SISTEMAS DIGITAIS

APONTAMENTOS DAS AULAS TEÓRICAS

Guilherme Arroz

Carlos Sêrro

Versão 1.1
3 de Agosto de 2005

Instituto Superior Técnico
Departamento de Engenharia Electrotécnica
e de Computadores
TagusPark
Porto Salvo

Historial

3 de Agosto de 2005	v1.1	Versão corrente.
22 de Fevereiro de 2005	v1.0	Foi adicionado o Capítulo 19, e foram feitas correcções no texto da versão 0.2
6 de Dezembro de 2004	v0.2	Foram adicionados os Capítulos 17 e 18, e feitas correcções no texto da versão 0.1
26 de Outubro de 2004	v0.1	Foram adicionados os Capítulos 15 e 16, e feitas correcções no texto da versão 0.0
14 de Setembro de 2004	v0.0	Versão original

Referências

Endereço de e-mail: `cas@digitais.ist.utl.pt`

Página da cadeira de Sistemas Digitais: `http://sd.tagus.ist.utl.pt`

Versão 1, revisão 1, de 3 de Agosto de 2005

Prefácio

Este texto foi desenvolvido a partir dos resumos das aulas teóricas da cadeira de Sistemas Digitais das licenciaturas em Engenharia Informática e de Computadores (LEIC), em Engenharia Electrónica (LEE), e em Redes de Comunicação e de Informação (LERCI), leccionados no ano lectivo de 2003/2004 no *campus* do TagusPark do Instituto Superior Técnico.

Como tal, o presente texto representa uma tentativa de incluir num único local os apontamentos teóricos essenciais à compreensão da matéria de Sistemas Digitais, tal como ela foi ensinada nesse ano lectivo. Por isso, não deve ser entendido como um texto completo, mas sim como uma colecção de resumos teóricos que **não dispensam o estudo mais aprofundado da matéria** por um dos textos referenciados na bibliografia indicada no fim de cada um dos capítulos.

No essencial, o texto segue a matriz das aulas teóricas, tais como elas foram desenvolvidas por um de nós (Guilherme Arroz) e postas à disposição dos alunos nesse ano lectivo. Aqui e ali o texto foi revisto e aumentado, com o objectivo de o tornar mais claro, por vezes mais completo.

Organização do texto

O presente texto encontra-se organizado em 3 partes: a primeira trata dos fundamentos teóricos subjacentes a todos os circuitos e sistemas digitais, e cobre os Capítulos 1 a 5; a segunda trata dos circuitos combinatórios, e vai do Capítulo 6 ao Capítulo 11; e a terceira parte lida com os circuitos sequenciais, nos Capítulos 12 a 19.

Em simultâneo desenvolveram-se ainda dois outros textos, pelo que presentemente estão disponíveis:

- o presente documento, com os resumos das aulas teóricas, que designamos por "Sistemas Digitais: Apontamentos das Aulas Teóricas" ou, mais sucintamente, por *SD:AAT*;
- um manual com a resolução de alguns exercícios, os mais importantes, com o nome "Sistemas Digitais: Exercícios Resolvidos", que designamos abreviadamente por *SD:ER*; e
- um Glossário, designado simbolicamente por *SD:GL*.

Em futuras versões prevê-se a inclusão, em *SD:AAT*, de alguns apêndices com matéria que complementa a que é dada nas aulas teóricas, e que a completa.

Salientam-se, em particular, um apêndice com uma descrição sucinta da notação de dependência usada na Norma IEC 60617-12, e um outro com as regras essenciais que estão na base de um correcto desenho dos diagramas de blocos, dos logigramas e dos esquemas eléctricos. Finalmente prevê-se a inclusão futura de um apêndice que descreve os métodos mais simples de análise que permitem o diagnóstico de falhas nos circuitos digitais.

Notas à margem e índice remissivo

Espalhados pelo texto podemos encontrar três tipos de notas à margem:

— chamadas de atenção para partes do texto particularmente importantes, geralmente escritas em itálico; são referenciadas pelo símbolo especial que se mostra nesta margem, à esquerda;

Este é um comentário obviamente pouco interessante.

— comentários *em itálico* destinados a complementar a matéria; serão incluídos ao longo das sucessivas versões do texto, à medida que a reacção dos alunos a determinados pontos mais obscuros ou difíceis justifique as suas inclusões; e

Os conceitos em **negrito** *estão no Glossário*

— conceitos chave, em **negrito**, que constituem as entradas do Glossário (*SD:GL*). Nesse sentido, o Glossário é m coadjuvante importante do texto, na medida em que sistematiza os principais conceitos num único documento. Os conceitos são referenciados por entradas à margem, em cor, o que facilita a sua procura após consulta ao Glossário ou ao Índice Remissivo.

No Índice Remissivo encontram-se três tipos de referências: (i) as normais, em tipo direito, para indicar as páginas do texto onde se encontram os conceitos que não precisam de ser salientados de forma especial; (ii) as referências *em itálico*, que identificam as páginas do texto onde existem conceitos chave, em negrito e acentuados à margem; e (iii) as entradas do Glossário ou de outros apêndices, **em negrito**.

Agradecimentos

Os autores estão agradecidos aos alunos Paulo Gomes, João Nunes e João Loureiro, que apontaram erros no texto de versões anteriores e que sugeriram, em alguns casos, alterações ao mesmo, com o propósito de o tornar mais claro.

Oeiras, 3 de Agosto de 2005

Guilherme Arroz Carlos Sêrro

Índice

Parte I

FUNDAMENTOS

Capítulo 1

Sistemas de Numeração

1.1 Sistemas de numeração posicionais

Estamos tão habituados a contar e a executar as operações básicas (adição, subtracção, multiplicação e divisão) usando um **sistema decimal**, com **base** $b = 10$, que nem paramos para pensar por um momento nos algoritmos que estão na base dessas operações.

Sistema decimal
Base 10

A utilização dessa base era inevitável, já que a humanidade se habituou, desde muito cedo, a contar pelos dedos das mãos. Ocasionalmente usa outros sistemas de numeração. Por exemplo, usa o sistema sexagesimal, com base $b = 60$, para contar as unidades horárias de "minutos" e "segundos", ou o sistema duodecimal com base $b = 12$, ou o sistema de base $b = 24$, para identificar as "horas" do dia.

Sistema sexagesimal
Base 60
Sistema duodecimal
Base 12

Todos estes sistemas são **posicionais** ou **ponderados**, na medida em que cada número é formado por uma sequência de **dígitos** ou **algarismos**, em que cada dígito possui um **peso** característico da posição que ocupa na sequência.

Sistemas posicionais ou ponderados
Dígito ou algarismo
Peso

Nada nos impede, contudo, de usar sistemas com bases não naturais, por exemplo com bases inteiras negativas, bases racionais e irracionais, reais, ou até bases complexas. E usamos ainda **sistemas não posicionais**, como é o caso do sistema de numeração romano, em que não há pesos associados aos dígitos.

Sistemas não posicionais

Neste curso vamos estar interessados apenas nos sistemas posicionais (que designamos, mais simplesmente, por **sistemas de numeração**) de base b e, em particular, nos sistemas de numeração binário (com $b = 2$) e hexadecimal (com $b = 16$). Ocasionalmente, refererir-nos-emos a outros sistemas de numeração, com uma base natural arbitrária.

Sistema de numeração de base b

Por outro lado, começaremos por estudar os **números sem sinal**, deixando para a Secção 1.4 o estudo dos números com sinal.

Números sem sinal

Para que cada quantidade numérica seja expressa por uma e só uma sequência de dígitos, um sistema de numeração de base b arbitrária possui b dígitos, sendo que cada um deles deve exprimir, por si só, um número diferente, menor do que b. Assim, um dígito exprimirá a ausência de qualquer quantidade, usando-se, nesse caso, o símbolo 0 para o representar. Outro dígito expressará a quantidade

unitária, e o símbolo 1 representará esse dígito. Os restantes dígitos exprimirão quantidades de duas, três, quatro, ..., $(b-1)$ unidades.

Um número sem sinal escrito num sistema de numeração de base b, que designaremos por $N_{(b)}$, vem representado por uma sequência infinita de dígitos,

$$N_{(b)} <> \cdots d_2\, d_1\, d_0, d_{-1}\, d_{-2} \cdots \ ,$$

em que o símbolo <> é utilizado indistintamente para significar que $N_{(b)}$ "é representado pela sequência", ou que "a sequência representa" $N_{(b)}$. Nesta sequência, cada dígito d_i é ponderado por um peso b^i associado à sua posição. Se agora efectuarmos a soma polinomial

$$N_{(10)} = \sum_{-\infty}^{+\infty} d_i b^i = \cdots + d_2 b^2 + d_1 b^1 + d_0 b^0 + d_{-1} b^{-1} + \cdots$$

Equivalente decimal

na base 10, obtemos o **equivalente decimal** $N_{(10)}$ de $N_{(b)}$.

A igualdade anterior sugere que um número sem sinal deverá ser representado por uma sequência infinita de dígitos. Tal sequência é, naturalmente, impossível de escrever ou de utilizar. Nessas condições, torna-se finita a sequência procedendo às seguintes operações: (i) truncamento dos dígitos não significativos à esquerda da representação, se existirem; (ii) truncamento dos dígitos à direita da representação ou, quando tal não for possível, arredondamento do dígito de menor peso.

Obtemos, nessas condições, uma sequência finita,

$$N'_{(b)} <> d_{q-1}\, d_{q-2} \ldots d_1\, d_0\ ,\ d_{-1} \ldots d_{-p}\ ,$$

com precisão eventualmente reduzida em relação à de $N_{(b)}$, sequência essa com q dígitos para a esquerda e p dígitos para a direita da vírgula (eventualmente, p pode ser nulo). O equivalente decimal de $N'_{(b)}$ vem, então, dado pela soma

$$N'_{(10)} \simeq \sum_{i=-p}^{q-1} d_i b^i = d_{q-1} b^{q-1} + \cdots + d_0 b^0 + d_{-1} b^{-1} + \cdots + d_{-p} b^{-p}\ .$$

Um exemplo ajuda a clarificar estas questões. O número $460,13_{(7)}$ tem por equivalente decimal o número

$$\begin{aligned} 4 \times 7^2 + 6 \times 7^1 + 1 \times 7^{-1} + 3 \times 7^{-2} = & \\ &= 4 \times 49 + 6 \times 7 + 1 \times \frac{1}{7} + 3 \times \frac{1}{49} \\ &= 196 + 42 + 0,142857 \cdots + 0,061224 \cdots \\ &= 238,204081 \cdots\ . \end{aligned}$$

Notemos que este resultado possui uma parte fraccionária infinita. Se a quisermos tornar finita — o que podemos ou não querer fazer, consoante o problema que tivermos em mãos — precisamos de a truncar ou de a arredondar.

Escrevemos, então, por exemplo, que

$$460,13_{(7)} \simeq 238,20408_{(10)}$$

se quisermos obter uma precisão de 5 casas decimais fraccionárias (neste caso truncámos a parte fraccionária) , ou

$$460,13_{(7)} \simeq 238,2041_{(10)}$$

se apenas precisarmos de 4 casas decimais (neste caso arredondámo-la).

Mais à frente estudaremos a questão da precisão a dar a um número fraccionário, numa base arbitrária, que é como quem diz, a questão da truncagem ou do arredondamento correcto da parte fraccionária.

Uma questão final sobre a designação a dar a um número escrito numa base arbitrária. Estamos habituados a dizer que o número "trinta e sete vírgula cinco" se representa por $37,5$. Mas, naturalmente, tal só é possível num sistema de base 10, em que "trinta" tem o significado de 3×10. Ou seja, a frase "trinta e sete vírgula cinco" representa apenas o número $37,5_{(10)}$ e não possui qualquer espécie de significado em qualquer outro sistema de numeração.

E se quisermos identificar verbalmente o número $37,5_{(8)}$? Ou o número $37,5_{(9)}$? Ou ainda o número $37,5_{(b)}$, com $b \geq 11$?

O maior dos dígitos no número $37,5_{(b)}$ é o 7. Logo, b tem que ser, pelo menos, igual a 8.

Naturalmente, já não podemos verbalizar esses números da mesma forma. A maneira correcta de o fazer consiste em escrever que o número $37,5_{(b)}$ se designa por "três sete vírgula cinco na base b". E isto porque o "três" na base $b = 10$ é ainda o "três" numa base em que $b \neq 10$ e $b \geq 4$, outro tanto acontecendo com os outros dois dígitos.

Pela mesma ordem de razões, dizemos que um número decimal tem "dízima" ou "parte decimal", mas que um número numa base arbitrária $b \neq 10$ tem "parte fraccionária".

1.1.1 Representação na base 2

No **sistema binário** ou sistema de **base b=2** usam-se apenas os dígitos 0 e 1, que também se designam por **bits**. Assim sendo, um número binário legítimo é, por exemplo, o número $1101,11_{(2)}$, que tem por equivalente decimal o número que resulta da soma

Sistema binário

Base 2

Bit

$$\begin{aligned} 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = \\ = 1 \times 8 + 1 \times 4 + 1 \times 1 + 1 \times \frac{1}{2} + 1 \times \frac{1}{4} \\ = 8 + 4 + 1 + 0,5 + 0,25 = 13,75_{(10)}\,. \end{aligned}$$

Como podemos constatar, a dízima é, neste caso, finita. Escrevemos, então,

$$1101,11_{(2)} = 13,75_{(10)}$$

se quisermos duas casas na parte decimal, ou

$$1101,11_{(2)} \simeq 13,8_{(10)}$$

se apenas quisermos uma casa decimal, ou ainda

$$1101,11_{(2)} \simeq 14_{(10)}$$

se não quisermos nenhuma.

Para o sistema de numeração binário é útil o conhecimento das potências de 2 mais importantes, porque nos ajuda a manejar com mais facilidade os pesos b^i necessários às conversões entre bases.

Por exemplo, ajuda saber que o 1 mais significativo (com maior peso, mais à esquerda) no número $110100011_{(2)}$ tem peso $2^8 = 256$.

Na Tabela 1.1 representam-se algumas potências 2^n, para valores inteiros de n entre -10 e $+10$.

Tabela 1.1: Potências 2^n, para n inteiro entre -10 e $+10$

2^n	n	2^{-n}
1	0	1,0
2	1	0,5
4	2	0,25
8	3	0,125
16	4	0,0625
32	5	0,03125
64	6	0,015625
128	7	0,0078125
256	8	0,00390625
512	9	0,001953125
1k = 1024	10	0,0009765625

Em particular, chama-se a atenção para a designação 1k dada à potência $2^{10} = 1\,024_{(10)}$, que é a potência de 2 que mais se aproxima de $1\,000_{(10)}$ (por essa razão, o factor k toma a designação habitual de "quilo" no sistema binário).

Para $10 < n < 20$ usam-se exclusivamente estas designações. Por exemplo,

$$2^{16} = 2^6 \times 2^{10} = 64 \times 1\,024_{(10)} = 65\,536_{(10)}\,,$$

valor este que habitualmente se designa por 64 k.

Para $n = 20$ usa-se a potência

$$2^{20} = 2^{10} \times 2^{10} = 1\,024 \times 1\,024 = 1\,048\,576\,.$$

Também neste caso $2^{20} = 1\,\mathrm{M} = 1\,\mathrm{k} \times 1\,\mathrm{k}$ é a potência de 2 que mais se aproxima de $10^6_{(10)}$, pelo que o factor M toma a designação habitual de "mega".

Finalmente, para $n = 30$ usa-se a potência

$$2^{30} = 1\,\mathrm{M} \times 1\,\mathrm{k} = 1\,\mathrm{G}\,.$$

Sendo 1 G a potência de 2 que mais se aproxima de $10^9_{(10)}$, o factor G toma a designação habitual de "giga".

Assim, temos, por exemplo,

$$2^{23} = 2^3 \times 2^{20} = 8\,\mathrm{M}\,,$$

e

$$2^{39} = 2^9 \times 2^{30} = 512\,\mathrm{G}\,.$$

1.1.2 Representação na base 16

Sistema hexadecimal
Base 16

No **sistema hexadecimal**, com **base b=16**, usamos 16 dígitos. É claro, teremos que encontrar 16 representações (símbolos) diferentes para cada um deles.

No sistema decimal usamos os dígitos 0 a 9, que podemos continuar a utilizar no sistema hexadecimal. Mas agora precisamos de "inventar" dígitos. A forma habitual de o fazer consiste em recorrer às primeiras 6 letras do alfabeto para representar os dígitos hexadecimais que, no sistema decimal, correspondem aos números (sequências de dois dígitos) 10 a 15. Então, no sistema hexadecimal usamos, para além dos símbolos 0 a 9, também as letras "a" a "f", ou, o que é o mesmo, "A" a "F".

E como estabelecer os equivalentes decimais destas letras? É simples: fazemos

$$\begin{aligned} \mathrm{A}_{(16)} = \mathrm{a}_{(16)} &= 10_{(10)}\,, \\ \mathrm{B}_{(16)} = \mathrm{b}_{(16)} &= 11_{(10)}\,, \\ &\cdots \\ \mathrm{F}_{(16)} = \mathrm{f}_{(16)} &= 15_{(10)}\,. \end{aligned}$$

Naturalmente os dígitos decimais mantêm-se no sistema hexadecimal: $0_{(16)} = 0_{(10)}$, $1_{(16)} = 1_{(10)}$, etc.

Por exemplo, um número hexadecimal legítimo é $\mathrm{F30{,}DA}_{(16)}$, que terá por equivalente decimal o número que resulta da soma

$$\begin{aligned} 15 \times 16^2 + 3 \times 16^1 + 0 \times 16^0 + 13 \times 16^{-1} + 10 \times 16^{-2} &= \\ &= 15 \times 256 + 3 \times 16 + 13 \times \frac{1}{16} + 10 \times \frac{1}{256} \\ &= 3840 + 48 + 0,8125 + 0,0390625 \\ &= 3888,8515625_{(10)}\,. \end{aligned}$$

Mais uma vez, estamos na presença de um número com uma dízima finita. Escrevemos, então, por exemplo

$$\mathrm{F30,DA}_{(16)} \simeq 3888,9_{(10)}$$

se apenas quisermos uma precisão de uma casa decimal (neste caso arredondámos a dízima).

1.2 Conversão entre Bases

1.2.1 Conversão entre a base 10 e as bases 2 e 16

Uma vez que já sabemos obter o equivalente decimal de um número escrito no sistema binário ou no sistema hexadecimal (no fundo, já sabemos converter da base 2 ou 16 para a base 10), precisamos agora de aprender a converter da base 10 para a base 2 ou para a base 16, e ainda entre as bases 2 e 16.

No caso geral, o problema da conversão entre sistemas de numeração consiste no seguinte: dado um número N representado num sistema de base b_1, ou

seja, dado $N_{(b_1)}$, pretende-se representar esse número num sistema de base b_2, obtendo-se $N_{(b_2)}$.

No caso geral, quando $b_1 \neq 10$ e $b_2 \neq 10$, a mudança pode efectuar-se directamente, utilizando operações aritméticas na base b_1 ou na base b_2, existindo para tanto algoritmos apropriados para os dois casos.

Contudo, a mudança vem geralmente facilitada se for efectuada através do sistema de numeração de base 10, utilizando-se exclusivamente operações aritméticas nessa base. Assim, por exemplo, para converter um número, dado no sistema de base $b_1 = 3$, para o seu equivalente no sistema com base $b_2 = 7$, converte-se primeiro o número para o seu equivalente na base 10 e, em seguida, opera-se a conversão para a base $b_2 = 7$.

Este processo não introduz erros no tocante à conversão da parte inteira. Contudo, a conversão da parte fraccionária nem sempre dá um resultado finito e, nesses casos, há que arredondá-lo ou truncá-lo. Então, a utilização da base 10 como passo intermédio pode, se não pusermos os cuidados devidos, provocar uma acumulação de erros que devemos evitar.

Quando $b_1 = 10$ e $b_2 \neq 10$, a conversão pode ser feita com as operações aritméticas na base b_2, o que torna esse método por vezes incómodo. Por isso, usa-se frequentemente um método diferente que utiliza a aritmética da base b_1, isto é, a aritmética decimal. Nesse método opera-se em duas fases, uma que trata a parte inteira, e outra que trata a parte decimal do número.

Método das divisões sucessivas

A conversão da parte inteira é feita pelo **método das divisões sucessivas**, que consiste em *reter os restos das sucessivas divisões por b_2 do número decimal dado e dos quocientes entretanto obtidos, até obter um quociente nulo*.

Como exemplo, admitamos que queríamos converter $136_{(10)}$ para a base 2. Fazemos as seguintes divisões:

136	2						
0	68	2					
↑	0	34	2				
d_0	↑	0	17	2			
	d_1	↑	1	8	2		
		d_2	↑	0	4	2	
			d_3	↑	0	2	2
				d_4	↑	0	1	2
					d_5	↑	1	0
						d_6	↑	
							d_7	

Obtemos, assim,

$$136_{(10)} <> 10001000_{(2)}\,.$$

Método das multiplicações sucessivas

Para a conversão da parte decimal utiliza-se o **método das multiplicações sucessivas**, que consiste em *reter as partes inteiras das multiplicações por b_2 da parte decimal dada e das sucessivas partes decimais entretanto obtidas, até que seja atingida a precisão pretendida para a parte fraccionária do número convertido* (mais à frente estudaremos a questão da precisão mais em pormenor).

Por exemplo, para converter $0,375_{(10)}$ para a base 2 fazemos

$$\begin{aligned} 0,375 \times 2 = 0,750 &\rightarrow d_{-1} = 0 \\ 0,750 \times 2 = 1,500 &\rightarrow d_{-2} = 1 \\ 0,500 \times 2 = 1,000 &\rightarrow d_{-3} = 1\,. \end{aligned}$$

Ou seja, $0,375_{(10)} <> 0,011_{(2)}$ (neste caso particular obteve-se uma parte fracionária finita).

A conversão de um número na base $b_1 = 10$ para a base $b_2 = 16$ pode ser feita usando os mesmos algoritmos. Basta, para isso, obter os restos das divisões por 16 e as partes inteiras das multiplicações por 16. Contudo, existe um processo mais expedito que consiste em converter o número dado para a base 2, e em seguida convertê-lo para a base 16 utilizando o método que iremos estudar já a seguir.

1.2.2 Conversão da base 2 para a base 16

Este processo de conversão vem simplificado (não necessitamos de passar pela base 10) porque a base $b_2 = 16$ é uma potência da base $b_1 = 2$.

Para percebermos o que está em jogo, vamos estabelecer um paralelo entre cada dígito hexadecimal e o seu equivalente binário.

Vamos então escrever, na Tabela 1.2, essas equivalências (já agora, incluímos também os equivalentes decimais).

Como podemos constatar, *cada dígito hexadecimal é formado sempre por sequências de* 4 *dígitos binários*.

Então, no processo de conversão para a base 16 basta-nos formar grupos de 4 bits a partir da vírgula do número binário, para a esquerda e para a direita, e escrever um dígito hexadecimal por cada grupo. Nos casos em que, ao fazermos os agrupamentos, restam grupos com menos do que 4 bits (nas posições mais significativas e menos significativas), preenchem-se esses agrupamentos com zeros não significativos até ficarem com 4 bits.

Por exemplo, para converter $11110,101_{(2)}$ para a base 16 fazemos os agrupamentos

$$0001 \quad 1110 \quad , \quad 1010$$

e as correspondências

$$\begin{array}{cccc} 0001 & 1110 & , & 1010 \\ 1 & \mathrm{E} & , & \mathrm{A}_{(16)} \end{array}$$

de onde resulta que $11110,101_{(2)} = 1\mathrm{E},\mathrm{A}_{(16)}$.

1.2.3 Conversão da base 16 para a base 2

Este processo de conversão é o oposto do anterior: o número na base 16 vê cada um dos seus dígitos decomposto em 4 bits. Por exemplo, se quisermos converter para binário o número $4\mathrm{A}9,\mathrm{E}_{(16)}$ fazemos

Tabela 1.2: Equivalências entre os dígitos hexadecimais e as sequência de dígitos (números) binários e decimais

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

$$\begin{array}{cccccc} 4 & A & 9 & , & E_{(16)} \\ 0100 & 1010 & 1001 & , & 1110 \end{array}$$

de onde resulta que $4A9,E_{(16)} = 10010101001,111$ (notar que se eliminaram os zeros não significativos nas posições mais e menos signiticativas).

É claro que este método é válido para qualquer conversão em que b_1 e b_2 se relacionam por potências de 2. Por exemplo, para converter um número da base 4 para a base 2 decompomos cada dígito do número na base 4 em dois dígitos binários (porque $4 = 2^2$). Identicamente, para converter um número da base 3 para a base 9 agrupamos cada conjunto de 2 dígitos do número na base 3 para formar um dígito na base 9 (porque $9 = 3^2$).

1.2.4 Truncagens e arredondamentos

Vamos definir em seguida os critérios de arredondamento da parte fraccionária de um número, quando há lugar a tal. Quando não se justificar fazer um arredondamento, truncamos a parte fraccionária.

Em primeiro lugar, nem sempre se torna necessário arredondar a parte fraccionária de um número. Por exemplo, se o número a representar for abstracto, sem conteúdo físico, pode não fazer sentido proceder ao seu arredondamento. Assim, uma representação para o número π com a parte fraccionária reduzida a p dígitos só faz sentido porque não queremos com ela encher várias folhas

de papel (nesses casos, o número p de casas fraccionárias é-nos imposto pela precisão dos cálculos em que π estiver envolvido).

Pelo contrário, se quisermos representar o resultado de uma medição física, por exemplo da corrente eléctrica que, num dado momento, circula por um circuito electrónico de baixa potência, a precisão da medição vem naturalmente limitada pela precisão do miliamperímetro que a mediu. Não faz sentido, nesse caso, "inventar" uma precisão que não existe, inserindo dígitos na parte fraccionária que não possuem conteúdo físico.

Consideremos um exemplo: uma determinada medição de corrente produziu um valor de 13,75 mA, ou seja, um resultado com uma precisão de 1 unidade em 100 (1/100). A conversão deste número decimal no seu equivalente numa base b_2, qualquer que ela seja, não deve "inventar" precisão. Quer isso dizer que não devemos obter um resultado que, após conversão, venha com uma precisão superior a 1/100.

Por exemplo, se convertermos o número $13,75_{(10)}$ para binário obtemos o número $1101,11000\cdots$. Mas qual será a representação binária correcta para 13,75 mA?

Se retivermos um dígito binário na parte fraccionária, obtemos um resultado com uma precisão de 1 unidade em $2^1 = 2$, isto é, de 0,5 (inferior à precisão do número original, que é igual a 1/100).

Se retivermos dois dígitos binários na parte fraccionária, obtemos um resultado com uma precisão de 1 unidade em $2^2 = 4$, isto é, de 0,25 (ainda inferior à precisão do número original). E por aí fora.

Se agora retivermos 6 dígitos binários na parte fraccionária, obtemos um resultado com uma precisão de 1 unidade em $2^6 = 64$, isto é, ainda inferior à precisão do número dado.

Mas se retivermos 7 dígitos binários na parte fraccionária, obtemos um resultado com uma precisão de 1 unidade em $2^7 = 128$, e 1/128 *é uma precisão superior à que nos é dada*. Ou seja, "inventámos" precisão onde ela não existia.

Segue-se que, com este exemplo, devemos apresentar 6 dígitos binários na parte fraccionária, e $13,75_{(10)}$ mA <> $1101,110000_{(2)}$ mA.

Neste exemplo não tivemos que arredondar o resultado (limitámo-nos a truncá-lo), mas há casos em que temos que o fazer. Isso é particularmente evidente quando o processo de conversão gera uma parte fraccionária infinita ou infinita periódica; mas também nos casos em que se obtêm partes fraccionárias finitas pode-se justificar arredondá-las se, por motivos de precisão, as quisermos com menos dígitos.

Considere-se então a representação

$$N_{(b)} <> d_{q-1}\, d_{q-2} \cdots d_1\, d_0\, ,\, d_{-1} \cdots d_{-i}\, d_{-i-1} \cdots ,$$

do número N na base b.

O problema do arredondamento da parte fraccionária de um número consiste em determinar o que fazer ao dígito d_{-i} em função do dígito d_{-i-1}. E essa decisão vai ser diferente consoante as bases forem pares ou ímpares.

Nas bases pares divide-se b por 2, formando-se desta forma dois conjuntos de dígitos que designaremos por parte baixa e por parte alta. Por exemplo, na

base 10 obtemos uma parte baixa com o conjunto de dígitos $\{0, 1, 2, 3, 4\}$ e uma parte alta com o conjunto $\{5, 6, 7, 8, 9\}$. O critério de arredondamento é, então, o seguinte: se o dígito d_{-i-1} estiver contido na parte baixa, d_{-i} não vem alterado; no caso contrário, soma-se uma unidade a d_{-i}.

Naturalmente, nos casos em que d_{-i} não vem alterado há truncagem em vez de arredondamento.

Por exemplo, o número $0,997_{(10)}$ vem arredondado para $1,00_{(10)}$ se quisermos reter duas casas decimais (*notar como a soma de uma unidade ao 9 na casa das centésimas se propagou até às unidades*).

Da mesma forma, o número $111,11110_{(2)}$ vem arredondado para $1000,000_{(2)}$ se quisermos reter três casas na parte fraccionária (a soma de uma unidade ao 1 com peso 2^{-3} propaga-se para a parte inteira).

Nas bases ímpares, ao dividir-se b por dois obtêm-se duas partes com igual número de dígitos, tal como acontece com as bases pares, mas obtém-se ainda um dígito que "fica de fora". Por exemplo, para uma base $b = 5$ obtém-se uma parte baixa com o conjunto de dígitos $\{0, 1\}$ e uma parte alta com o conjunto $\{3, 4\}$, "sobejando" o dígito $\{2\}$.

O que se faz nestes casos é não alterar o dígito d_{-i} se o dígito d_{-i-1} estiver contido na parte baixa, e somar uma unidade a d_{-i} nos outros dois casos, Por exemplo, os números $302,024_{(5)}$ e $302,022_{(5)}$ vêm arredondados para $302,03_{(5)}$ se quisermos reter duas casas na parte fraccionária, enquanto que o número $302,021_{(5)}$ vem truncado para $302,02_{(5)}$ nas mesmas condições.

1.3 Aritmética Binária

1.3.1 Adições na base 2 e noutras bases

Consideremos o seguinte exemplo de adição: $435_{(10)} + 389_{(10)} = 824_{(10)}$. Pretendemos, com o exemplo, deduzir o algoritmo da adição nesta base.

Esquematicamente temos:

	3	2	1	← colunas
	4	3	5	
+	3	8	9	
	8	2	4	

Algoritmo da adição

Transporte na adição

Começa-se pela coluna 1 (da direita). Se o resultado da adição nessa coluna não exceder o valor $b = 10$, o resultado vem colocado nessa coluna, sem alteração. Se o resultado da adição nessa coluna for igual a, ou exceder, o valor $b = 10$, o que se coloca nessa coluna é a *diferença entre o valor produzido e b*, gerando-se, então, um **transporte**, C, para a coluna seguinte (a designação C vem da palavra em inglês "Carry").

Passa-se em seguida à coluna 2, onde se repete a operação anterior, somando ainda o transporte proveniente da coluna 1, se existir. E o mesmo acontece com as restantes colunas.

No nosso exemplo a adição na primeira coluna produziu um transporte,

Tabela 1.3: Tabela da adição no sistema binário

Adição binária		
+	0	1
0	0	1
1	1	10

$$\begin{array}{rr} & \overset{1}{} \quad \leftarrow \text{coluna} \\ & 5 \\ + & 9 \\ \hline 1 & 4 \\ \smile & \\ C{=}1 & \end{array}$$

e na coluna 2 tivemos de adicionar esse transporte aos operandos na coluna, gerando um novo transporte,

$$\begin{array}{rr} & \overset{2}{} \quad \leftarrow \text{coluna} \\ & 1 \\ & 3 \\ + & 8 \\ \hline 1 & 2 \\ \smile & \\ C{=}1 & \end{array}$$

O algoritmo da adição binária é em tudo semelhante ao da adição no sistema decimal (e, de facto, ao da adição em qualquer sistema de numeração posicional), excepto que se utiliza $b = 2$ em vez de $b = 10$.

Começa-se, então, pela coluna da direita, vindo o resultado da adição colocado ainda nessa coluna. Desde que não exceda o valor $b = 2$, como por exemplo em

$$\begin{array}{rr} & 1 \\ + & 0 \\ \hline & 1_{(2)} \end{array}$$

o resultado vem colocado nessa coluna, sem alteração (para efectuar a adição binária, consultar a Tabela 1.3, que contém a operação de adição binária de dois operandos com um bit cada um).

Se o resultado da adição for igual a, ou exceder, o valor $b = 2$, haverá que colocar nessa coluna o valor da diferença entre o valor produzido e b, gerando-se, então, um transporte para a coluna seguinte. Por exemplo, $1_{(2)} + 1_{(2)}$ dá um valor $10_{(2)}$, ou $2_{(10)}$, que é igual a $b = 2$. Então, o que se coloca nessa coluna é a diferença entre o valor $2_{(10)}$ que se produziu e o valor $b = 2$, obtendo-se 0 e um transporte $C = 1$:

$$\begin{array}{rr} & 1 \\ + & 1 \\ \hline 1 & 0_{(2)} \\ \smile & \\ C{=}1 & \end{array}$$

Tabela 1.4: Tabela da adição no sistema hexadecimal

Adição hexadecimal

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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	E	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

O valor do transporte que for gerado é adicionado aos valores da coluna imediatamente à esquerda, e o algoritmo prossegue de forma idêntica para todas as colunas.

Por exemplo, para efectuar a adição $1111_{(2)} + 101_{(2)}$ fazemos

$$\begin{array}{cccccc} & {\scriptstyle 4} & {\scriptstyle 3} & {\scriptstyle 2} & {\scriptstyle 1} & \leftarrow \text{colunas} \\ & 1 & 1 & 1 & 1 & \\ + & & 1 & 0 & 1 & \\ \hline 1 & 0 & 1 & 0 & 0 & \end{array}$$

De notar que

$$\begin{aligned} 1111_{(2)} &<> 15_{(10)} \\ 101_{(2)} &<> 5_{(10)} \\ 10100_{(2)} &<> 20_{(10)}\,. \end{aligned}$$

De notar ainda que na coluna 3 se procedeu à adição de três unidades, gerando uma soma igual a 1 e um transporte $C = 1$.

No sistema hexadecimal utilizamos ainda o mesmo algoritmo da adição. Naturalmente, teremos agora que saber adicionar dois dígitos hexadecimais, para o que utilizamos a Tabela 1.4.

Vamos ver, com alguns exemplos, como se constroi esta tabela.

Por exemplo, a adição $4_{(16)} + 9_{(16)}$ é a mesma que $4_{(10)} + 9_{(10)}$, o que dá $13_{(10)}$. Mas $13_{(10)}$ corresponde a *um único* dígito na base 16, o dígito $\mathrm{D}_{(16)}$, que é menor do que a base $b = 16$. Logo, o resultado da soma $4_{(16)} + 9_{(16)}$ dá $\mathrm{D}_{(16)}$ e não gera transporte.

Notemos como fazemos o raciocínio na base 10, porque estamos habituados a raciocinar nesta base (*mas podíamos raciocinar numa base qualquer*).

Ainda um outro exemplo: $6_{(16)} + \mathrm{D}_{(16)}$. Como $\mathrm{D}_{(16)} = 13_{(10)}$, fazemos:

$$6_{(16)} + \mathrm{D}_{(16)} = 6_{(10)} + 13_{(10)} = 19_{(10)}\,.$$

Como $19_{(10)} > 16_{(10)}$, devemos reter o que resultar da subtracção $19_{(10)} - 16_{(10)} = 3_{(10)} <> 3_{(16)}$ e gerar um transporte. Logo, $6_{(16)} + \mathrm{D}_{(16)} = 13_{(16)}$.

Passemos agora à adição de dois números na base 16, por exemplo $4\mathrm{A}57_{(16)} + D293_{(16)}$. Faz-se:

$$\begin{array}{rccccc} & 4 & \mathrm{A} & 5 & 7 & _{(16)} \\ + & \mathrm{D} & 2 & 9 & 3 & _{(16)} \\ \hline 1 & 1 & \mathrm{C} & \mathrm{E} & \mathrm{A} & _{(16)} \\ & \smile & \smile & \smile & \smile & \\ & 1 & 0 & 0 & 0 & \end{array}$$

1.3.2 Subtracções na base 2 e noutras bases

Algoritmo da subtracção

Aditivo e subtractivo

O algoritmo da subtracção pode resumir-se ao seguinte: se numa coluna i, qualquer, o **aditivo** for maior do que o **subtractivo**, o resultado da subtracção vem na coluna i, sem alteração.

No caso contrário, *soma-se a base ao aditivo e coloca-se na coluna i o resultado da diferença entre este valor e o subtractivo*, gerando-se um **transporte** para a coluna $i + 1$.

Transporte na subtracção

Assim, por exemplo,

$$\begin{array}{rcc} & 2 & 3_{(10)} \\ - & & 8_{(10)} \\ \hline & 1 & 5_{(10)} \end{array}$$

fazendo-se a diferença $(b + 3) - 8 = (10 + 3) - 8 = 5$ na coluna mais à direita e gerando-se um transporte igual a 1 para a coluna imediatamente à esquerda.

De idêntica forma, no sistema binário faz-se, por exemplo,

$$\begin{array}{rcc} & 1 & 0_{(2)} \\ - & & 1_{(2)} \\ \hline & 0 & 1_{(2)} \end{array}$$

ou seja, faz-se a diferença $(b + 0) - 1 = (2 + 0) - 1 = 1$ e gera-se um transporte.

Por exemplo, para efectuar a subtracção $1001101_{(2)} - 10111_{(2)}$ fazemos:

	7	6	5	4	3	2	1	← colunas
	1	0	0	1	1	0	1	
-			1	0	1	1	1	
		1	1	0	1	1	0	
	0	1	1	0	1	1	0	

Neste exemplo não é gerado transporte da coluna 1 para a coluna 2. Porém, na coluna 2 é gerado um transporte ("1 para 2 igual a 1, com transporte igual a 1") para a coluna 3. Esse transporte é adicionado, na coluna 3, ao subtractivo ("$1 + 1 = 2$, isto é, $0_{(2)}$ e transporte 1") que depois é subtraído do aditivo ("0 para 1 igual a 1, mantendo-se o transporte"). De forma idêntica se procederia com as colunas 4 a 7.

Na base 16 temos o mesmo algoritmo. Por exemplo, para efectuar $9F1B_{(16)} - - 4A36_{(16)}$ faz-se:

	4	3	2	1	← colunas
	9	F	1	B $_{(16)}$	
-	4	A	3	6 $_{(16)}$	
	5	4	E	5 $_{(16)}$	
	0	0	1	0	

De notar, na coluna 2, que se faz "$3_{(10)}$ para $17_{(10)}$, o que dá $14_{(10)} <> E_{(16)}$, gerando-se um transporte igual a 1". De notar ainda que, na coluna 3, se faz "$A_{(16)} = 10_{(10)}$ mais o transporte 1 dá $11_{(10)}$; $11_{(10)}$ para $F_{(16)} <> 15_{(10)}$ dá $4_{(10)} <> 4_{(16)}$, gerando-se um transporte nulo".

É de notar que este algoritmo produz um resultado incorrecto quando o subtractivo é maior que o aditivo. Com efeito, a aplicação do algoritmo nestas situações produz um transporte que se vai propagar indefinidamente a partir de uma coluna i e sucessivamente para as colunas $i+1$, $i+2$, etc. como no seguinte exemplo

				1	0	1 $_{(2)}$
		-	1	0	1	1 $_{(2)}$
$\cdots$	1	1	1	0	1	0 $_{(2)}$
	1	1	1	0	1	0

Este problema só vem resolvido quando estudarmos os números com sinal e, em particular, quando se apresentar a representação em complemento para 2, na Subsecção 1.4.2.

1.3.3 Multiplicação na base 2 e noutras bases

O produto de dois números numa base arbitrária vai, mais uma vez, obedecer ao algoritmo da multiplicação. Por exemplo, na base $b = 10$ temos

Tabela 1.5: Tabela da multiplicação no sistema binário

Multiplicação binária		
×	0	1
0	0	0
1	0	1

$$\begin{array}{cccc} & 1 & 5 & 7_{(10)} \\ & \times & 2 & 3_{(10)} \\ \hline & 4 & 7 & 1 \\ 3 & 1 & 4 & \\ \hline 3 & 6 & 1 & 1_{(10)} \end{array}$$

Notemos que agora podemos ter transportes maiores do que a unidade. Por exemplo,

$$\begin{array}{cccc} 1 & 5 & 7_{(10)} \\ & \times & 3_{(10)} \\ \hline 4 & 7 & 1 \\ \overbrace{}^{1} & \overbrace{}^{2} & \end{array}$$

Da mesma forma, na base $b = 2$ podemos multiplicar dois números desde que conheçamos a tabela de multiplicação de dois dígitos nessa base (Tabela 1.5).

