ÍNDICE

PREFÁCIO

Até às últimas décadas do século XX, o mundo real era encarado como um sistema em que todas as grandezas tomavam valores contínuos. A percepção que os seres humanos tinham do mundo físico, em termos de sons, imagens ou outras sensações, era a de fenómenos de características inerentemente contínuas, assim como os modelos de sistemas físicos, tanto de sistemas inanimados como de sistemas biológicos. Para modelar e estudar diversos sistemas físicos, foram mesmo desenvolvidos, durante e imediatamente após a Segunda Guerra Mundial, computadores analógicos que permitiam justamente modelar os valores físicos de modo contínuo.

Esta visão veio a sofrer significativas alterações com o aparecimento dos computadores digitais e, sobretudo, com o desenvolvimento da sua utilização, potenciado pelos avanços tecnológicos decorrentes das tecnologias de circuitos integrados e de armazenamento magnético e óptico de informação.

Tornou-se claro, então, que grandezas como a intensidade da luz proveniente de uma dada direcção ou a pressão do ar num determinado ponto no tempo podem ser representadas, com diversas vantagens, por um valor numérico, guardado digitalmente na memória de um computador. A tecnologia veio a permitir não só guardar estes valores (imagens e sons, após adequada transformação do domínio analógico para o domínio digital), como reproduzi-los posteriormente (local ou remotamente) num monitor de computador ou num altifalante, dando origem a um conjunto de actividades muito importantes para a sociedade actual, que incluem, entre outras, as telecomunicações (telefones, telemóveis), e diversos aspectos da indústria do entretenimento (música, cinema, televisão, jogos) que são agora baseadas, de uma maneira ou de outra, na codificação digital de informação.

A estas aplicações, já existentes com tecnologias diferentes antes do aparecimento dos computadores digitais, veio juntar-se um conjunto de aplicações tornadas possíveis pelos computadores e pela sua utilização. De entre estas são de destacar os sistemas de informação (bases de dados, serviços bancários, comércio electrónico), muitos deles potenciados pelo aparecimento da Internet, a rede mundial que interliga a maioria dos computadores do mundo.

Juntamente com a rápida implantação dos computadores e tecnologias digitais, verificou-se que, até certo ponto, os sistemas biológicos são também codificados pela natureza de maneira comparável à da tecnologia digital, uma vez que a informação genética é guardada de um modo discreto nas moléculas de ADN que constituem os cromossomas dos organismos. Isto veio permitir que, como resultado de grandes projectos de sequenciação de genomas, esteja neste momento disponível em bases de dados in-

formação completa sobre o genoma de numerosos organismos, entre os quais o do ser humano. Do mesmo modo, da ligação entre as duas áreas está a emergir uma grande actividade e um interessante potencial de desenvolvimento nesta área de interface entre os sistemas biológicos e a tecnologia digital.

Vivemos, assim, numa era em que, progressivamente, toda a informação é guardada de forma digital. Na sua versão mais simples, a informação digital é armazenada usando algum mecanismo (electrónico, magnético ou óptico) que, ao nível mais baixo, é suportado fisicamente em grandezas com dois valores possíveis, ou mais correctamente, podendo variar dentro de dois leques de valores possíveis. A grandeza física usada para representar um destes valores poderá ser uma tensão eléctrica (no caso de uma memória de um computador), uma reflectividade (no caso de um disco óptico) ou um estado de magnetização (no caso de um disco magnético). Este tipo de informação é representado, logicamente, sempre da mesma forma, através de uma variável discreta, que tipicamente pode assumir os valores 0 ou 1. Uma vez que cada uma das variáveis em causa tem tão pequena capacidade de representação de informação, o uso efectivo de sistemas digitais implica a utilização de um enorme número de variáveis deste tipo para representar informação útil, desde registos em bases de dados até imagens, sons ou vídeo.

O estudo dos sistemas digitais é, assim, fundamental, não só para os profissionais que directamente projectam e operam computadores, mas para todos aqueles que querem perceber, de uma maneira profunda e sistemática, as fundações da sociedade actual. O modo pormenorizado como são codificados filmes, músicas ou comunicações está, obviamente, fora do âmbito de um livro de apresentação como este. Estas técnicas, que evoluíram ao longo das últimas décadas, representam um corpo acumulado de conhecimento muito importante, que não é possível abordar num primeiro estudo da matéria. No entanto, todas elas dependem do conhecimento de técnicas básicas de codificação e processamento de informação digital, que são o objectivo de estudo deste livro.

Pretendemos, assim, apresentar neste livro as técnicas básicas de codificação e representação de informação, e descrever os conceitos fundamentais que estão na base dos sistemas computacionais que processam e transformam esta informação. O leitor interessado poderá depois prosseguir mais a fundo o estudo destas matérias, quer no que respeita à codificação e representação de informação, quer no que respeita às arquitecturas dos sistemas computacionais.

Existem numerosas abordagens à problemática dos sistemas digitais. Num extremo, estão as abordagens puramente matemáticas ou algébricas, que ignoram completamente a componente de implementação. Noutro extremo, estão as abordagens que partem das tecnologias de sistemas electrónicos, dando especial ênfase aos aspectos físicos relacionados com a construção de sistemas digitais.

Neste livro, optou-se por uma abordagem intermédia, em que, embora não ignorando totalmente os aspectos físicos do problema, se encaram os sistemas digitais essencialmente como elementos abstractos de processamento de informação que constituem os blocos de um computador.

Nesta perspectiva, este livro foi concebido como suporte a uma primeira abordagem ao estudo das arquitecturas de computador, tipicamente feita no contexto de duas disciplinas semestrais de cursos do ensino superior. Prevemos que este livro possa ser usado com naturalidade nas áreas da Informática, Electrónica e Electrotecnia, mas também em outras áreas, de índole tecnológica, onde exista interesse em formar estudantes nas tecnologias de sistemas digitais, tais como a Engenharia Mecânica, Engenharia Física ou Engenharia Aeroespacial.

Num primeiro semestre, serão tipicamente cobertos os tópicos de sistemas digitais que constituem a primeira parte do livro, Capítulos 1 a 8, e num segundo semestre, as componentes de arquitectura de computadores descritas na segunda parte do livro, constituída pelos restantes capítulos.

Também é possível considerar a utilização dos Capítulos 9, 10, 11, 13, 14 e 15 como o suporte de uma disciplina de introdução à arquitectura de computadores, de um ponto de vista dos programadores, para alunos que tenham apenas conhecimentos básicos de sistemas digitais. Neste caso, algumas secções destes capítulos terão de ser abordadas de forma necessariamente superficial.

O Capítulo 1 descreve os conceitos fundamentais relacionados com a representação digital de informação, a utilização de diferentes bases de representação, as operações aritméticas nestas bases e as conversões entre bases.

O Capítulo 2 é dedicado ao estudo de funções lógicas e ao modo como elas são manipuladas, sintetizadas e optimizadas. A lógica booleana é apresentada, de uma maneira sistemática, mas numa perspectiva utilitária de descrever um formalismo usado para a manipulação de expressões lógicas.

O Capítulo 3 aborda, de uma maneira necessariamente breve e sintética, as tecnologias que são usadas para implementar circuitos lógicos e as limitações impostas pelas restrições físicas ao projecto de circuitos digitais.

O Capítulo 4 inicia o processo de integração de componentes fundamentais, com o objectivo de construir o que serão os blocos básicos de computadores. Neste capítulo, são descritos módulos combinatórios de média complexidade, construídos a partir das portas lógicas estudadas anteriormente.

O Capítulo 5 aborda a construção de módulos aritméticos que permitem executar as

operações lógicas elementares em base 2, e alguns aspectos das questões relacionadas com o desempenho desses módulos.

O Capítulo 6 apresenta pela primeira vez o conceito de comportamento sequencial, e descreve os circuitos que preservam o estado de um sistema (básculas) e que exibem este tipo de comportamento.

O Capítulo 7 é dedicado ao projecto, análise e optimização de circuitos sequenciais, que usam como elementos básicos as básculas estudadas no capítulo anterior.

O Capítulo 8, que poderá ser considerado como o último capítulo referente à primeira parte da matéria, descreve a forma como os circuitos de média complexidade estudados nos Capítulos 4, 5 e 6 podem ser interligados, de modo a poderem processar dados complexos, quando controlados pelos sistemas estudados no Capítulo 7.

O Capítulo 9 representa uma introdução geral aos computadores, vistos de uma perspectiva generalista. Tem como objectivo fazer a transição entre a análise pormenorizada que é feita na primeira parte do livro, dedicada aos sistemas digitais, e a análise de alto nível que irá caracterizar progressivamente a segunda parte do livro, dedicada à arquitectura de computadores.

O Capítulo 10 apresenta os conceitos de instrução e de conjunto de instruções de um processador, e estuda o modo como as instruções são especificadas, executadas e codificadas. Esta capítulo introduz também o processador P3, um Pequeno Processador Pedagógico, que é a plataforma que será usada para exercitar os conceitos de programação e arquitectura que são os tópicos dos dois capítulos seguintes.

O Capítulo 11 é dedicado às técnicas de programação em linguagem *assembly*, usando o processador P3 como plataforma para o estudo e desenvolvimento de pequenos projectos de programação.

O Capítulo 12 é dedicado ao estudo da estrutura interna de um processador, usando novamente o P3 como caso de estudo, desta vez como um exemplo concreto do modo como os circuitos internos de um processador simples são projectados, tanto na componente de circuito de dados como na componente de circuito de controlo.

Os três capítulos que se seguem cobrem, de um modo necessariamente sumário, tópicos de arquitectura que não podem ser abordados de uma maneira completa e sistemática num livro com as características de um livro de apresentação como o presente. Têm como objectivo apresentar os conceitos fundamentais envolvidos na utilização de memórias, periféricos e técnicas avançadas de arquitectura, sem ter a pretensão de cobrir estes tópicos de um modo completo ou exaustivo.

O Capítulo 13 é dedicado ao estudo dos sistemas de memória associados a processadores, sendo abordados os conceitos básicos relacionados com a organização de mapas de memória, utilização de *caches* e sistemas de memória virtual.

O Capítulo 14 aborda as questões relacionadas com operações de entradas/saídas e de utilização de periféricos em sistemas de computadores.

Finalmente, o Capítulo 15 representa uma breve introdução a tópicos mais avançados de arquitecturas de computadores, focando temas como a utilização de *pipelines* e as alternativas que se desenvolveram para a exploração do paralelismo inerente aos programas de computador. Para substanciar este estudo, é apresentado o processador P4 (Pequeno Processador Pedagógico com *Pipeline*) que ilustra alguns dos conceitos abordados.

Além do texto propriamente dito, e das séries de problemas incluídas no livro, existem diversos recursos que foram criados durante o desenvolvimento deste trabalho, e que podem ser usados para apoiar uma formação nesta área. Os conceitos apresentados sobre arquitecturas de microprocessadores são ilustrados através da sua concretização na realização de um processador simples e didáctico, o P3. De forma a permitir aos alunos realizar trabalhos nas aulas práticas sobre a mesma arquitectura estudada nas aulas teóricas, o P3 foi descrito em VHDL e implementado em hardware numa placa com uma FPGA, memória externa e um conjunto de dispositivos de interface. Em simultâneo, desenvolveu-se um *assembler* para o *a*ssembly do P3 e um simulador para esta mesma arquitectura. Assim, os alunos podem desenvolver os seus programas em *assembly* do P3, gerar o código executável, correr os programas no simulador e fazer o carregamento desse executável para a placa, através da porta paralela do computador. Pretendeu-se que o simulador emulasse por completo a placa, nomeadamente em termos dos periféricos disponíveis e da sua interface. Por um lado, o simulador permite que os alunos não estejam limitados ao laboratório para executar os seus programas e, por outro, representa uma ferramenta preciosa para a depuração dos seus programas, uma vez que o processo de depuração na placa é muito mais complexo. A limitação fundamental do simulador é a velocidade de execução. Na placa, o P3 tem uma frequência de funcionamento de $6{,}25$ MHz. Para além da programação a nível de *assembly*, ambas as versões do P3 permitem uma fácil alteração do conteúdo das ROMs de controlo, o que permite não só a alteração do microprograma das instruções existentes, como também a criação e microprogramação de novas instruções *assembly*. Estas ferramentas (*assembler*, simulador, implementação em VHDL) estão livremente disponíveis no sítio da Internet deste livro. Pormenores desta implementação podem ser consultados no Apêndice A.

Nenhum prefácio está concluído sem a necessária secção de agradecimentos. Face ao tempo que este livro demorou a ser concluído, os primeiros agradecimentos vão necessariamente para as nossas famílias que, durante alguns anos, aceitaram sem reclamações a já habitual desculpa para a nossa sistemática indisponibilidade. Para a Luísa, a

Irene e a Tila, a quem este livro é dedicado, assim como para os nossos filhos, Miguel, Ana, Helena, Susana e Francisco, a quem roubámos muitas horas para o poder terminar, aqui fica um agradecimento do fundo do coração.

Outros agradecimentos são devidos por contribuições mais técnicas. Os revisores, Pedro Diniz e João Cardoso, leram versões preliminares deste trabalho e contribuíram com muitas sugestões valiosas para o melhorar. Os alunos Jorge Santana, Nuno Barraló e Fausto Ferreira contribuíram para diversos aspectos dos simuladores e implementação em hardware. Os docentes das disciplinas de Arquitectura de Computadores do Instituto Superior Técnico, Carlos Ribeiro, João Gonçalves, José Costa, Nuno Roma e Alberto Cunha, contribuíram, ao longo do tempo, com comentários, sugestões e diversos melhoramentos. Os editores e colaboradores da IST Press, Joaquim Moura Ramos, Miguel Dionísio e Paulo Abreu, assim como os revisores, deram uma preciosa ajuda durante a fase final de edição e de composição. A todos, o nosso muito obrigado. As gralhas, erros e omissões que restam, e serão, seguramente, muitas, são, naturalmente, da nossa inteira responsabilidade.

Finalmente, a declaração, óbvia, mas indispensável, que tudo isto só foi possível devido aos nossos pais.

Lisboa, Julho de 2006

Os autores

1

REPRESENTAÇÃO DIGITAL DE INFORMAÇÃO

Neste capítulo ir-se-á iniciar o estudo do modo como a informação é representada, em forma digital, num computador. Em particular, irão ser abordados os mecanismos através dos quais diversas grandezas podem ser representadas num computador digital, cujos circuitos permitem apenas um de dois valores possíveis.

Para tal, é necessário começar por perceber o modo como números inteiros são representados, tanto em base 10, que é familiar a todos, como em outras bases de numeração, mais apropriadas à manipulação por computadores.

Começam por analisar-se, na Secção 1.1, os sistemas de numeração que estão subjacentes à representação binária de números. A Secção 1.2 é dedicada ao estudo de rudimentos da aritmética binária. A Secção 1.3 aborda a utilização de códigos, quer numéricos (com relevo para os decimais) quer alfanuméricos (para representar outro tipo de informação). A Secção 1.4 termina o capítulo com alguns conceitos de base de organização da representação binária de informação.

1.1 BASES DE NUMERAÇÃO

Este capítulo aborda a representação de números inteiros e fraccionários sem sinal. Mais tarde, no Capítulo 5, este assunto é retomado para abordar a representação de números inteiros com sinal. A representação de números em sistemas digitais tem de ser feita utilizando o suporte possível que é a existência de dispositivos com um funcionamento binário.

Recordando que a representação usual de números assenta na utilização de uma *base de numeração* que é a base 10, natural se torna pensar que a representação de números poderá ser feita, em sistemas digitais, utilizando a base 2.

Estudar-se-á primeiro o caso geral de representação usando uma base genérica b, concretizando posteriormente o caso da base 2

1.1.1 Representação de Números Inteiros em Base b

A representação de um número inteiro é feita utilizando uma sequência de algarismos. O número 435, por exemplo, está representado pela sequência dos algarismos 4, 3 e 5. A interpretação da representação de um número resulta, por um lado, dos algarismos utilizados e, por outro, da sua posição dentro da sequência. Como é evidente, $435 \neq 354$, muito embora os algarismos usados sejam os mesmos.

A posição dos algarismos indica o *peso* com que cada algarismo deve ser interpretado. No exemplo anterior, o algarismo 4, por estar na terceira posição a partir da direita, significa, de facto, 4 centenas. O algarismo 3 representa 3 dezenas, e o 5 representa 5 unidades.

Isso pode ser indicado mais formalmente do seguinte modo:

$$\begin{aligned} 435 &= 400 + 30 + 5 \\ &= 4 \times 100 + 3 \times 10 + 5 \end{aligned} \tag{1.1}$$

ou, explicitando as potências de 10 envolvidas,

$$435 = 4 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 \tag{1.2}$$

O número 435 diz-se representado em *base 10*, uma vez que resulta da soma de sucessivas potências de 10, pesadas cada uma pelo valor do algarismo correspondente de acordo com a Equação 1.2. Para indicar explicitamente que o número se encontra representado em base 10 é usada a seguinte notação: 435_{10}. Para representar um número em base 10 são usados, para indicar os pesos de cada potência de 10, algarismos de 0 a 9, no total de 10 algarismos distintos.

Nada impede a utilização de outra base para representar um número. Considere-se, por exemplo, o número 1161 representado em base 7, o que é habitualmente indicado por 1161_7. Nesse caso esta representação tem o seguinte significado:

$$\begin{aligned} 1161_7 &= 1 \times 7^3 + 1 \times 7^2 + 6 \times 7^1 + 1 \times 7^0 \\ &= 1 \times 343 + 1 \times 49 + 6 \times 7 + 1 \\ &= 435_{10} \end{aligned} \tag{1.3}$$

Verifica-se, assim, que 1161_7 é outra forma de representar o número 435_{10}.

De um modo geral, pode portanto representar-se qualquer número inteiro N em qualquer base b:

$$N = p_{n-1} \times b^{n-1} + p_{n-2} \times b^{n-2} + \cdots + p_1 \times b^1 + p_0 \times b^0 \tag{1.4}$$

ou

$$N = \sum_{j=0}^{n-1} p_j \times b^j \tag{1.5}$$

em que p_j é o algarismo que representa o peso da potência j da base. O número de algarismos necessários é b e é usual que os algarismos sejam os inteiros entre 0 e $b-1$:

$$p_j \in \{0, 1, \ldots, b-1\} \tag{1.6}$$

Assim, para a representação de números numa base b não podem ser utilizados algarismos de valor igual ou superior a b. Por exemplo, a representação de um número em base 7 não pode utilizar o algarismo 7 nem nenhum outro superior a 7. A sequência de dígitos 1742_7 não é assim uma representação válida de um número.

A *conversão* da representação de um número em base b para a representação em base 10 não oferece qualquer dificuldade, como se viu, na Equação 1.3. A conversão inversa, de um número representado em base 10 para a sua representação em base b é um pouco mais trabalhosa, mas é igualmente simples. Um dos métodos mais comuns é o *método das divisões sucessivas*. Considere-se, então, um número N representado em base b, de acordo com a Equação 1.4. Se se dividir o número por b obtém-se

$$\begin{aligned} \frac{N}{b} &= (p_{n-1} \times b^{n-2} + p_{n-2} \times b^{n-3} + \cdots + p_1 \times b^0) + \frac{p_0 \times b^0}{b} \\ &= (p_{n-1} \times b^{n-2} + p_{n-2} \times b^{n-3} + \cdots + p_1 \times b^0) + \frac{p_0}{b} \end{aligned} \tag{1.7}$$

em que p_0 é o resto da divisão de N por b (recorde-se que $p_0 < b$). Deste modo identifica-se o algarismo p_0 da representação do número em base b.

A repetição do procedimento anterior para o número $\frac{N}{b}$ permitirá obter p_1 e pela aplicação sucessiva do procedimento obter-se-ão todos os algarismos da representação do número.

Considere-se, a título de exemplo, a obtenção da representação em base 5 do número 273_{10}:

$$\frac{273}{5} = 54 + \frac{3}{5} \tag{1.8}$$

De igual modo se obtém

$$\begin{aligned} \frac{54}{5} &= 10 + \frac{4}{5} \\ \frac{10}{5} &= 2 + \frac{0}{5} \\ \frac{2}{5} &= 0 + \frac{2}{5} \end{aligned} \tag{1.9}$$

Das Equações 1.8 e 1.9 é fácil obter $p_0 = 3$, $p_1 = 4$, $p_2 = 0$ e $p_3 = 2$. Donde,

$$273_{10} = 2043_5 \tag{1.10}$$

1.1.2 Representação de Números em Base 2

A representação de números em *base* 2 é importante porque, para a utilização de computadores e outros sistemas digitais, a representação dos números terá de ser baseada num

conjunto de dois valores diferentes para uma determinada grandeza física. Em computadores digitais, essa grandeza física é habitualmente a tensão eléctrica entre dois pontos de um circuito electrónico.

Para a representação de um número inteiro em base 2, são necessários, naturalmente, 2 algarismos, usualmente designados por 0 e 1. Tal como para outras bases, um número inteiro é, portanto, representado por uma sequência de algarismos, neste caso, *algarismos binários* ou *bits* (do inglês, *Binary Digit*). Na Tabela 1.1 encontram-se os números inteiros de 0 a 15 representados em base 2.

TABELA 1.1: Representação dos inteiros de 0 a 15 em base 2.

Base 10	Base 2	Base 10	Base 2
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

Por exemplo 110101_2 é um número representado em base 2 ou, como também se diz, *representado em binário*.

Na secção anterior apresentou-se uma metodologia para, a partir de um número em binário (ou representado em qualquer outra base), obter a representação do mesmo número em base 10, uma vez que é essa a base habitual de representação.

A técnica utilizada consistiu em explicitar a representação do número em termos de somas pesadas de potências da base e calcular em base 10 o valor do número. Esta é uma técnica de aplicação geral que pode, portanto, ser também aplicada no caso de números representados em base 2. No caso do número 110101_2 acima referido, considerando que se trata de um número de 6 algarismos, virá

$$\begin{aligned} 110101_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 16 + 4 + 1 \\ &= 53_{10} \end{aligned} \tag{1.11}$$

O problema inverso, a determinação da representação de um número em base 2 (ou noutra base b qualquer) dada a sua representação em base 10 foi igualmente tratado, tendo sido apresentado o método das divisões sucessivas. Usando, agora, como exemplo o número 26_{10} note-se que

$$26 = 13 \times 2 + 0 \tag{1.12}$$

explicitando o quociente e o resto da divisão do número por 2.

O número 13 é, por sua vez, representável como $13 = 6 \times 2 + 1$, pelo que substituindo na Equação 1.12, se obtém

$$\begin{aligned} 26 &= (6 \times 2 + 1) \times 2 + 0 \\ &= 6 \times 2^2 + 1 \times 2 + 0 \end{aligned} \tag{1.13}$$

Considerando agora que $6 = 3 \times 2 + 0$, resulta

$$\begin{aligned} 26 &= (3 \times 2) \times 2^2 + 1 \times 2 + 0 \\ &= 3 \times 2^3 + 1 \times 2 + 0 \end{aligned} \tag{1.14}$$

E, como $3 = 1 \times 2 + 1$, vem

$$\begin{aligned} 26 &= (1 \times 2 + 1) \times 2^3 + 1 \times 2 + 0 \\ &= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2 + 0 \end{aligned} \tag{1.15}$$

Representando, por fim, explicitamente todas as potências de 2, vem

$$26 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \tag{1.16}$$

É agora fácil ver que o número 26_{10} se representa em base 2 por 11010_2. Os diversos algarismos binários são, como se viu, os sucessivos restos da divisão por 2 do número inicial e dos sucessivos quocientes. A forma mais habitual (e rápida) de realizar os cálculos é, porém, a da aplicação sucessiva do modo habitual de realizar o algoritmo da divisão recolhendo-se, no final, os diversos restos:

26	2				
0	13	2			
	1	6	2		
		0	3	2	
			1	1	2
				1	0

O algarismo de maior peso corresponde ao resto da última divisão e sucessivamente até ao algarismo de menor peso, que é o resto da primeira divisão.

1.1.3 Representação de Números Fraccionários em Base 2

A *conversão de números fraccionários*, representados em base 2 (ou em qualquer outra), para base 10 não apresenta quaisquer problemas, utilizando-se a mesma metodologia que se utiliza para inteiros. Um número fraccionário pode ter uma parte inteira e uma parte decimal, isto é, uma parte menor que 1. Essa parte decimal, com n algarismos, é representável pela seguinte expressão:

$$N = p_{-1} \times b^{-1} + p_{-2} \times b^{-2} + \cdots + p_{-n} \times b^{-n} \tag{1.17}$$

ou

$$N = \sum_{i=-1}^{-n} p_i \times b^i \tag{1.18}$$

Considere-se, a título de exemplo, o número $0,1011010_2$. A conversão é imediata:

$$\begin{aligned} 0,1011010_2 &= 1 \times 2^{-1} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-6} \\ &= 0{,}5 + 0{,}125 + 0{,}0625 + 0{,}015625 \\ &= 0{,}703125_{10} \end{aligned} \tag{1.19}$$

A conversão de um número fraccionário entre uma base b e a base 10 é feita, portanto, usando o mesmo algoritmo que no caso de números inteiros, tendo o cuidado de não acrescentar através do processo de conversão maior precisão ao número de que se partiu, uma vez que isso não teria qualquer significado.

De facto, o número $0{,}1011010_2$ é representado com 7 algarismos binários após a vírgula. Isso significa que está representado com uma precisão de 1 em 2^7, uma vez que com esses 7 algarismos, é possível representar 2^7 números diferentes. O número que se obtém por conversão da base representa a mesma grandeza e, portanto, o número de algarismos deve ser tal que a precisão não ultrapasse a da representação inicial. Teremos, assim, de escolher um número p de algarismos, tais que p seja o maior inteiro que verifique a equação

$$10^p \leq 2^7 \tag{1.20}$$

e venha, portanto,

$$p = \lfloor \log_{10} 2^7 \rfloor \tag{1.21}$$

Da Expressão 1.21[1] resulta $p = 2$. Assim, a representação correcta em base 10 do número $0{,}1011010_2$ é $0{,}70_{10}$, que se obtém por arredondamento do resultado obtido na Expressão 1.19

[1] A função $\lfloor x \rfloor$ (em inglês, *floor*, devolve, para o número real x, o maior inteiro menor ou igual a x

O problema inverso de conversão de um número fraccionário representado em base 10 para uma base qualquer, nomeadamente a base 2, utiliza, tal como no caso dos números inteiros, um algoritmo mais complexo, se não se pretender realizar as operações aritméticas na base para onde se converte. O procedimento a utilizar será apresentado de seguida, usando um exemplo. Considere-se, assim, o número $0{,}627_{10}$. Este número, quando representado em base 2, será

$$0{,}627_{10} = 0{,}p_{-1}p_{-2}p_{-3}p_{-4}\ldots p_{-n_2} \tag{1.22}$$

Pretende-se obter o valor dos sucessivos algarismos $p_{-1}, p_{-2}, p_{-3}, p_{-4}$ e p_{-n}. Multiplicando ambos os membros da Equação 1.22 por 2, resulta:

$$1{,}354_{10} = p_{-1}{,}p_{-2}p_{-3}p_{-4}\ldots p_{-n_2} \tag{1.23}$$

De facto, a multiplicação por 2 em base 2 corresponde ao deslocamento de todos os algarismos do número de uma posição para a esquerda, tal como a multiplicação por 10 em base 10. Da análise da Equação 1.23 e notando que as partes inteiras dos dois membros da equação têm de ser iguais, tal como as partes fraccionárias vem, portanto,

$$1 = p_{-1} \tag{1.24}$$

e

$$0{,}354_{10} = 0{,}p_{-2}p_{-3}p_{-4}\ldots p_{-n_2} \tag{1.25}$$

A Equação 1.24 permite identificar o algarismo p_{-1}. Para o segundo algarismo, p_{-2}, aplica-se agora o mesmo algoritmo à parte fraccionária resultante, representada na Expressão 1.25. A aplicação sucessiva da multiplicação por 2 permite obter a sucessão de algarismos da parte fraccionária do número. É óbvio que, tal como na conversão inversa, se deve utilizar, na base para onde se converte, o número máximo de algarismos que não aumente a precisão do número em relação à sua representação original. No exemplo acima, para não ultrapassar a precisão inicial ($\frac{1}{10^3}$) serão utilizados 9 algarismos na representação em base 2 ($2^9 = 512$ e $2^{10} = 1024$). Completando o exemplo:

$$0{,}627_{10} = 0{,}101011010_2 \tag{1.26}$$

Quando se pretender fazer a conversão da representação de números com parte inteira e fraccionária entre duas bases, procede-se à conversão separadamente da parte inteira e da parte fraccionária usando os respectivos algoritmos, somando-se no final os dois números obtidos.

1.1.4 Representação de Números em Bases Potências de 2

A representação de números em base 2 é necessária, pois é a utilizada pelos sistemas digitais para representar números internamente, mas tem a grande desvantagem de utilizar sequências de algarismos relativamente longas. Por exemplo, $153{,}845_{10}$ representa-se em base 2 por $10011001{,}110110001_2$. Estas representações tornam-se de difícil percepção e manipulação. A representação em base 10 não tem esses inconvenientes, mas não pode ser utilizada para representação interna em sistemas digitais. A solução para este dilema é a utilização de formas condensadas da representação binária, que se tornam possíveis representando o número em bases que são uma potência de 2, isto é, $4, 8, 16, \ldots$ Normalmente utiliza-se a base 8 ou, mais frequentemente, a 16. Estas representações permitem, como se verá, a representação condensada e facilmente convertível de números binários com muitos dígitos.

A base 8 utiliza os algarismos de 0 a 7, e a conversão de números entre a base 8 e a base 10 utiliza, naturalmente, os procedimentos atrás descritos. Um número representado em base 8 diz-se, também, representado em *octal*. A representação de números em base 16, ou em *hexadecimal*, como é hábito designar este tipo de representação, é semelhante a qualquer outra base, sendo apenas necessário ter em conta que existem 16 algarismos, de 0 a 15. No caso dos algarismos que representam $10, 11, 12, 13, 14$ e 15, usam-se habitualmente, as letras maiúsculas de A a F. Os 16 algarismos estão listados na Tabela 1.2:

Tabela 1.2: Algarismos da base 16.

Número	Algarismo	Número	Algarismo
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

O número 4A6F_{16} tem portanto em decimal a seguinte representação:

$$\begin{aligned} 4\text{A6F}_{16} &= 4 \times 16^3 + 10 \times 16^2 + 6 \times 16 + 15 \\ &= 4 \times 4096 + 10 \times 256 + 6 \times 16 + 15 \\ &= 19055_{10} \end{aligned} \tag{1.27}$$

Do mesmo modo, em base 8 o número 3605_8 é

$$\begin{aligned} 3605_8 &= 3 \times 8^3 + 6 \times 8^2 + 5 \\ &= 3 \times 512 + 6 \times 64 + 5 \\ &= 1925_{10} \end{aligned} \tag{1.28}$$

A conversão entre bases em que uma é uma potência da outra faz-se de um modo muito fácil. Considere-se, com algum pormenor, a conversão do número 101101110101_2 para a base 16. Começa-se por representar o número, explicitando a sua expressão em termos das potências da base, à semelhança do que já atrás se fez:

$$\begin{aligned} 101101110101_2 = {} & 1 \times 2^{11} + 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 0 \times 2^7 + \\ & +1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + \\ & +0 \times 2^1 + 1 \times 2^0 \end{aligned} \tag{1.29}$$

Seguidamente agrupam-se os termos em grupos de 4 a partir dos menos significativos:

$$\begin{aligned} 101101110101_2 = {} & (1 \times 2^{11} + 0 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8) + \\ & +(0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4) + \\ & +(0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) \end{aligned} \tag{1.30}$$

Colocam-se em evidência em cada grupo as potências de 2 necessárias para deixar cada grupo representado em termos de soma pesada das potências de ordem $0, 1, 2$ e 3.

$$\begin{aligned} 101101110101_2 = {} & (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \times 2^8 + \\ & +(0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \times 2^4 + \\ & +(0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) \times 2^0 \end{aligned} \tag{1.31}$$

Mas $2^8 = 16^2, 2^4 = 16^1$ e $2^0 = 16^0$ são potências de 16. Pode-se portanto escrever:

$$101101110101_2 = 11 \times 16^2 + 7 \times 16^1 + 5 \times 16^0 \tag{1.32}$$

É agora fácil concluir que

$$101101110101_2 = \mathrm{B75}_{16} \tag{1.33}$$

Repare-se que cada algarismo em base 16 foi definido pelos 4 algarismos binários que inicialmente foram agrupados. Então, se se conhecer a equivalência entre os algarismos da base 16 e a sua correspondência num número binário de 4 algarismos, pode

TABELA 1.3: Correspondência entre os algarismos da base 16 e a sua representação em binário com 4 algarismos.

Número	Algarismo	Número	Algarismo
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

determinar-se directamente o número em base 16 a partir do número em base 2. Essa correspondência pode ser facilmente obtida e está representada na Tabela 1.3

Se se isolarem os grupos de 4 algarismos binários do número $1011\ \ 0111\ \ 0101$, pode este ser assim convertido directamente para a base 16

$$\begin{aligned} 1011 &\Leftrightarrow \mathrm{B} \\ 0111 &\Leftrightarrow 7 \\ 0101 &\Leftrightarrow 5 \end{aligned}$$

obtendo-se o número $\mathrm{B75}_{16}$, como se viu. É de notar que os grupos de 4 algarismos são formados a partir do algarismo menos significativo, isto é, da direita. Tal decorre, como se pode deduzir facilmente, do modo como o algoritmo foi definido.

No caso de um número em binário ter um número de bits que não é múltiplo de 4, o método aplica-se exactamente da mesma maneira. Considere-se o número 1101011011_2. Isolando os algarismos em grupos de 4 a partir do menos significativo, obtém-se

$$\begin{aligned} 1101011011_2 &= 11\ \ 0101\ \ 1011_2 \\ &= 3\ \ 5\ \ \mathrm{B}_{16} \\ &= 35\mathrm{B}_{16} \end{aligned} \tag{1.34}$$

No caso de outra base potência de 2, como por exemplo a base 8, o método é semelhante, variando apenas o número de algarismos que se agrupam. No caso da base 8, os

algarismos juntam-se em grupos de 3 ($2^3 = 8$). Exemplificando:

$$\begin{aligned} 10010110101001_2 &= 10\ 010\ 110\ 101\ 001_2 \\ &= 2\ 2\ 6\ 5\ 1_8 \\ &= 22651_8 \end{aligned} \tag{1.35}$$

Do mesmo modo, a passagem de uma base potência de 2 para a base 2 realiza-se pelo processo inverso:

$$\begin{aligned} 7\text{DA3F}_{16} &= 7\ \text{D}\ \text{A}\ 3\ \text{F}_{16} \\ &= 0111\ 1101\ 1010\ 0011\ 1111_2 \\ &= 01111101101000111111_2 \end{aligned} \tag{1.36}$$

Exemplo semelhante pode ser considerado para base 8:

$$\begin{aligned} 3461_8 &= 3\ 4\ 6\ 1_8 \\ &= 011\ 100\ 110\ 001_2 \\ &= 11100110001_2 \end{aligned} \tag{1.37}$$

Uma maneira alternativa, muitas vezes utilizada, de indicar que um número está representado em base 16 consiste em terminar o número com a letra *h*. Assim, por exemplo, escrever 4871_{16} é o mesmo que escrever 4871h. Do mesmo modo é habitual usar as letras b, o e d para indicar que um número está representado, respectivamente em binário, octal ou decimal. Por exemplo, 1110101_2 pode ser representado por 1110101b. Este modo de representação será a partir daqui o preferido quando for necessário indicar a base em que um número está representado.

O uso da representação de números em bases potências de 2, nomeadamente nas bases 8 e 16, permite, assim, a representação em forma compacta de números binários com muitos dígitos. Existe, para além disso, como se viu, um modo muito simples de converter os números entre a base 2 e essas bases, que é praticamente independente em complexidade do número de algarismos da representação dos números. Por fim, permite conversões parciais de zonas do número que podem interessar. É fácil conhecer, por exemplo, os dígitos binários menos ou mais significativos de um número representado numa dessas bases, sem a necessidade da conversão completa. Por exemplo, no número A23Bh é fácil verificar que os 5 algarismos mais significativos em binário são 10100.

A conversão de números com parte fraccionária entre a base 2 e bases potências de 2 realiza-se exactamente do mesmo modo, tendo o cuidado de agrupar os algarismos binários a partir da vírgula, como pode ser facilmente concluído. Considere-se, a título

de exemplo, o número 1001010,101100111111b:

$$\begin{aligned} 1001010{,}101000111111\text{b} &= 100\ 1010,\ 1011\ 0011\ 1111\text{b} \\ &= 4\ \text{A},\ \text{C}\ 3\ \text{Fh} \\ &= 4\text{A,C3Fh} \end{aligned} \tag{1.38}$$

Do mesmo modo, a passagem de base potência de 2 para a base 2 não oferece qualquer dificuldade para números com parte fraccionária:

$$\begin{aligned} 5271{,}3527\text{o} &= 5\ 2\ 7\ 1,\ 3\ 5\ 2\ 7\text{o} \\ &= 101\ 010\ 111\ 001,\ 011\ 101\ 010\ 111\text{b} \\ &= 101010111001{,}011101010111\text{b} \end{aligned} \tag{1.39}$$

1.2 OPERAÇÕES ARITMÉTICAS EM BASES 2, 8 E 16

Nesta secção refere-se brevemente a aritmética elementar nas bases 2, 8 e 16. Limita-se a abordagem a somas e produtos de números inteiros positivos. O estudo da aritmética abordando inteiros negativos será feito mais tarde, no Capítulo 5. Aí será considerada também a subtracção.

1.2.1 Somas em Base 2

A *soma em base* 2, tal como em qualquer outra base, não é fundamentalmente diferente da soma em base 10. O procedimento adoptado assenta na existência de uma tabuada da soma e na metodologia de somar números algarismo a algarismo. A soma é efectuada, somando, para cada ordem a começar pelo algarismo menos significativo, os algarismos dos números a somar com o *transporte* da ordem anterior. A soma deve também gerar o transporte para a ordem seguinte. A tabuada da soma em base 2 (Tabela 1.4) é particularmente simples.

Tabela 1.4: Tabuada da soma em base 2.

$X+Y$		Y	
		0	1
X	0	0	1
	1	1	10

Repare-se que, na soma $1 + 1$, o resultado já não é representável num único algarismo, sendo, portanto, necessário utilizar dois algarismos. Isso significa que, no algoritmo da soma, surgirá, neste caso, um transporte de 1. Nos restantes casos da soma, o transporte é sempre 0.

Considere-se, a título exemplificativo, a soma de 10001111_2 com 1011010_2. Nos algarismos menos significativos, isto é, de ordem 0, na coluna mais à direita, a soma não tem de ter em conta o transporte da coluna anterior. A soma de 1 com 0 é, de acordo com a Tabela 1.4, 1, e o transporte, 0:

$$\begin{array}{r} 10001111 \\ +1011010 \\ \hline 1 \\ 0 \quad \text{transporte} \end{array}$$

Na segunda coluna, somando os algarismos de ordem 1, a soma de 1 com 1 tem como resultado 10 (isto é, 2_{10}). Como o transporte da coluna anterior é 0, a soma não se altera. Mas 10 é um número de dois algarismos, donde, nesta coluna, o resultado da soma é 0, e o transporte, 1:

$$\begin{array}{r} 10001111 \\ +1011010 \\ \hline 01 \\ 10 \quad \text{transporte} \end{array}$$

Na coluna seguinte, somando 1 com 0, obtém-se 1. Como o transporte é 1, é necessário somar ainda esse 1, obtendo-se 10. Logo, a soma é 0, e o transporte é 1:

$$\begin{array}{r} 10001111 \\ +1011010 \\ \hline 001 \\ 110 \quad \text{transporte} \end{array}$$

Na coluna de ordem 3, temos a soma de 1 com 1, o que resulta em 10, somado ainda com um transporte de 1, sendo o resultado 11. Neste caso, a soma é 1, tal como o transporte:

$$\begin{array}{r} 10001111 \\ +1011010 \\ \hline 1001 \\ 1110 \quad \text{transporte} \end{array}$$

Continuando o raciocínio, obter-se-á, por fim,

$$\begin{array}{r} 10001111 \\ +1011010 \\ \hline 11101001 \\ 00011110 \quad \text{transporte} \end{array}$$

Claro que é possível somar mais de dois números. O único aspecto a ter em conta é que, agora, o transporte pode não se limitar a um algarismo. Considere-se a soma de 4 números adiante indicada:

$$\begin{array}{r} 10011101 \\ 101011 \\ 11001 \\ +1011011 \\ \hline \end{array}$$

Na primeira coluna, a soma dá 100_2. Aqui, a soma é, naturalmente, 0, e o transporte, 10. Tendo em conta este aspecto, o resultado final será:

$$\begin{array}{r} 10011101 \\ 101011 \\ 11001 \\ +\ 1011011 \\ \hline 100111100 \end{array}$$

Normalmente, nos sistemas digitais, as somas são realizadas entre dois números, sendo as somas de mais de dois números realizadas por somas sucessivas das diversas parcelas a somar.

1.2.2 Multiplicações em Base 2

Tal como no caso da adição, a *multiplicação em base* 2 segue os mesmos métodos da multiplicação em base 10. A tabuada da multiplicação é representada na Tabela 1.5.

Cada algarismo do multiplicador é multiplicado pelo multiplicando, gerando um produto parcial. A soma de todos os produtos parciais é o produto dos dois números. De facto, o produto $M \times N$ pode ser explicitado, se representarmos o multiplicador N em

TABELA 1.5: Tabuada do produto em base 2.

$X \times Y$		Y	
		0	1
X	0	0	0
	1	0	1

termos da sua representação em base 2 (o mesmo se poderia fazer para qualquer base) pela Equação 1.40.

$$\begin{aligned} M \times N &= M \times (p_{n-1} \times 2^{n-1} + p_{n-2} \times 2^{n-2} + \cdots + p_1 \times 2^1 + p_0 \times 2^0) \\ &= M \times p_{n-1} \times 2^{n-1} + M \times p_{n-2} \times 2^{n-2} + \cdots + M \times p_1 \times 2^1 + \\ &\quad + M \times p_0 \times 2^0 \end{aligned} \tag{1.40}$$

Por exemplo, se se pretender multiplicar $M = 10001111$ por $N = 1010$, o produto obtém-se somando os quatro produtos parciais

$$\begin{array}{lcl} 10001111 & \times & 1000 \\ 10001111 & \times & 0 \\ 10001111 & \times & 10 \\ 10001111 & \times & 0 \end{array}$$

correspondentes aos quatro algarimos do multiplicador.

No algoritmo corrente, é fundamental, portanto, colocar os vários produtos parciais alinhados com o respectivo algarismo do multiplicador:

$$\begin{array}{rl} 10001111 & \\ \times\ 1010 & \\ \hline 00000000 & (= 10001111 \times 0) \\ 10001111 & (= 10001111 \times 10) \\ 00000000 & (= 10001111 \times 0) \\ 10001111 & (= 10001111 \times 1000) \\ \hline 10110010110 & \end{array}$$

Naturalmente que, tal como no caso da multiplicação em base 10, se podem omitir os resultados do produto do multiplicando por algarismos 0 do multiplicador:

$$
\begin{array}{r}
10001111 \\
\times\ 1010 \\
\hline
10001111 \\
10001111 \\
\hline
10110010110
\end{array}
$$

Repare-se que, em base 2, só há duas hipóteses para os produtos parciais: ou o algarismo do multiplicador é 0, e o produto parcial é 0, ou esse algarismo é 1, e o produto parcial é igual ao multiplicando com o deslocamento correspondente ao peso do algarismo. Ver-se-á, adiante, que esta característica é importante na realização desta operação por sistemas digitais.

1.2.3 Operações Aritméticas em Outras Bases

A realização de operações em outras bases não levanta quaisquer problemas para além de conhecer a tabuada das operações nessas bases. Exemplificando para a *soma em base* 16, pode construir-se a tabuada da soma (Tabela 1.6).

Tabela 1.6: Tabuada da soma em base 16.

$X+Y$		Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
X	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
	2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
	3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
	4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
	5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
	6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
	7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
	8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
	9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
	A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
	B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
	C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
	D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
	E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
	F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

A construção desta tabela pode ser feita de diversas formas. Uma delas é considerar, para cada soma, o valor dos algarismos, somá-los em decimal e transferir o resultado para hexadecimal. Exemplificando com a soma $5 + \mathrm{D}$, tem-se a soma de 5 com 13 (o valor do algarismo D). O resultado desta soma é 18d ou, em hexadecimal, 12h, que é o valor a colocar na tabuada.

A título de exemplo considere-se a soma de $1\mathrm{F}3\mathrm{A}5$h com $\mathrm{A}542$h:

$$\begin{array}{r} 1\mathrm{F}3\mathrm{A}5 \\ +\mathrm{A}542 \\ \hline 298\mathrm{E}7 \\ 01000 \end{array} \quad \text{transporte}$$

A multiplicação é também de fácil concretização. Exemplifica-se com a tabuada da *multiplicação em base* 8 (Tabela 1.7) e com o produto de dois números nessa base:

Tabela 1.7: Tabuada do produto em base 8.

$X \times Y$		Y							
		0	1	2	3	4	5	6	7
X	0	0	0	0	0	0	0	0	0
	1	0	1	2	3	4	5	6	7
	2	0	2	4	6	10	12	14	16
	3	0	3	6	11	14	17	22	25
	4	0	4	10	14	20	24	30	34
	5	0	5	12	17	24	31	36	43
	6	0	6	14	22	30	36	44	52
	7	0	7	16	25	34	43	52	61

O produto de 1537o por 314o será

$$\begin{array}{r} 1537 \\ \times\ 314 \\ \hline 6574 \\ 1537 \\ 5035 \\ \hline 527664 \end{array}$$

A operação de multiplicação em base b, embora não seja conceptualmente diferente da operação em base 10, obriga a um bom domínio da respectiva tabuada, o que, em geral,

não se verifica. Nessas circunstâncias, é evidente que a realização de multiplicações directamente nas bases consideradas reveste-se de alguma dificuldade. Não é por isso comum a realização deste tipo de operações. A soma, contudo, é muito útil, como se verá mais tarde, no contexto da arquitectura e programação de computadores.

1.3 CÓDIGOS

A representação em base 2 permite a representação de números em sistemas digitais. Nem toda a informação, porém, é numérica. Há muitos outros tipos de dados que necessitam de ser processados e tratados. O texto é um exemplo óbvio, mas está longe de ser único. Nesta secção serão analisados os métodos de representação de informação utilizando códigos. Será dado particular realce aos códigos binário e BCD e a códigos alfanuméricos. A representação de outros tipos de informação está fora do âmbito deste livro.

1.3.1 Codificação

A *representação de informação em sistemas digitais* assenta no facto de os sistemas deste tipo apenas terem como base, para representar seja o que for, as quantidades 0 e 1. Tal como no caso dos algarismos binários, uma entidade que pode assumir dois valores é designada por bit. Se existir uma entidade que gera informação e essa informação se concretizar em sequências de um de entre vários possíveis valores, a solução para a representação desses valores num sistema digital é a de os *codificar*. Isso significa que se faz corresponder, a cada valor possível, uma configuração determinada de bits que passam a representar aquele valor. Considere-se, como exemplo, um elevador num edifício com 6 pisos: duas caves, o r/c e mais 3 andares. Se se pretender registar num sistema digital o andar em que o elevador se encontra ou para onde se dirige, se em movimento, há que codificar essa informação, isto é, fazer corresponder a cada andar uma determinada configuração de bits. O número de bits a usar será, pelo menos, o necessário para realizar 6 combinações. Como facilmente pode ser visto, bastam, portanto, 3 bits para esse código. Um exemplo de codificação está ilustrado na Tabela 1.8.

À correspondência entre as entidades a representar e a sua codificação chama-se *código*. A tabela anterior define, assim, um código. Cada uma das configurações designa-se por *palavra do código*. O número de bits da palavra de código, se for igual para todas as configurações, designa-se por *comprimento do código*. Este código é, portanto, um código de comprimento 3. Como é óbvio, este não é o único código possível para esta aplicação. Na Tabela 1.9 ilustra-se outro código.

TABELA 1.8: Exemplo de código.

Andar	Codificação
2ª cave	000
1ª cave	001
R/C	010
1º andar	011
2º andar	100
3º andar	101

A única restrição para um código válido é a de não haver codificações repetidas, o que significa que, duas entidades diferentes não podem ser codificadas com a mesma palavra de código.

TABELA 1.9: Exemplo alternativo de código.

Andar	Codificação
2ª cave	110
1ª cave	111
R/C	000
1º andar	001
2º andar	010
3º andar	011

Qualquer das codificações é aceitável. Os dois códigos apresentados não têm nenhuma característica que os distinga significativamente um do outro. No entanto, podem ser concebidos códigos com algumas características especiais. Por exemplo, usando agora palavras de 4 bits para representar os vários andares, é possível conceber um código que represente cada andar com uma sequência com dois bits a 0 e dois bits a 1, como se mostra na Tabela 1.10.

Um possível uso para códigos deste tipo é detectar alguns erros de codificação ou de processamento. Se, em determinado momento, o sistema tiver registado que o elevador está no andar representado pela palavra 0010, é imediato que houve um erro de codificação ou processamento, uma vez que a palavra 0010 não pertence ao código.

TABELA 1.10: Exemplo de código com restrições.

Andar	Codificação
2ª cave	0011
1ª cave	0101
R/C	1001
1º andar	0110
2º andar	1010
3º andar	1100

1.3.2 CÓDIGOS NUMÉRICOS

Embora os números sejam habitualmente representados pela sua representação em base 2 (e, de algum modo, isso corresponde também a um código), é muitas vezes necessário utilizar códigos para representar números, nomeadamente números inteiros. Esses códigos denominam-se *códigos numéricos*. A maneira mais simples para representar um inteiro por um código é fazer representar cada número por uma palavra de código que é a sua representação em binário. Na Tabela 1.11 ilustra-se um código deste tipo com palavras de comprimento constante.

Repare-se que todas as palavras de código têm o mesmo comprimento. Por exemplo, o número 3 representa-se por 00011 e não por 11, que seria a sua normal representação em base 2. Este tipo de código em que cada número é codificado pela sua representação em base 2 com um número fixo de bits, denomina-se *código binário natural*. Como é evidente há códigos binários naturais com qualquer número de bits.

Uma situação muito comum, mas de diferente carácter, é a necessidade de codificar os algarismos decimais. Tal necessidade resulta de que, por vezes, é conveniente representar um número não pelo seu código binário natural, mas sim pela codificação em binário de cada um dos seus algarismos em base 10. Uma situação óbvia é a de um mostrador de uma máquina de calcular em que se representa um número em decimal para leitura fácil pelo utilizador, mas que, do ponto de vista da máquina, é representado com bits. A solução clássica é a de representar cada algarismo decimal do número pela sua representação em binário, de acordo com a Tabela 1.12.

Este código chama-se *código BCD* (do inglês, *Binary Coded Decimal*) o que significa, em português, *Decimal Codificado em Binário*. Repare-se que o código BCD tem, para cada algarismo, a mesma representação do código binário natural de 4 bits. Acontece, porém, que nem todas as posições do código têm significado. A indicação de que um

TABELA 1.11: Código binário natural de 5 bits.

Número	Codificação	Número	Codificação
0	00000	16	10000
1	00001	17	10001
2	00010	18	10010
3	00011	19	10011
4	00100	20	10100
5	00101	21	10101
6	00110	22	10110
7	00111	23	10111
8	01000	24	11000
9	01001	25	11001
10	01010	26	11010
11	01011	27	11011
12	01100	28	11100
13	01101	29	11101
14	01110	30	11110
15	01111	31	11111

TABELA 1.12: Código BCD.

Algarismo	Codificação
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

número está representado em BCD faz-se indicando BCD em índice. Por exemplo, 7_{10} é representado por 0111_{BCD} em BCD.

A representação de um número em código BCD faz-se representando cada algarismo

decimal pela sua representação binária, como se ilustra representando o número 2719 em BCD:

$$2719_{10} = 0010011100011001_{\text{BCD}} \tag{1.41}$$

em que 0010 corresponde ao algarismo 2_{10}; 0111, ao 7_{10}; 0001, ao 1_{10}; e 1001, ao 9_{10}.

Repare-se que nem todas as sequências de bits podem corresponder a uma codificação usando o código BCD. Por exemplo, a sequência 10110101 não é um número codificado em BCD, uma vez que os quatro bits da esquerda (1011) não correspondem a qualquer algarismo, conforme se pode ver na Tabela 1.12.

De igual modo é importante referir que uma representação de um número no código BCD não é uma representação desse número em binário. Retomando o exemplo anterior,

$$0010011100011001_{\text{BCD}} = 2719_{10} \tag{1.42}$$

mas a mesma sequência de bits interpretada como um número em base 2 tem outro significado:

$$0010011100011001_2 = 10009_{10} \tag{1.43}$$

o que, claramente, não corresponde ao mesmo número.

1.3.3 Códigos Reflectidos

Um tipo de códigos numéricos particularmente importante são os *códigos reflectidos* (por vezes, referidos como *código de Gray*). Estes códigos têm como característica fundamental o facto de duas palavras que codificam dois números sucessivos terem apenas um bit diferente. Esta característica é importante, como se verá no Capítulo 2.

Na Tabela 1.13 ilustra-se um código reflectido de 3 bits. Repare-se que, com 3 bits, é possível codificar 8 números diferentes.

Como é fácil de ver, neste código, cada número não é, excepto em alguns casos, representado pela sua representação em binário. É possível verificar que entre cada dois números sucessivos se altera apenas um bit da sua representação em binário. Isso acontece de igual modo entre as representações do 0 e do 7, os dois valores extremos representados.

A maneira mais directa de construir o código reflectido de n bits é partir do código de $n-1$ bits. Parte-se das 2^{n-1} configurações do código de $n-1$ bits e repetem-se essas

TABELA 1.13: Código reflectido de 3 bits.

Numero	Codificação
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

TABELA 1.14: Construção do código reflectido de 4 bits.

000
001
011
010
110
111
101
100
100
101
111
110
010
011
001
000

configurações por ordem inversa, isto é, como se estivessem reflectidas num espelho imaginário (daí o nome de código reflectido), como se ilustra na Tabela 1.14.

Seguidamente junta-se o quarto bit da codificação, fazendo-o igual a 0, nas posições iniciais, e igual a 1, nas posições «reflectidas», como se ilustra na Tabela 1.15, onde o código é já representado.

