacidade de ficar estável tanto com suas saídas em Q = 0 e $\overline{Q}$ = 1 como com suas saídas em Q = 1 e $\overline{Q}$ = 0.

Entretanto, essa estabilidade é válida quando as entradas estão em nível 0, ou seja, enquanto não há um estímulo em uma das entradas, que são ativas em nível 1. O estado em que o flip-flop se encontrar vai permanecer até que seja retirada a energia que o mantém funcionando.

Quando dizemos que as entradas são ativas em 1, significa que, se o valor da entrada for 1, a sua função é ativada; se o valor da entrada for 0, sua função para de ter efeito.

O pino S, que corresponde à função set, faz com que as saídas do flip-flop, uma vez acionado, vão para o estado Q = 1 e $\overline{Q}$ = 0, independentemente do estado anterior. Se o pino S voltar ao estado inativo, o valor de Q e $\overline{Q}$ será mantido.

O pino R, que corresponde à função reset, faz com que as saídas do flip-flop, uma vez acionado, vão para o estado Q = 0 e $\overline{Q}$ = 1, independentemente do estado anterior. Se o pino R voltar ao estado inativo, o valor de Q e $\overline{Q}$ será mantido.

Entretanto, como parte do projeto do flip-flop RS, há um estado proibido, que é o estado em que R e S estão ativos. Se esse estado ocorrer, o comportamento do flip-flop RS não pode ser determinado de antemão. Por isso, cabe ao desenvolvedor assegurar que essa situação não ocorra durante a operação normal do dispositivo que está sendo projetado.

Nos próximos parágrafos, vamos nos referir ao estado do flip-flop apenas pelo valor de Q, visto que $\overline{Q}$ é sempre o complemento deste. Mais adiante, no texto, o valor de $\overline{Q}$ será importante para a compreensão de como o flip-flop se mantém estável e pode ser utilizado como um elemento de memória.

Com base nesse comportamento, podemos criar uma tabela-verdade em que Qa é o estado atual do flip-flop e Qf, o estado futuro. Lembrando que o estado anterior de R e S sempre será de ambas as entradas em 0,

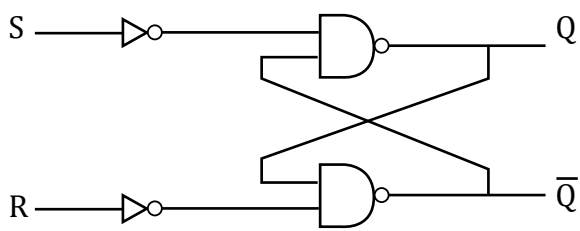
e o estado posterior dessas entradas, após o estado descrito na tabela 1, a seguir, será de ambas as entradas em 0 (IDOETA; CAPUANO, 1999).

Tabela 1 — Transição de estados do flip-flop RS básico

S	R	Qa	Qf
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	Proibido
1	1	1	Proibido

Resumidamente, se S ou R se tornam ativos, eles impõem o valor na saída. Para entender como isso ocorre, vamos analisar como um flip-flop RS é tradicionalmente construído, conforme expresso na figura 2.

Figura 2 — Esquema elétrico de um flip-flop RS



Fonte: adaptado de Idoeta e Capuano, 1999.

Esse elemento, como já mencionamos anteriormente, é um elemento biestável, ou seja, tem dois estados estáveis. Essa estabilidade é decorrente da retroalimentação de seu estado representado por Q e $\overline{Q}$.

Para que possamos entender como isso funciona, vamos analisar as situações em que o flip-flop pode se encontrar — exceto a situação proibida, pois essa é uma situação em que ele não consegue ficar estável, e a pessoa responsável pelo projeto do circuito deve evitá-la.

Para simplificar o texto, vamos nomear as NANDs do circuito da seguinte forma: NAND 1 é a porta cuja saída está ligada em Q, e NAND 2, a porta cuja saída está ligada em $\overline{Q}$.

1.1 Situação 1

A primeira situação que vamos analisar é aquela em que R e S estão em 0 (ou seja, inativos), Q = 0 e $\overline{Q}$ = 1.

Nessa situação, na NAND 1, a entrada vinda da porta inversora ligada à entrada S encontra-se em estado 1; na entrada vinda da saída $\overline{Q}$, o valor também é 1. Pela tabela-verdade de uma NAND, a saída da NAND 1 é obrigatoriamente 0, valor que já estava ali inicialmente.

Além disso, na NAND 2, temos uma das entradas, em estado igual a 1, vinda da inversora ligada à entrada R, e a outra entrada, em nível 0, vinda da entrada Q. Pela tabela-verdade de uma NAND, a saída dessa NAND 2 tem de ser 1, que é o valor que já estava em $\overline{Q}$.

Essa situação é estável, e se nenhuma das entradas, S ou R, ficar ativa, esse estado não vai mudar.

1.2 Situação 2

A segunda situação que vamos analisar é quando uma das entradas, S ou R, se torna ativa. O raciocínio é exatamente o mesmo para ambas; a diferença é que, ao ativarmos a entrada S, forçamos a saída Q para 1, e, ao ativarmos a entrada R, forçamos a saída Q para 0 – lembrando que o circuito é simétrico.

Para simplificar nossa análise, vamos tornar ativa a entrada S, que terá o efeito de mudar o estado que descrevemos na situação 1.

No estado prévio da entrada da NAND 1, ambas as entradas estavam em 1 e, portanto, as saídas da NAND 1 estavam em zero.

A entrada S é ativada ao colocarmos o valor 1 em S. Após a passagem pelo inversor, temos, na entrada dessa NAND, uma entrada em valor 1 (que vem de $\overline{Q}$) e um valor zero (que vem de S). A saída da NAND 1 vai para 1 (lembrando que a saída da NAND 1 é Q).