Por exemplo,

$$\begin{array}{cccccccc} & & 1 & 0 & 1 & 1 & 0 & 1_{(2)} \\ & & & & \times & 1 & 0 & 1_{(2)} \\ \hline & & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & & \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1_{(2)} \end{array}$$

Notemos como é fácil a multiplicação nesta base. Como cada dígito do multiplicador só pode ser 0 ou 1, a multiplicação desse dígito pelo multiplicando só pode dar o próprio multiplicando ou uma linha com zeros.

Para a multiplicação na base $b = 16$ precisamos de conhecer a respectiva tabela (Tabela 1.6).

Vejamos como deduzir uma posição desta tabela (sabendo-se deduzir uma, pode deduzir-se a tabela toda).

Por exemplo, consideremos o produto $\mathrm{C}_{(16)} \times \mathrm{B}_{(16)} = 84_{(16)}$. A melhor forma de obter este resultado é raciocinar em decimal, como fizemos para as adições e para

Tabela 1.6: Tabela da multiplicação no sistema hexadecimal

Multiplicação hexadecimal

×	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	0	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	0	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	0	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	0	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	0	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	0	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	0	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	0	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	0	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	4	D2
F	0	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

as subtracções. Assim, $C_{(16)} <> 12_{(10)}$ e $B_{(16)} <> 11_{(10)}$. Ora $12_{(10)} \times 11_{(10)} = 132_{(10)}$. Convertendo agora para hexadecimal obtemos

$$132_{(10)} = 128_{(10)} + 4_{(10)} = 8_{(10)} \times 16_{(10)} + 4_{(10)} <> 84_{(16)}\,.$$

Agora, multiplicar dois números hexadecimais é simples. Por exemplo,

$$\begin{array}{cccccccc}
 & & & 5 & C & 2 & A_{\,(16)} \\
 & & \times & 7 & 1 & D & 0_{\,(16)} \\
\hline
 & & 4 & A & E & 2 & 2 & \\
 & & 5 & C & 2 & A & & \\
2 & 8 & 5 & 2 & 6 & & & \\
\hline
2 & 8 & F & 9 & 6 & C & 2 & 0_{\,(16)}
\end{array}$$

Com efeito, reparemos como

$$\begin{array}{cccccc}
 & 5 & C & 2 & A_{\,(16)} \\
\times & & & & D_{\,(16)} \\
\hline
4 & A & E & 2 & 2_{\,(16)} \\
\underbrace{}_{4} & \underbrace{}_{9} & \underbrace{}_{2} & \underbrace{}_{8} &
\end{array}$$

ou como

$$
\begin{array}{rcccccccc}
 & & & 4 & A & E & 2 & 2 & (16) \\
 & & & 5 & C & 2 & A & & \\
+ & 2 & 8 & 5 & 2 & 6 & & & \\
\hline
 & 2 & 8 & F & 9 & 6 & C & 2 & (16) \\
 & \underbrace{} & \underbrace{} & \underbrace{} & \underbrace{} & \underbrace{} & \underbrace{} & \underbrace{} & \\
 & 0 & 0 & 1 & 1 & 0 & 0 & &
\end{array}
$$

1.4 Números com Sinal

1.4.1 Notação de sinal e módulo

A representação de números reais tem de ter em conta que os números podem ser positivos, negativos ou o número 0.

Num sistema digital, um número é sempre representado com um determinado número de bits, o comprimento do número, que corresponde ao número de bits dos registos onde o número é armazenado e ao número de bits dos circuitos que processam os números.

Suponhamos, para simplificar, e a titulo de exemplo, que o comprimento é de 4 bits.

Representação de um número em módulo e sinal

Bit de sinal

Uma primeira ideia consiste em representar o número em **módulo e sinal**. O módulo já não nos oferece dificuldade, mas não é óbvio como podemos representar o sinal. A forma mais evidente é reservar um bit para o **sinal** e definir, por exemplo, que se esse bit for 0 o número é positivo e se for 1 é negativo.

Com quatro bits podemos, então, representar os números inteiros que se indicam na Tabela 1.7.

Esta forma de representar números inteiros é uma das formas possíveis. Tem, porém, alguns inconvenientes.

Em primeiro lugar, repare-se que existem duas representações para o número 0, o que pode originar problemas.

Em segundo lugar, quando é necessario fazer operações sobre os números assim representados, é necessario processar de forma diferente o módulo e o sinal, e é ainda necessário escolher a operação a realizar em termos da operação desejada e do sinal dos números.

Por exemplo, se pretendermos fazer a operação $(+5) + (-3)$, o que é realmente necessário fazer é a subtracção $5 - 3$, ficando o sinal positivo. Se o problema for realizar $(-5) + (+3)$, então há que realizar também uma subtracção mas do módulo do número negativo menos o do positivo, sendo o resultado um número negativo.

Obviamente, tudo isto complica significativamente os circuitos lógicos que realizam a operação de soma (e também os que realizam a subtracção).

1.4.2 Notação de complemento para 2

Seria interessante se pudéssemos representar os números de tal forma que fosse possível realizar as operações de soma e de subtracção sempre da mesma forma,

Tabela 1.7: Números com 4 bits representados em módulo e sinal

Representação	Número representado
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	−0
1001	−1
1010	−2
1011	−3
1100	−4
1101	−5
1110	−6
1111	−7

independentemente de os números serem positivos ou negativos. E isso é possível utilizando a representação em complemento para 2.

Complemento para 2 de um número

Comecemos por perceber o que é o **complemento para 2 de um número**. Ou, mais correctamente, o complemento para 2^n. Designa-se por complemento para 2^n de um número x com n bits o resultado da operação $2^n - x$. Obtemos, desta forma, o simétrico de x.

Por exemplo, o complemento para 2^4 de 0101 é

$$\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ - & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 1 & 1 \end{array}$$

O número 1011 é, então, o complemento para 2^4 ou, mais simplesmente, o complemento para 2 de 0101.

Repare-se que, se o número x tem n bits, o seu complemento para 2 é representavel de igual modo com n bits.

É óbvio que o complemento para 2 do complemento para 2 de um número x é x, já que $2^n - (2^n - x) = x$. Assim, o complemento para 2 de 1011 é, como se pode verificar, 0101.

Uma forma mais expedita de obter o complemento para 2 de um número consiste, como se poderá verificar, em complementar (negar) todos os bits do número e somar-lhe 1. Novamente, tomando como exemplo o número 5 (0101),

pode obter-se o complemento para 2 de 5 negando-o (1010) e somando-lhe 1 (1011), o que vem dar o mesmo resultado que vimos atrás.

Ainda uma terceira forma para obter o complemento para 2 de um número utiliza o seguinte algoritmo: percorre-se a representação do número a complementar, desde o bit menos significativo até ao mais significativo. Por cada 0 que se encontra, reproduz-se esse 0. Quando se encontrar o primeiro 1, também se reproduz. A partir daí, trocam-se os "0"s com os "1"s.

Por exemplo, o complemento para 2 do número negativo 1011101100 é o número positivo 0100010100:

$$\begin{array}{cc} 1011101 & 100 \\ \underbrace{0100010}_{\text{trocam–se}} & \underbrace{100}_{\text{copiam–se}} \end{array}$$

e o complemento para 2 do número positivo 0100010100 é o número negativo 1011101100:

$$\begin{array}{cc} 0100010 & 100 \\ \underbrace{1011101}_{\text{trocam–se}} & \underbrace{100}_{\text{copiam–se}} \end{array}$$

Na notação de complemento para 2, os números são também representados de forma a que um dos bits (o **mais significativo**) represente o sinal. Tal como na representação em módulo e sinal, o bit a 0 indica que o número é positivo, e a 1 indica que o número é negativo. Portanto, com n bits para representar um número, "sobram" $n-1$ bits para além do sinal para representar, de alguma forma, o módulo do número (veremos já adiante que isso não é completamente verdade).

Bit mais significativo

Não confundir a representação de um número em notação de complemento para 2 com a obtenção do complemento para 2 de um número. A representação de +5 em complemento para 2 com 4 bits é 0101; o complemento para 2 de +5 é o seu simétrico, −5, que se representa por 1011 em notação de complemento para 2.

Na **representação em complemento para** 2 de um número positivo x, o número é representado pelo seu módulo. Por exemplo, o número $+5_{(10)}$ é representado, em notação de complemento para 2 com 4 bits, como $0101_{(C2)}$, em que o índice referencia este modo de representação, ou notação (a notação de complemento para 2).

Representação de um número em complemento para 2

Na representação em complemento para 2 de um número negativo, o número x é representado pelo complemento para 2 de x.

Por exemplo o número $-5_{(10)}$ é representado, em notação de complemento para 2, por $1011_{(C2)}$. Repare-se que a determinação do complemento para 2 de um número deixa já o bit de sinal do número simétrico com o valor correcto.

A representação dos 16 números possiveis em complemento para 2 com 4 bits vem ilustrada na Figura 1.1.

Como é natural, o número de números representáveis com n bits é de 2^n. Como se pode ver da figura, o intervalo representado é $[-2^{n-1}, +2^{n-1}-1]$, no nosso caso $[-8, +7]$. A razão da assimetria entre o número de positivos e o de negativos

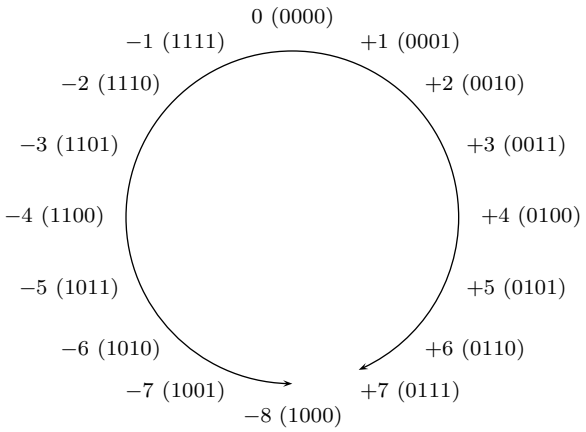


Figura 1.1: Os 16 números possiveis em complemento para 2 com 4 bits

radica na necessidade de representar o 0 e de não ter duas representações para ele.

Com esta representação, *pode operar-se sobre os números sem que o sinal de cada um deles receba qualquer espécie de tratamento particular, ou, se quisermos, o bit de sinal é tratado da mesma forma que todos os outros*.

Adição em complemento para 2

Estudemos alguns exemplos de **adições** nesta notação.

Soma de dois números positivos: $(+2) + (+5) = +7$.

$$\begin{array}{rcccc} & 0 & 0 & 1 & 0_{\,(C2)} \\ + & 0 & 1 & 0 & 1_{\,(C2)} \\ \hline & 0 & 1 & 1 & 1_{\,(C2)} \end{array}$$

Os dois números estão representados em complemento para 2, tal como a sua soma.

Soma de dois números negativos. $(-2) + (-5) = -7$.

A soma realiza-se normalmente somando os números bit a bit.

$$\begin{array}{rcccc} & 1 & 1 & 1 & 0_{\,(C2)} \\ + & 1 & 0 & 1 & 1_{\,(C2)} \\ \hline \not{1} & 1 & 0 & 0 & 1_{\,(C2)} \end{array}$$

O resultado tem um transporte que não consideraremos por sair fora dos n bits (4, neste caso) da representação. O resultado está certo: $-7_{(10)} <> 1001_{\,(C2)}$.

Soma de um número positivo com um número negativo, com resultado positivo. $(+5) + (-3) = +2$.

$$\begin{array}{rcccc} & 0 & 1 & 0 & 1_{\,(C2)} \\ + & 1 & 1 & 0 & 1_{\,(C2)} \\ \hline \not{1} & 0 & 0 & 1 & 0_{\,(C2)} \end{array}$$

Mais uma vez ignoramos o transporte por estar fora da nossa capacidade de representação. O resultado esta certo.

SOMA DE UM NÚMERO POSITIVO COM UM NÚMERO NEGATIVO, COM RESULTADO NEGATIVO. $(+2)+(-5)=-3$.

$$\begin{array}{rcccc} & 0 & 0 & 1 & 0_{\text{ (C2)}} \\ + & 1 & 0 & 1 & 1_{\text{ (C2)}} \\ \hline & 1 & 1 & 0 & 1_{\text{ (C2)}} \end{array}$$

Como se pode ver, o resultado está correcto: $-3_{(10)} <> 1101_{\text{ (C2)}}$.

Como é evidente, se somarmos dois números do mesmo sinal pode ocorrer um resultado que não é representavel com o número de bits disponível. Por exemplo, se somarmos $(+4)+(+5)=+9$, o número $+9$ não é representavel com 4 bits em notação de complemento para 2. Por isso, o resultado é incoerente:

$$\begin{array}{rcccc} & 0 & 1 & 0 & 0_{\text{ (C2)}} \\ + & 0 & 1 & 0 & 1_{\text{ (C2)}} \\ \hline & 1 & 0 & 0 & 1_{\text{ (C2)}} \ ? \end{array}$$

De acordo com a nossa metodologia, a soma daria -7 o que é, como se disse, incoerente. Houve **"overflow"** na adição em complemento para 2. Sempre que somarmos dois números positivos e o resultado pareça ser um número negativo, ou inversamente, estamos perante uma situação de "overflow" na adição em complemento para 2.

"Overflow" na adição em complemento para 2

De notar que a existência de um transporte para o exterior numa adição de números sem sinal é condição suficiente para concluirmos ter havido "overflow" na adição.

Repare-se na Figura 1.2 o significado de uma soma. A nossa soma anterior de $+2$ com $+5$ significa a rotação no anel representado na Figura 1.1, de $+5$ posições (no sentido dos ponteiros do relógio) a partir da posição $+2$.

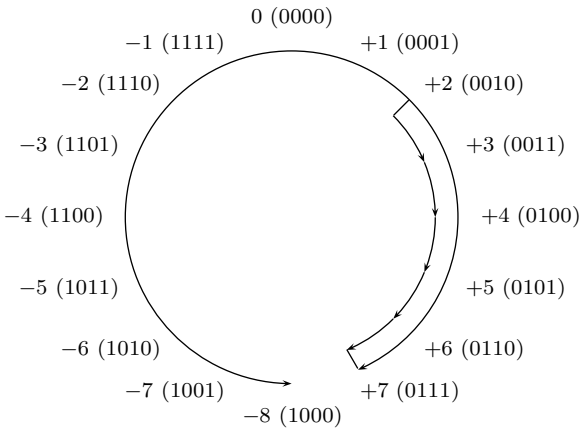


Figura 1.2: Representação gráfica da soma $(+2)+(+5)=+7$, com números de 4 bits representados em notação de complemento para 2

Vejamos agora o que acontece com a soma de $(+4)+(+5)$ na Figura 1.3: parte-se do $+4$ e roda-se $+5$ posições no sentido do ponteiro do relógio.

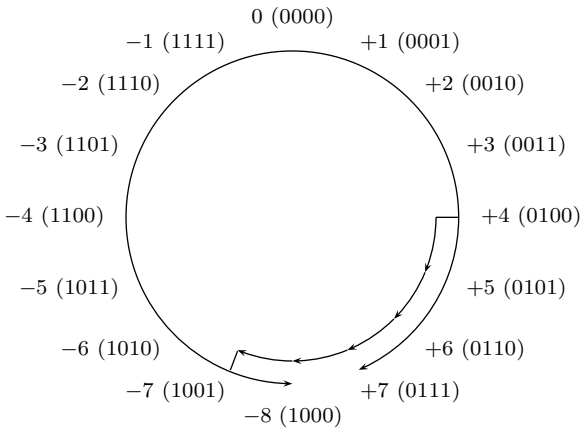


Figura 1.3: Representação gráfica da soma $(+4)+(+5) = -7$ (?), com números de 4 bits representados em notação de complemento para 2. Esta operação produziu "overflow"

Repare-se que agora ultrapassámos o limite de representação da notação de complemento para 2 com 4 bits, e "entrámos pelo outro lado". E passamos, portanto, a números com o sinal contrário, o que significa que houve "overflow"..

Como é evidente, nunca ocorre "overflow" somando números de sinais contrários.

Pode-se provar que o "overflow" ocorre sempre que o transporte do último bit é diferente do transporte do bit anterior, como acontece em:

$$\begin{array}{rcccc} & 0 & 1 & 0 & 0_{\text{(C2)}} \\ + & 0 & 1 & 0 & 1_{\text{(C2)}} \\ \hline & 1 & 0 & 0 & 1_{\text{(C2)}}\ ? \\ & \overset{0}{\frown} & \overset{1}{\frown} & & \end{array}$$

Subtracção em complemento para 2

A utilização de números representados em complemento para 2 permite realizar **subtracções** de forma muito simples. Como se sabe, *realizar a subtracção $x - y$ é o mesmo que realizar a soma $x + (-y)$. Por outro lado, dado um número y representado em notação de complemento para 2, obter o seu simétrico, isto é, o número $-y$, na mesma notação, significa, na prática, obter o complemento para 2 de y.*

Ou seja, podemos utilizar os mesmos circuitos para fazer somas e subtracções em complemento para 2, ao contrário do que sucede se os números estiverem representados em sinal e módulo. Com efeito, o complemento para 2 de um número representado em complemento para 2 pode ser efectuado trocando os "0"s com os "1"s — o que é fácil de fazer num circuito apropriado, como veremos à frente neste curso — e adicionando 1, o que pode ser feito com o somador que, de qualquer forma, é necessário para fazer as somas.

Do ponto de vista gráfico, a subtracção $x - y$ pode ser realizada rodando, no anel da Figura 1.1 a partir do número x, de y posições no sentido contrário ao dos ponteiros do relógio. Por exemplo, na Figura 1.4 está ilustrada a subtracção $(+2) - (+5) = -3$.

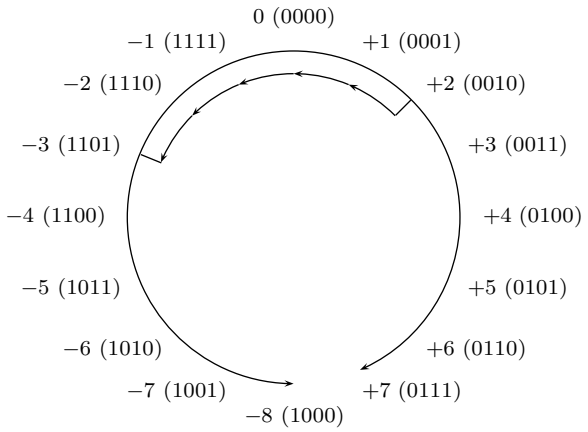


Figura 1.4: Representação gráfica da subtracção $(+2) - (+5) = -3$, com números de 4 bits representados em notação de complemento para 2

Esta subtracção não produziu **"overflow"** porque não ultrapassámos o limite de representação da notação de complemento para 2 com 4 bits (não "entrámos pelo outro lado", gerando números com o sinal contrário).

"Overflow" na subtracção em complemento para 2

1.5 Referências Bibliográficas

Sêrro, Carlos — *Sistemas Digitais: Fundamentos Algébricos*, IST Press, Lisboa, 2003, Secções 1.1, 1.2, e 1.4 a 1.7.

Arroz, G. S., Monteiro, J. C. e Oliveira, A. L. — *Introdução aos Sistemas Digitais e Microprocessadores*, Secções 2.1 e 2.2.

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 1.1 a 1.3.

1.6 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 1.1 Escrever os seguintes números em forma polinomial:

a) $23_{(10)}$; b) $4\,087_{(10)}$; c) $39,28_{(10)}$;
d) $36_{(8)}$; e) $\mathrm{E5},3_{(16)}$; f) $255,6_{(7)}$;
g) $1\,023,003_{(4)}$.

1.2 Passar para a base 10 os seguintes números:

a) $437_{(8)}$; b) $325_{(6)}$; c) $0,245_{(8)}$;
d) $0,46_{(7)}$; e) $10101,100101_{(2)}$; f) $\mathrm{A2D},9\mathrm{A}_{(16)}$;
g) $\mathrm{a32b},5\mathrm{a}_{(12)}$.

1.3 Escreva as representações binária e hexadecimal de:

a) $25,25_{(10)}$;
b) $212,5_{(10)}$;
c) $4,9875_{(10)}$.

(∗) 1.4 Determinar as bases b e c em:

a) $5\mathrm{A}_{(16)} = 132_{(b)}$;
b) $20_{(10)} = 110_{(c)}$.

1.5 Passe para as bases 4, 8 e 16 os seguintes números:

a) $1101101,1001101_{(2)}$;
b) $10111110,00001111_{(2)}$;
c) $111010,01111_{(2)}$.

1.6 Passe para a base 2 os seguintes números:

a) $2031,123_{(4)}$;
b) $432,56_{(8)}$;
c) $\mathrm{EA2,F5}_{(16)}$.

1.7 Passe para a base 3 os seguintes números:

a) $585_{(9)}$;
b) $467,3_{(9)}$.

1.8 Converter o número $257_{(10)}$ para as bases 8 e 16, directamente e através da base 2.

1.9 Escrever as potências de 2 desde 2^{-4} até 2^{15}.

(∗) 1.10 O resultado da leitura do valor de uma tensão eléctrica é de 25,76 V. Representar em binário esse valor.

1.11 Em que bases pode estar escrito o número $\mathrm{3A2,B7}_{(b)}$?

1.12 Escrever a tabela de multiplicação na base 5 e utilizá-la para calcular directamente $34_{(5)} \times 23_{(5)}$. Não utilizar a base 10 como intermediária.

(∗) 1.13 A primeira expedição a Marte provou a existência de civilizações inteligentes no planeta vermelho porque descobriu, gravada numa rocha, a equação

$$5\,x^2 - 50\,x + 125 = 0\,,$$

bem como as respectivas soluções, $x_1 = 5$ e $x_2 = 8$. O valor $x_1 = 5$ pareceu razoável aos elementos da expedição, mas a outra solução indicava claramente que os marcianos não utilizavam, como nós, o sistema decimal de contagem (talvez porque não possuíssem 10 dedos nas mãos). Quantos dedos acha que os marcianos tinham nas mãos? Justifique.

(∗) 1.14 Como sabe do exercício anterior, a primeira expedição a Marte provou a existência de antigas civilizações inteligentes no planeta vermelho. Uma das descobertas mais importantes consistiu em perceber que os marcianos usavam um sistema de numeração com 13 símbolos, incluindo os símbolos 0 a 9, tal como nós usamos na Terra, e ainda os símbolos, ©, ◁ e $\check{L}$. Por outro lado, conseguiu-se provar que os marcianos conheciam as operações aritméticas de adição e de subtracção. Tendo a expedição terrestre encontrado o seguinte fragmento de uma operação de adição gravada numa rocha,

$$\begin{array}{rcccccc} & \check{L} & \lhd & 9 & 3 & & 5 \\ + & & & 9 & \check{L} & & 4\ \textcircled{c} \\ \hline & \textcircled{c} & 9 & 6 & 4 & 2 & 3 \end{array}$$

decidiu enviar esse fragmento para a Terra para ser decifrado (os espaços em branco correspondem a símbolos que não se conseguiram ler). Refaça a adição preenchendo os fragmentos da operação que não puderam ser recuperados pela expedição terrestre, e diga quais os valores que descobriu para os símbolos ⓒ, $\lhd$ e $\check{L}$.

1.15 Representar, em notação de sinal e módulo com 6 bits, os números decimais $+24$, -24, $+57,75$ e $-3,625$.

1.16 Qual é a capacidade de representação da notação de sinal e módulo (o intervalo dentro do qual é possível representar, nesta notação, um número com n bits)?

1.17 Obter o complemento para 2 dos números binários 00001110101 e 1111111101, representados em notação de complemento para 2.

1.18 Qual é o número decimal que é equivalente aos números binários 00001110101 e 1111111101, representados em notação de complemento para 2?

1.19 Representar, em notação de complemento para 2 com 10 bits, os números decimais $+65$ e -5.

(∗) 1.20 Representar os números decimais $+5$, -5, $+54$ e -54 em notação de complemento para 2 com:

a) 4 bits; b) 5 bits; c) 6 bits; d) 7 bits;
e) 10 bits; f) 15 bits.

1.21 Realizar, em notação de complemento para 2 com 10 bits, as seguintes somas decimais:

a) $(+65)+(+5)$;
b) $(+65)+(-5)$;
c) $(-65)+(+5)$;
d) $(-65)+(-5)$.

1.22 A soma decimal $(+30)+(+5)$, realizada em notação de complemento para 2 com 6 bits, produz "overflow"? E a soma $(-17)+(-21)$, realizada nas mesmas circunstâncias? Justifique.

(∗) 1.23 Provar que a subtracção

$$110010_{\,(\mathrm{C2})} - 110110_{\,(\mathrm{C2})}$$

de dois números representados em notação de complemento para 2 pode ser substituída pela soma

$$110010_{\,(\mathrm{C2})} + 001010_{\,(\mathrm{C2})}$$

do aditivo com o complemento para 2 do subtractivo.

1.24 A subtracção decimal $(-30)-(+5)$, realizada em notação de complemento para 2 com 6 bits, produz "overflow"? E a subtracção $(+17)-(-21)$, realizada nas mesmas circunstâncias? Justifique.

Capítulo 2

Códigos

2.1 Conceito de Código

Quando queremos representar informação binária só podemos recorrer aos símbolos (bits) 0 e 1, como sabemos. Mas representar informação numérica em binário é apenas uma parte da realidade, sendo que também é necessário arranjar um meio de representar informação não numérica, como seja informação alfabética (letras maiúsculas e minúsculas, acentuadas ou não, símbolos de pontuação, símbolos de controlo, etc.).

Tal faz-se recorrendo a **códigos binários**, que mais não são do que maneiras de representar com "0"s e "1"s toda a informação que se enumerou acima. Para tal, estabelecem-se *palavras* do código binário com um número adequado de bits e em número suficiente para representar toda a informação que queremos.

Código binário

Naturalmente, desta forma podemos estabelecer um elevadíssimo número de códigos binários, e nem todos apresentam o mesmo interesse. Assim sendo, faz sentido começarmos por estabelecer uma taxonomia que tente agrupar os códigos mais interessantes em grupos distintos. Existem, assim, duas grandes classes de códigos, a dos códigos numéricos e a dos códigos alfanuméricos, que estudaremos mais à frente.

2.2 Códigos Numéricos

Um exemplo ajudar-nos-á a compreender alguns conceitos ligados aos códigos em geral, e aos códigos numéricos em particular.

Suponhamos que queríamos desenvolver um sistema digital para controlar o elevador de um prédio com r/c, duas caves e três andares. Como temos 6 possibilidades distintas de representar e individualizar os 6 pisos do prédio, precisamos de começar por escrever um código numérico com 6 **palavras** distintas, uma por cada piso a que o elevador tem acesso. Ou seja, vamos estabelecer uma correspondência biunívoca entre cada um dos pisos e um conjunto de bits por piso, diferente para cada um deles, e que constitui uma palavra do código.

Palavra

Naturalmente, a primeira pergunta que nos ocorre é sobre o **comprimento** das

Comprimento de uma palavra

palavras, isto é, o número de bits por palavra. Embora teoricamente cada palavra possa ter um comprimento diferente do de todas as outras, vamos limitar-nos apenas aos chamados **códigos regulares**, em que todas as palavras do código têm o mesmo comprimento.

Código regular

No caso do elevador, as palavras do código têm que ter um comprimento adequado. Como temos 6 possibilidades distintas, bastam-nos 3 bits para comprimento de cada uma das palavras. É claro que, nestas condições, apenas vamos utilizar 6 das 8 combinações possíveis. A única restrição que necessitamos de ter presente é que não devemos codificar dois pisos com a mesma palavra, naturalmente para evitar confusões.

Deste modo, já estabelecemos o número de palavras (6) e o comprimento de cada palavra (3) para o nosso sistema de controlo do elevador. E estamos agora em posição de escolher, de entre os $C_6^8 = 28$ códigos possíveis, um que sirva os nossos propósitos. Por exemplo, o código

$$\begin{aligned} 2\underline{^{a}} \text{ cave} &\rightarrow 000 \\ 1\underline{^{a}} \text{ cave} &\rightarrow 001 \\ \text{r/c} &\rightarrow 010 \\ 1\underline{^{o}} \text{ andar} &\rightarrow 011 \\ 2\underline{^{o}} \text{ andar} &\rightarrow 100 \\ 3\underline{^{o}} \text{ andar} &\rightarrow 101 \end{aligned}$$

serve. Reparemos que o formámos como se estivessemos a escrever números com 3 bits cujos equivalentes decimais fossem desde 0 até 5. Geramos, desta forma, um *código binário natural* ou CBN, com palavras de comprimento 3.

Mas podíamos escolher outro código, por exemplo,

$$\begin{aligned} 2\underline{^{a}} \text{ cave} &\rightarrow 000 \\ 1\underline{^{a}} \text{ cave} &\rightarrow 001 \\ \text{r/c} &\rightarrow 011 \\ 1\underline{^{o}} \text{ andar} &\rightarrow 010 \\ 2\underline{^{o}} \text{ andar} &\rightarrow 110 \\ 3\underline{^{o}} \text{ andar} &\rightarrow 111 \end{aligned}$$

Agora temos um código em que palavras consecutivas apenas diferem num bit. Dizemos, neste caso, que estamos em presença de um *código binário reflectido* ou CBR, com palavras de comprimento 3 (neste caso um CBR incompleto, como veremos mais à frente).

Nestes dois códigos escolhemos palavras com um *comprimento mínimo* de 3 bits. Porém, ninguém nos obriga a escolher o comprimento mínimo: podemos usar comprimentos maiores. Escolhamos um comprimento de 4 bits, por exemplo, e

o código

$$\begin{aligned} 2^{\underline{a}} \text{ cave} &\rightarrow 0000 \\ 1^{\underline{a}} \text{ cave} &\rightarrow 0001 \\ \text{r/c} &\rightarrow 0010 \\ 1^{\underline{o}} \text{ andar} &\rightarrow 0011 \\ 2^{\underline{o}} \text{ andar} &\rightarrow 0100 \\ 3^{\underline{o}} \text{ andar} &\rightarrow 0101 \end{aligned}$$

Mais uma vez, estamos a gerar um código binário natural, desta vez com palavras de comprimento 4

Dado que estamos a usar, neste caso, palavras com um comprimento maior do que o estritamente necessário, o código diz-se **redundante**.

Código redundante

A redundância confere ao código alguma capacidade de **detecção de erros** e, eventualmente, também de **correcção de erros**. Por exemplo, se alguma vez constatarmos que o elevador se encontra num piso codificado com 1111, que não é palavra do código, claramente existe um erro, que pode em seguida ser ou não corrigido consoante a capacidade de correcção de erros que conferirmos ao sistema que processa as palavras.

Detecção e correcção de erros

2.3 Código Binário Natural (CBN)

Como acabámos de ver, o **código binário natural** ou CBN é formado por palavras de comprimento fixo (trata-se de um código regular). Se o comprimento de cada palavra for igual a n, o número máximo de palavras do código é igual a 2^n.

Código binário natural (CBN)

O CBN é gerado formamdo os números na base 2 que têm por equivalentes decimais os números $0_{(10)}$ a $(2^n - 1)_{(10)}$. Por exemplo, o CBN com palavras de comprimento 5 possui 32 palavras, com equivalentes decimais que vão de $0_{(10)}$ até $31_{(10)}$ (Tabela 2.1).

2.4 Código Binário Reflectido (CBR)

Existem inúmeros exemplos de códigos binários reflectidos. Neste texto apenas iremos mencionar um deles, porventura o mais conhecido, que passaremos a designar, mais simplesmente, por **código binário reflectido**, ou CBR.

Código binário reflectido (CBR)

Trata-se de um código que não é ponderado na medida em que, ao contrário do que sucede com o CBN, não podemos atribuir pesos às posições dos bits nas palavras. Por outras palavras, não podemos associar a cada palavra do CBR um equivalente decimal.

A característica essencial do CBR é que certos pares de palavras apenas diferem num bit. Por isso essas palavras designam-se por **adjacentes**. Como veremos mais tarde, quando no Capítulo 5 estudarmos a minimização das funções booleanas simples, esta característica vai ser muito útil.

Palavras adjacentes

Na Tabela 2.2 representa-se o CBR com palavras de comprimento 4.

Tabela 2.1: O código binário natural (CBN) com palavras de comprimento 5

Equivalente decimal	Palavra do CBN	Equivalente decimal	Palavra do CBN
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 2.2: O código binário reflectido (CBR) com palavras de comprimento 4

Linha número	Palavra do CBR
1	0000
2	0001
3	0011
4	0010
5	0110
6	0111
7	0101
8	0100
9	1100
10	1101
11	1111
12	1110
13	1010
14	1011
15	1001
16	1000

Como podemos constatar, a primeira e a segunda linhas representam palavras adjacentes porque apenas diferem num bit. Outro tanto sucede com a segunda e a terceira linhas, a terceira e a quarta, etc.

Na realidade, para além destes pares de linhas contíguas, O código CBR possui ainda outros pares de linhas adjacentes como, por exemplo, as linhas 1 e 4. Por outro lado, a primeira e a última linhas também são adjacentes, o que resulta de o código ser completo, isto é, formado por todas as 2^n *configurações binárias possíveis com palavras de comprimento* n. Como iremos ver mais para a frente

neste curso, esta propriedade de adjacência entre vários pares de linhas vai ser muito importante.

2.4.1 Construção do CBR a partir do CBN

É fácil construit um CBR a partir de um CBN com palavras com o mesmo comprimento. Existindo mais do que uma maneira de o fazer, indicaremos de seguida a mais expedita (construção recursiva).

Comece-se por considerar o CBR com palavras com comprimento $n = 1$ bit:

$$\begin{array}{c} 0 \\ 1 \end{array}$$

Para formar um CBR com palavras de n bits parte-se do CBR com palavras de $n-1$ bits, repetindo em seguida cada uma das suas palavras por ordem inversa (como se estivessem reflectidas num espelho, daí a designação dada ao código). Em seguida junta-se-lhe o n-ésimo bit igual a 0 nas primeiras 2^{n-1} posições, e igual a 1 nas 2^{n-1} posições seguintes.

Por exemplo, a partir do CBR anterior com palavras de comprimento $n = 1$ fazemos

$$\begin{array}{cc} 0 & 0 \\ 0 & 1 \\ \hline 1 & 1 \\ 1 & 0 \end{array}$$

para obter o CBR com palavras de comprimento 2. E a partir deste, fazemos

$$\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 1 \\ \hline 0 & 1 & 1 \\ 0 & 1 & 0 \\ \hline 1 & 1 & 0 \\ 1 & 1 & 1 \\ \hline 1 & 0 & 1 \\ 1 & 0 & 0 \end{array}$$

para obter o CBR com palavras de comprimento 3. E o mesmo para maiores dimensões das palavras.

2.5 Código BCD

2.5.1 Representação de números em BCD

Uma situação muito frequente é a da necessidade de codificar numericamente dez quantidades distintas, correspondentes aos dígitos do sistema decimal, $0_{(10)}$

a $9_{(10)}$. Tal sucede, por exemplo, em máquinas de calcular que utilizam o sistema decimal para a entrada de dados e para a saída dos resultados.

Naturalmente, podemos utilizar as 10 primeiras palavras de comprimento 4 do CBN que passará, nestas circunstâncias, a ficar redundante (na medida em que apenas utilizamos 10 das 16 palavras desse código). Obtemos, nessas condições, o **código BCD** (do inglês "Binary Coded Decimal", ou Decimal Codificado em Binário) da Tabela 2.3.

Código BCD

Tabela 2.3: O código BCD

Dígito decimal	Palavra do código BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Em alternativa, utilizamos um dos muitos **códigos decimais-binários** existentes.

Códigos decimais-binários

Para distinguir entre uma palavra do código BCD e a representação do número binário com a mesma configuração, utilizamos a palavra BCD em índice quando nos referimos ao código. Por exemplo, $0111_{(2)}$ identifica, como sabemos, um número binário que tem $7_{(10)}$ como equivalente decimal. Este mesmo dígito decimal vem representado, no código BCD, pela palavra $0111_{(\mathrm{BCD})}$.

Devemos notar que uma palavra do código BCD codifica *um e só um* dígito decimal. Se quisermos codificar k dígitos decimais (num número decimal com parte inteira formada por um dígito das unidades, um dígito das dezenas, etc. e, eventualmente, com parte fraccionária com um dígito para as décimas, outro para as centésimas, etc.) precisaremos de k palavras do código BCD, uma por cada dígito decimal.

Por exemplo, codificamos o número $37,5_{(10)}$ por

$$37,5_{(10)} <> 0011\ 0111\,,\ 0101_{(\mathrm{BCD})}\,,$$

(separaram-se as palavras do código BCD apenas para ajudar à compreensão da sequência de bits). Isto é completamente distinto de

$$37,5_{(10)} <> 100101,1_{(2)}\,,$$

a representação do número decimal na base 2.

Inversamente, uma sequência arbitrária de bits significa números completamente diferentes consoante ela for interpretada como um número BCD ou como um número binário. Por exemplo,

$$10011001_{(2)} <> 99_{(\mathrm{BCD})}\,,$$

mas

$$10011001_{(2)} <> 2^7 + 2^4 + 2^3 + 2^0 = 155_{(10)}$$

e

$$10011001_{(2)} <> -103_{(\mathrm{C2})}\,.$$

Devemos notar, finalmente, que nem todas as sequências de bits podem representar codificações em BCD de números decimais (não esquecer que há sequências binárias de 4 bits que não constituem palavras do código BCD). Por exemplo, a sequência $00101101_{(2)}$, quando separada para identificar um possível número BCD, como em $0010\ 1101_{(2)}$, mostra a existência de uma palavra BCD e de uma outra que não é BCD.

2.5.2 Adição em BCD

Como referimos anteriormente, em certos sistemas os dados de entrada e os resultados vêm apresentados no código BCD. Nesses casos, e por razões de eficiência, devemos procurar realizar a operação de adição directamente no código BCD (já que as restantes operações aritméticas básicas podem sempre ser reduzidas a adições). Com efeito, a alternativa, mais complexa, consistiria em começar por converter do código BCD para o CBN os dados de entrada, realizar as operações aritméticas no CBN e, finalmente, converter os resultados do CBN para o código BCD.

Adição BCD

Quando se efectua, no código BCD, a **adição** de dois dígitos, três casos diferentes podem ocorrer:

1 o resultado da adição, A, é inferior ou igual a $9_{(10)}$ e representa uma palavra legítima do código;

2 A está compreendido entre $10_{(10)}$ e $15_{(10)}$, e o resultado obtido na adição não constitui uma palavra do código (porque corresponde a uma sequência binária entre 1010 e 1111) sendo, por isso, incorrecto;

3 A tem um valor compreendido entre $16_{(10)}$ e $18_{(10)}$ (é de notar que a adição em questão apenas envolve operandos com um dígito decimal, isto é, entre $0_{(10)}$ e $9_{(10)}$, pelo que o resultado da operação não pode ultrapassar $18_{(10)}$); neste caso, como iremos ver de seguida, a sequência binária do resultado corresponde a uma palavra do código BCD (porém corresponde a uma palavra errada) e gera-se um transporte para além do bit de maior peso da palavra de código; também neste caso o resultado obtido é, obviamente, incorrecto.

Em resumo, se tentarmos efectuar a adição de duas palavras do código BCD com o intuito de adicionar os seus equivalentes decimais, apenas no caso em que A é inferior ou igual a $9_{(10)}$ é que o resultado da adição vem correcto. Nos outros dois casos haverá que corrigir o resultado, sendo que o **algoritmo de correcção** é sempre o mesmo, como iremos ver já de seguida: adição de seis unidades a A.

Algoritmo de correcção da adição em BCD

Na Figura 2.1 ilustram-se os três casos possíveis de adição de dois dígitos no código BCD.

Caso 1	3		0011	
	+5		+0101	
	8		1000	$A = 8_{(10)}$
Caso 2	6		0110	
	+7		+0111	
	13		1101	← erro
			+0110	← correcção
		0001	0011	$A = 13_{(10)}$
Caso 3	8		1000	
	+9		+1001	
	17	1	0001	← erro
			+0110	← correcção
		0001	0111	$A = 17_{(10)}$

Figura 2.1: Casos possíveis de adição de dois dígitos no código BCD: (a) o resultado A da adição vem compreendido entre $0_{(10)}$ e $9_{(10)}$ e está correcto; (b) o resultado vem compreendido entre $10_{(10)}$ e $15_{(10)}$ e precisa de correcção; e (c) o resultado vem compreendido entre $16_{(10)}$ e $18_{(10)}$ e também necessita de ser corrigido

2.6 Os Códigos m–em–n

Os **códigos m-em-n** são códigos decimais binários com m uns e $(n-m)$ zeros em cada palavra de comprimento n.