TABELA 1.15: Construção do código reflectido de 4 bits (conclusão).

Número	Codificação
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

A título de exercício sugere-se a obtenção dos códigos reflectidos de 2 e 3 bits, a partir do código de 1 bit. Como é evidente, este último não difere de um código binário natural de 1 bit.

1.3.4 Códigos Alfanuméricos

Um tipo de informação que é importante representar em sistemas digitais é um texto. Por isso, é fundamental utilizar um código para representar todos os caracteres possíveis num texto. Ao longo da evolução dos sistemas digitais foram utilizados vários códigos com essa finalidade. Actualmente, o código mais usado é o *código ASCII* (do inglês, American Standard Code for Information Interchange), que, tendo tido origem nos Estados Unidos da América, rapidamente se tornou no código universalmente aceite. Um código deste tipo tem de codificar as 26 letras do alfabeto em maiúsculas e minúsculas, os 10 algarismos, todos os sinais de pontuação e alguns caracteres especiais. O código tem palavras de 7 bits, o número necessário para codificar todos os caracteres. É usual representar a palavra do código com os bits ordenados de 1 a 7 da seguinte maneira: $b_6b_5b_4b_3b_2b_1b_0$. O código está representado na Tabela 1.16.

TABELA 1.16: Código ASCII.

$b_6b_5b_4$	000	001	010	011	100	101	110	111
$b_3b_2b_1b_0$								
0000	NUL	DLE	SP	0	@	P	‘	p
0001	STH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	$*$	:	J	Z	j	z
1011	VT	ESC	$+$	;	K	[	k	{
1100	FF	FS	,	$<$	L	\	l	|
1101	CR	GS	$-$	$=$	M	]	m	}
1110	SO	RS	.	$>$	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

O carácter «SP» é o espaço.

Os caracteres das duas primeiras colunas e o DEL na última posição da tabela são *caracteres de controlo*. O significado dos caracteres de controlo está ligado, para a sua maioria, a aplicações do código em transmissão de informação entre equipamentos. Em muitos casos esses caracteres já não são utilizados por terem evoluído os protocolos de comunicação.

A representação de um texto em ASCII é feita indicando a sucessão de códigos dos caracteres do texto. Por exemplo, o texto «Sistemas Digitais» será codificado em ASCII como: 1010011 1101001 1110011 1110100 1100101 1101101 1100001 1110011 0100000 1000100 1101001 1100111 1101001 1110100 1100001 1101001 1110011 .

Torna-se conveniente representar as configurações do código ASCII em hexadecimal. Neste caso, o texto acima seria, portanto, codificado como: 53h 69h 73h 74h 65h 6Dh 61h 73h 20h 44h 69h 67h 69h 74h 61h 69h 73h.

A sequência ASCII 4Fh 73h 20h 73h 49h 73h 74h 65h 6Dh 61h 73h 20h 64h 49h 67h 69h 74h 61h 69h 73h 20h 75h 73h 61h 6Dh 20h 41h 53h 43h 49h 49h 20h 70h 61h 72h 61h 63h 6Fh 64h 49h 66h 69h 63h 61h 72h 20h 63h 61h 72h 61h 63h 74h 65h 72h 65h 73h 2Eh codifica a seguinte frase: «Os sistemas digitais usam ASCII para representar caracteres.».

Por vezes, a designação dos códigos ASCII dos caracteres faz-se em decimal e não em hexadecimal. Por exemplo, a letra «A», que tem o código 41h em hexadecimal, tem, em decimal, o código 65d.

O código ASCII permite representar um texto, mas tem algumas limitações. Por um lado, não permite representar caracteres especiais específicos de uma língua, como por exemplo o «ç» em português, o «ø» em sueco ou o «ß» em alemão. Por outro, não permite acentuar caracteres. Para além disso, não são representados outros alfabetos. Por isso, foram desenvolvidas várias extensões para representar os caracteres específicos e os caracteres acentuados. A extensão foi feita passando o código a 8 bits, mantendo compatibilidade com o ASCII de base. Infelizmente existem actualmente vários códigos incompatíveis entre si. Em todos eles, porém, para os valores 00h a 7Fh, a representação é igual à do ASCII clássico, de modo a manter compatibilidade entre essas extensões e o ASCII. Por exemplo, o código *ISO-8859-1*, a extensão normalizada pela ISO (do inglês, *International Standard Organization*), Organização Internacional de Normalização, para alfabetos de países da Europa Ocidental, tem para os caracteres codificados de 80h a FFh a correspondência ilustrada na Tabela 1.17.

O carácter «nbsp» é um espaço especial que inibe a quebra de uma linha de texto por um processador de texto ou um *browser*.

A norma ISO-8859 tem outras subnormas que definem códigos para caracteres de outras zonas ou grupos linguísticos.

Mais recentemente surgiu o código UNICODE, que pretende codificar todos os caracteres de todas as línguas. Cada palavra do código tem 16 bits e, se necessário, utiliza uma extensão de 20 bits. O ASCII normal e a extensão ISO-8859-1, apresentada acima, têm compatibilidade com o UNICODE. Este código foi já recomendado como suporte para todos os futuros protocolos da Internet.

1.4 REPRESENTAÇÃO DIGITAL DA INFORMAÇÃO

De um ponto de vista abstracto, a informação digital pode ser representada usando dois valores, habitualmente designados por 0 e 1. Como também já foi referido, uma

TABELA 1.17: Código ISO-8859-1 (com $b_7 = 1$).

$b_7b_6b_5b_4$	1000	1001	1010	1011	1100	1101	1110	1111
$b_3b_2b_1b_0$								
0000			nbsp	°	À	Ð	à	ð
0001			¡	±	Á	Ñ	á	ñ
0010			¢	^{2}	Â	Ò	â	ò
0011			£	^{3}	Ã	Ó	ã	ó
0100			€	´	Ä	Ô	ä	ô
0101			¥	μ	Å	Õ	å	õ
0110			|	¶	Æ	Ö	œ	ö
0111			§	·	Ç	×	ç	÷
1000			¨	¸	È	Ø	è	ø
1001			©	^{1}	É	Ù	é	ù
1010			ł	ž	Ê	Ú	ê	ú
1011			≪	≫	Ë	Û	ë	û
1100			¬	$^1/_4$	Ì	Ü	ì	ü
1101			-	$^1/_2$	Í	Ý	í	ý
1110			®	$^3/_4$	Î	Þ	î	þ
1111			¯	¿	Ï	ß	ï	ÿ

entidade desse tipo tem a designação de bit. Como é evidente, um bit — que só pode assumir um de dois valores — não pode, por si só, dar suporte ao tipo de informação que se referiu atrás. É habitual, para representar informação, organizar os bits em unidades de maior capacidade.

Se se considerar, por exemplo, informação alfanumérica representada utilizando o código ISO-8859-1, cada carácter, que é a unidade mínima útil de informação, ocupa 8 bits. Esses 8 bits, sendo uma unidade básica de informação, se se estiver a tratar de texto, devem ser processados, transferidos e armazenados em conjunto. Um conjunto de 8 bits é designado por *byte*, ou *octeto*.

Um byte tem essa designação, independentemente do significado que se atribui à informação que veicula. É indiferente se um byte representa um carácter, dois algarismos codificados em BCD, a cor de um pixel da imagem no monitor de um computador, ou qualquer outro tipo de informação. O conceito de byte está ligado ao conjunto de 8 bits interpretados como uma entidade coerente.

Do mesmo modo, se define o *nibble*, como um conjunto de 4 bits que é interpretado como uma unidade coerente. Um algarismo codificado em BCD, por exemplo, é representado

por um *nibble*. A unidade básica processada por um Intel 4004, um dos primeiros microprocessadores, que processava conjuntos de 4 bits, é, também um *nibble*.

Como é evidente, o conjunto de dois *nibbles*, se considerado como uma unidade, é um byte.

Uma outra entidade que convém desde já considerar é a *palavra* (*word*, em inglês). A palavra é a unidade mínima processada ou armazenada num sistema. Por exemplo, o Intel 4004 de que se falou atrás é um processador com palavras de 4 bits. O Intel 8080 ou o Motorola 6800, por exemplo, que processavam bytes, tinham palavras de 8 bits. O Intel 8086 ou o Motorola 68000 ou o processador P3, apresentado mais à frente neste livro, processam palavras de 16 bits. Há computadores com palavras de outras dimensões. Ao contrário dos conceitos de byte ou *nibble*, portanto, o conceito de palavra não está ligado a uma dimensão fixa. O número de bits de uma palavra depende do contexto que se considerar. No entanto, é comum admitir o comprimento de 16 bits quando se refere uma palavra, a não ser que explicitamente seja especificada outra dimensão.

Nenhum texto, imagem, filme ou som, não trivial, pode ser representado por um byte ou uma palavra. As aplicações úteis de sistemas digitais obrigam, na esmagadora maioria dos casos, a armazenar grandes quantidades de informação em conjuntos de bytes ou palavras.

Quando se consideram grandes quantidades de objectos ou entidades abstractas é usual a referência a centenas, milhares ou milhões de unidades. Essa referência assume, como é evidente, que a base de representação dos números com que se trabalha é a base 10, em que conceitos como milhar e milhão são naturais por serem potências da base. Nos sistemas digitais, porém, a base normal de trabalho para representar números é a base 2, por condicionantes tecnológicas. Em base 2, também as dimensões são habitualmente representadas como potências da base.

Em base 10, como se sabe dos sistemas de unidades de medida, a representação de números relativamente elevados pode ser feita recorrendo a múltiplos das unidades. Por exemplo, em vez de se falar de 1000 m é habitual falar de 1 km. O «k» representa aqui $1000\times$. Em base 2, $1000\text{d} = 1111101000\text{b}$. A filosofia que esteve na base da representação de múltiplos em base 10 pode ser aplicada em qualquer base, mas com diferentes valores. Esses valores devem ser, tal como em base 10, potências da base. Mas convinha ter em base 2 múltiplos «úteis», isto é, múltiplos que não se afastassem do que é habitual em base 10 para manter aquilo que é o hábito cultural dos seres humanos.

A potência de 2 mais perto de 1000d é

$$2^{10} = 10000000000\text{b} = 1024\text{d} \tag{1.44}$$

Por isso, convencionou-se que, em sistemas digitais de todos os tipos (incluindo computadores), 1k $= 2^{10} =$ 1024d. Assim, 1 kbyte significa exactamente 1024 bytes, mas pode ser tomado com pequeno erro por 1 milhar de bytes.

Os *múltiplos* habitualmente usados em sistemas digitais são os descritos na Tabela 1.18.

TABELA 1.18: Múltiplos em base 2.

Múltiplo	Potência	Relação com múltiplo inferior	Representação em base 10	Denominação
1k	2^{10}		1 024d	Quilo
1M	2^{20}	$= 2^{10}$k	1 048 576d	Mega
1G	2^{30}	$= 2^{10}$M	1 073 741 824d	Giga
1T	2^{40}	$= 2^{10}$G	1 099 511 627 776d	Tera

SUMÁRIO

Neste capítulo foram apresentadas diversas maneiras de representar informação em sistemas digitais.

O primeiro aspecto abordado foi o da representação de números inteiros e fraccionários não negativos, sendo apresentada a representação de números em binário e os processos de conversão entre a representação em binário e a representação em decimal. No sentido de facilitar a representação e manipulação de números binários com número elevado de algarismos foram ainda estudadas as representações em octal e hexadecimal.

A problemática da representação de números incluiu ainda a ilustração da utilização das regras habituais da aritmética para a realização de operações directamente em binário, octal ou hexadecimal.

A representação de informação de conteúdo não numérico foi também abordada, tendo sido introduzida a utilização de códigos. Para além de códigos numéricos e alfanuméricos, foram ainda introduzidos os códigos reflectidos, de grande importância no estudo dos métodos de simplificação de funções lógicas apresentados no capítulo seguinte.

O capítulo termina com algumas definições relacionadas com aspectos básicos de representação de informação.

EXERCÍCIOS

1.1 Represente os seguintes números em base 2:

1. 33_{10}.
2. 162_{10}.

1.2 Considere os seguintes números e represente-os em base 2, tendo o cuidado de não exceder a precisão representada nos números originais:

1. $28{,}54_{10}$.
2. $28{,}540_{10}$.
3. $143{,}7_{10}$.
4. $143{,}70_{10}$.

1.3 Represente os seguintes números nas bases 8, 10 e 16:

1. 10101101011_2.
2. $110000001{,}1101_2$.
3. $11111000{,}0011_2$.

1.4 Represente os seguintes números nas bases 2 e 8:

1. $\mathrm{A57D}_{16}$.
2. CAFE_{16}.
3. $17\mathrm{D}, \mathrm{A8}_{16}$.

1.5 Considere o número 35_x. Sabe-se que este número é inferior a 64_{10}. Que valores pode x assumir?

1.6 Os números são habitualmente representados em bases n, com n positivo. Tal não é, porém, estritamente necessário. Considere-se, por exemplo, a base -2. Nada impede a representação de números nessa base. Represente os primeiros 16 números inteiros em base -2. Note que o número de bits necessários pode ser superior a quatro, o número mínimo de bits necessários para representar 16 configurações.

1.7 Realize as seguintes operações nas bases indicadas:

1. $1001{,}11011_2 + 0{,}10101_2$
2. $10010{,}11_2 \times 10{,}01_2$
3. $2314{,}21_5 + 12{,}413_5$
4. $24{,}1_5 \times 3{,}2_5$

1.8 Proponha o algoritmo da subtracção em base 2.

1.9 Construa as tabelas da soma e da multiplicação em base 16. Use-as para realizar as seguintes operações:

1. $\mathrm{A57D}_{16} + \mathrm{CAFE}_{16}$.
2. $\mathrm{A57D}_{16} \times 37_{16}$.
3. $\mathrm{35A,7E}_{16} + \mathrm{2D,1A6}_{16}$.
4. $\mathrm{35A,7E}_{16} \times \mathrm{7,F}_{16}$.

1.10 A conversão de números de uma base b para a base 10 é, como se viu, fácil de realizar, descrevendo o número como soma pesada (pelos algarismos) de potências da base e realizando as operações assim explicitadas. As operações são realizadas na base 10. Na passagem inversa, da base 10 para a base b, foi apresentado um algoritmo diferente. No entanto, a situação é perfeitamente simétrica, e o número em base 10 pode ser descrito como uma soma pesada de potências de 10, com todos os números intervenientes expressos em base b. Realizando seguidamente as operações em base b pode obter-se a representação do número nessa base. Experimente utilizar este método para representar em base 2 e 5 os números 35_{10} e 572_{10} e confirme utilizando os algoritmos usuais. O método também funciona com números fraccionários?

1.11 Partindo do algoritmo habitual da divisão decimal, obtenha o algoritmo da divisão em binário, execute e valide (usando a operação inversa) a divisão de 1010110_2 por 101_2.

1.12 Considere a seguinte sequência de bits: 100001110011

1. Se a sequência for um número em binário, qual é o número representado?
2. Se a sequência for a representação de um número em BCD, qual é o número representado?
3. Se a sequência for um número em base 3, qual é o número representado?

1.13 Considere os números do Problema 1.1.

1. Represente-os no código BCD.
2. Represente-os no código ASCII.
3. Represente-os no código ISO-Latin (ISO-8859-1).

1.14 Qual é a frase representada pela sequência de palavras do código ISO-8859-1 indicada seguidamente?

41h, 20h, 75h, 74h, 69h, 6Ch, 69h, 7Ah, 61h, E7h, E3h, 6Fh, 20h, 64h, 65h, 20h, 63h, F3h, 64h, 69h, 67h, 6Fh, 73h, 20h, 70h, 65h, 72h, 6Dh, 69h, 74h, 65h, 20h, 72h, 65h, 70h, 72h, 65h, 73h, 65h, 6Eh, 74h, 61h, 72h, 20h, 69h, 6Eh, 66h, 6Fh, 72h, 6Dh, 61h, E7h, E3h, 6Fh, 2Eh

1.15 Os nomes dos aeroportos são codificados por sequências de três letras maiúsculas. Por exemplo, o Aeroporto da Portela em Lisboa tem o código LIS, o Aeroporto de El Prat em Barcelona tem o código BCN, e o Aeroporto John Kennedy em Nova Iorque tem o código JFK.

1. Quantos aeroportos podem ser codificados deste modo?
2. Quantos bits serão necessários em código ASCII para codificar em binário os códigos dos aeroportos? E se se utilizasse o código mais eficiente possível para codificar apenas letras maiúsculas?
3. Quantos bits seriam necessários se se atribuísse um código binário único para cada aeroporto com um número mínimo de bits?

1.16 Um código ponderado é um código numérico em que cada bit do código tem um peso, e o número codificado se obtém por soma dos pesos multiplicados pelo valor do bit em cada respectiva posição. O código binário natural e o código BCD são códigos ponderados de pesos 8, 4, 2 e 1. Represente os dez algarismos decimais nos códigos ponderados seguintes:

1. 2, 4, 2, 1.
2. 5, 4, 2, 1.
3. 8, -4, 2, 1.

2

FUNÇÕES LÓGICAS

Os sistemas digitais, como já foi referido, assentam na utilização de circuitos electrónicos que podem assumir, em cada momento, um de dois estados. No Capítulo 1 verificou-se como é possível representar informação de vários tipos usando grandezas binárias.

A concepção e análise de circuitos do tipo dos utilizados em sistemas digitais carece de ferramentas teóricas que sirvam de base a técnicas que permitam esse tipo de trabalho. Em muitos casos, os circuitos electrónicos básicos devem desempenhar uma determinada função lógica. Por exemplo, imagine-se um sistema de alarme de incêndio que deverá disparar quando a temperatura se encontrar acima de um patamar e um de dois sensores de fumo se encontrar activado. A realização deste sistema passa por uma fase de análise e especificação, seguida de uma possível simplificação e implementação do resultado.

A funcionalidade do sistema é tipicamente especificada usando formalismos matemáticos, dos quais o mais importante é a álgebra de Boole. Do ponto de vista matemático, a álgebra de Boole binária é a teoria sobre a qual se suportam muitas das ferramentas práticas que são correntemente utilizadas em sistemas digitais. A álgebra de Boole binária será apresentada na Secção 2.1.

Aquela álgebra permite a utilização de funções booleanas, também designadas por funções lógicas, que são essenciais para as metodologias de análise e síntese de circuitos normalmente utilizadas. Formas de representação de funções lógicas serão apresentadas na Secção 2.2.

Um aspecto central na concepção de circuitos digitais é a minimização das funções lógicas, no sentido de reduzir a complexidade das expressões e, implicitamente, a complexidade dos circuitos usados para dar suporte a essas funções em sistemas digitais. Essa problemática será focada na Secção 2.3.

2.1 ÁLGEBRA DE BOOLE BINÁRIA

Há diversas formas de abordar a *álgebra de Boole*. Do ponto de vista algébrico, a maneira mais correcta consiste na definição axiomática de uma álgebra de Boole abstracta, com a consequente derivação da estrutura. Para os sistemas digitais, refinar-se-ia então a estrutura obtida para o caso da *álgebra de Boole binária*, isto é, a dois valores, decorrendo daí todas as propriedades interessantes no contexto dos sistemas digitais.

A abordagem seguida neste livro é diferente. A estrutura irá ser construída a partir do facto de se ter de trabalhar com entidades binárias, e a álgebra de Boole aparecerá *a posteriori*.

Como se referiu já, e se verá com um pouco mais de pormenor no Capítulo 3, os circuitos electrónicos que são a base de sistemas digitais podem exibir em cada momento um valor de tensão de entre dois intervalos possíveis. Essa situação pode ser encarada a um nível de maior abstracção por uma grandeza que pode assumir dois valores. Esses valores são habitualmente referidos como 0 e 1. Repare-se que a grandeza não tem qualquer significado numérico e os dois valores 0 e 1 não são números. Repare-se, ainda, que, em vez de 0 e 1, se podia usar qualquer outro conjunto de duas designações diferentes. São comuns as designações alternativas F e T (do inglês, *False* e *True*) ou L e H (do inglês, *Low* e *High*), por exemplo.

Uma grandeza que pode assumir um de dois valores é representada por uma *variável binária* ou *variável lógica* ou, ainda, *variável booleana*. Do mesmo modo, aos valores 0 e 1 que a variável booleana pode assumir chamam-se *valores booleanos*, *valores binários* ou *valores lógicos*.

2.1.1 Funções Lógicas de Uma Variável

O universo de partida é, portanto, o conjunto $\mathbb{B} = \{0, 1\}$. Trata-se de um conjunto finito de dois elementos, o que vai permitir uma abordagem algébrica muito simples.

Sobre este conjunto podem ser definidas funções denominadas *funções lógicas*, *funções binárias* ou *funções booleanas*. Como só há dois elementos no conjunto, o número de funções é finito. O número de funções de uma variável $f(x)$, por exemplo, corresponde às diversas formas de fazer corresponder elementos do conjunto $\mathbb{B}$ a cada um dos dois elementos desse conjunto.

O número de funções diferentes de uma variável é, como é fácil de ver, quatro. Estas funções são descritas na Tabela 2.1.

Tabela 2.1: Funções lógicas de uma variável.

	$x=0$	$x=1$	Expressão da função	Nome da Função
$f(x)$	0	0	$f_0(x) = 0$	constante 0
	0	1	$f_1(x) = x$	identidade
	1	0	$f_2(x) = \overline{x}$	negação
	1	1	$f_3(x) = 1$	constante 1

Repare-se que, ao contrário do que acontece com o conjunto $\mathbb{R}$ dos números reais ou até com o $\mathbb{N}$ dos números inteiros, neste conjunto $\mathbb{B}$ as funções $f(x)$ podem ser definidas

pela indicação exaustiva dos seus valores para cada valor possível da variável x, uma vez que estamos perante um número não só finito, mas também muito limitado de possibilidades. É, assim, possível definir realmente uma função de uma variável pela indicação do seu valor para cada valor da variável. Este tipo de tabelas é designado por *tabela de verdade* ou simplesmente por *tabela da função*.

Tal acontece, por exemplo, com a função $f_2(x)$. Trata-se de uma função que faz corresponder a cada valor x de $\mathbb{B}$ o outro valor do mesmo conjunto. Esta função é uma função importante e tem a designação de *negação*, *complementação* ou ainda NOT.

A designação desta função decorre do estudo inicial de Boole, que estudou a álgebra das proposições em que a função negação transformava uma afirmação *falsa* numa *verdadeira*, e vice-versa. Os valores *falso* e *verdadeiro* constituem o conjunto de dois valores, homólogo do conjunto $\mathbb{B}$, que motivou Boole para iniciar o estudo deste tipo de álgebras.

Das quatro funções de uma variável, a negação é a única que, como veremos, assume relevo. De facto, $f_0(x)$ e $f_3(x)$ são as duas possíveis constantes, $f_0(x) = 0$ e $f_3(x) = 1$, e a função $f_1(x) = x$ é também trivial.

Estas funções podem também ser representadas pela sua *expressão lógica*, como já foi feito para as três funções triviais no parágrafo anterior. A negação, por sua vez, é representada pela Expressão 2.1.

$$f_2(x) = \overline{x} \tag{2.1}$$

Uma outra maneira de representar as funções lógicas é graficamente. A negação é graficamente representada por um dos símbolos da Figura 2.1.

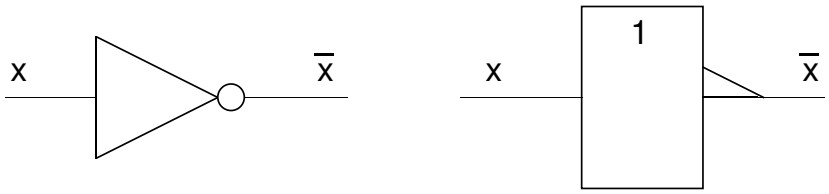


FIGURA 2.1: Representação gráfica da negação.

O facto de as funções lógicas poderem ser definidas pela sua tabela de verdade permite, no processo de algebrização formal, encarar a hipótese de demonstrar teoremas por indução completa. É óbvio que também é válida a metodologia que permite realizar essas demonstrações por dedução a partir de um conjunto de axiomas, mas o facto de existir a alternativa citada é, em si, um facto relevante com consequências várias.

O que se disse pode ser exemplificado para um importante teorema, o *teorema da dupla negação*:

$$\overline{\overline{x}} = x \tag{2.2}$$

A demonstração por indução completa parte do facto de só existirem dois valores possíveis para x. Se se verificar que, para cada um dos valores de x, a Expressão 2.2 se verifica, então o teorema fica demonstrado.

A Tabela 2.2 permite organizar essa verificação. De facto, verifica-se que, para os dois valores de x, a coluna x é igual à coluna $\overline{\overline{x}}$, o que demonstra por indução completa o teorema da dupla negação.

TABELA 2.2: Demonstração do teorema da dupla negação.

x	$\overline{x}$	$\overline{\overline{x}}$
0	1	0
1	0	1

2.1.2 FUNÇÕES DE DUAS VARIÁVEIS

As funções lógicas de duas variáveis $f(x, y)$ são também em número limitado, como é óbvio, se bem que em maior quantidade. De facto, uma tabela de duas variáveis tem $2^2 = 4$ linhas, ou, pensando em termos mais abstractos, o número de configurações possíveis de duas variáveis binárias é $2^2 = 4$; portanto, o número de funções será de $2^4 = 16$, uma vez que para cada função teremos uma sequência diferente de quatro grandezas binárias. As 16 funções estão representadas na Tabela 2.3.

2.1.3 AS FUNÇÕES AND E OR

Algumas das funções apontadas na Tabela 2.3 são mais relevantes que outras no contexto das aplicações ligadas aos sistemas digitais. A conjunção ou AND e a disjunção ou OR são, em particular, da maior relevância.

2.1.4 FUNÇÃO CONJUNÇÃO

A função *conjunção* ou AND é representada pela Expressão 2.3.

$$f(x, y) = x \wedge y \tag{2.3}$$

TABELA 2.3: Funções lógicas de duas variáveis.

x	0	0	1	1	Expressão da	Nome da
y	0	1	0	1	Função	Função
	0	0	0	0	$f_0(x,y) = 0$	constante 0
	0	0	0	1	$f_1(x,y) = x \wedge y$	conjunção ou AND
	0	0	1	0	$f_2(x,y) = \overline{x \Rightarrow y}$	implicação negada
	0	0	1	1	$f_3(x,y) = x$	identidade
	0	1	0	0	$f_4(x,y) = \overline{x \Leftarrow y}$	implicação negada
	0	1	0	1	$f_5(x,y) = y$	identidade
	0	1	1	0	$f_6(x,y) = x \oplus y$	disjunção exclusiva ou XOR
$f(x,y)$	0	1	1	1	$f_7(x,y) = x \vee y$	disjunção ou OR
	1	0	0	0	$f_8(x,y) = \overline{x \vee y}$	NOR
	1	0	0	1	$f_9(x,y) = x \Leftrightarrow y$	equivalência
	1	0	1	0	$f_{10}(x,y) = \overline{y}$	negação
	1	0	1	1	$f_{11}(x,y) = x \Leftarrow y$	implicação
	1	1	0	0	$f_{12}(x,y) = \overline{x}$	negação
	1	1	0	1	$f_{13}(x,y) = x \Rightarrow y$	implicação
	1	1	1	0	$f_{14}(x,y) = \overline{x \wedge y}$	NAND
	1	1	1	1	$f_{15}(x,y) = 1$	constante 1

Normalmente, porém, sempre que não haja possibilidade de confusão entre o AND e a multiplicação numérica, usa-se a representação da Expressão 2.4 que dá relevo ao facto de o AND ser uma função com carácter algébrico de produto.

$$f(x,y) = x \cdot y \tag{2.4}$$

Muitas vezes, para simplificar ainda mais, suprime-se o sinal $\cdot$ sempre que isso não cause confusão, como se representa na Expressão 2.5.

$$f(x,y) = x\ y \tag{2.5}$$

A tabela de verdade do AND é individualizada na Tabela 2.4.

A função AND, designação mais corrente que conjunção, é uma função que deriva o seu nome do facto de, como se pode ver na tabela, a função ser 1, apenas quando $x = 1$ **e** $y = 1$. É também comum designar a conjunção por *produto lógico*.

O AND é graficamente representado por um dos símbolos da Figura 2.2.

TABELA 2.4: Tabela de verdade da função conjunção de duas variáveis.

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

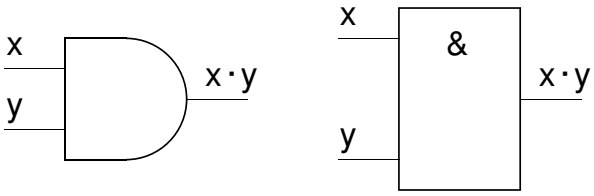


FIGURA 2.2: Representação gráfica da conjunção.

Há vários teoremas [1] envolvendo a conjunção que são importantes. Listam-se alguns:

Comutatividade da conjunção

$$x \cdot y = y \cdot x \tag{2.6}$$

Associatividade da conjunção

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{2.7}$$

Idempotência da conjunção

$$x \cdot x = x \tag{2.8}$$

Elemento neutro da conjunção

$$x \cdot 1 = x \tag{2.9}$$

Elemento absorvente da conjunção

$$x \cdot 0 = 0 \tag{2.10}$$

Complemento

$$x \cdot \overline{x} = 0 \tag{2.11}$$

2.1.5 Função Disjunção

A outra função já referida, que em conjunto com o AND é, como se verá na Secção 2.1.9, importante na definição da álgebra de Boole, é o OR ou *disjunção*. A expressão lógica da

[1] Na Secção 2.1.9 analisar-se-á em que contexto alguns destes teoremas o são realmente.

função disjunção ou OR está representada na Expressão 2.12

$$f(x, y) = x \vee y \tag{2.12}$$

Normalmente, porém, sempre que não haja possibilidade de confusão entre o OR e a soma numérica, usa-se a representação da Expressão 2.13, que dá relevo ao facto do OR ser uma função com carácter algébrico de soma.

$$f(x, y) = x + y \tag{2.13}$$

A tabela de verdade do OR é representada agora isoladamente na Tabela 2.5.

TABELA 2.5: Tabela de verdade da função disjunção de duas variáveis.

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

A disjunção ou OR é uma função que deriva o seu nome do facto de a função ser 1 quando $x = 1$ **ou** $y = 1$. É também comum designar a disjunção por *soma lógica*.

O OR é graficamente representado por um dos símbolos da Figura 2.3.

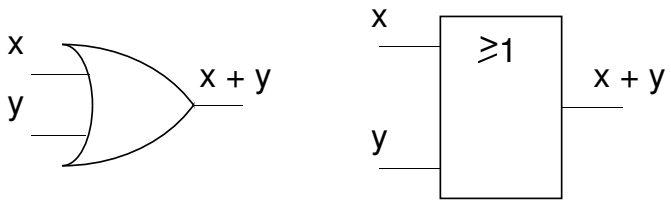


FIGURA 2.3: Representação gráfica da disjunção.

Tal como no caso da conjunção, há vários teoremas referentes à disjunção que são importantes:

Comutatividade da disjunção

$$x + y = y + x \tag{2.14}$$

Associatividade da disjunção

$$(x + y) + z = x + (y + z) \tag{2.15}$$

Idempotência da disjunção

$$x + x = x \tag{2.16}$$

Elemento neutro da disjunção

$$x + 0 = x \tag{2.17}$$

Elemento absorvente da disjunção

$$x + 1 = 1 \tag{2.18}$$

Complemento

$$x + \overline{x} = 1 \tag{2.19}$$

2.1.6 Princípio da Dualidade

Como se pode ver dos dois conjuntos de teoremas apresentados para a conjunção e disjunção, para cada expressão válida num dos conjuntos, existe uma expressão no outro que se obtém a partir da primeira, trocando entre si o par de valores 0 e 1 e o par de operadores AND e OR. Por exemplo, considerando o teorema do elemento neutro do produto lógico, $x \cdot 1 = x$, e substituindo o 1 pelo 0 e o operador $\cdot$ pelo $+$, obtém-se $x + 0 = x$, que é o teorema do elemento neutro da soma lógica. Essa situação é comum no tipo de estrutura algébrica que aqui se apresenta e resulta do *princípio da dualidade*.

O princípio da dualidade afirma, precisamente, que, se se verifica que uma expressão em termos de AND, OR e NOT é verdadeira, também o é a expressão que se obtém por troca de todos os operadores AND por OR, de todos os operadores OR por AND, de todos os valores 0 por 1 e de todos os valores 1 por 0.

O princípio da dualidade resulta de uma certa simetria que existe entre as operações conjunção e disjunção, patente nas respectivas tabelas. De facto, a conjunção ou AND é uma função em que o resultado só é 1 quando as duas variáveis são 1. Simetricamente, a disjunção ou OR só tem resultado 0 quando as duas variáveis da função são 0.

Ao longo de todo o desenvolvimento da matéria se poderá observar que esta simetria, que está na base do princípio da dualidade, vai apresentar diversas manifestações que suportam a oportunidade de observar sistematicamente a mesma realidade a partir de duas perspectivas simétricas.

2.1.7 Prioridade na Execução de Operações

Quando existem expressões lógicas que envolvem mais que uma operação, é necessário saber qual é a *prioridade na execução das operações*. Por exemplo, na expressão $x + y \cdot z$,

a ordem de execução das operações não é irrelevante. Na Tabela 2.6 ilustra-se o que acontece para o caso de se realizar em primeiro lugar o AND ou o OR.

TABELA 2.6: Tabela de verdade mostrando a ambiguidade da expressão $x + y \cdot z$.

x	y	z	Prioridade ao OR	Prioridade ao AND
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

A ambiguidade é eliminada pela utilização de parênteses. Assim, se se pretender realizar primeiro o produto lógico de y e z, deve indicar-se $x + (y \cdot z)$, enquanto que se se pretender executar primeiro a soma de x com y se deve representar $(x + y) \cdot z$. Para simplificar as expressões, porém, admite-se que o produto tem prioridade sobre a soma (como é usual na álgebra clássica) e, portanto, no primeiro caso apresentado podem suprimir-se os parênteses.

2.1.8 Teoremas Envolvendo Conjunção e Disjunção

Dois dos teoremas fundamentais que envolvem o AND e o OR simultaneamente são o par de teoremas referentes à *distributividade*. A existência de dois teoremas de distributividade, sendo um da soma em relação ao produto, e o outro, do produto em relação a soma, é compatível com o princípio da dualidade. Esses dois teoremas são o teorema da distributividade do produto em relação à soma (Expressão 2.20),

$$x \cdot (y + z) = x \cdot y + x \cdot z \tag{2.20}$$

e o teorema da distributividade da soma em relação ao produto (Expressão 2.21).

$$x + y \cdot z = (x + y) \cdot (x + z) \tag{2.21}$$

Qualquer dos teoremas apresentados nesta secção pode ser demonstrado por indução completa, à semelhança do que se fez para o teorema da dupla negação. A demonstração por indução completa para, por exemplo, o Teorema 2.21 está ilustrada na Tabela 2.7.

TABELA 2.7: Demonstração do teorema da distributividade da disjunção em relação à conjunção.

x	y	z	$x+y$	$x+z$	$(x+y)\cdot(x+z)$	$y\cdot z$	$x+y\cdot z$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	1	1	1	1
1	0	0	1	1	1	0	1
1	0	1	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Estas duas igualdades são importantes, uma vez que na aproximação axiomática da álgebra de Boole elas são postulados definidores da estrutura algébrica, como se verá na Secção 2.1.9.

Outros teoremas importantes envolvendo simultaneamente o AND e o OR são os seguintes:

Absorção

$$x + x \cdot y = x \tag{2.22}$$
$$x \cdot (x + y) = x \tag{2.23}$$

Redundância

$$x + \overline{x} \cdot y = x + y \tag{2.24}$$
$$x \cdot (\overline{x} + y) = x \cdot y \tag{2.25}$$

Consenso

$$x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z \tag{2.26}$$
$$(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z) \tag{2.27}$$

Leis de Morgan

$$\overline{x + y} = \overline{x} \cdot \overline{y} \tag{2.28}$$
$$\overline{x \cdot y} = \overline{x} + \overline{y} \tag{2.29}$$

As leis de Morgan, em particular, são extremamente importantes, uma vez que permitem «transformar» expressões em termos de somas em expressões em termos de produtos a menos de uma negação, como se ilustra, por exemplo, na Expressão 2.30.

$$\begin{aligned} x \cdot \overline{y} + \overline{z} \cdot w &= \overline{\overline{x \cdot \overline{y} + \overline{z} \cdot w}} \\ &= \overline{\overline{x \cdot \overline{y}} \cdot \overline{\overline{z} \cdot w}} \\ &= \overline{(\overline{x} + y) \cdot (z + \overline{w})} \end{aligned} \tag{2.30}$$

Graficamente, a mudança de estrutura é mais evidente. Na Figura 2.4 ilustra-se o mesmo exemplo.

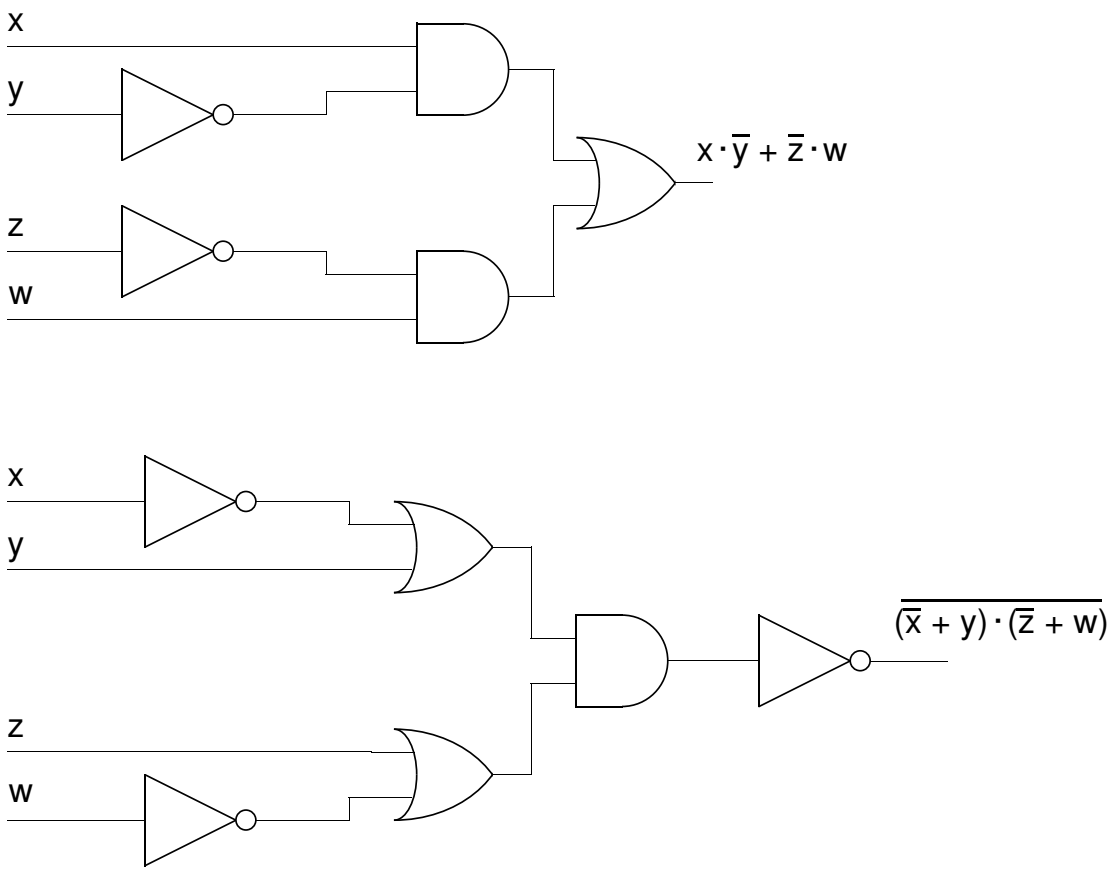


FIGURA 2.4: Exemplo de aplicação de uma das leis de Morgan.

A representação gráfica de uma função como a feita no exemplo anterior tem o nome de *logigrama* da função. O logigrama de uma função é, portanto, um dos modos de representar uma função.

2.1.9 Definição Formal de Álgebra de Boole

Ao longo deste capítulo tem vindo a construir-se, de maneira natural, uma estrutura algébrica a partir de um conjunto $\mathbb{B}$ de dois elementos e de algumas das funções que é possível definir sobre ele. Esta estrutura é uma de um conjunto de estruturas designadas por álgebras de Boole.

Na perspectiva que tem vindo a ser seguida, todas as propriedades anteriormente listadas são teoremas, uma vez que podem ser demonstradas a partir da definição das funções sobre o conjunto $\mathbb{B}$. Até aqui foi usado o método de indução completa para demonstrar esses teoremas. Os teoremas não são, contudo, independentes entre si e alguns podem ser demonstrados por dedução a partir de outros. No exemplo seguinte ilustra-se como se pode demonstrar por dedução o teorema representado na Expressão 2.31.

$$x + \overline{x} \cdot y = x + y \tag{2.31}$$

De facto, utilizando a distributividade, verifica-se que

$$x + \overline{x} \cdot y = (x + \overline{x}) \cdot (x + y) \tag{2.32}$$

Mas, considerando que $x + \overline{x} = 1$, resulta que

$$x + \overline{x} \cdot y = 1 \cdot (x + y) \tag{2.33}$$

De acordo com a comutatividade, vem

$$x + \overline{x} \cdot y = (x + y) \cdot 1 \tag{2.34}$$

E, em virtude do teorema do elemento neutro do produto lógico,

$$x + \overline{x} \cdot y = x + y \tag{2.35}$$

o que demonstra o teorema.

Um outro modo de encarar a definição de uma álgebra de Boole será a definição de um conjunto de postulados da estrutura algébrica, sendo, a partir deles, deduzidos os teoremas que permitem a caracterização da estrutura.

Nessa perspectiva, uma álgebra de Boole é um terno $\{\mathbb{B}, \vee, \wedge\}$ de um conjunto $\mathbb{B}$ e duas operações $\vee$ e $\wedge$ sobre os elementos do conjunto, que cumpre os seguintes postulados:

— P1: Se $a, b \in \mathbb{B}$, então

$$a \vee b = b \vee a \tag{2.36}$$
$$a \wedge b = b \wedge a \tag{2.37}$$

— P2: Se $a, b, c \in \mathbb{B}$, então

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \tag{2.38}$$
$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \tag{2.39}$$

— P3: O conjunto $\mathbb{B}$ inclui dois elementos, designados por 0 e 1, tais que, para cada $a \in \mathbb{B}$,

$$0 \vee a = a \vee 0 = a \tag{2.40}$$
$$1 \wedge a = a \wedge 1 = a \tag{2.41}$$

— P4: Para cada $a \in \mathbb{B}$, existe um elemento $\overline{a} \in \mathbb{B}$ designado por complemento de a, tal que,

$$a \vee \overline{a} = 1 \tag{2.42}$$
$$a \wedge \overline{a} = 0 \tag{2.43}$$

Desde que uma estrutura algébrica cumpra os postulados expostos, é uma álgebra de Boole. O conjunto dos subconjuntos de um conjunto e as operações reunião ($\cup$) e intersecção ($\cap$), por exemplo, constituem uma álgebra de Boole.

As álgebras de Boole de 2 elementos são um caso particular. A álgebra que se definiu atrás é, como se pode verificar, uma álgebra de Boole. Desse ponto de vista, todos os teoremas que foram apresentados podem ser deduzidos dos postulados agora expostos.

2.1.10 FUNÇÕES NAND E NOR

Para além das funções base da álgebra de Boole (AND, OR e NOT), há, como se apresentou já, um certo número de outras funções de duas variáveis definíveis sobre o conjunto binário $\{0, 1\}$. Duas outras importantes funções de duas variáveis são o NAND (a negação do AND) e o NOR (a negação do OR).

As expressões algébricas do NAND e do NOR são respectivamente a Expressão 2.44 e a Expressão 2.45.

$$f(x, y) = \overline{x \cdot y} \tag{2.44}$$

$$f(x, y) = \overline{x + y} \tag{2.45}$$

O NAND pode ser representado, mais simplesmente, pela Expressão 2.46.

$$f(x, y) = \overline{x\ y} \tag{2.46}$$

A Tabela 2.8 repete aqui, por conveniência, a tabela de verdade das duas funções.

TABELA 2.8: Funções NAND e NOR.

x	y	$\overline{x\ y}$	$\overline{x + y}$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

A representação gráfica da função NAND é ilustrada na Figura 2.5, e a da função NOR na Figura 2.6.

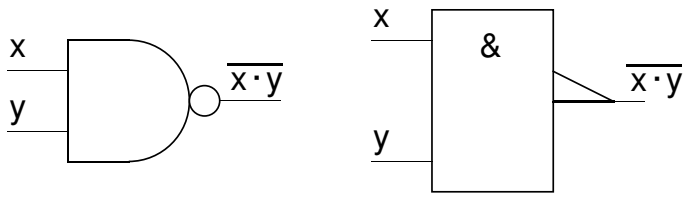


FIGURA 2.5: Representação gráfica do NAND.

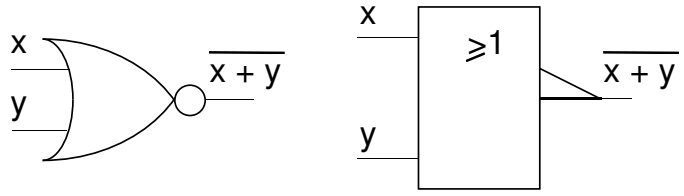


FIGURA 2.6: Representação gráfica do NOR.

Estas funções são comutativas, mas não gozam da propriedade da associatividade nem da da distributividade. A sua importância será clara um pouco mais adiante.

2.1.11 FUNÇÃO XOR

A última função de duas variáveis apresentada é a função *disjunção exclusiva* ou, como é também conhecida, XOR (do inglês, *eXclusive* OR).

A tabela de verdade do XOR é repetida, por conveniência, na Tabela 2.9.

TABELA 2.9: Função XOR.

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

A função disjunção exclusiva tem essa designação por assumir o valor 1, apenas quando uma e só uma das variáveis é 1. A Expressão 2.47 é a expressão lógica do XOR.

$$f(x, y) = x \oplus y \tag{2.47}$$

A representação gráfica do XOR está ilustrada na Figura 2.7.

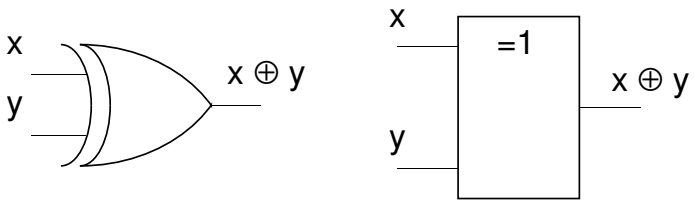


FIGURA 2.7: Representação gráfica do XOR.

A disjunção exclusiva é uma função comutativa e associativa. Outros teoremas mais específicos e de grande interesse prático estão representados nas Expressões 2.48 e 2.49.

$$x \oplus y = \overline{x} \cdot y + x \cdot \overline{y} \tag{2.48}$$

$$x \oplus y = (x + y) \cdot (\overline{x} + \overline{y}) \tag{2.49}$$

Estes dois teoremas permitem relacionar a disjunção exclusiva com a conjunção, a disjunção e a negação e representá-la em termos dessas funções.

Os dois teoremas ilustrados nas Expressões 2.50 e 2.51 permitem uma nova perspectiva do XOR.

$$x \oplus 0 = x \tag{2.50}$$

$$x \oplus 1 = \overline{x} \tag{2.51}$$

Considerando que uma das variáveis pode ser usada para controlar a função (c), e a outra ser a variável de entrada (x), o XOR $y = x \oplus c$ pode ser visto como uma negação controlada. De facto, se $c = 0$, virá $y = x$, e se $c = 1$, virá $y = \overline{x}$.

Outro interessante teorema está definido na Expressão 2.52.

$$\overline{x \oplus y} = \overline{x} \oplus y = x \oplus \overline{y} \tag{2.52}$$

Este teorema mostra que a negação do XOR é igual ao XOR de uma das variáveis negada com a outra variável. Tem como consequência óbvia o teorema da Expressão 2.53.

$$\overline{x} \oplus \overline{y} = x \oplus y \tag{2.53}$$

2.1.12 Funções de n Variáveis

Nas secções anteriores foram estudadas as funções lógicas de uma e duas variáveis e foi apresentada a álgebra de Boole, estrutura matemática que dá suporte à sua utilização e manipulação. Com base na álgebra de Boole é possível generalizar os conceitos apresentados a funções com mais de duas variáveis. Algumas das funções apresentadas são facilmente generalizáveis. É o caso das funções conjunção e disjunção. A função conjunção de três entradas é definida, aproveitando a associatividade, pela Expressão 2.54.

$$x \cdot y \cdot z = (x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{2.54}$$

A generalização para n variáveis é imediata.

Do mesmo modo, para a disjunção, a definição é feita pela Expressão 2.55

$$x + y + z = (x + y) + z = x + (y + z) \tag{2.55}$$

Noutros casos é necessário proceder com mais cuidado. No caso do NAND e NOR, por exemplo, em que essas duas funções não são associativas, a definição parte das funções conjunção e disjunção e aproveita o facto de essas duas serem, como vimos, funções associativas, como se ilustra nas Expressões 2.56 e 2.57.

$$\overline{x \cdot y \cdot z} = \overline{(x \cdot y) \cdot z} = \overline{x \cdot (y \cdot z)} \tag{2.56}$$

$$\overline{x + y + z} = \overline{(x + y) + z} = \overline{x + (y + z)} \tag{2.57}$$

A disjunção exclusiva, por outro lado, não tem generalização para mais de duas variáveis.

É possível, agora, definir funções com n variáveis, usando as funções anteriores. Por exemplo, a função representada na Expressão 2.58 é uma função de quatro variáveis definida através de uma expressão lógica.

$$f(x, y, z, w) = x \cdot y + \overline{x} \cdot y \cdot z + \overline{(\overline{x \cdot \overline{y}} + w)} + (x \cdot y \oplus z) \tag{2.58}$$

Nas expressões lógicas envolvendo a disjunção exclusiva, a função conjunção tem prioridade sobre a disjunção exclusiva, pelo que não há ambiguidade no último termo da Expressão 2.58. Não é definida qualquer ordem de prioridade entre a disjunção e a disjunção exclusiva, o que obriga, nesses casos, ao uso de parênteses.

Claro que se podem também representar funções de n variáveis através da sua tabela ou de um logigrama. A Figura 2.8 e a Tabela 2.10 são duas outras representações da função definida pela Expressão 2.58. Repare-se no símbolo do OR usado na figura que é comum quando o número de entradas o impõe. Do mesmo modo, o símbolo do AND pode ser alterado para acomodar um maior número de entradas, como se ilustra para o caso de cinco variáveis na Figura 2.9.

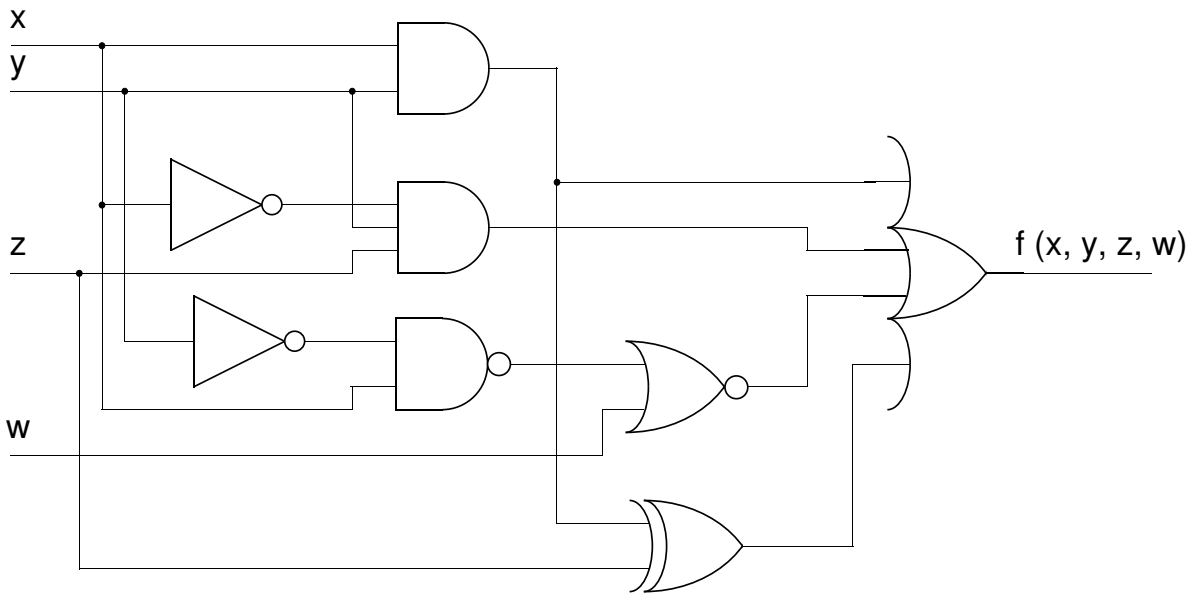


FIGURA 2.8: Representação gráfica da função definida pela Expressão 2.58.

TABELA 2.10: Tabela da função definida pela Expressão 2.58.

x	y	z	w	$f(x,y,z,w)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

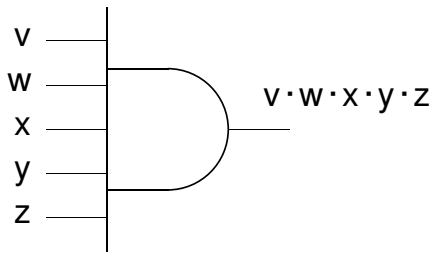


FIGURA 2.9: Símbolo de uma conjunção de cinco variáveis.

2.1.13 MANIPULAÇÃO DE EXPRESSÕES LÓGICAS

O conjunto de teoremas apresentados nas secções anteriores permite manipular algebricamente expressões lógicas, quer para as simplificar, quer para obter expressões equivalentes que cumpram determinados critérios. Nesta secção serão apresentados exemplos dos dois tipos de manipulação.

Como atrás se referiu, a função conjunção pode ser representada de três formas diferentes em termos de expressão lógica. A conjunção de duas variáveis x e y pode ser representada por $x \wedge y$, ou mais geralmente por $x \cdot y$ e até, quando não haja possibili-

dade de confusão, por $x\ y$. A partir desta secção, a menos que seja necessário proceder de outro modo, utilizar-se-á sistematicamente a terceira forma, mais comum, de representar a conjunção ou AND.