Essa mudança em Q provoca uma mudança na entrada da NAND 2, que agora vai ter ambas as entradas em 1, e, de acordo com a tabela-verdade de uma NAND, a saída da NAND 2 vai ser zero, que é o valor de $\overline{Q}$.

A partir do momento em que a mudança provocada pela ativação de S se propaga para Q e, em seguida, para $\overline{Q}$, mesmo desativando a entrada S (voltando para o valor S = 0), o flip-flop fica novamente estável e não muda de estado até que exista uma nova ativação em R.

1.3 Situação 3

A terceira situação que vamos abordar é a continuidade da situação 2. Neste ponto, Q = 1 e $\overline{Q}$ = 0, e R e S estão em zero, ou seja, estão inativos. Além disso, uma das entradas da NAND 1 está ligada pelo inversor à entrada S em valor igual a 1, e a outra está ligada na saída $\overline{Q}$ com valor igual a 0. Ou seja, a saída Q, neste momento, é igual a 1.

Ao ativarmos a entrada S colocando-a em nível 1, após a inversora, na entrada da porta NAND 1, teremos as duas entradas em nível zero. Isso mantém o estado do flip-flop inalterado.

Neste ponto, se tornarmos ativa a entrada R, o efeito que teremos é similar à situação 2, porém, com sinal trocado.

É importante notar que o efeito de memória é obtido pela retroalimentação das duas portas NAND do flip-flop.

As situações em que S e R ficam ativos ao mesmo tempo não são permitidas, porque o circuito ficaria instável. Seu comportamento dependerá da tecnologia de fabricação.

Entretanto, tipicamente, colocar um flip-flop RS em um estado não permitido gera uma oscilação dependente do tempo de propagação das NANDs envolvidas ou, no pior caso, leva à autodestruição do dispositivo; porém, não é possível prever com exatidão o comportamento desse modelo de flip-flop no caso de acionamento no estado proibido.

2 Clock

Antes de trabalharmos o conceito de clock, ou relógio, é necessário introduzir alguns conceitos fundamentais para entendê-lo.

O primeiro desses conceitos é o tempo de propagação, que nada mais é do que o tempo entre a transição do nível de uma entrada de uma porta lógica e o efeito dessa transição na saída dessa porta lógica.

O tempo de propagação de uma única porta lógica depende de duas variáveis importantes: a tecnologia de construção das portas lógicas e sua quantidade de transistores.

Entretanto, é muito comum que nos circuitos digitais combinacionais tenhamos diversas entradas que, no final, sejam ligadas a uma saída. Além disso, é comum que uma entrada seja utilizada em diversas portas lógicas, ou que seu efeito seja refletido em diversas portas lógicas, fazendo com que um único acionamento de uma entrada provoque oscilações na saída final do circuito.

Ou seja, entre o acionamento de uma entrada de um circuito combinacional e seu efeito final, existe um tempo de propagação, e esse

tempo não é necessariamente o mesmo quando o sinal de entrada faz a transição de 0 para 1, e de 1 para 0.

Além disso, como é possível ter vários caminhos para a propagação dessa transição, o pino de saída do circuito combinacional poderá ficar variando durante algum tempo até que se estabilize.

No jargão da eletrônica digital, essas transições são chamadas de "glitchs". Esses glitchs, em geral, são muito rápidos e não correspondem a informações reais, mas apenas a acomodações dos tempos de propagação.

Não é o objetivo desta obra discutir as diversas tecnologias disponíveis para a construção de portas lógicas, mesmo porque essa área do conhecimento humano avança de forma muito rápida, o que tornaria este material obsoleto em questão de semanas. O que importa, aqui, é notar que o tempo é um fator fundamental para o funcionamento correto de um circuito digital.

Temos de lembrar também que, em circuitos sequenciais, a sequência dos acionamentos é fundamental para que se tenha o valor correto nos flip-flops que mantêm o sistema funcionando.

Mas, analisando esses problemas, qual seria a forma de resolver a questão dos glitchs? Como podemos tornar nossos circuitos digitais confiáveis e evitar que os glitchs provoquem um funcionamento indesejado de nosso circuito sequencial?

A forma mais simples de conseguir esse resultado é fazer com que as entradas dos nossos flip-flops sejam avaliadas – ou seja, possam ter efeito no estado do flip-flop – apenas quando soubermos que todos os tempos de propagação possíveis já foram cumpridos e, portanto, que a saída dos circuitos combinacionais, que acionam os circuitos sequenciais, já estão estáveis.

É aí que entra o conceito de clock, ou relógio, em portugês. O clock tem a função de organizar o funcionamento de todo o circuito digital,

fazendo com que ele se comporte da forma como foi projetado para se comportar. Esse é um dos principais motivos pelo qual os processadores de todos os tipos possuem sinais de clock e determinam a velocidade com que as informações são tratadas dentro de um sistema digital.

É comum associarmos a velocidade de um processador à sua frequência de clock, ou seja, ao número de ciclos por segundo com que aquele sinal oscila do estado 0 para o estado 1, e do estado 1 para o estado 0. Mas o que determina a frequência máxima de clock de um sistema digital?

Essencialmente, o que determina a frequência máxima de um sistema digital são os tempos de propagação do seu circuito combinacional mais complexo. Ou seja, a pessoa que está projetando o sistema digital, ao analisar os diversos circuitos combinacionais presentes dentro do sistema, determina o tempo mais longo entre uma transição de uma entrada e a estabilização de sua saída, e ainda acrescenta um pequeno tempo para garantir que isso seja correto em todo o circuito.

Existem várias estratégias avançadas em relação a como distribuir o sinal de clock dentro de um sistema digital sequencial, ou ainda várias formas de variar a frequência de clock durante o funcionamento do circuito, mas o motivo para se fazer isso é um tópico que não será abordado em profundidade neste livro.