Nesta classe destacam-se os **códigos 1-em-n**, de que se apresenta um exemplo, o **código 1-em-10**, na Tabela 2.4. Trata-se de um código ponderado, com pesos $9, 8, \ldots, 1, 0$.

Tabela 2.4: O código 1-em-10

Equivalente decimal	Código 1-em-10 9876543210
0	0000000001
1	0000000010
2	0000000100
3	0000001000
4	0000010000
5	0000100000
6	0001000000
7	0010000000
8	0100000000
9	1000000000

2.7 Códigos Alfanuméricos

Como se afirmou no fim da Secção 2.1, existem duas grandes classes de códigos: a dos códigos numéricos, de que estudámos anteriormente os principais exemplos, e a dos códigos alfanuméricos, que estudaremos a seguir.

Trata-se de códigos que têm por objectivo codificar, para além de informação numérica, também informação alfabética, como sejam as letras maiúsculas e minúsculas, os símbolos de pontuação, as letras acentuadas características dos alfabetos latinos, os símbolos utilizados nas línguas orientais, etc, e ainda símbolos de controlo.

2.7.1 O código ASCII

Código ASCII

Na Tabela 2.5 representa-se o **código ASCII**. Trata-se de um código alfanumérico limitado, apenas capaz de codificar a principal informação alfabética (letras maiúsculas e minúsculas, e símbolos de pontuação) e informação de controlo (por exemplo mudança de linha, fim de ficheiro, tabulação, etc.), que utiliza palavras de comprimento 7.

Tabela 2.5: O código ASCII

	$b_6b_5b_4$							
$b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	’	p
0001	SOH	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	_	0	DEL

As duas primeiras colunas da tabela, que usam os dois bits com maior peso iguais a 00, identificam os caracteres de controlo. Por exemplo, SP manda um espaço para um dispositivo de saída (por exemplo, um écran de computador). BEL, por seu turno, faz soar uma campainha. CR significa “Carriage Return”, o que faz o cursor (no mesmo écran) mudar para a linha seguinte, na mesma coluna, enquanto que LF significa “Line Feed”, o que faz o mesmo cursor mudar para o início (primeira coluna) da linha corrente (naturalmente, CR+LF ou

LF+CR, pela ordem indicada, faz mudar o cursor para a primeira coluna da linha seguinte).

Notar que esta forma simplificada de representação substitui uma palavra ASCII, com 7 bits, por dois dígitos hexadecimais, com 8 bits.

Normalmente usa-se o código hexadecimal para simplificar a representação das palavras ASCII. Assim, por exemplo, a codificação ASCII da palavra SIM, isto é, 1010011 1001001 1001101 (separaram-se as palavras apenas para ajudar à compreensão da sequência de bits) vem simplificada para $53\,49\,4B_{(16)}$ ou, ainda mais simplesmente, por 53494B h ou 53494B H, em que "h" ou "H" são abreviaturas de "hexadecimal".

2.7.2 O código ISO-8859-1

Uma limitação séria do código ASCII resulta de ele ter sido desenhado para codificar informação alfabética na língua inglesa, que não contém símbolos de acentuação (como o "ç" português ou o ö germânico). Por outro lado, não é capaz de codificar os símbolos utilizados nas línguas orientais. Claramente, com 7 bits era impossível fazer melhor.

A primeira opção de extensão do código ASCII consistiu em aumentar o número de bits por palavra para 8, mantendo os 7 bits menos significativos iguais aos do ASCII. Ou seja, para os valores 00 h a 7F h os códigos coincidem. Infelizmente, foram criados vários códigos alfanuméricos com esses pressupostos, códigos esses que não são, contudo, compatíveis entre eles.

Código ISO-8859-1 ou Isolatin-1

Destaca-se, de entre eles, o **código ISO-8859-1**, vulgarmente conhecido por **código Isolatin-1**, que permite os caracteres acentuados das línguas latinas da Europa Ocidental, por exemplo o "ç" ou o "á" português, o "ñ" espanhol ou ainda o "ü" alemão.

Código UNICODE

A inclusão de caracteres de outros alfabetos (grego, cirílico, arménio, hebreu, árabe, indiano, etc.), de símbolos matemáticos e de figuras geométricas, e ainda de dezenas de milhar de caracteres ideográficos, como os utilizados em chinês, levou ao aparecimento do **UNICODE** ou ISO/IEC 10646 UCS-2 (Universal Character Set-2), um código evolutivo com 16 bits por símbolo, aberto à inclusão de novos caracteres e símbolos.

2.8 Referências Bibliográficas

Sêrro, Carlos — *Sistemas Digitais: Fundamentos Algébricos*, IST Press, Lisboa, 2003, Secções 2.1, e 2.3 a 2.5.

Arroz, G. S., Monteiro, J. C. e Oliveira, A. L. — *Introdução aos Sistemas Digitais e Microprocessadores*, Secções 2.3 e 2.4.

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 1.4 e 1.5.

2.9 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 2.1 Utilizar o CBN para codificar a seguinte informação decimal:

(a) $N = 31$; (b) $N = 1\ 674$; (c) $N = 52\ 674$.

(∗) 2.2 Construir um CBR com palavras de comprimento 5.

(∗) 2.3 Construir um **código reflectido** de valência 3 e com palavras de comprimento 3. Determinar as adjacências entre palavras do código (*Nota*. Por **valência** de um código entende-se o número de símbolos por ele utilizados. Assim, um código binário é um código de valência 2, e um código ternário tem valência 3).

Código reflectido

Valência

(∗) 2.4 Construir um código reflectido de valência 4 e com palavras de comprimento 2.

(∗) 2.5 Codificar os dígitos decimais $0, 1, \ldots, 9$ utilizando códigos binários ponderados com os pesos indicados:

(a) pesos $6\ \ 3\ \ 2\ \ -1$; (b) pesos $7\ \ 3\ \ 2\ \ -1$;
(c) pesos $7\ \ 3\ \ 1\ \ -2$; (d) pesos $5\ \ 4\ \ -2\ \ -1$;
(e) pesos $8\ \ 7\ \ -4\ \ -2$.

2.6 Considere o número octal $352,4_{(8)}$. Represente-o em decimal, em binário e no código BCD.

2.7 Realizar as operações indicadas no código BCD:

(a) $37_{(10)} + 12_{(10)}$; (b) $1024_{(10)} + 379_{(10)}$;
(c) $37_{(10)} - 12_{(10)}$; (d) $1024_{(10)} - 379_{(10)}$.

2.8 Converter para o código BCD os números decimais dados e, em seguida, executar as operações pedidas.

(a) $12,5 + 21$; (b) $123,1 - 21,5$; (c) $7,5 + 9,81$;
(d) $3,5 - 0,71$.

2.9 Descodificar a seguinte informação ASCII:

1000010 1001111 1010010 1001001 1001110 1000111 0100001 .

Capítulo 3

Álgebra de Boole Binária

3.1 Variáveis e Funções Booleanas

Sistemas e circuitos digitais

Os **sistemas digitais** assentam em circuitos (os **circuitos digitais**) que assumem, em cada instante, um de dois únicos estados possíveis. Se os designarmos por 0 e por 1 (veremos mais tarde que haveremos de preferir outras designações para os estados, porém a essência do problema mantém-se, independentemente das designações), podemos utilizar o sistema de numeração binário como suporte algébrico, e os códigos binários como suporte da informação processada e veiculada por esses circuitos e sistemas.

Álgebra de Boole binária

Do ponto de vista algébrico teremos, naturalmente, que recorrer a uma álgebra especial, designada por **álgebra de Boole binária**, que faça uso dos símbolos 0 e 1 e que os utilize de forma sistemática, com exclusão de todos os outros. Ou seja, vamos ter que desenvolver uma álgebra que, ao contrário da álgebra corrente, recorre a variáveis e a funções que apenas podem assumir os valores 0 e 1. Quais as operações (e operadores) que podemos utilizar nessa álgebra, é o que discutiremos na Secção 3.5. Porém, deve ficar desde já claro que essa álgebra irá desenvolver-se axiomaticamente, com base num conjunto restrito de postulados.

Quantidades booleanas simples e gerais

Antes de passarmos ao estudo dessa álgebra, porém, vamos apresentar informalmente alguns conceitos básicos. O primeiro tem a ver com as designações a dar a cada um dos símbolos 0 e 1. Designá-los-emos por **quantidades booleanas simples**. E aos conjuntos ordenados destes símbolos como, por exemplo, em $(0, 1, 1, 0, 0, 1)$, damos a designação de **quantidades booleanas gerais**.

Variáveis (booleanas simples)

Por outro lado, utilizaremos frequentemente variáveis com as designações habituais em qualquer álgebra, por exemplo x, A, z_5 ou ω, não esquecendo que elas apenas podem tomar valores no conjunto $\{0, 1\}$. Essas variáveis tomarão a designação colectiva de **variáveis booleanas simples** ou, por vezes e de forma mais fácil, de **variáveis (booleanas)**.

Variáveis booleanas gerais

Por outro lado, iremos precisar correntemente de conjuntos ordenados de variáveis, por exemplo $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, que designaremos por **variáveis booleanas gerais**.

Funções booleanas simples

Finalmente, precisaremos de definir funções booleanas, que mais não são do que aplicações do conjunto $\{0,1\}^n$ no conjunto $\{0,1\}$, por exemplo $f(x_1, x_2, \ldots, x_n)$. Trata-se, neste caso mais geral, de uma **função booleana simples**, f, de uma variável booleana geral $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. No caso particular em que apenas está envolvida uma variável booleana simples, digamos x, teremos que $f(x)$ é uma função booleana simples de uma variável booleana simples.

Nos casos mais complicados necessitaremos de conjuntos ordenados de funções booleanas simples, como em $\mathbf{f} = (f_1, f_2, \ldots, f_k)$, em que todas as funções booleanas simples f_1, f_2, ... f_k envolvem o mesmo número de variáveis booleanas simples, como, por exemplo, em

$$f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_k(x_1, x_2, \ldots, x_n)\,.$$

Funções booleanas gerais

Neste caso falamos de uma **função booleana geral**, $\mathbf{f}$, da variável booleana geral $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

Funções lógicas

Estas funções, por vezes também designadas por **funções lógicas**, vão desempenhar um papel central nas metodologias de análise e de síntese dos circuitos digitais, que empreenderemos a partir do Capítulo 8. Para isso, precisamos de saber representar adequadamente as funções booleanas — o que faremos no Capítulo 4 — e precisamos ainda de saber dar-lhes a forma ou as formas algébricas mais simples, um problema que designaremos por minimização das funções e que estudaremos no Capítulo 5.

3.2 Funções com Uma Variável

Complementação ou negação

Complemento de uma variável ou função

Antes de analisarmos as funções booleanas simples de 1 variável booleana simples, digamos $f_i(x)$, precisamos de introduzir a operação de **complementação** ou de **negação** da variável x (ou da função f_i). É fácil perceber o sentido desta operação: se x (ou f_i) tiver, num dado momento, o valor 0, o seu **complemento** tem o valor 1 nesse momento. E inversamente, se o valor de x (ou de f_i) for 1 a certa altura, então o seu complemento tem o valor 0 nessa altura. Só precisamos de arranjar um símbolo para o complemento da variável ou da função. Utilizaremos $\overline{x}$ e $\overline{f_i}$ para designar esses complementos.

Nestas condições, é fácil perceber que apenas podemos formar 4 funções booleanas simples de 1 ou menos variáveis booleanas simples: as funções

$$f_0 = x, \quad f_1 = \overline{x}, \quad f_2 = 0 \quad \text{e} \quad f_3 = 1\,.$$

Tabela de verdade (lógica)

Vamos escrever estas funções numa tabela especial, designada por **tabela de verdade lógica** ou, mais simplesmente, por **tabela de verdade** das funções, que representa os valores lógicos 0 ou 1 de cada função booleana simples para cada um dos valores lógicos 0 ou 1 da variável booleana simples. Como neste caso temos 4 funções booleanas simples, o que representamos na Tabela 3.1 são, na realidade, 4 tabelas de verdade compactadas numa única.

Função identidade

Notemos como a **função identidade**, $f_0(x) = x$, possui, para cada linha, o mesmo valor que x.

Função complementação (negação, NOT)

Notemos ainda como a **função complementação** (ou **função negação**, ou **função NOT**), $f_1(x) = \overline{x}$, possui, para cada linha, o complemento do valor de x.

Tabela 3.1: Tabelas de verdade das funções booleanas simples de uma ou zero variáveis booleanas simples, onde se enumeram as linhas pelos equivalentes decimais dos números binários correspondentes às quantidades booleanas simples 0 e 1 (*Nota: a numeração das linhas não faz parte da tabela*)

Linha #	x	**Função identidade** $f_0(x) = x$	**Função complementação** $f_1(x) = \overline{x}$	**Função constante 0** $f_2(x) = 0$	**Função constante 1** $f_3(x) = 1$
0	0	0	1	0	1
1	1	1	0	0	1

Funções constantes

Finalmente, notemos como as **funções constantes 0 e 1**, respectivamente $f_2(x) = = 0$ e $f_3(x) = 1$, possuem o valor constante 0 e 1 qualquer que seja o valor de x.

Estas funções vão intervir a dois níveis: (i) em *expressões booleanas*, ou *expressões lógicas*, como acabámos de ver nos quatro casos anteriores (expressões booleanas mais complexas serão estudadas na Secção 4.1); e (ii) em *logigramas* e *esquemas eléctricos*, que mais não são do que representações gráficas adequadas para as funções.

Operadores 0, 1, identidade e NOT

Porta lógica

Nestes últimos casos teremos que estabelecer uma simbologia própria para cada função, ou melhor, para cada operador funcional (**operador identidade**, **operador NOT**, **operador constante 0** e **operador constante 1**). Como veremos no Capítulo 6, a cada um destes operadores algébricos vai corresponder uma contrapartida física directa que pode assumir três variantes: (a) a forma de uma **porta lógica**, como acontece com o operador NOT; (b) a forma de um fio (uma linha, ou ligação), como acontece com o operador identidade; ou (c) a forma de níveis de tensão eléctrica H e L, em representação dos operadores constantes.

Importa, por isso, começar a apresentar alguns desses símbolos, à medida que se tornam necessários. Naturalmente, ao fazê-lo estamos a sair da esfera estrita da álgebra de Boole binária para passarmos para o campo da implementação física dos sistemas digitais, sob a forma de circuitos digitais. Como ainda é muito cedo para fazermos essa passagem, procederemos com cuidado por agora.

Porta NOT (inversora)

A representação de uma **porta NOT** ou **porta inversora** num logigrama ou esquema eléctrico faz-se à custa de um símbolo normalizado (aliás, os símbolos normalizados não serão apenas utilizados para representar as portas lógicas, como também representarão outros circuitos mais complexos que havemos de estudar umais tarde).

Norma IEC 60617-12

A norma internacional que está na base dessas representações designa-se por **norma IEC 60617-12** e foi estabelecida pela Comissão Electrotécnica Internacional (esta norma foi subsequentemente transposta para normas nacionais pelos países que integram a Comissão).

Neste curso seguiremos a norma IEC 60617-12 de forma tão escrupulosa quanto possível, o que tem três vantagens. Por um lado, a representação dos circuitos digitais torna-se rigorosa e independente de quaisquer "fantasias" representativas (o que sucede frequentemente, mesmo em textos com responsabilidades pedagógicas).

Por outro lado, um logigrama ou esquema eléctrico normalizado pode ser correctamente interpretado em qualquer parte do mundo (por quem conheça a norma, naturalmente), o que lhe confere um carácter de universalidade que uma simbologia *ad-hoc* não possui.

Finalmente, uma simbologia exacta como é esta dispensa a inclusão de um texto mais ou menos longo a explicar as características dos circuitos, porque essas características são evidentes: basta interpretar correctamente os símbolos.

Símbolos de uma porta NOT

Na Figura 3.1 apresentam-se dois **símbolos IEC para a porta NOT**. Embora intermutáveis, por vezes dá-se a primazia a um deles, consoante os circuitos em que se inserem (como veremos mais tarde).

Figura 3.1: Símbolos IEC possíveis para uma porta NOT

3.3 Funções com Duas Variáveis

O número de funções booleanas simples de 2 ou menos variáveis já é muito maior do que 4. Vamos tentar perceber quantas são essas funções, e em seguida generalizar para um número n qualquer de variáveis booleanas simples.

Comecemos por perceber a estrutura das tabelas de verdade para as funções de 2 variáveis. Enquanto para uma variável, por exemplo x, a tabela só tem duas linhas, para os valores 0 e 1 da variável (*vd.* a Tabela 3.1), para tabelas com 2 variáveis, por exemplo x e y, vamos precisar de 4 linhas, correspondentes aos 4 valores lógicos possíveis para essas variáveis: ou seja, as tabelas vão ter uma linha por cada quantidade booleana geral $(x, y) = (0, 0)$, $(x, y) = (0, 1)$, $(x, y) = (1, 0)$ e $(x, y) = (1, 1)$. E, se quisermos numerar as linhas, podemos atribuir-lhes os equivalentes decimais dos números binários 00, 01, 10 e 11, ou seja, respectivamente 0, 1, 2 e 3 se admitirmos que o peso de x é maior do que o de y. Quanto às colunas da tabela, existe uma coluna por função.

Vamos agora ver *quantas tabelas de verdade* podemos construir para funções de 2 ou menos variáveis ou, o que é o mesmo, quantas funções de 2 variáveis ou menos conseguimos construir. Na Tabela 3.1 incluímos 4 colunas, uma por função (ou seja, estão lá 4 tabelas de verdade); por outro lado, as 4 colunas corresponderam a *todos as quantidades booleanas gerais que conseguimos formar*.

E para 2 variáveis? Com 4 linhas por tabela de verdade, conseguimos formar 16 quantidades booleanas gerais. Logo, há 16 funções com 2 ou menos variáveis, como mostra a Tabela 3.2.

Generalizando, com n variáveis booleanas simples podemos formar 2^{2^n} funções booleanas simples, um número que cresce exponencialmente com n.

Tabela 3.2: Possíveis funções booleanas simples com duas ou menos variáveis booleanas simples

x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Das funções com 2 variáveis salientamos as seguintes:

$$\begin{aligned}
f_3(x,y) &= \overline{x} && \text{(a função complementação)}\\
f_5(x,y) &= \overline{y} && \text{(a função complementação)}\\
f_8(x,y) &= x\cdot y && \text{(a função AND)}\\
f_{14}(x,y) &= x+y && \text{(a função OR)}\\
f_7(x,y) &= \overline{x\,y} && \text{(a função NAND)}\\
f_1(x,y) &= \overline{x+y} && \text{(a função NOR)}\\
f_6(x,y) &= x\oplus y && \text{(a função OU-exclusivo)}\\
f_9(x,y) &= x\odot y && \text{(a função Equivalência)}
\end{aligned}$$

Função complementação (negação, NOT)

Vamos examinar estas funções mais em pormenor. A **função complementação** (ou **negação**, ou **NOT**) já é conhecida para 1 variável. Para 2 variáveis toma duas formas possíveis,

$$f_3(x,y) = \overline{x} \qquad \text{ou} \qquad f_5(x,y) = \overline{y}\,,$$

como mostra a sua tabela de verdade, incluída na Tabela 3.3.

Tabela 3.3: Tabelas de verdade das funções complementação, AND e OR com 2 variáveis, onde se enumeram as linhas pelos equivalentes decimais dos números binários correspondentes às quantidades booleanas gerais $(0,0)$, $(0,1)$, $(1,0)$ e $(1,1)$ (*Nota: a numeração das linhas não faz parte da tabela*)

Linha #	x	y	**Função complement.** $f_3(x,y)=\overline{x}$	**Função complement.** $f_5(x,y)=\overline{y}$	**Função AND** $f_8(x,y)=x\,y$	**Função OR** $f_{14}(x,y)=x+y$
0	0	0	1	1	0	0
1	0	1	1	0	0	1
2	1	0	0	1	0	1
3	1	1	0	0	1	1

Reparemos como $f_3(x,y) = \overline{x}$ vale 1 onde x vale 0, e vice-versa. E, identicamente, como $f_5(x,y) = \overline{y}$ vale 1 onde y vale 0, e vice-versa.

Função AND de 2 variáveis

A **função AND de 2 variáveis**, $f_8(x,y) = x\cdot y$, vale 1 apenas quando as duas variáveis valem 1. Notemos a simbologia do operador utilizado para representar

esta função, traduzida pelo símbolo "·" colocado entre as variáveis ou, para simplificar, sem o ponto, como em $f_8(x, y) = x\,y$.

Função OR de 2 variáveis

A **função OR de 2 variáveis**, $f_{14}(x, y) = x + y$, vale 1 se pelo menos uma das variáveis valer 1. A simbologia do operador que representa esta função traduz-se pelo símbolo "+".

Na Tabela 3.4 encontramos mais quatro funções que importa salientar.

Tabela 3.4: Tabelas de verdade das funções NAND, NOR, Ou-exclusivo e Equivalência com 2 variáveis, onde se enumeram as linhas pelos equivalentes decimais dos números binários correspondentes às quantidades booleanas gerais $(0, 0)$, $(0, 1)$, $(1, 0)$ e $(1, 1)$

x	y	**Função NAND** $f_7(x,y) = \overline{x\,y}$	**Função NOR** $f_1(x,y) = \overline{x+y}$	**Função OU-exclusivo** $f_6(x,y) = x \oplus y$	**Função Equivalência** $f_9(x,y) = x \odot y$
0	0	1	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1

Funções NAND e NOR de 2 variáveis

A **função NAND com 2 variáveis**, $f_7(x, y) = \overline{x\,y}$, é o complemento do AND. E a **função NOR com 2 variáveis**, $f_1(x, y) = \overline{x + y}$, é o complemento do OR .

Função OU-exclusivo

Por sua vez, a **função Ou-exclusivo**, $f_6(x, y) = x \oplus y$, vale 1 quando x e y tomam valores diferentes:

$$x \oplus y \stackrel{\text{def}}{=} x\,\overline{y} + \overline{x}\,y\,.$$

Função Equivalência

Quanto à **função Equivalência**, $f_9(x, y) = x \odot y$, vale 1 quando x e y tiverem o mesmo valor (quando forem "equivalentes"):

$$x \odot y \stackrel{\text{def}}{=} x\,y + \overline{x}\,\overline{y}\,.$$

Devemos notar que $x \odot y = \overline{x \oplus y}$.

Funções identidade
Funções constantes

Finalmente, mencionam-se as **funções identidade**, $f_{12}(x, y) = x$ e $f_{10}(x, y) = y$, e as **funções constantes**, $f_0(x, y) = 0$ e $f_{15}(x, y) = 1$.

Operadores AND, OR, NAND, NOR, XOR e XNOR

Para completar esta secção, vamos de seguida apresentar os símbolos IEC de algumas portas lógicas, as que implementam os **operadores AND, OR, NAND, NOR, XOR e XNOR**. Notemos que o operador XOR implementa a função Ou-exclusivo, e que o operador XNOR implementa a função Equivalência.

Símbolos das portas AND, OR, NAND. NOR, XOR e XNOR com 2 entradas

Na Figura 3.2 apresentam-se os **símbolos IEC** das portas lógicas com os mesmos nomes e com 2 entradas.

Como veremos mais tarde, no Capítulo 7 em que estudaremos a lógica de polaridade, veremos que estas designações só são válidas para certos contextos, e nessa altura apresentaremos designações mais apropriadas para algumas destas portas.

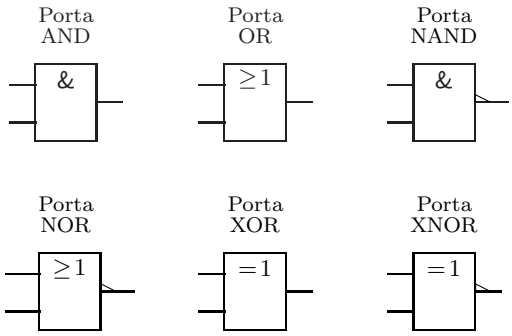


Figura 3.2: Símbolos IEC das portas AND, OR, NAND, NOR, XOR (que implementa a função OU-exclusivo) e XNOR (que implementa a função Equivalência) com 2 entradas

3.4 Funções com Mais do que Duas Variáveis

Como vimos na secção anterior, com n variáveis booleanas simples podemos formar 2^{2^n} funções booleanas simples. Naturalmente, este crescimento exponencial com o valor de n torna impraticável a enumeração de todas as funções para um valor arbitrário de n, com $n > 2$. Contudo, podemos enumerar algumas. Por exemplo, as funções AND, OR, NAND e NOR com mais de 2 variáveis, com expressões lógicas

$$\begin{aligned} f_{\text{AND}}(k, l, \ldots, z) &= k\, l\, \ldots\, z \\ f_{\text{OR}}(k, l, \ldots, z) &= k + l + \cdots + z \\ f_{\text{NAND}}(k, l, \ldots, z) &= \overline{k\, l\, \ldots\, z} \\ f_{\text{NOR}}(k, l, \ldots, z) &= \overline{k + l + \cdots + z} \end{aligned}$$

e com os **símbolos IEC** da Figura 3.3, nos casos em que as portas lógicas com as mesmas designações possuem 3 entradas (facilmente podemos extrapolar os símbolos para mais do que 3 entradas).

Símbolos das portas AND, OR, NAND e NOR com 3 entradas

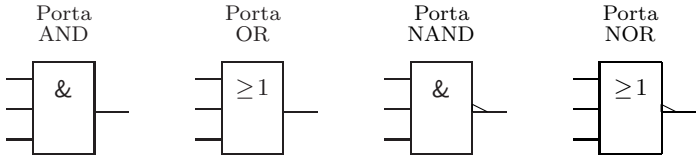


Figura 3.3: Símbolos IEC das portas AND, OR, NAND e NOR com 3 entradas

Podíamos ainda enumerar outras funções, como as funções identidade e constantes que envolvem 3 ou mais variáveis. Mas, naturalmente, jás as conhecemos dos casos em que apenas estão envolvidas 1 e 2 variáveis.

Quanto à função OU-exclusivo, podemos ainda extrapolá-la para mais do que 2 variáveis. Mas nesses casos preferem-se utilizar portas XOR com 2 entradas e utilizar a propriedade associativa da função (como estudaremos mais à frente neste capítulo) para formar portas XOR com 3 ou mais entradas. E outro tanto para a função Equivalência e para as correspondentes portas XNOR.

3.5 Axiomas e Teoremas da Álgebra de Boole Binária

3.5.1 Axiomas

Soma lógica e produto lógico

A álgebra de Boole binária é formada por um conjunto $\{A, B, C, \ldots\}$ e por duas operações binárias, $+$ (**soma lógica**) e $\cdot$ (**produto lógico**) que satisfazem a propriedade de fecho e que obedecem aos seguintes axiomas ou postulados:

Axioma das comutatividades

AXIOMA DAS COMUTATIVIDADES

A1a. $A \cdot B = B \cdot A$

A1b. $A + B = B + A$

Axioma das distributividades

AXIOMA DAS DISTRIBUTIVIDADES

A2a. $A \cdot (B + C) = A \cdot B + A \cdot C$

A2b. $A + B \cdot C = (A + B) \cdot (A + C)$

Axioma das identidades

AXIOMA DAS IDENTIDADES. Este axioma define a existência de dois elementos identidade, o 0 e o 1, que são os elementos neutro, respectivamente, da soma lógica (o 0) e do produto lógico (o 1):

A3a. $A \cdot 1 = A$

A3b. $A + 0 = A$

Axioma do complemento

AXIOMA DO COMPLEMENTO

A4a. $A \cdot \overline{A} = 0$

A4b. $A + \overline{A} = 1$

O produto lógico pode ser omitido, quando do facto não resultar confusão; por exemplo, pode escrever-se $A\,B$ em lugar de $A \cdot B$, como já vimos anteriormente.

Precedências

Por seu turno, em expressões que envolvam vários produtos e somas lógicas *o produto tem precedência sobre a soma*; ; assim,

$$A + B\,C$$

deve ser entendido como primeiro fazendo-se o produto lógico de B por C e só em seguida a soma lógica desse produto com A.

Quando necessário, utilizam-se parêntesis para tornar claras as sequências de operações numa expressão. Por exemplo,

$$(A + B)\,C$$

deve ser entendido como primeiro fazendo-se a soma lógica de A com B e só depois o produto lógico dessa soma com C.

3.5.2 Teoremas

É razoavelmente elevado o número de teoremas que podemos deduzir no contexto da álgebra de Boole binária acima definida. Limitar-nos-emos a enunciar em seguida os principais. Sugere-se que o aluno pelo menos tente provar alguns deles.

Teorema da idempotência. *Teorema da idempotência*

T1a. $A\,A = A$

T1b. $A + A = A$

Teorema dos elementos absorventes. O 0 e o 1, que são definidos axiomaticamente como elementos neutros, respectivamente, da soma lógica e do produto lógico, são igualmente elementos absorventes, respectivamente, do produto lógico e da soma lógica: *Teorema dos elementos absorventes*

T2a. $A \cdot 0 = 0$

T2b. $A + 1 = 1$

Teorema da associatividade. *Teorema da associatividade*

T3a. $(A + B) + C = A + (B + C)$

T3b. $(A\,B)\,C = A\,(B\,C)$

Teorema da involução.

T4. $\overline{\overline{A}} = A$ *Teorema da involução*

Teorema da absorção. *Teorema da absorção*

T5a. $A + A\,B = A$

T5b. $A\,(A + B) = A$

Teorema da redundância. *Teorema da redundância*

T6a. $A + \overline{A}\,B = A + B$

T6b. $A\,(\overline{A} + B) = A\,B$

Teorema da adjacência. *Teorema da adjacência*

T7a. $A\,B + A\,\overline{B} = A$

T7b. $(A + B)\,(A + \overline{B}) = A$

Princípio da dualidade. Este princípio estabelece o seguinte: toda a expressão formada por elementos do conjunto $\{A, B, C, \dots\}$ mais os elementos 0 e 1 e que envolva as operações de soma lógica, de produto lógico e de complementação, possui uma expressão dual que se obtém trocando cada soma por um produto lógico e cada produto por uma soma lógica, e ainda os "0"s por "1"s e os "1"s por "0"s. *Princípio da dualidade*

Assim, a expressão dual da expressão $A\,\overline{1} + \overline{A}\,1$ é $(A + \overline{0})\,(\overline{A} + 0)$. Repare-se que as variáveis ou os seus complementos (os chamados **literais**), no caso A e $\overline{A}$, não vêm alterados por dualidade. *Literal*

Devemos notar que os axiomas e a grande maioria dos teoremas da álgebra de Boole binária existe em versões duais (a excepção é o teorema da involução). Por exemplo, como o teorema da absorção na forma $A + A \cdot B = A$ é válido, também o é a sua forma dual, $A \cdot (A + B) = A$.

Leis de De Morgan

LEIS DE DE MORGAN.

T8a. $\overline{A \cdot B} = \overline{A} + \overline{B}$

T8b. $\overline{A + B} = \overline{A} \cdot \overline{B}$

Teoremas envolvendo o OU-exclusivo

TEOREMAS ENVOLVENDO O OU-EXCLUSIVO. Existem vários teoremas que envolvem a função OU-exclusivo de duas ou mais variáveis booleanas simples. Antes de enumerarmos alguns desses teoremas, relembremos a definição do OU-exclusivo,

$$A \oplus B \stackrel{\text{def}}{=} A\,\overline{B} + \overline{A}\,B = (A + B)\,(\overline{A} + \overline{B})\,.$$

Consideremos, então, os principais teoremas envolvendo OU-exclusivos:

T9. $A \oplus B = B \oplus A$ (Comutatividade da função OU-exclusivo)

T10. $A \oplus (B \oplus C) = (A \oplus B) \oplus C$ (Associatividade da função OU-exclusivo)

T11. $A \oplus 0 = A$

T12. $A \oplus 1 = \overline{A}$

T13. $\overline{A \oplus B} = \overline{A} \oplus B = A \oplus \overline{B} = A \odot B$

(como sabemos, a função Equivalência, $A \odot B$, é igual ao complemento da função OU-exclusivo).

3.6 Referências Bibliográficas

Sêrro, Carlos — *Sistemas Digitais: Fundamentos Algébricos*, IST Press, Lisboa, 2003, Capítulo 3.

Arroz, G. S., Monteiro, J. C. e Oliveira, A. L. — *Introdução aos Sistemas Digitais e Microprocessadores*, Secções 3.1.1 a 3.1.9.

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 2.1, 2.2, 2.6 e 2.7.

3.7 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 3.1 Mostre que a função Equivalência é comutativa e associativa.

(∗) 3.2 Prove a seguinte lei de De Morgan: $\overline{x + y} = \overline{x}\,\overline{y}$.

(∗) 3.3 Escreva tabelas de verdade adequadas para as seguintes funções booleanas simples:

a) $f(A, B, C) = A\,(\overline{B} + \overline{C})\,(B + C)$;
b) $f(A, B, C, D) = A\,(B + \overline{C}\,(\overline{B} + D))$;
c) $f(A, B, C) = \overline{A\,C} + \overline{B\,C}$.

(∗) 3.4 Prove, examinando todos os casos possíveis (demonstração por **indução completa**), que os seguintes teoremas são válidos (os pares de teoremas são duais): *Indução completa*

a) $\overline{\overline{A}} = A$;
b) $A + 0 = A \qquad A \cdot 1 = A$;
c) $A + 1 = 1 \qquad A \cdot 0 = 0$;
d) $A + A = A \qquad A\,A = A$;
e) $A + \overline{A} = 1 \qquad A\,\overline{A} = 0$.

(∗) 3.5 Prove, por indução completa, que $A\,B + \overline{A}\,C + B\,C = A\,B + \overline{A}\,C$ (**teorema do consenso**). *Teorema do consenso*

(∗) 3.6 Através de manipulações algébricas, e utilizando os axiomas e os teoremas da álgebra de Boole binária que conhece, verifique as seguintes igualdades:

a) $(A + \overline{B} + A\,B)\,(A + \overline{B})\,\overline{A}\,B = 0$;
b) $\overline{\overline{A}\,B\,(\overline{D} + D\,\overline{C}) + (A + D\,\overline{A}\,C)\,B} = B$;
c) $[(\overline{\overline{B} + C})\,A] + (\overline{C\,D}) = C\,D$.

3.7 Verifique as seguintes igualdades:

a) $X + Y\,Z = (X + Y)\,(X + Z)$;
b) $X\,(Y + Z) = X\,Y + X\,Z$;
c) $(A \oplus B) \oplus C = A \oplus (B \oplus C)$;
d) $(A + B)\,(\overline{A} + C)\,(B + C) = (A + B)\,(\overline{A} + C)$;
e) $\overline{A}\,\overline{C} + \overline{A}\,B + B\,C = \overline{A}\,\overline{C} + B\,C$.

Como as designa?

3.8 Usando os teoremas do texto, simplificar algebricamente as expressões booleanas das seguintes funções booleanas simples:

a) $F = \overline{X}\,Y\,Z + \overline{X}\,Y\,\overline{Z} + X\,Z$;
b) $G = \overline{X\,(\overline{Y}\,\overline{Z} + Y\,Z)}$.

(∗) 3.9 Simplifique algebricamente

a) $A\,B\,C\,D + A\,B\,C\,\overline{D} + \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$;
b) $\overline{X} + X\,Y\,\overline{Z} + \overline{Y}$;
c) $X\,Y + W\,X\,Y\,\overline{Z} + \overline{X}\,Y$;
d) $\overline{X}\,\overline{Y}\,Z + Y\,Z + X\,Z$.

3.10 Simplifique algebricamente as seguintes funções:

a) $f = \overline{C}\,B\,(A \oplus \overline{D}) + \overline{C}\,B\,A + B\,\overline{C}\,D + A\,D$;
b) $f = C\,(B \oplus \overline{A})\,\overline{D} + \overline{B}\,C\,\overline{D} + A\,\overline{D} + \overline{A}\,B\,\overline{D}$;
c) $f = \overline{A}\,(\overline{C} \oplus D) + A\,\overline{B} + \overline{A}\,\overline{C}\,D + \overline{A}\,C\,\overline{D}$;
d) $f = A\,\overline{C}\,\overline{D} + \overline{A}\,C\,\overline{D} + (\overline{A} \oplus C)\,\overline{D} + \overline{B}\,D$.

3.11 Simplifique algebricamente as seguintes funções:

a) $f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + B\,\overline{C}\,\overline{D} + A\,(C \oplus \overline{D})\overline{B} + A\,B\,C\,D$;
b) $f = \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C} + \overline{A}\,B\,D + A\,\overline{C}\,D + C\,D + A\,B\,C + A\,C\,\overline{D}$;
c) $f = A\,\overline{C}\,(\overline{B} \oplus D) + A\,B\,\overline{D} + A\,\overline{B}\,D + A\,B\,C\,D + \overline{A}\,C\,\overline{D}$;
d) $f = A\,B\,C + \overline{A\,\overline{C}}\,(C + D) + \overline{A + B + \overline{D}}$.

3.12 Verifique que:

a) se $A \odot B = 0$, então $A = B$;
b) se $A \odot B = A \odot C$, então $B = C$;
c) $X + Y = X \odot Y \odot X\,Y$;
d) $\overline{X} = X \odot 1$.

(∗) 3.13 Um técnico de laboratório químico possui quatro produtos químicos, A, B, C e D, que devem ser guardados em dois depósitos. Por conveniência, é necessário mover um ou mais produtos de um depósito para o outro de tempos a tempos. A natureza dos produtos é tal que é perigoso guardar B e C juntos, a não ser que A esteja no mesmo depósito. Também é perigoso guardar C e D juntos se A não estiver no depósito. Escreva uma expressão para uma função, Z, tal que $Z = 1$ sempre que exista uma combinação perigosa em qualquer dos depósitos.

(∗) 3.14 Existem três interruptores de parede, a, b e c. $A = 1$ representa a condição "interruptor a ligado", e $A = 0$ representa a condição "interruptor a desligado". De modo similar, as variáveis B e C estão associadas às posições dos interruptores b e c, respectivamente. Escreva uma expressão booleana para uma função Z, de modo que a alteração do estado de um interruptor, independentemente dos outros, vá provocar a mudança do valor da função.

Capítulo 4

Representação das Funções

4.1 Representação por Expressões Booleanas

Consideremos a seguinte função booleana simples,

$$F = A\,B + A\,\overline{C}\,.$$

Esta forma de representação designa-se por **forma normal disjuntiva** ou **soma de produtos** da função.

Forma normal disjuntiva ou soma de produtos

Em alternativa, a mesma função pode ser descrita pela seguinte **forma normal conjuntiva** ou **produto de somas**:

Forma normal conjuntiva ou produto de somas

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

Em ambos os casos estamos a representar a função por **expressões booleanas** ou **lógicas**. Devemos notar que existem muitas expressões booleanas para representar uma determinada função booleana simples, embora tenhamos representado F apenas por duas delas.

Expressões booleanas (lógicas)

Para obter uma expressão booleanas a partir de outra basta proceder a algumas manipulações algébricas. Por exemplo, a partir da forma normal disjuntiva $A\,B + A\,\overline{C}$ de F podemos obter:

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

por utilização do axioma da distributividade do produto lógico em relação à soma lógica.

As representações das funções booleanas simples em somas de produtos ou em produtos de somas designam-se, em conjunto, por **representação algébrica** das funções, por oposição a outras formas de representação como as que iremos estudar já de seguida.

Representação algébrica

Nem sempre a representação algébrica de uma função booleana simples vem em forma normal conjuntiva ou disjuntiva. Por exemplo, a expresssão algébrica

$$G = A\,B + \overline{A}\,B\,C\,(X + Y)$$

vem numa forma mista entre as duas. Naturalmente, por manipulação algébrica é sempre possível obter formas normais a partir de formas mistas, em geral por aplicação dos axiomas da distributividade. Porém, a inversa nem sempre é possível. Por exemplo, a expressão normal disjuntiva $A\overline{B} + \overline{A}\,B\,C$ não se consegue pôr em forma mista.

4.2 Representação por Tabelas de Verdade

Tabela de verdade (lógica)

Como já sabemos do capítulo anterior, uma das formas de representação de uma função booleana simples recorre à sua tabela de verdade lógica ou, mais simplesmente, **tabela de verdade**, que é única para cada função.

A tabela de verdade para uma função arbitrária, $f(x_n, x_{n-1}, \ldots, x_2, x_1)$, tem a seguinte estrutura, já conhecida: (i) $n+1$ colunas, sendo as primeiras n colunas para as variáveis booleanas simples e uma coluna para a função; e (ii) 2^n linhas, cada uma correspondente a uma quantidade booleana geral diferente; e (iii) as linhas são ordenadas pelos equivalentes decimais dos números binários que identificam as quantidades booleanas gerais.