A questão da simplificação de expressões é do maior interesse, pois, como se verá adiante, a implementação de circuitos que concretizem funções lógicas conduz a estruturas com maior ou menor número de componentes, de acordo com a complexidade das expressões que definem as funções.

Considere-se a título de exemplo a simplificação da Expressão 2.59 de uma função lógica de três variáveis:

$$f(a,b,c) = \overline{a}\ \overline{b}\ c + b\ c + a\ c \tag{2.59}$$

Por aplicação da comutatividade (Expressão 2.6), obtém-se,

$$f(a,b,c) = c\ \overline{a}\ \overline{b} + c\ b + c\ a \tag{2.60}$$

Tendo em conta a distributividade do produto em relação à soma (Expressão 2.20), temos

$$f(a,b,c) = c\ (\overline{a}\ \overline{b} + b) + c\ a \tag{2.61}$$

Da comutatividade (Expressão 2.6) e do Teorema 2.24 resulta que

$$f(a,b,c) = c\ (\overline{a} + b) + c\ a \tag{2.62}$$

Aplicando de novo a distributividade do produto em relação à soma:

$$f(a,b,c) = c\ (\overline{a} + b + a) \tag{2.63}$$

Tendo, de novo, em consideração a comutatividade (Expressão 2.6) e, ainda, o Teorema 2.19, vem

$$f(a,b,c) = c\ (1 + b) \tag{2.64}$$

e, portanto (Expressão 2.6, Expressão 2.18 e Expressão 2.9),

$$f(a,b,c) = c \tag{2.65}$$

Neste caso verifica-se que a função de três variáveis inicial só nominalmente o é, uma vez que, como se verifica, ficou reduzida à função trivial da Expressão 2.65.

Outro exemplo é a simplificação da expressão da função usada como exemplo na secção anterior (Expressão 2.58). Desta vez, é deixado como exercício a identificação dos diversos teoremas usados.

$$\begin{aligned} f(x,y,z,w) &= x\ y + \overline{x}\ y\ z + \overline{(\overline{x\ \overline{y}} + w)} + (x\ y \oplus z) \\ &= (x + \overline{x}\ z)\ y + \overline{(\overline{x} + y + w)} + x\ y\ \overline{z} + \overline{x\ y}\ z \\ &= x\ y + y\ z + x\ \overline{y}\ \overline{w} + x\ y\ \overline{z} + \overline{x\ y}\ z \\ &= x\ y + y\ z + x\ \overline{w} + \overline{x\ y}\ z \\ &= x\ y + y\ z + x\ \overline{w} + z \\ &= x\ y + x\ \overline{w} + z \end{aligned} \tag{2.66}$$

Como é fácil de ver, a manipulação da expressão permitiu obter uma expressão muito mais simples, que também se reflecte no logigrama da Figura 2.10, com muito menos operadores e com uma complexidade muito inferior à da Figura 2.8.

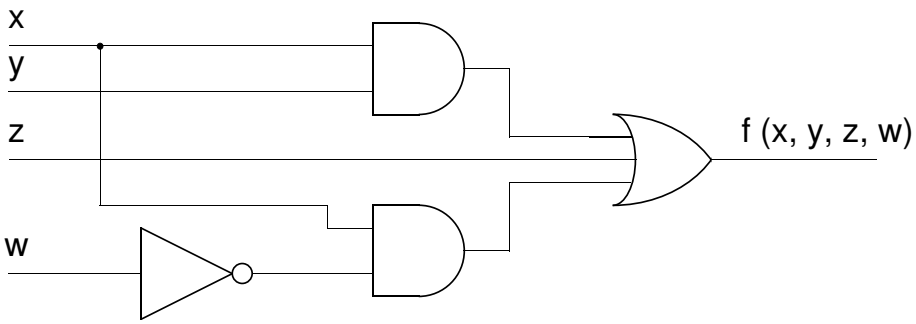


FIGURA 2.10: Representação gráfica da função definida pela Expressão 2.66.

Como se referiu, por vezes a manipulação de expressões lógicas tem como fim a obtenção de determinadas formas e não a sua simplificação. Como exemplo, considere-se a manipulação da Expressão 2.67 para obter uma representação utilizando apenas negações e operadores de duas variáveis de entrada:

$$f(a,b,c,d) = \overline{a}\ \overline{b}\ d + \overline{a}\ \overline{c}\ d + a\ c\ d + a\ \overline{b}\ \overline{d} \tag{2.67}$$

É fácil, aplicando o teorema da distributividade do produto em relação à soma, chegar à Expressão 2.68.

$$f(a,b,c,d) = \overline{a}\ (\overline{b}\ d + \overline{c}\ d) + a\ (c\ d + \overline{b}\ \overline{d}) \tag{2.68}$$

Esta expressão utiliza apenas operadores de duas entradas, para além de negações. Essa estrutura é, talvez, mais evidente na Figura 2.11.

Outro exemplo é o da manipulação da Expressão 2.69 para obter uma expressão da mesma função, usando apenas NAND, e negações.

$$f(a,b,c,d) = (a \oplus b)\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ (b + d) \tag{2.69}$$

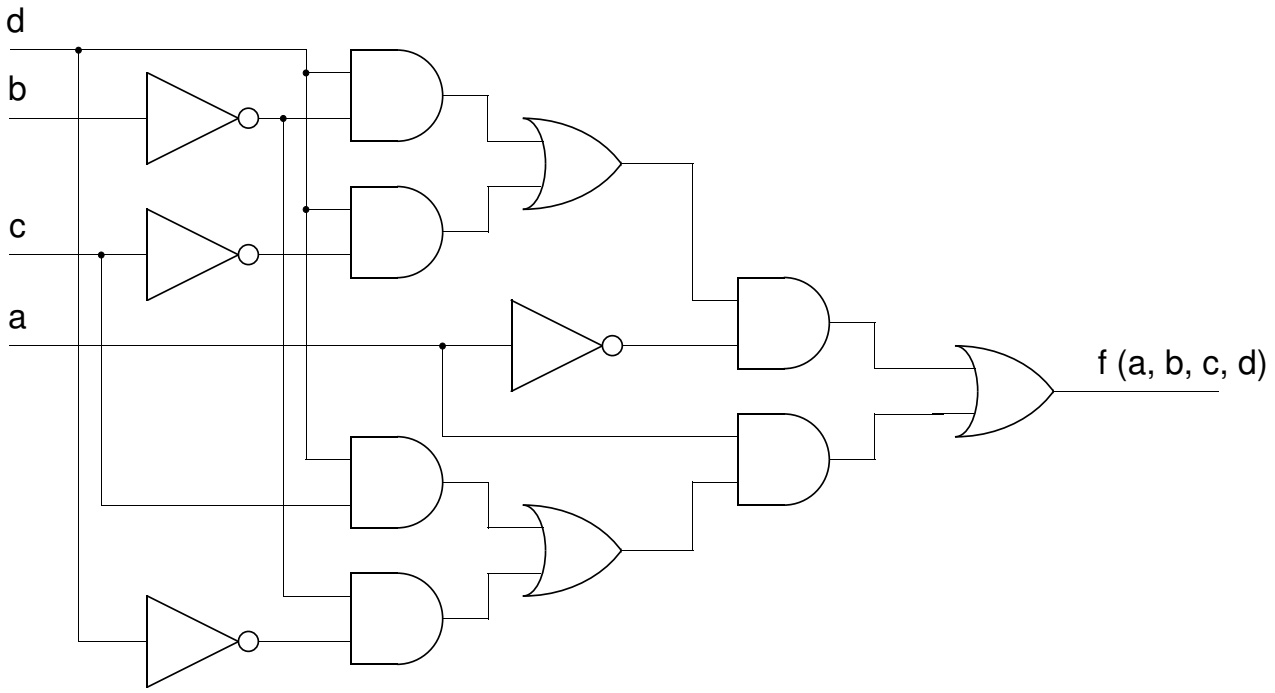


FIGURA 2.11: Representação gráfica da função definida pela Expressão 2.68.

Aplicando o Teorema 2.48 e a distributividade do produto em relação à soma (Expressão 2.20), obtém-se facilmente a Expressão 2.70.

$$\begin{aligned} f(a,b,c,d) &= (a \oplus b)\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ (b+d) \\ &= (a\ \overline{b} + \overline{a}\ b)\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ (b+d) \\ &= a\ \overline{b}\ c + \overline{a}\ b\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ b + \overline{a}\ \overline{c}\ d \end{aligned} \tag{2.70}$$

Por aplicação do teorema da dupla negação (Expressão 2.2) e de uma das leis de Morgan (Expressão 2.28), obtém-se a Expressão 2.71, que é a expressão pretendida.

$$\begin{aligned} f(a,b,c,d) &= a\ \overline{b}\ c + \overline{a}\ b\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ b + \overline{a}\ \overline{c}\ d \\ &= \overline{\overline{a\ \overline{b}\ c + \overline{a}\ b\ c + b\ \overline{c}\ d + \overline{a}\ \overline{c}\ b + \overline{a}\ \overline{c}\ d}} \\ &= \overline{\overline{a\ \overline{b}\ c}\ \overline{\overline{a}\ b\ c}\ \overline{b\ \overline{c}\ d}\ \overline{\overline{a}\ \overline{c}\ b}\ \overline{\overline{a}\ \overline{c}\ d}} \end{aligned} \tag{2.71}$$

2.2 REPRESENTAÇÃO DE FUNÇÕES LÓGICAS

Como se viu, há diversas formas de *representação de funções lógicas*, nomeadamente pela sua expressão lógica, pela sua tabela ou pelo seu logigrama. Considere-se, a título de exemplo, a função representada pela Expressão 2.72.

$$f(a,b,c) = b\ \overline{c} + a\ \ b \tag{2.72}$$

A expressão indicada é uma forma de representar a função. Não é a única. É fácil de ver que a Expressão 2.73 é igualmente uma representação da mesma função $f(a,b,c)$.

$$f(a,b,c) = b\ \ (\overline{c} + a) \tag{2.73}$$

A Expressão 2.72 tem a forma de uma disjunção de conjunções. Expressões com este formato dizem-se estar na *forma normal disjuntiva*. Na forma normal disjuntiva, uma ou mais das conjunções pode estar reduzida a uma variável eventualmente negada. A Expressão 2.73 tem a forma de uma conjunção de disjunções (uma das disjunções está reduzida à variável b). A este formato chama-se *forma normal conjuntiva*. À semelhança do caso anterior, uma ou mais das disjunções podem estar reduzidas a uma variável, negada ou não.

Uma outra forma de representar a função é pela sua tabela. A obtenção da tabela de uma função a partir da sua expressão é relativamente fácil. Obtêm-se tabelas para partes da expressão da função que sejam fáceis de obter por serem operadores simples e, a partir daí, constrói-se a tabela final, operando sobre as tabelas parciais. No caso da função em estudo, pode obter-se, por exemplo, a tabela de $\overline{c}$, construindo, a partir dela, a tabela de $b\ \overline{c}$ e, em paralelo, a tabela de $a\ \ b$. Por fim, pode obter-se a tabela da soma dos dois produtos, que é a tabela pretendida. Na Tabela 2.11 está ilustrada a metodologia exposta.

TABELA 2.11: Construção da tabela de uma função.

a	b	c	$\overline{c}$	$b\ \overline{c}$	$a\ \ b$	$f(a,b,c)$
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	1	1	0	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	1	0	1	1	1	1
1	1	1	0	0	1	1

A tabela de uma função, ao contrário da expressão, é, como é fácil de compreender, única.

Uma terceira forma de representar uma função é o seu logigrama. Como o logigrama corresponde à representação gráfica de uma expressão lógica, não há um logigrama único para representar uma função. No caso da função em estudo, haverá um logigrama para cada expressão possível da função. O logigrama correspondente à Expressão 2.72 está representado na Figura 2.12.

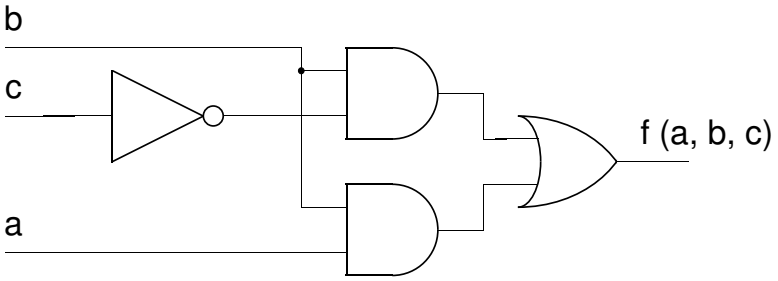


FIGURA 2.12: Representação gráfica da função definida pela Expressão 2.72.

Todas as formas de representar uma dada função representam a mesma informação, sendo, portanto, possível passar de cada uma delas a todas as outras. Se, por exemplo, se partir de uma expressão de uma função, a obtenção das outras duas formas é, como se viu, fácil. De igual modo, a passagem do logigrama para uma expressão e seguidamente para a tabela não oferece quaisquer dificuldades. Já a obtenção de uma expressão de uma função a partir da sua tabela, não é tão imediata. Nas subsecções seguintes, analisaremos esse problema.

2.2.1 Forma Canónica Normal Disjuntiva

Considere-se, de novo, a Tabela 2.11. Observando a tabela, é fácil concluir que a função representada tem valor 1 em três posições da tabela e valor 0 nas restantes cinco posições. Pode-se concluir daí que a função $f(a, b, c)$ representada pode ser considerada como a soma de três funções $f_x(a, b, c)$, $f_y(a, b, c)$ e $f_z(a, b, c)$, em que cada uma destas funções é representada por uma tabela com apenas uma linha com valor 1. A Tabela 2.12 ilustra, para além da função $f(a, b, c)$, essas três funções.

As linhas da tabela estão numeradas de 0 a 7 para facilitar a identificação. Repare-se que a numeração de cada linha corresponde à interpretação como um número binário das configurações das três variáveis lógicas a, b e c da respectiva linha.

É fácil perceber que

$$f(a, b, c) = f_x(a, b, c) + f_y(a, b, c) + f_z(a, b, c) \tag{2.74}$$

TABELA 2.12: Tabela das funções $f(a,b,c)$, $f_x(a,b,c)$, $f_y(a,b,c)$ e $f_z(a,b,c)$.

Linha	a	b	c	$f(a,b,c)$	$f_x(a,b,c)$	$f_y(a,b,c)$	$f_z(a,b,c)$
0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	1	0	1	1	0	0
3	0	1	1	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	1	0	0	0	0
6	1	1	0	1	0	1	0
7	1	1	1	1	0	0	1

A identificação das expressões das funções $f_x(a,b,c)$, $f_y(a,b,c)$ e $f_z(a,b,c)$ não é tão imediata. Mas, se se considerar a função $f_z(a,b,c)$, é fácil verificar que se trata de um AND de três variáveis:

$$f_z(a,b,c) = a\ \ b\ \ c \tag{2.75}$$

É também possível verificar que $f_y(a,b,c)$ é uma função que assume valor 1, apenas quando $a=1$, $b=1$ e $c=0$, isto é, quando $a=1$, $b=1$ e $\overline{c}=1$. Então,

$$f_y(a,b,c) = a\ \ b\ \ \overline{c} \tag{2.76}$$

Do mesmo modo é possível concluir que

$$f_x(a,b,c) = \overline{a}\ \ b\ \ \overline{c} \tag{2.77}$$

Das Expressões 2.74 a 2.77 é fácil obter uma expressão da função $f(a,b,c)$:

$$f(a,b,c) = a\ \ b\ \ c + a\ \ b\ \ \overline{c} + \overline{a}\ \ b\ \ \overline{c} \tag{2.78}$$

O método é utilizável com generalidade e, portanto, é sempre possível definir a expressão de uma função a partir da sua tabela. Este método possibilita, assim, a obtenção de uma expressão lógica de uma função a partir da sua tabela de verdade. A expressão obtida não é a expressão mais simples. Tem, no entanto, uma característica muito importante: trata-se de uma forma normal disjuntiva em que todos os produtos envolvem todas as variáveis da função, ainda que algumas possam estar negadas.

Os produtos deste tipo chamam-se *mintermos* ou *termos mínimos*. A expressão em termos de soma de mintermos é única. De facto, como a tabela de uma função é única e esta

expressão reflecte, na sua estrutura, a estrutura da tabela, esta é igualmente única. Esta expressão tem o nome de *primeira forma canónica* ou *forma canónica normal disjuntiva*. Na forma canónica normal disjuntiva de uma função, cada mintermo corresponde a uma linha da tabela, em que a função assume o valor 1.

Repare-se, uma vez mais, que cada mintermo corresponde a um dos valores 1 da função. Por exemplo, o mintermo $a\ b\ c$ corresponde ao 1 da última linha da tabela. É usual numerar as linhas da tabela, como se fez na Tabela 2.12. Podemos, então, referir cada um dos mintermos pelo número da respectiva linha da tabela. Por exemplo, o mintermo $a\ b\ c$ poderá ser designado por m_7. E, desse modo, podemos abreviar a Expressão 2.78 para

$$f(a,b,c) = m_2 + m_6 + m_7 \tag{2.79}$$

ou, de forma ainda mais abreviada,

$$f(a,b,c) = \sum m(2,6,7) \tag{2.80}$$

Como é evidente, esta convenção exige que seja assumida uma determinada ordem das variáveis na tabela da função. Nos exemplos referidos, a variável a foi tomada como a variável correspondente ao bit de maior peso, e a variável c, ao de menor peso na numeração das linhas da tabela.

A obtenção da expressão de um mintermo a partir do seu número realiza-se fazendo a correspondência do número em binário com as diversas variáveis ou as suas negações. O mintermo m_2, por exemplo, corresponde à posição da linha 2 da tabela, isto é, à linha em que $a = 0$, $b = 1$ e $c = 0$ (configuração 010 em binário). Daqui se conclui que $m_2 = \overline{a}\ b\ \overline{c}$. Inversamente, se se conhecer a expressão de um mintermo de uma função, por exemplo, $a\ \overline{b}\ \overline{c}$, é fácil concluir que a expressão só assume o valor 1 quando $a = 1$, $b = 0$ e $c = 0$, o que corresponde a 100 em binário, correspondendo, portanto, à linha 4 da tabela. Trata-se, pois, do mintermo m_4.

A partir de uma tabela de uma função não oferece, portanto, dificuldade obter a forma canónica normal disjuntiva da função representada, ainda que esta não seja em geral a expressão mais simples.

À forma canónica normal disjuntiva de uma função corresponde, como é evidente, um logigrama. No caso da função em estudo, o logigrama correspondente está ilustrado na Figura 2.13. Considerando o que já foi dito sobre a estrutura da forma canónica normal disjuntiva, é agora possível concluir que, no logigrama correspondente a essa forma, cada operador AND corresponde também a uma das linhas da tabela da função, em que esta assume o valor 1.

É fácil perceber, quer pela expressão da forma canónica normal disjuntiva, quer pela observação do logigrama correspondente, que a função é representada, se não tivermos

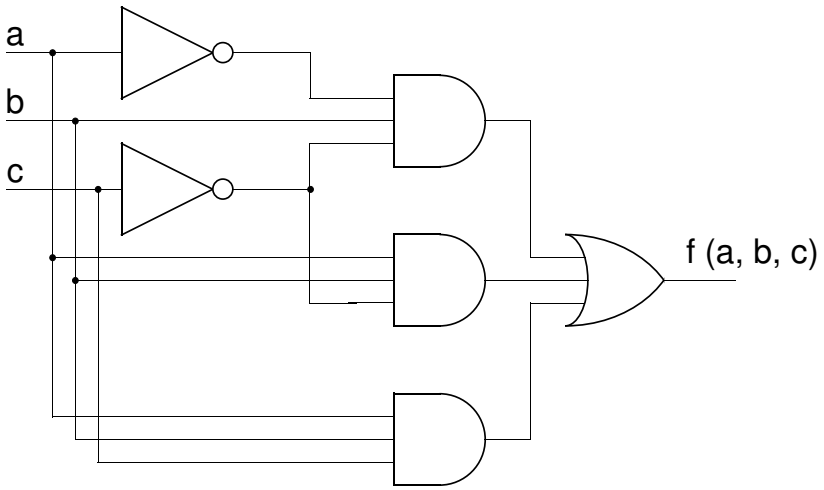


FIGURA 2.13: Logigrama correspondente à forma canónica normal disjuntiva de $f(a, b, c)$.

em conta as negações, por dois níveis de operadores. No primeiro nível encontram-se conjunções e, no segundo, uma disjunção. É usual designar este tipo de representação por *representação a dois níveis*. Como se verá, este tipo de representação é de grande importância. A não consideração das negações como um terceiro nível resulta de que, na prática, muitas vezes as variáveis estão disponíveis, quer na forma não negada, quer na negada, não sendo necessário, por isso, considerar explicitamente operadores de negação.

2.2.2 Forma Canónica Normal Conjuntiva

Na Secção 2.2.1 foi apresentado um processo de obter uma expressão de uma função, a forma normal disjuntiva, a partir da sua tabela. Como se viu, a expressão é construída através da soma dos seus mintermos, correspondendo estes às posições da tabela em que a função assume o valor 1. O princípio da dualidade permite, porém, antever que um método inteiramente semelhante é possível, partindo agora das configurações de variáveis em que a função vale 0. De facto, repetindo a tabela da função, isolando agora os valores 0, obtém-se a Tabela 2.13.

Cada uma das funções M_i tem na sua tabela apenas um 0, que corresponde a uma das possíveis configurações de variáveis. Por exemplo a função M_0 assume o valor 0, apenas para a configuração $a = b = c = 0$ e, como é fácil de ver, corresponde ao OR das variáveis:

$$M_0 = a + b + c \tag{2.81}$$

TABELA 2.13: Tabela da função $f(a,b,c)$ e dos seus maxtermos.

Linha	a	b	c	$f(a,b,c)$	M_0	M_1	M_3	M_4	M_5
0	0	0	0	0	0	1	1	1	1
1	0	0	1	0	1	0	1	1	1
2	0	1	0	1	1	1	1	1	1
3	0	1	1	0	1	1	0	1	1
4	1	0	0	0	1	1	1	0	1
5	1	0	1	0	1	1	1	1	0
6	1	1	0	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1

Do mesmo modo,

$$M_1 = a + b + \overline{c} \tag{2.82}$$
$$M_3 = a + \overline{b} + \overline{c} \tag{2.83}$$
$$M_4 = \overline{a} + b + c \tag{2.84}$$
$$M_5 = \overline{a} + b + \overline{c} \tag{2.85}$$

Cada uma destas funções é designada por *maxtermo* ou *termo máximo*. Maxtermos são somas lógicas em que estão representadas todas as variáveis da função, ainda que possam estar negadas.

A obtenção da expressão do maxtermo a partir do seu número é semelhante ao que se observou no caso dos mintermos, mas aqui com as alterações impostas pelo princípio da dualidade. Assim, por exemplo, o maxtermo M_3 é o maxtermo que só assume o valor 0 quando $a = 0$, $b = 1$ e $c = 1$. Isto corresponde à configuração binária 011. A função M_3 assume, portanto, o valor 1 quando $a = 1$ ou $b = 0$ ou $c = 0$. Daí que $M_3 = a + \overline{b} + \overline{c}$.

A expressão da função pode ser obtida por conjunção dos diversos maxtermos:

$$f(a,b,c) = (a+b+c)\ (a+b+\overline{c})\ (a+\overline{b}+\overline{c})\ (\overline{a}+b+c)\ (\overline{a}+b+\overline{c}) \tag{2.86}$$

Esta expressão, em termos de produto de maxtermos, tem o nome de *segunda forma canónica* ou *forma canónica normal conjuntiva* e, tal como no caso da forma canónica normal disjuntiva, é única. Na forma canónica normal conjuntiva de uma função cada maxtermo corresponde a uma linha da tabela em que a função assume o valor 0.

Na Figura 2.14 representa-se o logigrama correspondente à forma normal conjuntiva da função como descrita na Expressão 2.86.

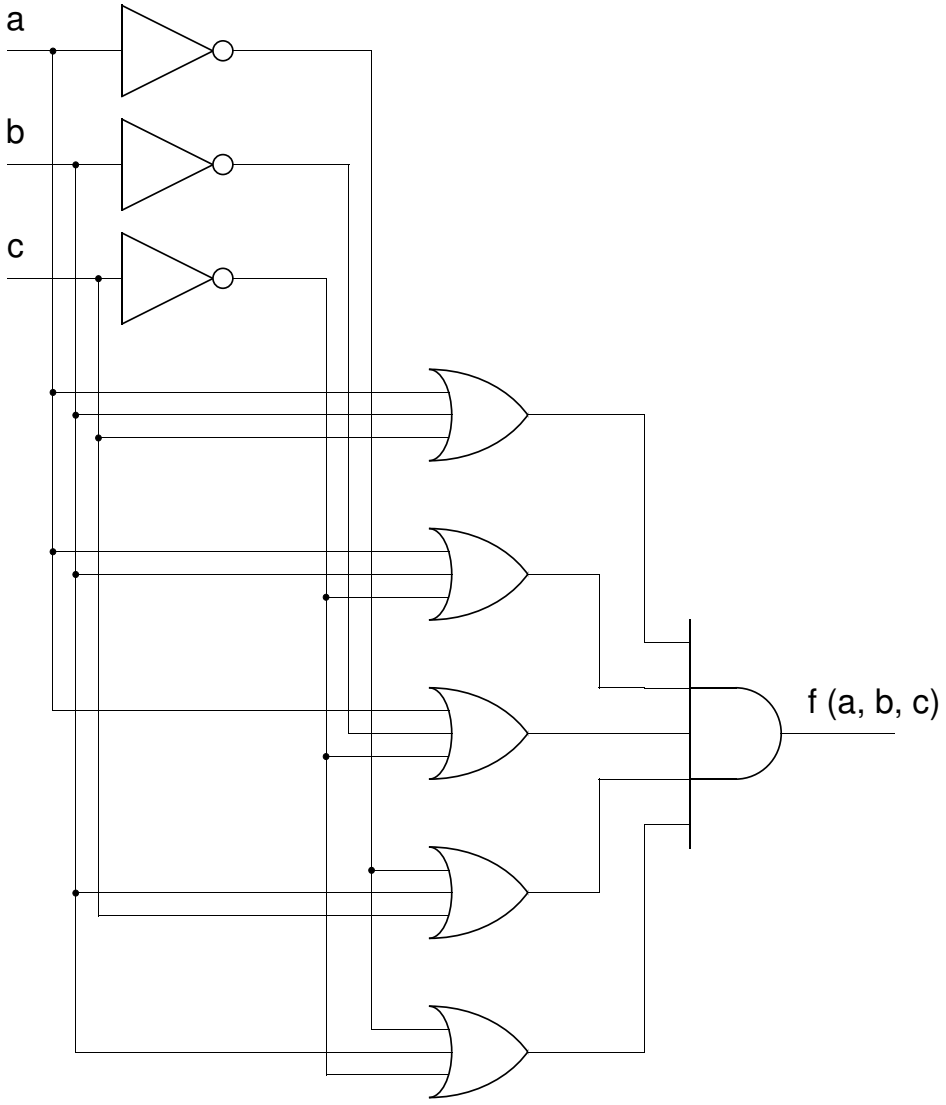


FIGURA 2.14: Logigrama correspondente à forma canónica normal conjuntiva de $f(a, b, c)$.

Do mesmo modo que acontece no caso da forma canónica normal disjuntiva, é possível indicar a expressão de forma abreviada por indicação dos maxtermos de que é feito o produto:

$$f(a, b, c) = M_0 \ M_1 \ M_3 \ M_4 \ M_5 \tag{2.87}$$

ou, de maneira ainda mais abreviada,

$$f(a, b, c) = \prod M(0, 1, 3, 4, 5) \tag{2.88}$$

Tal como no caso da forma normal disjuntiva, esta representação é uma representação a dois níveis.

Repare-se que, para qualquer função, em cada linha da tabela existe, em alternativa, um 0 ou um 1. Isso significa que se essa linha corresponde a um mintermo da função não pode corresponder a um maxtermo, e vice-versa. Tal como na tabela, se se conhecerem todas as linhas a 1, conhecem-se implicitamente todas as linhas a 0; se se conhecer a forma canónica normal disjuntiva de uma função, conhece-se a forma canónica normal conjuntiva, e vice-versa.

Por exemplo, para a função $f(a,b,c) = \sum m(0,3,6)$, é possível concluir de maneira imediata que $f(a,b,c) = \prod M(1,2,4,5,7)$. É assim possível obter directamente a expressão da forma canónica normal disjuntiva (Expressão 2.86) a partir da forma canónica normal conjuntiva (Expressão 2.78). O processo inverso é análogo.

2.2.3 Representação de Funções Usando Um só Tipo de Função

As duas secções anteriores apresentaram os procedimentos para obter uma expressão lógica de uma função a partir da sua tabela. O método usado, para obter as formas canónicas normais disjuntiva e conjuntiva, conduz a expressões em que apenas são utilizadas as funções negação, disjunção e conjunção. Como não se fez qualquer restrição ao tipo possível de funções, qualquer função de qualquer número de variáveis pode ser submetida ao procedimento apresentado, sendo este, portanto, geral. Uma consequência óbvia da maior importância é que qualquer função lógica pode ser representada por uma expressão utilizando apenas funções do tipo das indicadas, isto é, negações, disjunções e conjunções.

Um conjunto de funções que permite, quando utilizado em expressões lógicas, representar qualquer outra função chama-se *conjunto completo*. O conjunto {negação, conjunção, disjunção} é, então, um conjunto completo.

Há diversos outros conjuntos completos, incluindo dois subconjuntos do conjunto anterior. Dois conjuntos completos assumem, porém, grande importância. Trata-se do conjunto constituído apenas pela função NAND e do conjunto constituído pela função NOR.

Se for possível representar qualquer das três funções — negação, conjunção e disjunção — apenas por funções NAND ou NOR, considerando que qualquer função pode ser representada por aquelas três funções, fica provado que qualquer função se pode representar usando apenas funções NAND ou NOR. É fácil fazer a prova. Exemplifica-se para o caso do NAND (o caso do NOR seria semelhante).

Considere-se a função negação. Recorrendo ao teorema da idempotência (Expressão 2.8)

é fácil verificar a Expressão 2.89.

$$\overline{x} = \overline{x\ x} \tag{2.89}$$

Mas $\overline{x\ x}$ é a função NAND.

Utilizando, agora, o teorema da dupla negação (Expressão 2.2) e a Expressão 2.89, é possível verificar que a conjunção pode ser facilmente representada, utilizando apenas NAND (Expressão 2.90).

$$\begin{aligned} x\ y &= \overline{\overline{x\ y}} \\ &= \overline{\overline{x\ y}\ \overline{x\ y}} \end{aligned} \tag{2.90}$$

Para a representação da disjunção, recorre-se de novo ao teorema da dupla negação e a uma das leis de Morgan (Expressão 2.28), como se ilustra na Expressão 2.91.

$$\begin{aligned} x + y &= \overline{\overline{x + y}} \\ &= \overline{\overline{x}\ \overline{y}} \\ &= \overline{\overline{x\ x}\ \overline{y\ y}} \end{aligned} \tag{2.91}$$

A representação usando logigramas é mais clara, como se pode ver na Figura 2.15.

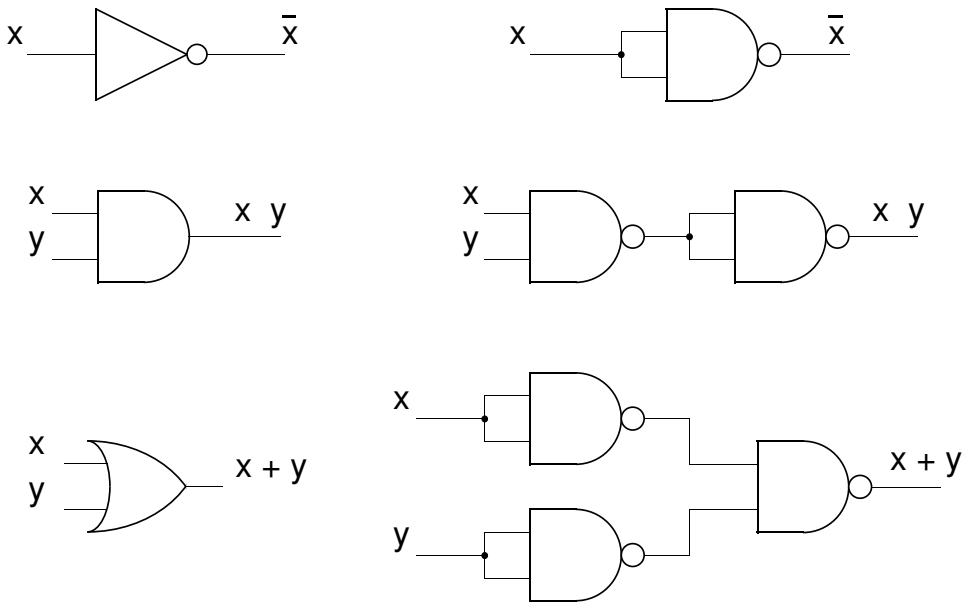


FIGURA 2.15: Representação das funções NOT, AND e OR com NANDs.

É, portanto, possível representar qualquer função por uma expressão lógica, utilizando apenas funções NAND ou NOR. A forma de fazer isso pode conduzir a expressões muito

complexas se se proceder a partir de outra expressão, substituindo cada função negação, conjunção e disjunção por NAND ou NOR, de acordo com as regras apresentadas. Há, contudo, em alguns casos, formas mais simples e estruturadas de obter esse tipo de expressões.

Se uma função estiver representada em termos de forma normal disjuntiva, a aplicação do teorema da dupla negação e, seguidamente, de uma das leis de Morgan conduz directamente à expressão em termos de NAND. Considere-se, a título de exemplo, a função anteriormente estudada, representada pela Expressão 2.72. A metodologia referida conduz à Expressão 2.92.

$$\begin{aligned} f(a,b,c) &= b\ \overline{c} + a\ b \\ &= \overline{\overline{b\ \overline{c} + a\ b}} \\ &= \overline{\overline{b\ \overline{c\ c}}\ \overline{a\ b}} \end{aligned} \tag{2.92}$$

Da Expressão 2.86 por manipulação é fácil encontrar a Expressão 2.93, que representa a mesma função na forma normal conjuntiva, numa forma muito mais simplificada.

$$f(a,b,c) = (a + \overline{b})\ (\overline{b} + \overline{c}) \tag{2.93}$$

A partir desta expressão é fácil obter uma expressão em termos de NOR, como se ilustra na Expressão 2.94.

$$\begin{aligned} f(a,b,c) &= (a + \overline{b})\ (\overline{b} + \overline{c}) \\ &= \overline{\overline{(a + \overline{b})\ (\overline{b} + \overline{c})}} \\ &= \overline{\overline{a + \overline{b + b}} + \overline{\overline{b + b} + \overline{c + c}}} \end{aligned} \tag{2.94}$$

A função negação pode ser considerada como um caso particular de NAND e de NOR. Seria o NAND e o NOR de uma variável. Por isso, quando se pretende representar funções em expressões lógicas só com uma daquelas funções, usam-se, muitas vezes, expressões não transformando as negações. Assim as Expressões 2.92 e 2.94 seriam nesse caso representadas respectivamente pelas Expressões 2.95 e 2.96.

$$f(a,b,c) = \overline{\overline{b\ \overline{c}}\ \overline{a\ b}} \tag{2.95}$$

$$g(a,b,c) = \overline{b + \overline{b} + \overline{b} + \overline{c}} \tag{2.96}$$

Como é evidente, pode aplicar-se a metodologia a uma expressão nas formas canónicas normais. Se se partir da forma canónica normal disjuntiva, obtém-se uma expressão

em termos de NAND, que se designa por *terceira forma canónica*. Se, por outro lado, se partir da forma canónica normal conjuntiva, obtém-se uma expressão, apenas em termos de NOR, designada por *quarta forma canónica*. As Expressões 2.97 e 2.98, em que, para não tornar a expressão mais complexa, não se substituíram as negações, são, respectivamente, a terceira e quarta forma canónica da função $f(a,b,c) = b\ \overline{c} + a\ b$, que temos vindo a estudar.

$$f(a,b,c) = \overline{\overline{a\ b\ c}\ \overline{a\ b\ \overline{c}}\ \overline{\overline{a}\ b\ \overline{c}}} \tag{2.97}$$

$$f(a,b,c) = \overline{\overline{(a+b+c)} + \overline{(a+b+\overline{c})} + \overline{(a+\overline{b}+\overline{c})} + \overline{(\overline{a}+b+c)} + \overline{(\overline{a}+b+\overline{c})}} \tag{2.98}$$

2.3 MINIMIZAÇÃO DE EXPRESSÕES LÓGICAS

Como se discutiu na Secção 2.1.13 é possível, utilizando as regras da álgebra de Boole, tentar obter expressões simplificadas de funções lógicas. Essa simplificação é importante, uma vez que, como se verá, os diversos circuitos em sistemas digitais são implementados a partir de expressões lógicas das funções, e expressões mais simples conduzem a circuitos mais simples. O ideal é mesmo obter as expressões mínimas das funções. Esse processo denomina-se *minimização de funções lógicas*. Para casos complexos de funções com muitas variáveis, pode ser extremamente difícil ou mesmo impossível, mas para funções com um reduzido número de variáveis, a determinação da expressão mínima é relativamente fácil. Como, por vezes, existem várias expressões com a mesma complexidade que não podem simplificadas, usa-se a designação *expressão minimal* e não a de expressão mínima para designar essa formas.

Existem muitos graus de liberdade na obtenção de expressões de funções, como se viu. Assim, aquilo que parece ser uma expressão minimal utilizando um determinado tipo de funções e um determinado tipo de estrutura para a expressão pode ser simplificado se optarmos por outras funções ou estruturas. Os métodos que irão ser apresentados inicialmente aplicam-se à obtenção de expressões em termos de formas normais. Este tipo de expressão designa-se, como já foi referido na Secção 2.2.1, por expressão a dois níveis ou a duas camadas. Há técnicas que permitem a obtenção de expressões a mais de dois níveis, embora não sejam tratadas neste livro.

Um primeiro processo consiste, como é óbvio, na aplicação dos teoremas da álgebra de Boole para reduzir as expressões. No entanto, nem sempre é fácil garantir que se minimiza uma expressão. Isso depende muito da experiência e perícia de quem executa a minimização. Considere-se, por exemplo, a função representada na Expressão 2.99.

$$f(a,b,c,d) = b\ (a \oplus c) + a\ b\ d + b\ c\ (a \oplus d) + \overline{a}\ c\ d + \overline{a}\ c\ \overline{d} \tag{2.99}$$

Para qualquer expressão há em geral muitas formas de atacar a simplificação. Seguidamente exemplifica-se uma dessas formas. Aplicando o Teorema 2.48, obtém-se

$$f(a,b,c,d) = a\ b\ \overline{c} + \overline{a}\ b\ c + a\ b\ d + \overline{a}\ b\ c\ d + a\ b\ c\ \overline{d} + \overline{a}\ c\ d + \overline{a}\ c\ \overline{d} \tag{2.100}$$

Considerando, agora, a distributividade do produto em relação à soma (Expressão 2.20),

$$f(a,b,c,d) = a\ b\ (\overline{c} + d + c\ \overline{d}) + \overline{a}\ b\ c + \overline{a}\ c\ (b\ d + d + \overline{d}) \tag{2.101}$$

Tendo em conta as Expressões 2.18, 2.19 e 2.24, vem

$$f(a,b,c,d) = a\ b + \overline{a}\ b\ c + \overline{a}\ c \tag{2.102}$$

E, por fim (Expressões 2.20 e 2.24),

$$f(a,b,c,d) = a\ b + b\ c + \overline{a}\ c \tag{2.103}$$

A Expressão 2.103 pode parecer uma expressão minimal da função lógica considerada. No entanto, se se aplicar o teorema do consenso (Expressão 2.26), verifica-se que se pode simplificar a expressão um pouco mais, obtendo a Expressão 2.104.

$$f(a,b,c,d) = a\ b + \overline{a}\ c \tag{2.104}$$

que é de facto uma expressão minimal da função.

Este exemplo simples mostra que, embora a manipulação algébrica das expressões permita simplificá-las, seria mais interessante ter disponível um método para obter, de forma sistemática, uma expressão minimal. Na secção seguinte vai estudar-se um primeiro método com esse objectivo.

2.3.1 Método de Karnaugh

Retome-se o exemplo da função $f(a,b,c) = b\ \overline{c} + a\ b$. Como se viu na Secção 2.2.1, essa função é representada pela Tabela 2.11, que se repete aqui no seu essencial. Como se viu, a leitura da tabela produz a Expressão 2.105 em forma canónica normal disjuntiva.

$$f(a,b,c) = \overline{a}\ b\ \overline{c} + a\ b\ \overline{c} + a\ b\ c \tag{2.105}$$

TABELA 2.14: Tabela da função $f(a,b,c) = b\ \overline{c} + a\ b$.

Linha	a	b	c	$f(a,b,c)$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

É possível simplificar esta expressão, de modo a obter a expressão mais simplificada já conhecida. Assim:

$$\begin{aligned} f(a,b,c) &= \overline{a}\ b\ \overline{c} + a\ b\ \overline{c} + a\ b\ c \\ &= \overline{a}\ b\ \overline{c} + a\ b\ \overline{c} + a\ b\ \overline{c} + a\ b\ c \\ &= (\overline{a} + a)\ b\ \overline{c} + a\ b\ (\overline{c} + c) \\ &= b\ \overline{c} + a\ b \end{aligned} \tag{2.106}$$

Do mesmo modo que a representação de funções pode ser feita indiferentemente de uma forma tabular ou com uma expressão lógica, o próprio processo de simplificação pode ser feito dos dois modos. Repare-se na forma como se chegou ao termo $a\ b$ na Expressão 2.106. Isso conseguiu-se por junção dos mintermos $a\ b\ \overline{c}$ e $a\ b\ c$, mintermos 6 e 7 que correspondem às linhas 6 e 7 da tabela.

Mas, na tabela, podemos observar que essas linhas têm uma particularidade: em ambas a função vale 1 e, por outro lado, são as duas únicas linhas da tabela em que a e b são simultaneamente 1. A diferença entre as duas está na variável c. Logo, pode concluir-se que, nesta função, basta que a e b sejam 1 para que a função assuma o valor 1. Daí pode concluir-se que $f(a,b,c) = a\ b + \cdots$ Podia ser feito o mesmo raciocínio para o outro produto. Mas, aqui, as linhas em causa são a 2 e a 6. É fácil de ver que, aqui, basta que b seja 1 e c seja 0 para concluir que a função vale 1.

Podemos então simplificar directamente a função por observação da tabela, associando linhas em que a função assuma o valor 1 e que difiram apenas de uma variável: no primeiro produto associámos as linhas 6 e 7. que diferem apenas na variável c, e no segundo associámos 2 e 6, que diferem apenas na variável a. Definem-se como *adjacentes*, linhas que diferem apenas de uma variável.

Mapa de Karnaugh de Três Variáveis

Para aplicação da metodologia apresentada, é claro que facilitaria a leitura da expressão a partir da tabela, como aconteceu no primeiro caso, que todas as posições que diferem apenas no valor de uma variável estivessem fisicamente adjacentes. Mas, para funções de três variáveis, cada posição tem sempre três posições adjacentes, e isso impossibilita que, numa tabela com a estrutura da Tabela 2.14, todas as adjacências estejam contíguas. Isso levou a alterar a estrutura, de modo que se satisfizessem estes requisitos, obtendo-se uma nova organização em que cada célula da tabela é adjacente a células que diferem apenas no valor de uma variável. Esse novo tipo de tabela é o *mapa de Karnaugh*. Uma possível estrutura de um mapa de Karnaugh para funções de três variáveis a, b e c está representada na Figura 2.16.

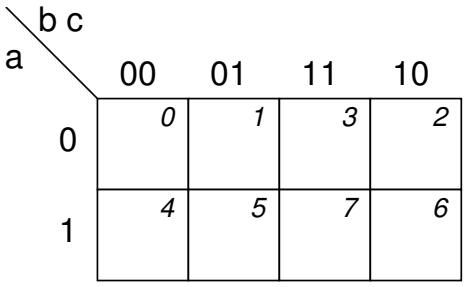


Figura 2.16: Mapa de Karnaugh de três variáveis.

O mapa de Karnaugh é uma tabela de duas entradas, em que a codificação das diversas variáveis em cada uma das duas dimensões das entradas é feita segundo o código reflectido. Isso conduz directamente a que cada posição na tabela tenha sempre fisicamente justapostas, quer na horizontal, quer na vertical, posições adjacentes. A contrapartida é uma maior complexidade na numeração das posições da tabela. No mapa ilustrado na Figura 2.16 está indicado, para cada posição, o número da linha da tabela de verdade correspondente.

Repare-se que, agora, cada posição tem justapostas as três posições adjacentes. Por exemplo, se considerarmos o mintermo $m_5 = a\ \overline{b}\ c$, correspondente à posição 5 na tabela e no mapa de Karnaugh, as três posições adjacentes correspondem a $m_1 = \overline{a}\ \overline{b}\ c$, que se encontra no mapa na posição imediatamente acima, $m_7 = a\ b\ c$, que se encontra no mapa na posição imediatamente à direita, e $m_4 = a\ \overline{b}\ \overline{c}$, na posição imediatamente à esquerda no mapa.

No mapa de Karnaugh, tudo se passa como se as posições laterais estivessem encostadas, o que se conseguiria com o mapa desenhado sobre um cilindro de eixo vertical. Por exemplo, o mintermo m_0 corresponde à posição 0 do mapa cujas posições adjacentes

são a posição 4, imediatamente abaixo, a posição 1, imediatamente à direita, e a posição 2 (que seria à esquerda, isto é, que se situa na outra extremidade da linha do mapa). Uma outra maneira de visualizar as adjacências no mapa de Karnaugh é relacioná-las com eixos de simetria correspondentes à posição dos «espelhos» no código reflectido. Na Figura 2.17 repete-se o mapa da Figura 2.16, assinalando os eixos de simetria.

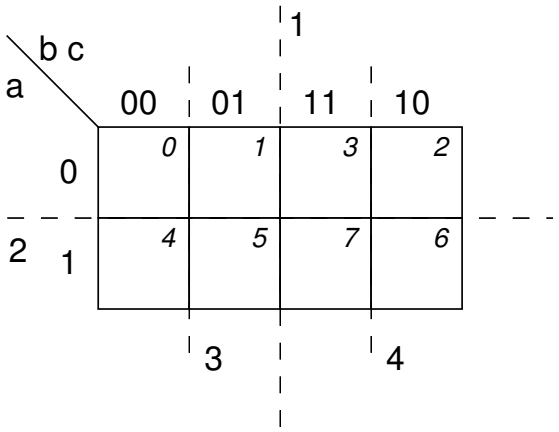


FIGURA 2.17: Mapa de Karnaugh de três variáveis assinalando os eixos de simetria.

O eixo 1 evidencia a simetria entre as posições indicadas nos seguintes pares: $(0, 2)$, $(1, 3)$, $(4, 6)$ e $(5, 7)$. O eixo 2 evidencia a simetria entre as posições agrupadas nos pares $(0, 4)$, $(1, 5)$, $(3, 7)$ e $(2, 6)$. O eixo 3 está ligado aos pares $(0, 1)$ e $(4, 5)$, e o eixo 4, aos pares $(2, 3)$ e $(6, 7)$.

A função que tem vindo a considerar-se está ilustrada no mapa representado na Figura 2.18.

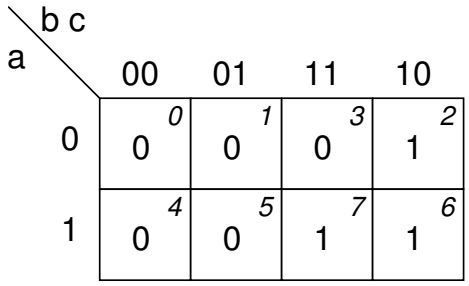


FIGURA 2.18: Mapa de Karnaugh da função $f(a, b, c)$.

É agora possível observar no mapa as adjacências e marcá-las com a utilização de *laços* que se encontram representados na Figura 2.19. Nessa figura está ainda representada a leitura do produto simplificado correspondente a cada laço.

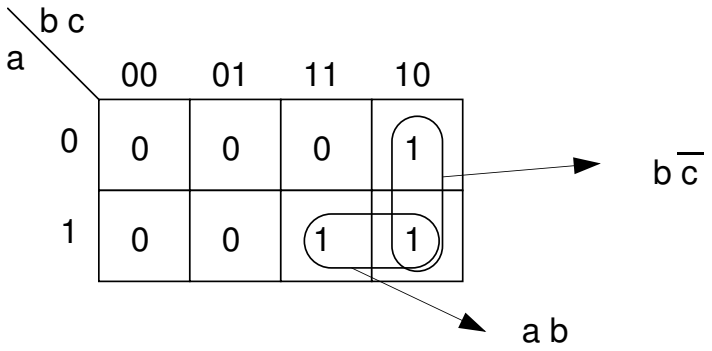


FIGURA 2.19: Leitura do mapa de Karnaugh da função $f(a, b, c)$.

A leitura da expressão de cada produto resulta da observação de quais são as variáveis que se mantêm constantes nas posições abraçadas pelos laços. Por exemplo, no grupo lido como $a\ b$, é fácil de observar que a variável a assume o valor 1 em ambas as posições e que, de igual modo, a variável b assume também o valor 1. A variável c, pelo contrário, vale 0, na posição 6, e 1 na posição 7. Conclui-se que o valor do produto não é sensível à variável c e que só assume o valor 1 quando simultaneamente $a = 1$ e $b = 1$, isto é, quando $a\ b = 1$ e corresponde, portanto, ao produto $a\ b$. De igual modo é fácil verificar que o outro laço assume o valor 1, apenas quando $b = 0$ e $c = 1$, sendo insensível ao valor de a, correspondendo, portanto, ao produto $\overline{b}\ c$. Do mapa lê-se, portanto, directamente a Expressão 2.107.

$$f(a, b, c) = \overline{b}\ c + a\ b \tag{2.107}$$

É, portanto, possível ler directamente do mapa de Karnaugh a expressão simplificada da função.

São válidas no mapa de Karnaugh todas as associações em laços que agrupem duas posições adjacentes não só no sentido formal do termo, mas também no de localização contígua no mapa.

Não é obrigatório que o mapa de Karnaugh tenha as variáveis distribuídas da maneira indicada na Figura 2.16, nem que tenha o formato apresentado na mesma figura. Por um lado, as variáveis podem ser distribuídas de outro modo, desde que utilizando o código binário reflectido e, por outro, o mapa pode ser representado ao alto com 4 linhas de 2 posições ou até noutras configurações. Alguns exemplos alternativos de representação do mapa de Karnaugh para funções de três variáveis estão representados na Figura 2.20. Como é natural, a correspondência entre as posições do mapa e as linhas da tabela de verdade é também alterada e, na mesma figura, essa correspondência é apresentada.

Considere-se outro exemplo, representando a função $g(a, b, c) = \sum m(0, 2, 4, 5, 6)$ num

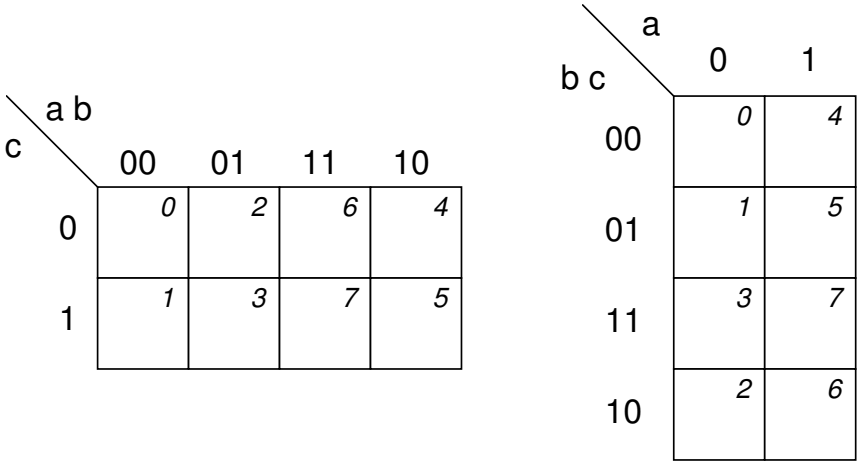


FIGURA 2.20: Variantes de representação do mapa de Karnaugh de três variáveis.

mapa de Karnaugh. A Figura 2.21 ilustra o respectivo mapa. É fácil de ler no mapa a

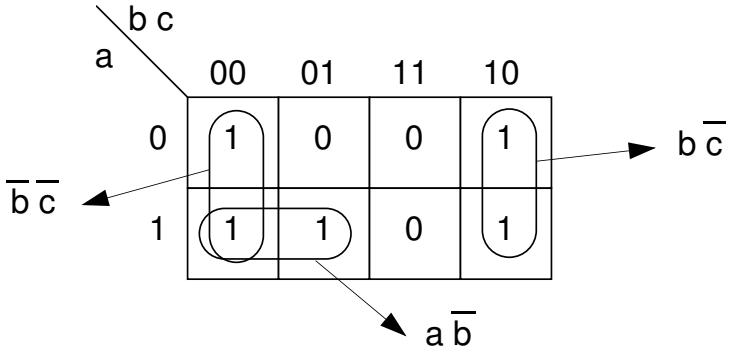


FIGURA 2.21: Mapa de Karnaugh da função $g(a, b, c)$.

função dada pela Expressão 2.108.

$$g(a, b, c) = a\ \overline{b} + \overline{b}\ \overline{c} + b\ \overline{c} \tag{2.108}$$

Como é evidente, porém, os dois termos finais podem ser associados colocando $\overline{c}$ em evidência, obtendo-se a Expressão 2.109 mais simples que a anterior.

$$g(a, b, c) = a\ \overline{b} + \overline{c} \tag{2.109}$$

Ora isso é legível do Mapa de Karnaugh. De facto, os dois grupos de dois 1 nas pontas do mapa são adjacentes e podem juntar-se entre si para originar o mapa representado na Figura 2.22, onde é fácil agora ler directamente a expressão mais simplificada.