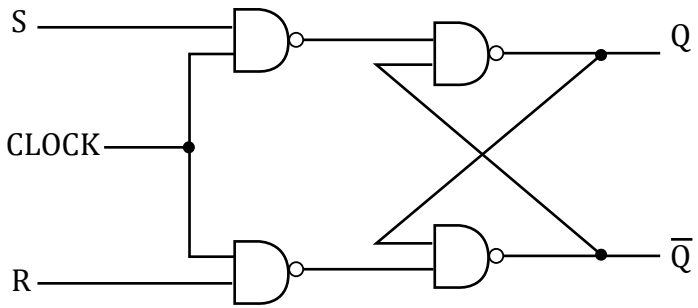
3 Flip-flop RS com entrada de clock

Uma vez que compreendemos o funcionamento de um flip-flop RS básico e o objetivo básico do sinal de clock em um sistema digital, vamos evoluir nosso flip-flop RS básico para que ele absorva a funcionalidade de um sinal de clock de entrada.

Como já descrito anteriormente, uma das principais funções de um sinal de clock em um circuito digital é proteger os circuitos sequenciais das oscilações normais dos circuitos combinacionais.

A seguir, na figura 3, temos o circuito que expande a funcionalidade de nosso flip-flop RS básico para que ele tenha uma entrada de clock.

Figura 3 — Flip-flop RS com clock

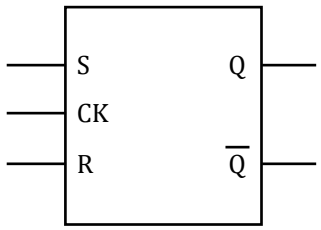


Fonte: adaptado de Idoeta e Capuano, 1999.

Nesse circuito, o sinal de clock controla os momentos em que o flip-flop RS pode ser acionado. Aplicando a tabela-verdade de uma NAND, a saída das portas 3 e 4 só poderá ser modificada enquanto o sinal de clock estiver em nível 1. Da mesma forma, se o sinal de clock estiver em nível 0, mesmo que os sinais R e S oscilem, isso não terá efeito no estado do flip-flop.

O símbolo que representa o flip-flop RS com clock é apresentado na figura 4, a seguir.

Figura 4 – Símbolo de um flip-flop RS com clock



Fonte: adaptado de Idoeta e Capuano, 1999.

Note que a inserção do pino de clock no flip-fop RS diminui o problema das oscilações nas entradas S e R, porém, não elimina a contrariedade das combinações proibidas de S e R. Esse problema será resolvido no próximo modelo de flip-flop a ser analisado.

4 Flip-flop JK

Os flip-flops JK têm por objetivo resolver um dos problemas que ainda persistem nos flip-flops RS, que é o fato de haver combinações proibidas em suas entradas S e R. A solução para esse problema foi criar um comportamento para quando ocorrer essa condição.

Para tanto, nos flip-flops JK, no lugar de S temos a entrada J, e no lugar de R temos a entrada K. Sem a lógica que implementa a função de J e K, se aplicássemos o valor 1 em ambas as entradas, teríamos a condição proibida; porém, como solução, escolheu-se inverter o estado atual do flip-flop.

Para exemplificar essa situação, vamos chamar de Qa a condição em que o sinal de clock está em 0 (ou seja, a fase em que o clock bloqueia alterações no estado do flip-flop) e em que aplicamos J = 1 e K = 1; e vamos chamar de Qf a condição após o clock ser modificado para 1 (ou seja, a fase em que o clock permite que ocorram alterações no estado do flip-flop).

Se Qa = 1, assim que o clock mudar de estado, teremos Qf = 0. Se Qa = 0, após o clock mudar de estado, teremos Qf = 1. Nas demais situações, o comportamento do flip-flop JK é idêntico ao de um flip-flop RS.

Assim, temos como resultado a tabela-verdade a seguir.

Tabela 2 — Transição de estado do flip-flop JK

J	K	Qf
0	0	Qa
0	1	0
1	0	1
1	1	$\overline{Qa}$

Uma condição importante é que o tempo em que o clock fica no estado em que se permite a alteração de estados do flip-flop deve ser curto, para evitar que o estado do flip-flop se altere novamente, tornando-o instável. Entretanto, no item 6 deste capítulo, esse problema será resolvido de forma que o flip-flop fique sempre estável.

O símbolo do flip-flop JK é idêntico ao do flip-flop RS, exceto pelo nome das entradas, que em vez de serem S e R são J e K, respectivamente.

5 Flip-flop JK com preset e clear

Com a adição do circuito para resolver o problema do estado proibido do flip-flop RS, eliminamos uma contrariedade que poderia fazer com que o flip-flop assumisse valores desconhecidos durante o funcionamento do produto que está sendo desenvolvido. Porém, isso não resolve todos os problemas de não se saber de antemão qual o estado em que o flip-flop deveria estar.

Quando se coloca a alimentação em um circuito digital sequencial, *a priori* não se sabe em qual estado cada um dos flip-flops vai estar, e precisamos de uma forma de ter certeza desse estado no início da operação do produto em desenvolvimento. Então, necessitamos de uma forma de forçar o estado desejado na inicialização.

A forma de resolver esse problema passa pela adição de dois sinais especiais, que têm prioridade em relação ao clock e permitem que o

dispositivo inicie seu funcionamento em um estado conhecido. Esses sinais são o preset e o clear.

Em geral, na construção dos flip-flops, esses sinais são ativos em nível baixo e, quando ativos, têm prioridade em relação a todos os demais sinais do flip-flop.

Entretanto, assim no caso do flip-flop RS, esses dois sinais não podem estar ativos ao mesmo tempo, então, cabe à pessoa que está desenvolvendo o dispositivo garantir que apenas um deles seja acionado.

É muito comum — porém, não é obrigatório — que o projetista escolha ligar um desses sinais a um circuito externo analógico, que só permite o funcionamento do circuito digital sequencial após o tempo necessário para que a alimentação e mesmo o clock do sistema estejam estáveis o suficiente para que o sistema funcione normalmente.