Como exemplo, consideremos, na Tabela 4.1, a tabela de verdade de uma função arbitrária de 3 variáveis booleanas simples, $F(A, B, C)$.

Tabela 4.1: Tabela de verdade da função booleana simples $F(A, B, C) = = A\,B + A\,\overline{C}$, onde se enumeram as linhas pelos equivalentes decimais dos números binários correspondentes às quantidades booleanas gerais (*Nota: a numeração das linhas não faz parte da tabela*)

Linha #	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Porque precisamos de numerar as linhas da tabela pelos equivalentes decimais dos números binários que as identificam, precisamos de atribuir pesos às variáveis booleanas simples A, B e C. *Regra geral, admite-se que a variável mais à esquerda na tabela é a que tem maior peso, e que a variável mais à direita tem o menor peso (mas esta regra não tem que ser sempre seguida).*

Se admitirmos esta distribuição de pesos, a variável booleana simples C tem um peso igual a $2^0 = 1$, a variável B tem um peso igual a $2^1 = 2$, e a variável A tem um peso igual a $2^2 = 4$.

Nestas condições:

1. a linha $(A, B, C) = (0, 0, 0)$, correspondente ao número binário 000, tem como equivalente decimal o valor $0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 0$;
2. a linha $(A, B, C) = (0, 0, 1)$, correspondente ao número binário 001, tem como equivalente decimal o valor $0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 1$;
3. a linha $(A, B, C) = (0, 1, 0)$, correspondente ao número binário 010, tem como equivalente decimal o valor $0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2$; etc.;
4. a linha $(A, B, C) = (1, 1, 1)$, correspondente ao número binário 111, tem como equivalente decimal o valor $1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7$,

e as linhas da tabela encontram-se ordenadas pelos equivalentes decimais dos números binários que as identificam (de 0 até 7), como se disse acima.

Põe-se, de seguida, o problema de *ler uma tabela de verdade* e, dessa leitura, deduzir a expressão booleana da função nela representada. Esse problema será abordado em pormenor mais à frente neste capítulo, quando se estudarem as formas canónicas das funções booleanas simples.

Entretanto, podemos deduzir o seguinte:

1. a função F possui "1"s nas linhas 4, 6 e 7 da tabela de verdade;
2. a linha 4 é caracterizada pela quantidade booleana geral $(A, B, C) = = (1, 0, 0)$, pelo que $F = 1$ quando $A = 1$, $B = 0$ e $C = 0$ e, portanto, quando $A\,\overline{B}\,\overline{C} = 1$;
3. a linha 6 é caracterizada pela quantidade booleana geral $(A, B, C) = = (1, 1, 0)$, pelo que $F = 1$ quando $A = 1$, $B = 1$ e $C = 0$ e, portanto, quando $A\,B\,\overline{C} = 1$;
4. a linha 7 é caracterizada pela quantidade booleana geral $(A, B, C) = = (1, 1, 1)$, pelo que $F = 1$ quando $A = 1$, $B = 1$ e $C = 1$ e, portanto, quando $A\,B\,C = 1$;

Segue-se que

$$F = A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C\,,$$

que podemos, obviamente, tentar simplificar usando os axiomas e os teoremas da álgebra de Boole binária:

$$\begin{aligned} F &= A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C \\ &= A\,\overline{C}\,(\overline{B} + B) + A\,B\,(\overline{C} + C) \\ &= A\,B + A\,\overline{C}\,. \end{aligned}$$

A leitura de uma tabela de verdade pode ser feita de forma mais expedita por simples análise do seu conteúdo. Para vermos como, consideremos mais uma vez a Tabela 4.1 e, nela, os pares de linhas em que a função vale 1 e que apenas diferem numa quantidade booleana simples.

Como $F = 1$ nas linhas 4, 6 e 7, apenas existem dois pares de linhas nestas condições: o par $(4, 6)$ e o par $(6, 7)$.

As linhas do par $(4, 6)$ diferem na variável B — que vale 0 na linha 4 e 1 na linha 6 — mantendo-se as restantes variáveis com valores constantes, mais

exactamente $A = 1$ e $C = 0$. Segue-se que podemos identificar esse par de linhas pela expressão booleana $A\,\overline{C}$ (ver a Tabela 4.2), independentemente do valor de B, já que com $A = 1$ e $C = 0$ se tem $A\,\overline{C} = 1$. Como $F = 1$ para o par, segue-se que

$$F = A\,\overline{C} + \ldots .$$

Nas reticências está incluída a contribuição do outro par.

Tabela 4.2: Tabela de verdade da função booleana simples $F(A, B, C) = = A\,B + A\,\overline{C}$, onde se identificam pelas suas expressões booleanas os pares de linhas em que $F = 1$ e que só diferem numa quantidade booleana simples (*Nota: a numeração das linhas não faz parte da tabela*)

Linha #	A	B	C	F	
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	0	
4	1	0	0	1	Este par de linhas representa-se por $A\,\overline{C}$
5	1	0	1	0	
6	1	1	0	1	Este par de linhas representa-se por $A\,B$
7	1	1	1	1	

Logo, $F = A\,B + A\,\overline{C}$

Consideremos agora o par $(6, 7)$, que muda na variável C e que mantém constantes $A = 1$ e $B = 1$. Então, a expressão booleana que identifica este par de linhas é $A\,B$ (ver ainda a Tabela 4.2). Segue-se que

$$F = A\,B + \ldots .$$

Também aqui, as reticências incluem a contribuição do outro par de linhas.

Da conjugação destas duas expressões conclui-se que

$$F(A, B, C) = A\,B + A\,\overline{C}\,,$$

como já tinhamos obtido.

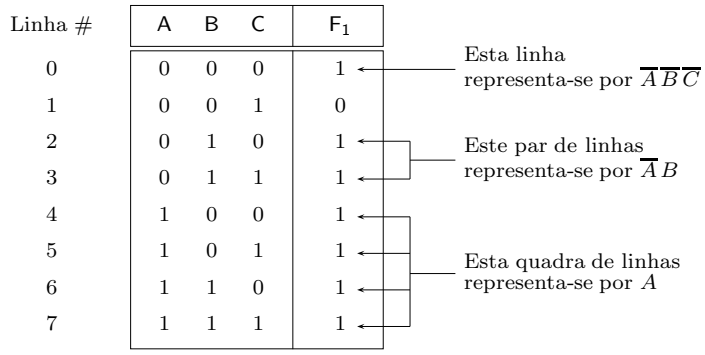
Podemos agora passar ao problema inverso, o da *escrita de uma tabela de verdade* — a obtenção da tabela a partir da expressão booleana da função que representa. Consideremos, como exemplo, a função

$$F_1(A, B, C) = A + \overline{A}\,B + \overline{A}\,\overline{B}\,\overline{C}\,.$$

A expressão parcial $\overline{A}\,\overline{B}\,\overline{C}$ vai identificar *uma (e só uma) linha da tabela em que a função vale* 1, mais concretamente a linha para a qual $A = 0$, $B = 0$ e $C = 0$ — isto é, a linha 0 (ver a Tabela 4.3).

Isto porque, para estes valores das variáveis, temos $\overline{A}\,\overline{B}\,\overline{C} = 1$ e, por conseguinte, $F_1 = 1$ (não esquecer que F_1 é igual à soma lógica desta expressão com outras, e se a expressão vale 1, a função também vale 1).

Tabela 4.3: Tabela de verdade da função booleana simples $F_1(A, B, C) = A + \overline{A}\,B + \overline{A}\,\overline{B}\,\overline{C}$ (*Nota: a numeração das linhas não faz parte da tabela*)



Reparar que uma linha da tabela vai corresponder a um produto lógico que envolve todos os literais da função, no caso os literais $\overline{A}$, $\overline{B}$ e $\overline{C}$.

Consideremos agora a expressão parcial $\overline{A}\,B$ de F_1. Como a esta expressão falta um literal, ela vai identificar *duas linhas da tabela em que a função vale* 1. Mais exactamente, trata-se das linhas em que $A = 0$ e $B = 1$, ou seja, as linhas 2 e 3 (independentemente dos possíveis valores de C).

Finalmente, a expressão parcial A (em que faltam 2 literais) vai identificar *quatro linhas da tabela em que a função vale* 1. Ou seja, identifica todas as linhas em que $A = 1$, isto é, as linhas 4 a 7 (independentemente dos possíveis valores de B ou de C).

Segue-se que F_1 vale 1 nas linhas 0, 2, 3 e 4 a 7, e vale 0 na que resta, a linha 1.

Naturalmente, existe uma outra forma de escrever a tabela de verdade de uma função, dada a sua expressão booleana. Basta ir gerando tabelas de verdade parciais para cada uma das parcelas da soma lógica (ou factores do produto lógico) da expressão. Por exemplo, a tabela de verdade da função anterior pode ser obtida como se ilustra na Tabela 4.4.

Tabela 4.4: Tabela de verdade da função booleana simples $F_1(A, B, C) = A + \overline{A}\,B + \overline{A}\,\overline{B}\,\overline{C}$, gerada a partir de tabelas de verdade parciais para cada uma das parcelas da soma lógica

A	B	C	A	$\overline{A}\,B$	$\overline{A}\,\overline{B}\,\overline{C}$	$F_1 = A + \overline{A}\,B + \overline{A}\,\overline{B}\,\overline{C}$
0	0	0	0	0	1	1
0	0	1	0	0	0	0
0	1	0	0	1	0	1
0	1	1	0	1	0	1
1	0	0	1	0	0	1
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	1	0	0	1

4.3 O Conjunto {AND, OR, NOT}

Qualquer função booleana simples pode ser representada recorrendo apenas a este conjunto de três funções. Por exemplo, a função Equivalência, $A \odot B$, pode ser escrita de diversas formas,

$$\begin{aligned} A \odot B &\stackrel{\text{def}}{\equiv} A\,B + \overline{A}\,\overline{B} \\ &= (\overline{A} + B)\,(A + \overline{B}) \\ &= \overline{A\,\overline{B}}\;\overline{\overline{A}\,B} \\ &= \overline{A\,\overline{B} + \overline{A}\,B}\,, \end{aligned}$$

que apenas utilizam as funções do conjunto {AND, OR, NOT}.

Para justificar que *qualquer* função booleana simples pode ser expressa usando apenas as funções do conjunto {AND, OR, NOT}, consideremos a tabela de verdade da Tabela 4.5, onde se apresentam três funções, $F1$, $F2$ e $F3$, obtidas da tabela de verdade de uma função F arbitrária, de modo a que cada uma das funções $F1$ a $F3$ apenas tem um 1 na tabela.

Tabela 4.5: Tabela de verdade das funções $F1$, $F2$ e $F3$, obtidas da tabela de verdade da função F

A	B	C	F1	F2	F3	F
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	1	1
1	0	1	0	0	0	0
1	1	0	0	1	0	1
1	1	1	1	0	0	1

É facil ver que $F = F1 + F2 + F3$. É facil de ver também que $F1 = A\,B\,C$, que $F2 = A\,B\,\overline{C}$ e que $F3 = A\,\overline{B}\,\overline{C}$ Logo, será

$$F = A\,B\,C + A\,B\,\overline{C} + A\,\overline{B}\,\overline{C}\,.$$

Outros conjuntos universais são, por exemplo, {NAND} e {NOR}, como veremos à frente.

Conjunto completo (universal)

Este método é utilizável com generalidade. Logo, é sempre possível definir *qualquer* função booleana simples utilizando o conjunto {AND, OR, NOT}, pelo que este conjunto se designa por **completo** ou **universal**.

4.4 Representação por Somas de Mintermos

Repare-se que o método que se acabou de descrever possibilitou a passagem da representação por tabela de verdade para uma representação por expressão

algébrica. Por acaso, como sabemos, não se obteve a expressão algébrica mais simples possível para a função F.

Contudo, esta expressão tem uma característica muito importante: trata-se de uma uma soma de produtos e TODOS os produtos envolvem TODOS os literais da função (recordemos da página 49 que um literal é uma variável ou o seu complemento).

Mintermo ou termo minimal

Os produtos lógicos deste tipo chamam-se **mintermos** ou **termos minimais** da função em causa. Repare-se que cada mintermo corresponde a um dos "1"s da função. Por exemplo, o mintermo $A\,B\,C$ corresponde ao 1 da última linha da tabela.

Se numerarmos as linhas da tabela de verdade como se fez atrás, com os equivalentes decimais dos números binários que correspondem às quantidades booleanas gerais em cada linha [a linha correspondente à quantidade booleana geral $(A, B, C) = (0, 0, 0)$ é numerada com 0, a linha correspondente à quantidade booleana geral $(A, B, C) = (0, 0, 1)$ é numerada com 1, etc., se A for a variável com maior peso], obtém-se uma tabela de verdade como a da Tabela 4.1, que se repete na Tabela 4.6 por comodidade.

Tabela 4.6: Tabela de verdade da função booleana simples $F(A, B, C) = = A\,B + A\,\overline{C}$ onde se enumeram as linhas pelos equivalentes decimais dos números correspondentes às quantidades booleanas gerais (*Nota: a numeração das linhas não faz parte da tabela*)

Linha #	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Podemos, assim, referir cada um dos mintermos pelo número da respectiva linha da tabela. Por exemplo, $A\,B\,C$ será o mintermo m_7 e $A\,\overline{B}\,\overline{C}$ será o mintermo m_4. Mais uma vez, esta numeração só faz sentido *depois* de termos atribuído pesos às variáveis (no caso, admitiu-se que A era a variável com maior peso).

Sendo assim, podemos escrever

$$\begin{aligned} F &= m_4 + m_6 + m_7 \\ &= A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C\,, \end{aligned}$$

ou ainda,

$$F = \sum m(4, 6, 7)\,.$$

Soma de mintermos
Primeira forma canónica ou forma canónica disjuntiva

A expressão, que é única para a função, é uma **soma de mintermos**. Trata-se da **primeira forma canónica** ou **forma canónica disjuntiva** da função.

Cada função booleana simples é representável por uma e só uma forma *canónica* disjuntiva, embora possa ser representada algébricamente por diversas formas *normais* disjuntivas.

4.5 Representação por Produtos de Maxtermos

Tal como se construíu a expressão anterior em termos dos "1s" da função, poderíamos ter trabalhado com os "0"s, como se ilustra na Tabela 4.7.

Tabela 4.7: Tabela de verdade das funções $G1$ a $G5$, obtidas da tabela de verdade da função F

A	B	C	G1	G2	G3	G4	G5	F
0	0	0	0	1	1	1	1	0
0	0	1	1	0	1	1	1	0
0	1	0	1	1	0	1	1	0
0	1	1	1	1	1	0	1	0
1	0	0	1	1	1	1	1	1
1	0	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

Agora é fácil de ver que

$$F = G1 \cdot G2 \cdot G3 \cdot G4 \cdot G5\,,$$

e, com algum trabalho de análise, conclui-se que

$$\begin{aligned} G1 &= A + B + C \\ G2 &= A + B + \overline{C} \\ G3 &= A + \overline{B} + C \\ G4 &= A + \overline{B} + \overline{C} \\ G5 &= \overline{A} + B + \overline{C} \end{aligned}$$

Por exemplo, $G1$ vale 1 por toda a parte excepto para a linha $(A, B, C) = = (0, 0, 0)$, em que vale 0. Logo, $G1$ deve ser o complemento da função que apenas vale 1 nessa linha e 0 nas restantes, isto é, o complemento de $m_0 = \overline{A}\,\overline{B}\,\overline{C}$. Ou seja,

$$G1 = \overline{\overline{A}\,\overline{B}\,\overline{C}} = A + B + C\,.$$

Em resumo,

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

Maxtermo ou termo maximal
Produto de maxtermos
Segunda forma canónica ou forma canónica conjuntiva

Estamos, agora, na presença de um produto de somas em que TODOS os factores do produto envolvem TODOS os literais da função. Cada uma das somas é um **maxtermo** ou **termo maximal** da função. A expressão, por seu turno, é um **produto de maxtermos**. Trata-se da **segunda forma canónica** ou **forma canónica conjuntiva** da função, que é única para F.

Cada função booleana simples é representável por uma e só uma forma *canónica* conjuntiva, embora possa ser representada algébricamente por diversas formas *normais* conjuntivas.

Os maxtermos estão associados com os "0"s da tabela de verdade da função e podem ser numerados. Assim, a função será

$$\begin{aligned} F &= (A+B+C)\,(A+B+\overline{C})\,(A+\overline{B}+C)\,(A+\overline{B}+\overline{C})\,(\overline{A}+B+\overline{C}) \\ &= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_5 \\ &= \prod M(0,1,2,3,5)\,. \end{aligned}$$

Notemos a forma como são "lidos" da tabela de verdade os maxtermos e os mintermos da função:

— enquanto que para a obtenção do índice de um mintermo contam as variáveis a 1 na tabela, para a formação de um maxtermo contam as variáveis s 0;

— a expressão de um mintermo é formada lendo cada variável a 1 e incluindo-a na expressão do mintermo na forma *não complementada*; por exemplo, o mintermo m_6 de F é formado pelo produto dos literais A, B e $\overline{C}$;

— a expressão de um mintermo é formada pelo produto de literais que resultam de se considerar todas as variáveis; por exemplo, a expressão do mintermo m_6 de F é $m_6 = A\,B\,\overline{C}$;

— a expressão de um maxtermo é formada lendo cada variável a 0 e incluindo-a na expressão do maxtermo na forma *complementada*; por exemplo, o maxtermo M_2 de F é formado pela soma dos literais A, $\overline{B}$ e C;

— a expressão de um maxtermo é formada pela soma de literais que resultam de se considerar todas as variáveis; por exemplo, a expressão do maxtermo M_2 de F é $M_2 = A + \overline{B} + C$.

Para terminar, vejamos a relação entre os mintermos da primeira forma canónica e os maxtermos da segunda forma canónica de uma dada função. Como vimos atrás para uma função arbitrária de 3 variáveis, o maxtermo $M_0 = A + B + C$ é o complemento do mintermo $m_0 = \overline{A}\,\overline{B}\,\overline{C}$, e vice-se versa. Na realidade esta igualdade estende-se a todo o par (m_i, M_i),

$$m_i = \overline{M_i} \qquad \text{e} \qquad M_i = \overline{m_i}\,, \qquad 0 \leq i \leq 2^n - 1\,,$$

para qualquer função booleana simples de n variáveis booleanas simples. Porém, não esquecer que, se a função possui m_i na sua forma canónica disjuntiva, não pode possuir M_i na sua forma canónica conjuntiva (a função vale 1 ou vale 0 na linha i da tabela de verdade, mas não pode valer simultaneamente 1 e 0 nessa linha).

4.6 Representação por Logigrama

Logigrama

Comecemos por considerar, na Figura 4.1, o **logigrama** correspondente à expressão em forma normal disjuntiva $F = A\,B + A\,\overline{C}$.

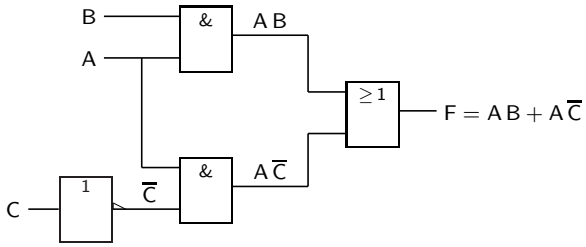
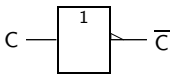


Figura 4.1: Logigrama correspondente à expressão em forma normal disjuntiva $F = A\,B + A\,\overline{C}$

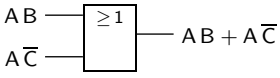
Devemos notar, no logigrama, a forma gráfica de várias portas lógicas, já conhecidas do Capítulo 3, que representam as funções lógicas envolvidas:

1. uma porta NOT, ou porta inversora, que implementa a função complementação que transforma a variável booleana simples de entrada, C, na função booleana simples $\overline{C}$ à saída;



2. duas portas AND com 2 entradas cada uma, uma que implementa a função produto lógico das variáveis booleanas simples A e B às entradas para dar a função booleana simples $A\,B$ à saída, e a outra que implementa a função produto lógico da variável booleana simples A e da função booleana simples $\overline{C}$ às entradas para dar a função booleana simples $A\,\overline{C}$ à saída; e

3. finalmente, uma porta OR que implementa a função soma lógica das funções booleanas simples $A\,B$ e $A\,\overline{C}$ às entradas para dar a função booleana simples $F = A\,B + A\,\overline{C}$ à saída do circuito.



Considere-se agora a representação por soma de mintermos. O logigrama correspondente será o que se ilustra na Figura 4.2.

Repare-se que cada mintermo é representado por uma porta lógica AND com 3 entradas (porque cada mintermo envolve 3 literais). Há, portanto, uma correspondência entre os "1"s da tabela de verdade de uma função booleana simples,

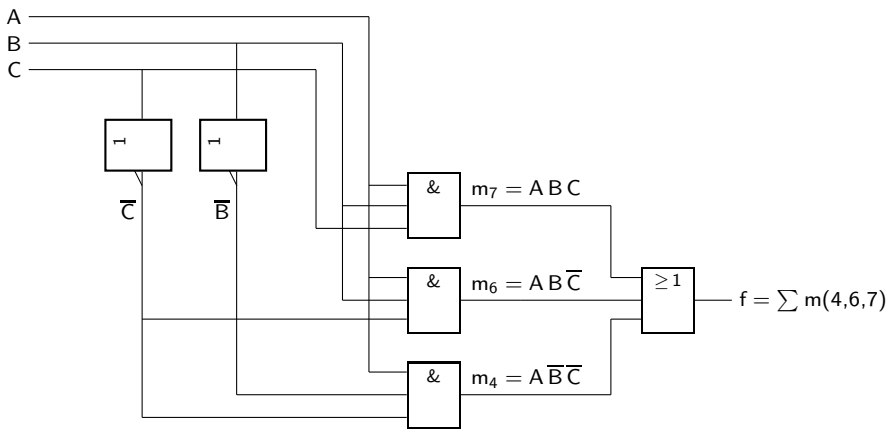


Figura 4.2: Logigrama correspondente à expressão em forma canónica disjuntiva $F = \sum m(4, 6, 7)$

os produtos na expressão booleana da primeira forma canónica da função, e as portas lógicas na representação gráfica (logigrama).

De notar ainda que os símbolos IEC das portas NOT se encontram numa posição que não é habitual. Tal decorre das regras muito rigorosas definidas na **norma IEC 61802-1** quanto ao posicionamento dos símbolos. Esta questão vem aprofundada no Apêndice B.

Norma IEC 61082-1

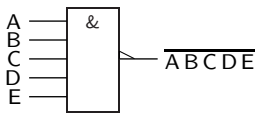
Dado conhecermos já os símbolos IEC das portas lógicas mais usuais, devemos agora perguntar quantas entradas podem essas portas possuir (no Capítulo anterior apenas se apresentaram símbolos para portas com 1, 2 e 3 entradas).

Obviamente com a excepção da porta NOT, que apenas tem uma entrada, qualquer porta lógica pode, teoricamente, ter o número de entradas que quisermos. Contudo, razões tecnológicas associadas ao fabrico dos circuitos integrados que as implementam fisicamente nos circuitos digitais (como veremos no Capítulo 6) fixam o número de entradas das portas a duas, três, quatro, cinco e, por vezes, oito. Mas não há nada de rígido nesta afirmação, e o aluno deve assegurar-se, caso a caso, que a porta lógica de que necessita possui o número de entradas pretendidas.

A título de exemplo, apresentam-se de seguida os símbolos de duas portas com um número de entradas pouco habitual:

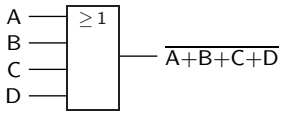
— uma porta NAND com 5 entradas:

Símbolo de uma porta NAND com 5 entradas

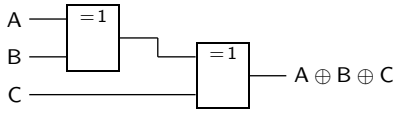


Símbolo de uma porta NOR com 4 entradas

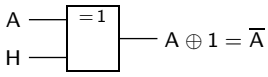
— uma porta NOR com 4 entradas:



Note-se, mais uma vez, que as portas XOR e XNOR em geral apenas possuem 2 entradas, e que as funções OU-exclusivo e Equivalência com mais do que 2 variáveis booleanas simples são construídas à custa de múltiplas portas com 2 entradas, utilizando as correspondentes propriedades associativas. Por exemplo, a função $A \oplus B \oplus C$ é formada da seguinte maneira:

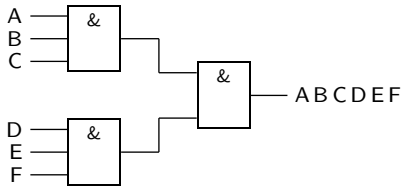


Por outro lado, utilizam-se muitas vezes os teoremas relativos ao OU-exclusivo da Subsecção 3.5.2 para formar negações de variáveis ou funções. Por exemplo, atendendo a que $A \oplus 1 = \overline{A}$, podemos substituir uma porta NOT da seguinte maneira:



Notemos como neste logigrama o valor lógico 1 vem representado por um *nível* H (algo que estudaremos mais tarde, quando se falar na lógica de polaridade no Capítulo 7).

Finalmente, há ocasiões em que necessitamos de uma porta com um elevado número de entradas mas não dispomos de tal porta. No caso em que a correspondente função booleana é associativa podemos fazer como anteriormente para a função Ou-exclusivo, como em



Mas atenção que as funções NAND e NOR não são associativas, pelo que a solução anterior não lhes é aplicável.

4.7 Importância das Funções NAND e NOR

Como se viu, qualquer função pode ser representada como uma soma de mintermos. Seja, por exemplo, a função anteriormente dada como exemplo:

$$F = A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C\,.$$

Como sabemos, a dupla negação não altera uma função (teorema da involução), donde:

$$F = \overline{\overline{A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C}}\,.$$

Mas, aplicando uma das leis de de Morgan obtemos

$$F = \overline{\overline{(A\,\overline{B}\,\overline{C})}\cdot\overline{(A\,B\,\overline{C})}\cdot\overline{(A\,B\,C)}}\,.$$

Ora, nesta expressão só surgem NANDs e NOTs:

1. o NAND com 3 entradas $\overline{A\,\overline{B}\,\overline{C}}$;
2. o NAND com 3 entradas $\overline{A\,B\,\overline{C}}$;
3. o NAND com 3 entradas $\overline{A\,B\,C}$;
4. o NAND global, ainda com 3 entradas, $\overline{\overline{(A\,\overline{B}\,\overline{C})}\cdot\overline{(A\,B\,\overline{C})}\cdot\overline{(A\,B\,C)}}$; e
5. os NOTs $\overline{B}$ e $\overline{C}$.

Por outro lado, um NOT pode ser feito com um NAND, uma vez que $\overline{A\cdot A} = \overline{A}$ ou ainda que $\overline{A\cdot 1} = \overline{A}$. Isso significa que podemos utilizar apenas NANDs na representação da função.

Do mesmo modo, e partindo da segunda forma canónica, pode-se mostrar que a função pode ser representada apenas por NORs.

E, naturalmente, o que é válido para a função F é válido para qualquer função booleana simples. Daí que os conjuntos {NAND} e {NOR} sejam conjuntos completos, tal como o é o conjunto {AND, OR, NOT}, como vimos anteriormente.

4.8 Referências Bibliográficas

Sêrro, Carlos — *Sistemas Digitais: Fundamentos Algébricos*, IST Press, Lisboa, 2003, Capítulo 5.

Arroz, G. S., Monteiro, J. C. e Oliveira, A. L. — *Introdução aos Sistemas Digitais e Microprocessadores*, Secções 3.1.10 a 3.1.13, e 3.2.

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 2.3 e 2.6.

4.9 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

4.1 Dada a expressão em forma normal conjuntiva da função booleana simples

$$f_1 = (\overline{a} + b)\,(a + c)\,(b + c)\,,$$

determinar a sua expressão mais simples em forma normal disjuntiva.

4.2 Dada a expressão em forma mista da função booleana simples

$$f_2 = a\,b + \overline{a}\,b\,c\,(x + y)\,,$$

colocá-la na forma: (i) de uma soma de produtos; e (ii) de um produto de somas.

(∗) 4.3 Desenhar as tabelas de verdade das seguintes funções booleanas simples,

a) $F_1(A, B, C) = \overline{A}\,B\,C + \overline{A}\,B\,\overline{C} + A\,C$;
b) $F_2(A, B, C) = A\,(B + \overline{C})\,(\overline{B} + C)$;
c) $F_3(A, B, C, D) = A\,[\overline{B} + \overline{C}\,(\overline{B} + D)]$;
d) $F_4(A, B, C) = \overline{A\,C} + B\,C$;
e) $F_5(A, B, C) = \overline{A\,(\overline{B}\,\overline{C} + B\,C)}$,

e identificar, para cada uma delas, as correspondentes formas canónicas.

4.4 Verificar, examinando todos os casos possíveis, que $A\,B + \overline{A}\,C + B\,C = A\,B + \overline{A}\,C$.

(∗) 4.5 Considere as seguintes funções booleanas simples:

a) $f(A, B, C) = (A \oplus B)\,C + \overline{A}\,(B \oplus C)$;
b) $f(A, B, C, D) = A + \overline{A}\,B\,C + C\,\overline{D} \oplus (C\,\overline{D} + \overline{C}\,D) + \overline{C}\,D \oplus (\overline{C}\,D + C\overline{D})$.

Escreva-as na forma normal conjuntiva (produto de somas) mais simples que conseguir.

4.6 Considere a função de 3 variáveis dada pela expressão

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

Escreva esta função na forma de uma soma de produtos.

4.7 Numere os seguintes mintermos e maxtermos:

(a) $A + B$; (b) $A\,\overline{C}$;
(c) $A\,B\,\overline{C}$; (d) $A + B + C$;
(e) $A\,\overline{B}\,C\,\overline{D}$; (f) $\overline{A} + B + B + \overline{D}$.

(∗) 4.8 Represente por uma soma de mintermos e por um produto de maxtermos a função

$$f(A, B, C) = (A + B)\,C + (A \oplus C)\,A\,B\,.$$

(∗) 4.9 Dada a função

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

obtenha:

a) a tabela de verdade;
b) a expressão em soma de mintermos;
c) a expressão em produto de maxtermos;
d) a expressão em soma de mintermos da função $\overline{f}(A, B, C, D)$.

(∗) 4.10 Utilizar o conjunto completo {AND,OR,NOT} para representar algebricamente (em somas de produtos) as seguintes funções booleanas simples:

a) $f_1 = \overline{(a \oplus b \oplus c)}\,\overline{a}$;
b) $f_2 = \overline{(a \odot b) \odot c}$.

(∗) 4.11 Representar as seguintes funções booleanas simples em primeira forma canónica:

a) $f_1 = \overline{(a \oplus b \oplus c)}\,\overline{a}$;
b) $f_2 = \overline{(a \odot b) \odot c}$.

(∗) 4.12 Representar as seguintes funções booleanas simples em segunda forma canónica:

a) $f_1 = \overline{(a \oplus b \oplus c)}\,\overline{a}$;
b) $f_2 = \overline{(a \odot b) \odot c}$.

4.13 Traçar os logigramas correspondentes às expressões dadas para as seguintes funções booleanas simples:

a) $f = a\,\overline{c} + b\,c + \overline{a}\,b\,c$;
b) $g = (a + b\,c)\,\overline{a}$;
c) $h = \prod M(2, 4, 6, 7)$.

(∗) 4.14 Considere o logigrama da Figura 4.3.

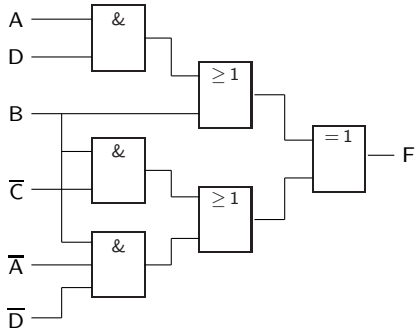


Figura 4.3: Logigrama utilizado no Exercício 4.14

Redesenhe-o da forma mais simples que conseguir.

(∗) 4.15 Usando apenas:

a) NANDs;
b) NORs;
c) AOIs,

desenhe o logigrama da seguinte função:

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

(*Nota*: **AOI** é a sigla de **"And-Or Invert"**. Ou seja, o logigrama deve apresentar um primeiro andar com portas AND e um segundo andar com uma porta NOR). *"And-Or Invert" (AOI)*

4.16 Modificar o logigrama do Exercício 4.14 por forma a apenas se usarem portas NANDs.

4.17 Modificar o logigrama do Exercício 4.14 por forma a apenas se usarem portas NORs.

4.18 Ponha as seguintes funções

a) $f = A\,(B + (C \oplus D)\,(A + \overline{B})) + \overline{A}\,B\,C + \overline{B}\,(C \oplus \overline{D})$;
b) $f = (A + B)\,(C + D)\,(\overline{A} + \overline{B})\,(\overline{A} + D)$;
c) $f = (A + \overline{B} + C)\,(\overline{C} + D)\,(A \oplus D)$,

na forma de:

a) uma soma de produtos;
b) um produto de somas.

Capítulo 5

Método de Karnaugh

5.1 Simplificação Algébrica de Funções Booleanas Simples

A simplificação algébrica pode ser utilizada para se obterem expressões booleanas mais simples (**expressões simplificadas**) para as funções, ou para se obterem expressões sob determinadas formas.

Expressões simplificadas

Como exemplo, consideremos a simplificação algébrica da seguinte função booleana simples,

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

Fazemos:

$$\begin{aligned} f(a,b,c) &= \overline{a}\,\overline{b}\,c + a\,c + b\,c \\ &= (\overline{a}\,\overline{b} + a + b)\,c \\ &= (\overline{b} + a + b)\,c \\ &= 1 \cdot c \\ &= c\,. \end{aligned}$$

Suponhamos agora que pretendemos obter, a partir da expressão de uma função booleana simples,

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

uma outra expressão que apenas possua operadores de duas variáveis e negações. Fazemos:

$$\begin{aligned} g(a,b,c,d) &= \overline{a}\,\overline{b}\,d + \overline{a}\,\overline{c}\,d + a\,c\,d + a\,\overline{b}\,\overline{d} \\ &= \overline{a}\,(\overline{b}\,d + \overline{c}\,d) + a\,(c\,d + \overline{b}\,\overline{d})\,. \end{aligned}$$

E se pretendermos colocar a expressão do exemplo anterior só em NANDs fazemos:

$$\begin{aligned} g(a,b,c,d) &= \overline{a}\,\overline{b}\,d + \overline{a}\,\overline{c}\,d + a\,c\,d + a\,\overline{b}\,\overline{d} \\ &= \overline{\overline{(\overline{a}\,\overline{b}\,d)} \cdot \overline{(\overline{a}\,\overline{c}\,d)} \cdot \overline{(a\,c\,d)} \cdot \overline{(a\,\overline{b}\,\overline{d})}}\,. \end{aligned}$$

5.2 Minimização de Funções Booleanas Simples

Expressões mínimas

O que se pretende agora é obter a **expressão ou expressões mínimas** de uma dada função. Naturalmente, o conceito de mínimo depende do critério que escolhermos. Usaremos como critério o número de termos na expressão (termos produto numa soma de produtos ou termos soma num produto de somas) e o número de literais nos termos.

Presume-se a representação de uma função a dois níveis, isto é, numa das suas formas normais ou canónicas, disjuntivas ou conjuntivas.

A minimização algébrica por utilização dos axiomas e teoremas da álgebra de Boole binária é possível (temos vindo a fazê-la) mas é, por vezes, difícil e carece de experiência. Preferimos, por isso, recorrer a um método semi-gráfico designado por método de Karnaugh.

5.3 Adjacências em Mapas de Karnaugh

Antes de passarmos ao método semi-gráfico de Karnaugh, iremos ver de seguida que há *métodos tabulares* interessantes para simplificar funções booleanas.

Como se viu anteriormente, uma função pode ser representada de várias formas. Por exemplo, a função $F = A\,B + A\,\overline{C}$ que temos vindo a usar pode ser representada pela tabela de verdade da Tabela 4.1 e que se reproduz, por comodidade, na Tabela 5.1.

Tabela 5.1: Tabela de verdade da função booleana simples $F(A, B, C) = A\,B + + A\,\overline{C}$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A partir desta tabela podemos "ler" a correspondente forma canónica disjuntiva,

$$F = A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C\,,$$

que já vimos que pode vir simplificada algébricamente para:

$$\begin{aligned} F &= A\,\overline{B}\,\overline{C} + A\,B\,\overline{C} + A\,B\,C \\ &= A\,\overline{C}\,(\overline{B} + B) + A\,B\,(\overline{C} + C) \\ &= A\,B + A\,\overline{C}\,. \end{aligned}$$

Do mesmo modo que a representação de funções pode ser feita indiferentemente utilizando uma forma tabular ou uma expressão booleana, o próprio processo de simplificação pode ser feito dos dois modos.

Reparemos, então, na forma como se chegou ao termo $A\,B$ na expressão acima. Isso conseguiu-se por junção dos mintermos $A\,B\,C$ e $A\,B\,\overline{C}$ que correspondem às linhas 6 e 7 (a **negrito**) na tabela.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	**1**	**0**	**1**
1	**1**	**1**	**1**

Reparemos que essas linhas têm uma particularidade: em ambas a função vale 1 (naturalmente) e, são as duas únicas linhas da tabela em que A e B valem simultaneamente 1. A diferenca entre as duas linhas está na variável C. Logo, pode-se concluir que, nesta função, basta A e B valerem 1 para a função também valer 1. Daí a conclusão que $F = A\,B + \cdots$.

Podia ser feito o mesmo raciocínio para o outro produto. Mas, nesse caso, as linhas em causa são a 4 e a 6.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	**0**	**0**	**1**
1	0	1	0
1	**1**	**0**	**1**
1	1	1	1

É fácil de ver que, agora, basta que A seja 1 e C seja 0 para se concluir que a função vale 1. Daí que também podemos escrever $F = A\,\overline{C} + \cdots$.

E como não há mais "1"s, podemos concluir que $F = A\,B + A\,\overline{C}$, como tinhamos visto anteriormente.

Podemos, então, simplificar directamente a função por observação da tabela *associando linhas em que a função tenha o valor* 1 *e que difiram apenas de uma variável*: no primeiro produto associamos as linhas 6 e 7, que diferem apenas na variável C, e no segundo associamos as linhas 4 e 6, que diferem apenas na variável B.

Linhas adjacentes

Destas linhas, que diferem apenas de uma variável, diz-se serem **adjacentes**.

É claro que era bom, como aconteceu no primeiro caso, que todas as linhas adjacentes estivessem fisicamente "encostadas". Mas, com 3 variáveis, cada linha tem sempre 3 linhas adjacentes e, na tabela, é impossível colocar fisicamente uma linha "encostada" a outras três.

Esta situação levou à alteração da tabela, de forma a que se satisfizessem estes requisitos. Esta nova forma de desenhar a tabela designa-se por **quadro de Karnaugh** ou **mapa de Karnaugh**.

Quadro (mapa) de Karnaugh

Quadro de Karnaugh com 3 variáveis

Na Figura 5.1 apresenta-se um **quadro de Karnaugh genérico** para uma função booleana simples de 3 variáveis (ou seja, um quadro em que não se representa, internamente, a função).

A \ BC	00	01	11	10
0		$A2$		
1	$A1$	P	$A3$	

Figura 5.1: Quadro de Karnaugh genérico de uma função de 3 variáveis onde se ilustram alguns quadrados adjacentes

Quadrados adjacentes

Repare-se que, para cada posição do mapa, há 3 **quadrados adjacentes**. Por exemplo a posição P tem, como posições adjacentes: (i) a posição $A1$, que se diferencia de P apenas por causa da variável C; (ii) a posição $A2$, que é diferente apenas na variável A; e (iii) e a posição $A3$, que é diferente apenas na variável B.

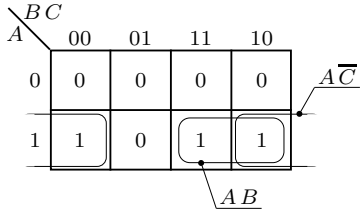
Repare-se que tudo se passa como se as posições laterais estivessem encostadas, o que se conseguiria com o mapa desenhado sobre um cilindro de eixo vertical (Figura 5.2).

A \ BC	00	01	11	10
0				$A2$
1	$A3$		$A1$	P

Figura 5.2: Quadro de Karnaugh de uma função de 3 variáveis onde se ilustram outros quadrados adjacentes

A função $F = A\,B + A\,\overline{C}$ que temos vindo a considerar está ilustrada no mapa de Karnaugh da Figura 5.3, que também mostra a leitura da sua expressão simplificada.