Repare-se que a leitura do produto correspondente ao laço de quatro posições obedece à mesma filosofia que no caso de duas posições. De facto, a única variável que é constante

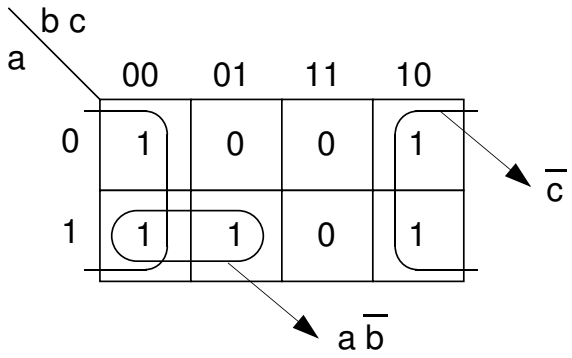


FIGURA 2.22: Segunda leitura do mapa de Karnaugh da função $g(a, b, c)$.

nas quatro posições é a variável c com o valor 0. Assim, o produto fica reduzido a apenas uma variável com o valor $\overline{c}$.

Num mapa de Karnaugh, um laço de quatro posições corresponde à existência de dois laços de duas posições adjacentes entre si. Dois laços são *laços adjacentes* quando cada posição de um deles é adjacente a uma posição diferente do outro e não há posições comuns. Não é possível associar quatro posições que não correspondam a dois laços adjacentes. Os laços a associar têm de corresponder a produtos cuja expressão difere apenas em uma variável representada directamente num deles e negada no outro para se poder, de um ponto de vista algébrico «pôr em evidência».

É assim interessante, quando da utilização do mapa de Karnaugh para minimizar expressões lógicas, procurar obter os laços de maior dimensão possível. Mas, como se verá adiante, este não é um critério absoluto.

É fácil de generalizar este procedimento para associar grupos adjacentes de quatro posições para formar grupos de oito posições. Em mapas correspondentes a funções com número maior de variáveis é possível agrupar laços de oito posições para obter laços de 16 posições, e assim por diante. Repare-se que, uma vez que se agrupam sempre laços com o mesmo número de posições, se obtêm sempre laços com uma dimensão que é uma potência de 2.

Na Figura 2.23 representam-se algumas associações ilegais no mapa de Karnaugh.

Em (a) estão associados dois laços iguais de duas posições, mas os dois grupos não são adjacentes. A posição 1 de um dos grupos é adjacente à posição 3 do outro, mas as restantes posições não o são (a posição 2 não é adjacente à posição 5). Em (b) a situação é semelhante. Assim, a posição 4 do laço da esquerda é adjacente à posição 5 do laço da direita, tal como a posição 5 do da esquerda é adjacente à posição 7 do laço da direita, mas há sobreposição, uma vez que a posição 5 é comum aos dois laços. Finalmente em

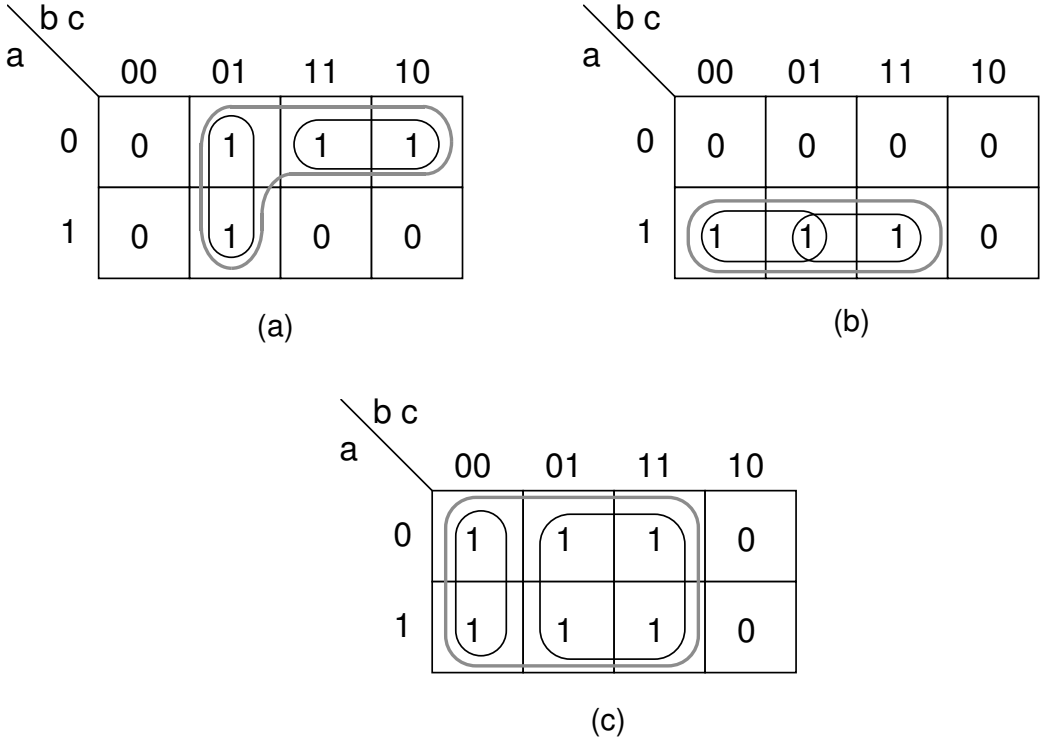


FIGURA 2.23: Associações ilegais no mapa de Karnaugh.

(c) associam-se dois laços de dimensões diferentes.

Até agora apenas se exemplificou a utilização do mapa de Karnaugh para obter expressões na forma normal disjuntiva, associando mintermos. O princípio da dualidade (Secção 2.1.6) sugere, porém, que se poderão associar maxtermos para obter expressões na forma normal conjuntiva. Considere-se, então, de novo o mapa de Karnaugh da função que temos vindo a estudar, procurando, agora, agrupar posições com valor 0. Obtém-se o mapa ilustrado na Figura 2.24.

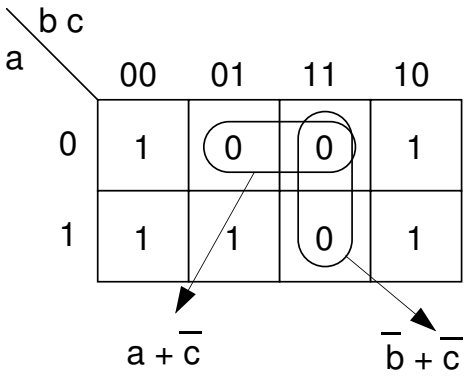


FIGURA 2.24: Mapa de Karnaugh da função $g(a, b, c)$ com associação de maxtermos.

É possível ler agora a Expressão 2.110.

$$g(a,b,c) = (a + \overline{c})\ \ (\overline{b} + \overline{c}) \tag{2.110}$$

Repare-se que, na leitura, tal como já tinha acontecido atrás na Secção 2.2.2, estamos agora a ler produtos (associações de maxtermos), e as variáveis, quando estão a 0, são lidas não negadas e, quando a 1, são lidas negadas.

A Expressão 2.110 é uma expressão minimal na forma normal conjuntiva. É evidente que, para cada função, podem obter-se dois tipos de expressões minimais. Trata-se de expressões minimais em termos de cada uma das formas normais. A função que temos vindo a discutir tem uma expressão minimal, na forma normal disjuntiva (Expressão 2.109), e outra expressão minimal, na forma normal conjuntiva (Expressão 2.110).

Mapa de Karnaugh de Quatro Variáveis

O mapa de Karnaugh foi apresentado para três variáveis, mas pode ser usado, em teoria, com qualquer número de variáveis. Na prática, com mais de seis variáveis torna-se muito difícil e há outros métodos, de que veremos um exemplo na Secção 2.3.2, que, no entanto, assentam nos mesmos princípios do método agora apresentado. O mapa de quatro variáveis constrói-se facilmente. A partir de um mapa de três variáveis, replica-se o quadro através de uma reflexão num espelho imaginário, obtendo-se um quadro com quatro linhas e quatro colunas. O espelho imaginários referido constitui um novo eixo de simetria do mapa. As configurações de variáveis correspondem a um código reflectido, como se ilustra na Figura 2.25, em que se indica ainda a linha correspondente na tabela de verdade, para cada posição.

Para além de todas as adjacências que já existiam, passam a ser válidas as adjacências determinadas pelo novo eixo de simetria introduzido. Na Figura 2.26 ilustram-se os eixos de simetria. Na Figura 2.27 mostram-se alguns exemplos de associações válidas no mapa de 4 variáveis.

Considere-se, a título de exemplo, a função $f(a,b,c,d) = \sum m(2,3,5,7,9,11,14,15)$ de quatro variáveis. A Figura 2.28 ilustra o correspondente mapa de Karnaugh. No mapa é fácil ler a Expressão 2.111.

$$f(a,b,c,d) = \overline{a}\ \ \overline{b}\ \ c + \overline{a}\ \ b\ \ d + a\ \ b\ \ c + a\ \ \overline{b}\ \ d \tag{2.111}$$

À primeira vista poderia parecer interessante juntar os quatro valores 1 da coluna assinalada num laço único, que corresponderia ao produto $c\ \ d$. No entanto, para cobrir os

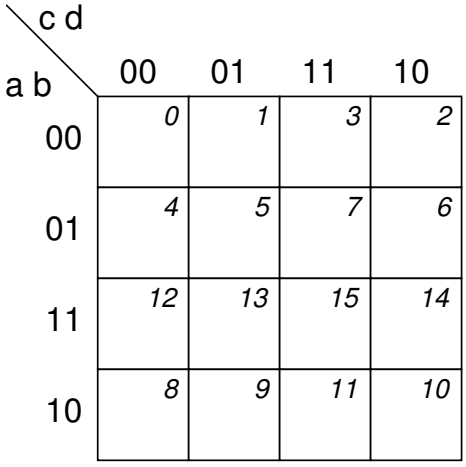


FIGURA 2.25: Mapa de Karnaugh de quatro variáveis.

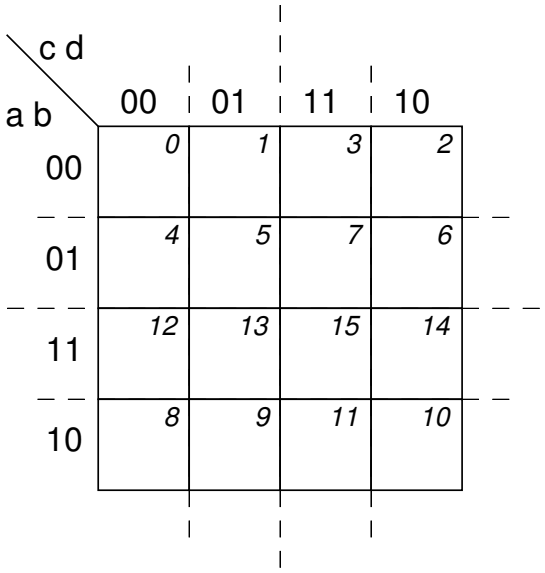


FIGURA 2.26: Eixos de simetria no mapa de Karnaugh de quatro variáveis.

restantes valores 1, haveria necessidade de associar cada um deles com um dos valores 1 do laço vertical. Verifica-se, portanto, que esse laço seria redundante. Não é verdade, portanto, que, neste método de minimização, se deva sempre optar pelos laços de maior dimensão. Adiante se regressará a esta questão.

Analise-se outro exemplo, partindo agora da especificação da função por uma sua expressão lógica. Considere-se a função representada pela Expressão 2.112.

$$f(a,b,c,d) = \overline{a}\ \overline{b}\ \overline{c} + \overline{a}\ (c \oplus d) + \overline{a}\ b\ \overline{c}\ \overline{d} + \overline{b}\ (c \oplus d) + a\ \overline{b}\ c \qquad (2.112)$$

A construção do mapa de Karnaugh para esta função assenta no processo inverso ao

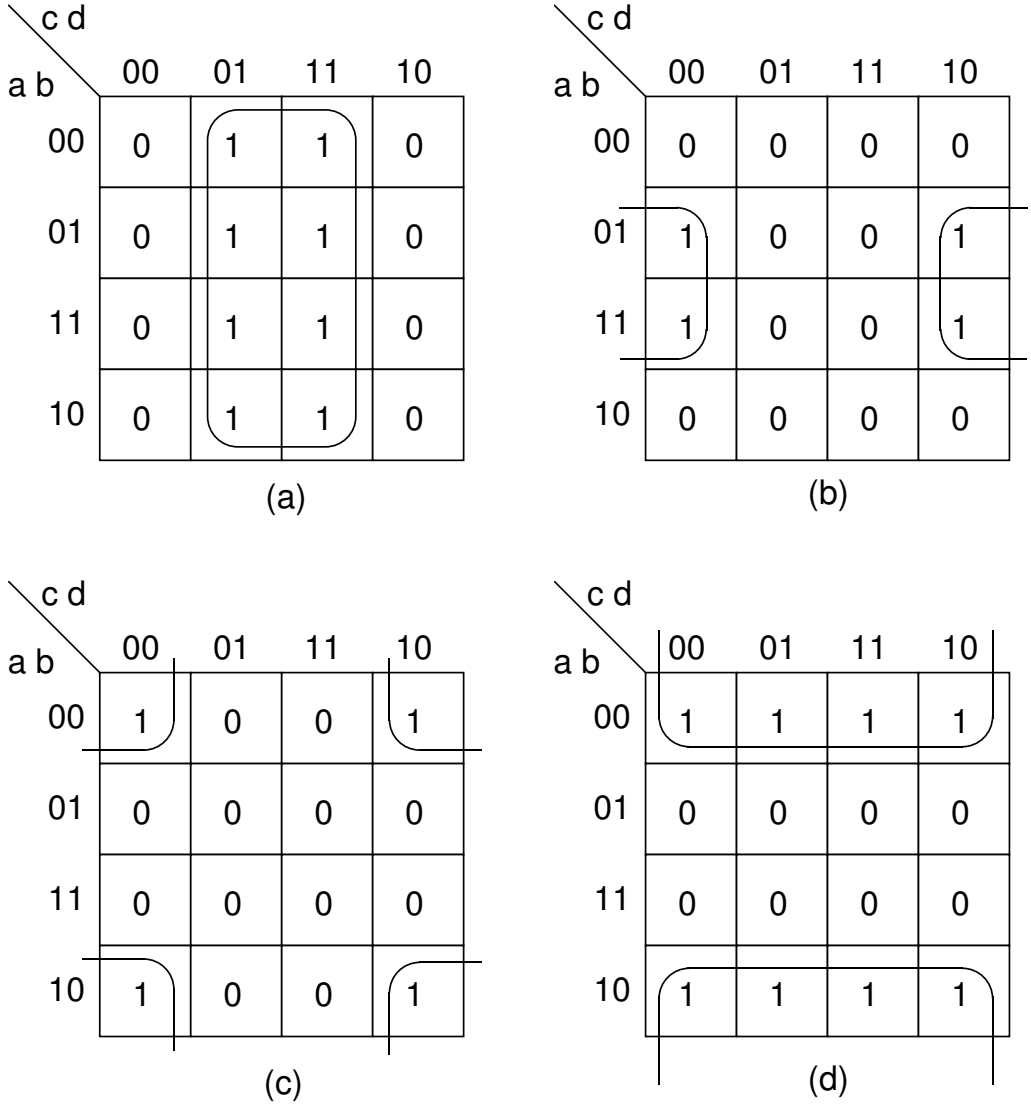


FIGURA 2.27: Exemplos de associações válidas no mapa de Karnaugh.

da leitura da expressão a partir do mapa. Para cada termo são assinaladas no mapa as posições que têm valor 1 para esse termo. Por exemplo, o primeiro termo $\overline{a}\ \overline{b}\ \overline{c}$ corresponde no mapa à existência de valores 1 nas posições em que $a = b = c = 0$, como se assinala no mapa da Figura 2.29.

A marcação no mapa das posições correspondentes ao segundo termo parece mais difícil, uma vez que surge no termo uma disjunção exclusiva. O problema pode resolver-se aplicando o Teorema 2.48 e desdobrando o termo $\overline{a}\ (c \oplus d)$ em dois produtos. No entanto, é possível raciocinar directamente sobre o termo, se se tiver em conta que uma disjunção exclusiva é uma função que apenas exibe o valor 1 quando as duas variáveis são diferentes. Assim, o termo em causa corresponde às posições em que, sendo a variável $a = 0$, é simultaneamente verdade que as variávais c e d são diferentes. Isso

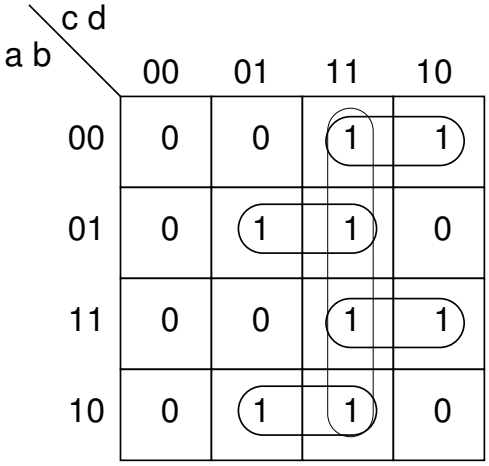


FIGURA 2.28: Mapa de Karnaugh da função $f(a, b, c, d)$.

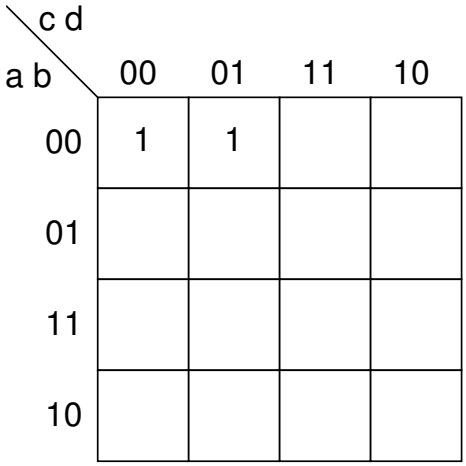


FIGURA 2.29: Mapa de Karnaugh com o termo $\overline{a}\ \overline{b}\ \overline{c}$ assinalado.

corresponde as posições assinaladas no mapa da Figura 2.30.

É fácil, portanto, construir o mapa de Karnaugh que representa a função em estudo, que se ilustra na Figura 2.31.

O mapa tem assinalados um conjunto de laços que permite ler a Expressão 2.113, que é uma expressão minimal.

$$f(a, b, c, d) = \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + a\ \overline{b}\ d + a\ \overline{b}\ c \tag{2.113}$$

A expressão obtida é uma expressão minimal, mas não é única. De facto, se, em vez de se associar o mintermo m_{10} com o m_{11}, se associarem, por exemplo, os mintermos m_{10} e m_2, como se assinala na Figura 2.32, poderá obter-se uma nova expressão para a função, a Expressão 2.114.

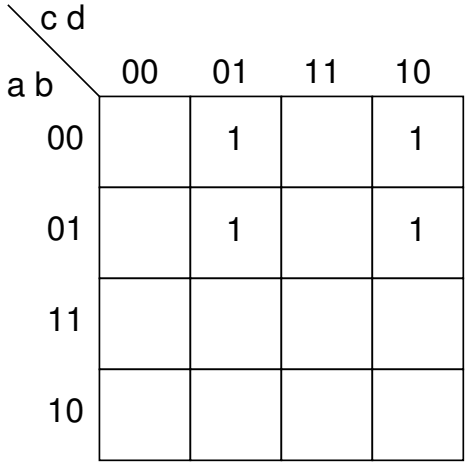


FIGURA 2.30: Mapa de Karnaugh com o termo $\overline{a}\ (c \oplus d)$ assinalado.

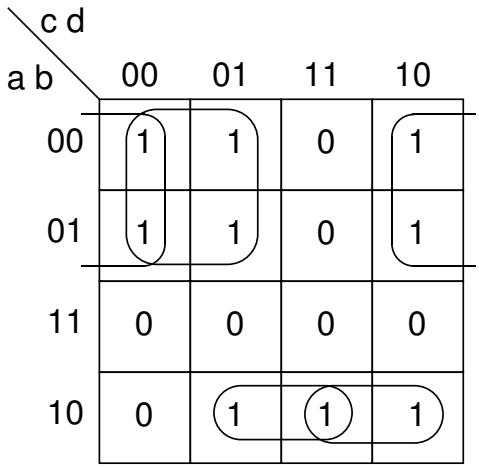


FIGURA 2.31: Mapa de Karnaugh representando a função em estudo.

$$f(a,b,c,d) = \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + a\ \overline{b}\ d + \overline{b}\ c\ \overline{d} \tag{2.114}$$

Existe ainda uma terceira variante com a Expressão 2.115, como se pode facilmente verificar.

$$f(a,b,c,d) = \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + \overline{b}\ \overline{c}\ d + a\ \overline{b}\ c \tag{2.115}$$

As três expressões têm o mesmo grau de complexidade e são todas tomadas como expressões minimais da função. Existem, portanto, funções com várias expressões minimais.

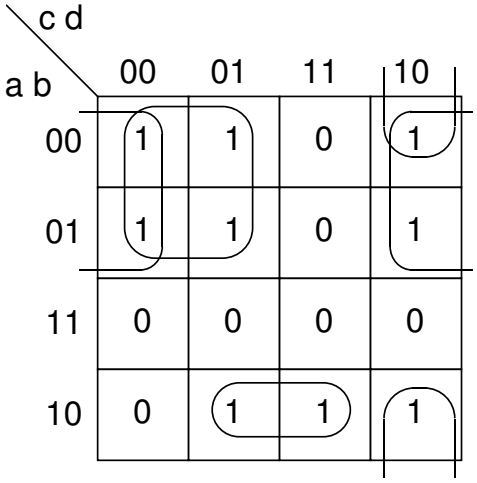


FIGURA 2.32: Mapa de Karnaugh alternativo representando a função em estudo.

FUNDAMENTAÇÃO DO MÉTODO DE KARNAUGH

Na continuação, o método de Karnaugh vai ser estudado com um pouco mais de profundidade, e as questões que foram sendo deixadas em aberto vão ser analisadas numa perspectiva mais integrada. Uma melhor compreensão dos mecanismos apresentados vai permitir definir procedimentos mais adequados ao objectivo de minimizar expressões lógicas.

O primeiro conceito importante é o de *implicação entre funções*. Para funções das mesmas variáveis, diz-se que uma função f_1 implica outra f_2 com a representação $f_1 \Rightarrow f_2$, quando, para todas as configurações de entrada em que a função f_1 vale 1, a função f_2 também vale 1. Quando f_1 é um produto, diz-se ser um *implicante* da função f_2.

Num mapa de Karnaugh, de uma função f, todos os produtos correspondentes a associações válidas de posições com valor 1 são implicantes. Por exemplo, na função representada no mapa de Karnaugh da Figura 2.28, que se repete na Figura 2.33 por conveniência, o produto $a\ b\ c$ é um implicante da função $f(a,b,c,d)$, o que pode ser indicado por $a\ b\ c \Rightarrow f(a,b,c,d)$. Do mesmo modo, o produto $c\ d$ é um implicante da função. Repare-se que o produto $a\ c\ d$, não representado por um laço no mapa, é igualmente um implicante da função $f(a,b,c,d)$. De igual modo se pode observar que $a\ c\ d \Rightarrow c\ d$. Repare-se, ainda, que todos os mintermos da função são implicantes dessa função. A implicação goza de *transitividade*, isto é, se $f_1 \Rightarrow f_2$ e $f_2 \Rightarrow f_3$, então $f_1 \Rightarrow f_3$.

O mintermo $m_3 = \overline{a}\ \overline{b}\ c\ d$ é um dos implicantes do produto $\overline{a}\ \overline{b}\ c$ e da função $f(a,b,c,d)$. Isto é, $m_3 \Rightarrow \overline{a}\ \overline{b}\ c \Rightarrow f(a,b,c,d)$.

Um implicante que não implica nenhum outro implicante designa-se por *implicante*

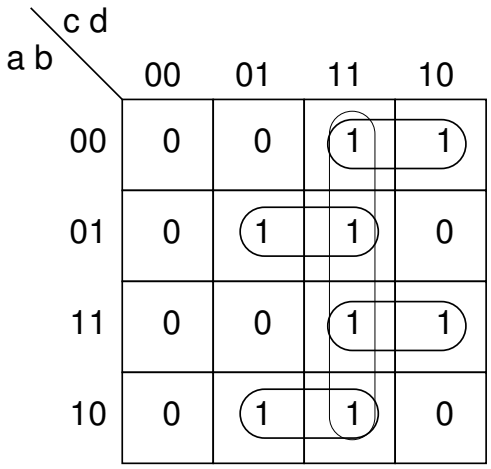


FIGURA 2.33: Mapa de Karnaugh da função $f(a, b, c, d)$.

primo ou *implicante principal*. No exemplo que tem vindo a utilizar-se, o implicante $\overline{a}\ \overline{b}\ c$ é um implicante primo, tal como o implicante $c\ d$. Pelo contrário, o implicante $a\ c\ d$ não é um implicante primo, uma vez que $a\ c\ d \Rightarrow c\ d$. A importância dos implicantes primos decorre de eles corresponderem, no mapa de Karnaugh, às associações maiores, que não podem ser expandidas. Ora acontece que esses são exactamente os grupos que nos interessam para as simplificações. Daí que a expressão algébrica minimizada de uma função expressa na forma normal disjuntiva é sempre uma soma de implicantes primos.

No mapa da função que tem sido usada como exemplo, estão assinalados todos os implicantes primos da função. Porém, como se viu, nem todos os implicantes primos da função foram usados na expressão minimizada. Só foram utilizados os implicantes necessários para incluir todos os mintermos da função ou, em termos do mapa de Karnaugh, foram usadas as associações necessárias para incluir todos os 1 do mapa. Os implicantes usados foram os quatro seguintes: $\overline{a}\ \overline{b}\ c$, $\overline{a}\ b\ d$, $a\ b\ c$ e $a\ \overline{b}\ d$.

Repare-se que cada um dos implicantes usados tem uma particularidade fundamental: o implicante primo $\overline{a}\ \overline{b}\ \overline{c}$, por exemplo, é o único que associa o mintermo m_2. Do mesmo modo, cada um dos outros implicantes primos associa um mintermo que não pode ser associado de outra forma. Ao contrário, o implicante primo $c\ d$ (que acabou por não ser usado) associa apenas mintermos que podem ser associados de outra forma. Os implicantes primos que associam mintermos que não podem ser associados em implicantes primos de outra forma são chamados *implicantes primos essenciais*. No exemplo que tem vindo a ser usado, os quatro produtos usados na expressão lógica da função (Expressão 2.116) são, portanto, implicantes primos essenciais.

$$f(a, b, c, d) = \overline{a}\ \overline{b}\ c + \overline{a}\ b\ d + a\ b\ c + a\ \overline{b}\ d \tag{2.116}$$

Como se viu já, a expressão algébrica de uma função em termos de soma de produtos é uma soma de implicantes primos. Mas não têm de ser usados todos os implicantes primos da função. No entanto, todos os implicantes primos essenciais têm de estar presentes na expressão. Se um implicante primo essencial não estivesse presente na expressão da função, isso significaria que pelo menos um mintermo não estaria incluído, e a função representada pela expressão diferiria da função a representar pelo menos na configuração de variáveis de entrada correspondente a esse(s) mintermo(s).

Convém referir que, embora neste exemplo os únicos implicantes primos usados sejam implicantes primos essenciais, isso não é uma regra. Considere-se, agora, a função representada pelo mapa de Karnaugh da Figura 2.31.

Os implicantes primos essenciais desta função são: $\overline{a}\ \overline{d}$, que é o único implicante primo a incluir o mintermo m_6 e $\overline{a}\ \overline{c}$, que é o único implicante primo a incluir m_5. A função possui ainda quatro implicantes primos não essenciais: $a\ \overline{b}\ d$, $a\ \overline{b}\ c$, $\overline{b}\ \overline{c}\ d$ e $\overline{b}\ c\ \overline{d}$. As três possíveis expressões minimais da função incluem os implicantes primos essenciais e, cada uma delas, dois dos implicantes primos não essenciais, como se pode rever nas Expressões 2.117.

$$\begin{aligned} f(a,b,c,d) &= \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + a\ \overline{b}\ d + a\ \overline{b}\ c \\ f(a,b,c,d) &= \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + a\ \overline{b}\ d + \overline{b}\ c\ \overline{d} \\ f(a,b,c,d) &= \overline{a}\ \overline{c} + \overline{a}\ \overline{d} + \overline{b}\ \overline{c}\ d + a\ \overline{b}\ c \end{aligned} \tag{2.117}$$

Convém também chamar a atenção para o facto de que pode ocorrer a existência de funções sem implicantes primos essenciais. Considere-se o exemplo da Figura 2.34. A função representada tem duas possíveis expressões minimais representadas pelas Expressões 2.118 e 2.119 sem quaisquer implicantes primos essenciais.

$$f(a,b,c,d) = \overline{a}\ \overline{b}\ c + \overline{a}\ b\ d + a\ b\ \overline{c} + a\ \overline{b}\ \overline{d} \tag{2.118}$$

$$f(a,b,c,d) = a\ \overline{c}\ \overline{d} + b\ \overline{c}\ d + \overline{a}\ c\ d + \overline{b}\ c\ \overline{d} \tag{2.119}$$

O princípio da dualidade dá suporte ao desenvolvimento de uma visão dual da exposta em termos de utilização dos zeros da função, *funções implicadas*, maxtermos, *implicados*, *implicados primos* e *implicados primos essenciais*. Sugere-se, como exercício, a repetição da análise dos dois exemplos anteriores neste novo contexto.

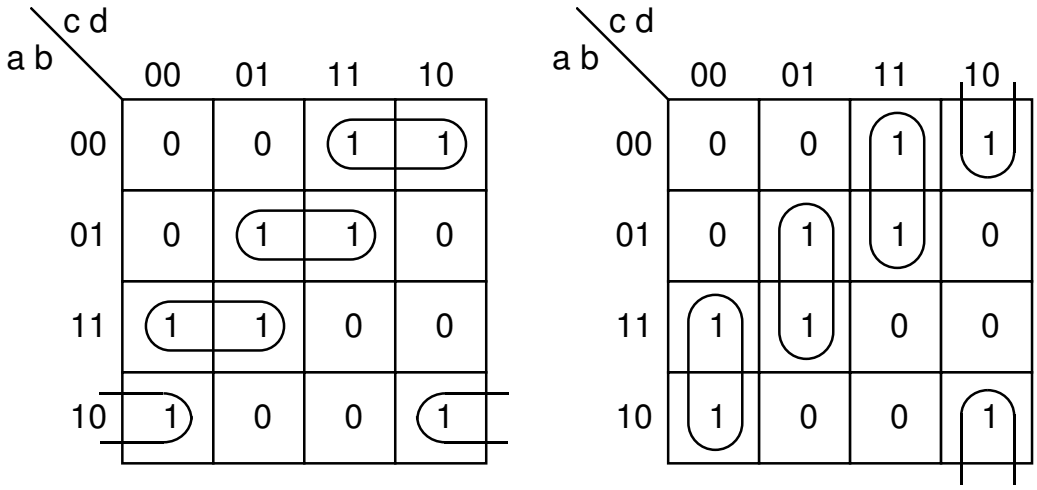


FIGURA 2.34: Exemplo de uma função sem implicantes primos essenciais.

FUNÇÕES COM INDIFERENÇAS

Por vezes acontece que, numa função lógica, certas configurações de entradas nunca ocorrem. É possível tirar partido desse facto para, em muitos casos, minimizar adicionalmente a expressão algébrica da função. Ir-se-á estudar um exemplo, ao longo do qual se exporá a metodologia a usar. Uma outra forma de perspectivar esta situação é a de considerar que as funções nessas circunstâncias são *funções incompletamente especificadas*.

Considere-se que se pretende obter uma função que tem como variáveis de entrada os quatro bits de uma representação em código BCD e que deve dar saída 1, apenas quando o número representado for múltiplo positivo de 3. Há quatro variáveis de entrada, A_3, A_2, A_1 e A_0 que representam os quatro bits do código BCD, representando os índices das variáveis o peso dos respectivos bits. A Tabela 2.15 representa a função pretendida.

Repare-se que há três situações diferentes:

- Números BCD que são múltiplos positivos de 3 ($f = 1$)

- Números BCD que não são múltiplos positivos de 3 ($f = 0$)

- Configurações que não são BCD.

Neste último caso, não é importante considerar o valor da função, uma vez que as configurações de entrada respectivas nunca ocorrem por não serem algarismos BCD e, portanto, o valor que a função teria nessa situação é indiferente. É habitual referir o valor assumido pela função nestes casos como *indiferença* (em inglês, *don't care*). Isso equivale

TABELA 2.15: Tabela da função detectora de múltiplos de 3 em BCD.

BCD	A_3	A_2	A_1	A_0	f	Observações
0	0	0	0	0	0	Não é múltiplo positivo de 3
1	0	0	0	1	0	Não é múltiplo positivo de 3
2	0	0	1	0	0	Não é múltiplo positivo de 3
3	0	0	1	1	1	É múltiplo positivo de 3
4	0	1	0	0	0	Não é múltiplo positivo de 3
5	0	1	0	1	0	Não é múltiplo positivo de 3
6	0	1	1	0	1	É múltiplo positivo de 3
7	0	1	1	1	0	Não é múltiplo positivo de 3
8	1	0	0	0	0	Não é múltiplo positivo de 3
9	1	0	0	1	1	É múltiplo positivo de 3
-	1	0	1	0	-	Não é BCD
-	1	0	1	1	-	Não é BCD
-	1	1	0	0	-	Não é BCD
-	1	1	0	1	-	Não é BCD
-	1	1	1	0	-	Não é BCD
-	1	1	1	1	-	Não é BCD

a dizer que, para essa configuração de entrada, a função não está especificada. Inicialmente não se terá isso em conta e vai obter-se a expressão minimal da função com valores 0 nessas posições (a opção mais conservadora). O mapa de Karnaugh correspondente está ilustrado na Figura 2.35.

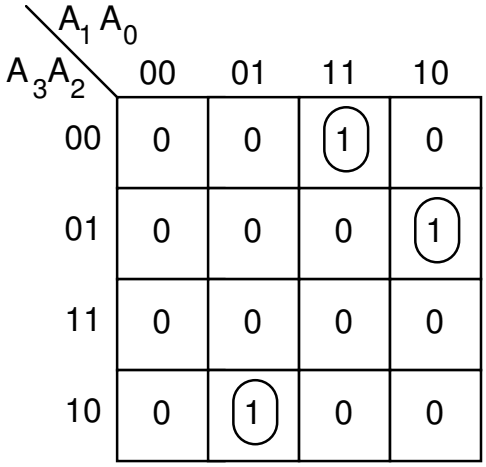


FIGURA 2.35: Mapa de Karnaugh da função detectora de múltiplos de 3 em BCD.

Verifica-se que não há associações possíveis no mapa e que todos os mintermos são, simultaneamente, implicantes primos essenciais. A expressão minimal da função é, portanto, a sua forma normal disjuntiva representada na Expressão 2.120.

$$f(A_3, A_2, A_1, A_0) = \overline{A_3}\ \overline{A_2}\ A_1\ A_0 + \overline{A_3}\ A_2\ A_1\ \overline{A_0} + A_3\ \overline{A_2}\ \overline{A_1}\ A_0 \tag{2.120}$$

Mas experimente-se agora protelar para mais tarde a atribuição de valores à função nas configurações que não correspondem a BCD. Obtém-se um mapa de Karnaugh, em que se assinalam, para além das posições com valores 0 e 1, as posições com indiferenças, representadas por X. O mapa assim construído está representado na Figura 2.36.

A_3A_2 \ A_1A_0	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	X	X	X	X
10	0	1	X	X

FIGURA 2.36: Mapa de Karnaugh da função detectora de múltiplos de 3 em BCD com as indiferenças assinaladas.

Considere-se, agora, o mintermo m_9. Se, nas posições 11, 13 e 15, correspondentes a indiferenças, o valor da função fosse 1, poder-se-iam associar os quatro mintermos respectivos, simplificando consideravelmente a expressão. Mas como, de facto, o valor que a função toma nestas posições é indiferente, é possível colocar lá os valores que mais convêm à simplificação, sem alterar a funcionalidade pretendida. O mesmo acontece para outras posições no mapa.

Pode, então, fazer-se a minimização baseada nos agrupamentos ilustrados no mapa da Figura 2.37, obtendo-se a Expressão 2.121.

$$f(A_3, A_2, A_1, A_0) = A_3\ A_0 + A_2\ A_1\ \overline{A_0} + \overline{A_2}\ A_1\ A_0 \tag{2.121}$$

Como se vê, é muito mais simples a expressão obtida por se ter recorrido à flexibilidade que existe por serem certas posições indiferentes. Tenha-se em conta que a função descrita pela Expressão 2.121 deixou de ter posições não definidas. As configurações de

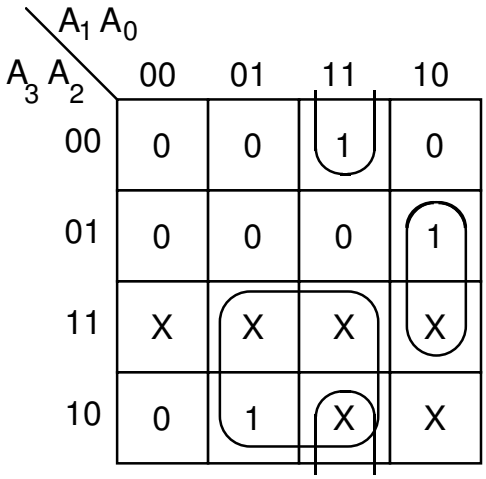


FIGURA 2.37: Mapa de Karnaugh da função detectora de múltiplos de 3 em BCD com as indiferenças aproveitadas.

variáveis de entrada correspondentes às indiferenças que foram associadas com mintermos da função passaram a ter saída 1 da função. As não associadas passaram a ter saída 0, uma vez que foram, de facto, tratadas como 0.

Esta técnica de utilização das indiferenças é generalizadamente usada para minimizar as expressões das funções lógicas, sempre que possível.

MAPA DE KARNAUGH DE CINCO VARIÁVEIS

A obtenção de mapas de Karnaugh de cinco variáveis faz-se a partir de um mapa de quatro variáveis, do mesmo modo que este se obteve a partir de um mapa de três variáveis. Ilustra-se, na Figura 2.38, um mapa de cinco variáveis mostrando também a numeração das diversas posições, assumindo que a variável de maior peso é a a e a de menor peso é a e e ainda os eixos de simetria do mapa. Tal como acontece nos mapas de funções de menor número de variáveis, há muitas outras hipóteses de construir o mapa, colocando as variáveis noutras posições e, naturalmente, numerando as posições de maneira coerente.

Para além de todas as adjacências válidas no mapa de 4 variáveis (em cada uma das «metades»), existem agora adjacências entre posições simétricas em relação ao novo eixo de simetria vertical.

À medida que o número de variáveis aumenta, a complexidade do mapa aumenta da mesma maneira, mas, sobretudo, aumenta a dificuldade em encontrar os agrupamentos adequados para realizar a minimização da função. Considere-se então um exemplo,

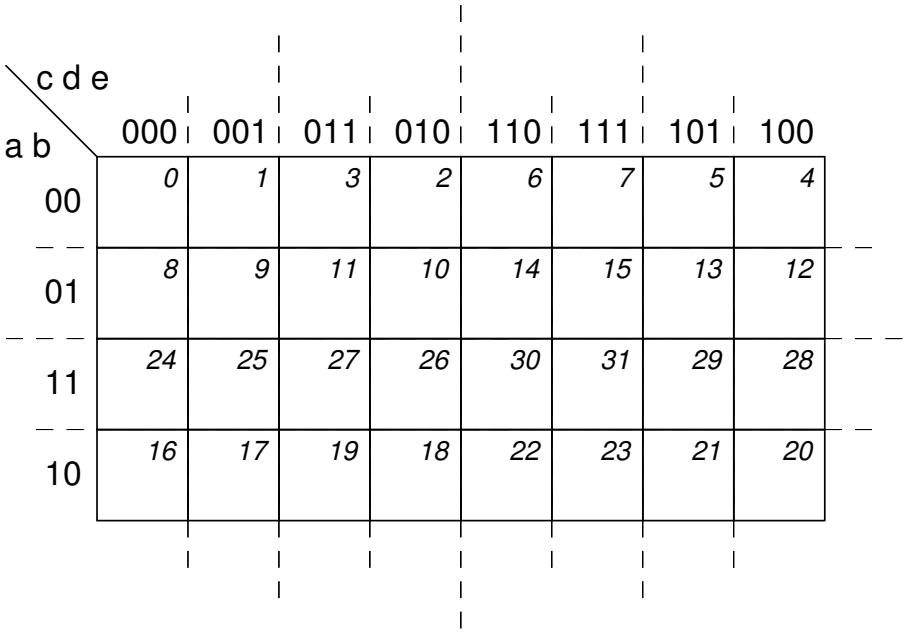


FIGURA 2.38: Mapa de Karnaugh de cinco variáveis.

utilizando o mapa de Karnaugh para minimizar a função dada pela Expressão 2.122 em termos de soma de mintermos.

$$f(a,b,c,d,e) = \sum m(1,3,5,6,7,8,9,12,15,20,21,22,29,30,31) \tag{2.122}$$

Considere-se, ainda, que a função tem indiferenças para as configurações de entrada correspondentes às posições $10, 17, 24, 25$ e 27. Isto é muitas vezes indicado na expressão da função por um somatório de termos designados por md [2]. Nessa hipótese de representação, a função em causa seria representada pela Expressão 2.123.

$$\begin{aligned} f(a,b,c,d,e) &= \sum m(1,3,5,6,7,8,9,12,15,20,21,22,29,30,31) + \\ &\quad + \sum md(10,17,24,25,27) \end{aligned} \tag{2.123}$$

O mapa de Karnaugh correspondente a esta função está ilustrado na Figura 2.39.

A melhor estratégia para atacar o problema encontrando os agrupamentos adequados é, naturalmente, começar pela procura dos implicantes primos essenciais. Se existirem, terão de ser considerados e, para além disso, ficam já cobertos vários dos mintermos da função. Analisando o mapa, é possível verificar que quase todos os mintermos se podem agrupar pelo menos de duas formas em implicantes primos. Há duas excepções, que estão assinaladas na Figura 2.40 com fundo cinzento. O mintermo m_3 apenas se

[2] o "d" provêm da designação em inglês, *don't care*

a b \ c d e	000	001	011	010	110	111	101	100
00	0	1	1	0	1	1	1	0
01	1	1	0	X	0	1	0	1
11	X	X	X	0	1	1	1	0
10	0	X	0	0	1	0	1	1

FIGURA 2.39: Mapa de Karnaugh de uma função de cinco variáveis.

pode agrupar no implicante primo $\overline{a}\ \overline{b}\ e$, assinalado com a referência «1» na figura. Por sua vez, o mintermo m_{12} apenas pode agrupar-se no implicante primo $\overline{a}\ b\ \overline{d}\ \overline{e}$ (referência «2»). Repare-se que os implicantes primos essenciais, neste exemplo, não agrupam indiferenças, mas tal não é o caso geral.

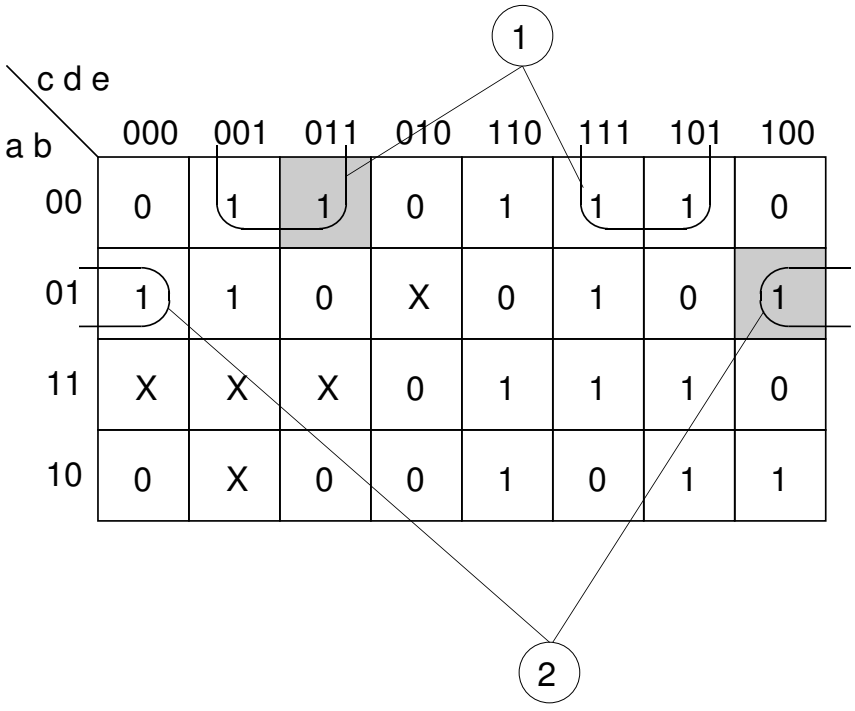


FIGURA 2.40: Mapa de Karnaugh de uma função de cinco variáveis com os implicantes primos essenciais assinalados.

A minimização prossegue com a determinação de um conjunto mínimo de implicantes

primos, correspondentes a agrupamentos da maior dimensão possível, que cubra todos os mintermos. Esta procura pode conduzir ao mapa da Figura 2.41.

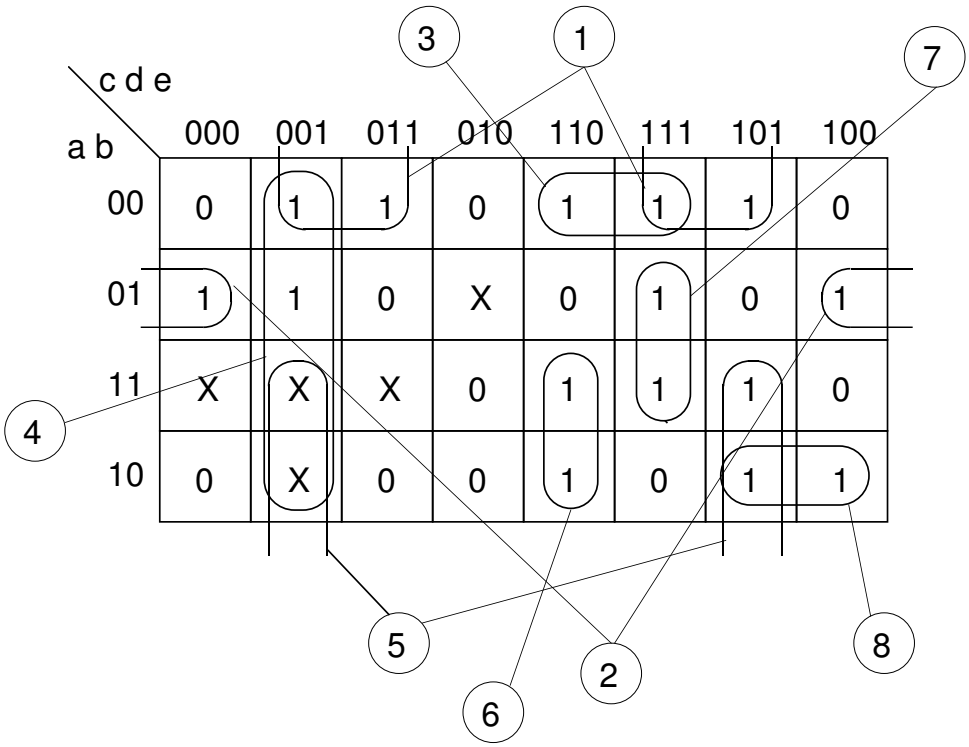


FIGURA 2.41: Mapa de Karnaugh de uma função de cinco variáveis com um conjunto mínimo de implicantes primos assinalados.

Na figura, numeraram-se os implicantes primos, para facilitar a sua identificação. Do trabalho feito resulta a Expressão 2.124, em que os termos estão pela mesma ordem.

$$\begin{aligned} f(a,b,c,d,e) &= \overline{a}\ \overline{b}\ e + \overline{a}\ b\ \overline{d}\ \overline{e} + \overline{a}\ \overline{b}\ c\ d + \overline{c}\ \overline{d}\ e + a\ \overline{d}\ e + \\ &\quad + a\ c\ d\ \overline{e} + b\ c\ d\ e + a\ \overline{b}\ c\ \overline{d} \end{aligned} \tag{2.124}$$

Uma métrica da *complexidade de uma expressão* pode ser obtida, de forma simplificada, pelo número de implicantes envolvidos e pelo número de variáveis envolvidos em cada um deles. Este critério pode ser resumido ao número total de entradas em todas as funções AND e OR da expressão. Desse ponto de vista, esta expressão tem oito implicantes primos, dos quais três são produtos de três variáveis, e os restantes cinco são produtos de 4 variáveis. A complexidade C seria, portanto, dada por $C = 8 + 3 \times 3 + 5 \times 4 = 37$. Como se viu já, por vezes existem várias expressões minimais diferentes para a mesma função com, naturalmente, a mesma complexidade. Na Figura 2.42, um diferente conjunto de associações no mapa de Karnaugh conduz à Expressão 2.125, que tem a mesma complexidade que a expressão anterior, como é fácil de verificar. Esta função tem, na

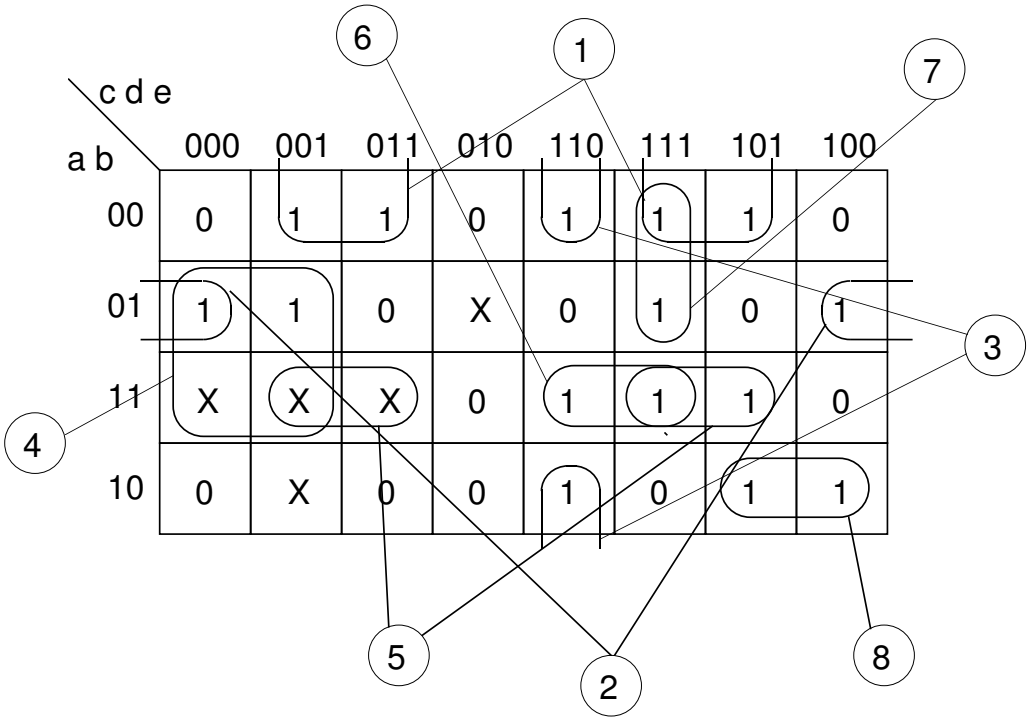


FIGURA 2.42: Minimização alternativa da função representada no mapa de Karnaugh da Figura 2.41.

realidade, um grande número de expressões minimais, como se pode constatar.

$$\begin{aligned} f(a,b,c,d,e) &= \overline{a}\ \overline{b}\ e + \overline{a}\ b\ \overline{d}\ \overline{e} + \overline{b}\ c\ d\ \overline{e} + b\ \overline{c}\ \overline{d} + a\ b\ e + a\ b\ c\ d + \\ &\quad + \overline{a}\ c\ d\ e + a\ \overline{b}\ c\ \overline{d} \end{aligned} \tag{2.125}$$

Até aqui, a função que tem vindo a ser usada como exemplo deu origem a expressões na forma normal disjuntiva. É evidentemente possível obter expressões minimais na forma normal conjuntiva, se se trabalhar com os 0 da função, maxtermos, implicados, etc.. A Figura 2.43 ilustra, para a mesma função, os agrupamentos escolhidos no mapa de Karnaugh. Note-se que todos os implicados primos assinalados são essenciais, com excepção do $(b+c+d)$. Para cada um dos implicados primos essenciais está assinalado a cinzento o maxtermo responsável pela sua essencialidade.