Esse circuito externo é comumente chamado de circuito de reset. O sinal de preset, quando ativo, força o estado do flip-flop para Q = 1. O sinal de clear, quando ativo, força o estado do flip-flop para Q = 0.

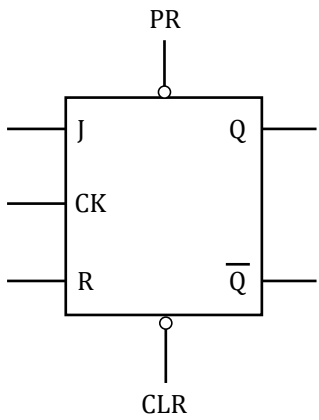
Novamente, utilizando a nomenclatura de Qf para indicar o estado de funcionamento do dispositivo após se tornarem ativos os sinais em análise – e lembrando que preset e clear são ativos com nível lógico 0 –, temos a tabela 3, a seguir, de transição.

Tabela 3 — Transição de estado do flip-flop JK com preset e clear

PRESET	CLEAR	Qf
0	0	Não permitido
0	1	1
1	0	0
1	1	Funcionamento normal

O símbolo do flip-flop JK com preset e clear é representado na figura 5, a seguir.

Figura 5 — Símbolo do Flip-flop JK com preset e clear



Fonte: adaptado de Idoeta e Capuano, 1999.

Os pinos de preset e clear estão com bolinhas para sinalizar que eles são ativos em nível 0.

6 Flip-flop JK mestre e escravo

No final do item “Flip-flop JK”, apontamos um problema construtivo no projeto dos flip-flops JK que trabalham com clock em nível: o fato de que o tempo que o clock pode permitir alterações no estado do flip-flop deve ser curto, caso ele seja utilizado no modo de inversão (J = 1 e K = 1). Além disso, vimos que enquanto o clock estiver em nível habilitado, o estado do flip-flop será alterado, e isso pode ser um problema em circuitos mais complexos.

Para resolver esse problema, foi introduzido o modelo de flip-flop JK mestre e escravo. Esse modelo é construído com base na junção de

dois flip-flop RS, porém, na entrada do conjunto, é adicionado um circuito para implementar a função de JK (ou seja, retirar o estado proibido, transformando-o em modo de inversão).

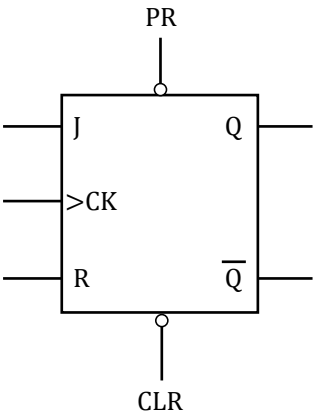
Um detalhe é que cada flip-flop RS trabalha em um nível diferente de clock ativo. Assim, enquanto o clock permite a modificação do estado do primeiro flip-flop, isso não se reflete no resultado do segundo flip-flop; e quando o clock permite a alteração do estado do segundo flip-flop, sua entrada já está estável, pois o estado do clock do primeiro flip-flop não permite que mais alterações ocorram, independentemente do que possa ocorrer nas entradas J e K. Dessa forma, o valor que vale, para fins de computação do resultado, é o momento da segunda transição do valor do clock.

Essas transições do valor do clock são chamadas de "bordas". Quando o clock faz a mudança de estado de 0 para 1, chamamos essa transição de "borda de subida". Quando o clock faz a mudança de estado de 1 para 0, chamamos essa transição de "borda de descida".

Quando a segunda transição de um flip-flop JK mestre e escravo ocorre numa borda de subida, dizemos que ele é ativo na borda de subida; na outra direção, quando a segunda transição ocorre em uma borda de descida, dizemos que o flip-flop é sensível à borda de descida.

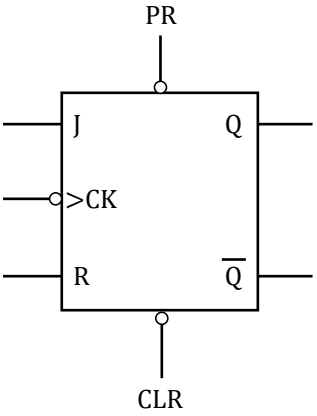
Para sinalizar que um flip-flop é do tipo mestre e escravo sensível à borda de subida, utilizamos o símbolo expresso pela figura 6, a seguir.

Figura 6 — Símbolo de um flip-flop JK mestre e escravo sensível à borda de subida



Para sinalizar que um flip-flop é do tipo mestre e escravo sensível à borda de descida, utilizamos o símbolo expresso pela figura 7, a seguir.

Figura 7 — Símbolo de um flip-flop JK mestre e escravo sensível à borda de descida



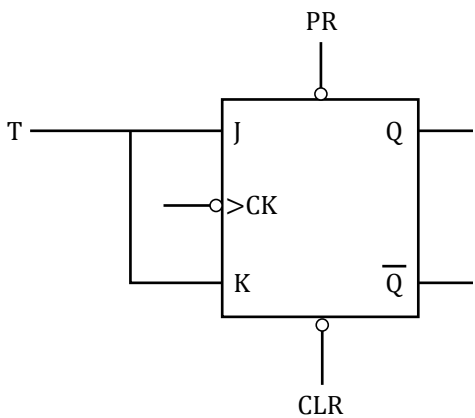
7 Flip-flop tipo T

Os flip-flops do tipo T são um caso particular de implementação dos flip-flops JK sensíveis à borda, em cuja fabricação as entradas J e K são unidas no circuito. Quando o flip-flop tipo T está ligado dessa forma e sua entrada T está ativa, seu estado é comutado; ou seja, se seu estado era Q = 1, após a borda ativa do clock ele será Q = 0, e se seu estado era Q = 0, após a borda ativa do clock será Q = 1.