São válidas no mapa de Karnaugh todas as associações de 2 elementos adjacentes, não só no sentido formal do termo, mas também no de localização contígua no mapa.



$$F = A\,B + A\,\overline{C}$$

Figura 5.3: Quadro de Karnaugh da função $F = A\,B + A\,\overline{C}$, com leitura da sua expressão simplificada

Como é fácil de ver, a numeração das linhas da tabela de verdade da função passam para as correspondentes posições no quadro de Karnaugh pela ordem que se indica na Figura 5.4.

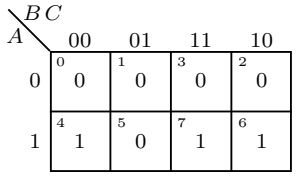
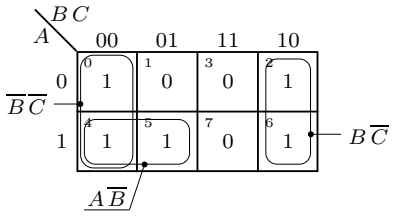


Figura 5.4: Quadro de Karnaugh de uma função booleana simples de 3 variáveis onde se identificam os seus 8 quadrados, numerando-os de 0 a 7, de acordo com os equivalentes numéricos dos números binários correspondente às quantidades booleanas gerais $(A, B, C) = (0, 0, 0)$ a $(A, B, C) = (1, 1, 1)$

Devemos acentuar que esta não é a única forma de desenhar o mapa de Karnaugh de uma função de 3 variáveis.

Consideremos outro exemplo: $F = \sum m(0, 2, 4, 5, 6)$ e o correspondente mapa de Karnaugh da Figura 5.5.



$$F = A\,\overline{B} + B\,\overline{C} + \overline{B}\,\overline{C}$$

Figura 5.5: Quadro de Karnaugh da função $F = \sum m(0, 2, 4, 5, 6)$, com os agrupamentos de unidades que dão uma expressão simplificada, $F = A\,\overline{B} + + B\,\overline{C} + \overline{B}\,\overline{C}$

É fácil de ler do mapa a expressão

$$F = A\,\overline{B} + B\,\overline{C} + \overline{B}\,\overline{C}\,.$$

Mas também é fácil de ver que os dois termos finais se podem simplificar,

obtendo-se para a função a expressão mais simples

$$F = A\,\overline{B} + \overline{C}\,.$$

Ora esta expressão é igualmente legível no mapa de Karnaugh. De facto, os dois grupos de dois "1"s nas extremidades do mapa são adjacentes e podem juntar-se entre si para originar o quadro da Figura 5.6, onde é fácil agora ler directamente a expressão mais simplificada.

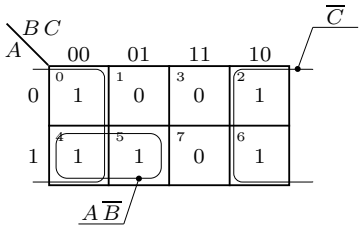


Figura 5.6: Quadro de Karnaugh da função $F = \sum m(0,2,4,5,6)$, com os agrupamentos de unidades que dão uma expressão mais simplificada, $F = A\,\overline{B} + \overline{C}$

5.4 Quadros de Karnaugh com 4 Variáveis

O mapa de Karnaugh anterior foi apresentado para 3 variáveis booleanas simples. Contudo, pelo menos em teoria, podemos construir um mapa com um número arbitrário de variáveis. Na prática, porém, com mais de 6 variáveis torna-se bastante difícil a utilização dos mapas de Karnaugh e, além disso, há outros métodos de minimização mais eficazes (que, no entanto, assentam nos mesmos princípios do método agora apresentado).

Quadro de Karnaugh com 4 variáveis

Um **quadro com** 4 **variáveis** constrói-se facilmente. A partir de um mapa de 3 variáveis, replica-se o quadro por reflexão num espelho imaginário, como se sugere na Figura 5.7.

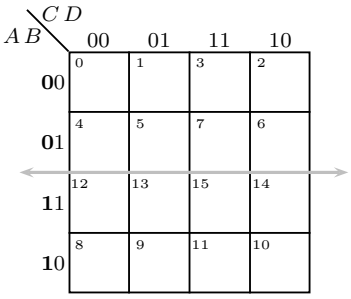


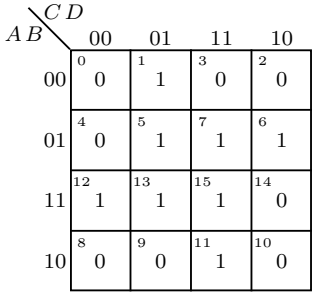
Figura 5.7: Quadro de Karnaugh para uma função booleana simples com 4 variáveis booleanas simples

Repare-se na variável assinalada a negrito que é a que distingue os dois lados do "mapa". Repare-se, ainda, na numeração das diversas posições.

Uma vez construído um quadro de 4 variáveis, vejamos agora como podemos inserir no mapa uma função com o mesmo número de variáveis, por exemplo

$$F = \sum m(1, 5, 6, 7, 11, 12, 13, 15)\,.$$

O quadro da função vem ilustrado na Figura 5.8.



$F = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$

Figura 5.8: Quadro de Karnaugh da função $F = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$

Passemos agora à minimização desta função.

À primeira vista, poderia parecer interessante juntar os 4 "1s" centrais num grupo único. No entanto, para cobrir os restantes "1s" haveria a necessidade de associar cada um deles com um dos "1s" do grupo central, como se ilustra na Figura 5.9(a).

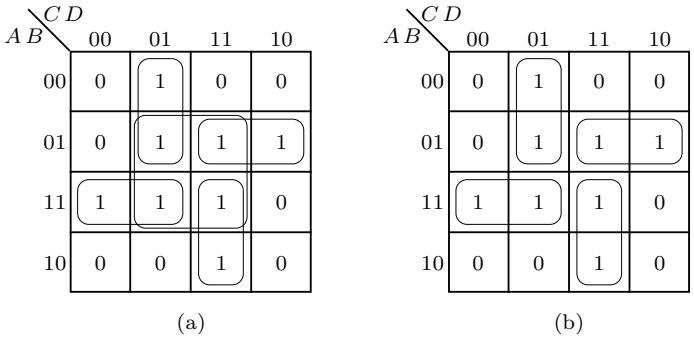


Figura 5.9: (a) Quadro de Karnaugh da função anterior, com uma tentativa de agrupamentos que não é a mais simples; (b) remove-se o grupo central, por ser redundante

Verifica-se, portanto, que o grupo central não é necessário. Restam os outros grupos, como se indica na Figura 5.9(b). A leitura deste quadro é fácil (Figura 5.10), conduzindo à seguinte expressão mínima em soma de produtos:

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

Notemos como se lê cada um dos agrupamentos. Para tanto, vamos dar dois exemplos.

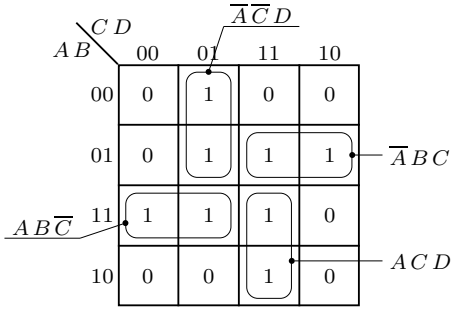


Figura 5.10: Leitura do quadro de Karnaugh da função anterior

1. Grupo $A\,B\,\overline{C}$ — a expressão deste agrupamento resulta do facto de os literais A e B e $\overline{C}$ se manterem com um valor constante dentro do grupo; quanto à variável D, não aparece na expressão porque não se mantém constante dentro do grupo.

2. Grupo $\overline{A}\,B\,C$ — os literais $\overline{A}$, B e C mantêm-se constantes dentro do grupo, enquanto que a variável D desaparece porque varia dentro do grupo.

5.5 Implicantes e Implicantes Primos

Para um pouco mais de aprofundamento sobre o método de Karnaugh que nos permita perceber melhor o que realmente estamos a fazer e, portanto, traçar tácticas adequadas, vamos dar algumas definições:

Função implicação

Para funções das mesmas variáveis, diz-se que uma função booleana simples $F1$ **implica** outra função $F2$ quando, para todas as quantidades booleanas gerais de entrada em que a função $F1$ vale 1, a função $F2$ também vale 1. Quando $F1$ é um produto, diz-se ser um **implicante** da função $F2$.

Implicante

No nosso caso, por exemplo, $A\,C\,D$ é um implicante da função F. Repare-se que todos os mintermos de uma função F são implicantes dessa função.

Nos quadros de Karnaugh, *os implicantes correspondem a associações válidas de "1"s*. No mapa da Figura 5.11, indicam-se alguns implicantes da função de 3 variáveis, $F = \sum m(0, 2, 4, 5, 6)$.

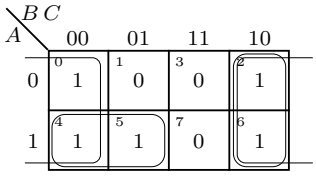


Figura 5.11: Quadro de Karnaugh da função de 3 variáveis $F = \sum m(0, 2, 4, 5, 6)$ com alguns implicantes da função

Repare-se que o mintermo m_6, por exemplo, para além de ser um implicante da função é também um implicante do produto que resulta da associação dos mintermos m_2 e m_6, isto é, $B\,\overline{C}$.

Repare-se, ainda, que este produto é implicante do produto correspondente à associação dos mintermos m_0, m_2, m_4 e m_6, isto é, $\overline{C}$.

Por fim, constate-se que este último implicante não implica mais nenhum produto, implicando apenas a função.

Ao implicante de uma função booleana simples que não implica nenhum outro implicante chama-se **implicante primo** da função. A importância dos implicantes primos decorre de eles corresponderem, no mapa de Karnaugh, aos *grupos maiores*, que não podem ser expandidos. Ora acontece que esses são exactamente os grupos que nos interessam para as minimizações, já que possuem as expressões mais simples, com menos literais.

Implicante primo

A expressão algébrica minimizada de uma função expressa em soma de produtos é sempre uma **soma de implicantes primos** da função.

Soma de implicantes primos

Retome-se o exemplo anterior da função de 4 variáveis,

$$F = \sum m(1, 5, 6, 7, 11, 12, 13, 15)\,.$$

Na Figura 5.9(a) estão assinalados todos os implicantes primos de F. Porém, como vimos, nem todos os implicantes primos da função foram usados na expressão minimizada, como se viu no quadro da Figura 5.9(b) e que se repete na Figura 5.12, por comodidade.

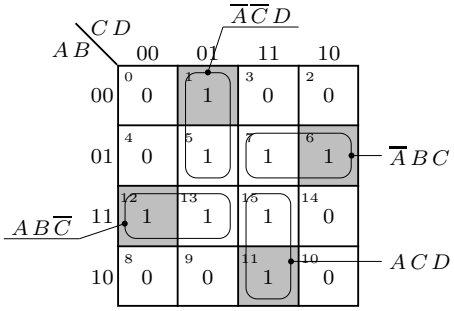


Figura 5.12: (a) Quadro de Karnaugh da função booleana simples $F = \sum m(1, 5, 6, 7, 11, 12, 13, 15)$ que assinala os implicantes primos da função por intermédio dos quadrados assinalados a cinzento

Repare-se que cada um dos implicantes primos usados tem uma particularidade fundamental:

O implicante *primo* $A\,C\,D$, por exemplo, é o único que associa o mintermo correspondente ao quadrado 11 (assinalado a cinzento), isto é, m_{11}.

Do mesmo modo, cada um dos outros implicantes primos associa um mintermo que não pode ser associado de outra forma. Assim, o implicante $A\,B\,\overline{C}$ é *primo* por causa do mintermo m_{12}, no quadrado 12 assinalado a cinzento; o implicante $\overline{A}\,\overline{C}\,D$ é *primo* por causa do mintermo m_1, no quadrado 1 assinalado a

cinzento; e o implicante $\overline{A}\,B\,C$ é *primo* por causa do mintermo m_6, no quadrado 6 assinalado a cinzento.

Ao contrário, o implicante primo correspondente às posições centrais do mapa (que acabou por não ser usado), associa mintermos que podem ser associados de outra forma, pelo que não é um implicante fundamental para a minimização da função.

Implicante primo essencial

Os implicantes primos que associam mintermos que não podem ser associados em implicantes primos de outra forma, são designados por **implicantes primos essenciais** da função.

Quadrado essencial

E os quadrados que contêm mintermos que tornam essencial um dado implicante primo (como os assinalados a cinzento na Figura 5.12) designam-se por **quadrados essenciais**.

Como vimos, a expressão algébrica de uma função em termos de soma de produtos é uma soma de implicantes primos. Mas não têm que ser usados todos os implicantes primos. No entanto, *todos os implicantes primos essenciais têm de estar presentes na expressão mínima.*

No exemplo anterior, os implicantes primos $A\,C\,D$, $A\,B\,\overline{C}$, $\overline{A}\,\overline{C}\,D$ e $\overline{A}\,B\,C$ são implicantes primos essenciais da função, e a expressão minimizada é a soma desses implicantes.

Convém referir que, embora neste exemplo os únicos implicantes usados sejam implicantes primos essenciais, isso não é uma regra geral. Com efeito, repare-se na função da Figura 5.13.

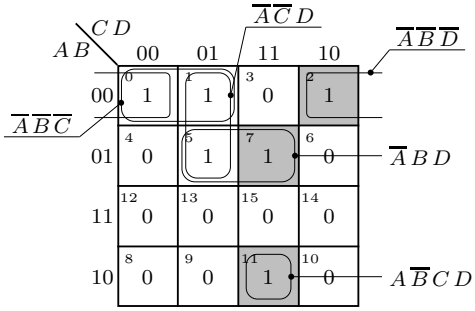


Figura 5.13: (a) Quadro de Karnaugh de uma função que possui implicantes primos essenciais (devido aos quadrados assinalados a cinzento) e não essenciais

Neste exemplo, os implicantes primos são os seguintes:

1. $\overline{A}\,\overline{B}\,\overline{D}$ — é um implicante primo *essencial* por ser o único implicante primo a associar o mintermo m_2, correspondente ao quadrado 2 no mapa (quadrado essencial);
2. $\overline{A}\,B\,D$ — é um implicante primo *essencial* por ser o único implicante primo a associar o mintermo m_7, correspondente ao quadrado 7 no mapa (quadrado essencial);
3. $A\,\overline{B}\,C\,D$ — é um implicante primo *essencial* por ser um mintermo (o mintermo m_{11}) que não pode ser associado com qualquer outro mintermo (quadrado essencial);

4. $\overline{A}\,\overline{B}\,\overline{C}$ — é um implicante primo *não essencial* porque ambos os mintermos nele associados podem ser associados de outras formas;

5. $\overline{A}\,\overline{C}\,D$ — é um implicante primo *não essencial* porque ambos os mintermos nele associados podem ser associados de outras formas.

Neste caso, a expressão mínima da função incluirá todos os implicantes primos essenciais e um dos não essenciais, dessa forma se associando todos os mintermos da função. Há, portanto, duas expressões minimas para a função:

$$F = \overline{A}\,\overline{B}\,\overline{D} + \overline{A}\,B\,D + A\,\overline{B}\,C\,D + \overline{A}\,\overline{B}\,\overline{C}\,,$$

e

$$F = \overline{A}\,\overline{B}\,\overline{D} + \overline{A}\,B\,D + A\,\overline{B}\,C\,D + \overline{A}\,\overline{C}\,D\,.$$

Convém também chamar a atenção para o facto de que podem existir funções sem implicantes primos essenciais na sua soma de produtos mínima.

Com efeito, considere-se na Figura 5.14 o exemplo da função

$$G = \sum m(0, 1, 5, 7, 8, 10, 14, 15)\,.$$

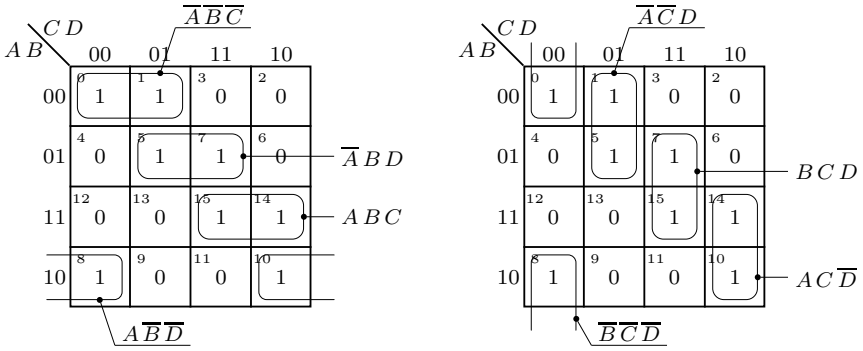


Figura 5.14: Quadro de Karnaugh de uma função, G, que não possui implicantes primos essenciais e que tem duas somas de produtos mínimas

Como podemos constatar, esta função possui duas expressões minimas,

$$G = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,D + A\,B\,C + A\,\overline{B}\,\overline{D}$$

e

$$G = \overline{A}\,\overline{C}\,D + B\,C\,D + A\,C\,\overline{D} + \overline{B}\,\overline{C}\,\overline{D}\,,$$

e não tem quaisquer implicantes primos essenciais.

5.6 Minimização 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.

Estudemos um exemplo, ao longo do qual se explora a metodologia a usar. Consideremos que se pretende obter uma função que tem como variáveis de entrada os quatro bits de uma representação em código BCD, e que deve gerar na saida um 1 apenas quando o número representado for múltiplo de 3.

Podemos construir a tabela de verdade da função pretendida (Tabela 5.2). As variáveis de entrada são $A3$, $A2$, $A1$ e $A0$, que representam os quatro bits do código BCD ($A3$ tem o maior peso).

Tabela 5.2: Tabela de verdade de uma função booleana simples detectora de dígitos BCD que são múltiplos de 3

BCD	A3	A2	A1	A0	F	**Observações**
0	0	0	0	0	0	Não é múltiplo de 3
1	0	0	0	1	0	Não é múltiplo de 3
2	0	0	1	0	0	Não é múltiplo de 3
3	0	0	1	1	1	É múltiplo de 3
4	0	1	0	0	0	Não é múltiplo de 3
5	0	1	0	1	0	Não é múltiplo de 3
6	0	1	1	0	1	É múltiplo de 3
7	0	1	1	1	0	Não é múltiplo de 3
8	1	0	0	0	0	Não é múltiplo de 3
9	1	0	0	1	1	É múltiplo 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

Repare-se que temos três situações diferentes:

— dígitos BCD que são múltiplos de 3 ($F = 1$);

— dígitos BCD que não são múltiplos de 3 ($F = 0$);

— configurações de entrada que não são dígitos BCD (assinaladas com os símbolos × na tabela), indicadoras de **indiferenças**.

Indiferença

Neste último caso não é importante considerar o valor da função, uma vez que as configurações de entrada respectivas nunca ocorrem e, portanto, o valor que a função teria nessa situação é indiferente.

Inicialmente não vamos ter isso em conta e vamos obter a expressão mínima da função com valores 0 nessas posições (a opção mais conservadora), como se ilustra na Figura 5.15.

Obtemos, neste caso,

$$F = \overline{A3}\,\overline{A2}\,A1\,A0 + \overline{A3}\,A2\,A1\,\overline{A0} + A3\,\overline{A2}\,\overline{A1}\,A0\,.$$

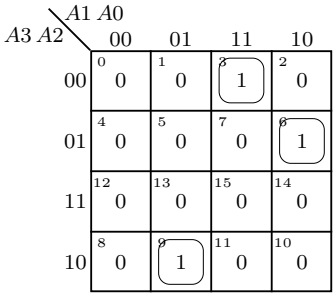


Figura 5.15: Quadro de Karnaugh de uma função booleana simples que detecta dígitos BCD são múltiplos de 3; admite-se que se colocam "0"s nos quadrados 10 a 15, que correspondem a valores nas entradas que não são dígitos BCD

Mas experimentemos agora protelar para mais tarde a atribuição de valores à função nas configurações que não correspondem a dígitos BCD, colocando indiferenças no quadro (Figura 5.16).

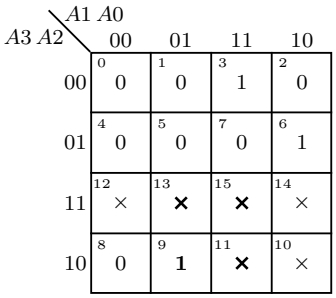


Figura 5.16: Quadro de Karnaugh de uma função booleana simples que detecta dígitos BCD que são múltiplos de 3; agora incluem-se indiferenças nos quadrados que correspondem a quantidades booleanas gerais de entrada que não são dígitos BCD

Considere-se, por exemplo, o mintermo m_9, a negrito. Se nas posições correspondentes às indiferenças o valor da função fosse 1, poder-se-iam associar os quatro mintermos a negrito, simplificando consideravelmente a expressão. Mas como, de facto, o valor que a função toma nestas posições é indiferente, porque não colocar nesses quadrados os valores que mais nos convêm? O mesmo acontece para outras posições no mapa.

Podemos, então, fazer a minimização que se indica na Figura 5.17.

E, neste caso, obtemos a seguinte soma de produtos mínima para a função:

$$F = A3\,A0 + A2\,A1\,\overline{A0} + \overline{A2}\,A1\,A0\,.$$

Como se vê, é muito mais simples a expressão obtida por recorrermos à flexibilidade que existe por serem certas posições indiferentes.

Tenha-se em conta que a função descrita pela expressão obtida deixou de ter posições não definidas. As configurações de variáveis de entrada correspondentes às indiferenças que foram associadas com mintermos da função passaram

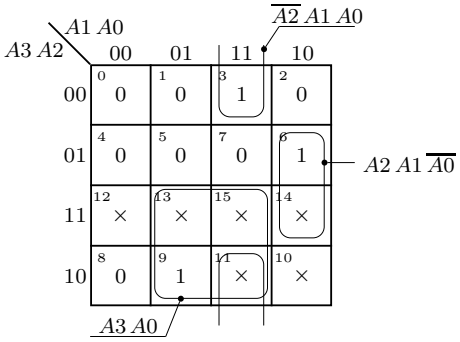


Figura 5.17: Quadro de Karnaugh de uma função booleana simples que detecta dígitos BCD que são múltiplos de 3, e correspondente minimização quando se consideram as indiferenças

a provocar uma saida igual a 1 para a função. E as que não foram associadas passaram a gerar uma saida igual a 0.

Funções completa e incompletamente especificadas

Esta técnica de utilização das indiferenças é generalizadamente usada para minimizar as expressões das funções lógicas, que passam então a designar-se por **funções incompletamente especificadas** ou **funções incompletas**, por oposição às **funções completamente especificadas** ou **funções completas**, que não possuem indiferenças.

5.7 Quadros de 5 Variáveis

A obtenção de mapas de Karnaugh de 5 variáveis faz-se a partir de um mapa de 4 variáveis, da mesma forma que este se obteve a partir de um mapa de 3 variáveis.

Quadro de Karnaugh com 5 variáveis

Ilustra-se na Figura 5.18 uma estrutura possível de um **quadro de 5 variáveis**, mostrando também a numeração das diversas posições admitindo que a variável de maior peso é A e a de menor peso é E.

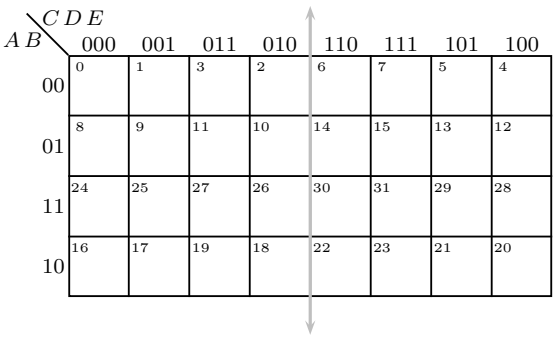


Figura 5.18: Estrutura de um quadro de Karnaugh de 5 variáveis, em que A é a variável de maior peso

Para além de todas as adjacências válidas num mapa de 4 variáveis (em cada uma das "metades") existem agora adjacências entre posições simétricas em relação ao eixo de simetria vertical.

Vejamos um exemplo. Consideremos a função

$$F = \sum m(0, 2, 6, 9-11, 13, 14, 16, 18, 20, 24, 31)$$

com indiferenças nas posições 1, 4, 22 e 27, o que se pode representar, em alternativa, da seguinte forma

$$F = \sum m(0, 2, 6, 9-11, 13, 14, 16, 18, 20, 24, 31) + \sum m_d(1, 4, 22, 27)\,,$$

(notar que $9-11$ é o mesmo que $9, 10, 11$).

O preenchimento do quadro permite obter a Figura 5.19.

$A\,B$ \ $C\,D\,E$	000	001	011	010	110	111	101	100
00	0: 1	1: ×	3: 0	2: 1	6: 1	7: 0	5: 0	4: ×
01	8: 0	9: 1	11: 1	10: 1	14: 1	15: 0	13: 1	12: 0
11	24: 1	25: 0	27: ×	26: 0	30: 0	31: 1	29: 0	28: 0
10	16: 1	17: 0	19: 0	18: 1	22: ×	23: 0	21: 0	20: 1

Figura 5.19: Quadro de Karnaugh da função booleana simples $F = \sum m(0, 2, 6, 9-11, 13, 14, 16, 18, 20, 24, 31) + \sum m_d(1, 4, 22, 27)$

Vamos mostrar como devemos minimizar esta função, começando por identificar os implicantes primos essenciais (a cinzento na Figura 5.20).

Como esta função possui vários implicantes primos, essenciais e não essenciais, optou-se por gerá-los em mais de um quadro. Por outro lado, optou-se por não incluir os "0"s no mapa, visto que apenas vamos tratar dos "1"s e das indiferenças. Esta é uma prática usual, que seguiremos daqui para a frente (excepto, claro está, quando estivermos a lidar com os "0"s e com as indiferenças, como faremos no parágrafo seguinte, caso esse em que eliminaremos os "1"s do quadro).

Os implicantes primos essenciais são:

— o assinalado com IPE1, com a expressão $\overline{B}\,\overline{E}$, pois os mintermos m_{18} e m_{20} não têm outra forma de se associar em outros implicantes primos;

— O assinalado com IPE2, com a expressão $\overline{A}\,B\,\overline{D}\,E$, por causa do mintermo m_{13};

— O assinalado com IPE3, com a expressão $\overline{A}\,D\,\overline{E}$, por causa do mintermo m_{14};

— O assinalado com IPE4, com a expressão $A\,B\,D\,E$, por causa do mintermo m_{31};

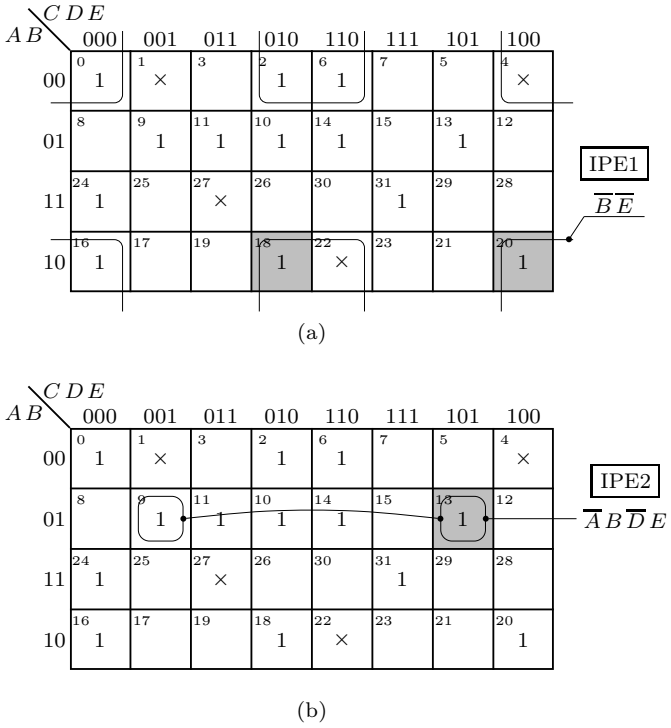


Figura 5.20: Minimização da função booleana simples $F = \sum m(0, 2, 6, 9 - -11, 13, 14, 16, 18, 20, 24, 31) + \sum m_d(1, 4, 22, 27)$

— O assinalado com IPE5, com a expressão $A\,\overline{C}\,\overline{D}\,\overline{E}$, por causa do mintermo m_{24}.

Repare-se que o grupo formado pela soma de mintermos $m_9 + m_{10} + m_{11} + m_{14}$, embora muito tentador, não é correcto.

Uma soma de produtos mínima é

$$F = \overline{B}\,\overline{E} + \overline{A}\,B\,\overline{D}\,E + \overline{A}\,D\,\overline{E} + A\,B\,D\,E + A\,\overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,B\,\overline{C}\,E\,,$$

mas devemos notar que esta expressão não é única.

5.8 Minimização Usando os Maxtermos

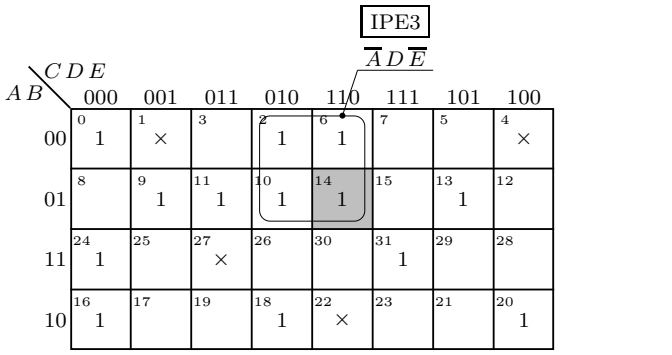
Do mesmo modo que se usaram, até agora, no contexto do método de Karnaugh, associações de mintermos para obter expressões em somas de produtos (forma normal disjuntiva), é igualmente possível associar maxtermos, obtendo-se expressões em produtos de somas (forma normal conjuntiva).

Produto de implicados primos

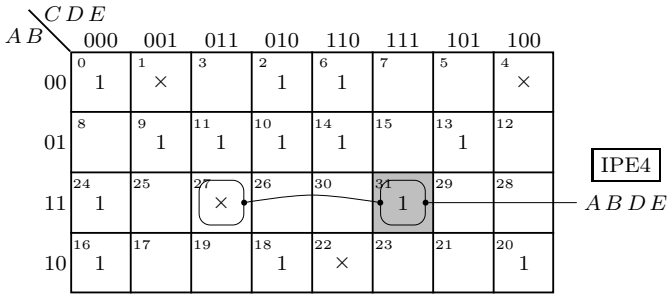
Implicado primo

Implicado

A expressão algébrica minimizada de uma função expressa em produto de somas é sempre um **produto de implicados primos**, entendendo-se por **implicado primo** um **implicado** da função (ou que é implicado pela função) e que não é implicado por nenhum outro implicado da função.



(c)



(d)

Figura 5.20: *cont.*

Recorde-se que os maxtermos são identificados por "0"s nos mapas de Karnaugh (ou nas tabelas de verdade) das funções, pelo que os implicados primos corresponderão aos agrupamentos maiores que é possível fazer com os "0"s e que obedecem às regras de formação de agrupamentos no método de Karnaugh.

Estudemos então um exemplo. Seja

$$F = \prod M(1,3,9,11,12,13,14), \quad \text{com indiferenças nas posições 4, 6 e 8},$$

o que também se representa, de forma condensada, por

$$F = \prod M(1,3,9,11,12,13,14) \cdot \prod M_d(4,6,8)\,.$$

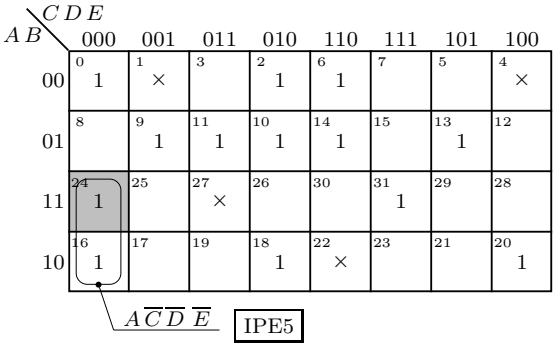
O mapa de Karnaugh para esta função tem a forma que se indica na Figura 5.21.

É de notar que o facto de se representar a função como um produto de maxtermos na sua especificação não significa que ela tenha de ser minimizada (ou simplificada) para se obter uma expressão na forma de produto de somas. A especificação inicial é uma coisa, e a forma como se opta por obter a expressão da função é outra, independente.

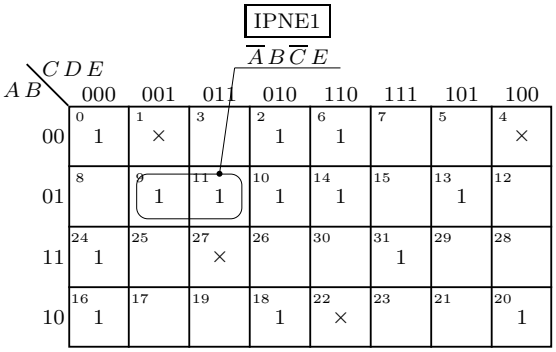
O produto de somas mínimo que se obtém para F é:

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

acentuando-se que todos os implicados primos são **essenciais**: *Implicado primo essencial*



(e)



(f)

Figura 5.20: *cont.*

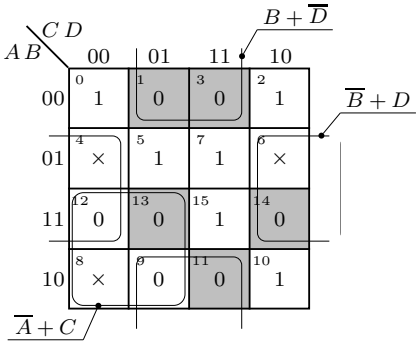


Figura 5.21: Minimização da função $F = \prod M(1,3,9,11,12,13,14) \cdot \cdot \prod M_d(4,6,8)$

— $\overline{B} + D$ por causa do maxtermo M_{14};

— $B + \overline{D}$ por causa dos maxtermos M_1, M_3 e M_{11}; e

— $\overline{A} + C$ por causa do maxtermo M_{13}.

Recorde-se que, quando se "lêem" somas nos mapas de Karnaugh (ou nas tabelas

de verdade), as variáveis que se mantêm em 0 são lidas na forma não negada e as que se mantêm em 1 são lidas na forma negada.

5.9 O Algoritmo de Karnaugh

O método de Karnaugh de minimização de funções booleanas simples pode ser implementado pelo algoritmo que se esboça a seguir:

1. se se quiserem obter apenas a ou as somas de produtos mínimas para a função, prosseguir com o passo 2; se se pretende obter apenas o produto ou produtos de somas mínimos para a função, ir para o passo 6;
2. estabelecem-se *todos os implicantes primos essenciais*, e incluem-se na soma de produtos mínima da função;
3. escolhe-se *o menor número de implicantes primos não essenciais que cobrem todos os "1"s do quadro*; se apenas existir uma solução, continuar no passo 5;
4. de entre as várias alternativas obtidas no passo anterior, escolhem-se as soluções que contêm *o menor número de implicantes primos não essenciais com o menor número de literais que cobrem todos os "1"s do quadro* (ou seja, de entre as alternativas obtidas no passo 3, escolhem-se as mais simples);
5. na soma de produtos mínima da função, somam-se logicamente os implicantes primos essenciais obtidos no passo 2 com os implicantes primos não essenciais obtidos nos passos 3 e 4; o algoritmo termina aqui;
6. repetir os passos 2 a 5 para os "0"s do quadro, obtendo-se implicados primos em vez de implicantes primos e produtos de somas mínimos em vez de somas de produtos mínimas.

Deve realçar-se que os passos 3 e 4 podem gerar mais do que uma solução, existindo, nesses casos, mais do que uma soma de produtos mínima (ou um produto de somas mínimo) para a função.

Para finalizar, refere-se que este algoritmo (e todo o método de Karnaugh) é de difícil aplicação para funções com mais do que 5 ou 6 variáveis booleanas simples, dada a dimensão dos quadros necessários e a dificuldade na percepção de todas as adjacências, com a consequente dificuldade na geração dos implicantes ou implicados primos.

Por outro lado, o algoritmo pode ser extendido para funções booleanas gerais, mas nesse caso o número de funções booleanas simples envolvidas deve ser pequeno, no máximo de 3. Neste curso não iremos desenvolver esta possibilidade, que deixamos para as referências adequadas.

5.10 Referências Bibliográficas

Sêrro, Carlos — *Sistemas Digitais: Fundamentos Algébricos*, IST Press, Lisboa, 2003, Capítulo 7.

Arroz, G. S., Monteiro, J. C. e Oliveira, A. L. — *Introdução aos Sistemas Digitais e Microprocessadores*, Secções 3.1.8 e 3.3.1.

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 2.4 e 2.5.

5.11 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 5.1 Nas tabelas de verdade das funções booleanas simples

a) $f_1(A,B,C) = (A+B)\,C + \overline{A}\,(B+C)$; e
b) $f_2(A,B,C,D) = A + \overline{A}\,B\,C + C\,\overline{D} \oplus (C\,\overline{D} + \overline{C}\,D)$,

identificar todas as linhas adjacentes às linhas em que as funções têm o valor 1.

5.2 Dado, no quadro de Karnaugh da função booleana simples $F(A,B,C)$, o quadrado correspondente à quantidade booleana geral $(A,B,C) = (1,1,1)$, quantos quadrados lhe são adjacentes? E para o quadrado $(A,B,C,D,E) = = (0,1,1,1,0)$ no quadro da função booleana simples $F(A,B,C,D,E)$?

(∗) 5.3 Identificar todos os agrupamentos legítimos de dois "1"s no quadro de Karnaugh da função booleana simples $F(A,B,C) = \sum m(1-7)$, e indicar as correspondentes expressões booleanas.

5.4 Identificar todos os agrupamentos máximos no quadro de Karnaugh da função booleana simples $F(A,B,C,D) = \sum m(1-4,10,12-14)$.

(∗) 5.5 Dada a função $F = \sum m(0-2,4-7,10)$, dizer se os seguintes mintermos e somas de mintermos são ou não implicantes de F:

a) m_1 ;
b) m_3 ;
c) $m_1 + m_2$;
d) $m_1 + m_3$;
e) $m_0 + m_1 + m_2$;
d) $m_4 + m_5 + m_6 + m_7$.

(∗) 5.6 Identificar todos os implicantes primos essenciais das funções booleanas simples que se seguem (admita que A é a variável booleana simples com maior peso):

a) $F_1(A,B,C,D) = \sum m(0-2,4-7,10)$;
b) $F_2(A,B,C,D) = A\,B\,C + A\,\overline{B}\,D + B\,C + D$,

e identificar os correspondentes quadrados essenciais. Identificar ainda, para cada uma das funções, pelo menos 2 implicantes primos não essenciais.

5.7 Escrever a ou as formas normais disjuntivas mínimas para as funções booleanas simples que se seguem (A é a variável booleana simples com maior peso):

a) $F(A,B,C,D) = \sum m(0-2, 4-7, 10)$;
b) $G(A,B,C,D) = A\,B\,C + A\,\overline{B}\,D + B\,C + D$.

(∗) 5.8 Escrever a ou as formas normais disjuntivas mínimas para a função booleana simples

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

tendo em conta que nunca surgem as combinações de valores nas entradas correspondentes aos mintermos 1, 4, 7, 10 e 11.

(∗) 5.9 Porque é que o agrupamento formado pela soma de mintermos $m_9 + m_{10} + m_{11} + m_{14}$ num quadro de Karnaugh de 4 variáveis está incorrecto? E num quadro com $n > 4$ variáveis?

5.10 Obter a ou as somas de produtos mínimos para a função

$$f = \sum m(0, 1, 4-7, 9, 12, 14-17, 20, 21, 25, 28, 30)\,.$$

5.11 Que outras somas de produtos mínimos existem para a função

$$F = \sum m(0, 2, 6, 9-11, 13, 14, 16, 18, 20, 24, 31) + \sum m_d(1, 4, 22, 27)$$

estudada no texto?

5.12 Obter a ou as somas de produtos mínimos para a função

$$f = \sum m(2, 6-10, 13, 18, 23, 25, 29)\,,$$

com indiferenças nas posições 0, 3, 11, 21, 26 e 27.

5.13 a) Minimize a função

$$F = \prod M(1, 2, 5, 8, 9, 11, 12, 15, 17, 19, 21, 23-25, 28-31)\,,$$

com indiferenças em $6, 7, 10, 14, 16, 18, 26$ e 27, por forma a facilitar uma futura implementação com NAND's com qualquer número de entradas.

b) Indique se existe algum implicado primo essencial e, em caso afirmativo, escreva a ou as correspondentes expressões lógicas.