A Expressão 2.126 é a expressão minimal lida do mapa de Karnaugh.

$$\begin{aligned} f(a,b,c,d,e) &= (a+b+d+e)\cdot(\overline{b}+c+\overline{d})\cdot(a+\overline{b}+\overline{d}+e)\cdot \\ &\quad \cdot(a+\overline{b}+\overline{c}+d+\overline{e})\cdot(\overline{a}+\overline{b}+d+e)\cdot \\ &\quad \cdot(\overline{a}+b+\overline{d}+\overline{e})\cdot(b+c+d) \end{aligned} \tag{2.126}$$

A expressão obtida tem um grau de complexidade semelhante, ainda que um pouco

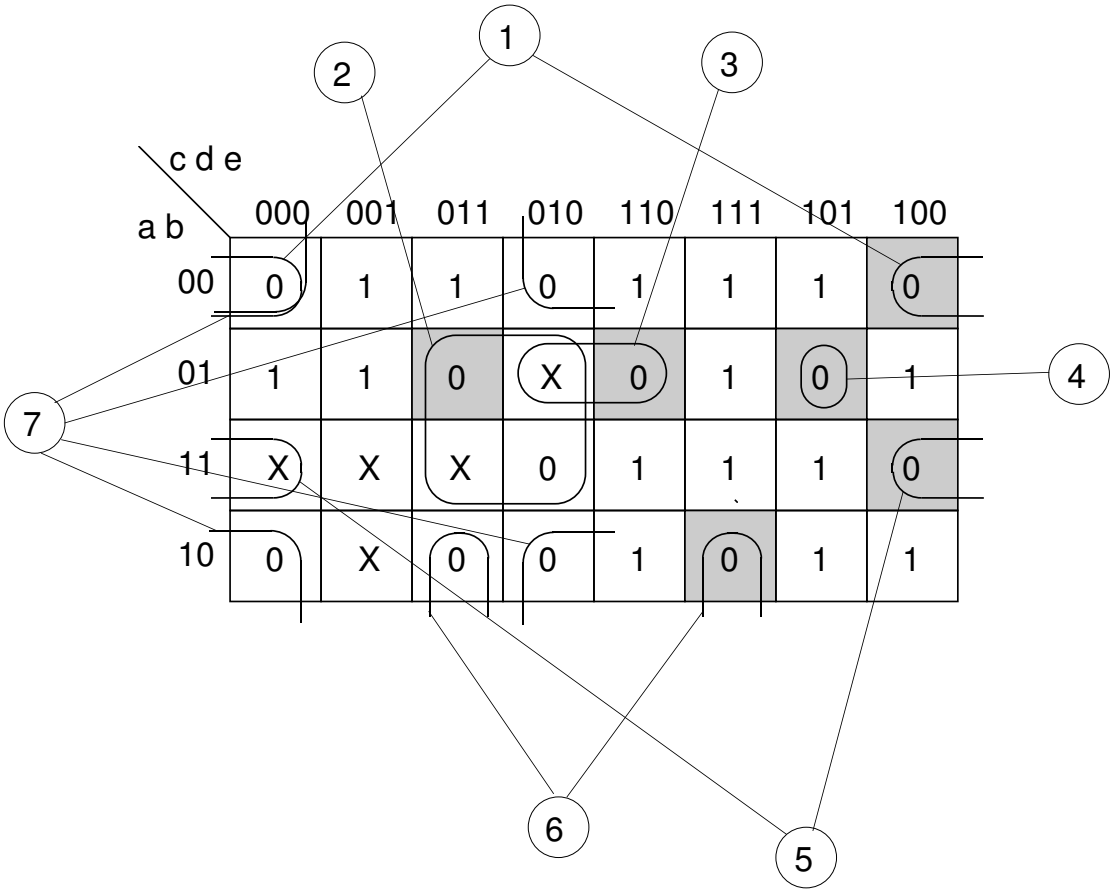


FIGURA 2.43: Minimização alternativa da função representada no mapa de Karnaugh da Figura 2.41, usando agrupamento de maxtermos.

inferior ($C = 34$) ao obtido com as minimizações em termos da forma normal disjuntiva. Note-se que, apesar de esta expressão ter um implicado primo não essencial, não é possível obter outra expressão minimal em termos de produtos de somas.

As diversas expressões obtidas correspondem a funções onde, como se referiu já, deixou de haver indiferenças. De facto, cada indiferença foi escolhida, no mapa de Karnaugh, como assumindo o valor 0 ou o valor 1 para serem realizados os agrupamentos que conduziram às expressões minimais. É importante salientar que, embora cada expressão seja uma expressão minimal, as diversas expressões correspondem a diferentes funções. Assim, por exemplo, como se pode verificar no mapa correspondente, a Expressão 2.124 corresponde a uma função que assume o valor 0, para a configuração correspondente ao mintermo m_{24}, e 1, para a configuração correspondente ao mintermo m_{17}, enquanto que a Expressão 2.125 corresponde a uma função em que $m_{24} = 1$ e $m_{17} = 0$. Existem mais diferenças, mas estas são suficientes para mostrar que as duas funções – e não apenas as expressões – são diferentes. Repare-se porém que, onde a função original tinha valores especificados, todas as funções correspondentes às expressões minimais assumem

os mesmos valores.

2.3.2 Método de Quine-McCluskey

O método de minimização usando o mapa de Karnaugh tem limitações decorrentes do facto de ser difícil de utilizar para funções com um número elevado de variáveis. Para mapas de sete ou oito variáveis é já praticamente impossível identificar as adjacências entre posições. No entanto, os aspectos basilares da técnica utilizada – determinação dos implicantes (ou implicados) primos e primos essenciais, com escolha de um conjunto mínimo de implicantes (ou implicados) – são gerais e podem ser utilizados para qualquer função.

Uma metodologia diferente, que consiste no processamento de tabelas em vez de observação de padrões gráficos de adjacência, é utilizada no *método de Quine-McCluskey*, que irá ser analisado de seguida. Este método permite trabalhar com funções com um número muito maior de variáveis, alargando-se muito os limites de praticabilidade da minimização óptima. No estudo deste método vai considerar-se apenas a variante correspondente à utilização de implicantes para obtenção de expressões em termos de forma normal disjuntiva. Sugere-se, como exercício, a utilização de implicados para obter a forma normal conjuntiva.

O método de Quine-McCluskey começa por obter todos os implicantes primos. Seguidamente selecciona os implicantes primos essenciais. Se estes não forem suficientes para a representação da função, selecciona, seguidamente, um conjunto suficiente e mínimo de implicantes primos.

Para se obter a lista de implicantes primos, começa-se pela obtenção de implicantes correspondentes à associação de dois mintermos, que serão designados por *implicantes de nível 1*. Os mintermos não associados são já implicantes primos. Seguidamente, associam-se todos os possíveis pares de implicantes de nível 1 para obter *implicantes de nível 2* (implicantes resultantes da associação de implicantes de nível 1). Os implicantes de nível 1 não associáveis são, naturalmente, implicantes primos. Procede-se da mesma forma, recursivamente, até não poder haver mais associação de implicantes, obtendo-se, assim, a lista completa de implicantes primos.

O exemplo seguinte procura esclarecer o método. Considere-se a função dada pela Expressão 2.127.

$$f(a,b,c,d,e) = \sum m(0,1,2,3,8,11,13,15,17,18,21,23,25,27,29,31) \qquad (2.127)$$

Para obter a associação inicial de mintermos em implicantes primos de nível 1, haverá

que representar os diversos mintermos e testar a possibilidade de se associar cada um deles com todos os outros. A este nível podem ser tomadas três medidas para simplificar o processamento: por um lado, em vez de representar os mintermos pela sua expressão, representar-se-ão pela configuração de variáveis para a qual assumem o valor 1. O mintermo m_{11}, por exemplo, será representado por 01011 em vez de $\overline{a}\ b\ \overline{c}\ d\ e$.

A segunda medida resulta de se compreender que, só podendo ser associados mintermos que diferem apenas numa variável, isso corresponde, com a representação anteriormente assumida, a pares de mintermos em que um deles tem n uns na sua representação e o outro $n+1$ uns na representação. Por exemplo, o mintermo m_3, representado por 00011, e o m_{11}, representado por 01011, diferem apenas na posição correspondente à variável b e têm respectivamente dois e três uns na sua representação. De tudo isto resulta que se devem agrupar os mintermos em conjuntos de mintermos com o mesmo número de uns na representação e que só vale a pena testar pares de mintermos correspondentes a conjuntos sucessivos de número de uns.

Por fim, a terceira verificação é que, sendo os pares de mintermos obtidos não ordenados, não vale a pena comparar o mintermo m_y com o mintermo m_x, se o mintermo m_x já tiver sido comparado com m_y.

De tudo isto resulta que se pode organizar uma tabela com uma certa estrutura reflectindo as considerações anteriormente feitas. Assim, para este exemplo, organiza-se uma *tabela de mintermos* com a estrutura ilustrada na Tabela 2.16. Repare-se que se agruparam os mintermos pelo número de uns. A primeira coluna da tabela indica o número do mintermo respectivo.

Seguidamente procede-se à comparação dos mintermos de grupos contíguos, à medida que se vai construindo uma segunda tabela, a dos implicantes de nível 1. Assim, o mintermo m_0 difere do mintermo m_1, apenas na posição da direita. Cria-se uma entrada na nova tabela, onde se lança essa associação e, simultaneamente, marcam-se os dois mintermos como já associados, usando o sinal $\surd$. Esse processo está ilustrado na Tabela 2.17. Vai-se construindo, deste modo, a *tabela de implicantes de nível 1*.

Na nova tabela são referidos, para cada associação, os mintermos associados (neste caso os mintermos 0 e 1). Repare-se que a variável que foi reduzida no implicante (neste caso a variável e) está assinalada com o símbolo «X».

Prosseguindo o processo até esgotar todas as associações possíveis, obtém-se a Tabela 2.18.

Repare-se que, na Tabela 2.18, todos os mintermos foram associados e que, portanto, nenhum deles é implicante primo.

TABELA 2.16: Tabela dos mintermos da função usada como exemplo.

	$abcde$
(0)	00000
(1)	00001
(2)	00010
(8)	01000
(3)	00011
(17)	10001
(18)	10010
(11)	01011
(13)	01101
(21)	10101
(25)	11001
(15)	01111
(23)	10111
(27)	11011
(29)	11101
(31)	11111

TABELA 2.17: Início da tabela dos implicantes de nível 1 e dos mintermos associados da função dada como exemplo.

	$abcde$	
(0)	00000	$\surd$
(1)	00001	$\surd$
(2)	00010	
...	...	

	$abcde$
(0, 1)	0000X
...	...

Pelo modo como estas tabelas são construídas, obtêm-se grupos em que o número de uns presentes em cada linha da tabela é sucessivamente maior. Esses grupos estão separados por linhas horizontais, tal como já acontecia com a tabela dos mintermos.

Na fase seguinte vão procurar associar-se pares de implicantes deste nível para obter implicantes de nível 2. Só são associáveis pares com o símbolo X na mesma posição, como é evidente. Na Tabela 2.19 listam-se os implicantes de nível 2 e, novamente, os implicantes de nível 1, assinalando agora os implicantes associados.

TABELA 2.18: Tabela completa (a) dos mintermos e (b) dos implicantes primos de nível 1 da função usada como exemplo.

	$abcde$	
(0)	00000	$\surd$
(1)	00001	$\surd$
(2)	00010	$\surd$
(8)	01000	$\surd$
(3)	00011	$\surd$
(17)	10001	$\surd$
(18)	10010	$\surd$
(11)	01011	$\surd$
(13)	01101	$\surd$
(21)	10101	$\surd$
(25)	11001	$\surd$
(15)	01111	$\surd$
(23)	10111	$\surd$
(27)	11011	$\surd$
(29)	11101	$\surd$
(31)	11111	$\surd$

(a)

	$abcde$
$(0,1)$	0000X
$(0,2)$	000X0
$(0,8)$	0X000
$(1,3)$	000X1
$(1,17)$	X0001
$(2,3)$	0001X
$(2,18)$	X0010
$(3,11)$	0X011
$(17,21)$	10X01
$(17,25)$	1X001
$(11,15)$	01X11
$(11,27)$	X1011
$(13,15)$	011X1
$(13,29)$	X1101
$(21,23)$	101X1
$(21,29)$	1X101
$(25,27)$	110X1
$(25,29)$	11X01
$(15,31)$	X1111
$(23,31)$	1X111
$(27,31)$	11X11
$(29,31)$	1111X

(b)

A Tabela 2.19 mostra que nem todos os implicantes de nível 1 foram associados entre si para produzir implicantes de nível 2. Os que ficaram por associar são já implicantes primos. Trata-se dos implicantes listados na Tabela 2.20.

Por não existirem, agora, dois pares com «X» na mesma posição, não é possível associar quaisquer pares de implicantes de nível 2 para obter implicantes de nível 3, pelo que todos os implicantes da Tabela 2.18 são implicantes primos. Trata-se dos implicantes listados na Tabela 2.21.

Neste ponto do algoritmo foram determinados todos os implicantes primos. A fase se-

TABELA 2.19: Tabela dos (a) implicantes de nível 1 associados e dos (b) implicantes de nível 2 da função usada como exemplo.

	$abcde$	
$(0,1)$	0000X	$\surd$
$(0,2)$	000X0	$\surd$
$(0,8)$	0X000	
$(1,3)$	000X1	$\surd$
$(1,17)$	X0001	
$(2,3)$	0001X	$\surd$
$(2,18)$	X0010	
$(3,11)$	0X011	
$(17,21)$	10X01	$\surd$
$(17,25)$	1X001	$\surd$
$(11,15)$	01X11	$\surd$
$(11,27)$	X1011	$\surd$
$(13,15)$	011X1	$\surd$
$(13,29)$	X1101	$\surd$
$(21,23)$	101X1	$\surd$
$(21,29)$	1X101	$\surd$
$(25,27)$	110X1	$\surd$
$(25,29)$	11X01	$\surd$
$(15,31)$	X1111	$\surd$
$(23,31)$	1X111	$\surd$
$(27,31)$	11X11	$\surd$
$(29,31)$	111X1	$\surd$

(a)

	$abcde$
$(0,1,2,3)$	000XX
$(17,21,25,29)$	1XX01
$(11,15,27,31)$	X1X11
$(13,15,29,31)$	X11X1
$(21,23,29,31)$	1X1X1
$(25,27,29,31)$	11XX1

(b)

TABELA 2.20: Implicantes primos de nível 1 da função usada como exemplo.

Designação	Expressão
$(0,8)$	$\overline{a}\ \overline{c}\ \overline{d}\ \overline{e}$
$(1,17)$	$\overline{b}\ \overline{c}\ \overline{d}\ e$
$(2,18)$	$\overline{b}\ \overline{c}\ d\ \overline{e}$
$(3,11)$	$\overline{a}\ \overline{c}\ d\ e$

TABELA 2.21: Implicantes primos de nível 2 da função usada como exemplo.

Designação	Expressão
$(0, 1, 2, 3)$	$\overline{a}\ \overline{b}\ \overline{c}$
$(17, 21, 25, 29)$	$a\ \overline{d}\ e$
$(11, 15, 27, 31)$	$b\ d\ e$
$(13, 15, 29, 31)$	$b\ c\ e$
$(21, 23, 29, 31)$	$a\ c\ e$
$(25, 27, 29, 31)$	$a\ b\ e$

guinte consiste em identificar, destes, quais são os implicantes primos essenciais. Para isso, é necessário construir a *tabela dos implicantes primos*, que permite verificar, para cada mintermo, quais os implicantes primos que o representam. Essa tabela está representada na Tabela 2.22.

TABELA 2.22: Tabela dos implicantes primos da função usada como exemplo.

	0	1	2	3	8	11	13	15	17	18	21	23	25	27	29	31
$(0, 8)$	√				√											
$(1, 17)$		√							√							
$(2, 18)$			√							√						
$(3, 11)$				√		√										
$(0, 1, 2, 3)$	√	√	√	√												
$(17, 21, 25, 29)$									√		√		√		√	
$(11, 15, 27, 31)$						√		√						√		√
$(13, 15, 29, 31)$							√	√							√	√
$(21, 23, 29, 31)$											√	√			√	√
$(25, 27, 29, 31)$													√	√	√	√

Por análise da Tabela 2.22 verifica-se que o único implicante primo que associa o mintermo m_8 é o implicante primo designado por $(0, 8)$ ($\overline{a}\ \overline{c}\ \overline{d}\ \overline{e}$), que é, assim, um implicante primo essencial. De igual modo, é fácil verificar que o implicante primo $(2, 18)$ é essencial por causa do mintermo m_{18}, que $(13, 15, 29, 31)$ é essencial por causa do mintermo m_{13}, e que, finalmente, $(21, 23, 29, 31)$ é essencial por causa do mintermo m_{23}. Estão, assim, encontrados os implicantes primos essenciais da função. É fácil ver que este conjunto de implicantes primos essenciais associa muitos dos mintermos da fun-

TABELA 2.23: Tabela refeita dos implicantes primos da função usada como exemplo.

		1	3	11	17	25	27
A	$(1, 17)$	√			√		
B	$(3, 11)$		√	√			
C	$(0, 1, 2, 3)$	√	√				
D	$(17, 21, 25, 29)$				√	√	
E	$(11, 15, 27, 31)$			√			√
F	$(25, 27, 29, 31)$					√	√

ção – os mintermos $m(0, 2, 8, 13, 15, 18, 21, 23, 29, 31)$ –, mas que há ainda mintermos da função não representados $(1, 3, 11, 17, 25, 27)$.

A fase seguinte do processo consiste em refazer a tabela dos implicantes primos, tomando agora em consideração apenas os mintermos não representados e retirando da tabela os implicantes primos essenciais, já seleccionados. Obtém-se, desse modo, a Tabela 2.23.

Esta tabela poderia levar a definir novos implicantes primos que, nas actuais circunstâncias, fossem necessários para cobrir algum ou alguns dos mintermos ainda não representados. Esta é uma situação comum, mas não é esse o caso para esta função. Aquilo que se obteve é uma tabela que indica que, para cada mintermo, há dois implicantes primos que podem ser escolhidos. Nestas circunstâncias, o problema passa a ser o da escolha de um conjunto mínimo de implicantes primos que associem a totalidade dos mintermos. Isso pode ser feito de forma heurística por observação da tabela. Para facilitar a exposição do raciocínio, os implicantes primos foram, nesta nova tabela, designados por um conjunto de letras de A a F. Repare-se que, se forem escolhidos os implicantes A e B, ficam cobertos os primeiros quatro mintermos e, claramente, já não precisamos do implicante C. Ficaram por cobrir os dois últimos mintermos, o que pode ser conseguido, escolhendo os implicantes D e E ou, melhor ainda, o implicante F. Uma outra hipótese é a de escolher os implicantes C, D e E. Em ambos os casos foram escolhidos três implicantes para cobrir os mintermos restantes. Não é possível resolver o problema com menos implicantes, como é possível concluir da análise da tabela. Podem, então, ser usados os conjuntos de implicantes A, B e F ou C, D e E. A escolha não é, porém, indiferente. De facto, A e B são produtos de quatro variáveis, enquanto que todos os outros são produtos de três variáveis, sendo, assim, mais simples a expressão que inclua C, D e E.

De todo este processo resulta, portanto, a Expressão 2.128 como a expressão minimal (na forma normal disjuntiva) da função que temos vindo a usar como exemplo. Os

quatro primeiros termos correspondem aos implicantes primos essenciais encontrados, e os restantes três, aos implicantes primos seleccionados adicionalmente para obter a Expressão 2.128.

$$f(a,b,c,d,e) = \overline{a}\ \overline{c}\ \overline{d}\ \overline{e} + \overline{b}\ \overline{c}\ d\ \overline{e} + b\ c\ e + a\ c\ e + \overline{a}\ \overline{b}\ \overline{c} + a\ \overline{d}\ e + b\ d\ e \quad (2.128)$$

O processo de escolha de implicantes primos, após a obtenção dos implicantes primos essenciais, assentou numa análise pouco sistemática, que só foi possível de realizar por a tabela ser relativamente simples. Há, contudo, uma forma sistemática de obter o conjunto a seleccionar de implicantes primos não essenciais. Para isso, é necessário obter a *função de implicantes primos* ou *função-p*. Esta função determina quais os implicantes primos necessários na expressão da função e usa como variáveis as designações dos implicantes primos ainda presentes na tabela dos implicantes primos. É fácil perceber que, na expressão final, terá de existir pelo menos um implicante que cubra cada um dos mintermos da função. Por exemplo, terá de estar presente o implicante A ou o C para cobrir m_1, o implicante B ou o C para cobrir m_3, e assim sucessivamente. Daí resulta que a função-p tem a Expressão 2.129.

$$p = (A + C)\ (B + C)\ (B + E)\ (A + D)\ (D + F)\ (E + F) \quad (2.129)$$

Manipulando algebricamente a Expressão 2.129 para obter a forma normal disjuntiva, obtém-se a Expressão 2.130.

$$p = A\ B\ D\ E + A\ B\ F + B\ C\ D\ E + B\ C\ D\ F + C\ D\ E + A\ C\ E\ F \quad (2.130)$$

A leitura desta expressão é fácil de fazer: para a expressão minimal da função é necessário utilizar os implicantes primos A e B e D e E ou os implicantes A e B e F, e assim por diante. Verifica-se, portanto, que o número mínimo de implicantes a utilizar é três e que se pode optar pelos conjuntos A, B e F ou C, D e E.

O método de Quine-McCluskey é uma réplica do método de Karnaugh, utilizando manipulação de tabelas e, no caso da função-p, de expressões, em vez do reconhecimento de padrões no mapa. É mais trabalhoso, mas, por ser mais sistemático, tem a vantagem de ser mais facilmente implementável num programa, permitindo minimizar expressões de funções com grande número de variáveis.

No caso de as funções terem indiferenças, a utilização do método pouco se altera. Consideram-se as posições em que há indiferenças como mintermos para a obtenção dos implicantes primos, mas omitem-se na tabela de implicantes primos para selecção dos implicantes necessários na expressão da função. O método será exemplificado com a função detectora de algarismos BCD múltiplos de 3 descrita na Tabela 2.15, que foi usada para introduzir este tópico na apresentação do método de Karnaugh.

A função tem os mintermos m_3, m_6 e m_9 e tem indiferenças nas posições 10 a 15. As diversas tabelas de mintermos e de implicantes de nível 1 e 2 (não há implicantes de nível 3) encontram-se nas Tabelas 2.24 e 2.25.

TABELA 2.24: Tabelas de (a) mintermos e (b) implicantes de nível 1 da função em estudo.

	$A_3A_2A_1A_0$		
3	0011	1	$\surd$
6	0101	1	$\surd$
9	1001	1	$\surd$
10	1010	X	$\surd$
12	1100	X	$\surd$
11	1011	X	$\surd$
13	1101	X	$\surd$
14	1110	X	$\surd$
15	1111	X	$\surd$

(a)

	$A_3A_2A_1A_0$	
$(3,11)$	X011	
$(6,13)$	X101	
$(9,11)$	10X1	$\surd$
$(9,13)$	1X01	$\surd$
$(10,11)$	101X	$\surd$
$(10,14)$	1X10	$\surd$
$(12,13)$	110X	$\surd$
$(12,14)$	11X0	$\surd$
$(11,15)$	1X11	$\surd$
$(13,15)$	11X1	$\surd$
$(14,15)$	111X	$\surd$

(b)

TABELA 2.25: Tabela de implicantes de nível 2 da função em estudo.

	$A_3A_2A_1A_0$	
$(9,11,13,15)$	1XX1	
$(10,11,14,15)$	1X1X	
$(12,13,14,15)$	11XX	

A tabela de implicantes primos desta função, representada na Tabela 2.26, tem apenas três mintermos. Dos implicantes primos obtidos no passo anterior, verifica-se que dois deles, $(10,11,14,15)$ e $(12,13,14,15)$, não associam qualquer mintermo, pelo que não são incluídos na tabela. Desta tabela é fácil concluir que os três implicantes primos são essenciais e são também suficientes para representar a função. A Expressão 2.131 é, portanto, a expressão minimal da função estudada, de acordo aliás com o resultado já obtido na Expressão 2.121.

$$f(A_3,A_2,A_1,A_0) = A_3\ A_0 + A_2\ \overline{A_1}\ A_0 + \overline{A_2}\ A_1\ A_0 \tag{2.131}$$

TABELA 2.26: Tabela de implicantes primos da função em estudo.

	3	6	9
$(3, 11)$	$\surd$		
$(6, 13)$		$\surd$	
$(9, 11, 13, 15)$			$\surd$

O exemplo final corresponde à função já minimizada pelo método de Karnaugh, definida anteriormente pela Expressão 2.123, aqui repetida (Expressão 2.132).

$$f(a,b,c,d,e) = \sum m(1,3,5,6,7,8,9,12,15,20,21,22,29,30,31) + \\ + \sum md(10,17,24,25,27) \quad (2.132)$$

Na Tabela 2.27 apresentam-se os mintermos e implicantes da função.

A partir das tabelas anteriores é fácil construir a tabela de implicantes primos, que, como anteriormente, só vai incluir os mintermos e os implicantes primos que associem mintermos – todos no caso presente –, sendo excluída, portanto, a necessidade de incluir indiferenças. A Tabela 2.28 representa a tabela de implicantes primos.

Nesta tabela pode observar-se que o implicante primo $(8, 10)$, uma vez que a posição 10 corresponde a uma indiferença, apenas cobre o mintermo m_8, que é coberto também pelo implicante primo $(8, 12)$. Podemos, assim, prescindir daquele implicante primo. A observação da tabela permite concluir que há apenas dois implicantes primos essenciais: $(8, 12)$, por ser o único que associa o mintermo m_{12}, e $(1, 3, 5, 7)$, por ser o único implicante primo que cobre o mintermo m_3. A expressão da função incluirá, portanto, estes dois implicantes, mas não fica limitada a eles, uma vez que há ainda vários mintermos não representados. Reestrutura-se, portanto, a tabela, retirando os mintermos já cobertos, obtendo-se a nova Tabela 2.29.

Nesta nova tabela é fácil ver que há alguns implicantes primos que só associam mintermos incluídos noutros implicantes primos. É o caso dos implicantes primos representados por $(6, 7)$, $(7, 15)$ e $(1, 5, 17, 21)$, que não serão, portanto, considerados. De igual modo, os implicantes primos $(1, 9, 17, 25)$ e $(8, 9, 24, 25)$ associam apenas o mintermo m_9 nesta tabela já reduzida, podendo, assim, prescindir-se de um deles. Da análise da tabela resulta, então, natural a escolha dos implicantes primos $(6, 22)$, $(15, 31)$ e $(1, 9, 17, 25)$. Uma vez que não se cobriram ainda todos os mintermos, avança-se para nova versão mais reduzida da tabela, representada agora na Tabela 2.30.

TABELA 2.27: Tabelas de (a) mintermos, (b) implicantes de nível 1 e (c) implicantes de nível 2 da função em estudo.

	a,b,c,d,e		
1	00001	1	√
8	01000	1	√
3	00011	1	√
5	00101	1	√
6	00101	1	√
9	01001	1	√
10	01010	X	√
12	01100	1	√
17	10001	X	√
20	10100	1	√
24	11000	X	√
7	00111	1	√
21	10101	1	√
22	10110	1	√
25	11001	X	√
15	01111	1	√
27	11011	X	√
29	11101	1	√
30	11110	1	√
31	11111	1	√

(a)

	a,b,c,d,e	
$(1,3,5,7)$	00XX1	
$(1,5,17,21)$	X0X01	
$(1,9,17,25)$	XX001	
$(8,9,24,25)$	X100X	
$(17,21,25,29)$	1XX01	
$(25,27,29,31)$	11XX1	

(c)

	a,b,c,d,e	
$(1,3)$	000X1	√
$(1,5)$	00X01	√
$(1,9)$	0X001	√
$(1,17)$	X0001	√
$(8,9)$	0100X	√
$(8,10)$	010X0	
$(8,12)$	01X00	
$(8,24)$	X1000	√
$(3,7)$	00X11	√
$(5,7)$	001X1	√
$(5,21)$	X0101	√
$(6,7)$	0011X	
$(6,22)$	X0110	
$(9,25)$	X1001	√
$(17,21)$	10X01	√
$(17,25)$	1X001	√
$(20,21)$	1010X	
$(20,22)$	101X0	
$(24,25)$	1100X	√
$(7,15)$	0X111	
$(21,29)$	1X101	√
$(22,30)$	1X110	
$(25,27)$	110X1	√
$(25,29)$	11X01	√
$(15,31)$	X1111	
$(27,31)$	11X11	√
$(29,31)$	111X1	√
$(30,31)$	1111X	

(b)

Por razões semelhantes às anteriores é agora possível prescindir dos implicantes primos

TABELA 2.28: Tabela dos implicantes primos da função usada como exemplo.

	1	3	5	6	7	8	9	12	15	20	21	22	29	30	31
$(8, 10)$						√									
$(8, 12)$						√		√							
$(6, 7)$				√	√										
$(6, 22)$				√								√			
$(20, 21)$										√	√				
$(20, 22)$										√		√			
$(7, 15)$					√				√						
$(22, 30)$												√		√	
$(15, 31)$									√						√
$(30, 31)$														√	√
$(1, 3, 5, 7)$	√	√	√		√										
$(1, 5, 17, 21)$	√		√								√				
$(1, 9, 17, 25)$	√						√								
$(8, 9, 24, 25)$						√	√								
$(17, 21, 25, 29)$											√		√		
$(25, 27, 29, 31)$													√		√

TABELA 2.29: Tabela refeita dos implicantes primos da função usada como exemplo.

	6	9	15	20	21	22	29	30	31
$(6, 7)$	√								
$(6, 22)$	√					√			
$(20, 21)$				√	√				
$(20, 22)$				√		√			
$(7, 15)$			√						
$(22, 30)$						√		√	
$(15, 31)$			√						√
$(30, 31)$								√	√
$(1, 5, 17, 21)$					√				
$(1, 9, 17, 25)$		√							
$(8, 9, 24, 25)$		√							
$(17, 21, 25, 29)$					√		√		
$(25, 27, 29, 31)$							√		√

TABELA 2.30: Tabela refeita dos implicantes primos da função usada como exemplo.

	20	21	29	30
$(20,21)$	$\surd$	$\surd$		
$(20,22)$	$\surd$			
$(22,30)$				$\surd$
$(30,31)$				$\surd$
$(17,21,25,29)$		$\surd$	$\surd$	
$(25,27,29,31)$			$\surd$	

$(20,22)$ e $(25,27,29,31)$ e também de um dos do par $(22,30)$ e $(30,31)$. Da análise da tabela é possível seleccionar os implicantes primos $(20,21)$, $(30,31)$ e $(17,21,25,29)$. A função pode, portanto, ser representada pela Expressão 2.133.

$$\begin{aligned} f(a,b,c,d,e) &= \overline{a}\ b\ \overline{d}\ \overline{e} + \overline{a}\ \overline{b}\ e + \overline{b}\ c\ d\ \overline{e} + b\ c\ d\ e + \\ &\quad + \overline{c}\ \overline{d}\ e + a\ \overline{b}\ c\ \overline{d} + a\ b\ c\ d + a\ \overline{d}\ e \end{aligned} \tag{2.133}$$

Repare-se que, neste caso, não foi necessário recorrer à função-p para determinar os implicantes primos não essenciais. Foi possível, por análise sucessiva das tabelas de implicantes primos, seleccionar a expressão minimal da função. Um outro ponto interessante é, neste caso, a comparação entre o método de Karnaugh e o de Quine McCluskey para a mesma função. No método de Karnaugh tinha-se verificado que havia muitas expressões minimais possíveis, enquanto que, usando o método de Quine-McCluskey, se obteve uma expressão onde há, aparentemente, poucas opções. Isso resultou da maneira como foi evoluindo a tabela dos implicantes primos, que escondeu opções possíveis para a obtenção da expressão final. Por exemplo, na última tabela, prescindimos do implicante primo $(20,22)$, uma vez que o $(20,21)$ cobria o mintermo m_{20}, único coberto pelo primeiro implicante primo. No entanto, tal não era necessário. Se fosse escolhido o implicante $(20,22)$, isso seria uma opção válida, já que o mintermo m_{21} é também coberto pelo implicante primo $(17,21,25,29)$, que foi também seleccionado. Muitas outras situações deste tipo acontecem neste exemplo. Se tivesse sido aplicada a função-p à primeira tabela de implicantes primos, teria sido possível obter uma expressão em que todas as opções de minimização da expressão desta função teriam resultado explicitamente.

SUMÁRIO

Neste capítulo é feito um estudo sob vários aspectos das funções que descrevem formalmente os sistemas digitais, habitualmente designadas por funções lógicas. Inicialmente é apresentada a álgebra de Boole, a ferramenta matemática de suporte ao estudo dos circuitos digitais. Embora se possam definir álgebras de Boole com mais de dois elementos, no caso que nos interessa, o estudo é restringido a álgebras de Boole binárias.

Com base na álgebra de Boole são descritas as diversas formas de representar funções lógicas, com ênfase na equivalência, por um lado, e na complementaridade, por outro, das diversas formas de representação. O aspecto particular da minimização das expressões que representam funções lógicas é discutido com grande pormenor para o caso em que se pretendem obter expressões a dois níveis. São apresentados, neste contexto, os métodos de Karnaugh e de Quine-McCluskey. São ainda discutidos os aspectos ligados à minimização de expressões correspondentes a funções incompletamente especificadas.

EXERCÍCIOS

2.1 Elabore as tabelas de verdade das seguintes funções lógicas:

1. $f(A,B,C) = A\ B + \overline{A}\ \overline{B}\ \overline{C}$
2. $f(A,B,C,D) = A\ B\ (\overline{C} \oplus D) + A\ B\ C + \overline{A}\ \overline{C}\ D + \overline{B}\ C\ D$
3. $f(A,B,C) = \overline{\overline{A}\ B}\ C + A\ \overline{\overline{B} + C}$

2.2 Minimize algebricamente as seguintes funções:

1. $f(A,B,C,D) = A\ B\ (\overline{C} \oplus D) + A\ B\ C + \overline{A}\ \overline{C}\ D + \overline{B}\ C\ D$
2. $f(A,B,C,D) = A\ B\ C + \overline{A\ \overline{B}}\ (C + D) + \overline{A + B + \overline{D}}$

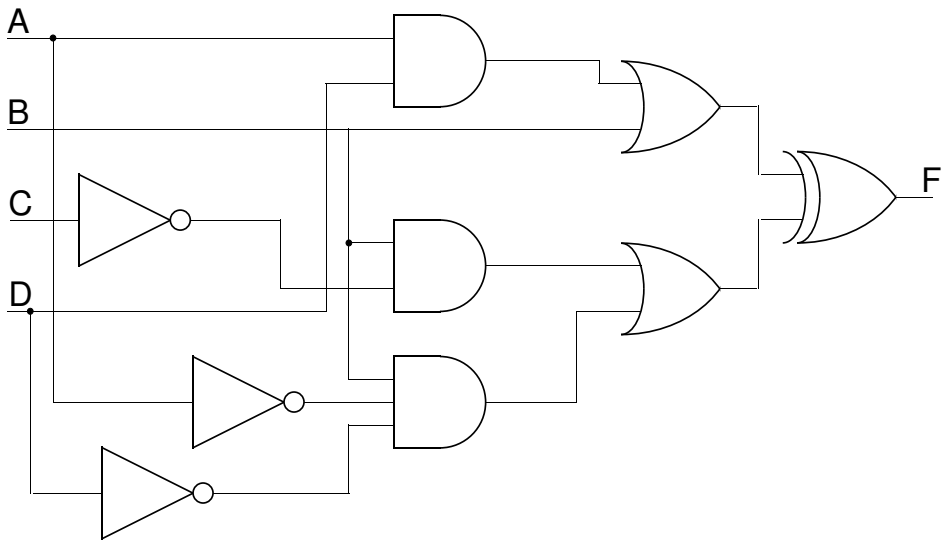
2.3 Considere a seguinte função: $f(A,B,C) = (A \oplus B)\ C + \overline{A}\ (B \oplus C)$:

1. minimize algebricamente a função para obter um produto de somas;
2. minimize algebricamente a função para obter uma soma de produtos;
3. obtenha a forma canónica normal disjuntiva;
4. obtenha a forma canónica normal conjuntiva.

2.4 Obtenha a expressão em forma de soma de produtos da função maioria com 3 variáveis. A função maioria é uma função que assume o valor 1 quando o número de variáveis a 1 é superior ao número de entradas a 0. Repita para 4 e 5 variáveis.

2.5 Obtenha a expressão de uma função lógica que indica se uma determinada configuração de quatro bits é uma palavra do código BCD.

2.6 Considere o seguinte logigrama:



Determine uma expressão algébrica para a função F, minimize-a e desenhe o logigrama correspondente à expressão resultante.

2.7 Construa o logigrama correspondente a um conjunto de funções de três variáveis que calculam o factorial do número binário representado pelas variáveis de entrada.

2.8 Determine um conjunto de funções lógicas que realize a multiplicação de dois números binários de dois algarismos.

2.9 Determine um conjunto de funções lógicas que obtenha o valor inteiro da raiz quadrada de um número binário de 5 algarismos.

2.10 Minimize as seguintes funções lógicas, utilizando o método de Karnaugh. Para cada caso utilize, quer somas de produtos, quer produtos de somas, e conclua qual produz a expressão mais simples:

1. $f(A,B,C,D,E) = \sum m(1,3,5,8,10,12,14,21,23,24,26,31)$ com indiferenças nas posições 0, 4, 7, 15, 17, 18, 27 e 28;
2. $f(A,B,C,D,E) = \Pi M(0,2,5,7,9,10,11,20,22,23,27,30)$ com indiferenças nas posições 1, 8, 12, 15, 16, 25 e 31;
3. $f(A,B,C,D,E) = \sum m(3,4,6,7,10,13,19,23,28,29,31)$ com indiferenças nas posições 5, 12, 15, 21, 22 e 25;

4. $f(A,B,C,D,E) = \sum m(1,3,5,6,7,8,9,12,15,20,21,22,29,30,31)$ com indiferenças nas posições 10, 17, 24, 25 e 27;
5. $f(A,B,C,D) = \overline{A}\ (\overline{C} \oplus D) + A\ \overline{D} + \overline{A}\ \overline{C}\ D + \overline{A}\ C\ \overline{D}$ com indiferenças nas posições 1, 7, 11 e 13.

2.11 Repita o problema anterior, utilizando o método de Quine-McCluskey.

2.12 Para as funções lógicas do problema anterior determine todos os implicantes(ados) primos e, no caso de os haver, os implicantes(ados) primos essenciais.

2.13 Considere a função $f(A,B,C,D) = \sum m(0,1,3,4,7,9,10,12)$ com indiferenças nas posições 2, 5, 14 e 15, Obtenha todas as expressões mínimas da função, quer utilizando somas de produtos, quer produtos de somas. Verifique, utilizando tabelas de verdade construídas para cada uma das expressões obtidas, que as diversas expressões não correspondem à mesma função. Justifique o facto.

2.14 Considere a seguinte função:

$f(A,B,C,D,E) = \Pi M(1,7,9,12,14,18,19,21,22,25,28,30,31)$ com indiferenças nas posições 4, 15, 16, 17 e 20.

1. Minimize-a utilizando o método de Karnaugh.
2. A partir da função $f(A,B,C,D,E)$ e sem a alterar, usando o mínimo de lógica possível, projecte uma função $g(A,B,C,D,E)$ com uma tabela semelhante tendo como única diferença a existência de saída 1 nas posições 1, 4 e 21.

2.15 Manipule as expressões lógicas obtidas no Problema 2.10 para obter:

1. expressões utilizando apenas funções NAND de até quatro entradas;
2. expressões utilizando apenas funções AND e OR com duas variáveis de entrada e NOT;
3. quando possível, expressões mais simples com mais de duas camadas.

3

REALIZAÇÃO FÍSICA DE CIRCUITOS LÓGICOS

Neste capítulo são apresentadas algumas das tecnologias que permitem construir circuitos lógicos que dão suporte à realização de funções lógicas, isto é, circuitos eléctricos cujo comportamento pode ser descrito por aquelas funções.

Na Secção 3.1 são referidas algumas das tecnologias de circuitos integrados usados para construir circuitos lógicos. São também referidos alguns dispositivos úteis para o projecto de sistemas, mas que não implementam directamente funções como as definidas no capítulo anterior.

O uso dos dispositivos descritos para a implementação de circuitos e as metodologias que lhe estão associadas são descritos na Secção 3.2.

Um aspecto importante neste contexto é o das técnicas para identificação de erros que produzem falhas no funcionamento dos circuitos. A Secção 3.3 aborda, de maneira muito introdutória, essa questão.

Na Secção 3.4 são analisados os aspectos referentes ao facto de os circuitos electrónicos necessitarem de tempo para operar e as consequências deste facto nos circuitos digitais.

Uma metodologia alternativa de implementação de circuitos directamente a partir da sua especificação formal e utilizando dispositivos lógicos programáveis é apresentada na Secção 3.5.

3.1 CIRCUITOS INTEGRADOS DIGITAIS

O interesse do estudo que se fez das funções lógicas reside no facto de que estas funções podem ser usadas para descrever e especificar acções e processamentos pretendidos de um equipamento. Simultaneamente, há metodologias que permitem uma fácil passagem de uma descrição em termos de funções lógicas para a implementação de circuitos electrónicos com um comportamento especificado por essas funções, como se verá ao longo deste capítulo.

3.1.1 Famílias Lógicas

Uma possível implementação de circuitos digitais é feita com base em dispositivos electrónicos que têm um comportamento semelhante a algumas das funções lógicas descritas no Capítulo 2, como as funções AND, OR, NOT, ou outras. É usual designar esses dispositivos por *portas* (em inglês, *gates*) ou portas lógicas. As portas lógicas estão normalmente incluídas em *circuitos integrados*.

A tecnologia dos circuitos integrados é a responsável pelo tremendo desenvolvimento dos sistemas digitais, uma vez que possibilitou a construção de circuitos electrónicos digitais muito complexos, a custos extremamente reduzidos e ocupando espaços muito limitados.

Um circuito integrado é um pequeno cristal de silício (há outras tecnologias, mas o silício é largamente dominante) onde se difundiram impurezas em determinadas áreas dando origem a transístores, díodos, resistências e outros componentes electrónicos interligados entre si, de modo a formarem circuitos electrónicos de maior ou menor complexidade. Os circuitos integrados digitais são de fácil concepção e têm custos de produção relativamente baixos.

Cada circuito integrado pode ter entre algumas portas e alguns milhões de portas. Por exemplo, um dos circuitos integrados mais simples, disponível no mercado, é um circuito com portas AND que tem quatro portas AND de duas entradas, que podem ser usadas separadamente. Um circuito como um contador tem algumas dezenas de portas que estão previamente interligadas para formar o contador. Um processador das famílias Pentium ou PowerPC tem vários milhões de portas.

Existem várias tecnologias para fabricar circuitos integrados digitais, das quais as mais importantes actualmente são a *TTL*, a *CMOS* e a *ECL*:

— TTL: Uma tecnologia que foi durante muitos anos, de longe, a mais utilizada, tendo, na prática, estabelecido todo um referencial de arquitecturas, critérios e normas, com consequências no desenvolvimento de outras famílias. Utiliza *transístores do tipo bipolar* a funcionar ao corte (o dispositivo comporta-se como um interruptor aberto) ou na saturação (o dispositivo comporta-se como um interruptor fechado), dois dos modos de funcionamento desse tipo de dispositivos. Tem um comportamento médio no que diz respeito à velocidade e ao consumo. Subdivide-se em várias subfamílias com características diferenciadas, mas todas compatíveis entre si. Dado ter-se tornado muito popular, outras tecnologias têm circuitos com entradas e saídas compatíveis com TTL. É uma tecnologia que, actualmente, perde mercado a favor da CMOS. A sigla TTL é construída com as iniciais de *Transistor-Transistor Logic*, designação que põe em realce a estrutura interna destas portas, construídas com dois níveis de transístores.

— CMOS: O CMOS é, actualmente, a tecnologia mais utilizada, tendo ultrapassado a TTL no final dos anos 80 do século XX. Utiliza *transístores do tipo MOSFET*. Inicialmente era uma tecnologia mais lenta que a TTL, com a vantagem de consumir muito menos potência. Hoje, com a evolução da tecnologia dos circuitos integrados, as frequências de operação obtidas são já muito elevadas, embora parcialmente à custa

do consumo de energia. Os ganhos de velocidade da tecnologia CMOS, combinados com a grande capacidade de integração, colocaram-na como a tecnologia mais importante no projecto e fabrico de circuitos integrados. Existem também algumas subfamílias nesta tecnologia. É a tecnologia com um mercado em mais rápido crescimento. A sigla CMOS deriva das iniciais de *Complementary Metal Oxide Semiconductor*, designação que se refere à tecnologia usada, que utiliza transístores do tipo n e p na mesma porta.

— ECL: Trata-se de uma tecnologia que permite construir circuitos muito rápidos, embora à custa de um enorme consumo energético. Tem nichos de mercado muito limitados, nomeadamente as aplicações militares. Utiliza transístores bipolares na zona activa. Enquanto que as tecnologias anteriores podem ser utilizadas em projectos de circuitos, a partir dos dispositivos existentes, nomeadamente portas, requerendo poucos conhecimentos em electrónica e em propagação de sinais em linhas, a tecnologia ECL não dispensa esse tipo de conhecimentos para uma concepção correcta de circuitos com bom desempenho. A sigla ECL é construída com as iniciais de *Emitter Coupled Logic*, numa referência ao tipo de estrutura de circuitos electrónicos utilizada.

Fisicamente, um circuito integrado apresenta-se em vários formatos de encapsulamento, dos quais o que é utilizado mais frequentemente em laboratório é o tipo *DIL* (do inglês, *Dual In-Line*). Este encapsulamento consiste numa caixa de plástico ou cerâmica, paralelepipédica, onde existem duas filas de *terminais* ao longo dos lados maiores do paralelepípedo. Estes terminais são usualmente designado por *pins*. Na Figura 3.1 ilustra-se um circuito com 14 terminais. É importante ter em conta, porém, que, actualmente, a maior parte dos circuitos integrados disponíveis no mercado usa tipos diferentes de encapsulamento, que permitem *montagem superficial*, um tipo de montagem que permite muito maior compactação e o uso de circuitos com um número de terminais muito mais elevado.

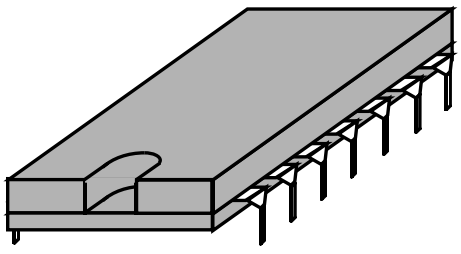


FIGURA 3.1: Circuito integrado em encapsulamento DIL.

Os fabricantes produzem séries de circuitos com características eléctricas semelhantes, em que as designações são cadeias de algarismos e letras com a mesma estrutura. Muitas vezes, vários fabricantes acordam em usar as mesmas designações para produtos

semelhantes. No que diz respeito a circuitos digitais de baixa ou média complexidade, as séries 74nn são ainda muito utilizadas actualmente, embora a tendência, como se refere na Secção 3.5, seja a de utilizar circuitos programáveis ou desenhados propositadamente para aplicações específicas, em ambos os casos com um número muito elevado de portas. Um exemplo de circuito integrado é o 7408, que existe quer na família TTL, quer na CMOS. Este circuito integrado disponibiliza quatro portas AND de duas entradas. Considerando o circuito visto de cima, a disposição está indicada na Figura 3.2. A distribuição de terminais é referida em inglês por *pin-out*.

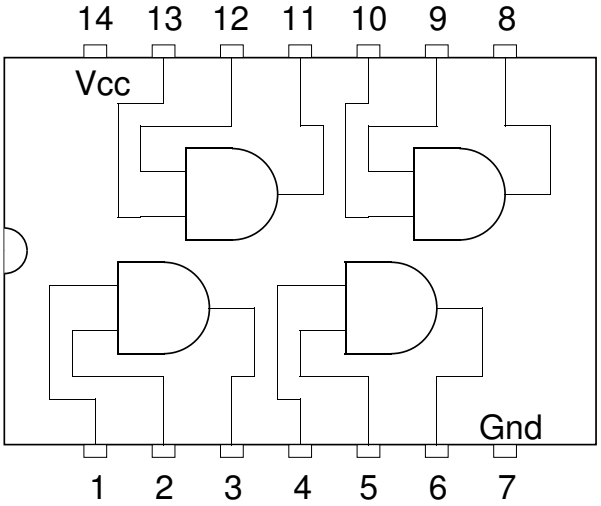


FIGURA 3.2: Distribuição de terminais do circuito integrado 7408.

Repare-se na numeração dos terminais, que é feita com início no canto inferior esquerdo e prossegue, rodando em torno do circuito no sentido directo, até ao canto superior esquerdo. A marca existente num dos lados menores do circuito (à esquerda, na figura) permite orientar o dispositivo. Os terminais 14 e 7 são os terminais que permitem alimentar electricamente o circuito. Na família TTL têm, respectivamente, a designação de V_{CC} e Gnd (abreviatura do inglês, *Ground*). O terminal V_{CC} deve ser ligado a uma tensão positiva de 5V, e o Gnd, a 0V, usualmente designado por *terra*.

Nos catálogos de circuitos integrados podem ser encontradas as distribuições de terminais para todos os circuitos integrados de qualquer família. A título de exemplo, indicam-se na Tabela 3.1 as referências para alguns circuitos correspondentes a portas lógicas, sem procurar uma enumeração exaustiva.

Estas referências são válidas na família TTL e em várias das subfamílias CMOS, com excepção da subfamília 4000/4500, que se encontra obsoleta.

Como já se referiu anteriormente, existem várias subfamílias TTL com características diferenciadas no que diz respeito à potência gasta e à rapidez de funcionamento. A

TABELA 3.1: Descrição de alguns circuitos integrados digitais.

Referência	Descrição
7400	Quatro portas NAND de duas entradas
7402	Quatro portas NOR de duas entradas
7404	Seis portas NOT
7408	Quatro portas AND de duas entradas
7410	Três portas NAND de três entradas
7411	Três portas AND de três entradas
7420	Duas portas NAND de quatro entradas
7421	Duas portas AND de quatro entradas
7430	Uma porta NAND de oito entradas
7432	Quatro portas OR de duas entradas
7486	Quatro portas XOR

indicação da subfamília é feita pela colocação de letras na referência do integrado a seguir aos algarismos 74. Por exemplo, um 74LS00 é um circuito TTL da subfamília LS (do inglês, *Low power Shotcky*). A ausência de letra indica que se está perante a série normal TTL, que, actualmente, é já de uso muito restrito, por estar tecnologicamente obsoleta. Os catálogos dos fabricantes indicam as características de cada subfamília. Em CMOS, a indicação das subfamílias faz-se do mesmo modo que em TTL. Por exemplo, em CMOS existem duas subfamílias muito populares actualmente, a 74HC e a 74HCT (esta última, aceitando os níveis típicos da família CMOS, mantém a compatibilidade dos níveis eléctricos com a família TTL). O 74HC08 e o 74HCT08 têm a mesma distribuição de terminais que o 7408 ou o 74LS08.

3.1.2 PORTAS BÁSICAS

Neste livro não será analisada a estrutura interna das portas TTL, por apresentarem alguma complexidade e por o seu estudo não contribuir significativamente para os objectivos apontados.

Já no caso das portas CMOS é mais fácil perceber a sua estrutura e funcionamento. As portas CMOS utilizam transístores MOSFET de canal n e de canal p no mesmo circuito, razão da designação CMOS, em que o C provém do inglês *Complementary*, uma vez que se utilizam transístores de tipos complementares. Na Figura 3.3 representa-se o símbolo de um transístor de canal p e o seu equivalente funcional para o nível de abstracção que interessa no contexto deste livro. Tudo se passa como se o transístor fosse um

interruptor que, enquanto a tensão eléctrica na porta estiver abaixo de um limiar V_l, se conserva fechado, criando uma ligação entre os terminais de fonte e de dreno, e que, se a tensão na porta ultrapassar um outro limiar V_h, interrompe a ligação. Na Figura 3.4 representa-se um transístor de canal n e, igualmente, o seu equivalente funcional. Neste transístor, a ligação entre o dreno e a fonte só se estabelece quando é aplicada uma tensão que ultrapasse o limiar V_h à porta, mantendo-se o circuito aberto se a tensão na porta for inferior a V_l. O funcionamento dos transístores entre os valores dos dois limiares é irrelevante no caso de circuitos digitais.

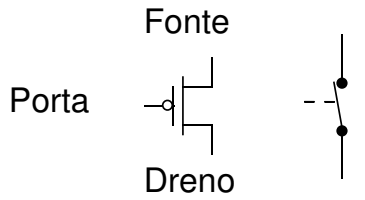


FIGURA 3.3: Transístor de canal p.

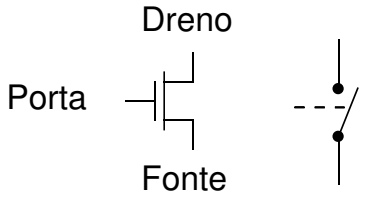


FIGURA 3.4: Transístor de canal n.

A estrutura de uma porta NOT na tecnologia CMOS está representada na Figura 3.5 e, de uma maneira simplificada, a relação entre a tensão apresentada à entrada, V_I, e a tensão resultante à saída, V_O. O índice I provem do inglês, *Input* e o índice O do inglês, *Output*. Nesta figura representa-se, para além da estrutura da porta, o seu circuito equivalente, no caso de se apresentar na entrada uma tensão $V_I = V_0$ inferior ao limiar V_l e uma tensão $V_I = V_1$ superior ao limiar V_h. Mantendo o nível de abstracção a que esta matéria está a ser analisada, verifica-se que, para uma entrada a um nível inferior a V_l, a porta responde com uma tensão V_O aproximadamente igual a V_{DD}, a designação habitual da tensão de alimentação na tecnologia CMOS, que é superior ao limiar V_h e, para uma entrada a um nível superior a V_h, responde com uma saída V_O a um nível inferior a V_l. Se se fizer corresponder o nível inferior a V_l a um 0 e o nível superior a V_h a um 1, o circuito comporta-se, de facto, como uma negação.

Nas Figuras 3.6 e 3.7 ilustram-se, respectivamente, a estrutura de uma porta NAND e de uma porta NOR em CMOS. É deixado ao leitor, como exercício, a análise destes circuitos utilizando os pressupostos usados no caso do NOT.

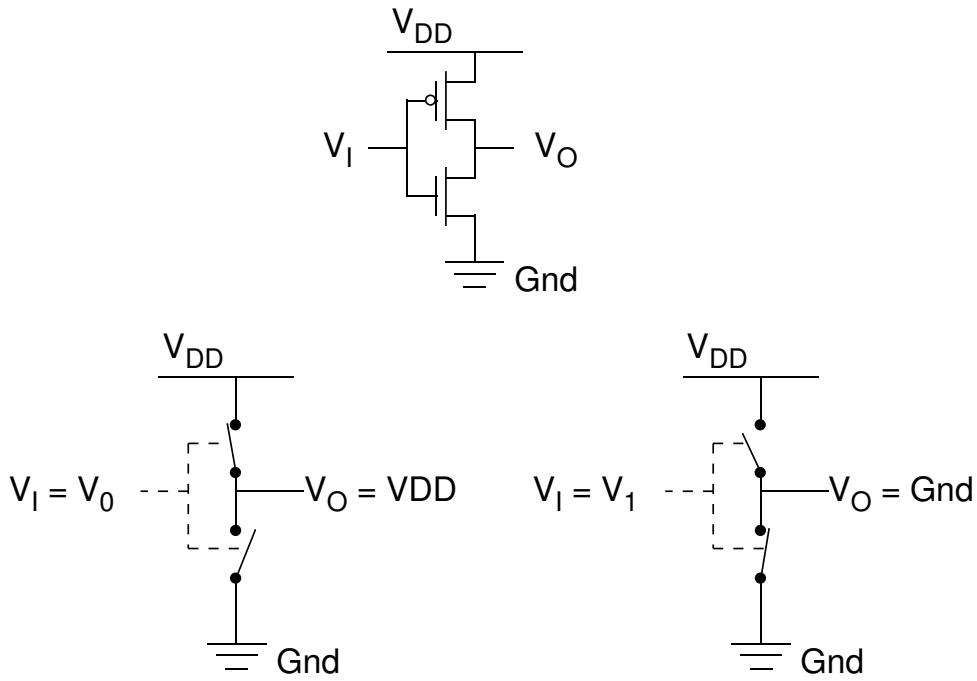


FIGURA 3.5: Estrutura de uma porta NOT em CMOS.