Além disso, em geral, esse flip-flop é fabricado com as entradas pre-set e clear, permitindo que se fixe um estado inicial para ele.

Na figura 8, a seguir, temos o símbolo de um flip-flop tipo JK ligado na configuração de flip-flop tipo T ativo na borda de descida do clock.

Figura 8 — Ligação de um flip-flop JK para formar um flip-flop do tipo T



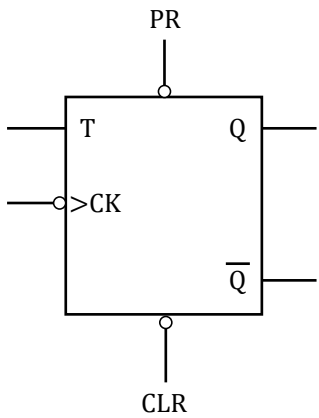
Dessa forma, a tabela-verdade do flip-flop tipo T é exibida na tabela 4, a seguir.

Tabela 4 — Tabela de transição de estados do flip-flop tipo T

T	J	K	Qf
0	0	0	Qa
1	1	1	$\overline{Qa}$

Na figura 9, a seguir, temos o símbolo de um flip-flop tipo T ativo na borda de descida do clock.

Figura 9 — Símbolo de um flip-flop do tipo T sensível à borda de descida



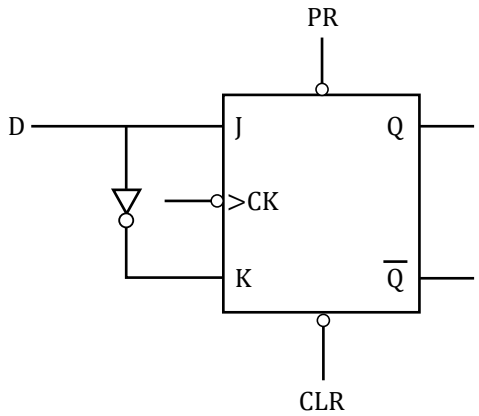
Esses dispositivos podem ser sensíveis à borda de descida ou à borda de subida.

8 Flip-flop tipo D

Os flip-flops do tipo D são outro caso particular de implementação dos flip-flops JK sensíveis à borda, porém, em vez de simplesmente unir as entradas J e K na entrada D, o que se faz é ligar D em J, além de ligar uma porta NOT entre a entrada D e a entrada K.

Confira o esquema de ligações da figura 10, a seguir.

Figura 10 — Ligação de um flip-flop JK para formar um flip-flop do tipo D



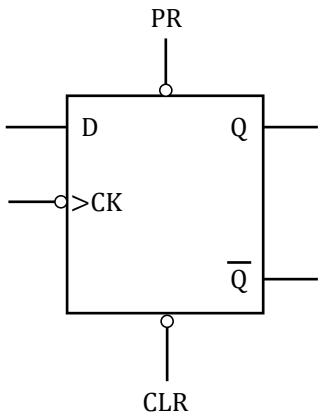
O efeito de fazer essa ligação é que, a cada borda ativa do clock, o estado de D é copiado para Q. Assim, temos como resultado a tabela de transições a seguir, que é válida a cada transição da borda ativa no flip-flop.

Tabela 5 — Tabela de transição de estados do flip-flop tipo D

D	J	K	Qf
0	0	1	0
1	1	0	1

Na figura 11, a seguir, temos o símbolo de um flip-flop do tipo D.

Figura 11 — Símbolo de um flip-flop do tipo D sensível à borda de descida



Esse modelo de flip-flop pode ser fornecido com borda ativa na subida ou ativa na descida.

Considerações finais

Neste capítulo, tivemos contato com os elementos básicos de memória e exploramos um pouco sua evolução em termos de complexidade e funcionalidade. Além disso, foi introduzido o conceito de clock e foram abordadas algumas das técnicas utilizadas para tornar os circuitos digitais sequenciais mais robustos.

É importante que você explore formas de utilizar os diversos modelos de flip-flops apresentados, e onde cada um deles é mais ou menos útil. Isso certamente lhe trará ideias para a implementação de projetos cada vez mais práticos.

Referências

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

Capítulo 8

Registradores e contadores

Neste capítulo, vamos aprofundar um conceito muito importante para o desenvolvimento de sistemas digitais sequenciais complexos: os registradores.

Esses elementos são os componentes que mantêm a informação nos sistemas sequenciais e permitem comportamentos mais complexos, como máquinas de estado.

Em particular, os registradores são componentes muito presentes em unidades centrais de processamento, comumente chamadas de CPU ou processadores.

Não vamos nos aprofundar na descrição de como é construída uma CPU; entretanto, é importante saber que são os registradores dentro das CPUs que permitem que as instruções sejam executadas pelos

circuitos combinatórios. Da mesma forma, é por meio dos registradores que o programador interage com a CPU, por exemplo, configurando o comportamento que espera dela ou lendo o resultado de uma instrução executada anteriormente.

1 Registradores

Os registradores, como citado anteriormente, armazenam e permitem a manipulação de informações no domínio digital, além de serem a base de qualquer sistema digital sequencial.

Para a construção de um sistema sequencial, são necessárias várias funcionalidades atribuídas aos registradores, e com isso temos diversas formas de construir os registradores mais comumente utilizados em sistemas digitais sequenciais.

O elemento básico da construção dos registradores são os flip-flops, que já foram abordados no capítulo anterior. Além dos flip-flops, vamos precisar de circuitos combinatórios que controlam quando e como uma informação é armazenada ou manipulada com o uso dos registradores.

Para que um chip seja chamado de registrador, ele deve possuir mais do que um único flip-flop; é comum que um único chip tenha arranjos com 2 a 16 flip-flops.

Em circuitos integrados complexos – como microprocessadores, memórias, elementos de lógica programável, entre outros –, o número de bits de um registrador teoricamente não tem limite, e eles sempre são fabricados de acordo com a necessidade de cada projeto.