5.14 Minimizar as seguintes funções booleanas simples (admita que A é a variável booleana simples com maior peso):

a) $f(A,B,C,D) = \prod M(3, 4, 6, 7, 11, 12, 14)$;
b) $f(A,B,C,D) = \prod M(0, 2-8, 10, 12, 13)$.

5.15 Minimizar as seguintes funções booleanas simples, em que A é a variável com maior peso:

a) $f(A,B,C,D) = \sum m(2-5, 8, 9, 14, 15)$;
b) $f(A,B,C,D,E) = \sum m(0, 1, 3, 8, 9, 13-17, 19, 24, 25, 27, 31)$.

5.16 Minimizar a seguinte função booleana simples:

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

5.17 Minimize as seguintes funções booleanas simples:

a) $f = \sum m(0,2,6,8-10,12) + \sum m_d(1,3,4,11,13,14)\,;$
b) $f = \sum m(1,2,6,9,13-15,17,22,25,29-31) + \sum m_d(7,8,18,23)\,.$

(∗) 5.18 Minimizar a seguinte função booleana simples:

$$f = \sum m(1,3,5,6,9,12,17,19,22,27,28,30) + \sum m_d(4,11,14,20,21,25)\,.$$

5.19 Minimize a seguinte função booleana simples,

$$f = \sum m(0,4,6,11,12,14-16,24,31) + \sum m_d(2,8,27,28)\,,$$

e escreva a expressão booleana de um implicante primo da função.

5.20 a) Minimize a função

$$x = \prod M(7,10,14,15,17,23-26,30,31)\,,$$

com indiferenças nas posições 1, 6, 8, 9, 12, 20, 21 e 22, e desenhe o seu logigrama usando apenas NANDs de 2 entradas. Dispõe, nas entradas, das variáveis complementadas e não complementadas.

b) Nas posições em que havia indiferenças na função da alínea anterior coloque "1"s. Desenhe o logigrama desta nova função partindo do logigrama anterior, sem o alterar, excepto por acrescentar outros elementos.

5.21 a) Utilizando mapas de Karnaugh, minimize a função

$$f = \prod M(1,2,4,9,10,12,15,17,24,30,31)\,,$$

com indiferenças nas posições 3, 5, 8, 14, 18, 20, 22 e 25.

b) Minimizando esta função em produto de somas e em soma de produtos obteria duas expressões equivalentes? Porquê?

5.22 Minimize a função

$$f = \sum m(0,1,3,5,7-10,15,18,19,22,25,27,29,31)\,,$$

sabendo que existem indiferenças nas posições 2, 11, 12, 20 e 30. A função deve ser minimizada por forma a ser implementada facilmente com NANDs.

5.23 Dada a seguinte função:

$$f = \sum m(0,5,6,8,12,14,22,29)\,,$$

com indiferenças nas posições 1, 4, 10, 21, 16, 19, 20, 23, 25 e 26,

a) minimize-a e represente-a sob a forma de uma soma de produtos;

b) identifique um implicante primo essencial e um não essencial.

(∗) 5.24 Uma função de 4 variáveis é dada na forma

$$y = (m_1 + m_3 + m_5 + m_9 + m_{10} + m_{11} + m_{12} + m_{14})\,(M_8 \cdot M_{10})\,.$$

O factor $(M_8 \cdot M_{10})$ é necessário para a definição da função, ou não fornece qualquer indicação que não esteja já contida no primeiro factor do produto lógico? Responda referindo-se separadamente aos dois termos máximos que constituem o segundo factor.

5.25 a) Minimize a seguinte função

$$f = \sum m(2, 7, 9, 11, 12, 14, 15, 17, 23) + \sum m_d(0, 4, 6, 8, 10, 13, 20, 22, 28)\,.$$

b) Identifique todos os implicantes primos essenciais da função.

5.26 Dada a seguinte função,

$$f = \prod M(1, 3, 5, 8, 10, 12 - 14, 21, 23, 24, 26, 31) \cdot \cdot \prod M_d(0, 4, 7, 15, 17, 18, 27, 28)\,,$$

a) minimize-a de modo a facilitar uma futura implementação com NANDs;

b) identifique na expressão mínima anterior os implicantes primos essenciais.

5.27 Minimizar a seguinte função:

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

5.28 a) Minimizar a seguinte função usando o método de Karnaugh:

$$f = \prod M(1, 7, 9, 12, 14, 18, 19, 21, 22, 25, 28, 30, 31)\,,$$

com indiferenças nas posições 4, 15, 16, 17 e 20.

b) A partir da função f, sem a alterar e usando o mínimo de lógica possível, construir uma função g com uma tabela de verdade lógica semelhante mas que gera saída 1 nas posições 1, 4 e 21.

c) Indique, na função f, um implicante (ou implicado) primo que não seja essencial, e outro que o seja.

5.29 a) Minimize, pelo método de Karnaugh, a função

$$f(A, B, C, D, E) = \prod M(2, 4, 7, 9, 10, 12, 18, 24, 30, 31)\,,$$

com indiferenças nas posições 11, 15, 26, 28 e 29 (admita que A é a variável com maior peso), de forma a ser facilmente implementada com NANDs de qualquer número de entradas, Não desenhe o logigrama, mas determine a expressão da função em NANDs.

b) Considere a função obtida na alínea anterior. Qual o valor que ela assumirá para a quantidade booleana geral de entrada $(A, B, C, D, E) = = (1, 1, 0, 1, 0)$? Porquê?

5.30 Dada a seguinte função, e sabendo que na sua implementação é indiferente o valor que ela toma para as combinações de entrada referentes aos mintermos 3, 9, 10, 16, 20 e 30,

$$f = \prod M(0, 1, 3, 4, 7, 11, 12, 16, 17, 19, 26, 29, 30, 31)\,,$$

a) minimize-a de modo a realizá-la apenas com NORs;

b) se, por acaso, ocorrerem as combinações de entrada correspondentes às quantidades booleanas gerais com afixos 9, 16 e 30, quais são os valores que a implementação obtida na alínea anterior apresenta?

5.31 a) Usando o método de Karnaugh, minimize a função

$$f = B\,\overline{A} + B\,A\,\overline{D} + C\,\overline{D} + A\,B\,C\,D + \overline{C}\,B\,D\,.$$

b) Indique um implicante primo essencial e um não essencial.

Parte II

CIRCUITOS COMBINATÓRIOS

Capítulo 6

Elementos Tecnológicos

6.1 Portas Lógicas

O interesse do que se tem vindo a analisar reside no facto de as funções lógicas poderem ser usadas para representar as acções e os processamentos pretendidos para um determinado equipamento, e poderem ser implementadas fisicamente por circuitos electrónicos.

Considere-se o seguinte problema: pretende-se desenhar um alarme de um carro que assinale, tocando um besouro, que o carro circula com uma porta aberta ou que está estacionado com as luzes acesas.

Se o carro tiver 4 portas, podemos representar cada porta por uma variável, $P1$ a $P4$. Admitamos que cada uma das variáveis estará a 1 quando a porta correspondente está aberta. A função P significará então, quando a 1, que existe pelo menos uma porta aberta. Então teremos $P = P1 + P2 + P3 + P4$.

Do mesmo modo, L será uma variável que está a 1 quando as luzes estão acesas e C uma variável que está a 1 quando a chave está ligada.

É facil de ver que o alarme A será obtido pela expressão

$$A = C\,P + \overline{C}\,L\,,$$

representável pelo logigrama da Figura 6.1

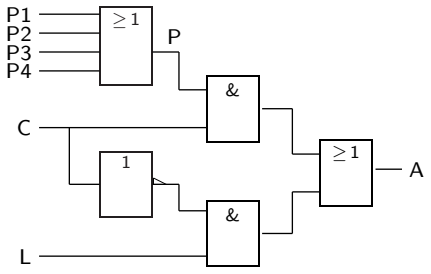


Figura 6.1: Logigrama do alarme $A = C\,P + \overline{C}\,L$

A implementação fisica do logigrama é feita por um circuito eléctrico em que os diversos operadores lógicos são implementados por portas lógicas ("gates") realizadas em *circuitos integrados*. Um exemplo de circuito integrado é o caso do 74LS08, que implementa os ANDs (ver a folha de especificação do integrado).

6.2 Sinais Binários

Sinal binário ideal

A Figura 6.2(a) ilustra a variação com o tempo de um **sinal binário ideal** numa ligação entre a saída de uma porta lógica e a entrada ou entradas de outras portas.

Como se pode constatar, a representação do sinal ideal apresenta descontinuidades nos instantes de tempo designados por t_1, t_2, e t_3, possuindo um dos dois valores binários, 0 ou 1, nos intervalos entre esses instantes.

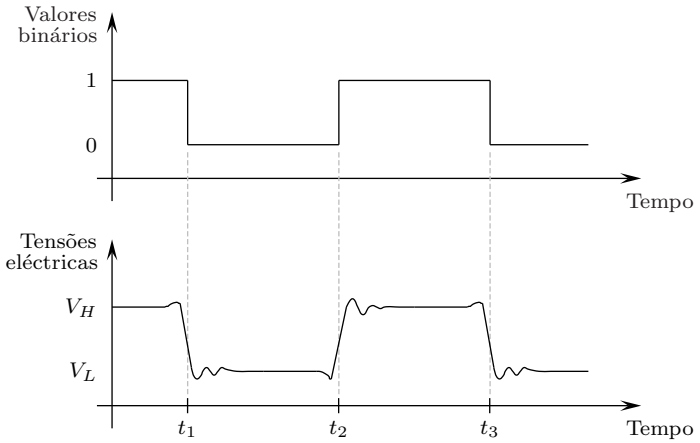


Figura 6.2: Sinal binário: (a) ideal; (b) real

Sinal binário real

Sinal real (analógico)

Num sistema digital prático, contudo, não existem sinais ideais. Os **sinais binários reais** são, nesses sistemas, representados por valores de grandezas eléctricas — geralmente tensões ou correntes — pelo que a representação da informação deverá neles ser feita de forma contínua, por intermédio de **sinais analógicos**.

Assim sendo, a forma ideal da Figura 6.2(a) deve ser entendida apenas como uma representação aproximada dos sinais que podemos encontrar nas linhas (ou condutores) que ligam as entradas e as saídas das portas lógicas.

Flancos

A Figura 6.2(b) ilustra um sinal binário real, tendo como ordenadas tensões eléctricas. Como podemos constatar, as descontinuidades da Figura 6.2(a) dão agora lugar a transições ou **flancos** entre as tensões V_H e V_L, flancos esses que levam tempo a estabelecer-se.

Flanco descendente

Quando um sinal muda de uma tensão alta, V_H, para uma tensão baixa, V_L, dizemos que ocorreu um **flanco descendente**.

Tempo de decrescimento t_f ou "fall time"

Como esta transição não é instantânea, um flanco descendente possui um certo **tempo de decrescimento** para efectuar a mudança. Este tempo designa-se habitualmente por $\mathbf{t_f}$ ou **"fall time"**.

Flanco ascendente

Tempo de crescimento t_r ou "rise time"

Pelo contrário, uma transição de V_L para V_H dá origem a um **flanco ascendente**, com um certo **tempo de crescimento**, designado por $\mathbf{t_r}$ ou **"rise time"**, para efectuar a mudança entre os dois valores de tensão.

Por outro lado, como resultado de imperfeições existentes nos componentes electrónicos das portas lógicas, sinais que se deviam manter constantes entre transições, como na Figura 6.2(a), podem flutuar ligeiramente em torno dos valores V_H e V_L, como mostra a Figura 6.2(b).

Nível alto, HIGH ou H

Nível baixo, LOW ou L

Isso significa que os valores binários 1 e 0 devem corresponder, na realidade, a intervalos de valores ou a faixas de tensões eléctricas, designados habitualmente por **níveis (de tensão) H ou L**, respectivamente um **nível alto** (**HIGH**) ou um **nível baixo** (**LOW**).

A associação entre os intervalos de tensão H e L e os valores lógicos 0 e 1 é matéria que deixaremos para o Capítulo 7. Basta dizer, por agora, que muitas vezes se faz a associação de H a 1 e de L a 0, mas que nada impede a associação contrária.

Enquanto um sinal tiver um valor de tensão eléctrica contido no interior de uma das faixas H ou L, o seu valor binário (1 ou 0) é bem definido. Quando o valor da tensão eléctrica sair fora dessas faixas, o valor binário do sinal fica indefinido (Figura 6.3).

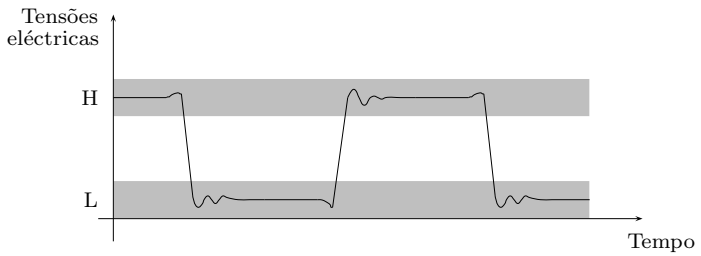


Figura 6.3: Sinal binário da Figura 6.2 indicando as faixas de tensão H e L

Uma forma de representação idealizada de um sinal binário será, então, a da Figura 6.4(a), onde se acentuam os flancos entre os níveis H e L, com os seus tempos de crescimento e de decrescimento, mas em que se simplifica a forma do sinal, por forma a aproximar-se da representação ideal da Figura 6.2(a).

Nos casos em que não é necessário acentuar os tempos de crescimento e de decrescimento dos flancos, usa-se a representação ideal da Figura 6.4(b).

6.3 Tipos de Circuitos Integrados Digitais

A tecnologia dos circuitos integrados é responsável pelo tremendo desenvolvimento dos sistemas digitais, uma vez que possibilitou a construção de circuitos electrónicos digitais muito complexos a custos extremamente reduzidos.

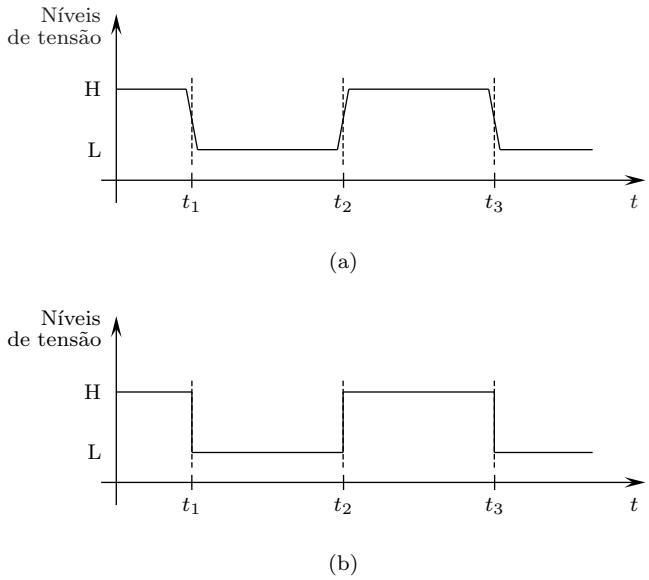


Figura 6.4: (a) Forma de representação habitual dos sinais binários, com ênfase nos flancos de mudança dos sinais e dos seus tempos de crescimento e decrescimento; e (b) representação simplificada, ideal, onde não aparecem os tempos de crescimento e de decrescimento

Circuito integrado (CI)

Um **circuito integrado** (CI) é formado por um pequeno cristal de silício (em geral, mas há outras tecnologias) onde se difundiram impurezas para formar transistores, díodos, resistências e outros elementos de circuitos, interligados entre si para formarem circuitos de maior ou menor complexidade.

Circuito integrado digital

Um **circuito integrado digital** materializa circuitos digitais. Os circuitos integrados digitais são fáceis (relativamente) de conceber e produzir, assumindo custos muito baixos.

Cada CI digital pode ter entre algumas de portas lógicas e alguns milhões delas. Por exemplo, o circuito integrado usado habitualmente no nosso laboratório com circuitos AND tem 4 portas AND independentes, que podem ser usadas separadamente. Um circuito como um contador tem algumas dezenas de portas que estão previamente interligadas para formar o contador. Já um microprocessador avançado, como o Pentium ou o PowerPC, tem vários milhões de portas ou circuitos equivalentes.

Existem várias tecnologias para fabricar circuitos integrados digitais.

Tecnologia TTL
Transistores bipolares

1. **Tecnologia TTL**: uma tecnologia que foi muito usada e que estabeleceu critérios e normas. Utiliza **transistores bipolares** a funcionar ao corte ou à saturação. Tem um comportamento médio no que diz respeito ao tempo de propagação por porta (este tempo mede, de forma simplificada, a *velocidade* da porta), ao consumo em energia eléctrica e à potência dissipada sob a forma de calor. Os circuitos com esta tecnologia formam a **família TTL**, que se subdivide em várias **sub-familias** com características diferenciadas, mas todas compatíveis entre si. Dado ter-se tornado muito popular,

Família TTL
Subfamílias lógicas

outras tecnologias têm circuitos com entradas e saídas compatíveis com os circuitos TTL. É uma tecnologia em perda a favor da tecnologia CMOS.

2. **Tecnologia CMOS**: a outra tecnologia de grande divulgação. Utiliza **transistores MOS** com canais de 2 tipos. É relativamente mais lenta que a TTL, mas consome menos. Recentemente, os ganhos de velocidade da tecnologia CMOS, combinados com a grande capacidade de integração e os baixos consumos energéticos e potência dissipada, colocaram-na como a tecnologia mais importante no projecto e fabrico de CIs. Os circuitos com esta tecnologia formam a **família CMOS**, também dividida em diversas sub-familias. É a tecnologia em mais rápido crescimento.

Tecnologia CMOS

Transistores MOS

Família CMOS

3. **Tecnologia ECL**: utiliza transistores bipolares na zona activa. É a tecnologia mais rápida comercialmente disponivel. Tem um consumo muito elevado. É de relativamente difícil utilização. Está reservada a nichos de aplicação muito reduzidos.

Tecnologia ECL

4. Outras tecnologias: **nMOS** e **pMOS**, **GaAs**, **IIL**, etc.

Tecnologias nMOS, pMOS, GaAs, IIL

6.4 A Família TTL

Usaremos no laboratório circuitos integrados da família TTL por ser electricamente mais robusta criando, por isso menos problemas a alunos com pouca experiência.

Os circuitos desta família caracterizam-se por um conjunto de parâmetros, de que se destacam:

— as **tensões de alimentação**: $+5V \pm 5\%$ para as séries normais (comerciais), e $+5V \pm 10\%$ para as séries militares; a tensão de $+5V$ é habitualmente designada por V_{cc}, e os circuitos TTL são alimentados entre a tensão V_{cc} e a tensão da massa (GND = 0V);

Tensões de alimentação (TTL)

— as **temperaturas de funcionamento**: $[0, 70]$ ^{o}C para as séries comerciais, enquanto que as séries militares usam circuitos mais robustos, que suportam temperaturas no intervalo $[-55, 125]$ ^{o}C;

Temperaturas de funcionamento (TTL)

— as **sub-familias**: N (Normal), L ("Low Power"), H ("High Speed"), LS ("Low Power Schottky", a mais usada actualmente), S ("Schottky"), ALS ("Advanced Low Power Schottky"), AS ("Advanced Schottky"), F ("Fast"), etc.;

Sub-famílias TTL

— os **tempos de atraso** (ou **tempos de propagação**) e os **consumos de corrente** típicos: por exemplo, para uma porta 74x00 (em que "x" designa uma das subfamílias TTL), os valores são os indicados na Tabela 6.1.

Tempo de propagação (atraso)

Consumo de corrente (TTL)

— em cada sub-família, a série 54xnn é a série militar e a 74xnn é a série comercial.

Tabela 6.1: Tempos de propagação e consumos de corrente típicos para o mesmo circuito integrado em três sub-famílias TTL diferentes

Sub-família TTL	Tempo de atraso típico (ns)	Corrente de alimentação típica (mA)
N (7400)	7 a 11	4 a 12
LS (74LS00)	9 a 10	0, 8 a 2, 4
S (74S00)	3 a 5	10 a 20

6.4.1 Níveis eléctricos dos circuitos TTL

Como vimos anteriormente, numa ligação entre duas portas arbitrárias, em tecnologia TTL ou noutra tecnologia qualquer, os fabricantes de circuitos integrados apenas garantem certos intervalos (gamas) de tensões eléctricas nas suas saídas e entradas, que designam por níveis H e L.

Deste modo, se a saída de uma porta se encontrar, em cada instante, num desses intervalos, pode-se considerar que a porta impôs na saída um determinado nível de tensão, que é aceite pelas entradas das portas às quais a saída se liga.

No caso da família TTL os fabricantes garantem que, aparte os momentos em que se verifica uma transição à saída de uma porta em resultado de uma mudança nas entradas, a saída da porta estará no nível H (entre +2,4V e +5V), ou no nível L (entre 0V e +0,4V), como mostra a Figura 6.5, se as entradas estiverem também num desses níveis.

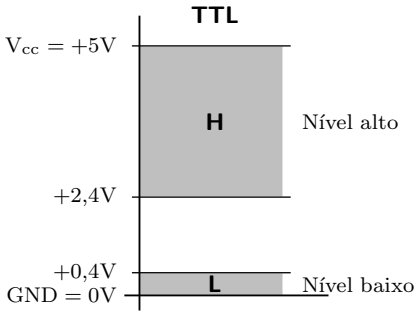


Figura 6.5: Níveis H e L na tecnologia TTL

Repare-se, contudo, que existe aquilo a que habitualmente chamamos 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, ruído provocado pelo arranque e paragem de motores eléctricos, emissão electromagnética de outras ligações do mesmo circuito, etc.).

Assim, é possível que à saída da porta tenhamos um valor válido (por exemplo +0,3V) e à entrada da porta que lhe está ligada se observe um valor inválido (por exemplo, +0,5V).

Para ter em conta esta possibilidade, as entradas das portas têm uma especificação um pouco mais ampla, sendo os valores entre +2V e +5V interpreta-

dos como nível H, e os valores entre 0V e +0,8V interpretados como nível L (Figura 6.6).

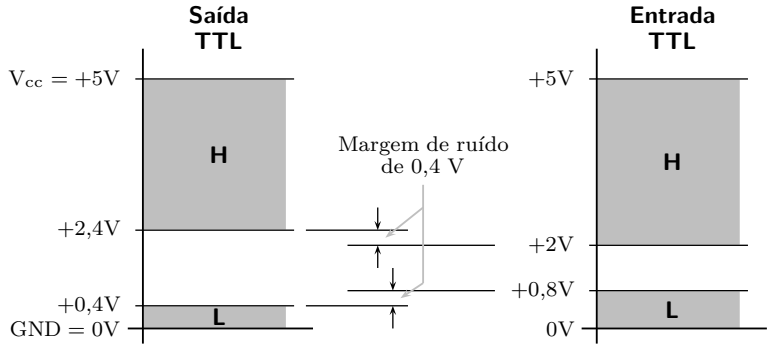


Figura 6.6: A margem de ruído em TTL é de 0,4V

O intervalo de 0,4V entre os valores extremos à saida e à entrada designa-se, naturalmente, por **margem de ruido**.

Margem de ruído

6.4.2 Saídas totem-pole e tri-state

Não é possível ligar as saídas das portas TTL habituais, sob pena de as podermos destruir por excessivo consumo de corrente e consequente dissipação de potência (Figura 6.7).

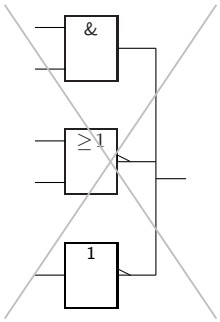


Figura 6.7: A ligação das saídas das portas TTL totem-pole pode conduzir às suas destruições devido ao excessivo consumo de corrente eléctrica

Tal deve-se ao modo como são construídos os andares de saída dessas portas, designados por **totem-pole**.

Saídas totem-pole

Há, contudo, portas TTL especiais, designadas por **portas tri-state**, em que os andares de saída — que são controlados por uma entrada suplementar designada por **entrada de Enable** (EN) — podem vir interligados.

O desenho dos circuitos TTL e, em particular, dos seus andares de saída sai fora do âmbito desta cadeira.

Saídas e portas tri-state

Os circuitos que usam saídas tri-state apresentam uma simbologia IEC específica, como mostra a Figura 6.8 para o caso de uma porta NAND com 3 entradas.

Como se pode ver, a porta NAND tem as 3 entradas habituais, A, B e C, e a entrada suplementar de Enable, EN, que controla a função tri-state da saída

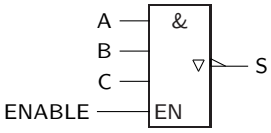
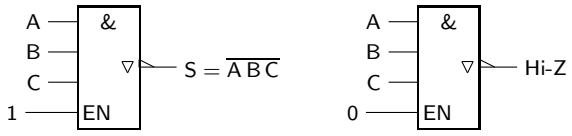


Figura 6.8: Símbolo IEC de uma porta NAND com saída tri-state

Qualificador de saída ▽

S. Essa função vem representada no símbolo por um **qualificador de saída** com a forma de um triângulo invertido, ▽ .

Quando à entrada de Enable aplicamos um determinado valor lógico, digamos 1 [Figura 6.9(a)], a porta reage como um vulgar NAND, gerando os valores lógicos na saída compatíveis com a expressão habitual $S = \overline{A\,B\,C}$. O seu andar de saída comporta-se, nestas circunstâncias, como o de um vulgar NAND com saída totem-pole.



(a) (b)

Figura 6.9: (a) A porta NAND com saída tri-state tem a entrada de Enable, EN, com o nível de tensão que lhe permite funcionar normalmente, como se de um totem-pole se tratasse; (b) quando a entrada EN tem o outro nível de tensão, a saída fica em alta impedância (Hi-Z)

A designação "tri-state" provém do facto de podermos encontrar na saída os habituais valores 0 e 1 (ou, se quisermos, os níveis L e H) de um totem-pole, e ainda a condição de alta impedância, Hi-Z.

Saída em alta impedância (Hi-Z)

Quando a EN aplicamos um 0 [Figura 6.9(b)], a saída da porta fica em **alta impedância** (o que se simboliza por Hi-Z), e *tudo se passa como se a saída estivesse desligada (ou seja, não conseguimos identificar "1"s e "0"s na saída).*

É fácil de entender que podemos ligar as saídas de duas ou mais portas tri-state *desde que controlemos devidamente as portas por forma a que uma e apenas uma das portas funcione, de cada vez, como um totem-pole, e as outras tenham as suas saídas em alta impedância.*

É o que se faz, por exemplo, no logigrama da Figura 6.10.

Se $EN1$ estiver a 1 e $EN2$ e $EN3$ estiverem a 0, por exemplo, as duas portas inferiores têm as suas saídas em alta impedância (estão "desligadas") e o que aparece na saída comum é o que se encontra à saída do AND superior, ou seja, $S = A\,B$.

Se, noutra altura diferente, se fizer $EN2 = 1$ e $EN1 = EN3 = 0$, agora é o AND e o NOT que têm a saída em alta impedância, e o que surge na saída comum é o que está à saída do NOR, isto é, $S = \overline{C + D}$.

E se numa terceira altura, diferente das duas enteriores, se tiver $EN1 = EN2 = = 0$ e $EN3 = 1$, agora é o NOT que debita $\overline{E}$ para a saída S, porque o AND e o NOR têm as suas saídas em alta impedância.

Em resumo, às entradas de Enable apenas deveremos aplicar as quantidades booleanas gerais $(EN1, EN2, EN3) = (1, 0, 0)$, ou $(0, 1, 0)$, ou ainda $(0, 0, 1)$, e não outras. Em particular *não faz sentido* aplicar $(EN1, EN2, EN3) = (0, 0, 0)$

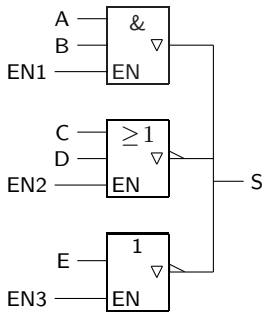


Figura 6.10: As saídas tri-state podem vir interligadas desde que se garanta, por intermédio dos níveis de tensão aplicados às diversas entradas de Enable, que uma e apenas uma saída de cada vez não está em alta impedância

às entradas, porque então as três portas estariam em alta impedância *e nenhuma função era gerada na saída nessas circunstâncias*.

Esta operação de selecção de uma das saídas das portas de cada vez designa-se por **multiplexagem temporal**.

Multiplexagem temporal

6.5 A Familia CMOS

Os circuitos desta família caracterizam-se por um conjunto de parâmetros, de que se destacam:

Sub-famílias CMOS

— as **sub-familias**: 4000, 74HC e 74HCT. A sub-família 4000 é a primeira família CMOS, que tem vindo a ser progressivamente abandonada e substituída pelas outras. Os circuitos da sub-família 74HC têm as referências e os pinos compatíveis com os dos circuitos equivalentes (com a mesma designação) da familia TTL. Mas não são electricamente compatíveis. Os circuitos da sub-família 74HCT são uma variante de circuitos HC com níveis de tensão totalmente compativeis com os da família TTL.

Tensões de alimentação (em CMOS)

— as **tensões de alimentação**: +5V ± 5% para os circuitos 74HC e 74HCT, e +3 a +18V para os circuitos 4000; estas tensões são habitualmente designadas por V_{dd}, e os circuitos CMOS são alimentados entre a tensão V_{dd} e a tensão da massa (GND = 0V).

6.5.1 Níveis eléctricos dos circuitos CMOS

Os niveis de tensão nos circuitos da familia CMOS são diferentes dos da familia TTL (Figura 6.11).

No caso da série 4000 os niveis são variáveis com a tensão de alimentação, que pode ir de +3V a +18V. Aplicam-se as mesmas convenções que nos circuitos TTL. Os níveis ilustrados são os das entradas das portas.

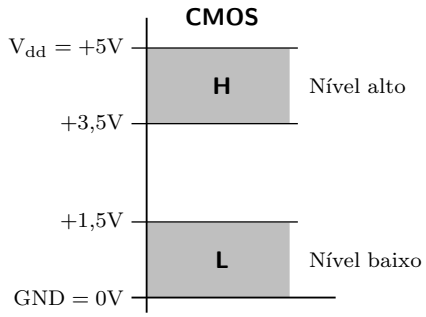


Figura 6.11: Níveis H e L na tecnologia CMOS

6.6 Encapsulamento dos Integrados

Os circuitos integrados são pequenos cristais de silício que, para poderem ser facilmente utilizados, estão inseridos em envólucros que facilitam o seu manuseio e interligação.

"Dual in-line package"

Pinos

Os integrados que vamos usar estão encapsulados num tipo particular de envólucro designado por DIP ("Dual in-line package") por terem dois conjuntos de **pinos** (terminais), organizados em duas filas em lados opostos do encapsulamento.

Existem integrados deste tipo com um número de pinos entre 8 e 64. Na Figura 6.12 ilustra-se um integrado com 14 pinos.

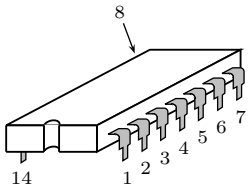


Figura 6.12: Exemplo de circuito integrado que utiliza um envólucro com 14 pinos

Repare-se na forma como os pinos são numerados. O pino 1 e o último estão próximos de uma pequena reentrância no encapsulamento, que pode tomar a forma ilustrada ou outras.

6.7 "Fan-out"

"Fan-out"

O "fan-out" de uma saida é o número de entradas normais a que essa saida pode estar ligada. Esse valor anda na ordem de 10 a poucas dezenas.

6.8 Dissipação de Potência

Os circuitos integrados são alimentados electricamente e, no seu funcionamento, dissipam a energia eléctrica sob a forma de calor. O problema do aquecimento e das técnicas de arrefecimento dos integrados são, com a elevada integração, cada vez mais difíceis de tratar.

No caso dos circuitos CMOS, a energia é fundamentalmente gasta nas transições de nível, sendo desprezável o valor gasto em repouso, quando não há transições. É por isso que os processadores dissipam mais se a frequência de funcionamento é maior.

Já no caso dos circuitos TTL, a energia é gasta, quer nas transições entre níveis, quer quando os circuitos se mantêm num nível constante, H ou L.

6.9 Tempos de Propagação das Portas

Como vimos na Secção 6.2, as portas lógicas apresentam flancos nas transições entre níveis, aos quais estão associados tempos de crescimento e de decrescimento dos sinais binários.

Por outro lado, e porque os circuitos integrados são dispositivos reais e físicos, as suas saídas também não reagem instantaneamente às mudanças que ocorrem nas suas entradas. Isso quer dizer que, se observarmos os seu comportamentos ao longo do tempo, notamos **tempos de propagação** (ou **tempos de atraso**) na resposta dos integrados.

Tempos de propagação (atraso)

Vejamos o exemplo de uma porta AND na Figura 6.13, onde se salientam os tempos de propagação da porta, $\mathbf{t_{pLH}}$ na transição do nível L para o nível H na saída, e $\mathbf{t_{pHL}}$ na transição do nível H para o nível L, sempre na saída.

t_{pLH} *e* t_{pHL}

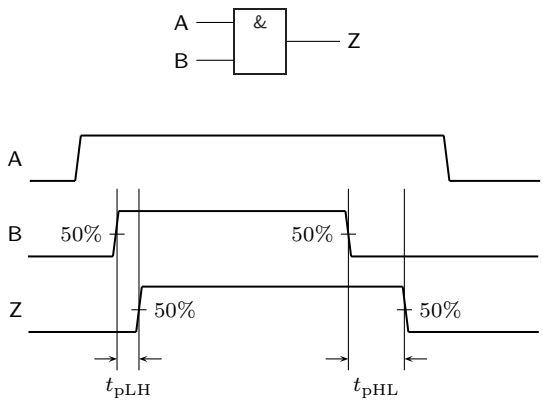


Figura 6.13: Uma porta AND e um diagrama temporal onde se representa a variação na saída Z que resulta de determinadas variações nas entradas A e B, impostas arbitrariamente

Deve notar-se que o tempo de propagação t_{pLH} pode ser diferente do tempo t_{pHL}.

t_{pd} (tempo de propagação de uma porta)

Por vezes designam-se os dois tempos colectivamente por **tempo de propagação da porta**, com a sigla $\mathbf{t_{pd}}$ (**"propagation delay time"**).

Diagrama temporal

Os diagramas do género do que se representa na Figura 6.13 designam-se por **diagramas temporais**.

6.10 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 2.1, 2.8 e 2.9.

Capítulo 7

Lógica de Polaridade

7.1 Lógicas de Polaridade, Positiva e Negativa

7.1.1 Símbolos dos circuitos digitais

Os símbolos que utilizámos para as portas lógicas estão definidos internacionalmente em **normas** emanadas da **IEC** ("International Electrotechnical Commission´´ ou Comissão Electrotécnica Internacional).

Normas IEC

Trata-se de símbolos aplicáveis não só às portas lógicas, como a outros circuitos mais complexos, geralmente sob a forma de circuitos integrados, que estudaremos nos capítulos que se seguem à medida que forem sendo necessários (multiplexers, descodificadores, contadores, registos, memórias, etc.).

Esta uniformização possui algumas vantagens óbvias:

— os símbolos são de utilização universal em **logigramas** e **esquemas eléctricos**; ou seja, desde que seja respeitada a sua representação simbólica nos circuitos lógicos em que intervêm (representação essa que está contida na **norma IEC 60617-12**), qualquer pessoa é capaz de entender um circuito desenhado por outrem;

Logigramas e esquemas eléctricos

Norma IEC 60617-12

— como a simbologia é rigorosa, não há margem de manobra para o desenho de símbolos mais ou menos fantasiosos (símbolos fantasiosos significa símbolos arbitrários, abertos a interpretações incorrectas); e

— os símbolos dispensam a descrição dos circuitos em língua natural (português ou outra); com efeito, tal descrição é não só redundante como geralmente pouco correcta e incompleta, pelo que só temos a ganhar com a sua dispensa.

Não devemos, contudo, imaginar que esta simbologia se deve ou pode aplicar a *qualquer* circuito digital. Com efeito, circuitos muito complexos — por exemplo, certas memórias de grande dimensão, os circuitos lógicos programáveis ou os microprocessadores e respectivos circuitos auxiliares, cujas funcionalidades não são facilmente descritíveis pela norma IEC — possuem símbolos muito complexos, e as vantagens que se acabaram de apontar podem, então, perder-se. Nesses

casos sugere-se a utilização de uma simbologia simplificada, não normalizada, acompanhada de uma descrição da correspondente funcionalidade.

Norma IEC 61082

Para além da norma 60617-12, existem outras normas IEC que se preocupam com o desenho dos logigramas e dos esquemas eléctricos. Neste contexto apenas nos interessa a **norma IEC 61082**, que estabelece, entre outros conceitos importantes, que o fluxo dos sinais eléctricos decorre da esquerda para a direita ou de cima para baixo nos desenhos dos circuitos, e que qualquer alteração a estas orientações deve ser explicitamente referenciada com uma seta.

Qualificador geral
Qualificador geral &
Qualificador geral ≥ 1
Qualificador geral 1
Qualificador geral $= 1$

No que diz respeito às portas lógicas, os símbolos IEC possuem contornos rectangulares colocados verticalmente, com um **qualificador geral** no topo que identifica o tipo de porta: o **qualificador geral &** para uma porta AND, o **qualificador geral $\geq$ 1** para uma porta OR, o **qualificador geral 1** para uma porta NOT, ou o **qualificador geral =1** para uma porta XOR.

De propósito não foram referidas as portas NAND e NOR. Isso deve-se a que, por razões que serão apresentadas mais tarde, estas portas passam a ser substituídas, respectivamente, por portas AND e OR.

Importa salientar aqui um facto importante: *qualquer porta lógica possui dois símbolos IEC em alternativa, que resultam um do outro pela aplicação de uma das leis de De Morgan.*

Por exemplo, os símbolos que se seguem são equivalentes,

como resulta de

$$\overline{A+B} = \overline{A}\,\overline{B}\,,$$

pela aplicação de uma das leis de de Morgan.

Da mesma forma, os símbolos

são equivalentes, como resulta de

$$\overline{\overline{A}+B} = A\,\overline{B}\,.$$

Qualificadores de entrada e de saída ◺

Indicadores de polaridade

Mais à frente iremos caracterizar estes e outros símbolos, que passarão a ter designações apropriadas consoante existem ou não os **qualificadores de entrada ou de saída** ◺ , *com um significado que não é o de negação lógica*, como podem sugerir as equivalências anteriores (estes qualificadores também se designam por **indicadores de polaridade**).

7.1.2 Razão da lógica de polaridade

Como vimos nas Subsecções 6.4.1 e 6.5.1, os níveis de tensão H e L são, na realidade, gamas de valores de tensão eléctrica que dependem da tecnologia utilizada.

Em qualquer dos casos que foram apresentados (tecnologias TTL e CMOS), contudo, esses níveis são *positivos* porque os circuitos são alimentados entre uma tensão $V_{cc} = +5V$ ou $V_{dd} = +5V$ (eventualmente, outra tensão positiva) e a tensão da massa (GND = 0 V).

Porém, há tecnologias em que os circuitos são alimentados com tensões *negativas*. É o caso, por exemplo, da tecnologia ECL, em que os circuitos integrados são habitualmente alimentados entre a tensão da massa (GND = 0 V) e uma tensão negativa, ($V_{ee} = -5,2V$), como ilustra a Figura 7.1.

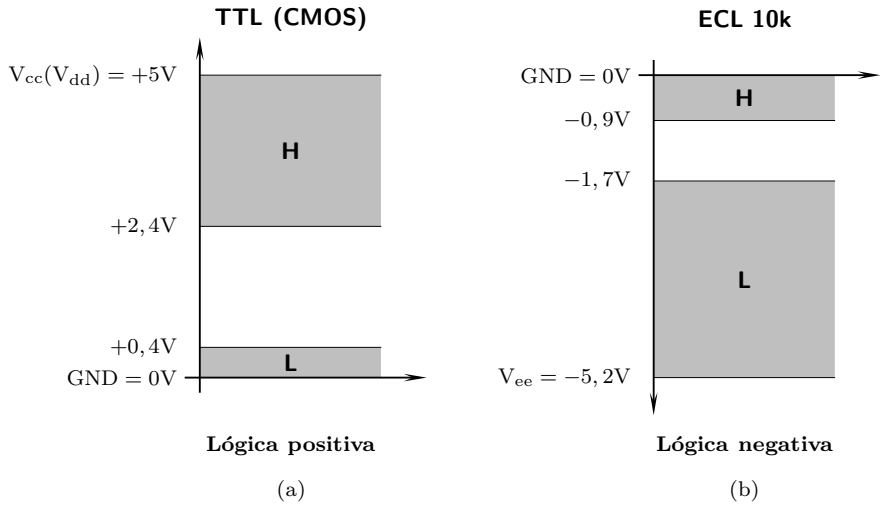


Figura 7.1: (a) A utilização de uma tecnologia TTL (ou CMOS) favorece a escolha de uma lógica positiva, enquanto que (b) a utilização de uma tecnologia ECL favorece a utilização de uma lógica negativa

Ora o problema que se levanta é *como associar os valores lógicos 0 e 1 da álgebra de Boole que temos utilizado até aqui, com os níveis de tensão nos circuitos utilizados nas diversas tecnologias físicas*, de uma forma que seja independente da tecnologia ou de preferências pessoais.