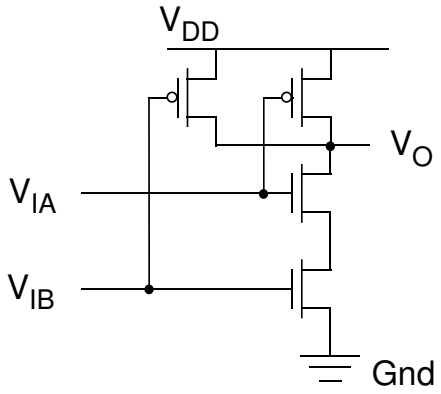


FIGURA 3.6: Estrutura de uma porta NAND em CMOS.

3.1.3 Níveis de Tensão e Níveis Lógicos

Os circuitos electrónicos que materializam as funções lógicas estudadas no Capítulo 2 esperam nas suas entradas e apresentam nas suas saídas valores de tensão eléctrica. Os fabricantes de circuitos lógicos não garantem valores fixos de tensão para representar os valores lógicos. O que garantem é um intervalo de tensões de tal modo que, se a saída de uma porta se encontrar nesse intervalo, pode considerar-se que está representado um determinado valor lógico.

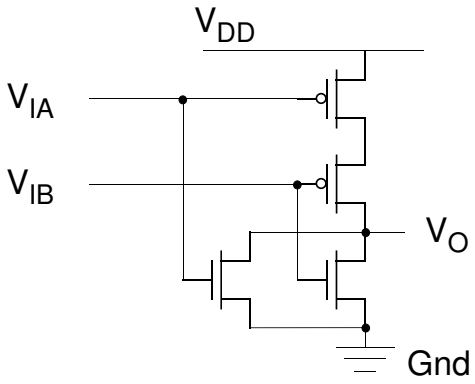


FIGURA 3.7: Estrutura de uma porta NOR em CMOS.

No caso da família TTL, a alimentação é uma tensão contínua de $V_{CC} = 5\text{V} \pm 5\%$. Os fabricantes garantem que, para lá dos momentos em que se verifica uma transição à saída da porta em resultado de uma mudança nas entradas, a tensão estará ou no valor *Alto*, também designado por H (do inglês, *High*), que corresponde a um valor dentro do leque $2{,}4\text{V}$ a 5V, ou no valor *Baixo*, também designado por L (do inglês, *Low*), que corresponde a um valor dentro do leque 0V a $0{,}4\text{V}$. A Figura 3.8 ilustra os dois níveis. O valor mínimo de tensão à saída da porta, que ainda corresponde a H, é usualmente designado por V_{OHmin}, *tensão alta mínima à saída*. O valor máximo de tensão à saída da porta, que ainda corresponde a L, é usualmente designado por V_{OLmax}, *tensão baixa máxima à saída*. Em TTL verificam-se, portanto, as definições das Equações 3.1.

$$\begin{aligned} V_{OHmin} &= 2{,}4\text{V} \\ V_{OLmax} &= 0{,}4\text{V} \end{aligned} \tag{3.1}$$

A tendência natural é associar o intervalo Alto (H) ao valor lógico 1 e o intervalo Baixo (L) ao valor lógico 0. Adiante voltar-se-á a esta questão, para mostrar que há outras hipóteses porventura mais interessantes.

É necessário ter em conta que existe aquilo que habitualmente se designa por ruído eléctrico e que mais não é do que a influência electromagnética de todos os campos que existem na zona onde está o circuito (emissões de rádio, TV, telemóveis, interferências provocadas por motores eléctricos, emissão electromagnética de outras ligações do mesmo circuito, etc.). Assim, é possível que à saída de uma porta exista um valor válido de tensão (por exemplo, $0{,}3\text{V}$) e à entrada de uma porta que lhe esteja ligada se observe um valor inválido (por exemplo, $0{,}5\text{V}$).

Para ter em conta que isso pode ocorrer e deve ser tolerado, as entradas das portas, ainda em TTL, têm uma especificação um pouco mais ampla, sendo os valores entre 2V e 5V interpretados como H, e os valores entre 0V e $0{,}8\text{V}$ interpretados como L, de

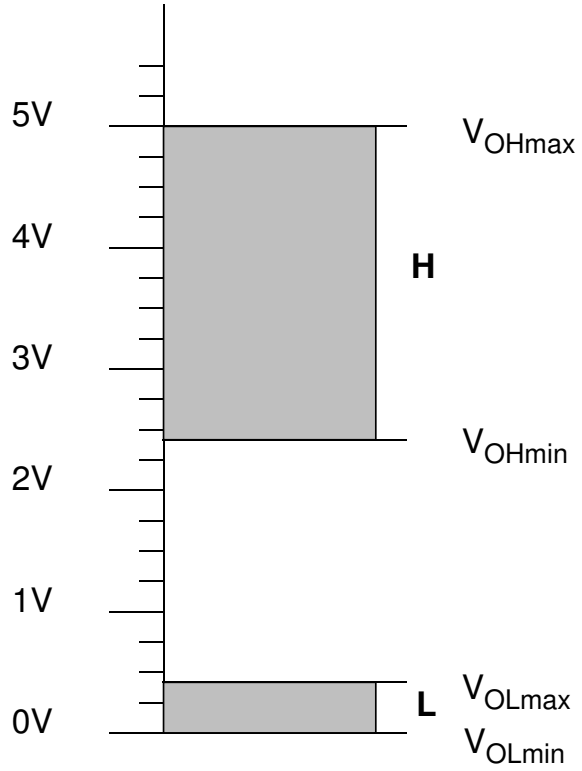


FIGURA 3.8: Definição dos níveis de tensão na saída de um circuito TTL.

acordo com as Equações 3.2. V_{IHmin} corresponde à *tensão alta mínima à entrada*, e V_{ILmax} à *tensão baixa máxima à entrada*. Na Figura 3.9 resume-se esta questão dos níveis.

$$\begin{aligned} V_{IHmin} &= 2\text{V} \\ V_{ILmax} &= 0,8\text{V} \end{aligned} \tag{3.2}$$

O intervalo de $0{,}4$V entre os valores extremos à saída e à entrada é chamado *margem de ruído*.

No caso da família 74HC, em CMOS, as tensões de alimentação são mais flexíveis que em TTL, podendo assumir valores entre 2V e 6V. Os níveis que referenciam os intervalos de tensão corespondentes ao H e ao L variam com a tensão de alimentação, como é óbvio. Os valores especificados, por exemplo, pela Philips para a tensão de alimentação de $4{,}5$V, por exemplo, à temperatura de $25\,^{\circ}$C, são listados nas Equações 3.3.

$$\begin{aligned} V_{IHmin} &= 3{,}15\text{V} \\ V_{ILmax} &= 1{,}35\text{V} \\ V_{OHmin} &= 4{,}32\text{V} \\ V_{OLmax} &= 0{,}26\text{V} \end{aligned} \tag{3.3}$$

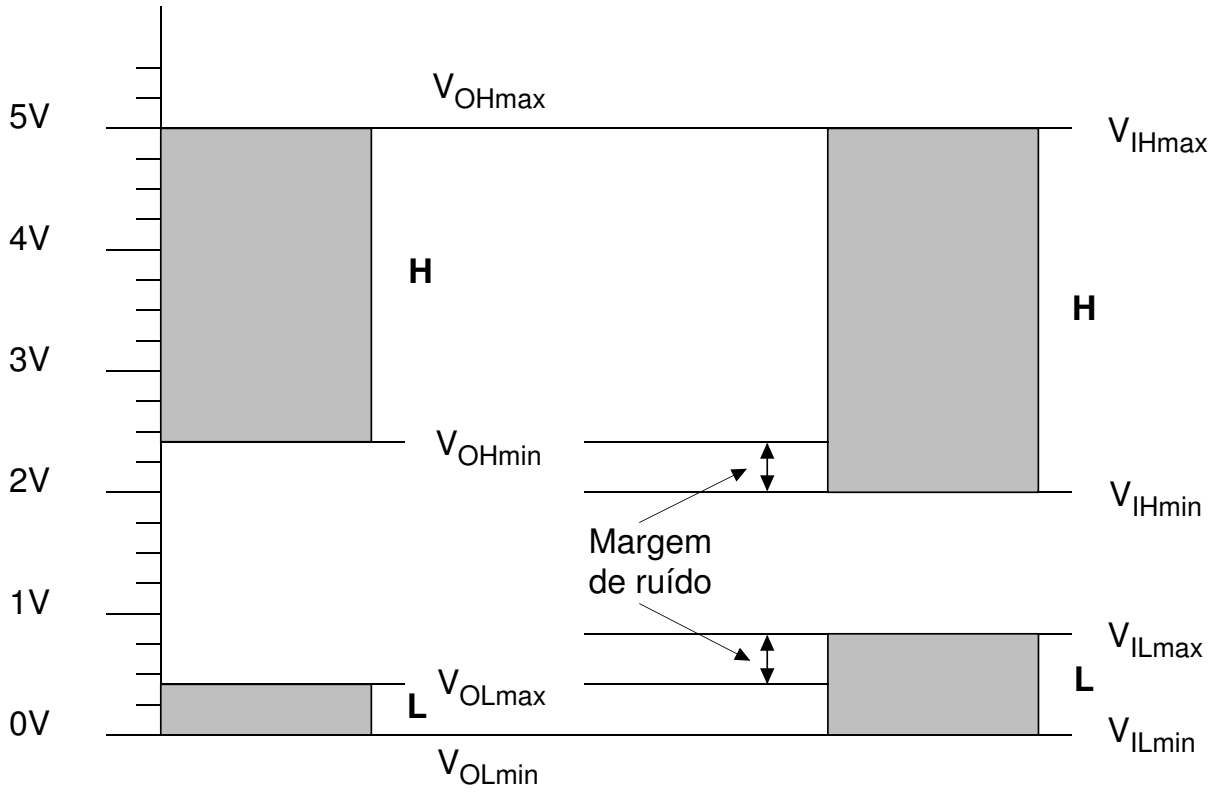


FIGURA 3.9: Definição dos níveis de tensão na família TTL à saída e à entrada de uma porta.

Nestas condições, a margem de ruído é de cerca de $1{,}15$V, o que é superior ao valor para TTL.

No caso das famílias CMOS, a subfamília 74HC tem as referências e a distribuição de terminais compatíveis com as famílias TTL. Não é, no entanto, electricamente compatível, como se infere das Equações 3.1, 3.2 e 3.3. A família 74HCT é uma variante da HC com níveis lógicos compatíveis com a família TTL.

3.1.4 ATRASOS

Dado que os circuitos integrados são dispositivos reais e físicos, não reagem instantaneamente a mudanças nas suas entradas. Isso quer dizer que, no seu comportamento ao longo do tempo, podem ser observados atrasos entre alterações provocadas nas entradas de uma porta e as eventuais alterações produzidas na sua saída. Na Figura 3.10 analisa-se um exemplo de uma porta AND num circuito 7408.

A figura ilustra um *diagrama temporal*. Um diagrama temporal é uma representação esquemática da evolução de sinais ao longo do tempo. No caso da figura, mostra-se a

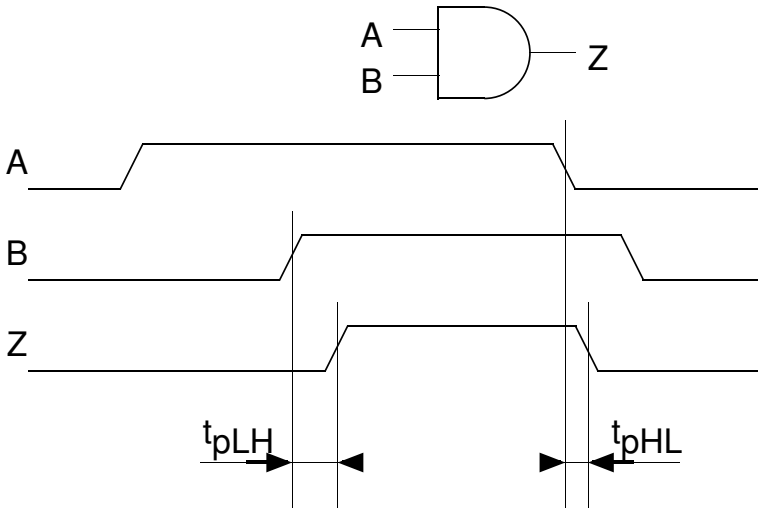


FIGURA 3.10: Exemplo de atrasos numa porta AND.

evolução dos níveis à entrada e saída de uma porta AND. Os sinais não mudam instantaneamente. Aqui a representação disso é feita por uma linha inclinada na transição. Na realidade a transição é mais complexa, mas ao nível de abstracção a que é feita esta análise, esta forma de representação é suficiente. Por vezes, nos diagramas temporais, omite-se esta representação de uma transição não instantânea quando isso ajuda a torná-los mais claros, sendo as transições representadas por linhas verticais.

Pode ser observado que na porta a primeira alteração da entrada A não tem qualquer efeito na saída, uma vez que a entrada B permanece a 0, mantendo a saída a 0. É um caso claro em que uma alteração numa entrada não afecta a saída e em que, portanto, não se pode falar de atraso.

A transição de 0 para 1 na entrada B leva a que a saída transite para 1. No entanto, a resposta da saída Z à mudança em B ocorre apenas após um certo tempo, representado na figura por t_{pLH}. Este tempo, chamado *tempo de atraso*, *tempo de reacção* e, por vezes, *tempo de propagação*, é uma consequência dos fenómenos físicos necessários para a realização da comutação da porta. O índice pLH na designação do tempo de atraso referido indica que se trata da transição da saída de L para H. Os tempos de atraso nas transições de L para H não são obrigatoriamente iguais aos das transições inversas de H para L. Por isso, na figura fez-se uma representação em que t_{pLH} é diferente de t_{pHL}. Na Secção 3.4 voltar-se-á, mais pormenorizadamente a este tema.

3.1.5 Energia

O funcionamento de circuitos electrónicos obriga ao consumo de energia. Essa energia dissipa-se, aquecendo os circuitos. Isto coloca dois problemas. Por um lado, é necessário garantir aos circuitos a energia necessária para o seu funcionamento. Por outro, é necessário garantir que há mecanismos de arrefecimento que conservam estável a temperatura de um circuito, para evitar que os componentes se degradem. Como é óbvio, com a tendência actual para o desenvolvimento de produtos portáteis, estas questões têm vindo a tornar-se cada vez mais importantes.

Pela sua estrutura, as portas TTL, e sobretudo as ECL, são portas que têm consumos de energia relativamente elevados. As portas CMOS, pelo contrário, caracterizam-se por dissipar menos energia que as anteriores. Isso deve-se a que as portas CMOS dissipam a maior parte da energia durante o tempo em que estão a comutar de nível. Se uma porta CMOS mantiver o nível de saída estável durante algum tempo, praticamente não consome energia. O que se paga, em contrapartida, é um consumo que cresce com o número de transições por unidade de tempo, isto é, com o *nível de actividade*. A dissipação de potência relativa à comutação, a parte mais importante, é dada pela Expressão 3.4.

$$P_d = C_L V_{DD}^2 \alpha \tag{3.4}$$

Nesta expressão, P_d é a potência dissipada, C_L a capacidade vista pela saída da porta, V_{DD} a tensão de alimentação, e α o nível de actividade da porta, ou seja, o número médio de transições por unidade de tempo.

Uma forma que tem vindo recentemente a ser utilizada pelos fabricantes para diminuir o consumo dos circuitos lógicos, apesar do aumento das suas frequências de funcionamento e, portanto, do nível de actividade das portas, é a diminuição da tensão de alimentação. Esta solução que, de acordo com a Expressão 3.4, parece óbvia, tem o inconveniente de diminuir a margem de ruído e a velocidade dos circuitos, levando à necessidade de equilibrar cuidadosamente as vantagens e inconvenientes da solução.

3.1.6 Dispositivos Especiais

Para além das portas básicas, é frequente em sistemas digitais a utilização de outros dispositivos, que são importantes para garantir certo tipo de funcionalidades. Analisam-se seguidamente, pela sua importância, os *buffers de três estados* (em inglês, *tri-state buffer*) e as *portas de passagem* (em inglês, *transmission gates*).

Buffers DE TRÊS ESTADOS

O *buffer* de três estados é um dispositivo com uma entrada e uma saída de dados e ainda uma entrada de controlo. Quando a entrada de controlo está a H, o valor da saída é igual ao valor que se apresenta na entrada de dados. Quando, pelo contrário, a entrada de controlo está a L, então a saída está em alta impedância, isto é, está desligada de toda a tensão eléctrica. A Tabela 3.2 resume o funcionamento deste dispositivo. A Figura 3.11 mostra os símbolos utilizados para este circuito. No símbolo da direita o triângulo é o sinal de saída de três estados.

TABELA 3.2: Tabela de um *buffer* de três estados.

C	I	O
L	X	sem tensão
H	L	L
H	H	H

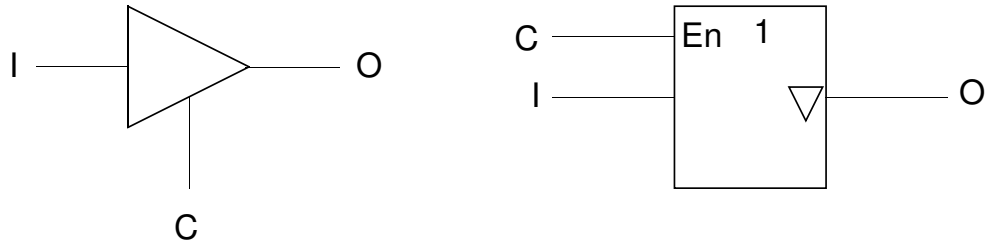


FIGURA 3.11: Símbolos do *buffer* de três estados.

A Figura 3.12 ilustra a estrutura interna de um *buffer* de três estados. As portas representadas na figura pretendem simplificar o circuito, de modo a tornar mais fácil a compreensão da sua estrutura, mas, como é evidente, são, por sua vez, implementadas com um conjunto de transístores. Repare-se, por comparação com os circuitos anteriormente apresentados, que, quando a linha C está a L, ambos os transístores de saída estão cortados, isto é, os interruptores equivalentes estão abertos e a saída não fica ligada nem a V_{DD} nem a Gnd, ficando, como se pretende, desligada de toda a tensão eléctrica. Quando a entrada C está activa, isto é, em H, a saída reproduz o nível da entrada.

Há duas aplicações fundamentais para os *buffers* de três estados. Por um lado, potenciam a utilização de linhas bidireccionais. Por outro, permitem interligar um de vários sinais a uma dada linha.

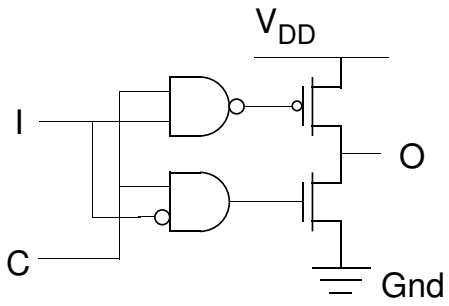


FIGURA 3.12: Estrutura interna de um *buffer* de três estados em tecnologia CMOS.

Considere-se o logigrama da Figura 3.13. Neste logigrama é possível observar que existem duas linhas que ligam duas entidades que se encontram, uma, à direita e, outra, à esquerda do logigrama. Se a variável *DIR* estiver a H, então o *buffer* de três estados da

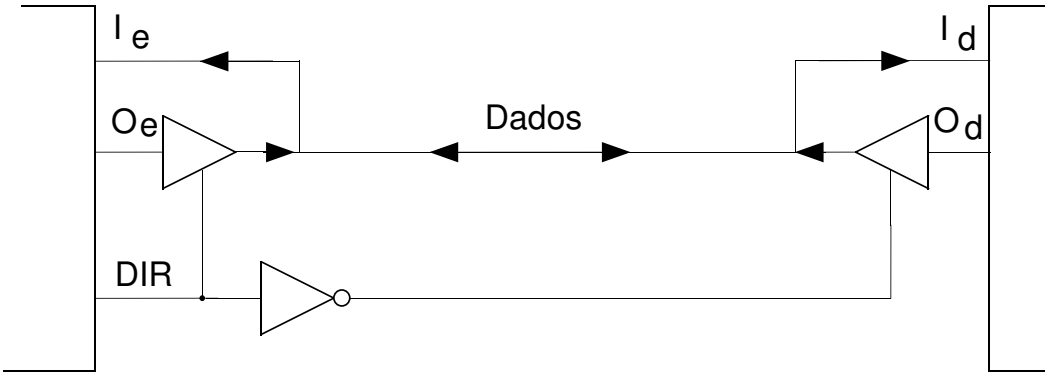


FIGURA 3.13: Linha bidireccional.

direita está inibido e, consequentemente, porta-se como se não estivesse presente. Em contrapartida, o *buffer* da esquerda está activo, e a informação que se encontrar presente na linha O_e é transmitida pela linha $Dados$ para a parte direita, onde pode ser lida através da entrada I_d. Se a linha *DIR* estiver a L, observa-se exactamente o contrário. Este é, assim, o princípio de uma linha bidireccional, que permite trocar, sobre as mesmas linhas, informação nas duas direcções (uma de cada vez, claro). É evidente que se podem ter várias linhas de dados que interliguem as duas entidades, sendo todas comandadas solidariamente pelo mesmo sinal de controlo *DIR*. Este tipo de aplicação é muito importante, como se verá, na arquitectura de computadores.

Considere-se, agora, o segundo tipo de aplicação dos *buffers* de três estados, a interligação de um de vários sinais a uma dada linha. Observe-se o logigrama da Figura 3.14.

Considere-se a linha Sel ao nível L. Nestas circunstâncias, o *buffer* inferior é desactivado e o superior está activo, colocando o valor da linha I_0 na linha de saída O. Inversamente, se Sel assumir o nível H, será o *buffer* inferior o que está activo, e o superior, inibido,

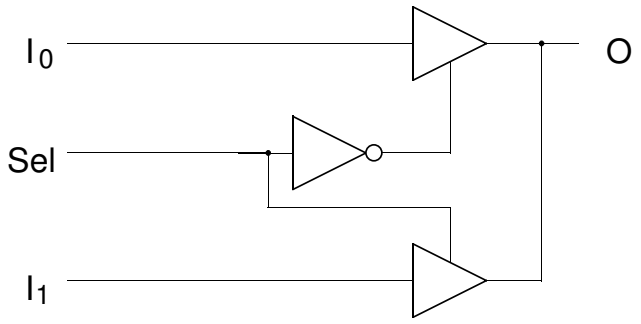


FIGURA 3.14: Utilização de *buffers* de três estados para selecção de sinais.

fazendo $O = I_1$. A estrutura apresentada permite, assim, seleccionar o valor presente numa das linhas de entrada para colocar na saída. É fácil generalizar esta configuração para mais de duas linhas de entrada. Ir-se-á encontrar esta funcionalidade, de novo, no Capítulo 4, quando se analisarem os multiplexadores.

ELEMENTOS INCOMPLETOS

Um outro tipo de dispositivo, que é interessante considerar, é o conjunto de portas cuja saída apenas consegue impor um dos estados lógicos. São portas com saídas do tipo *colector aberto* (em inglês, *open-colector*), em tecnologia TTL, ou do tipo *dreno aberto* (em inglês, *open-drain*), em tecnologia CMOS.

À primeira vista, este tipo de portas parece pouco útil. No entanto, como se verá, permite em certos casos específicos resolver problemas que, de outro modo teriam solução mais complexa. Para compreender a motivação para o desenvolvimento deste tipo de portas é necessário compreender uma limitação tecnológica das portas clássicas. De facto, com portas clássicas não é possível interligar directamente saídas de portas, quer se esteja a trabalhar em TTL, quer em CMOS.

Considere-se a situação ilustrada na Figura 3.15, em que duas portas NOT em tecnologia CMOS têm as suas saídas directamente interligadas. Se as duas entradas V_{I0} e V_{I1} estiverem ao mesmo nível, o funcionamento do circuito não levanta qualquer problema, uma vez que as duas saídas também estão ao mesmo nível, e V_O apresenta o valor que qualquer das duas saídas apresentaria independentemente. No entanto, se os valores das entradas forem diferentes, ocorre na saída um problema para o qual o circuito das portas não está preparado. Nessas circunstâncias, o que acontece é que uma das portas «liga» o transístor de cima enquanto a outra «liga» o de baixo. Passa então a haver um caminho directo entre a alimentação V_{DD} e a massa do circuito (Gnd) com uma resistência eléctrica muito baixa, provocando o aparecimento de uma corrente eléctrica muito

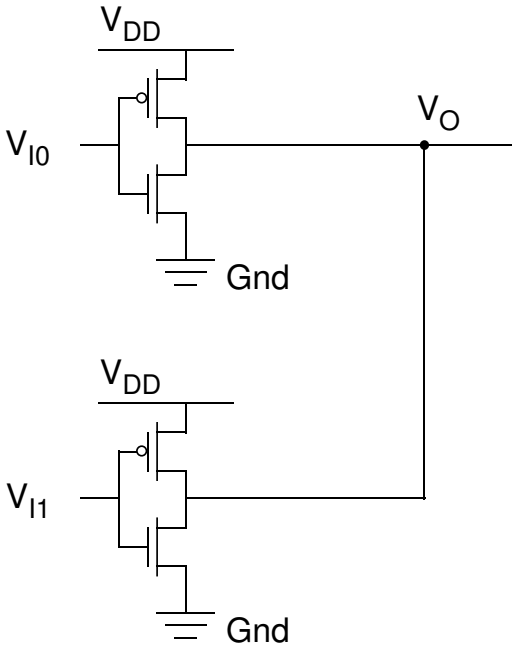


FIGURA 3.15: Circuito com duas portas NOT com saídas interligadas.

elevada que destrói os circuitos. É verdade que o circuito ilustrado não parece, em si, à primeira vista, particularmente interessante, não sendo sequer claro que função lógica se pretende implementar com ele.

Mas considere-se agora, de novo, a mesma estrutura, mas com portas que não têm o transístor superior, como se ilustra na Figura 3.16. Na parte esquerda da figura é ilustrada uma porta NOT com dreno aberto (este nome advêm do facto de o dreno do transístor inferior estar em aberto sem qualquer ligação). É ainda ilustrada uma resistência que não faz parte da porta, sendo acrescentada exteriormente. Uma vez que o único transístor em cada porta é o inferior, consegue-se impor um nível Low na saída de cada porta. No entanto, se não for utilizada a ligação a V_{DD} através da resistência e se a porta pretender colocar a sua saída em $High$, não há maneira de o fazer, uma vez que não existe ligação à tensão positiva V_{DD}. Para conseguir ter a saída com o valor $High$, utiliza-se então a resistência externa, que, quando o transístor inferior da porta não está a impor o nível Low, permite que a saída da porta esteja ligada à tensão positiva, assumindo o nível $High$.

Nestas condições, a estrutura da parte direita da figura, que replica, para portas deste tipo, o circuito da Figura 3.15, já não apresenta qualquer problema para a funcionalidade das portas, uma vez que nunca é possível obter um caminho de baixa resistência entre V_{DD} e Gnd. É agora possível perceber a função lógica materializada pela ligação directa das saídas das portas. Sempre que uma das portas, por si, pretende colocar a sua saída a Low, há um transístor que interliga V_O e Low, colocando a saída ao nível Low. Só se

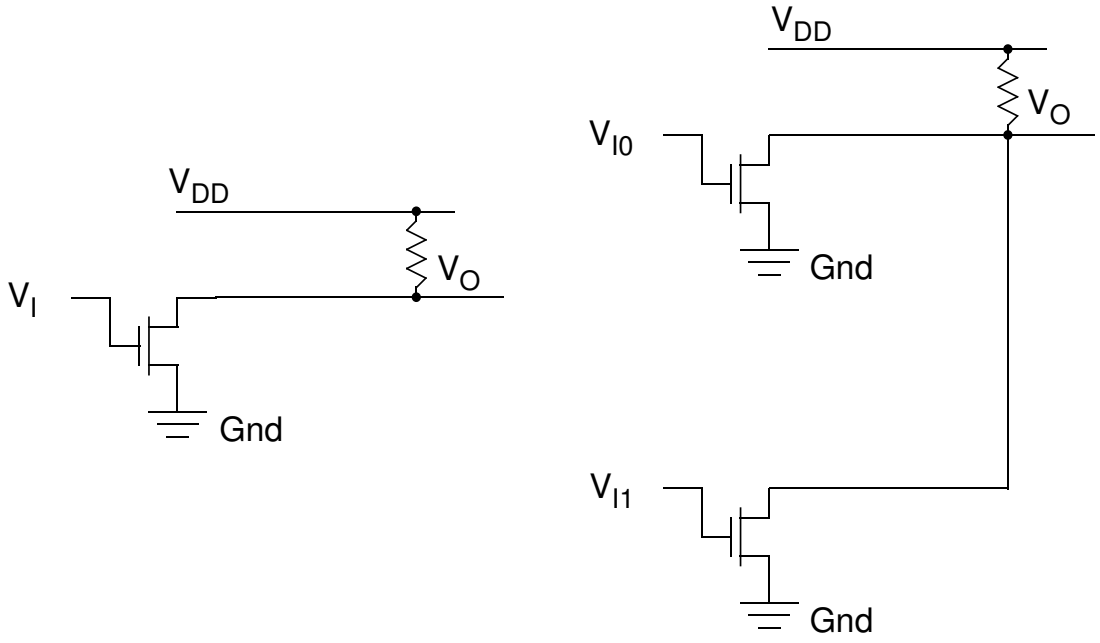


FIGURA 3.16: Porta NOT com saída de dreno aberto e interligação de duas portas desse tipo.

nenhuma das portas pretender colocar a sua saída em Low é que a ligação através da resistência impõe o nível $High$ na saída V_O. É, assim, fácil perceber que se está perante um AND virtual, apesar de não existir qualquer porta. Este tipo de ligação é usualmente designado em inglês por *wired-and*.

A ideia de que a utilização de ligações deste tipo permite utilizar menos portas, e que, portanto, deve ser comum e recomendável, não corresponde à verdade. Há várias razões que levam a isso. Uma delas é que é mais fácil utilizar circuitos com maior nível de integração que permitem a interligação interna de muito mais portas. Outra é que este tipo de ligação é, comparativamente, lento.

No entanto há circunstâncias em que o uso de ligações de dreno aberto (ou de colector aberto em TTL) é particularmente útil. Um caso corrente é o de cada um dos circuitos se encontrar em placas separadas, que podem ser inseridas, através de fichas, numa placa global de interligação. Nessas circunstâncias, basta colocar a resistência na placa de interligação e ligar o número de placas de circuito necessário para uma dada aplicação, garantindo o funcionamento, qualquer que seja a configuração. Mais adiante, na Secção 3.5, outra aplicação será apresentada.

Portas de passagem

Um tipo de dispositivo de grande utilidade na concepção de alguns circuitos é a chamada *porta de passagem*. A porta de passagem é um circuito constituído por dois transístores complementares ligados, como se ilustra na Figura 3.17, que permitem, quando ambos activados, a passagem de sinais em toda a gama de tensão entre a massa e a tensão de alimentação, permitindo a passagem de sinais dentro ou fora dos níveis digitais do CMOS. Quando os dois transístores estão desactivados, a saída da porta está isolada da entrada e, portanto, desligada de toda a tensão eléctrica. Adiante se poderá verificar a utilidade destes dispositivos.

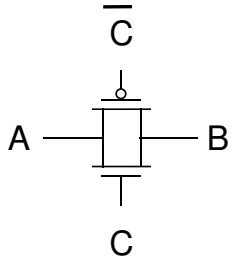


Figura 3.17: Circuito da porta de passagem.

A Figura 3.18 ilustra o símbolo pelo qual são habitualmente representados estes circuitos.

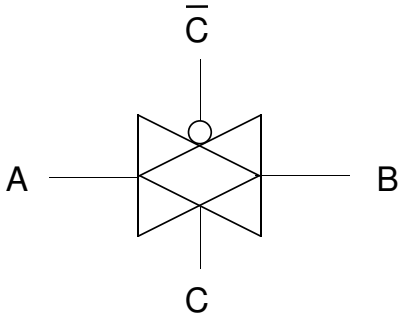
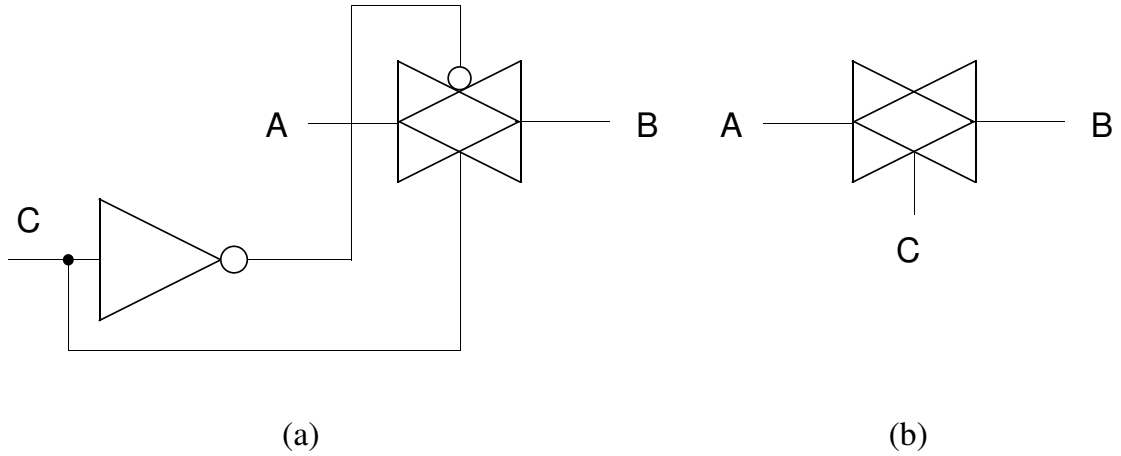


Figura 3.18: Símbolo de uma porta de passagem.

As portas de passagem são muitas vezes utilizadas com a configuração de ligações ilustrada na Figura 3.19(a). Aquela configuração é habitualmente representada pelo símbolo da Figura 3.19(b).

A funcionalidade deste tipo de dispositivo e do *buffer* de três estados é, em muitos casos, compatível. De facto, se na saída de uma porta lógica for incluída uma porta de



(a) (b)

FIGURA 3.19: (a) Configuração típica de aplicação de uma porta de passagem. (b) Símbolo habitual da configuração anterior.

passagem, o comportamento global obtido é semelhante ao da inclusão de um *buffer* de três estados. Os dois dispositivos, têm, porém, características diferentes que, sem aprofundar o seu comportamento físico (o que está fora do âmbito deste livro), não são de fácil compreensão. Por outro lado, uma porta de passagem é bidireccional, o que não acontece no *buffer* de três estados. Há ainda que recordar que a porta de passagem não é um dispositivo puramente digital, possibilitando a passagem de sinais analógicos.

3.2 REALIZAÇÃO DE FUNÇÕES LÓGICAS USANDO PORTAS BÁSICAS

As funções lógicas NOT, OR, NOR, etc., estudadas no Capítulo 2, podem, naturalmente, ser implementadas, utilizando portas lógicas do tipo das que têm vindo a ser apresentadas. Há algumas convenções, porém, que terão de ser explicitadas, para que isso possa ser feito de forma não ambígua.

3.2.1 LÓGICA POSITIVA E NEGATIVA

Como se viu na Secção 3.1.3, uma porta tem um funcionamento binário, assumindo um de dois leques de nível de tensão eléctrica, quer nas suas entradas, quer na sua saída. Um NOT, por exemplo, produz na sua saída um valor H, sempre que a sua entrada está a L. A utilização de circuitos deste tipo para dar suporte a funções lógicas resulta do facto de que, em ambos os contextos, circuitos e funções lógicas, se está a trabalhar com conjuntos de dois elementos. É natural, como também já se referiu, associar o nível alto

H num terminal de um circuito ao valor lógico 1, e o nível baixo L, ao valor lógico 0. Essa correspondência está ilustrada na Tabela 3.3.

TABELA 3.3: Lógica positiva.

Valor lógico	Nível de tensão
0	L
1	H

Esta convenção é denominada *lógica positiva* e é muito utilizada, talvez por ser muito intuitiva. Analise-se um pouco mais a fundo este conceito. Uma porta AND de um circuito 74LS08, por exemplo, tem o funcionamento, especificado pelos fabricantes, descrito na Tabela 3.4, em que A e B são as entradas, e Z a saída. Repare-se que os fabricantes especificam o funcionamento da porta em níveis de tensão, uma vez que é o que podem, de facto, garantir.

TABELA 3.4: Tabela de uma porta do circuito 74LS08.

A	B	Z
L	L	L
L	H	L
H	L	L
H	H	H

Aplicando a correspondência da Tabela 3.3, obtém-se a clássica tabela da função lógica AND, como se pode observar na Tabela 3.5.

TABELA 3.5: Tabela de uma porta do circuito 74LS08 interpretada em lógica positiva.

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

Como é evidente, a lógica positiva não é a única convenção possível. A alternativa é a *lógica negativa* descrita na Tabela 3.6.

TABELA 3.6: Lógica negativa.

Valor lógico	Nível de tensão
0	H
1	L

Repare-se que, utilizando a lógica negativa, a interpretação do funcionamento do 74LS08 é diferente. Interpretando a Tabela 3.4, que descreve o funcionamento eléctrico do 74LS08, à luz da convenção da lógica negativa, obtém-se a Tabela 3.7.

TABELA 3.7: Tabela de uma porta do circuito 74LS08 interpretada em lógica negativa.

A	B	Z
1	1	1
1	0	1
0	1	1
0	0	0

Esta tabela corresponde, como se pode observar, à função OR. Isso quer dizer que, em lógica negativa, as portas incluídas no 74LS08 são interpretadas como circuitos OR. É óbvio que o circuito electrónico funciona sempre da mesma maneira. A interpretação que se faz do seu funcionamento é que é diferente.

É, por tudo isto, natural que a designação dada pelos fabricantes às portas incluídas no 74LS08 seja *Positive* AND *Gate*, numa alusão a que a interpretação do circuito como um AND está associada ao pressuposto de se estar a usar lógica positiva.

A lógica negativa é muito menos usada que a lógica positiva. Actualmente, porém, reconhece-se que nenhuma das duas convenções é a ideal. Para mostrar porquê, considere-se um exemplo.

Admita-se que se pretende construir um alarme para um automóvel que assinale, accionando um besouro, que o carro circula com uma porta aberta ou que está estacionado com as luzes acesas. A obtenção de uma função lógica que dê suporte ao comportamento pretendido é fácil.

Se o carro tiver quatro portas, o estado de cada porta pode ser representado por uma variável Pi de um conjunto que vai de $P1$ a $P4$. Cada uma destas variáveis estará a 1 quando a respectiva porta estiver aberta. É fácil, agora, verificar que a Expressão 3.5

corresponde a uma função que indica, quando a 1, se existe pelo menos uma porta aberta.

$$P = P1 + P2 + P3 + P4 \tag{3.5}$$

Do mesmo modo, pode ser definida a variável L, que assume o valor 1, quando as luzes estão acesas, e a variável C, uma variável que está a 1 quando a chave de ignição está ligada. É fácil de ver que o alarme pretendido pode ser descrito pela função A com a Expressão 3.6.

$$A = C \ P + \overline{C} \ L \tag{3.6}$$

Substituindo P pela expressão em termos das variáveis Pi, obtém-se a Expressão 3.7.

$$A = C \ (P1 + P2 + P3 + P4) + \overline{C} \ L \tag{3.7}$$

A função A é representável pelo logigrama da Figura 3.20.

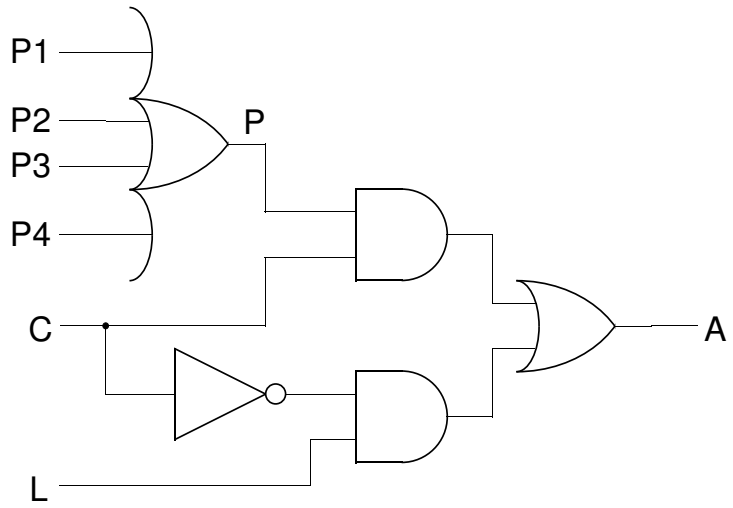


FIGURA 3.20: Logigrama da função A.

É fácil reconhecer neste logigrama a função atrás referida, e a lógica do problema está perfeitamente visível. Está a partir-se do princípio que estão acessíveis as variáveis $P1$ a $P4$, C e L, bem como uma entrada A na sirene de alarme. Se todas as variáveis estiverem implementadas usando lógica positiva, o logigrama pode corresponder mais ou menos directamente a um circuito que implemente o sistema de alarme. Para as variáveis estarem implementadas em lógica positiva, porém, isso significa que temos de ter sinais que materializam as variáveis que estão a H quando as variáveis estão a 1, e a L, quando estão a 0. Acontece, porém, que nem sempre isso é possível e que é frequente que certas variáveis estejam representadas em lógica negativa. A título de exemplo, considere-se que, quando uma porta do carro está aberta, o fio que dá suporte

físico à respectiva variável Pi assume o nível L por razões que têm a ver com o circuito eléctrico do carro. Do mesmo modo, considere-se que o mesmo acontece com a variável C. Se se analisar a situação em termos de lógica positiva, isso significa que aquilo a que se tem acesso é, na realidade, a essas variáveis negadas. O logigrama terá de ser adaptado em conformidade, passando à situação ilustrada na Figura 3.21.

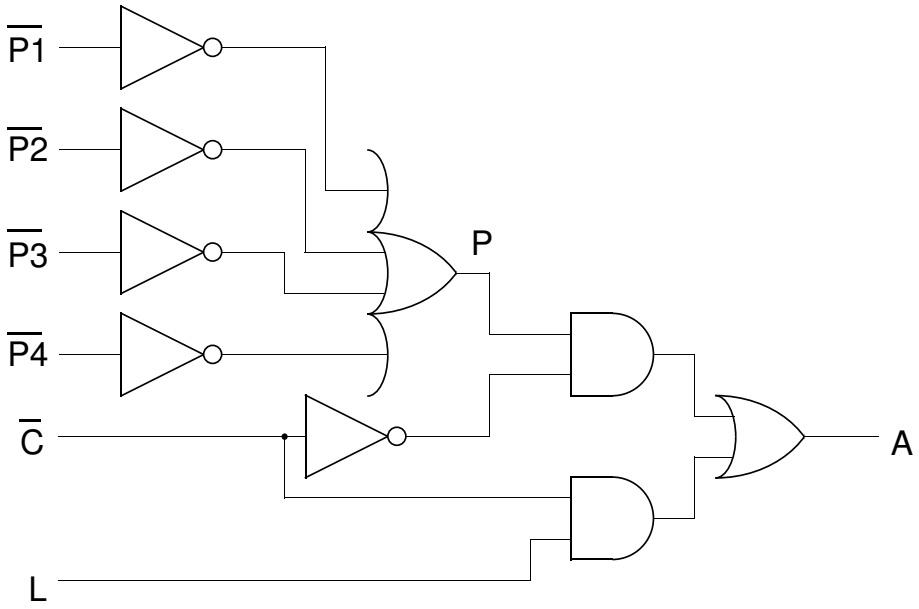


FIGURA 3.21: Logigrama adaptado da função A, utilizando lógica positiva.

No entanto, este logigrama é demasiado complexo e é possível alterá-lo para ficar mais simples e para usar menos circuitos integrados diferentes.

— O OR com as negações pode ser reconvertido num NAND de quatro entradas por aplicação de uma das leis de Morgan.

— A estrutura AND-OR do final pode ser substituída por uma estrutura NAND-NAND de novo por aplicação das leis de Morgan.

— Como, após as alterações anteriores, o circuito usa apenas três portas NAND de um integrado que tem quatro, o NOT pode ser substituído por esse quarto NAND, prescindindo-se, assim, de um integrado com portas NOT.

A Figura 3.22 mostra o logigrama já simplificado, com a indicação dos integrados envolvidos e dos terminais ligados, sendo, portanto, já um *esquema eléctrico*. Este circuito é, evidentemente, muito mais simples que o anterior, uma vez que usa apenas dois circuitos integrados diferentes em vez de quatro, como o anterior. O único problema é

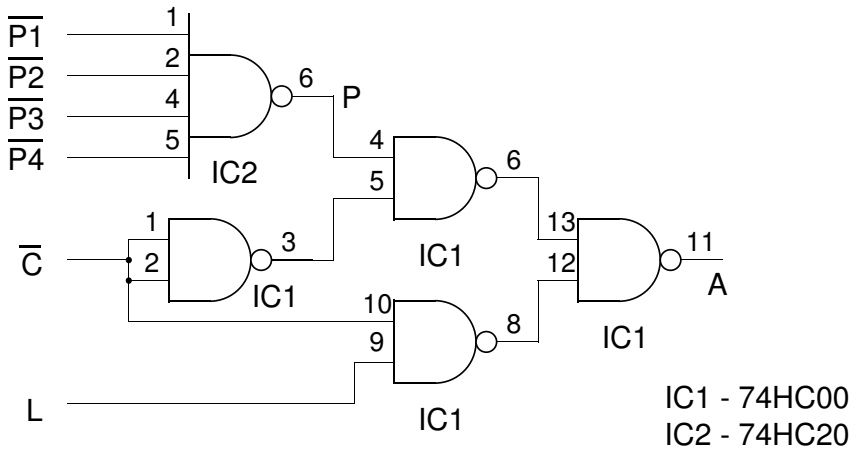


FIGURA 3.22: Esquema eléctrico do circuito que implementa a função A.

que, observando este circuito, não é de modo algum evidente a lógica inicial do problema. Não só as diversas funções foram alteradas, como também algumas variáveis de entrada aparecem negadas, o que para um observador desprevenido não tem causas evidentes. O problema resulta de, para o circuito se adaptar às condições exteriores, por um lado, e para o simplificar, por outro, se ter alterado o tipo de funções (e, portanto, de integrados) usadas. Essa alteração é conveniente para simplificar o circuito, mas tem as consequências negativas apontadas.

3.2.2 LÓGICA DE POLARIDADE

Na secção anterior ficou claro que é possível simplificar os circuitos digitais usados para implementar funções lógicas, à custa de tornar menos óbvia para o observador a lógica do problema. É, no entanto, possível, manipulando a forma de representação do circuito no esquema, representar este, de modo a não perder informação em relação ao logigrama inicial. Com essa finalidade criou-se a *lógica de polaridade*, forma alternativa às lógicas positiva e negativa. Em lógica de polaridade procura-se optar em cada ponto do circuito pelo tipo de lógica que melhor contribua para clarificar o seu funcionamento. Claro que isso obriga a que, em cada ponto do circuito, se indique o tipo de convenção utilizado.

Voltando ao exemplo que tem vindo a ser estudado, recorde-se que as variáveis C e $P1$ a $P4$ estavam activas, isto é, em termos de álgebra de Boole, assumiam o valor 1, quando o sinal eléctrico que lhes dá suporte está no nível L. Assume-se essa circunstância, indicando-a explicitamente na designação dos respectivos sinais eléctricos. Por exemplo, $P1\_L$ indica que a variável $P1$ é representada por um sinal $P1\_L$, que estará a L, sempre que a variável $P1$ está *activa* (isto é, assume o valor 1) e, claro, o sinal estará

a H, sempre que $P1$ estivar inactiva (assumindo o valor 0). Repare-se que a designação de um sinal assume agora um conteúdo semântico interessante. Para além da designação do sinal e, implicitamente, da variável a que ele dá suporte físico, o nome do sinal indica o tipo de relação entre o sinal e a variável. De acordo com esta metodologia, as quatro variáveis que indicam o estado das portas e da chave de ignição serão representadas pelos sinais $P1\_L$, $P2\_L$, $P3\_L$, $P4\_L$ e C_L. Repare-se que, ao contrário do que aconteceu quando o problema se abordou no contexto da lógica positiva, não há aqui o conceito de negação da variável mas apenas uma correspondência entre o comportamento da variável e o do sinal eléctrico.

Claro que, agora, é necessário indicar o que se passa com as restantes variáveis, isto é, qual é a representação dos sinais eléctricos de suporte às variáveis L, P e A. Como estas variáveis estão suportadas em sinais que estão ao nível H quando elas estão activas, isto é, com o valor lógico 1, os sinais correspondentes às variáveis L, P e A devem ser representados respectivamente por L_H, P_H e A_H.

De algum modo, pode assumir-se que, por exemplo, no caso de $P1\_L$ se representa a variável $P1$ em lógica negativa, enquanto que, para o caso de A_H, se pode assumir que a variável A está representada em lógica positiva. Mas a metodologia da lógica de polaridade é mais rica em significado semântico do que uma simples indicação deste tipo, sendo a relação entre o sinal e as portas a que está ligado importante para a definição desse significado, como se verá adiante.

Considere-se agora o circuito OR de quatro entradas no logigrama inicial. Se as variáveis de entrada são suportadas em sinais com as características descritas, então isso tem de ser assumido pela entrada do OR. A forma como tal se indica é através de um símbolo

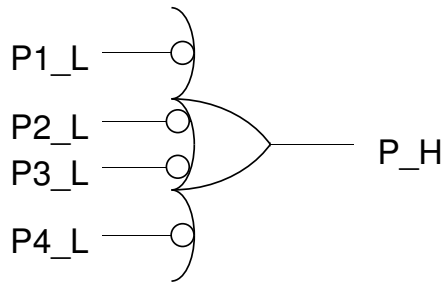


FIGURA 3.23: Porta OR do logigrama da função A representado em lógica de polaridade.

de inversão de polaridade, nessas entradas, como se pode ver na Figura 3.23. O símbolo indicador de inversão de polaridade é o círculo que indica também a negação nas portas até agora utilizadas. Este duplo significado será tratado adiante mais pormenorizadamente. Um símbolo de inversão de polaridade alternativo usado com frequência

é um triângulo como se mostra na Figura 3.24. Neste livro, usar-se-á apenas o primeiro símbolo.

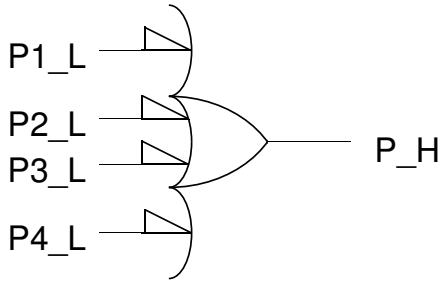


FIGURA 3.24: Porta OR do logigrama da função A representado em lógica de polaridade com o símbolo de inversão de polaridade alternativo.

Repare-se que a relação que existia entre as variáveis Pi e a função OR no logigrama da Figura 3.20 se mantém aqui. De facto, uma vez que, em ambos os casos, a um valor activo das variáveis e das entradas da porta corresponde um nível L do sinal, a porta «vê» directamente o valor das variáveis à sua entrada. Do mesmo modo se procede em relação à variável C, assinalando com indicadores de inversão de polaridade as entradas que lhe estão ligadas. O logigrama global assim alterado está representado na Figura 3.25. Repare-se que, neste logigrama, se mantém explícita a lógica inicial do problema, não tendo sido alterados os símbolos de base das portas lógicas utilizadas, existindo já, porém, a informação acerca do nível em que as entradas são activas. A diferença fundamental em relação ao método anteriormente exposto é o de não confundir inversões de polaridade de sinais resultantes de condicionantes físicas com negações lógicas.

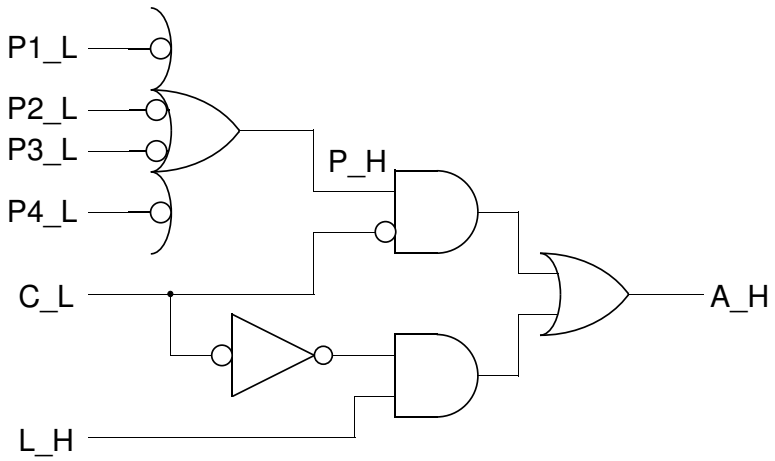


FIGURA 3.25: Evolução do logigrama que representa a função A.

A passagem a esquema eléctrico implica identificar os integrados que possam dar su-

porte às funções do logigrama. No caso da função OR de quatro entradas, por exemplo, parte-se da tabela de verdade do OR, fazendo corresponder as entradas e a saída aos níveis eléctricos correctos, de acordo com a respectiva convenção. Esse procedimento está ilustrado na Tabela 3.8.

TABELA 3.8: Tabela de conversão para lógica de polaridade do OR do circuito.

$P1$	$P2$	$P3$	$P4$	P	$P1\_L$	$P2\_L$	$P3\_L$	$P4\_L$	P_H
0	0	0	0	0	H	H	H	H	L
0	0	0	1	1	H	H	H	L	H
0	0	1	0	1	H	H	L	H	H
0	0	1	1	1	H	H	L	L	H
0	1	0	0	1	H	L	H	H	H
0	1	0	1	1	H	L	H	L	H
0	1	1	0	1	H	L	L	H	H
0	1	1	1	1	H	L	L	L	H
1	0	0	0	1	L	H	H	H	H
1	0	0	1	1	L	H	H	L	H
1	0	1	0	1	L	H	L	H	H
1	0	1	1	1	L	H	L	L	H
1	1	0	0	1	L	L	H	H	H
1	1	0	1	1	L	L	H	L	H
1	1	1	0	1	L	L	L	H	H
1	1	1	1	1	L	L	L	L	H

Mas esta é a tabela do circuito 74LS20, já utilizado atrás. Trata-se daquilo que, em lógica positiva, seria um NAND. Aqui, numa situação correspondente à aplicação da metodologia associada à lógica de polaridade, este mesmo circuito é interpretado como um OR com entradas activas a L. Não se altera o símbolo no esquema para mostrar que a função desempenhada é, de facto, um OR. E, como pode ser entendido do esquema, mais do que isso, um OR directo das variáveis $P1$, $P2$, $P3$ e $P4$.