Podemos classificar os registradores pela forma como as informações são armazenadas e extraídas; além disso, a pessoa que está projetando o circuito pode combinar mais de um modelo para atender às necessidades de seu projeto.

A entrada de dados em um registrador pode ser feita de duas formas: paralela ou serial. Do mesmo modo, a extração dos dados de um registrador pode ser feita de duas formas: paralela ou serial.

Assim, podemos ter as seguintes combinações de configurações de entrada e saída de um registrador:

- entrada paralela, saída paralela;
- entrada paralela, saída serial;
- entrada serial, saída paralela; ou
- entrada serial, saída serial.

Os elementos básicos de um registrador são os flip-flops do tipo D, em que o valor que é colocado em sua entrada é o mesmo valor que será apresentado na saída. O flip-flop tipo D já foi descrito em detalhe no capítulo anterior.

Em todos os casos, qualquer registrador terá uma entrada de clock, que é o elemento que vai dar sincronismo e sinalizar aos registradores o momento em que os dados de entrada estarão estabilizados para que se possa fazer a amostragem.

Além da entrada de clock, é comum existir ao menos um pino de enable, que é o pino que permite que cada flip-flop faça a amostragem dos dados no momento adequado. Entretanto, no caso dos registradores no formato de chips discretos (ou seja, aqueles que podemos comprar e soldar em nosso circuito) é comum que haja dois ou três pinos destinados a essa tarefa.

No caso de haver mais de um pino de enable, é comum que se tenha ao menos um pino de enable que habilita em nível lógico 0 e ao menos um pino de enable que habilita em nível lógico 1. Essa característica tem o objetivo de facilitar o trabalho de quem vai usar o chip em seus projetos.

Além disso, é comum que se tenha ao menos um pino para forçar um nível lógico conhecido em todos os flip-flops do registrador. Em geral, são os pinos de preset ou clear, e o mais comum é que se tenha apenas o pino de clear, para forçar todos os flip-flops a terem valor 0.

Ou seja, são inúmeras as possibilidades de organização dos flip-flops para a criação de registradores, e é impossível fornecer, em um volume didático, exemplos de todas as possibilidades.

Também podemos ligar as entradas e saídas de um registrador com o objetivo de modificar sua funcionalidade original. Como exemplo dessa possibilidade, vamos utilizar um chip comercial de um registrador de entrada paralela e saída paralela em um registrador com entrada serial e saída paralela. Para isso, vamos utilizar o chip SN74HC174.

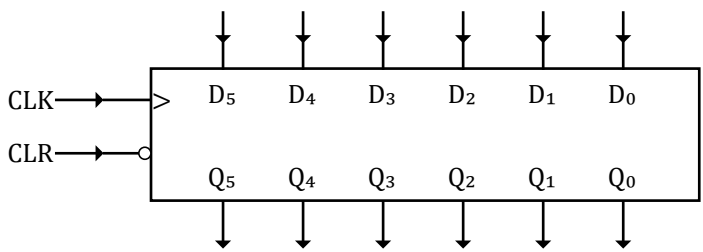
O SN74HC174 é um registrador com 6 flip-flops do tipo D, com entrada paralela e saída paralela. Cada flip-flop desse chip obedece à tabela-verdade a seguir.

Tabela 1 – Tabela-verdade de cada flip-flop do registrador 74HC174

INPUTS			OUTPUT
$\overline{\text{CLR}}$	CLK	D	Q
L	X	X	L
H	↑	H	H
H	↑	L	L
H	L	X	Q_0

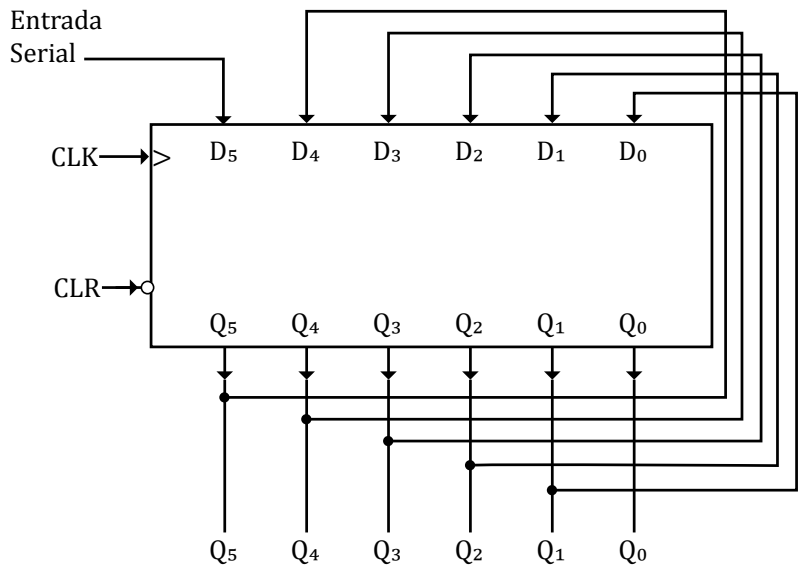
Na figura 1, a seguir, temos o símbolo do chip.

Figura 1 — Símbolo do SN74HC174



Ligando os pinos do SN74HC174, conforme a figura 2, a seguir, teremos um circuito que, a cada pulso de clock, copia o estado do bit de entrada e desloca os demais bits. Ao final de seis pulsos de clock, teremos os 6 bits inseridos de forma serial, na saída paralela.

Figura 2 — Esquema de ligações para transformar o SN74HC174 em um registrador de entrada serial com saída paralela



Analisando o funcionamento do circuito da figura 2, vemos que o que ocorre é que, a cada pulso de clock, a entrada de cada flip-flop é copiada na respectiva saída. Como o circuito foi acionado de forma que a saída de um flip-flop seja ligada na entrada do flip-flop seguinte, temos um deslocamento da informação a cada pulso do clock, transformando uma informação que estava chegando de forma serial em uma informação na forma paralela.