Uma associação possível — ilustrada na Figura 7.1(a) — que "casa" bem com as tecnologias que usam tensões positivas, como é o caso da TTL e da CMOS, faz corresponder o 0 lógico ao nível L e o 1 lógico ao nível H. É a **lógica positiva**, que temos vindo a utilizar até aqui.

Lógica positiva

Mas existe outra associação possível, a da **lógica negativa** da Figura 7.1(b), que faz a ligação oposta: faz-se corresponder o 0 lógico ao nível H e o 1 lógico ao nível L. Esta associação faz sentido se utilizarmos uma tecnologia com os circuitos a serem alimentados por tensões negativas (como a ECL), porque o 0 lógico corresponde à gama de tensões próximas da da massa (0V), isto é, ao nível alto (H), e o 1 lógico fica reservado para a outra gama de tensões, o nível baixo (L).

Lógica negativa

Naturalmente, quando se utiliza a álgebra de Boole estas questões não fazem sentido, apenas se tornando importantes quando queremos *implementar um determinado circuito*. Como daqui para a frente vamos estar interessados nas implementações, teremos de decidir por uma ou por outra ou, melhor ainda, achar uma solução alternativa.

Lógica de polaridade

É o que faremos com a **lógica de polaridade**, *em que prescindimos das associações entre "0"s, "1"s, "L"s e "H"s*. E de caminho *vamos deixar cair completamente os "0"s e os "1"s* (ou seja, apenas lidaremos com os "H"s e os "L"s), *reservando os primeiros apenas para o tratamento algébrico das funções em tabelas de verdade, quadros de Karnaugh, expressões lógicas, etc.*

7.1.3 Tabelas de verdade físicas e lógicas

Para podermos utilizar a lógica de polaridade tal como a enunciámos anteriormente, há que proceder a algumas mudanças que possivelmente já estão enraizadas no nosso pensamento. Por exemplo, consideremos a tabela de verdade de uma certa função booleana simples. No contexto da álgebra de Boole não tivemos nenhuma dificuldade em estabelecer essa tabela com os seus "0"s e "1"s.

Mas se vamos prescindir dos "0"s e "1"s na implementação dos circuitos digitais e na utilização da lógica de polaridade, temos que poder desenhar um tipo diferente de tabela de verdade, apenas com "L"s e "H"s. Para distinguir os dois tipos de tabela designamos a primeira por **tabela de verdade lógica** — ou, mais simplesmente, e na linha do que fizemos nos capítulos anteriores, por **tabela de verdade** — e a segunda por **tabela de verdade física**.

Tabela de verdade (lógica)

Tabela de verdade física

Naturalmente, temos de perceber como passar das já nossas conhecidas tabelas de verdade lógicas para as correspondentes tabelas de verdade físicas, e vice-versa, para todo e qualquer circuito digital.

Para o fazer vamos recorrer a um exemplo. Suponhamos, então, que tinhamos que resolver o seguinte problema prático: temos à nossa disposição um determinado circuito, que desconhecemos (por exemplo, o circuito está contido num circuito integrado cuja denominação está apagada), mas sabemos onde estão as suas entradas e saídas. Com estes dados podemos injectar nas entradas níveis H ou L, e verificar os níveis gerados pelo circuito nas suas saídas. Desta forma podemos estabelecer a tabela de verdade física do circuito. A questão é saber de que circuito se trata.

Suponhamos, então, que com esta experiência obtinhamos a tabela de verdade física da Tabela 7.1 para um circuito com duas entradas, A e B, e uma saída, S.

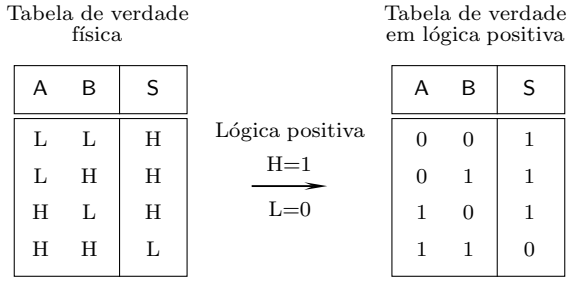
Se conseguirmos perceber o que este circuito representa em lógica positiva (e já agora, porque não em lógica negativa?), podemos deduzir a expressão lógica da função booleana simples que ele implementa.

Ora a conversão da tabela de verdade física para lógica positiva é simples: basta substituir H por 1 e L por 0, como vimos acima pela definição desta lógica. Obtemos, então, a Figura 7.2.

Facilmente percebemos que o circuito em questão é uma porta NAND com 2 entradas.

Tabela 7.1: Tabela de verdade física de um circuito desconhecido, cuja função pretendemos determinar

A	B	S
L	L	H
L	H	H
H	L	H
H	H	L

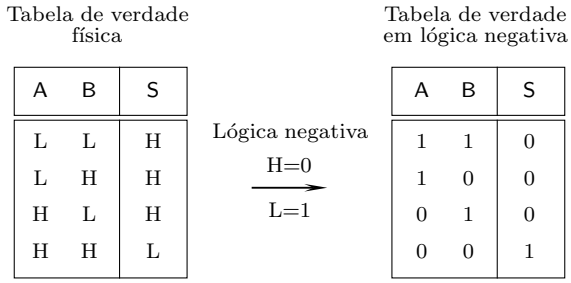


A	B	S
L	L	H
L	H	H
H	L	H
H	H	L

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

Figura 7.2: A conversão da tabela de verdade física da esquerda para a lógica positiva dá a tabela de verdade em lógica positiva de um NAND com 2 entradas

E se fizessemos o mesmo exercício para tentar perceber o significado do circuito desconhecido em lógica negativa, obtinhamos a Figura 7.3 e, da tabela de verdade lógica, percebíamos que estávamos em presença de um porta NOR com 2 entradas.



A	B	S
L	L	H
L	H	H
H	L	H
H	H	L

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

Figura 7.3: A conversão da tabela de verdade física da esquerda para a lógica negativa dá a tabela de verdade em lógica negativa de um NOR com 2 entradas

Mas afinal trata-se de um NAND ou de um NOR? Naturalmente, *o circuito é o mesmo*. A nossa interpretação é que é diferente.

A dificuldade com que nos deparamos provém de querermos forçar uma das interpretações. A única solução razoável para este dilema consiste em, *quando estivermos a lidar com os circuitos lógicos, prescindir totalmente da lógica positiva e da lógica negativa e, consequentemente, das tabelas de verdade lógicas,*

dos "0"s e dos "1"s, e ficarmos pelas tabelas de verdade físicas dos circuitos e pela lógica de polaridade.

Isto é, passamos a distinguir claramente entre:

Valores lógicos 0 e 1

— os **valores lógicos** 0 e 1 que possuem as variáveis booleanas simples e as funções booleanas simples, quando estas últimas são representadas por expressões booleanas como, por exemplo,

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

ou por tabelas de verdade lógicas, como em

x	y	f(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

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

ou por quadros de Karnaugh como em

x \ y	0	1
0	0	1
1	1	0

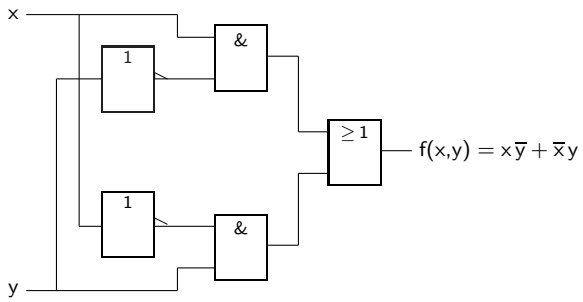
$$\begin{aligned} f(x,y) &= x\,\overline{y} + \overline{x}\,y + \overline{\overline{x}\,\overline{y}} \\ &= x\,\overline{y} + \overline{x}\,y \end{aligned}$$

Contexto algébrico

isto é, no **contexto algébrico**; e

Níveis de tensão H e L

— os **níveis de tensão** H e L nos circuitos lógicos que representam essas variáveis e funções, e nas suas representações por logigramas como em



Contexto físico

ou nos correspondentes esquemas eléctricos e implementações físicas numa determinada tecnologia, ou seja, no **contexto físico**.

Desta forma asseguramos que o contexto físico dos circuitos é totalmente preservado a partir do contexto algébrico, no processo de implementação dos circuitos lógicos e das suas representações por logigramas e esquemas eléctricos. Mas, obviamente, teremos de cuidar do interface entre o contexto algébrico e o físico, tornando fácil e natural a passagem de um para o outro. É o que faremos já de seguida.

7.1.4 Portas lógicas em lógica de polaridade

Em lógica de polaridade existem 2^n variantes de um determinado circuito (porta lógica, por exemplo) com n entradas e saídas. Por exemplo, existem 8 variantes de portas AND com 2 entradas, 4 variantes de portas Buffer (apenas com uma entrada e uma saída), 16 variantes de portas OR com 3 entradas, etc.

Símbolos IEC das portas AND com 2 entradas

Como exemplo, consideremos as 8 *variantes de portas AND com* 2 *entradas*. Os símbolos das portas são construídos colocando ou não indicadores de polaridade (qualificadores $\triangleright$) nas entradas e na saída dos ANDs, como sugere a Figura 7.4.

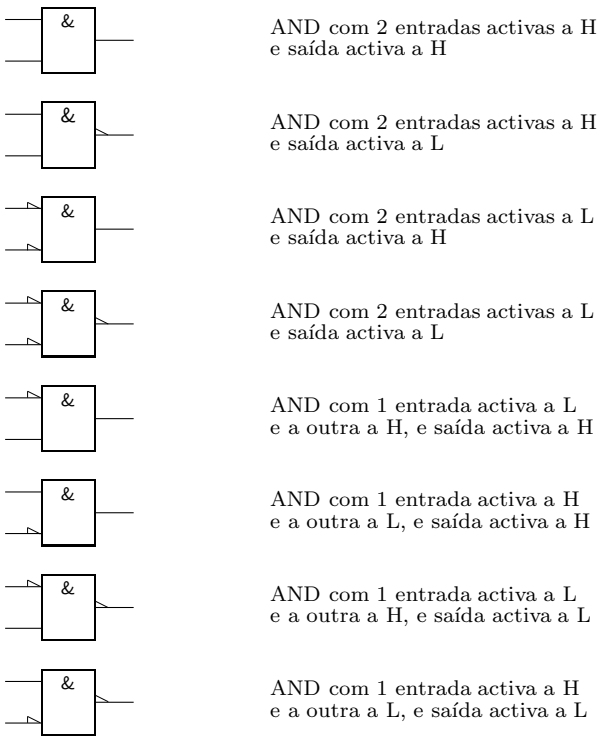


Figura 7.4: Símbolos IEC das 8 variantes de portas AND com 2 entradas

Nível de actividade

Naturalmente, temos de distinguir estas 8 variantes, *já que se trata de* 8 *circuitos distintos*. A maneira de as distinguir é pelo **nível de actividade** das entradas e das saídas.

Entrada ou saída activa a H ou a L

Para tanto, *convenciona-se que a existência do qualificador* $\triangleright$ *numa entrada ou numa saída torna essa entrada ou essa saída* **activa a L**, *enquanto que a não existência do qualificador torna a entrada ou saída* **activa a H**.

Assim, por exemplo, a primeira porta designa um AND com as entradas activas a H e a saída activa a H, enquanto que a segunda porta é um AND com as entradas activas a H e a saída activa a L; etc. Notemos, em particular, que *a presença do qualificador ▷ numa entrada ou numa saída não significa negação*, mas apenas actividade a L. Daí que, por exemplo, *não falemos em NAND para o segundo símbolo da Figura 7.4, mas em porta AND com entradas activas a H e saída activa a L.* De facto, a designação NAND (ou, pela mesma razão, a designação NOR) apenas faz sentido em lógica positiva ou em lógica negativa, já que sugere a existência de uma *negação* à saída do AND (do OR), o que *não se verifica* com a presença do qualificador ▷ .

Da mesma forma poderíamos estabelecer os 8 símbolos de portas OR com 2 entradas, ou os 16 símbolos de portas AND com 3 entradas, etc. (a questão dos 4 símbolos de Buffers, com uma entrada e uma saída, será estudada mais à frente).

7.1.5 Tabelas de verdade genéricas e físicas

Agora que já sabemos identificar as diversas portas do mesmo tipo, importa colocar a seguinte questão: dado o símbolo IEC de uma porta, como é que ela pode ser implementada, por exemplo em TTL (que usa preferencialmente uma lógica positiva) ou em ECL (que usa preferencialmente uma lógica negativa)?

Para responder à questão, precisamos de estabelecer a tabela de verdade física de cada uma das portas e a correspondente tabela de verdade lógica, em lógica positiva ou negativa. Como já sabemos gerar a tabela de verdade lógica a partir da correspondente tabela de verdade física, temos que aprender a gerar a tabela de verdade física de uma porta a partir do seu símbolo IEC.

Para o podermos fazer, devemos colocar previamente uma outra questão: se temos 2^n símbolos IEC para outras tantas portas do mesmo tipo com n entradas e saídas, como é que elas se relacionam do ponto de vista físico? De facto, deve haver uma relação, dado que os seus símbolos se obtêm uns dos outros por inclusão ou remoção de indicadores de polaridade.

Tabela de verdade genérica

Por exemplo, dado que há 8 portas AND com 2 entradas, o que é que há de comum e de diferente para todas elas? O que há de comum pode traduzir-se numa **tabela de verdade genérica** para todos os 8 ANDs com 2 entradas.

Tabela de verdade genérica das portas AND com 2 entradas

Essa tabela (Tabela 7.2) diz em que condições é que a saída do AND vem activa (A), por definição de AND (ou, para sermos mais precisos, por definição de produto lógico, adaptado a estas circunstâncias): apenas quando as duas entradas estiverem activas. Basta que uma delas esteja inactiva (I) para a saída também vir inactiva.

Para passarmos desta tabela de verdade genérica — para todos os ANDs de 2 entradas — à tabela de verdade física de um AND em particular, basta considerar os respectivos níveis de actividade nas entradas e nas saídas.

Por exemplo, na Figura 7.5 ilustra-se a obtenção da tabela de verdade física de um AND com 2 entradas activas a L e saída activa a H. Como as entradas X e Y são activas a L, onde se encontra um A nas correspondentes colunas da tabela de verdade genérica é colocado um L na tabela de verdade física, e onde

Tabela 7.2: Tabela de verdade genérica para todas as portas AND com 2 entradas

X	Y	Z
I	I	I
I	A	I
A	I	I
A	A	A

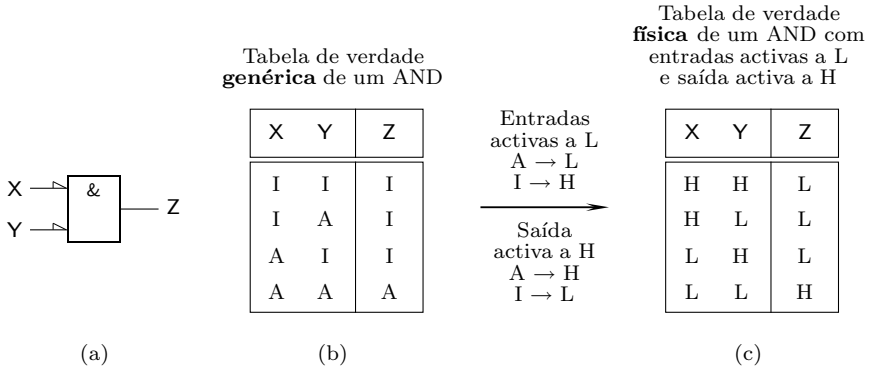


Figura 7.5: Porta AND com 2 entradas activas a L e saída activa a H; (a) símbolo IEC; (b) tabela de verdade genérica, válida para todos os ANDs com 2 entradas; e (c) tabela de verdade física para o AND da parte (a)

se encontra um I coloca-se um H. Por outro lado, como a saída Z é activa a H, onde estiver um A coloca-se um H e onde estiver um I põe-se um L.

A partir da tabela de verdade física podemos obter em seguida a tabela de verdade em lógica positiva e em lógica negativa para a porta lógica em questão (Figura 7.6).

Finalmente, destas tabelas concluímos estar em presença de um NOR em lógica positiva, e de um NAND em lógica negativa.

Vamos agora ver a questão da representação da porta das Figuras 7.5(a) e 7.6 em lógica de polaridade, em lógica positiva e em lógica negativa.

Relembremos, mais uma vez, que qualquer porta possui dois símbolos. Assim sendo, a porta da Figura 7.5(a) vem representada, em lógica de polaridade, pelo símbolo de um AND com 2 entradas activas a L e saída activa a H ou, em alternativa, pelo símbolo de um OR com 2 entradas activas a H e saída activa a L, como já sabemos da Subsecção 7.1.1:

Mas como a porta em questão é representada por um NOR em lógica positiva,

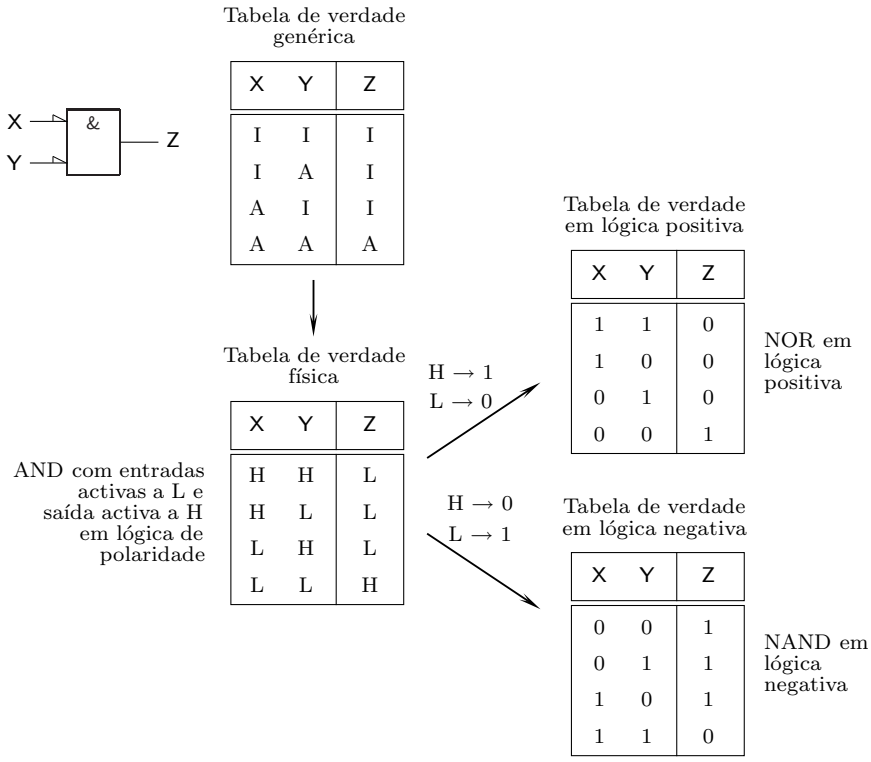
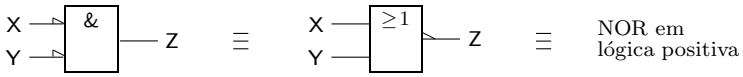
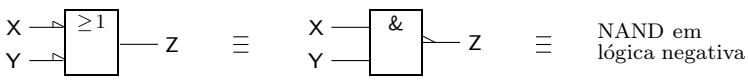


Figura 7.6: A partir da tabela de verdade física de uma porta AND com 2 entradas activas a L e saída activa a H, obtemos as correspondentes tabelas de verdade em lógica positiva e em lógica negativa

concluímos que a porta vem representada, nesta lógica, pelos dois símbolos alternativos de um NOR (um NOR propriamente dito ou um AND com as entradas negadas), isto é, os mesmos símbolos da sua representação em lógica de polaridade:



A representação simbólica é, contudo, diferente em lógica negativa. Agora a porta vem representada pelos símbolos alternativos de um NAND (um NAND propriamente dito, ou um OR com as entradas negadas):



Naturalmente, este exercício pode ser estendido identicamente a qualquer outra porta. Daqui resulta o seguinte processo de conversão simbólica: *dados os símbolos de uma porta em lógica de polaridade, podemos obter os símbolos cor-*

respondentes em lógica positiva sem fazer alteração nenhuma; para obter os símbolos em lógica negativa, podemos fazer uma de duas coisas: (i) mantemos a função (AND ou OR) e trocamos as polaridades nas entradas e na saída; ou (ii) mantemos as polaridades e trocamos a função (AND por OR ou vice-versa).

Por outro lado, *para obtermos o símbolo alternativo numa determinada lógica, trocamos a função (AND por OR e vice-versa) e também trocamos as polaridades*. Por exemplo,

ou

como tinhamos visto anteriormente na Secção 7.1.1 quando se afirmou que estas representações alternativas são consequência das leis de Morgan.

De notar como se distinguem cuidadosamente as designações possíveis *para a mesma porta*: desenhamos a porta em lógica de polaridade atendendo às polaridades das entradas e da saída e à função (AND ou OR). Em lógica positiva ou negativa designamos as funções por NAND ou por NOR (tais funções *não existem* em lógica de polaridade), e falamos em *negações* nas entradas e nas saídas, em vez de polaridades.

Tabela de verdade genérica das portas AND

Em resumo, em lógica de polaridade temos portas AND com n entradas, todas com a tabela de verdade genérica da Tabela 7.3, (com a saída activa só quando todas as entradas estiverem activas), que podemos instanciar em 2^{n+1} portas AND físicas distintas, consoante os níveis de actividade nas suas entradas e saídas.

Tabela 7.3: Tabela de verdade genérica de todas as portas AND com n entradas

E_1	E_2	E_3	$\cdots$	E_{n-1}	E_n	S
I	I	I	$\cdots$	I	I	I
I	I	I	$\cdots$	I	A	I
I	I	I	$\cdots$	A	I	I
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
A	A	A	$\cdots$	A	I	I
A	A	A	$\cdots$	A	A	A

Tabela de verdade genérica das portas OR

Da mesma forma, temos portas OR com n entradas, todas com a tabela de verdade genérica da Tabela 7.4, (com a saída activa desde que pelo menos uma entrada esteja activa), que podemos instanciar em 2^{n+1} portas OR físicas distintas, consoante os níveis de actividade nas suas entradas e saídas.

Tabela 7.4: Tabela de verdade genérica de todas as portas OR com n entradas

E_1	E_2	E_3	$\cdots$	E_{n-1}	E_n	S
I	I	I	$\cdots$	I	I	I
I	I	I	$\cdots$	I	A	A
I	I	I	$\cdots$	A	I	A
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
A	A	A	$\cdots$	A	I	A
A	A	A	$\cdots$	A	A	A

Tabela de verdade genérica dos Buffers

Finalmente, temos 4 variantes de Buffers, que são portas com uma entrada e uma saida com a tabela de verdade genérica da Tabela 7.5. Para todas elas, a saída vem activa se a entrada estiver activa.

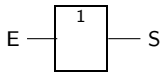
Tabela 7.5: Tabela de verdade genérica para todas as portas Buffer

E	S
I	I
A	A

Se agora instanciarmos as 4 variantes que resultam de atribuirmos níveis de actividade à entrada e à saída, obtemos as seguintes portas lógicas:

— um Buffer com a entrada e a saída activas a H, com o símbolo IEC e a tabela de verdade física que se seguem.

Buffer com a entrada e a saída activas a H

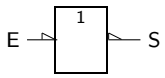


E	S
L	L
H	H

Do ponto de vista lógico, este Buffer reproduz na saída o nível de tensão que tiver sido aplicado à entrada; apesar da aparente inutilidade lógica desta porta, ela irá ser útil mais à frente, quando estudarmos a utilização da sua variante com saída tri-state.

— um Buffer com a entrada e a saída activas a L, com o símbolo IEC e a tabela de verdade física que se seguem.

Buffer com a entrada e a saída activas a L

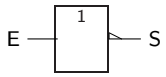


E	S
H	H
L	L

Tal como o anterior, este Buffer reproduz na saída o nível que for aplicado à entrada, tendo os dois a mesma utilidade.

— um Buffer com a entrada activa a H e a saída activa a L, com o símbolo IEC e a tabela de verdade física que se seguem.

Buffer com a entrada activa a H e a saída activa a L (porta NOT ou conversor de polaridade)



E	S
L	H
H	L

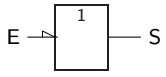
Trata-se de uma porta que umas vezes se comporta como uma **porta NOT**, ou **inversor** ou **negação**, e outras vezes como um **conversor de polaridade** (esta última designação será tornada clara mais à frente).

Porta NOT (inversor, negação)

Conversor de polaridade

— finalmente, um Buffer com a entrada activa a L e a saída activa a H, com o símbolo IEC e a tabela de verdade física que se seguem.

Buffer com a entrada activa a L e a saída activa a H (porta NOT ou conversor de polaridade)



E	S
H	L
L	H

Tal como o Buffer anterior, temos agora também uma porta NOT (por vezes, um conversor de polaridade), com o símbolo alternativo.

Também por vezes se usa o símbolo de uma porta XOR em lógica de polaridade (a seguir ilustrado com 2 entradas activas a L e a saída activa a H).

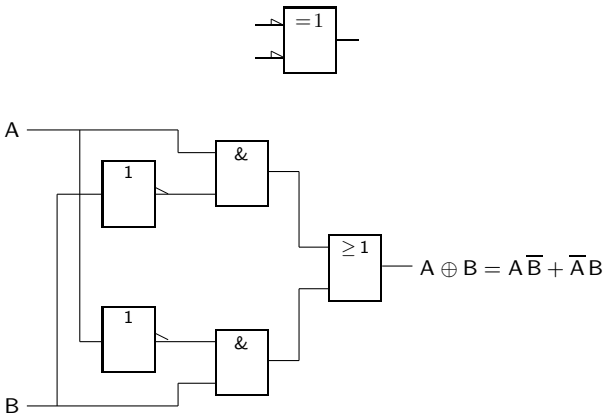


Figura 7.7: Circuito equivalente a uma porta XOR

Dado que uma porta XOR pode ser sempre ser reduzida, por exemplo, ao circuito lógico da Figura 7.7 (mas há outros circuitos representativos de um XOR), o símbolo desta porta justifica-se sobretudo como forma compactada do referido circuito. Há, contudo, uma outra razão para o uso desse símbolo: na tecnologia CMOS uma porta XOR não é implementada pelo cicrcuito anterior, ou equivalente, antes recorrendo a um tipo de portas diferentes designadas por *portas*

de transmissão; nessas condições, justifica-se a utilização do símbolo do XOR como representação de uma porta básica, ao mesmo nível do das portas AND e OR.

7.1.6 Logigramas e esquemas eléctricos em lógica de polaridade

Logigrama

Num **logigrama** desenhado em lógica de polaridade é legítimo incluir portas complexas, com diferentes níveis de actividade, como a que se indica na Figura 7.8(a).

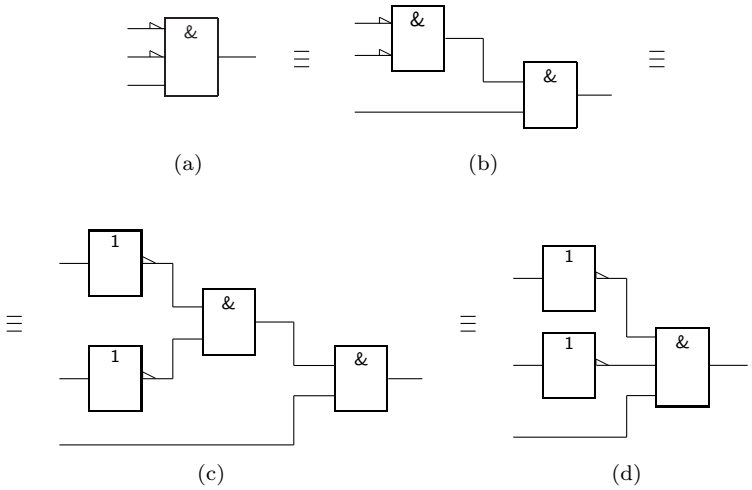


Figura 7.8: (a) Porta complexa, para implementar em TTL; (b) uma primeira transformação, que substitui a porta AND com 3 entradas por duas portas AND com 2 entradas cada uma; (c) uma segunda transformação, que desloca os indicadores de polaridade da entrada do AND para a saída de dois Buffers, transformando-os em portas NOT; e (d) uma terceira transformação, que substitui os dois ANDs anteriores por um único

Esquema eléctrico

Porém, quando queremos implementar a porta e passamos ao **esquema eléctrico** numa determinada tecnologia, deixamos de poder utilizar este símbolo se ele não possuir contrapartida directa num circuito integrado ou em parte dele; nessas condições, temos de desenhar pormenorizadamente o *circuito* equivalente à porta.

Em TTL existem portas AND com 3 entradas, mas as entradas devem ter todas os mesmos níveis de actividade: se os níveis de actividade forem H, a porta é um AND ou un NAND em lógica positiva; se forem L, a porta é um NOR ou um OR em lógica positiva.

Por exemplo, em TTL não existe a porta da Figura 7.8(a) sob a forma de um circuito integrado. Para obter o circuito equivalente procedemos por etapas. Como em TTL existem portas com 2 entradas com os mesmos níveis de actividade, uma transformação possível é a que se ilustra na Figura 7.8(b).

Se em seguida colocarmos nas linhas de entrada dois Buffers com entradas e saídas activas a H, a função não vem alterada. E se depois deslocarmos os indicadores de polaridade das entradas do primeiro AND para as saídas dos Buffers, que são então convertidos em portas NOT, obtemos o logigrama da Figura 7.8(c) sem alterar a função.

Finalmente, se substituirmos os dois ANDs de 2 entradas por um único AND com 3 entradas activas a H, obtemos o logigrama da Figura 7.8(d).

Como os três logigramas que obtivemos têm contrapartida sob a forma de circuitos integrados TTL, todos são passíveis de ser utilizados em esquemas eléctricos. Por exemplo, o logigrama da Figura 7.8(b) dá origem ao esquema eléctrico da Figura 7.9, formado por 1/4 de um NOR em lógica positiva (do tipo 74LS02), e por 1/4 de um AND em lógica positiva (do tipo 74LS08).

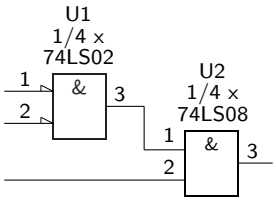


Figura 7.9: Esquema eléctrico que corresponde ao logigrama da Figura 7.8(b), e que utiliza portas lógicas da sub-família "Low Power Schottky" TTL

Para finalizarmos esta questão, vamos estabelecer a tabela de verdade física da porta complexa da Figura 7.8(a). Porque se trata de um AND com 3 entradas, podemos desde logo gerar a sua tabela de verdade genérica (Tabela 7.6).

Tabela 7.6: Tabela de verdade genérica de um AND com 3 entradas

A	B	C	S
I	I	I	I
I	I	A	I
I	A	I	I
I	A	A	I
A	I	I	I
A	I	A	I
A	A	I	I
A	A	A	A

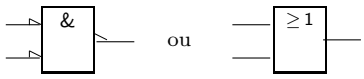
Porque se trata de um AND com duas entradas activas a L e uma entrada activa a H, e saída activa a H, podemos converter a tabela de verdade genérica na tabela de verdade física da porta em questão (Tabela 7.7, onde se admitiu, arbitrariamente, que as variáveis A e B vêm ligadas às entradas activas a L e que a variável C vem ligada à entrada activa a H).

Evidentemente, em geral deparamo-nos com ciruitos mais complexos do que a porta da Figura 7.8(a). Porém, os princípios que nos permitem obter as tabelas de verdade físicas desses circuitos, bem como as suas implementações numa determinada tecnologia, são os mesmos que se acabaram de apresentar.

Um outro problema que se põe é o seguinte: *dado que dispomos, para cada porta, de dois símbolos alternativos, qual deles devemos utilizar em cada caso*? Por exemplo, qual dos símbolos alternativos

Tabela 7.7: Tabela de verdade física da porta da Figura 7.8(a), onde se admite que as variáveis A e B vêm ligadas às entradas activas a L e que a variável C vem ligada à entrada activa a H

A	B	C	S
H	H	L	L
H	H	H	L
H	L	L	L
H	L	H	L
L	H	L	L
L	H	H	L
L	L	L	L
L	L	H	H



devemos usar?

A resposta a esta questão depende da forma do circuito que estamos a sintetizar. Por exemplo, se estivermos a representar o logigrama de uma função booleana simples em forma disjuntiva (soma de produtos) devemos usar o símbolo que explicite que estamos a *somar logicamente* um certo número de produtos lógicos, como se ilustra na Figura 7.10.

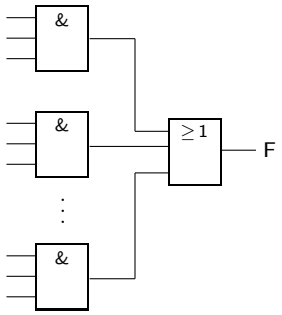


Figura 7.10: Logigrama (e esquema eléctrico simplificado) a utilizar quando se representa uma soma de produtos

E mesmo que, por razões de oportunidade, estejamos a implementar esse circuito com NANDs em lógica positiva, devemos usar *a mesma estrutura* no logigrama e no esquema eléctrico, como se indica na Figura 7.11(a).

Ou seja, descartamos a possibilidade de usar o logigrama da Figura 7.11(b) porque, embora correcto do ponto de vista lógico e, tal como o da Figura 7.11(a), equivalente ao do da Figura 7.10, não acentua o facto de estarmos a representar uma *soma de produtos*.

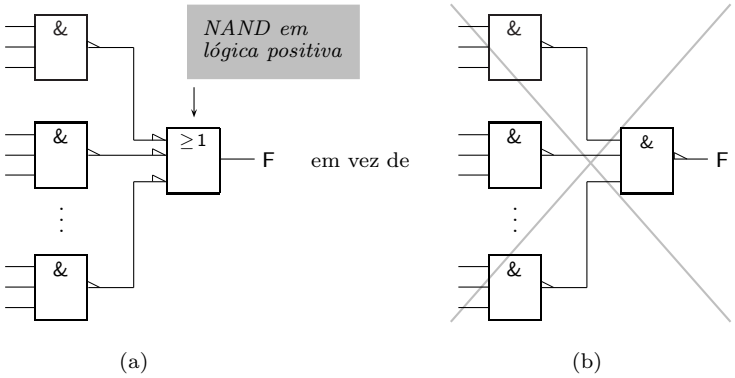


Figura 7.11: (a) Logigrama (e esquema eléctrico simplificado) a utilizar quando se representa uma soma de produtos implementada com NANDs em lógica positiva; (b) embora este logigrama esteja correcto do ponto de vista lógico e seja, como o anterior, equivalente ao logigrama da Figura 7.10, não acentua que estamos a representar uma soma de produtos, pelo que *não deve ser utilizado*

Por outro lado, podemos alterar o logigrama da Figura 7.10 substituindo os ANDs com entradas e saídas activas a H por ORs com entradas e saídas activas a L, e o OR com entradas e saída activa a H por um AND com entradas e saída activa a L, como se faz na Figura 7.12 (b).

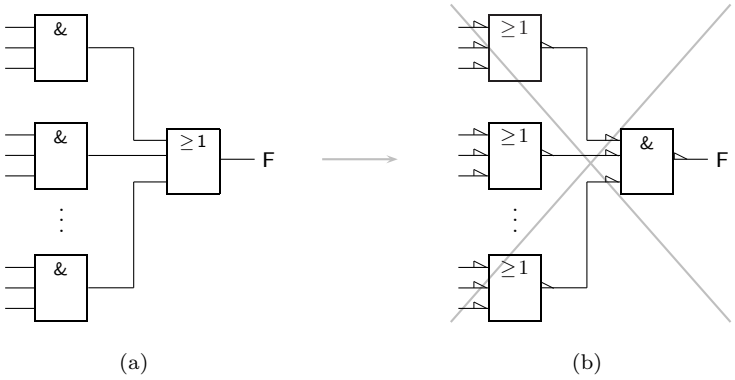


Figura 7.12: (a) Logigrama (e esquema eléctrico simplificado) inicial; (b) logigrama equivalente e correcto do ponto de vista lógico, mas que também não acentua o facto de estarmos a representar uma soma de produtos, pelo que *não deve ser utilizado*

Mas, mais uma vez, nesse logigrama não se acentua o facto de estarmos a representar uma *soma de produtos*, pelo que não o devemos utilizar.

Identicamente, se a função for dada em forma conjuntiva (produto de somas) como na Figura 7.13(a), podemos transformar o seu logigrama (e esquema eléctrico parcial) como na Figura 7.13(b), implementando-a com NORs em lógica positiva, porque preservámos a estrutura da função.

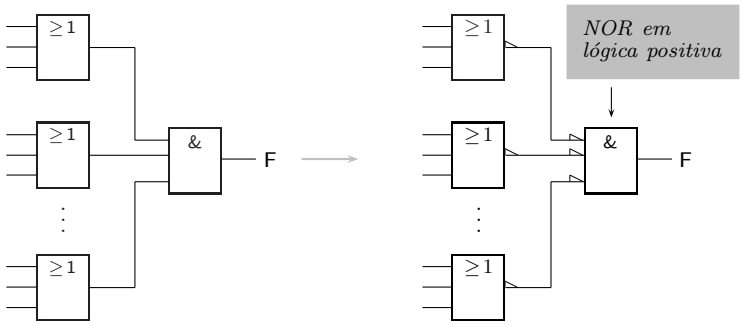


Figura 7.13: (a) Logigrama (e esquema eléctrico simplificado) de uma função em forma conjuntiva (produto de somas); (b) logigrama equivalente e correcto do ponto de vista lógico, que reflecte o facto de estarmos a lidar com um produto de somas, embora implementado com NORs em lógica positiva

7.2 Conteúdo Semântico

Até agora utilizámos designações para as variáveis e funções booleanas simples, por exemplo

$$A, B, C, x, y, z, \ldots ,$$

com designações que não ilustram as *acções* a elas associadas. Ou seja, procurámos dar designações simplificadas às variáveis e funções, a fim de as poder utilizar eficazmente nas expressões lógicas, nos quadros de Karnaugh, etc.

Quando apenas temos de lidar com os aspectos algébricos dos sistemas lógicos é vantajoso reter esses tipos de designações. Contudo, quando passamos ao nível físico e à implementação dos circuitos lógicos, torna-se imprescindível aprofundar esta questão, passando a conferir conteúdo semântico às variáveis a às funções.

Explicitemos melhor o significado desta afirmação. Se uma determinada variável (ou função) booleana simples identificar uma situação em que uma certa porta está aberta, por exemplo, será mais interessante designá-la por

$$PORTA.ABERTA$$

Conteúdo semântico

ou por outra designação semelhante, em vez de A ou de x. Diz-se, nessas circunstâncias, que se conferiu **conteúdo semântico** à variável (ou à função) em jogo.

Sensor (entrada)

Para percebermos melhor o significado destes conteúdos, comecemos por considerar um **sensor** de entrada para o alarme de um carro, alarme esse que deve detectar a situação em que a porta do condutor se encontra aberta.

Vamos concentrar a nossa atenção no sensor e na ligação ao alarme.

O sensor é constituído por um comutador de duas posições, por uma resistência de "pull-up" ligada à tensão de alimentação, V_{cc}, e por uma porta NOT (Figura 7.14).

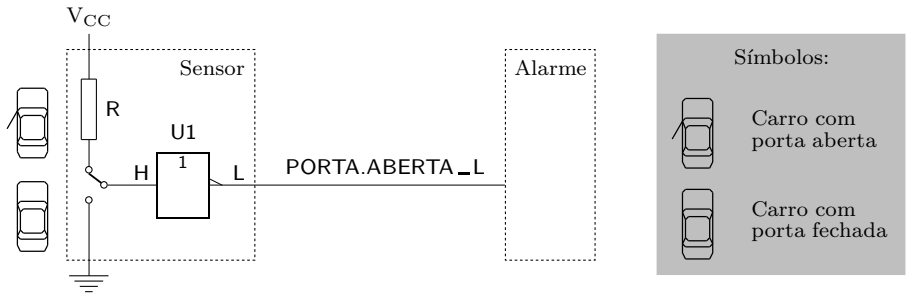


Figura 7.14: Sensor de alarme de um carro: quando a porta do condutor está aberta, na linha de saída do sensor aparece um nível L

Se o comutador estiver ligado à resistência (o que corresponde, com esta montagem, à situação de porta aberta), teremos na sua saída uma tensão próxima de V_{cc} e, na saída do NOT, um nível L adequado para aplicação à entrada do circuito de alarme.