A negação presente no logigrama da Figura 3.25 entre o sinal C_L e o AND inferior é, agora, uma negação com uma entrada activa a L. Repare-se que o que estava inicialmente à entrada do AND era $\overline{C}$. Não havendo alteração na entrada do AND, o sinal que deveria lá estar seria $\overline{C}\_H$. Mas repare-se que o sinal $\overline{C}\_H$ é o mesmo que C_L, que é o sinal que está presente. A Tabela 3.9 mostra isso.

Por outro lado, pode mostrar-se igualmente que a negação com entrada activa a L presente no logigrama é um operador identidade. Chamando X_H ao sinal à saída da

TABELA 3.9: Tabela mostrando que $\overline{C}\_H = C\_L$.

C	$\overline{C}$	C_L	$\overline{C}\_H$
0	1	H	H
1	0	L	L

porta, pode ser feita a Tabela 3.10. É fácil ver na tabela que o que se pretende é um circuito que faça corresponder um L a um L e um H a um H. Sublinhe-se, de novo, que $\overline{C}\_H = C\_L$. A maneira mais óbvia de fazer isso é com uma ligação directa. O esquema evolui, portanto, para o ilustrado na Figura 3.26.

TABELA 3.10: Efeito da negação com entrada activa a L.

C_L	Entrada da porta	Saída da porta	X_H
L	1	0	L
H	0	1	H

A escolha dos restantes circuitos poderia ser feita de maneira mais ou menos directa. O OR final seria implementado por uma das portas incluídas num circuito de portas OR de duas entradas, tal como o AND inferior poderia ser implementado usando um circuito de portas AND de duas entradas. O caso do AND superior seria mais complexo, uma vez que se trataria de um AND com uma entrada activa a H e a outra activa a L, o que não existe nas séries 74nn. Mas já se viu que é possível diminuir esta diversidade, implementando toda esta zona com portas de apenas um integrado. No entanto, tal como anteriormente, vai manter-se a simbologia existente. Para tratar esta situação começa-se por acrescentar um inversor na linha que liga a entrada C_L à entrada do AND. Seguidamente transformam-se os sinais que ligam a saída das portas AND às entradas do OR em sinais activos a L. O inversor será implementado com uma porta que sobra do circuito que vai implementar os ANDs e o OR. Obtém-se, assim, o esquema da Figura 3.26.

Em termos finais, sublinhe-se que se chegou, naturalmente, ao mesmo circuito físico que se tinha obtido anteriormente, mas com um esquema muito melhor documentado em que toda a lógica inicial do problema está explicitamente representada, uma vez que os símbolos das diversas funções não se alteraram pela passagem a portas pertencentes a circuitos integrados reais. A única aparente excepção é a da falta de negação entre a entrada C e o AND inferior e o aparecimento de um inversor entre a mesma entrada e o AND superior. Mas essa excepção é só aparente. Repare-se que, por exemplo, quando

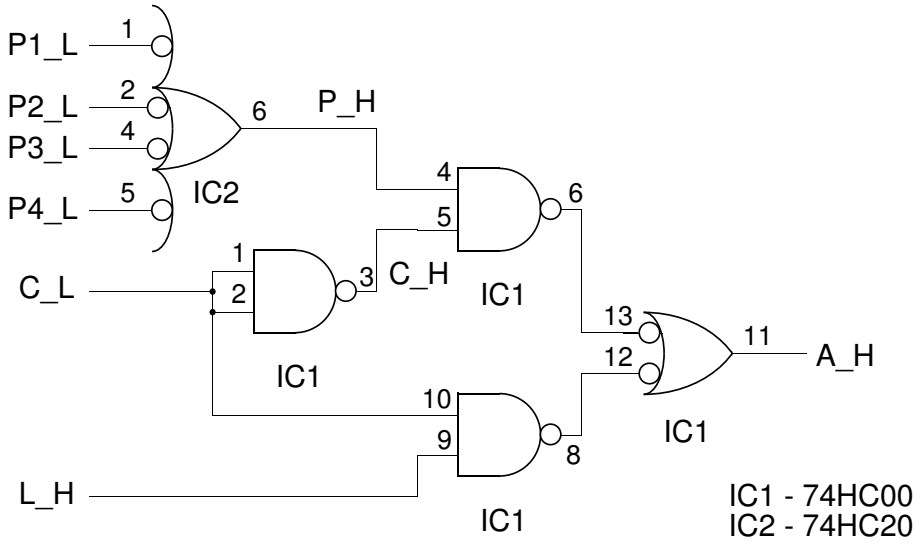


FIGURA 3.26: Esquema eléctrico do circuito que implementa a função A.

a variável C está activa, o sinal que a liga ao terminal 10 do integrado IC1 está a L. Mas nessa entrada do integrado L é interpretado como uma entrada inactiva, isto é, a negação da variável C. De facto, a variável C está representada em lógica negativa, enquanto que a entrada da porta está representada em lógica positiva, o que implica a existência de uma inversão lógica entre a variável e a entrada da porta. A negação está, portanto, lá. De igual modo, quando C está activo, as entradas 1 e 2 do integrado IC1 estão a L. Essa porta irá apresentar um H na sua saída, que será interpretado como uma entrada activa no terminal 5. Então não há negação entre C e a entrada dessa porta AND. Repare-se, assim, que a porta com entradas 1 e 2 e saída em 3 deve ser interpretada, não como uma negação, mas sim como um inversor do nível de actividade do sinal. A sua saída deverá chamar-se, assim, C_H, aparecendo a variável C suportada por um sinal activo a H numa entrada que é também activa a H.

Esta metodologia, que pode parecer um pouco confusa inicialmente, possibilita o desenho de esquemas muito mais claros, uma vez que incluem toda a informação eléctrica real, mas mantêm a informação lógica de base. Por outro lado, é eliminada toda a ambiguidade acerca da relação entre o estado lógico das variáveis e das entradas e saídas das diversas portas.

Para tornar o esquema mais legível há algo mais que pode ser feito. As variáveis utilizadas não têm um significado particularmente claro. Pi, P, C, L e A são designações demasiado abstractas. Nada impede que os nomes das variáveis explicitem melhor o seu significado. Por exemplo, em vez de L pode ser usada a designação Luz_Acesa, o que tira toda a ambiguidade de interpretação. Se se proceder desse modo, pode atingir-se, por exemplo, o esquema da Figura 3.27.

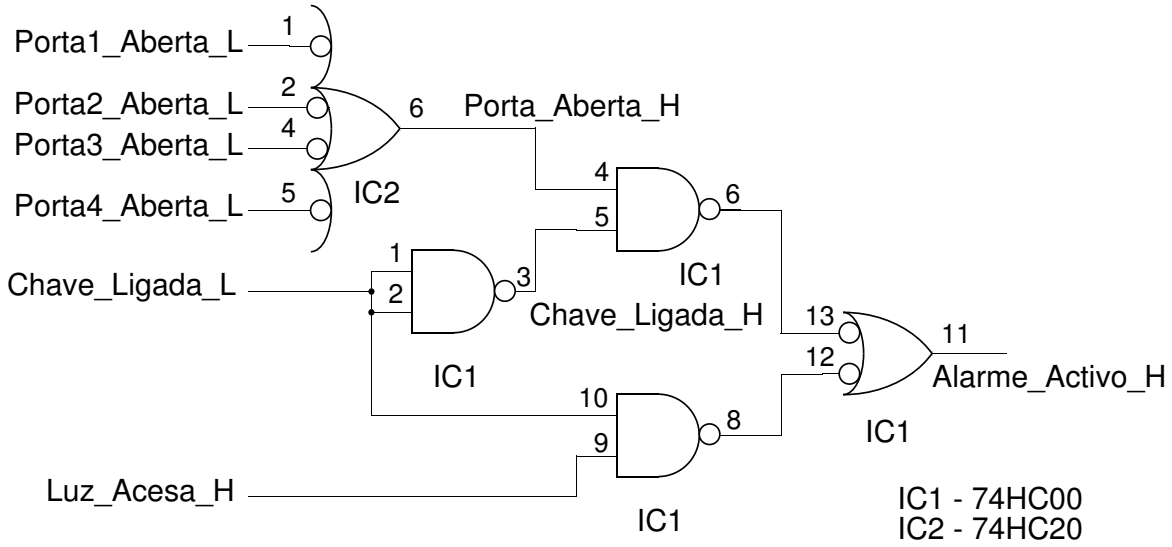


FIGURA 3.27: Esquema eléctrico final do circuito que implementa a função A.

3.3 DETECÇÃO DE FALHAS

Um circuito digital, tal como qualquer outro, pode exibir um funcionamento diferente do especificado, isto é, pode ter um funcionamento que não corresponde ao pretendido. Isso pode ocorrer por várias causas.

— A especificação do circuito pode não corresponder ao que é pretendido. Esta situação é frequente e resulta da ambiguidade da linguagem corrente usada para descrever o comportamento pretendido de um sistema. Quando se passa da descrição de um sistema para a sua especificação formal, as ambiguidades são forçosamente resolvidas e, por vezes, não da maneira desejada por quem necessita do sistema. É, por isso, muito importante, nesta fase, que todos os envolvidos na concepção e projecto, bem como a entidade a quem o projecto se destina, verifiquem exaustivamente se o sistema especificado tem a funcionalidade pretendida.

— O projecto do circuito pode não corresponder à especificação. Esta situação está naturalmente associada a erros de projecto ou a decisões de compromisso relacionadas com o custo associado a cumprir integralmente a especificação. Esta segunda hipótese corresponde, na prática, a uma revisão da especificação inicial conduzida por motivos de ordem económica ou de praticabilidade. No caso de erro de projecto, haverá que identificá-lo e corrigi-lo.

— A implementação pode não corresponder ao projecto. Neste caso, verifica-se que houve troca nos componentes utilizados ou erro na execução das ligações entre eles. A solução é, como é óbvio, identificar a fonte do problema e eliminá-la.

— Finalmente, pode ocorrer um mau funcionamento num circuito bem especificado, projectado e construído, por *falha* de um componente ou da sua interligação ocorrida já em funcionamento. Pode verificar-se uma falha de um ou mais componentes por funcionamento fora das especificações, por excesso de tempo de vida ou por deficiente concepção ou fabrico. Essa falha provoca um *erro* no comportamento do sistema. Esse erro manifesta-se numa *avaria*, isto é, na ocorrência de um não cumprimento das especificações do sistema. A avaria verifica-se. Essa verificação leva a que se procure e detecte o erro. Encontrado o erro, localiza-se a falha e corrige-se.

A avaria é, portanto, verificada ao nível do sistema e corresponde à sua incapacidade de desempenhar a sua função, de acordo com o especificado, por existência de erros nos seus elementos ou no seu ambiente (incluindo os sistemas com que faz interface).

As falhas podem ter, portanto, várias origens:

— Erros de projecto, erros na especificação ou na implementação.

— Erros devidos a problemas de fabrico dos elementos.

— Deterioração dos elementos, quer por danificação, quer por fadiga.

— Perturbações de causa externa, como condições ambientais agressivas, interferência electromagnética, radiações ionizantes, valores de tensão ou corrente eléctrica inadequados nas entradas, ou utilização errada.

Os erros no comportamento do elemento (o que se observa) são causados por falhas que se manifestam. No entanto, há falhas que não provocam erros, seja porque não alteram as características pertinentes do elemento, seja por não colocarem as suas especificações fora dos limites aceitáveis. Uma falha é classificada de *latente*, se ainda não provocou um erro e, por conseguinte, uma avaria.

As falhas podem, quanto à sua ocorrência, ser classificadas em três classes:

— *Falhas transientes*: ocorrem, em geral, por influência externa e ocorrem uma vez sem se repetir. São muito difíceis de tratar, uma vez que não persistem, em geral, o tempo suficiente para serem estudadas.

— *Falhas intermitentes*: o sistema oscila entre o funcionamento correcto e avariado sem controlo do utilizador. Este tipo de falhas resulta do funcionamento de um elemento ou de uma ligação de maneira marginal ou instável. Trata-se de um tipo de falhas difícil de detectar, uma vez que nem sempre se encontra presente.

— *Falhas permanentes*: a falha mantém-se ao longo do tempo e é, portanto, possível elaborar e aplicar estratégias para a sua detecção.

3.3.1 Tipos de Falhas em Circuitos Digitais

Há vários tipos de falhas em circuitos digitais que devem ser conhecidas para possibilitar a elaboração de estratégias para a sua detecção.

O primeiro tipo de falhas é constituído pelas *falhas tecnológicas* nos integrados, também referidas, por vezes, como *falhas analógicas*. Podem provocar o funcionamento incorrecto do circuito, do ponto de vista de tensões, potências de saída e/ou tempos de comutação, ainda que o circuito permaneça funcionalmente correcto no sentido estrito. Algumas falhas externas (radiação, campos electromagnéticos fora das especificações) podem ter efeitos semelhantes.

Manifestações deste tipo de falhas podem ser, por exemplo:

— a incapacidade de uma saída de, num dos níveis, atacar convenientemente as entradas a que está ligada;

— a ocorrência de atrasos demasiado elevados com consequências sobre o comportamento funcional do módulo onde o circuito está inserido;

— a ocorrência de níveis de tensão ilegais na saída ou a incorrecta interpretação nas entradas de níveis dentro dos limites;

— a ocorrência de excesso de consumo nos integrados, com aquecimento excessivo destes, e eventual deterioração global dos níveis de tensão de alimentação.

Este tipo de falhas pode ser difícil de detectar em alguns casos ou de classificar como falhas analógicas noutros.

Um segundo tipo de falhas corresponde às *falhas lógicas* dentro do integrado, isto é, a que as portas do integrado não cumpram a sua tabela de verdade. Ao nível em que o circuito é já interpretado como um dispositivo lógico, as falhas reflectem-se num mau comportamento funcional. Estas falhas podem ser caracterizadas segundo vários modelos de falhas, que resumem, em comportamentos verificáveis semelhantes, falhas que podem ser, na sua essência, diferentes. Num circuito electrónico, um *nó* é um ponto de interligação entre vários componentes.

— Nó preso a um valor (em inglês, designado por *stuck at*): neste caso, tudo se passa como se um ponto do circuito (não necessariamente de acesso exterior) estivesse bloqueado a H ou L.

— Ligação entre dois sinais (em inglês, designada por *bridging*): neste caso, tudo se passa como se dois sinais estivessem em curto-circuito ou, em casos menos definidos, como se um influenciasse indevidamente outro.

— Nó flutuante ou desligado: Neste caso, tudo se passa como se um determinado ponto de um circuito estivesse desligado do sinal que o deveria activar. Pode ser lida uma tensão, no leque intermédio de tensões entre o L e o H, mas a tensão não está fixa e pode variar por influência exterior.

Há, por fim, que considerar as falhas lógicas externas ao integrado, isto é, que são causadas pelas interligações entre integrados. Causas típicas das falhas lógicas exteriores ao integrado são as seguintes:

— Curto-circuito entre sinais. Este tipo de falhas é, no contexto de falha externa, semelhante aos dois primeiros tipos de falhas lógicas internas ao integrado. Podem ter várias causas, como por exemplo:

  — por excesso de solda, que faz pontes entre terminais de um integrado;

  — por um resíduo condutor ficar preso entre dois condutores;

  — por dois terminais de componentes terem sido dobrados e entrado em contacto;

  — por defeito ou degradação do circuito impresso (por aquecimento, por exemplo).

  Os curto-circuitos entre sinais podem verificar-se entre sinais lógicos ou entre um sinal lógico e a alimentação ou a massa. No caso de montagens experimentais em *réguas de terminais* (em inglês, *breadboards*), as falhas deste tipo resultam de erros de ligação dos fios.

— Falta de ligação num nó, o que corresponde ao terceiro tipo de falha interna em circuitos integrados. As causas típicas são:

  — por quebra de traços (as interligações) no circuito impresso;

  — por soldaduras mal feitas, ou com oxidação interna;

  — por quebra de terminais.

  A falta de ligação pode ocorrer num nó de sinal com ausência de sinal em algumas entradas de circuitos ou com a alimentação, caso em que os circuitos não são alimentados. Mais uma vez, este tipo de falhas pode manifestar-se em montagens laboratoriais por erros (ou ausência) de colocação de fios.

3.3.2 Ponta de Prova Lógica

Existem vários instrumentos de laboratório úteis para a localização de falhas em circuitos. Os mais comuns são: a *ponta de prova lógica* (em inglês, *logic probe*), que permite observar o nível presente num determinado nó de um circuito; o *pulsador lógico* (em inglês, *logic pulser*), que permite impor instantaneamente um nível determinado num nó de um circuito; o *osciloscópio*, que permite observar a forma do sinal eléctrico ao longo do tempo em um ou mais nós do circuito (tipicamente quatro a oito, no máximo); e o *analisador lógico* (em inglês, *logic state analyzer* ou *data analyzer*), que permite, já a um nível de abstracção lógico, observar a evolução de um conjunto de nós do circuito ao longo do tempo (tipicamente 32 a 64 no máximo). Ao nível a que este livro se coloca, ir-se-á privilegiar a ponta de prova lógica.

O êxito da localização de falhas num circuito digital simples depende mais dos procedimentos a realizar e da experiência de quem realiza o trabalho do que da sofisticação e complexidade do equipamento. A ponta de prova lógica é um pequeno (e barato) instrumento de desempanagem de circuitos digitais que, para circuitos de baixa complexidade, é um auxiliar precioso. Consiste num pequeno dispositivo que pode ser facilmente suportado na mão com uma pequena ponta metálica e alguns *díodos emissores de luz*, tipicamente designados por LED (do inglês, *Ligth Emitting Diode*) de sinalização. A Figura 3.28 ilustra um possível formato. A alimentação é garantida pela mesma fonte que alimenta o circuito em teste.

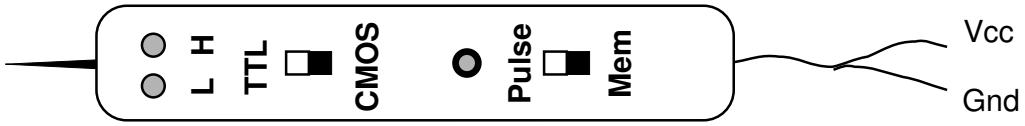


Figura 3.28: Possível formato para uma ponta de prova lógica.

Os LED que equipam a ponta permitem conhecer o estado lógico de um terminal onde a ponta seja encostada.

Uma ponta de prova tem habitualmente dois comandos, através de interruptores, e três LED.

— LED H: Este LED, tipicamente vermelho, ilumina-se apenas quando a ponta está encostada a um terminal com nível válido *H* para a família lógica seleccionada.

— LED L: Este LED, tipicamente verde, ilumina-se apenas quando a ponta está encostada a um terminal com nível válido *L* para a família lógica seleccionada.

— LED *Pulse*: LED, tipicamente amarelo, destinado a assinalar a existência de transições num terminal a que a ponta de prova esteja encostada.

— Selecção TTL/CMOS: Este interruptor permite seleccionar qual das famílias lógicas está em observação. É importante esta indicação para que a ponta possa seleccionar quais os níveis de tensão limite do H e do L que deve adoptar.

— Selecção memória/impulso: Um impulso muito curto pode não ser visto nos LED por ser demasiado reduzido para ser detectado pelo olho durante o tempo em que estão acesos, ou por ser demasiado curto para provocar reacção do próprio LED. Para evidenciar esses impulsos, a ponta detecta flancos de transição de sinais entre os valores H e L e vice-versa. A forma de indicar isso depende da posição deste selector.

  — Impulso: Um impulso de duração visível é mostrado no LED auxiliar.

  — Memória: A transição é memorizada, ficando o LED aceso até o selector deixar de estar na posição Memória.

  Em qualquer dos casos, não há indicação do sentido da transição.

Se nenhum dos LED se iluminar, isso significa uma de duas coisas: ou se está perante um nível eléctrico que não corresponde a qualquer nível lógico (zona intermédia), ou não existe tensão aplicada ao contacto, por a ponta estar no ar sem contacto com nenhum sinal ou por estar a detectar um sinal de alta impedância (ver Secção 3.1.6).

3.3.3 Estratégias para Detecção de Falhas

A detecção de falhas em circuitos digitais é um processo complexo que exige uma grande capacidade de organização e, em geral, um conhecimento razoável do circuito e da sua funcionalidade. Quando surge uma avaria, a primeira coisa a fazer é definir claramente qual é a avaria e, tanto quanto possível, em que situações surge. A colocação dos controlos, os valores, ou sequências, nas entradas e o comportamento que é qualificado de defeituoso devem ser especificados com a maior clareza possível. É fundamental estar na posse do esquema do circuito e conhecer o material de teste e o seu funcionamento. Uma observação visual, por vezes, ajuda. Um componente queimado ou um suporte de integrados na placa de circuito impresso (que devia ter um componente) vazio, por exemplo, podem ser uma boa pista.

Para circuitos de grande complexidade existem técnicas específicas, e, inclusivamente, os circuitos podem ser já projectados de modo a detectarem eles próprios uma falha

ou a auxiliarem na sua detecção. Em circuitos simples, como os que vão ser naturalmente utilizados num laboratório ao nível deste livro, a detecção é feita manualmente, utilizando instrumentos simples, como uma ponta de prova lógica.

Considere-se um sistema digital qualquer. Num circuito digital, um nó será, por exemplo, a ligação entre a saída de uma porta e as entradas de portas a ela ligadas. Um nó tem uma certa extensão física, uma vez que é materializado por ligações entre componentes. Em toda a sua extensão física o nó assume o mesmo nível de tensão e, implicitamente, lógico.

Verificada uma avaria, o caminho necessário para a corrigir passa obrigatoriamente por encontrar o nó que é fonte da falha. Esse é o nó realmente em falha e que provoca erros provavelmente em vários outros nós.

O teste para detecção de uma falha num circuito pode iniciar-se quer nas entradas, quer nas saídas, quer no miolo do circuito. Tudo depende da experiência de quem está a operar o circuito. Para circuitos com alguma complexidade, porém, o mais aconselhável é iniciar a busca pelas saídas recuando sucessivamente em direcção às entradas até à detecção da falha. Há duas razões para isso: por um lado, as saídas são o local em que o erro habitualmente se manifesta e, por outro, os circuitos digitais têm, muitas vezes, mais entradas que saídas. Começando pelas saídas e detectando quais os nós responsáveis por uma saída errada, elimina-se a necessidade de pesquisar numa área grande do circuito.

Na continuação, serão apresentadas algumas estratégias para detecção de falhas. Como é evidente, trata-se de uma pequena introdução a um tema de grande complexidade, que é aqui grandemente simplificado e colocado a um nível instrumentalmente útil para os leitores deste livro. De igual modo, restringe-se, aqui, o assunto à detecção de falhas em circuitos combinatórios.

A procura de falhas inicia-se pela verificação das alimentações. Um circuito, para funcionar correctamente, tem de ter o valor da tensão de alimentação dentro dos valores especificados. Mas a verificação da alimentação tem mais finalidades, nomeadamente detectar possíveis curto-circuitos e/ou circuitos abertos. É, assim, natural iniciar a procura de possíveis falhas verificando se o valor da tensão de alimentação está dentro dos valores especificados. Se a tensão estiver abaixo do valor especificado (nomeadamente se for de 0V), e assumindo que a fonte de alimentação está activa e ligada ao circuito, deverá existir um curto-circuito neste, ou a ligação à fonte de alimentação está cortada. Assumindo que a tensão de alimentação é correcta, é necessário verificar, nos terminais de alimentação e massa dos integrados, se cada um deles está devidamente alimentado, uma vez que pode haver uma quebra na ligação de alimentação apenas a algum ou alguns integrados.

A localização da falha propriamente dita exige uma boa disciplina de trabalho. Como se referiu atrás, será apenas analisado o caso dos circuitos combinatórios.

Neste tipo de análise, a técnica é a seguinte:

1. Encontra-se um nó com um erro.
2. Verifica-se qual o elemento de circuito [1] (uma porta, em geral) de que esse nó é saída.
3. Pesquisam-se os nós de entrada desse elemento. Se estiverem todos correctos, a falha está no elemento.
4. Se algum(ns) estiver(em) errados, assumir esse(s) nó(s) como nó(s) de partida e continuar a pesquisa voltando ao ponto 1.

Procede-se seguidamente à identificação da falha. Assumindo que possíveis erros relacionados com a alimentação dos integrados estão excluídos (ou resolvidos), encontrado um nó com um comportamento errado, esse comportamento é um dos seguintes:

— nível errado;

— nó flutuante;

— nó preso a L;

— nó preso a H.

Note-se que observar o nível errado significa que um nível errado está presente, mas a saída do circuito que o produz assume outros níveis noutras circunstâncias. Uma saída flutuante significa que o circuito apresenta permanentemente a sua saída flutuante. Um nó preso a um determinado nível significa, obviamente, que não é possível fazer a saída do circuito abandonar esse nível.

Se existir um nível errado, isso pode resultar do seguinte: mau funcionamento do circuito, entrada do circuito desligada do ponto que se está a observar (por quebra da pista de circuito impresso ou do fio) ou, por vezes, curto-circuito com outro nó.

A existência de um nó flutuante pode resultar de um mau funcionamento do componente ou de a saída do circuito estar fisicamente desligada do ponto que se está a observar, por razões semelhantes às do caso anterior.

[1] Em alguns casos, como por exemplo com saídas de três estados, pode haver mais de um elemento.

Um nó preso a L é sintoma de uma das seguintes causas: mau funcionamento do componente, curto-circuito externo à massa do circuito (por erro de ligação, degradação do circuito impresso ou presença de um elemento estranho que provoque o curto-circuito) e, por vezes, curto-circuito com outra saída a L.

Finalmente, um nó preso a H, para além da hipótese de mau funcionamento do componente, pode ser devido a um curto-circuito externo à alimentação do circuito.

Para simplificar a identificação da falha à saída ou à entrada de um elemento, deve verificar-se o estado dessa saída ou entrada no terminal do circuito e não algures na extensão física do nó, uma vez que isso permite isolar a hipótese de quebra de ligação das restantes.

— **Nível errado**

Sempre que possível, deve ser desligada a saída do componente do resto do circuito, para verificar se o comportamento do componente tem origem nele próprio ou num curto-circuito da sua saída com outro sinal. No primeiro caso, haverá que substituir o componente. No segundo caso, há que corrigir a interligação.

— **Nó flutuante**

Só se assume esta situação quando o nó está permanentemente flutuante. Um nó pode estar flutuante, por vezes, de acordo com a sua funcionalidade normal, se as saídas que a ele estão ligadas são saídas de três estados. A detecção de um nó flutuante significa que não há ligação eléctrica entre a saída interna do integrado e o nó.

A distinção entre as duas causas possíveis é simples. Se a ponta de prova lógica encostada directamente ao terminal do integrado continuar a detectar um nó flutuante, então o problema é interno e há que substituir o integrado. Se o terminal apresentar um valor diferente, então há uma ligação exterior desligada, que há que reparar. É evidente que, passeando a ponta de prova por todo o nó, é possível localizar exactamente o local da falha.

Em certas circunstâncias, podem ainda ocorrer tensões intermédias, que sugerem um nó flutuante, mas resultam de curtos-circuitos entre saídas diferentes.

— **Nó preso a L**

Embora seja possível com equipamento adicional identificar exactamente esta situação, ela pode ser tratada como o caso do nível errado.

— **Nó preso a H**

Tal como no caso anterior, ao nível a que se está a analisar o problema, pode este caso ser tratado como o caso de nível errado.

3.3.4 Exemplos de Localização e Detecção de Falhas

Considere-se, como exemplo, o circuito da Figura 3.29, onde se ilustrou uma situação caracterizada pelo funcionamento correcto do circuito com um conjunto de valores para os sinais de entrada e os sinais que, em consequência, se instalaram nos diversos nós internos e na saída.

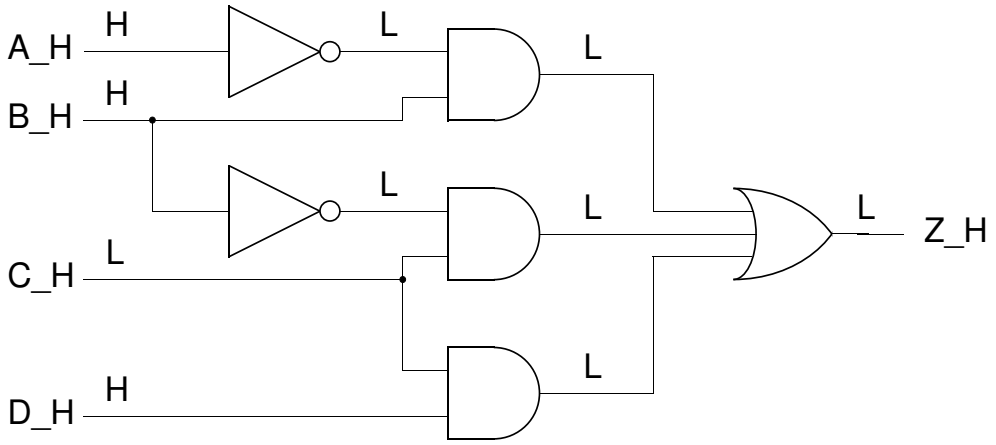


Figura 3.29: Circuito tomado como exemplo para ilustrar a identificação de uma falha.

Assuma-se, agora, a existência de uma falha na porta AND superior, cuja entrada fica curto-circuitada com a alimentação. A situação do circuito passa a ser a ilustrada na Figura 3.30.

Esta situação é identificada como um erro. É fácil verificar que a saída do último circuito, um OR, está errada, uma vez que deveria estar em L e está em H. Neste caso, é possível identificar, por análise das várias entradas da porta OR, por exemplo com uma ponta de prova lógica, que o problema provém da entrada superior.

Daí, analisando o AND origem desse nó, pode ser verificado que, aparentemente, funciona correctamente. Recuando de novo no circuito, aparece, agora, uma negação que aparenta um funcionamento incorrecto, uma vez que tem níveis H, quer na entrada, quer na saída. A conclusão imediata é a de que a porta NOT tem uma falha, conclusão errada, neste caso. Desligando a saída da porta NOT do circuito, pode ser verificado que está a funcionar correctamente. Nessas circunstâncias, analisando o nível da entrada da porta AND com a ponta de prova, pode concluir-se que, em vez de estar «no ar», a entrada superior da porta tem o nível H, que é indício de que ou a porta tem uma falha ou que há um curto-circuito externo. O diagnóstico diferencial pode ser feito, se possível, removendo todas as ligações à porta que possam estar no circuito e verificando se a porta mantém o comportamento. Em situação real, num circuito impresso, isto é, em

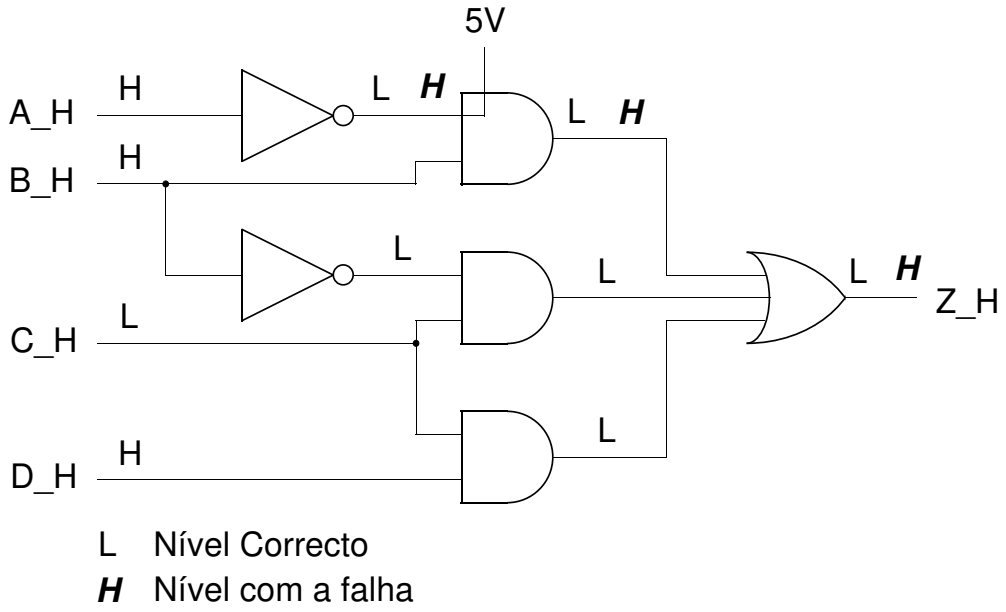


FIGURA 3.30: Circuito tomado como exemplo com uma falha.

geral, de difícil execução. A observação visual do circuito pode ajudar a detectar uma ligação incorrecta ou a presença de algum resíduo condutor.

Um segundo exemplo com o mesmo circuito é também interessante. Considere-se a situação ilustrada na Figura 3.31. Há agora um curto-circuito exterior entre a entrada inferior da porta AND e a terra do circuito. Na figura está ilustrada uma situação em que o circuito está a funcionar correctamente e a situação resultante da falha.

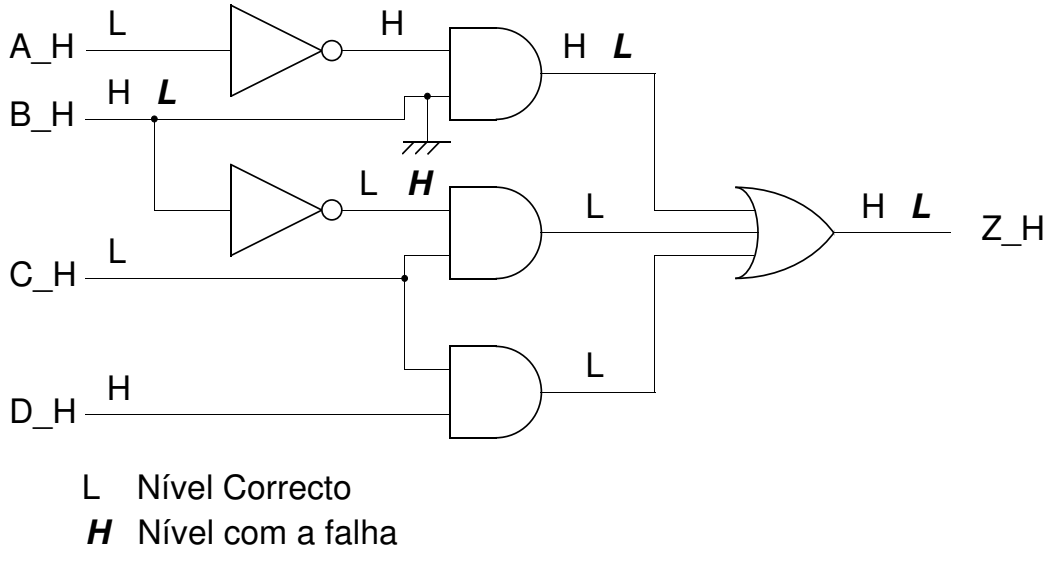


FIGURA 3.31: Circuito tomado como exemplo com uma falha.

Aqui a situação é mais difícil de analisar. Verifica-se que a porta OR está com a sua saída a L e devia tê-la a H. Isso não é o suficiente para identificar a causa do problema. De facto, qualquer uma das entradas do OR podia ser a responsável pela situação. Se fossem conhecidos os valores que deviam estar nos nós de entrada, tudo estaria resolvido, mas, em geral, isso não acontece. Aqui haverá, portanto, que analisar mais pormenorizadamente todo o circuito.

No entanto, se se obtiver (ou preexistir) uma representação mais abstracta do circuito, pode realizar-se a detecção sem ter o trabalho de proceder à análise exaustiva do circuito para encontrar a entrada do OR com comportamento errado. Uma hipótese interessante (quando possível) passa por construir o mapa de Karnaugh do circuito. Neste caso, as três portas AND correspondem aos seguintes produtos: $\overline{A}\,B$, $\overline{B}\,C$ e $C\,D$. Isso corresponde ao mapa da Figura 3.32.

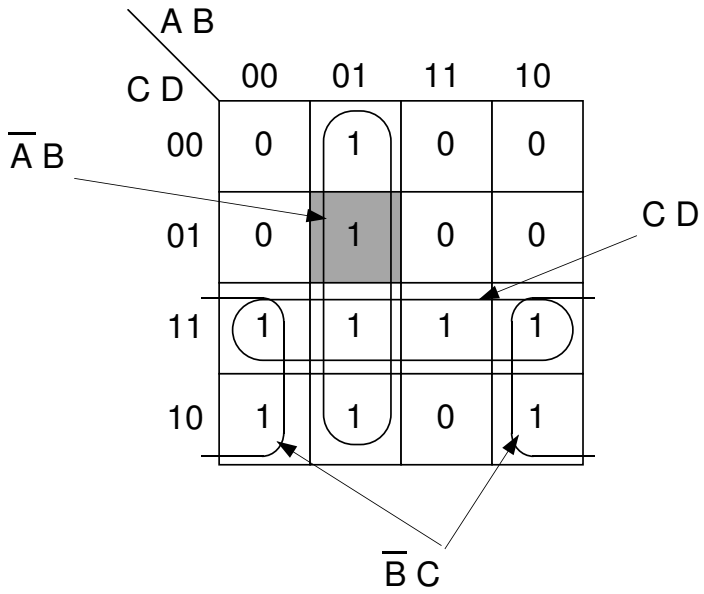


FIGURA 3.32: Mapa de Karnaugh do circuito tomado como exemplo com uma falha.

No mapa é agora fácil de analisar a situação. Na configuração onde se verifica o erro, temos $A = 0$, $B = 1$, $C = 0$ e $D = 1$. No mapa, isso corresponde ao quadrado sombreado. O quadrado sombreado está incluído no grupo correspondente ao produto $\overline{A}\,B$. Esse produto é implementado pelo AND superior do esquema, pelo que é ele o responsável por impor um H na saída, o que não está a acontecer. A partir desta identificação da fonte do problema, a identificação da falha prossegue como no caso anterior.

Por vezes, estes procedimentos não são suficientes, e é necessário ter capacidade de identificar situações mais complexas que são de mais difícil análise. Aí, de facto, a experiência é fundamental.

3.4 CARACTERÍSTICAS TEMPORAIS

Na Secção 3.1.4 foi já apontado que o funcionamento dos dispositivos físicos utilizados em circuitos digitais não é instantâneo, causando, portanto, a ocorrência de atrasos. A Figura 3.33 repete a figura da Secção 3.1.4.

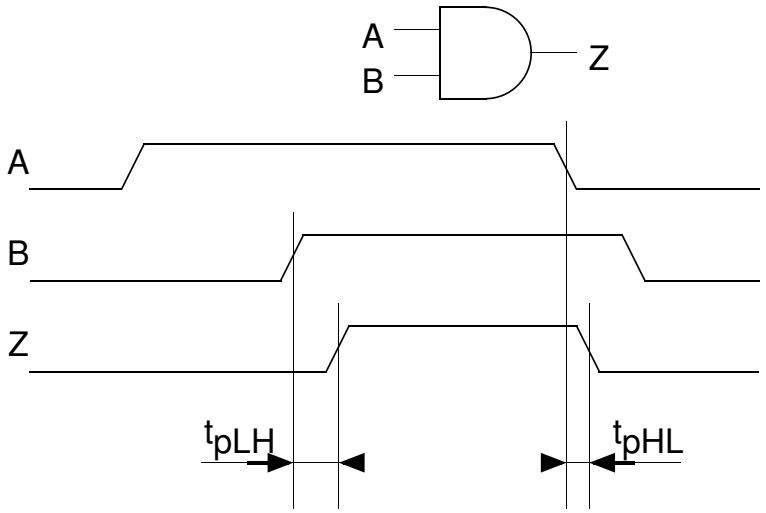


FIGURA 3.33: Exemplo de atrasos numa porta AND.

Os tempos de atraso em consideração são especificados para cada integrado nos catálogos. Por exemplo, um 74LS08 tem as especificações temporais descritas na Tabela 3.11 no catálogo da Texas Instruments. Recorde-se que, por exemplo, t_{pLH} corresponde ao atraso entre uma alteração na entrada de uma porta e a ocorrência de uma transição de L para H na sua saída.

TABELA 3.11: Especificações temporais do 74LS08.

Parâmetro	Valor mínimo	Valor típico	Valor máximo
t_{pLH}		8ns	15ns
t_{pHL}		10ns	20ns

A Tabela 3.11 sugere alguns comentários. O valor dos atrasos é, aqui, medido em nanossegundos (ns), com 1 ns = 10^{-9} s. Os valores dos atrasos especificados na tabela parecem baixos, revelando uma elevada velocidade de funcionamento, mas o atraso de uma porta num microprocessador actual é muito mais baixo, tipicamente abaixo de 1 ns. Por razões tecnológicas, são muito raros os fabricantes que especificam os valores mínimos do atraso. O valor típico dos atrasos tem apenas um valor indicativo. O fabricante não o garante, e o utilizador não pode confiar nele. Só se pode confiar no valor

máximo, também chamado de pior caso (em inglês, *worst-case*). Repare-se, por fim, que, como já se referiu, os valores de atraso dependem do sentido em que se dá a comutação. Em alguns casos, no que se refere ao pior caso, há circuitos em que os dois tempos são iguais, como acontece, por exemplo, no 74LS00. Mas o normal é os dois tempos serem diferentes.

Com conhecimento dos valores dos atrasos das portas é possível determinar o tempo de atraso de circuitos combinatórios mais complexos. No que se segue serão apenas considerados os tempos no pior caso, pois é este tempo que limitará a velocidade máxima de funcionamento garantida do circuito. É muito provável que o circuito funcione com maior rapidez, mas a única velocidade que se pode garantir é aquela.

3.4.1 Análise de Atrasos em Circuitos

Considere-se o circuito da Figura 3.34.

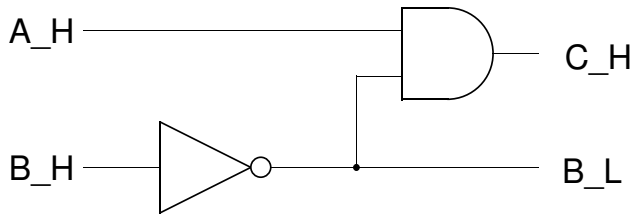


Figura 3.34: Exemplo de circuito combinatório simples.

Num circuito com várias entradas e várias saídas, como é o caso, podem ser especificados dois tipos de atrasos importantes:

— o atraso entre uma determinada entrada e uma dada saída;

— o maior atraso entre uma qualquer entrada e uma qualquer saída.

A sequência de nós envolvidos no segundo tipo de atrasos recebe o nome de *caminho crítico*. Há técnicas e algoritmos sofisticados para determinar aqueles valores para circuitos complexos. No contexto deste livro far-se-á apenas uma abordagem introdutória ao assunto.

Considere-se, no circuito da Figura 3.34, o atraso entre a entrada B_H e a saída C_H. O primeiro aspecto a considerar é se a entrada B_H influencia a saída C_H. Há, de facto, um caminho entre a entrada e a saída referidas, através do nó B_L.

Repare-se, porém, que as variações em B_H só afectam C_H se a entrada A_H assumir o nível H. No caso contrário, uma vez que 0 é o elemento absorvente do produto lógico, a saída C_H estaria em L independentemente do valor de B_H. Este é o segundo aspecto a ter em conta: quais são as condições que, para um par entrada-saída em estudo, permitem que uma variação na entrada provoque uma variação na saída. O estabelecimento dessas condições no circuito tem o nome de *sensibilização do caminho*.

Sensibilizado o caminho, o terceiro aspecto a ter em conta é a decisão sobre o sentido de transição na entrada que provoca maior atraso no circuito. Na Tabela 3.11 estão listados os atrasos do circuito 74LS08. A tabela equivalente para o 74LS04 é a Tabela 3.12. Assuma-se inicialmente que A_H está no nível H e que B_H passa de L para H.

TABELA 3.12: Especificações temporais do 74LS04.

Parâmetro	Valor mínimo	Valor típico	Valor máximo
t_{pLH}		9ns	15ns
t_{pHL}		10ns	15ns

A linha B_L passa de H para L e a linha C_H passa igualmente de H para L. O tempo de atraso entre a transição na entrada e na saída é dado pela Expressão 3.8.

$$\begin{aligned} t_{pHLB\rightarrow C} &= t_{pHLNOT} + t_{pHLAND} \\ &= 15\text{ns} + 20\text{ns} \\ &= 35\text{ns} \end{aligned} \tag{3.8}$$

Inversamente, se B_H passar de H para L com o caminho sensibilizado, então B_L e C_H passam de L para H, e o tempo de atraso é calculado pela Expressão 3.9.

$$\begin{aligned} t_{pLHB\rightarrow C} &= t_{pLHNOT} + t_{pLHAND} \\ &= 15\text{ns} + 15\text{ns} \\ &= 30\text{ns} \end{aligned} \tag{3.9}$$

O atraso máximo será o máximo dos dois atrasos calculados. Neste caso, ter-se-á o valor $t_{pMAXB\rightarrow C} = 35$ ns. Este exemplo ilustra o procedimento geral para este tipo de problema.

Para determinar o atraso máximo do bloco de lógica com n entradas e m saídas, uma solução possível será testar todos os pares possíveis e escolher o maior atraso. Como é evidente, há procedimentos mais adequados, o que é particularmente importante em circuitos complexos. No circuito usado como exemplo, não vale a pena considerar os

pares com saída em B_L, uma vez que as modificações em B_L ainda vão influenciar C_H e, portanto, qualquer atraso entre uma dada entrada e B_L é inferior ao atraso entre essa mesma entrada e C_H. Do mesmo modo, qualquer atraso entre A_H e C_H é menor que outro atraso entre B_H e C_H, uma vez que o AND é comum aos dois caminhos, e o segundo tem ainda outra porta entre a entrada e a saída. No exemplo dado, portanto, o atraso que se obteve entre B_H e C_H é o pior atraso do circuito considerado e corresponde ao seu caminho crítico.

Analise-se, agora, um exemplo um pouco mais complexo tendo como base o circuito da Figura 3.35.

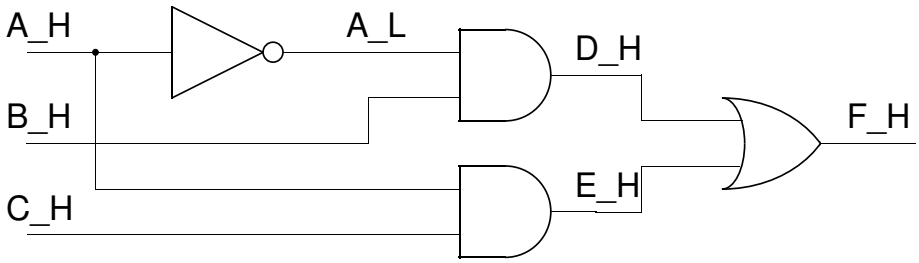


FIGURA 3.35: Exemplo de circuito combinatório para cálculo de atrasos.

Neste caso há influência de todas as variáveis de entrada na variável de saída F. Dada uma certa regularidade do circuito, é fácil perceber que o maior atraso entre uma alteração numa entrada e a alteração na saída corresponde a alterações na linha A_H. Parte-se do princípio de que a porta AND que gera o sinal E_H tem as mesmas características que a que gera D_H e que, portanto, o tempo de atraso do caminho que inclui as portas NOT e AND é sempre maior que o caminho que apenas inclui uma porta AND. Repare-se que há dois caminhos que ligam a entrada A_H à saída F_H. Quando há mais que um caminho que possibilita que um sinal de entrada influencie o valor de um sinal de saída, esses caminhos tomam o nome de *caminhos reconvergentes*.

A Tabela 3.13 descreve os atrasos do circuito 74LS32 que disponibiliza o OR.

TABELA 3.13: Especificações temporais do 74LS32.

Parâmetro	Valor mínimo	Valor típico	Valor máximo
t_{pLH}		14ns	22ns
t_{pHL}		14ns	22ns

Da análise de todos os tempos de atraso em jogo conclui-se que os tempos de atraso, quer no NOT, quer no OR, são idênticos para transições do tipo $H \rightarrow L$ e $L \rightarrow H$. É, portanto, o AND que define qual o sentido da transição nas entradas que conduz ao

pior caso de atraso. Como $t_{pHLAND} > t_{pLHAND}$, resulta que a transição na entrada que conduz ao pior caso de atraso é uma transição de L para H, uma vez que, após o NOT, o AND sente essa transição, quer na sua entrada, quer na saída, como uma transição de H para L. O atraso será, portanto, dado pela Expressão 3.10.

$$\begin{aligned} t_{pA \rightarrow FMAX} &= t_{pHLNOT} + t_{pHLAND} + t_{pHLOR} \\ &= 15\text{ns} + 20\text{ns} + 22\text{ns} \\ &= 57\text{ns} \end{aligned} \tag{3.10}$$

Quanto à determinação de quais os valores nas entradas que sensibilizam o caminho entre a entrada escolhida e a saída, ir-se-á exemplificar um método, utilizável em circuitos relativamente simples, que assenta na utilização do Mapa de Karnaugh.

O mapa de Karnaugh para a função correspondente a este circuito é o representado na Figura 3.36.

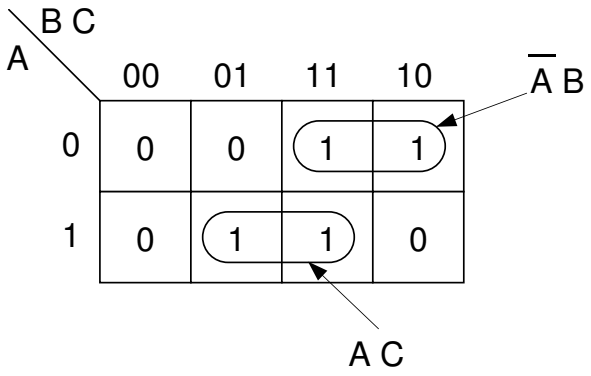


FIGURA 3.36: Mapa de Karnaugh do circuito combinatório em estudo.

Como se viu, a transição de A_H terá de ser uma transição de L para H, o que significa, na variável, uma transição de 0 para 1. E essa transição tem de corresponder, na função e, portanto, no mapa, a uma transição de 1 para 0. A única posição em que isso acontece está assinalada na Figura 3.37.

Para esta transição se dar, terão, portanto, de ser $B = 1$ e $C = 0$, valores que correspondem à coluna onde se pode dar a transição. Terá, pois, de ser B_H colocado em H e C_H colocado em L para que o caminho do pior atraso fique sensibilizado.

3.4.2 Transições Espúrias em Circuitos Combinatórios

Um outro aspecto relevante para o correcto projecto ou análise de circuitos combinatórios, entrando em consideração com os atrasos das portas, é o problema das *transições*

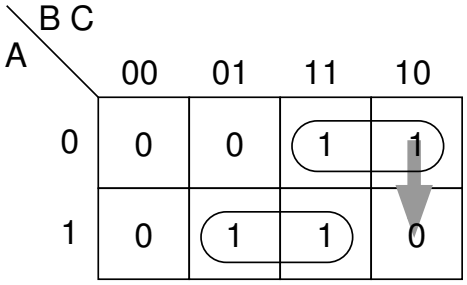


FIGURA 3.37: Mapa de Karnaugh do circuito combinatório com a transição em estudo assinalada.

espúrias, também designadas por *picos*. Retomemos o circuito anterior, para explicar qual é o problema. Considere-se que A_H, B_H e C_H estão em H. A saída assume, naturalmente, o valor H, uma vez que a porta AND inferior tem a sua saída activa. Se A_H transitar de H para L, uma vez que a saída da porta AND superior fica activa, a saída assume da mesma forma o valor H. Isto está de acordo com o que se observa no mapa de Karnaugh, uma vez que, nas duas posições do mapa correspondentes às situações descritas, a função vale 1. Observe-se, agora, um diagrama temporal pormenorizado, mostrando essa transição, incluindo as transições nos nós intermédios do circuito. Para a realização do diagrama temporal utilizam-se os tempos de atraso descritos atrás e analisa-se a evolução no tempo de cada linha. A Figura 3.38 ilustra o diagrama temporal.

Como se pode ver, vai existir um pequeno intervalo de tempo de cerca de 10 ns em que a saída, ao contrário do previsto, vai assumir o valor L. A este acontecimento é usual chamar transição espúria ou pico, pois surge como um pequeno pico num sinal, para além disso, constante. Estas situações podem ter importância ou não, consoante a utilização da linha F_H. Se a linha F_H se destinar simplesmente a assinalar o seu estado através de um LED, por exemplo, o pico é inofensivo e nem sequer é visto, dada a sua brevidade. Se a linha for actuar um dispositivo em que uma transição possa ter algum efeito, as coisas mudam completamente de figura e a situação é intolerável.

Felizmente, é fácil resolver o problema. Analise-se, então, com mais cuidado o que se passou. Inicialmente, o AND inferior estava activo, e o superior, não. Quando a linha A_H passou a L, a entrada superior do AND inferior passou imediatamente a L e o AND respondeu 20 ns depois, baixando a sua saída. No entanto, o AND superior só 15 ns depois de A_H descer é que sentiu a subida de A_L. E levou mais 15 ns a reagir. Ficou, portanto, activo cerca de 30 ns após a descida de A_H, enquanto que o outro AND já estava inactivo há 10 ns. O resultado é que durante 10 ns as entradas do OR estiveram ambas inactivas. A resposta do OR é colocar a sua saída inactiva durante um intervalo

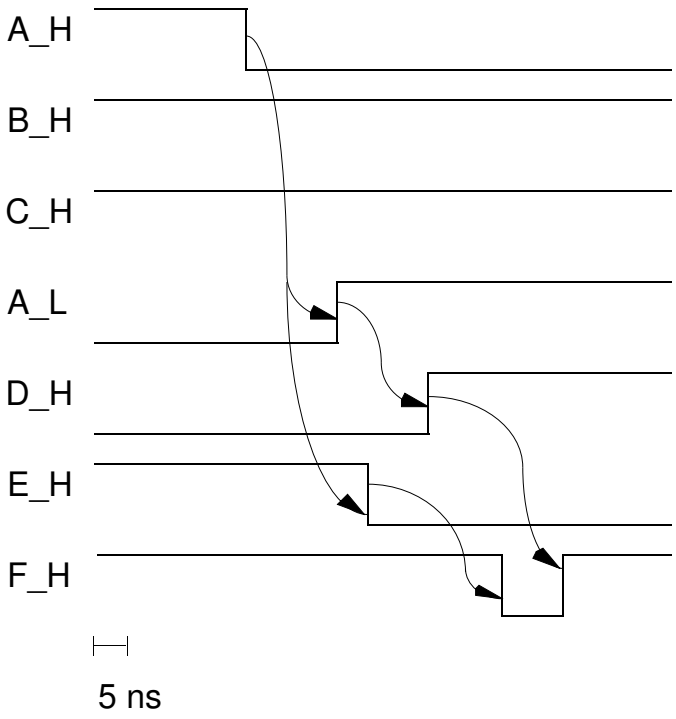


FIGURA 3.38: Diagrama temporal mostrando transições espúrias.

de 10 ns, 22 ns mais tarde [2].