Perceba que o valor de cada bit depende da sequência de acionamentos, que é o que caracteriza um circuito digital sequencial.

2 Contadores

Os contadores são uma classe especial de circuitos digitais sequenciais cujos estados variam em função do acionamento de um sinal de clock, em uma sequência que se repete continuamente.

Um contador pode ser desenhado para as mais diversas tarefas dentro de um sistema digital sequencial. Entre outras finalidades, pode ser utilizado para:

- contar intervalos de tempo;
- contar o número de pulsos dentro de um intervalo de tempo;
- gerar formas de onda;
- fazer uma divisão de frequência; e
- determinar a frequência de um sinal de entrada.

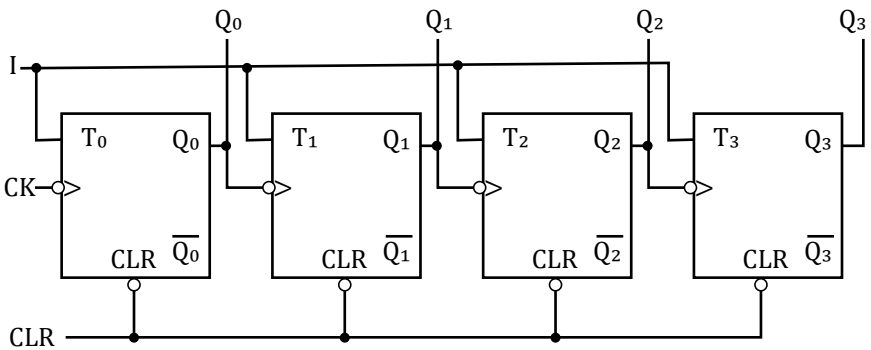
Um contador pode ter uma ou mais saídas, de acordo com a função que se deseja que ele tenha, e o seu número de flip-flops depende do número de estados necessários para que ele realize a tarefa para a qual foi projetado.

Podemos classificar os contadores em duas categorias: assíncronos e síncronos (IDOETA; CAPUANO, 1999).

Os contadores assíncronos têm como característica fundamental o fato de não terem todos os flip-flops acionados pelo mesmo sinal de clock. São circuitos que dependem dos tempos de propagação inerentes às tecnologias de fabricação, e são adequados para algumas aplicações bem específicas.

Resumidamente, eles se caracterizam pelo fato de a saída de dados de um estágio de contagem estimular o clock do estágio seguinte. Um exemplo dessa técnica de construção está na figura 3, a seguir.

Figura 3 — Contador binário assíncrono



Fonte: adaptado de Idoeta e Capuano, 1999.

No circuito da figura 3, a entrada CLR, que podemos ler como "clear" — ou "limpeza" —, faz com que o estado de todos os flip-flops do circuito seja conhecido, ou seja, impõe que o valor de Q0 a Q3 seja igual ao nível lógico 0.

A entrada CK, por outro lado, estimula o clock do primeiro flip-flop, e, na primeira borda de descida de CK, como a entrada T está em nível lógico 1, a saída Q0 terá seu estado invertido, ou seja, Q0 vai comutar do estado 0 para o estado 1. A saída Q0, que está ligada à entrada de clock do segundo flip-flop, teve uma borda de subida, mas como esse flip-flop é sensível à borda de descida, nesse momento nada muda.

Na segunda borda de descida de CK, novamente o estado de Q0 é comutado. Dessa vez, ele sai do nível lógico 1 para o nível lógico 0, o que significa uma borda de descida para o segundo flip-flop do circuito, e isso provoca a comutação do estado de Q1, que iniciou seu funcionamento em 0 e agora tem seu valor modificado para 1. Esse processo se repete em todos os flip-flops do circuito.

Note que a cada duas bordas de descida de CK temos uma borda de descida de Q0, e o mesmo acontece sucessivamente em todos os flip-flops do circuito. Resumidamente, a cada flip-flop que acrescentamos

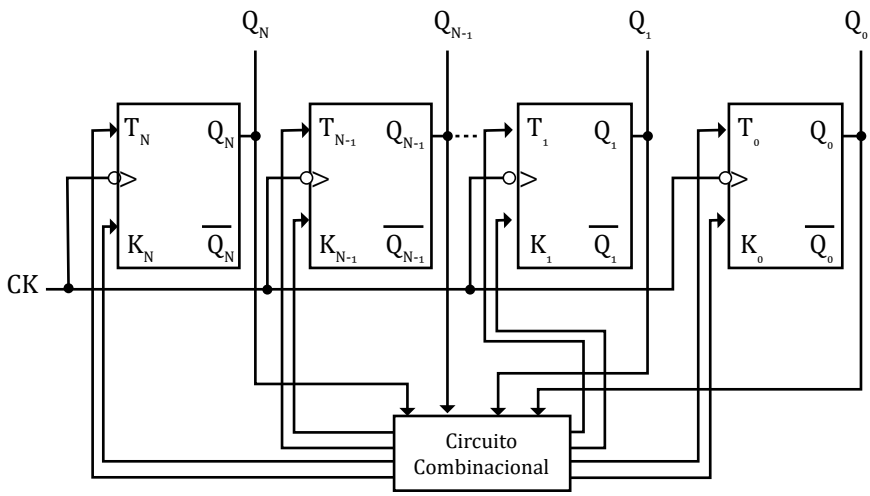
ao circuito, dividimos a frequência de entrada por dois. Nesse circuito, a frequência de Q3 será 16 vezes menor do que a frequência de CK.

Existem diversas técnicas para modificar esse projeto básico, tornando o contador assíncrono bastante flexível. Porém, essas técnicas não serão aprofundadas neste capítulo.