Considere-se, de novo, o mapa de Karnaugh. O que se passou, nesta perspectiva, foi a ocorrência da transição assinalada na Figura 3.39.

Nessa transição, o implicante primo $\overline{A}\,B$ deixou de estar activo e passou o implicante primo $A\,C$ a ser o implicante primo activo. O que aconteceu de errado foi que nenhum implicante primo garantiu, em termos temporais, a manutenção da função a 1. Na transição do grupo de cima para o de baixo, falta uma garantia que o valor da função permaneça a 1. A solução para o problema consiste em usar um novo implicante primo redundante que, não sendo algebricamente necessário, colmate a falha detectada quando se observa o comportamento físico do circuito. O mapa fica, portanto, com mais um implicante, como se ilustra na Figura 3.40.

[2]Na realidade, a situação é bem mais complexa por dois motivos: em primeiro lugar, porque as diversas portas apresentam normalmente atrasos reais que não são os *worst-case* nem são previsíveis; por outro, porque a estrutura interna do OR, com os atrasos internos envolvidos, pode levar, por vezes, a que o pico acabe por não se manifestar. Não é, assim, possível, na maior parte dos casos, prever para todas as circunstâncias de funcionamento do circuito se haverá ou não manifestação do pico e com que duração. Mas é possível prever a possibilidade de ocorrência e agir em conformidade nos casos em que a possível ocorrência tenha consequências desastrosas.

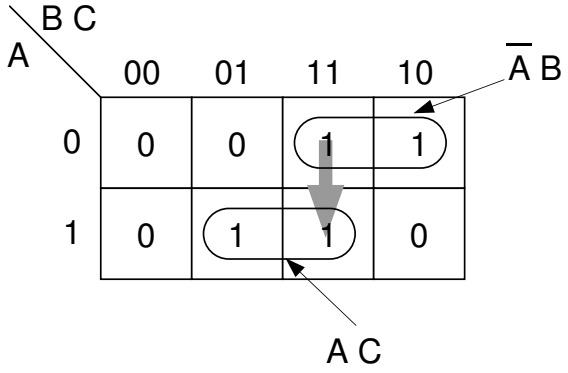


FIGURA 3.39: Mapa de Karnaugh mostrando a possível ocorrência de uma transição espúria.

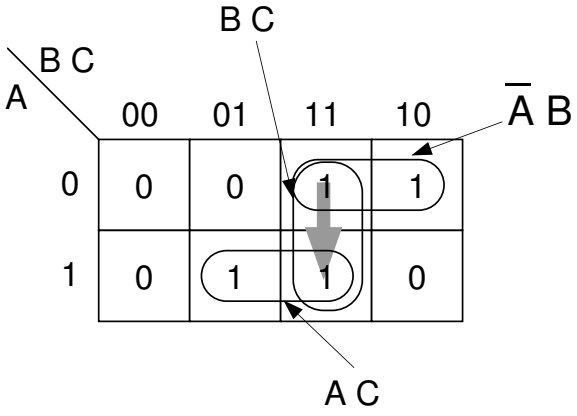


FIGURA 3.40: Mapa de Karnaugh mostrando um implicante redundante para evitar a possível ocorrência de uma transição espúria.

A função passa, assim, a ser representada pela Expressão 3.11.

$$F = \overline{A}\,B + A\,C + B\,C \tag{3.11}$$

Construindo, agora, o circuito a partir desta expressão, como se ilustra na Figura 3.41, o problema fica resolvido.

Para casos simples, este tipo de problema pode ser resolvido através deste tipo de solução em qualquer caso. Claro que, para funções de muitas variáveis, em que o mapa de Karnaugh já não é utilizável, outras ferramentas têm de ser usadas. Por outro lado, é preciso recordar que se está a partir do princípio, em toda esta análise, que apenas se altera uma variável de entrada, o que não é, de forma alguma, uma situação corrente. Por isso, na prática, para circuitos muito complexos, pode ser preferível garantir que o funcionamento do circuito que vai ser alimentado pelos sinais gerados pelo circuito

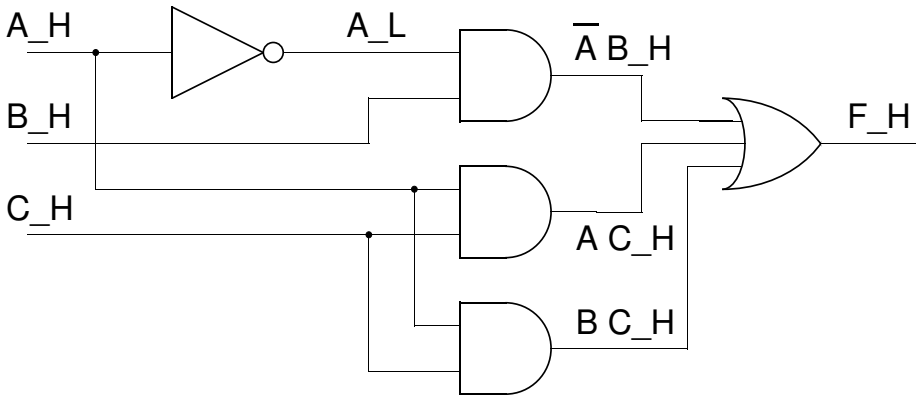


FIGURA 3.41: Alteração ao circuito combinatório da Figura 3.35 para evitar transições espúrias.

combinatório em análise não seja afectado pela existência de picos. Mais adiante, no Capítulo 6, este assunto será retomado.

3.5 REALIZAÇÃO DIRECTA

O tipo de implementação atrás descrito assenta na utilização de portas incluídas em circuitos integrados e que são usadas para replicar, em termos de circuito, a estrutura do logigrama da função. Esta replicação, que passa pelos cuidados anteriormente referidos na elaboração do esquema eléctrico, pode passar por alguma transformação por razões já apontadas, mas, no essencial, o que se implementa como circuito está muito próximo do que é descrito como logigrama e, portanto, como expressão algébrica da função.

A existência de dispositivos de memória permite uma abordagem diferente desta questão, como se verá nesta secção. Inicialmente, apresenta-se uma metodologia de implementação, que permite partir directamente da tabela da função e reduz substancialmente o número de circuitos integrados necessários à implementação e, seguidamente, mantendo-se a vantagem decorrente de uma maior integração, apresentar-se-ão outros dispositivos programáveis, que permitem, recuperando a implementação a partir da estrutura do logigrama, diminuir a complexidade dos dispositivos necessários para implementar um dado circuito.

3.5.1 Realização Usando ROM

Uma memória apenas de leitura conhecida pela sigla *ROM* (do inglês, *Read Only Memory*), é um dispositivo onde é possível gravar, na fase de fabrico, informação de uma

forma permanente. As ROM são, por vezes, consideradas circuitos sequenciais, um tipo de circuito que será estudado a partir da Secção 6.8, mas, na realidade, são circuitos combinatórios. A ROM tem uma estrutura ilustrada na Figura 3.42 para um exemplo de uma ROM de oito palavras de 4 bits. O pequeno quadrado no cruzamento de duas linhas significa que aquela ligação pode ou não estar feita, consistindo a gravação de informação na ROM na decisão de quais as ligações realmente feitas. A estrutura apresentada não corresponde, realmente, à forma como a ROM é de facto implementada, mas é suficientemente ilustrativa dos seus princípios de funcionamento.

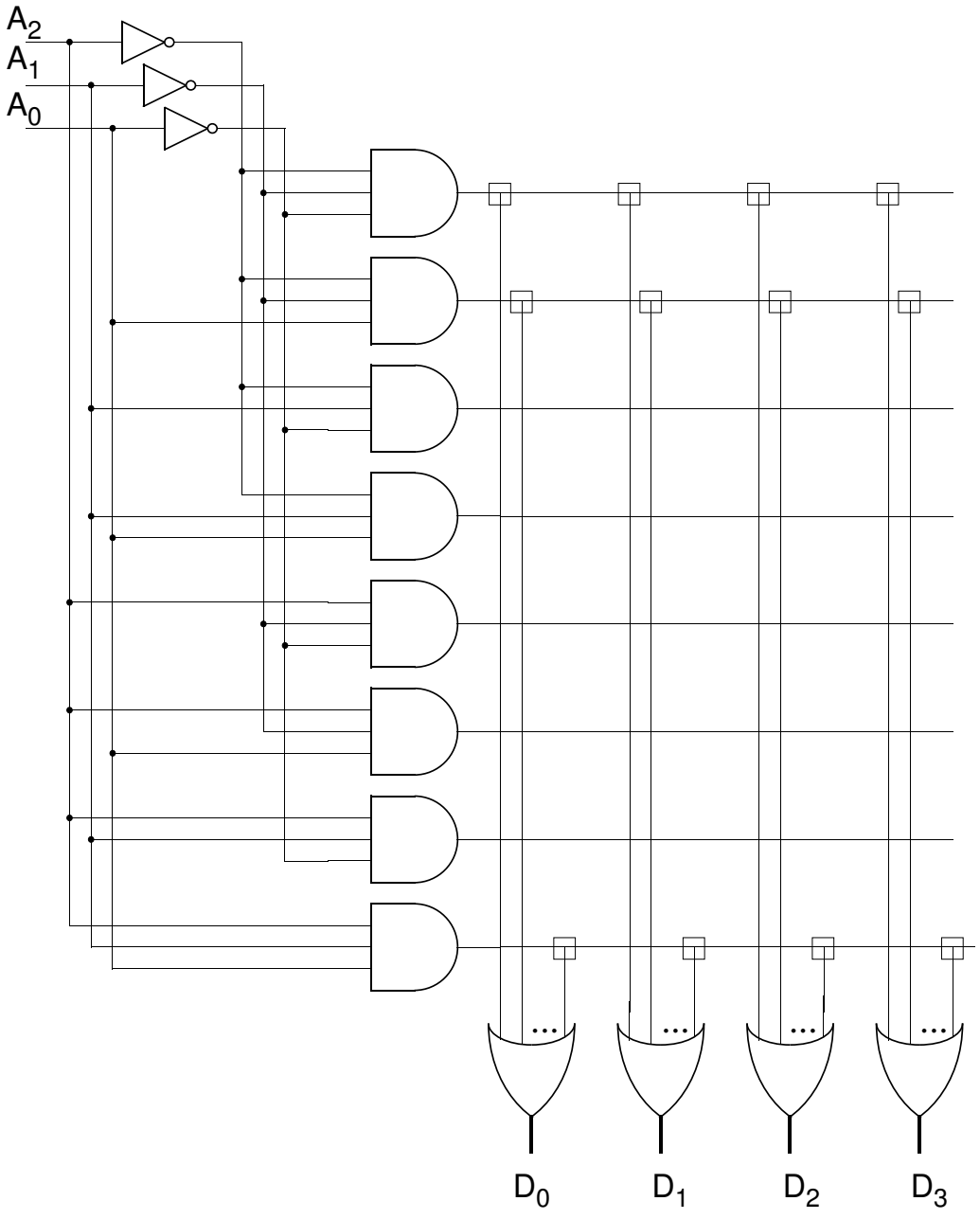


FIGURA 3.42: Estrutura funcional interna de uma ROM.

Na realidade, a estrutura da ROM é ligeiramente diferente, sendo os OR efectivamente realizados por uma ligação do tipo dreno aberto, utilizando, portanto, apenas uma linha que pode ou não ser ligada à saída de cada AND. Na Figura 3.43 ilustra-se a estrutura utilizando essa tecnologia.

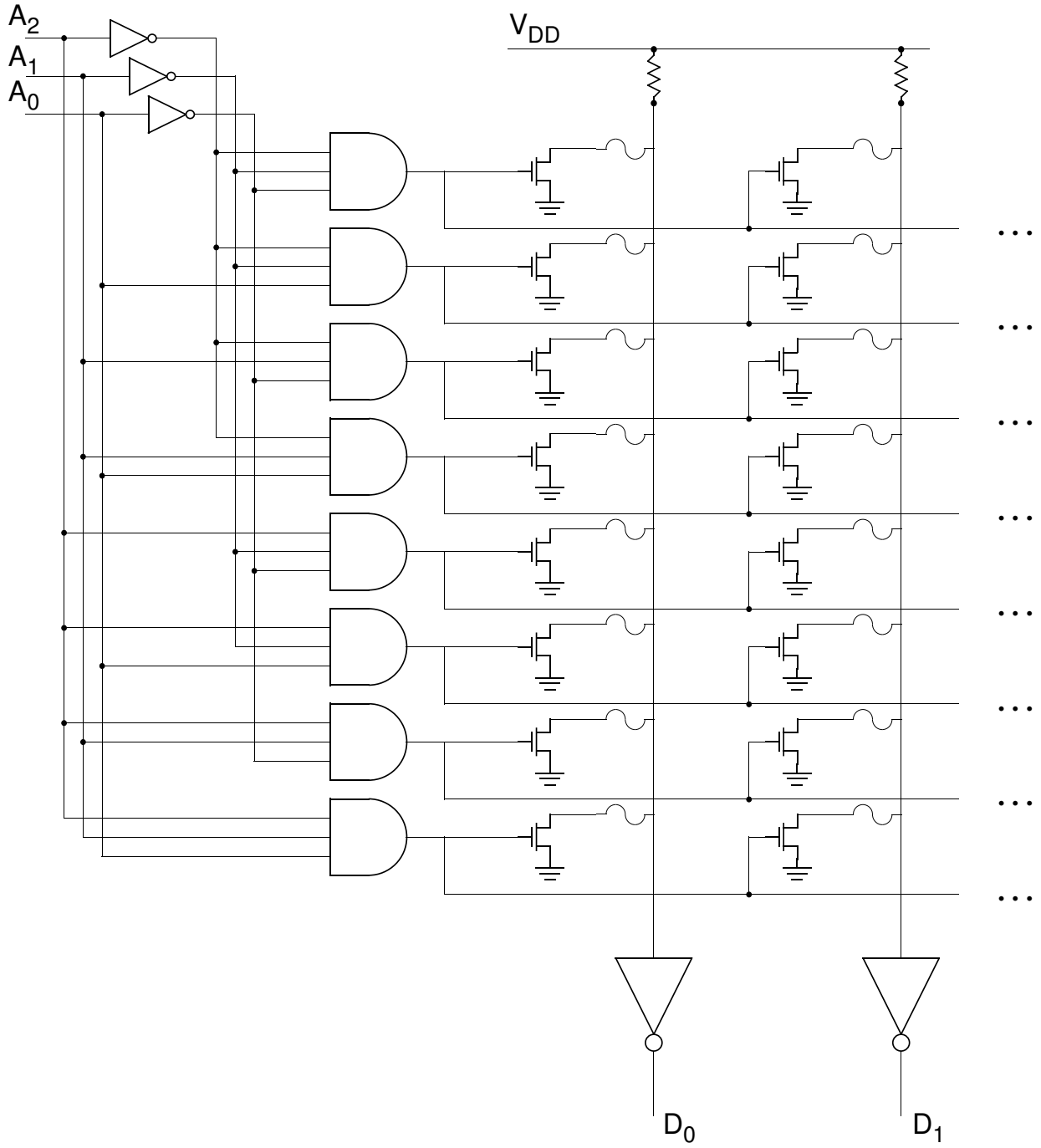


FIGURA 3.43: Estrutura interna de uma ROM.

O funcionamento de uma ROM pode ser descrito de um modo muito simples. O con-

junto das linhas A_i constitui aquilo que se convenciona designar por *endereço* (*address*, em inglês). Como se pode perceber, para cada endereço colocado na entrada da ROM, uma das linhas que sai das portas AND, a correspondente a esse endereço, é colocada a 1. As linhas de saída que tiverem uma linha de entrada efectivamente ligada a essa linha ficarão com o valor 1 na saída. Os restantes ficarão com o valor 0. O conjunto de bits das saídas designa-se por *palavra*. A ROM pode, então, ser vista como um conjunto de palavras armazenadas, das quais uma pode ser seleccionada pelas linhas de endereço e colocada nas saídas. No entanto, como é claro do logigrama, a ROM é, realmente, um circuito combinatório.

Na Figura 3.44 ilustra-se, a título de exemplo, o conteúdo de uma ROM já programada, isto é, com o seu conteúdo já definido. Refiram-se duas convenções utilizadas a partir daqui. Por um lado, em vez de desenhar a estrutura interna como realmente é, o que tornaria complexo este tipo de representação esquemática, a ligação é retomada como se existissem as portas OR de saída, mas colapsando todas as linhas numa só. Por outro lado, as ligações a fazer às saídas das portas AND são indicadas pela ocorrência de uma cruz. Por exemplo, a saída D_0 está ligado às saídas das portas AND $P2$, $P3$, $P5$ e $P7$.

As ligações entre as linhas verticais e horizontais na figura dariam origem, se fossem apenas como está descrito acima, a um enorme curto-circuito. Por exemplo, no caso que se ilustra de pormenor da programação da ROM anterior, todas as linhas estariam em curto-circuito e, portanto, a funcionalidade não seria a pretendida. De facto, no cruzamento das linhas existem pequenos dispositivos, díodos, para ser mais preciso, que impedem que essa situação se verifique.

O conteúdo de uma ROM pode ser especificado por uma tabela que indica, para cada endereço, a palavra de dados correspondente. Por exemplo, a ROM ilustrada na Figura 3.44 pode ser, de uma forma mais compacta, descrita pela Tabela 3.14.

A forma gráfica de representar uma ROM num logigrama é muito mais compacta que a representada nas figuras anteriores, que serviram apenas para ilustrar o conceito e um exemplo de estrutura interna de um dispositivo desse tipo. Na Figura 3.45 está descrito o modelo de terminais da ROM usada como exemplo anteriormente.

As memórias só de escrita podem ser de diversos tipos. As ROM propriamente ditas são circuitos integrados projectados de origem com o conteúdo pretendido. Não chegam a ser programadas pelo utilizador e não são alteráveis. As *PROM* (do inglês, *Programmable ROM*), dispositivos mais comuns que as ROM, são dispositivos que podem ser programadas uma vez por fusão de pequenos fusíveis dentro do integrado. São compradas virgens e programadas pelos utilizadores. No caso de ser necessário alterar a programação há que programar um novo dispositivo, e o anterior fica inutilizado. As *EPROM* (do inglês, *Erasable PROM*) são dispositivos que podem ser programados

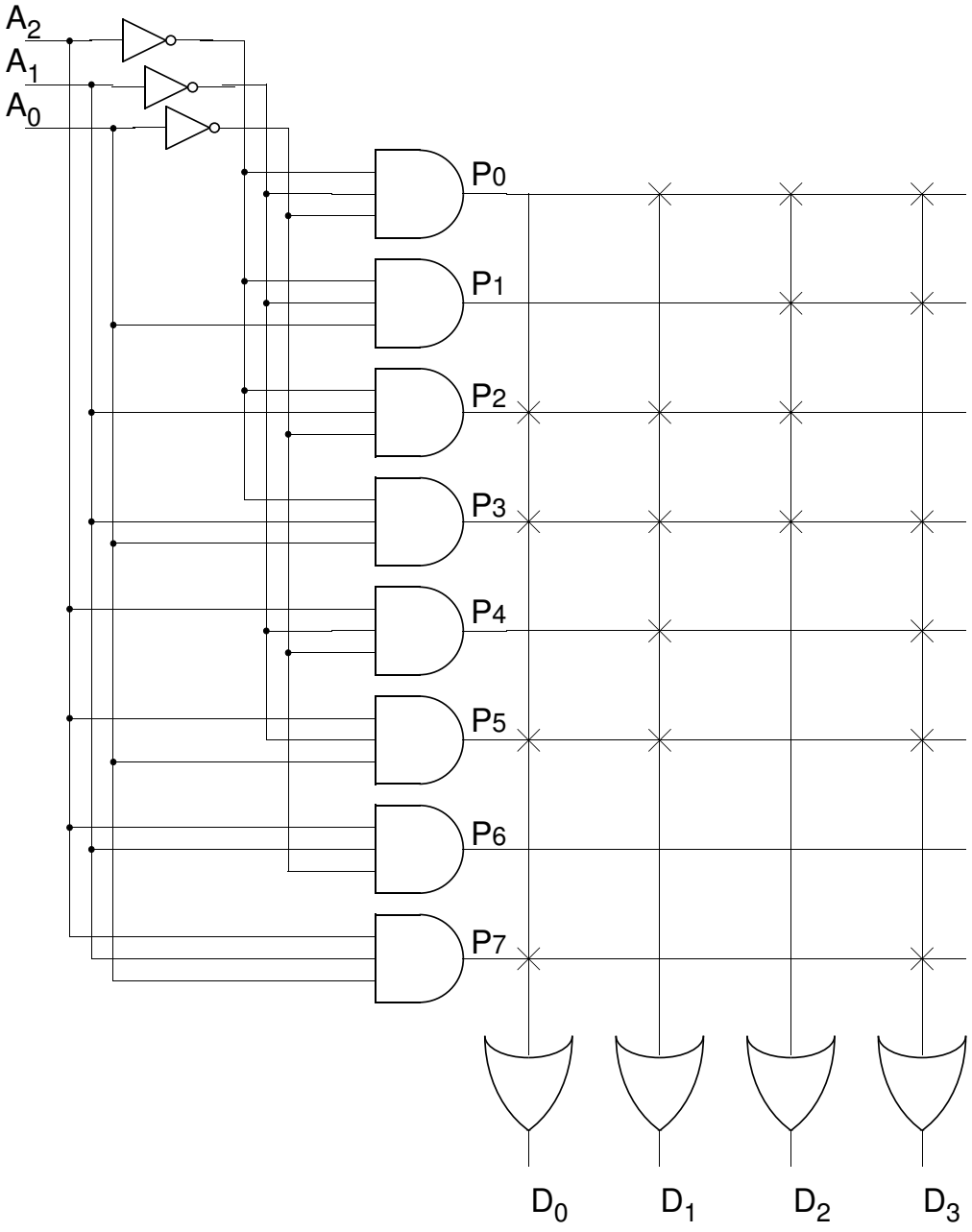


FIGURA 3.44: Exemplo de uma ROM programada.

e podem ser apagados (usualmente por um banho de alguns minutos de radiação ultravioleta). As EPROMs podem ser regravadas um certo número de vezes (dezenas a milhares). As *EEPROM* (do inglês, *Electrically Erasable PROM*) são ROM alteráveis electricamente sem necessidade de as retirar do circuito em que estão inseridas. Do mesmo modo são alteráveis um número limitado de vezes.

As ROM podem ter várias aplicações:

TABELA 3.14: Especificação por tabela da ROM da Figura 3.44.

Endereço			Dados			
A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	1	1	1	0
0	0	1	1	1	0	0
0	1	0	0	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	0
1	0	1	1	0	1	1
1	1	0	0	0	0	0
1	1	1	1	0	0	1

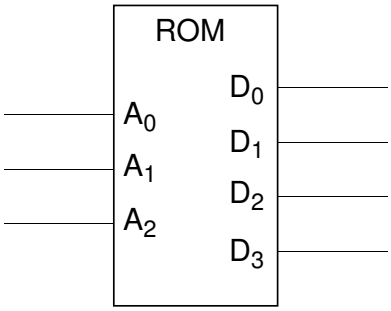


FIGURA 3.45: Modelo de terminais da ROM usada como exemplo.

— suporte a programas em sistemas embebidos que não serão alterados (uso de ROM ou PROM) ou que podem ser actualizados um número pequeno de vezes;

— memorização de tabelas e, por fim,

— implementação de lógica.

Esta última aplicação, a única que interessa neste contexto, necessita de análise mais aprofundada. A descrição de uma ROM faz-se por indicação do seu símbolo para se poder conhecer a sua estrutura física e funcionalidade geral e por uma tabela com o seu conteúdo, como se viu atrás. Usando a ROM indicada atrás como exemplo e considerando em particular a linha de dados $D0$. Uma das interpretações possíveis do conteúdo desta ROM na coluna de dados $D0$ é o de ser a tabela de uma função lógica de três variáveis (A_2, A_1 e A_0). Ora, se se utilizar esta ROM com este conteúdo num circuito, isso é equivalente à realização da função lógica descrita pela tabela e, na maior parte dos casos, gastar-se-á menos material. O circuito ficaria, portanto, reduzido à ROM, como

se ilustra na Figura 3.46, se a função for $F(X, Y, Z)$ com X correspondendo a A_2, Y a A_1 e Z a A_0.

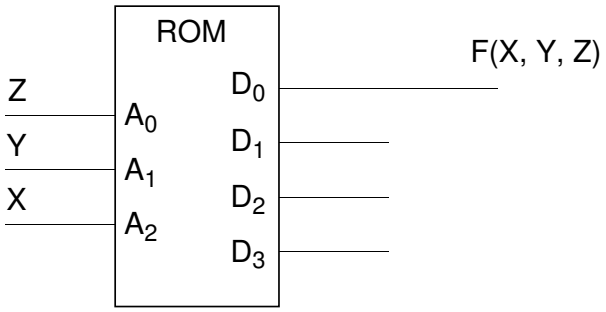


FIGURA 3.46: Circuito de uma função usando uma ROM.

É óbvio que, no exemplo anterior, a ROM poderia dar suporte a outras funções. No caso exemplificado, seria possível utilizar a ROM para dar suporte a quatro funções de três variáveis, desde que as variáveis sejam as mesmas. Pode-se considerar que este método de implementação utiliza a tabela da função em vez da sua expressão lógica, como no caso da implementação com portas. Essa é uma verificação curiosa, porque permite levar até ao nível da implementação de um circuito a dualidade de representação de funções lógicas que se descreveu no Capítulo 2.

Claro que os circuitos ROM comerciais têm muito mais linhas de endereço e, eventualmente, bits de dados, possibilitando, assim, a implementação de várias funções de um número elevado de entradas num circuito. Por exemplo, uma ROM de 8K $\times$ 8 bits pode implementar até oito funções lógicas de até 13 variáveis ($8K = 2^{13}$). Claro que algumas destas funções podem não depender de todas as variáveis de entrada.

A implementação de funções usando circuitos ROM apresenta algumas vantagens significativas. Por um lado, como é evidente, a quantidade de circuitos integrados utilizados na implementação diminui drasticamente, especialmente para circuitos de alguma complexidade, com diminuição correspondente no peso e volume dos equipamentos e, até certo ponto, da potência consumida. Por outro lado, repare-se que qualquer alteração da funcionalidade pretendida se resolve utilizando uma nova ROM ou reprogramando a já existente (no caso das EPROM) sem alterar o resto do circuito eventualmente implementado. Por fim, o uso de circuitos ROM dificulta a terceiros a análise do circuito desenvolvido, protegendo um pouco, dessa forma, a confidencialidade de um projecto.

3.5.2 Realização Usando Matrizes Lógicas Programáveis

Uma ROM gasta muito do seu espaço físico nas portas AND ligados às linhas de endereço. Por exemplo, a ROM de 8K atrás referida teria 8192 portas AND de 13 entradas. Para as aplicações em que se pretende guardar na ROM informação (programas ou tabelas) não há outro remédio. No entanto, para gerar funções lógicas, muitas vezes não são necessários todos as portas AND. Por isso, é muitas vezes mais vantajoso ter menos portas AND com maior número de entradas, o que permite realizar funções com mais entradas em integrados não muito complexos. Esta ideia foi explorada levando à concepção de dispositivos programáveis em que existe um certo número n de entradas, um número p de portas AND e um número s de portas OR, podendo as ligações entre os AND e as entradas por um lado e entre os OR e os AND, por outro, ser programadas. Estas estruturas e outras designam-se por *matrizes lógicas programáveis*. Como exemplo muito simples, considere-se o circuito da Figura 3.47. Nesta figura, o símbolo da Figura 3.48

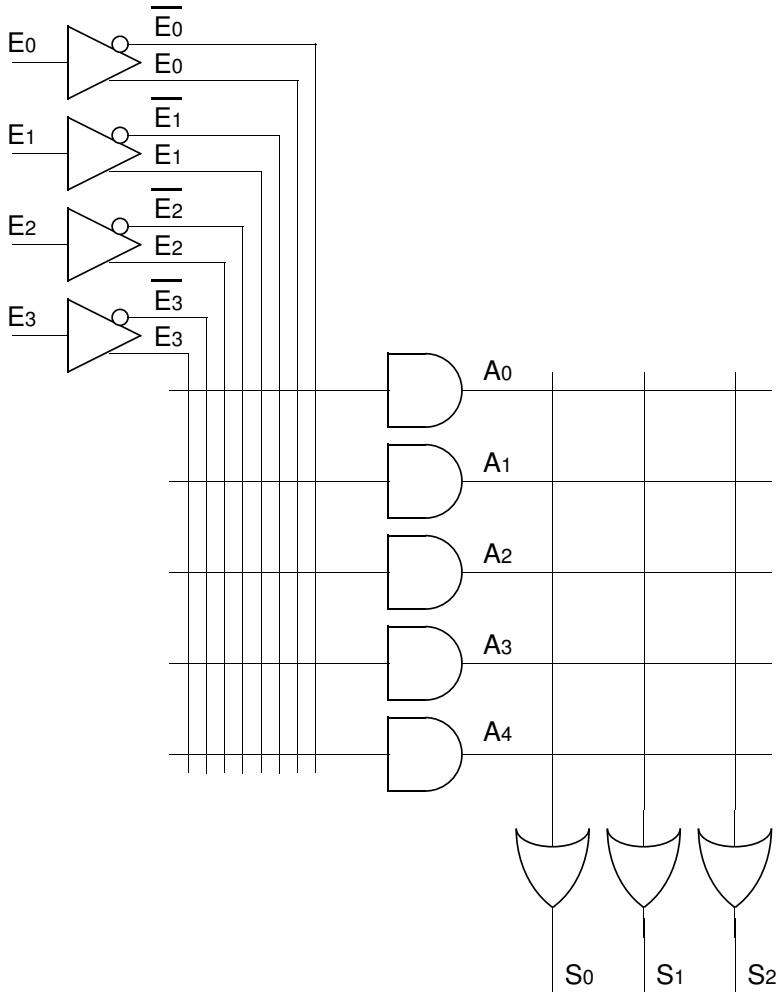


Figura 3.47: Exemplo de matriz lógica programável.

(a) é a representação abreviada do que está ilustrado na Figura 3.48 (b). Repare-se que

(a) (b)

FIGURA 3.48: Estrutura do *buffer* duplo.

o dispositivo ilustrado tem quatro entradas. Se se construísse uma ROM, teria de haver à entrada 16 portas AND. No circuito apresentado existem apenas cinco, e o número poderia ser qualquer outro, como é óbvio.

A título de exemplo, considere-se a implementação das funções descritas na Tabela 3.15 usando a matriz anterior. Se a implementação fosse feita com base numa ROM, a simples gravação da tabela numa ROM com pelo menos 16 palavras e duas linhas de dados resolveria o problema. O uso de matrizes lógicas programáveis, porém, obriga a obter expressões algébricas para as funções.

TABELA 3.15: Funções usadas como exemplo.

$ABCD$	$f_x(A,B,C,D)$	$f_y(A,B,C,D)$
0000	0	0
0001	1	1
0010	0	0
0011	1	1
0100	0	0
0101	1	1
0110	0	0
0111	1	1
1000	1	1
1001	0	1
1010	1	1
1011	1	0
1100	0	0
1101	0	1
1110	0	0
1111	1	1

No caso em análise, utilizando os métodos descritos no Capítulo 2, é possível obter as Expressões 3.12.

$$\begin{aligned} f_x(A,B,C,D) &= \overline{A}\,D + C\,D + A\,\overline{B}\,\overline{D} \\ f_y(A,B,C,D) &= \overline{A}\,D + \overline{C}\,D + B\,D + A\,\overline{B}\,\overline{D} \end{aligned} \tag{3.12}$$

A programação da matriz lógica programável está ilustrada na Figura 3.49, em que as ligações estão assinaladas por cruzes no cruzamento das linhas a interligar. Este tipo

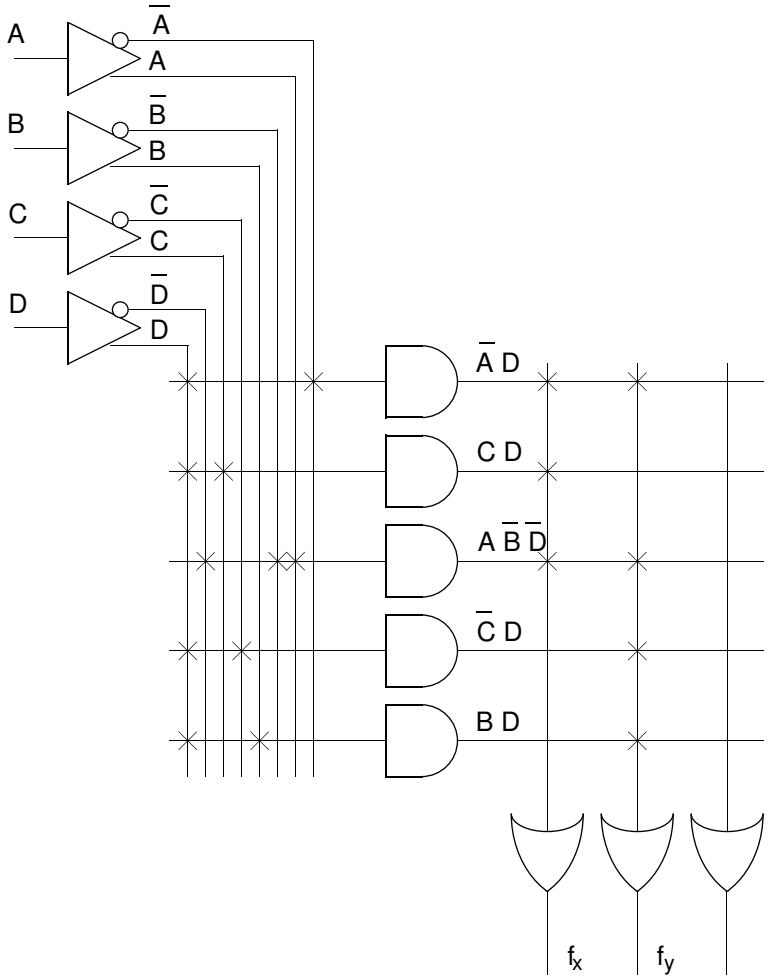


FIGURA 3.49: Exemplo de programação de matriz lógica programável.

de matriz lógica programável é usualmente designada por *PLA*, abreviatura do nome inglês *programmable logic array*.

No caso das ROM, como se viu, as ligações dos AND estão fixas e é possível programar as ligações dos OR. No caso das PLA, é possível programar ambos os tipos de ligação. Verifica-se que as potencialidades mais interessantes advêm da capacidade de programar os AND. Assim, desenvolveu-se um novo tipo de matriz lógica programável em

que as ligações entre os AND e os OR estão fixas e apenas é possível programar as ligações dos AND às entradas. Para além de, na prática, essa opção não ser limitativa, tem a vantagem de optimizar a arquitectura do dispositivo, colocando a área programável na zona onde promove uma maior flexibilidade.

Essa estrutura designa-se, em inglês, *programmable array logic*, com a abreviatura *PAL*. Um exemplo de uma possível PAL é o da Figura 3.50. É frequente, neste tipo de matriz, que algumas das saídas sejam realimentadas como se de entradas se tratasse, para permitir, por esse meio, alargar o número de portas AND usado numa função. No exemplo dado é o caso das saídas S_0 e S_1.

Apresenta-se um exemplo de programação de uma PAL para implementação de um conjunto de funções, usando a estrutura apresentada. Nesse exemplo serão implementadas as funções descritas pelas Expressões 3.13.

$$\begin{aligned} f_0 &= A\,\overline{B} + \overline{A}\,\overline{C} + \overline{B}\,C\,\overline{D} \\ f_1 &= A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C\,\overline{D} \\ f_2 &= \overline{A}\,\overline{B}\,\overline{C} + \overline{B}\,C\,\overline{D} + \overline{A}\,B\,C\,D + A\,B\,\overline{D} + A\,\overline{B}\,D \end{aligned} \tag{3.13}$$

A programação da PAL está ilustrada na Figura 3.51. Repare-se que a função f_2 necessitou de utilizar duas das estruturas AND-OR para ser implementada.

A utilização de matrizes lógicas programáveis tem vantagens semelhantes à da utilização de circuitos ROM, com alguns aspectos adicionais. Por um lado, as matrizes programáveis permitem melhorar a utilização da área programável, permitindo mais entradas para o mesmo nível de complexidade interna do dispositivo. Por outro, o uso de matrizes lógicas programáveis permite implementar funções com eliminação de transições espúrias, como se explica na Secção 3.4.2, o que não é possível no caso das ROM.

Como é evidente, a programação deste tipo de dispositivos é, em geral, feita com auxílio de software adequado, que permite a introdução da especificação das funções a programar na forma mais conveniente para o utilizador e produz um ficheiro com os dados necessários para o equipamento que programa o dispositivo realizar fisicamente a programação.

O uso de circuitos ROM e de matrizes lógicas programáveis, como se acaba de descrever, permite, portanto, a implementação de funções lógicas, isto é, de circuitos combinatórios. Em capítulos posteriores estudar-se-ão os circuitos sequenciais. A utilização de matrizes lógicas programáveis pode ser estendida para esse tipo de circuitos desde que elementos de memória sejam incluídos na estrutura da matriz.

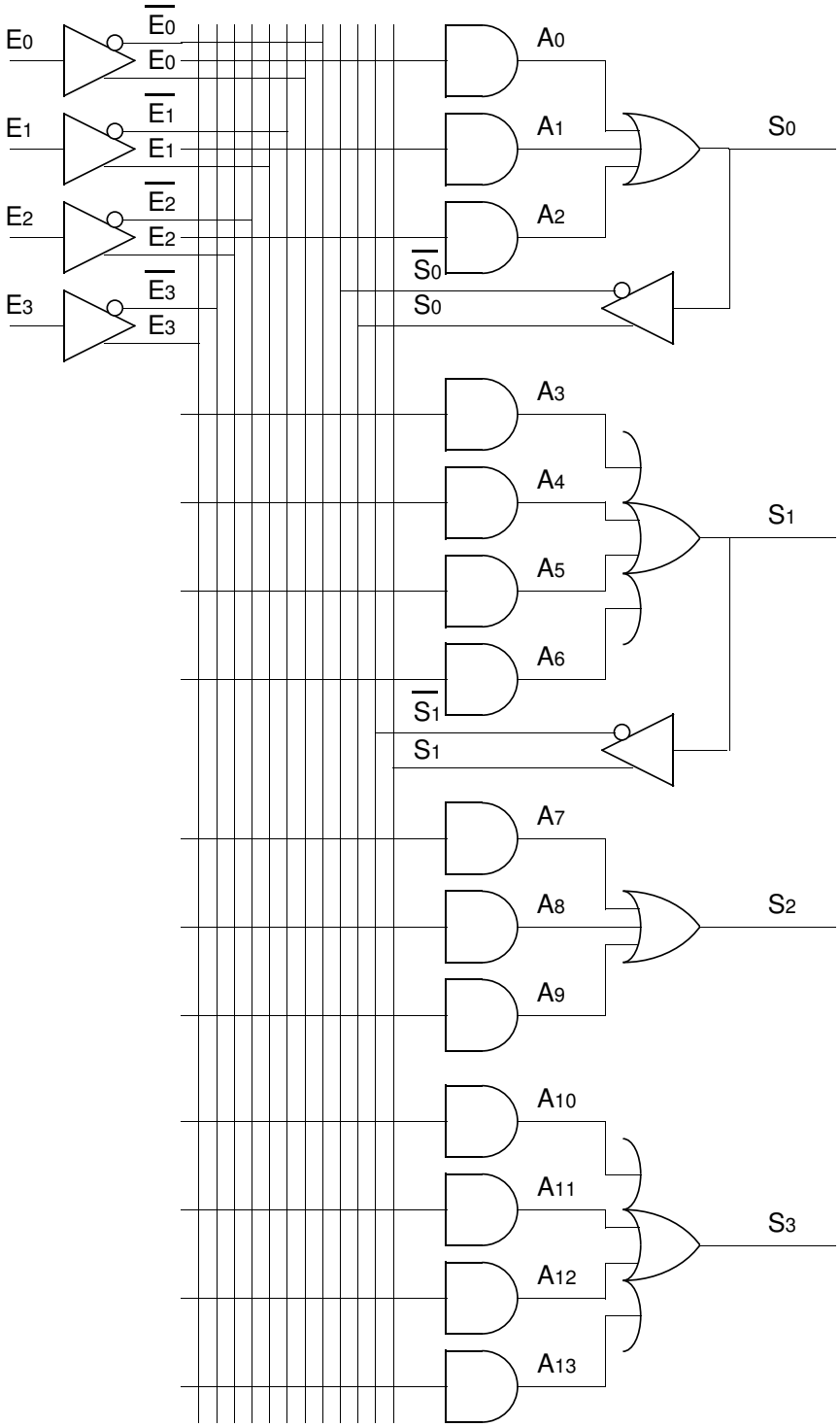


FIGURA 3.50: Exemplo de matriz lógica programável do tipo PAL.

SUMÁRIO

Este capítulo é dedicado a aspectos relacionados com a realização de circuitos lógicos a partir de funções lógicas e com as suas características físicas mais relevantes. Inicia-se

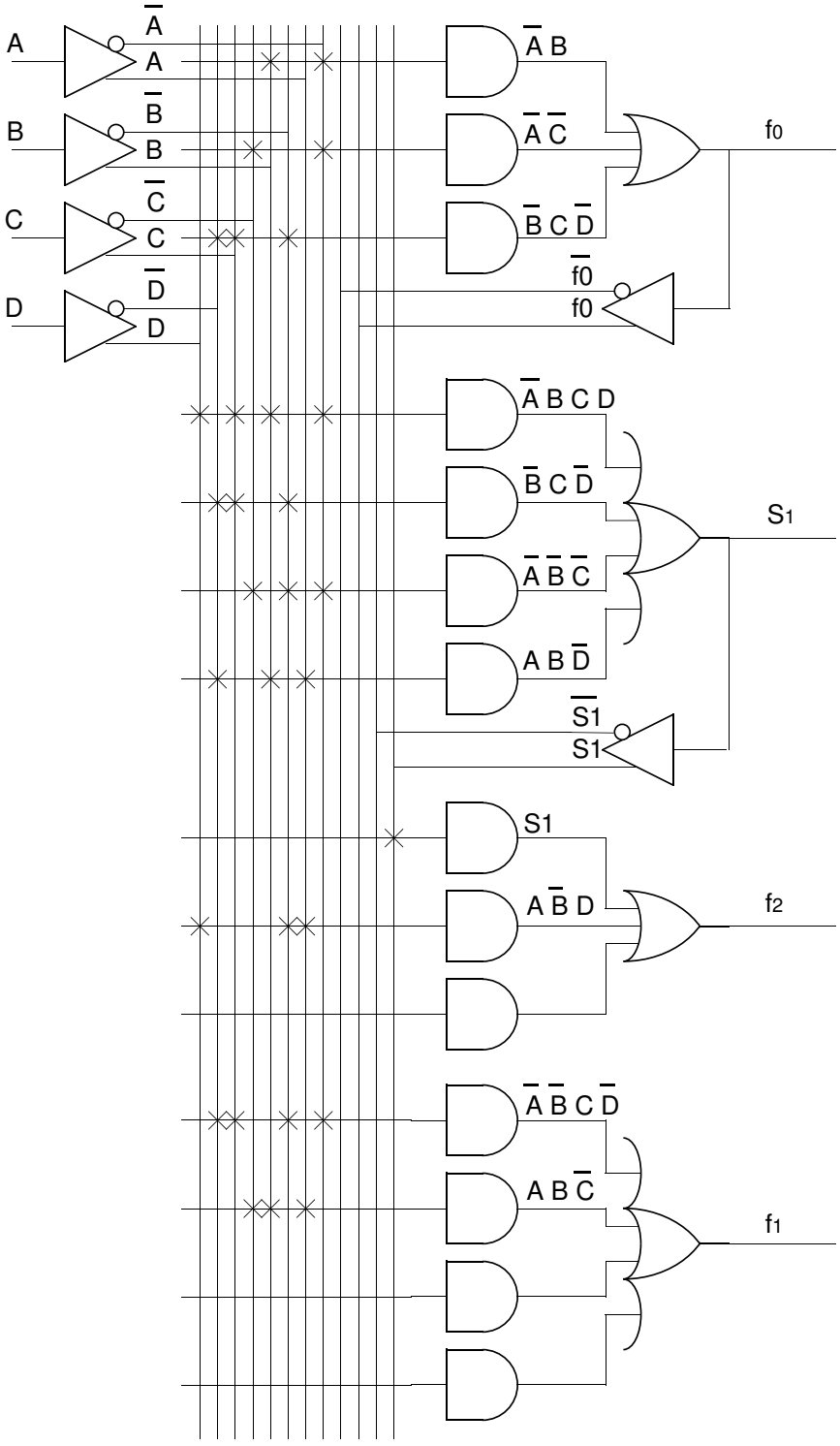


FIGURA 3.51: Exemplo de programação de uma PAL.

com a descrição dos blocos clássicos constituintes dos circuitos lógicos, isto é, os circuitos integrados digitais, incluindo a sua tecnologia e comportamento electrónico básico. O capítulo prossegue com a descrição das metodologias associadas à construção de cir-

cuitos usando portas básicas a partir da descrição de uma função lógica. É feita também uma breve introdução às técnicas de detecção de falhas. São ainda referidos com algum pormenor os aspectos ligados às características temporais dos circuitos, incluindo a geração de transições espúrias. O capítulo termina com a análise da utilização de ROM e lógica programável para implementar circuitos digitais combinatórios.

EXERCÍCIOS

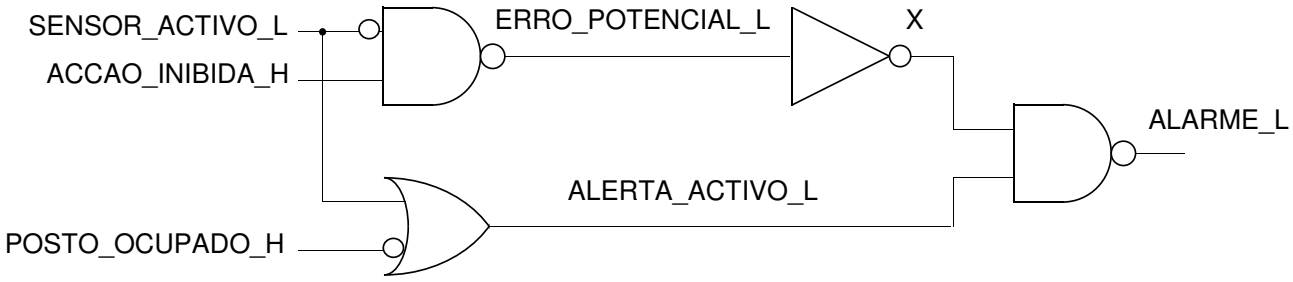
3.1 Consulte na Internet folhas de especificações de circuitos com a mesma referência, em TTL na sub-família LS e em CMOS, quer na subfamília HC, quer na HCT. Considere, por exemplo, o circuito 7400 nas diversas versões.

1. Verifique, consultando os parâmetros dos valores da tensão de entrada e saída, que a família HCT é compatível com a LS em termos de níveis de tensão, mas que, no caso das famílias HC e LS, isso não é verdade.
2. Verifique quais os atrasos típicos das três subfamílias e compare-as desse ponto de vista.

3.2 Para cada uma das seguintes funções, construa um esquema eléctrico de um circuito que as implemente, usando o número mínimo de circuitos integrados (consulte catálogos *on-line*):

1. $f(A,B,C) = A\ B + \overline{A}\ \overline{B}\ \overline{C}$
2. $f(A,B,C,D) = A\ B\ (\overline{C} \oplus D) + A\ B\ C + \overline{A}\ \overline{C}\ D + \overline{B}\ C\ D$
3. $f(A,B,C) = \overline{\overline{A}\ B}\ C + A\ \overline{\overline{B} + C}$

3.3 Considere o seguinte circuito representado em lógica de polaridade:



Supondo que as variáveis *SENSOR_ACTIVO* e *ACCAO_INIBIDA* estão activas e que a linha *POSTO_OCUPADO_H* está no nível *LOW*:

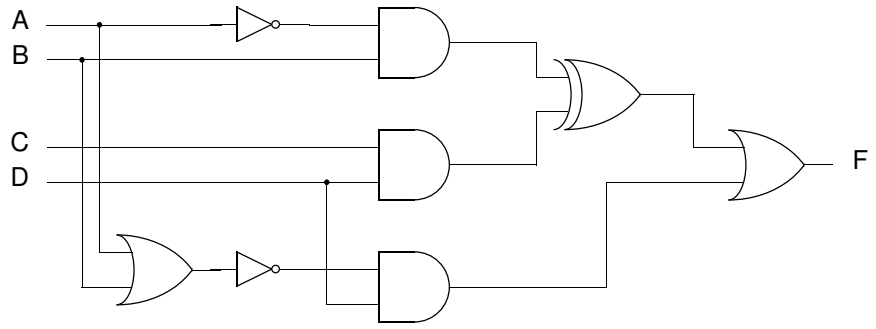
1. A variável *ERRO_POTENCIAL* está activa ou não activa?
2. A linha *ALARME_L* está a que nível?

3. Indique um nome razoável para a linha indicada com um X.
4. Qual é a função lógica implementada pelo circuito?

3.4 Considere que dispõe dos seguintes sinais que implementam variáveis segundo a convenção de lógica de polaridade: A_H, B_L, C_L e D_H. Obtenha o esquema eléctrico das seguintes funções, utilizando lógica de polaridade, considerando que a função f é activa a LOW.

1. $f(A,B,C,D) = A\ B + \overline{A}\ B\ D + A\ \overline{B}\ C\ \overline{D}$
2. $f(A,B,C,D) = A\ B(\overline{C} \oplus D) + A\ B\ C + \overline{A}\ \overline{C}\ D + \overline{B}\ C\ D$

3.5 Considere o circuito seguinte:



Obtenha, a partir de um catálogo *on-line*, as características temporais das portas envolvidas.

1. Qual o caminho crítico deste circuito?
2. Há um sentido da transição da variável de entrada do caminho crítico que conduza a um tempo de atraso superior ao da transição em sentido oposto?
3. Qual(is) a(s) configuração(ões) das variáveis de entrada que sensibilizam o caminho crítico?

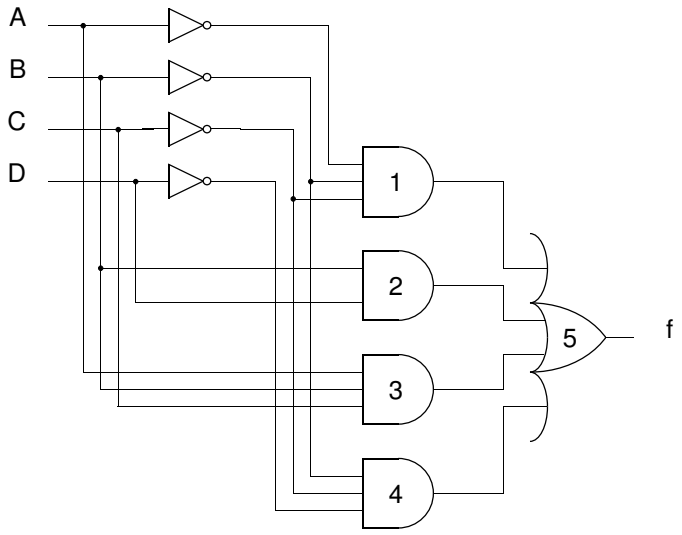
3.6 Utilizando uma ROM com a capacidade que especificar, construa um circuito combinatório que calcule n^3, sendo n um número binário de quatro bits.

3.7 Programe a PAL da Figura 3.50 para implementar o seguinte conjunto de funções lógicas:

$$\begin{aligned}
f(A,B,C,D) &= \overline{A}\ \overline{C} + \overline{B}\ \overline{C}\ D + A\ B\ C + \overline{A}\ \overline{B}\ \overline{D} \\
f(A,B,C,D) &= \overline{A}\ B + A\ D + \overline{A}\ B\ \overline{C} + \overline{A}\ \overline{C}\ \overline{D} \\
f(A,B,C,D) &= \overline{C}\ D + A\ D + \overline{A}\ \overline{B}\ \overline{C} + \overline{A}\ B\ \overline{C} + \overline{A}\ \overline{B}\ C\ \overline{D} \\
f(A,B,C,D) &= \overline{C}\ D + \overline{A}\ B\ \overline{C} + \overline{A}\ \overline{B}\ C\ \overline{D}
\end{aligned}$$

3.8 Considere de novo o circuito do Problema 3.3 Desenhe um esquema eléctrico que o implemente.

3.9 Considere o seguinte circuito:



1. Construa o mapa de Karnaugh correspondente.
2. Considere que, devido a uma avaria, na configuração $A = B = C = 1$, $D = 0$ o resultado da função dá 0 em vez do valor 1, que devia dar. Quais das portas podem estar envolvidas no problema?
3. Se a ligação entre as portas 4 e 5 não estiver feita e, em vez disso, o fio que devia estar ligado à porta 4, estiver ligado da saída da porta 3 à porta 5, quais são os valores errados no funcionamento da função?

3.10 Considere o circuito do Problema 3.9.

1. Este circuito pode ter transições espúrias quando uma determinada variável se altera. Em que circunstâncias?
2. Altere o circuito anterior, de modo a eliminar as transições espúrias referidas.