Diferentemente de um contador assíncrono, temos os contadores síncronos, cujos clocks de todos os flip-flops são ligados ao clock de entrada do circuito. Assim, ao contrário de um contador assíncrono, o que determina a contagem de um contador síncrono é como as entradas de dados dos flip-flops são ligadas para criar as transições que caracterizam um contador. Ou seja, dado determinado estado das saídas dos flip-flops, estabelecem-se quais valores devem estar presentes nas entradas dos flip-flops para que, na próxima transição de clock, o próximo valor esperado ocorra.

Do ponto de vista de blocos, um contador síncrono terá um esquema de ligações similar ao da figura 4, a seguir.

Figura 4 – Esquema básico de um contador síncrono:



Fonte: adaptado de Idoeta e Capuano, 1999.

A título de exemplo, vamos construir um contador que gera uma sequência binária crescente nas suas saídas Q. A tabela 2, a seguir, mostra a sequência desejada.

Tabela 2 – Sequência de valores desejados em nosso contador binário

CLK	Q3	Q2	Q1	Q0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0
16	1	1	1	1

Existem várias formas de se chegar a um circuito que tenha esse conjunto de transições. Em nosso exemplo, vamos aproveitar uma característica dos flip-flops JK, que é a propriedade de inverter o valor da saída quando as entradas J e K estão em nível 1.

Analisando as transições do valor de Q0 na tabela 2, podemos notar que, a cada pulso de clock, ele troca de valor. Tendo em vista a propriedade do flip-flop de comutar o valor existente a cada transição de clock, se ambas as entradas estiverem em 1, a solução mais simples para gerar o resultado esperado é simplesmente ligar as entradas J e K do flip-flop 0 no nível lógico 1.

Analisando as transições do valor de Q1 na tabela 2, podemos notar que a mudança de valor ocorre todas as vezes que Q0 está com valor igual a 1. Assim, a solução mais simples para a implementação da lógica combinacional para as entradas J e K do flip-flop 1 é ligar essas duas entradas na saída Q0.

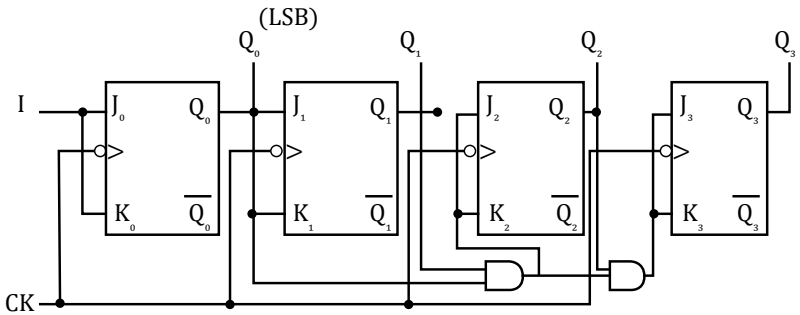
Continuando a nossa análise das transições, agora de Q2, podemos notar que Q2 troca de valor sempre que Q1 e Q0 estão, ambos, com valor igual a 1. Ou seja, devemos comutar o valor de Q2 sempre que Q0 e Q1 estiverem em 1. Essa lógica corresponde a uma porta AND entre Q0 e Q1, cuja saída deve acionar J e K do flip-flop 2.

Podemos notar que um padrão semelhante ocorre com as transições de valor de Q3, ou seja, quando Q0, Q1 e Q2 estiverem com valor 1, devemos comutar o valor de Q3. De forma similar ao que ocorreu com Q2, temos uma operação AND, dessa vez de três entradas, sendo Q0, Q1 e Q2, cuja saída aciona as entradas J e K do flip-flop 3.

Como já temos uma porta AND de duas entradas no circuito – que é a AND que aciona as entradas J e K do flip-flop 2 –, basta fazer uma operação AND entre essa porta e o valor de Q2 para obtermos o circuito que gera o acionamento desejado.

O circuito resultante é apresentado na figura 5, a seguir.

Figura 5 — Contador binário síncrono



Fonte: adaptado de Idoeta e Capuano, 1999.

Note que podemos modificar o circuito combinacional para gerar qualquer sequência de transições necessária à aplicação que se deseja implementar.

Também é possível ligar as entradas preset e clear dos flip-flops a fim de ter estados iniciais conhecidos quando da inicialização do sistema em desenvolvimento.

Por fim, em vez de uma entrada fixa em 1 na entrada do primeiro flip-flop, é possível ter um sinal binário, e o contador poderá contar o número de vezes que o sinal de entrada está em 1 e que ocorreram transições de clock.

Ou seja, há inúmeras formas de aplicar esse conceito básico a fim de usá-lo em projetos de circuitos digitais sequenciais.

Considerações finais

Neste capítulo, pudemos analisar algumas aplicações que usam flip-flops como os elementos básicos de circuitos mais complexos. Pudemos compreender como os elementos digitais sequenciais podem ser utilizados em conjunto com os elementos combinatórios para que tenhamos funcionalidades bastante interessantes e complexas.

Em particular, o modelo de contador síncrono tem inúmeras aplicações e, com algumas modificações, se torna a base dos elementos mais versáteis dos circuitos digitais sequenciais, que são as máquinas de estado, não abordadas neste livro.

Referências

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

Sobre o autor

Stelvio Barboza é mestre em engenharia elétrica pela Universidade de São Paulo (2014), graduado em engenharia elétrica pelo Centro Universitário FEI (2006) e possui curso técnico-profissionalizante em eletrônica pelo Grupo Educacional Flamingo (1999). Atualmente, é coordenador do curso de engenharia de computação e professor de graduação do Centro Universitário Senac. Tem experiência na área de engenharia elétrica, com ênfase em computação e sistemas digitais, atuando principalmente nos seguintes temas: ultra-wideband (UWB), transmissores UWB, micro-ondas, tomógrafo, imagens médicas e processamento de sinais.