Se, pelo contrário, o comutador estiver ligado à massa (e, portanto, a porta está fechada), teremos uma tensão nula à entrada do NOT e, em consequência, um nível H na sua saída (Figura 7.15).

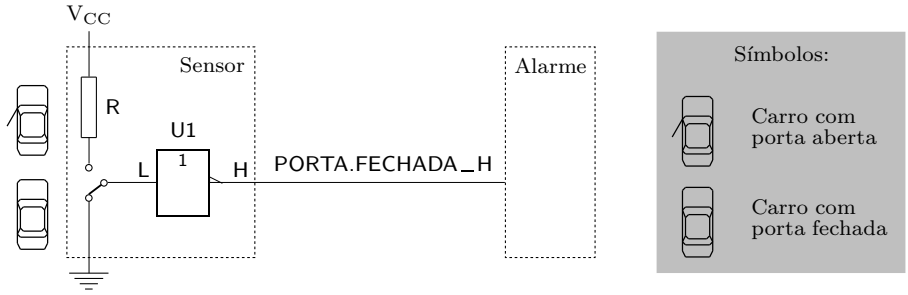


Figura 7.15: O mesmo sensor de alarme da figura anterior: quando a porta do condutor está fechada, na linha de saída do sensor aparece um nível H

A designação "linha" deve ser tomada num sentido lato, incluindo nela toda e qualquer ligação contida num logigrama ou esquema eléctrico, seja ela representativa de um fio num protótipo em "breadboard", de uma pista de um circuito impresso ou de um circuito integrado, etc.

O problema que se põe reside, então, *na designação a dar à linha de saída do sensor e à variável que nela se suporta.*

Em relação à designação da variável, podemos escolher uma de duas em alternativa,

$$PORTA.ABERTA \qquad \text{ou} \qquad PORTA.FECHADA \ ,$$

consoante a ligação que fizermos do sensor à porta do carro. Para a ligação que se ilustra na Figura 7.14, ou seja, em que a posição de porta aberta corresponde à situação em que o comutador está ligado ao V_{cc}, com um nível L à saída do NOT sempre que a porta estiver aberta (naturalmente, podíamos ter optado pela ligação contrária), a designação da linha deverá ser

$$PORTA.ABERTA\_\text{L} \ .$$

Variável activa

Quere-se com esta designação afirmar que, quando o nível de tensão na linha é L, a porta está aberta e a **variável** $PORTA.ABERTA$ está **activa**. A designação anterior deve identificar, então, a variável $PORTA.ABERTA$ como sendo **activa a L**.

Variável activa a L

Variável inactiva

Naturalmente, se nestas circunstâncias o nível na linha for H é porque o comutador está na outra posição, a porta do carro está fechada (admitimos que a porta apenas pode encontar-se numa destas duas posições, já que estamos a lidar com variáveis binárias), e a **variável** $PORTA.ABERTA$ está **inactiva**.

Na Figura 7.15 ilustra-se a situação alternativa, em que damos a outra designação semântica à variável, que se passa a chamar $PORTA.FECHADA$. Para que o problema seja exactamente o mesmo da Figura 7.14, devemos agora designar a linha por

$$PORTA.FECHADA\_\mathrm{H} \ ,$$

Variável activa a H

o que significa que o carro tem a porta fechada quando na linha tivermos um nível H e aberta para o nível L. Neste caso, a designação anterior deve, então, identificar a variável $PORTA.FECHADA$ como sendo **activa a H**.

Em resumo, verifica-se que a linha que serve de suporte a uma variável tem pelo menos duas designações em alternativa, para as mesmas condições no circuito, sendo opcional a escolha entre as duas.

Constata-se ainda que a designação da linha é composta pela designação da variável seguida do nível de tensão em que a variável vem activa. Por exemplo, a linha com a designação

$$PORTA.ABERTA\_\mathrm{L}$$

suporta a variável

$$PORTA.ABERTA \ ,$$

que é activa a L.

Função activa (a H, a L)

Actuador (saída)

O que se acabou de dizer para uma variável de *entrada* de um circuito lógico é igualmente válido para uma **função** de *saída* e para a sua ligação a um **actuador** (actuador esse tomado num sentido lato, incluindo indicadores luminosos).

Finalmente, dado que

$$\overline{PORTA.ABERTA} \equiv PORTA.FECHADA$$

e

$$\overline{PORTA.FECHADA} \equiv PORTA.ABERTA \ ,$$

concluímos que podemos usar, para a linha em questão, não duas mas quatro opções,

$$\begin{array}{l} PORTA.ABERTA\_\mathrm{L} \\ PORTA.FECHADA\_\mathrm{H} \\ \overline{PORTA.ABERTA}\_\mathrm{H} \\ \overline{PORTA.FECHADA}\_\mathrm{L} \ , \end{array} \tag{7.1}$$

escolhendo-se em cada situação a mais adequada.

Assim, se na designação da linha pretendermos acentuar o facto de o nível L indicar uma porta aberta, como na Figura 7.14, utilizaremos

$$PORTA.ABERTA\_\text{L} \quad \text{ou} \quad \overline{PORTA.FECHADA}\_\text{L} \ .$$

Se, pelo contrário, quisermos realçar que é o nível H que deve indicar porta fechada, como na Figura 7.15, utilizaremos

$$PORTA.FECHADA\_\text{H} \quad \text{ou} \quad \overline{PORTA.ABERTA}\_\text{H} \ .$$

Não esqueçamos, porém, que temos uma situação completamente diferente se quisermos que o nível de tensão H indique porta aberta e que o nível L indique porta fechada. Nesse caso teremos:

$$\begin{array}{l} PORTA.ABERTA\_\text{H} \\ PORTA.FECHADA\_\text{L} \\ \overline{PORTA.ABERTA}\_\text{L} \\ \overline{PORTA.FECHADA}\_\text{H} \ , \end{array} \tag{7.2}$$

que é formalmente diferente do conjunto de designações (7.1). Estas designações são aplicáveis à situação oposta à que foi representada nas Figuras 7.14 e 7.15, com o sensor a indicar porta aberta quando ligado à massa.

Uma consequência importante do facto de podermos, em geral, designar qualquer linha por uma de quatro maneiras diferentes, mas equivalentes, é que as variáveis ou funções que se suportam nessa linha possuem níveis de actividade que *não estão correlacionados com os níveis de actividade nas entradas e nas saídas nos símbolos das portas às quais a linha se liga*, como mostra a Figura 7.16 para um caso particular.

7.2.1 Buffers, inversores e conversores de polaridade

As múltiplas designações que podemos dar a uma determinada linha permite-nos ainda justificar a afirmação, feita a propósito da tabela de verdade genérica de um Buffer, na Tabela 7.5, página 118, de que as portas Buffer que possuem níveis de actividade diferentes nas entradas e nas saídas podem ser interpretados como portas NOT ou, alternativamente, como conversores de polaridade.

Com efeito, suponhamos que se aplica à entrada de uma dessas portas uma variável $ACTUA$ (naturalmente, sendo gerada por uma outra porta qualquer, poderá igualmente ser considerada como uma função), e que essa variável (ou função) é activa a L. A designação da linha de entrada da porta deverá, então, ser $ACTUA\_\text{L}$.

Se considerarmos que a porta se comporta como um NOT [Figura 7.17(a)], obtemos à saída a função $\overline{ACTUA}$ ou, em alternativa, a função $NAO.ACTUA$ — o complemento da função de entrada — com o mesmo nível de actividade. Ou seja, designamos a linha de saída por $\overline{ACTUA}\_\text{L}$ ou por $NAO.ACTUA\_\text{L}$.

Se, pelo contrário, interpretarmos a porta como um conversor de polaridade [Figura 7.17(b)], designamos a saída por $ACTUA\_\text{H}$, ou seja, usando a mesma

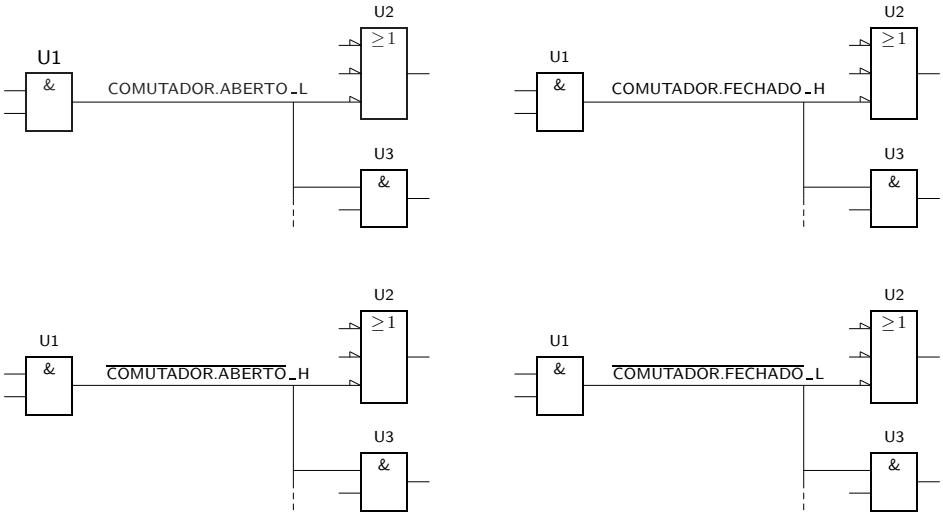


Figura 7.16: Quatro exemplos do *mesmo circuito*, com designações e níveis de actividade variados, porém totalmente equivalentes, para as funções booleanas simples geradas pela porta AND da esquerda. Como podemos constatar, os níveis de actividade das funções não estão correlacionados com o nível de actividade da saída da porta geradora da função nem com os níveis de actividade das entradas das portas à qual a função vem aplicada

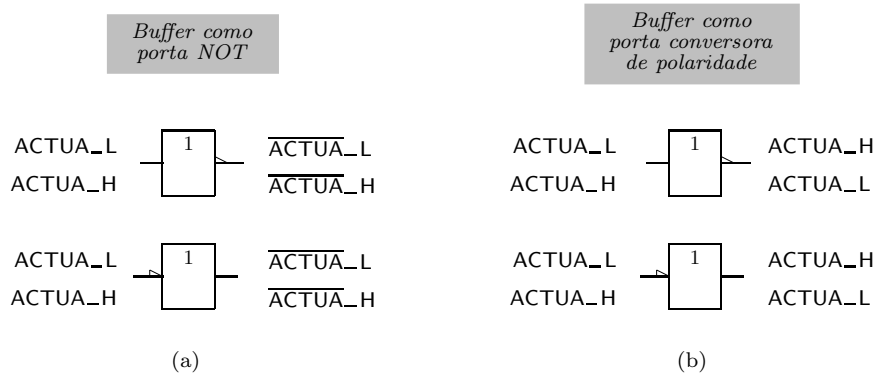


Figura 7.17: (a) Utilização de um Buffer como porta inversora (ou NOT); e (b) como conversor de polaridade

variável (ou função) de entrada, $ACTUA$, mas com a polaridade trocada. Em qualquer dos casos, a tabela de verdade genérica do Buffer é a da Tabela 7.5, e a tabela de verdade física do NOT ou do conversor de polaridade é igual nos dois casos.

Naturalmente, outro tanto se passaria se a variável (ou função) de entrada fosse activa a H, como aliás também se explicita na Figura 7.17.

7.3 Logigramas e Expressões Booleanas

Nesta Secção vamos abordar dois problemas típicos na lógica de polaridade: (i) dada a expressão booleana de uma dada função booleana simples, pretende-se obter o logigrama da função se forem conhecidos os níveis de actividade das variáveis independentes e da própria função; e (ii) o problema oposto, isto é, dado o logigrama de um circuito, pretende-se obter a expressão da função de saída e a correspondente tabela de verdade física.

7.3.1 Geração de logigramas

Vamos então supor que é conhecida a expressão booleana de uma dada função booleana simples e ainda os níveis de actividade das variáveis de entrada e da própria função. Pretende-se obter o logigrama da função.

Para abordar este problema consideremos o seguinte exemplo: é dada a função $F = \overline{A}\,B + A\,C$ e as variáveis A, B e C são geradas de tal forma que A_L, B_L, e C_H. Pretende-se ainda que F_L por razões que têm a ver com o circuito a jusante, ao qual F vai ser aplicada.

Começamos por gerar um logigrama para F *que não leva em linha de conta os níveis de actividade nas entradas e na saída*, como se ilustra na Figura 7.18.

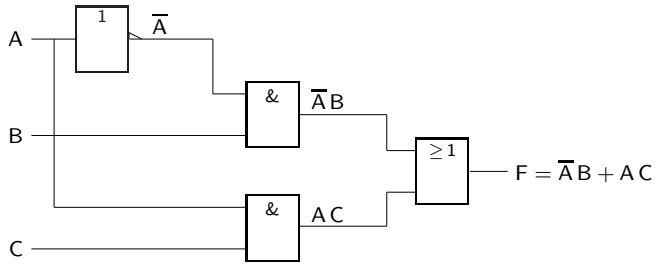


Figura 7.18: Logigrama da função $F = \overline{A}\,B + A\,C$ que não leva em linha de conta os níveis de actividade nas entradas e na saída

Em seguida inserimos no logigrama as variáveis e a função com níveis de actividade a H. Naturalmente, o logigrama não vem alterado por esta operação (Figura 7.19).

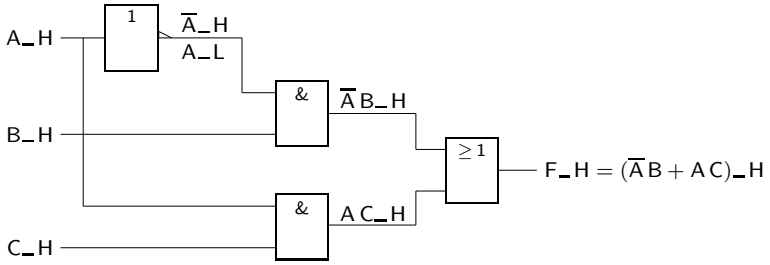


Figura 7.19: Inclusão, no logigrama anterior, das variáveis e da função com níveis de actividade a H

A terceira fase passa agora a tomar em consideração os níveis de actividade exigidos. Para tanto, temos de adaptar o logigrama obtido a esses níveis, ajustando os níveis de actividade das entradas das portas de entrada e da saída da porta de saída, como sugere a Figura 7.20.

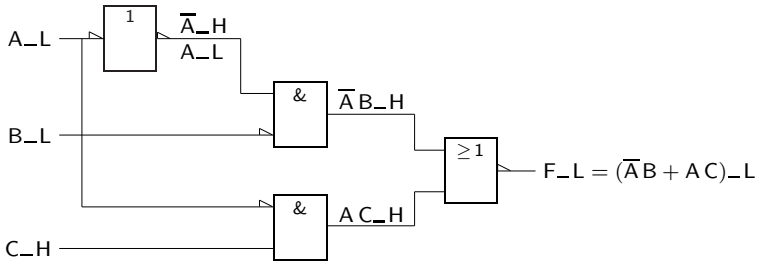


Figura 7.20: Logigrama modificado que leva em linha de conta os níveis de actividade correctos para as variáveis de entrada e para a função

Em relação ao logigrama obtido ressalta a redundância do Buffer com entrada e saída activa a L, que pode ser removido. Para reforçar esta ideia, notar que a entrada e a saída do Buffer possuem a mesma designação, A_L. Obtém-se, então, o logigrama da Figura 7.21, na sua forma final.

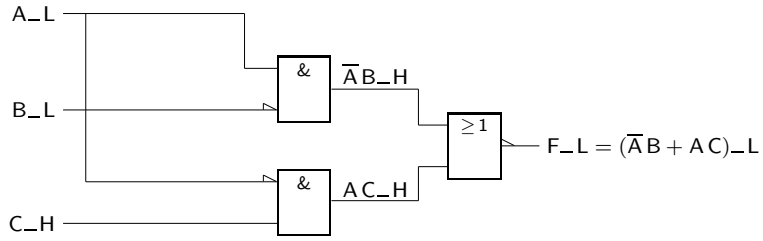


Figura 7.21: Logigrama final para a função

Embora legítimo, este logigrama não está descrito numa forma adequada para ser implementado fisicamente. Por outras palavras, *não é um esquema eléctrico*. Tal deve-se, como já sabemos, ao facto de não existirem portas AND integradas com níveis de actividade diferentes nas entradas.

Se quisermos implementar, por exemplo em TTL, o logigrama anterior, temos que modificá-lo *sem alterar a função*. Comecemos por incluir dois Buffers com entrada e saída activas a H nas linhas A_L e B_L (Figura 7.22), o que não altera a função.

Notemos que à saída dos Buffers temos funções com a mesma designação e nível de actividade das variáveis nas suas entradas.

Em seguida deslocamos os indicadores de polaridade que estão às entradas dos ANDs para as saídas dos Buffers, o que também não altera em nada a função. Ao fazê-lo, porém, passamos a ter ANDs que existem em TTL. Por outro lado, passamos, agora, a ter nas saídas dos Buffers as mesmas funções mas com a polaridade oposta, como mostra o esquema eléctrico da Figura 7.23. Nessas condições, os Buffers passam a funcionar como portas NOT ou como conversores de polaridade (iremos ver qual a designação mais adequada já a seguir).

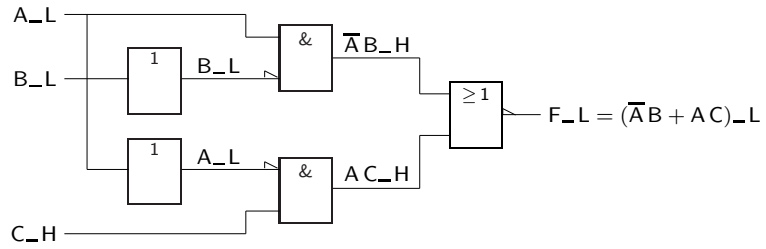


Figura 7.22: Alteração do logigrama anterior, que não altera a função

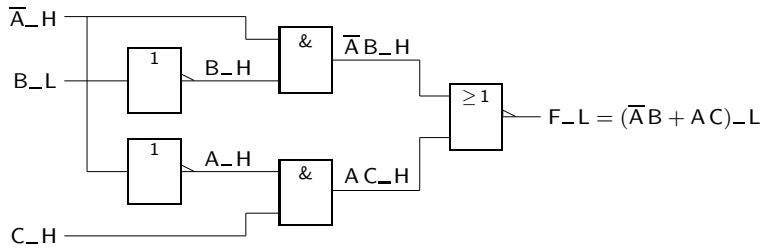


Figura 7.23: Esquema eléctrico para implementação da função dada

Finalmente, mudamos a designação A_L na entrada superior para $\overline{A}$_H. Esta nova designação, sendo equivalente, tem, contudo, o nível de actividade adaptado ao nível de actividade da entrada do AND a que se liga.

Este é um princípio fundamental para a correcta geração e leitura dos logigramas em lógica de polaridade: *de entre as várias designações alternativas que cada linha do logigrama pode ter, escolhemos sempre a alternativa que faz coincidir o nível de actividade da variável ou função que a linha suporta com o nível de actividade da entrada da porta a que a linha está ligada.*

Embora este procedimento não seja estritamente necessário, como o provam os exemplos da Figura 7.16, na página 128, esta forma de actuar garante a correcta interpretação das várias funções intermédias e final do logigrama. Por exemplo, lemos directamente a função $\overline{A}\,B$ à saída do AND superior (porque às suas entradas aplicamos a variável $\overline{A}$ e a função B), a função $A\,C$ à saída do AND inferior (porque temos A e C às suas entradas), e $F = \overline{A}\,B + A\,C$ à saída do esquema eléctrico (porque as entradas do OR são $\overline{A}\,B$ e $A\,C$).

Por outro lado, devemos notar a designação dada às linhas de saída dos Buffers, que suportam funções com níveis de actividade adaptados aos níveis de actividade das entradas dos ANDs a que se ligam. O Buffer que transforma B_L em B_H é um *conversor de polaridade*. O outro Buffer, que transforma $\overline{A}$_H em A_H, é uma *porta NOT*.

7.3.2 Expressões booleanas a partir de logigramas

Vamos agora considerar o problema inverso do anterior: dado um logigrama em lógica de polaridade, pretendemos determinar a expressão lógica da função de saída e a correspondente tabela de verdade física.

No que toca à obtenção da expressão lógica da função de saída procuramos, como se disse anteriormente, escolher para cada linha do logigrama a variável ou função mais adequada, que garanta a igualdade entre o nível de actividade dessa variável ou função e o nível de actividade da entrada da porta a que a linha se liga.

Consideremos o logigrama da Figura 7.24.

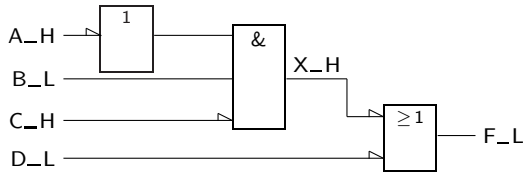


Figura 7.24: Logigrama que serve de exemplo

Notemos que o logigrama contém alguns dos desajustes que acabámos de mencionar. Por essa razão redesignamos algumas das linhas de entrada, mantendo a função e o logigrama (Figura 7.25).

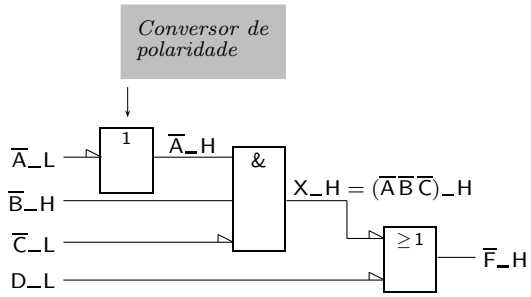


Figura 7.25: Logigrama que resulta do anterior por redesignação de algumas das linhas de entrada

Podemos agora constatar que à entrada do AND existem duas variáveis, $\overline{B}$ e $\overline{C}$, e uma função, $\overline{A}$, cujos níveis de actividade coincidem com os níveis de actividade das entradas da porta. Logo, a função X à saída do AND tem a expressão $X = \overline{A}\,\overline{B}\,\overline{C}$.

Como esta função tem o nível de actividade oposto ao nível da entrada do OR à qual a linha de suporte se liga, devemos redesignar $X_$H para $\overline{X}_$L, como mostra a Figura 7.26. Por idêntica razão, redesignamos também a função de saída, de $F_$L para $\overline{F}_$H, também como mostra a figura.

Nestas condições, e só nestas, o OR tem presente às suas entradas $D_$L e $\overline{X}_$L, gerando na sua saída $\overline{F}_$H a função $\overline{F} = \overline{X} + D$. Então,

$$\begin{aligned} F &= \overline{\overline{X} + D} \\ &= \overline{\overline{\overline{A}\,\overline{B}\,\overline{C}} + D}\,, \end{aligned}$$

que poderemos simplificar para $F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$.

No que diz respeito à geração da tabela de verdade física do circuito, temos que estender o que se fez anteriormente relativamente a este tipo de tabelas, e

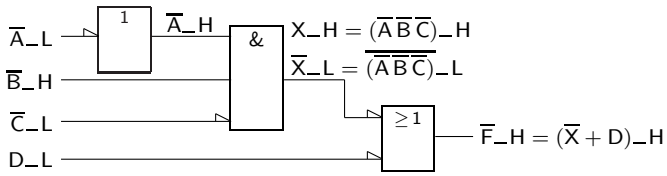


Figura 7.26: Logigrama que resulta do anterior por redesignação das linhas que suportam a função intermédia X e a função de saída F

incluir nela os níveis de actividade das variáveis de entrada e da função. Com efeito, só assim podemos analisar uma linha qualquer da tabela e perceber quais as variáveis que estão activas e, em consequência disso, deduzir o nível de tensão na saída percebendo, assim, se a função de saída está ou não activa.

Consideremos na Tabela 7.8 a tabela de verdade física do circuito da Figura 7.24. A tabela tem as linhas numeradas para facilitar a exposição do raciocínio.

Tabela 7.8: Tabela de verdade física do circuito da Figura 7.24

# linha	A_H	B_L	C_H	D_L	F_L
0	L	L	L	L	H
1	L	L	L	H	H
2	L	L	H	L	H
3	L	L	H	H	H
4	L	H	L	L	H
5	L	H	L	H	L
6	L	H	H	L	H
7	L	H	H	H	H
8	H	L	L	L	H
9	H	L	L	H	H
10	H	L	H	L	H
11	H	L	H	H	H
12	H	H	L	L	H
13	H	H	L	H	H
14	H	H	H	L	H
15	H	H	H	H	H

Podemos preencher a tabela linha a linha, deduzindo a actividade ou inactividade das entradas e saídas das portas e, por esse modo, deduzir os níveis de tensão nas diversas linhas do circuito, até chegarmos ao nível de tensão na linha que suporta a função de saída.

Por exemplo, a linha 0 da tabela impõe níveis L em todas as linhas de entrada. Ora com um L na linha A_H, a entrada do conversor de polaridade está activa, e, por conseguinte, a sua saída também deve estar activa (por definição de Buffer, *vd.* a Tabela 7.5). Como essa saída é activa a H, temos um H na linha

que liga essa saída à entrada superior do AND [Figura 7.27(a)].

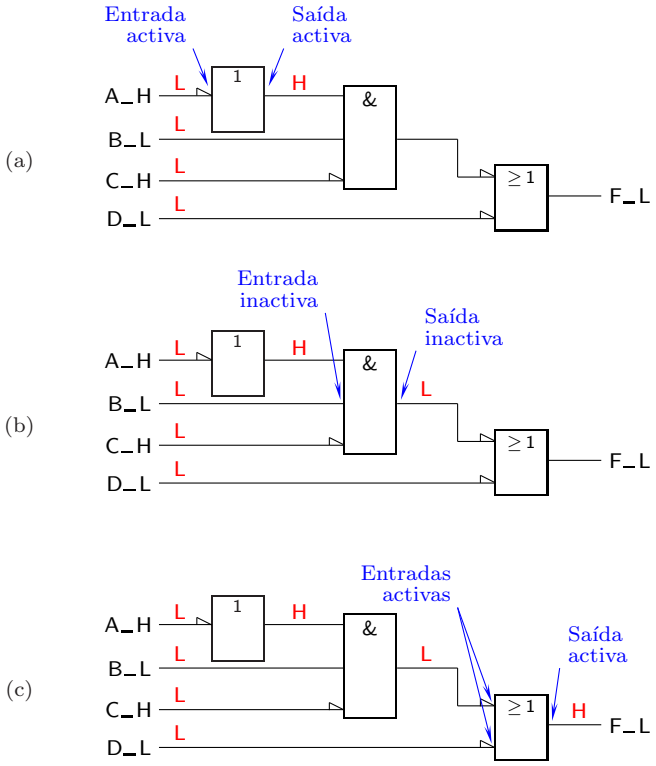


Figura 7.27: Estado das linhas do circuito quando aplicamos às entradas os níveis de tensão que correspondem à primeira linha da Tabela 7.8

Ou seja, o AND tem um nível H na linha superior e dois níveis L nas outras duas linhas. Isso quer dizer que a entrada superior e a entrada inferior estão activas, mas a entrada do meio está inactiva. Ora este facto é suficiente para tornar inactiva a saída do AND (por definição de AND, *vd.* a Tabela 7.3), e como essa saída é activa a H, significa que temos um L na linha de saída do AND [Figura 7.27(b)].

Segue-se que o OR tem as duas entradas activas, pelo que a sua saída deve estar activa (por definição de OR, *vd.* a Tabela 7.4). Como essa saída é activa a H, devemos ter um H na linha de saída [Figura 7.27(c)], como salienta a Tabela 7.8 na linha 0.

E, da mesma forma, poderíamos construir as restantes linhas da Tabela 7.8.

Mas também podemos preenchê-la globalmente e de forma mais expedita, se olharmos com atenção para o logigrama da Figura 7.24. Por exemplo, podemos constatar que o AND tem a saída inactiva, a L, desde que pelo menos uma das suas entradas esteja inactiva (por definição de AND). Ora isso acontece em três situações:

— a linha B_L está a L (e a entrada a que essa linha se liga é activa a H), o que acontece nas linhas 0 a 3 e 8 a 11 da tabela;

— a linha C_H está a H (e a entrada a que essa linha se liga é activa a L), o que acontece nas linhas 2 e 3, 6 e 7, 10 e 11, e 14 e 15 da tabela;

— a linha A_H está a H e, por conseguinte, a saída do Buffer está a L (e a entrada a que essa linha se liga é activa a H), o que acontece nas linhas 8 a 15 da tabela.

Nestas três situações a entrada superior do OR está activa, a L, pelo que a sua saída está activa, a H (por definição de OR). Logo, a saída do circuito vem a H (e a função de saída está inactiva) nas linhas 0 a 3 e 6 a 15 da tabela.

Sobejam as linhas 4 e 5. Devemos notar que, para as linhas pares, com D_L a L, a entrada inferior do OR final está activa, o que é suficiente para que a saída dessa porta venha activa, isto é, a H. Logo, a função F vem a H na linha 4.

Resta a linha 5, em que a função F vem a L.

Retomemos então a tabela anterior, que reproduzimos na Tabela 7.9.

Tabela 7.9: Ainda a tabela de verdade física do circuito da Figura 7.24

# linha	A_H	B_L	C_H	D_L	F_L = $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$_L
0	L	L	L	L	H
1	L	L	L	H	H
2	L	L	H	L	H
3	L	L	H	H	H
4	L	H	L	L	H
5	L	H	L	H	L
6	L	H	H	L	H
7	L	H	H	H	H
8	H	L	L	L	H
9	H	L	L	H	H
10	H	L	H	L	H
11	H	L	H	H	H
12	H	H	L	L	H
13	H	H	L	H	H
14	H	H	H	L	H
15	H	H	H	H	H

Como a função F só vem activa na linha 5 (não esquecer que F é activa a L), ela deve ser constituída por apenas um mintermo. Vamos ver que mintermo é esse. À primeira vista poderia parecer que se trata do mintermo m_5, visto que ele aparece na linha 5 da tabela. Mas isso não é verdade, como podemos constatar pelo facto de F estar activa quando:

- A_H está a L ou, o que é o mesmo, $\overline{A}\_L$ está a L, ou ainda a variável $\overline{A}$ está activa;

- B_L está a H ou, o que é o mesmo, $\overline{B}$_H está a H, ou ainda a variável $\overline{B}$ está activa;
- C_H está a L ou, o que é o mesmo, $\overline{C}$_L está a L, ou ainda a variável $\overline{C}$ está activa; e, finalmente,
- D_L está a H ou, o que é o mesmo, $\overline{D}$_H está a H, ou ainda a variável $\overline{D}$ está activa;

Segue-se que a função só vem activa quando as variáveis $\overline{A}$ e $\overline{B}$ e $\overline{C}$ e $\overline{D}$ estão activas, pelo que F deve ser, na realidade, o produto lógico dessas variáveis, tal como tinhamos concluido directamente do logigrama da função.

Ou seja, $F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} = m_0$.

Notemos, então, que *o número da linha que corresponde a um determinado mintermo da função não coincide com o índice do mintermo*. Tal aconteceria se as variáveis fossem todas activas a H porque, nesse caso, elas vinham activadas com um H e inactivadas com um L, qualquer que fosse a linha da tabela que estivessemos a considerar. Como, neste exemplo, os níveis de actividade são diferentes de H, segue-se que são diferentes o número da linha em que a função vem activa e o índice do mintermo que constitui a primeira forma canónica da função.

Repare-se que podíamos ter raciocinado da forma alternativa, considerando os maxtermos da função nas linhas em que F está inactiva. Por exemplo, na linha 0 temos F inactiva. Ora, para que a função venha inactiva, é suficiente que uma das seguintes variáveis, A ou $\overline{B}$ ou C ou $\overline{D}$, venha inactiva, o que corresponde ao maxtermo M_5. E outro tanto podíamos deduzir em relação às outras linhas da tabela.

7.4 Exemplo de Utilização

Vamos considerar novamente o exemplo do alarme de carro do início do Capítulo 6 e que usa não apenas a porta do lado do condutor, mas todas as 4 portas, para fazer actuar o alarme caso alguma esteja aberta quando a chave de iginição está ligada. Pretende-se ainda activar o alarme caso o carro esteja estacionado com as luzes acesas.

Vamos representar cada porta por uma variável, $PA1$ a $PA4$, que estará a L quando a porta correspondente está aberta (os sensores que geram estas variáveis foram desenhado anteriormente). A função $PORTAS.ABERTAS$ significará, quando activa, que existe pelo menos uma porta aberta. Então teremos

$$PORTAS.ABERTAS = PA1 + PA2 + PA3 + PA4\,.$$

Do mesmo modo, LUZ será uma variável que está a H quando as luzes estão acesas, e $CHAVE$ uma variável que está a L quando a chave da ignição está ligada.

É fácil de ver que o alarme (função $ALARME$) será obtido pela expressão

$$ALARME = CHAVE \cdot PORTAS.ABERTAS + \overline{CHAVE} \cdot LUZ\,.$$

Deduzimos do enunciado do problema que as variáveis de entrada $PA1$ a $PA4$ são activas L, que a variável LUZ é activa a H e que a variável $CHAVE$ é activa a L. Nada é dito, porém, sobre o nível de actividade da função $ALARME$, de saída, nem sobre o nível de actividade da função $PORTAS.ABERTAS$, uma função intermédia.

É claro, o nível de actividade da função $ALARME$ tem a ver com o circuito a jusante, que há-de accionar o sinalizador acústico ou luminoso do alarme do carro. Pelo contrário, a decisão sobre o nível de actividade da função $PORTAS.ABERTAS$ é exclusivamente de quem desenha o logigrama do circuito. Admitiremos, arbitrariamente, que as duas funções são activas a H.

Nestas condições, podemos de imediato desenhar o logigrama da Figura 7.28 (comparar com o logigrama da Figura 6.1, na página 95, desenhado *em lógica positiva*).

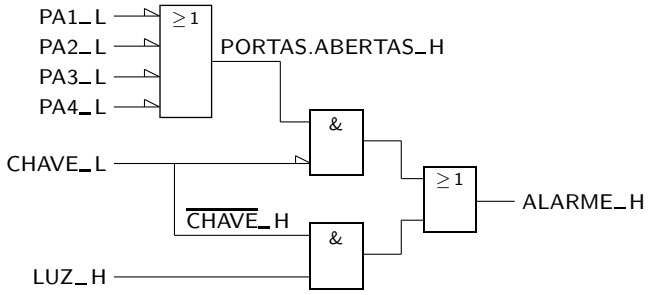


Figura 7.28: Logigrama do alarme do carro

A passagem a esquema eléctrico implica a introdução de um conversor de polaridade entra a linha $CHAVE_$L e a entrada inferior do AND superior, que é activa a H (Figura 7.29).

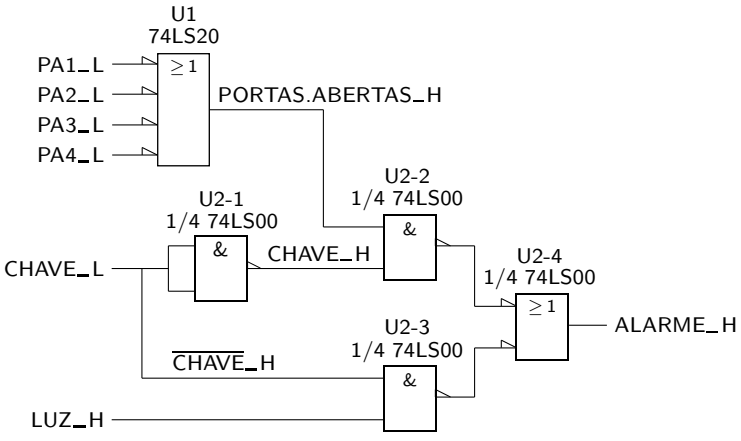


Figura 7.29: Esquema eléctrico do alarme do carro

A linha de saída do conversor será, então, designada por $CHAVE_$H, e a variável $CHAVE$ terá, nessa linha, o mesmo nível H de polaridade da entrada da porta.

Por outro lado, aproveitamos para obter o menor número de integrados no esquema eléctrico em lógica positiva (TTL), o que se consegue substituindo o conversor de polaridade por uma porta NAND de 2 entradas. No total vamos precisar de 1 integrado, U1, formado por uma porta NAND de 4 entradas (do tipo 74LS20), e de outro integrado, U2, com 4 portas NAND de 2 entradas (do tipo 74LS00).

7.5 Referências Bibliográficas

Este Capítulo não tem referências bibliográficas.

7.6 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 7.1 Em lógica de polaridade não existem portas NAND e NOR. Existem apenas portas AND, OR, NOT e conversores de polaridade (por vezes também portas XOR, embora estas possam sempre ser compostas por portas dos tipos anteriores). Em contrapartida, a inclusão de indicadores de polaridade permite obter todas as variantes de portas de que necessitamos. Diga que integrados TTL (em lógica positiva) utilizaria para implementar as seguintes portas com 2 entradas, e desenhe os símbolos IEC para cada uma delas:

(a) porta OR com as entradas e a saída activas a L;
(b) porta AND com as entradas e a saída activas a L;
(c) porta OR com as entradas activas a H e a saída activa a L;
(d) porta AND com as entradas activas a H e a saída activa a L;
(e) porta OR com as entradas activas a L e a saída activa a H;
(f) porta AND com as entradas activas a L e a saída activa a H.

(∗) 7.2 Repita o exercício anterior para as seguintes portas com 3 entradas (possivelmente necessitará de circuitos mais complexos do que uma simples porta de substituição):

(a) porta OR com 2 entradas activas a H e uma a L, e a saída activa a H;
(b) porta OR com 2 entradas activas a H e uma a L, e a saída activa a L;
(c) porta AND com 2 entradas activas a H e uma a L, e a saída activa a H;
(d) porta AND com 2 entradas activas a H e uma a L, e a saída activa a L;
(e) porta XOR com as 3 entradas activas a H e a saída activa a H;
(f) porta XOR com as 3 entradas activas a L e a saída activa a H.

Tabela de verdade genérica das portas XOR com 2 entradas

(∗) 7.3 A descrição da função OU-exclusivo em lógica de polaridade traduz-se pela tabela de verdade genérica da Tabela 7.12 para uma porta XOR com 2 entradas.

Tabela 7.12: Tabela de verdade genérica para uma porta XOR com 2 entradas

A	B	A ⊕ B
I	I	I
I	A	A
A	I	A
A	A	I

a) Desenhe o símbolo IEC desta porta que tenha as entradas activas a L e a saída activa a H.

b) Repita a alínea anterior para o caso de a entrada A ser activa a L, B ser activa a H e a saída ser activa a L.

c) Mostre que as portas das alíneas a) e b) são fisicamente apenas uma, e ainda que são idênticas a um XOR com entradas activas a H e a saída activa a H.

(∗) 7.4 Pretende-se implementar a função booleana simples $OUT = IN1 + IN2$ admitindo que:

a) os níveis de actividade de $IN1$, de $IN2$ e de OUT são todos H;

b) $IN1$ é activa a L e $IN2$ e OUT são activas a H;

c) $IN1$ e $IN2$ são activas a L e OUT é activa a H;

d) $IN1$ e $IN2$ são activas a H e OUT é activa a L.

Estabelecer, para todos os casos, a tabela de verdade de OUT.

(∗) 7.5 Que função ou funções booleanas simples são geradas pelas portas lógicas da Figura 7.30?

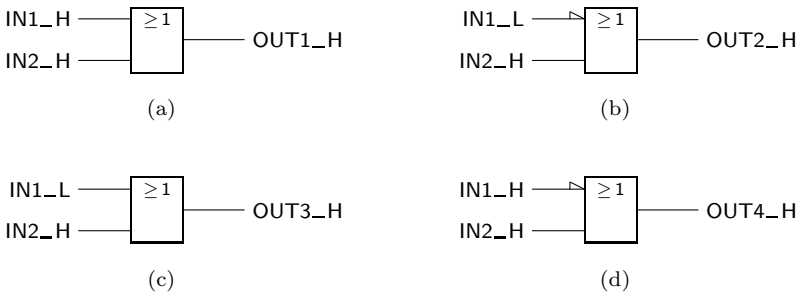


Figura 7.30: Logigramas do Exercício 7.5

(∗) 7.6 Dada a porta da Figura 7.31, determinar a expressão lógica da função de saída e construir a tabela de verdade física correspondente.

(∗) 7.7 Obter, em lógica HCT, os esquemas eléctricos correspondentes aos logigramas dos Exercícios 7.4 e 7.6. Para simplificar os esquemas, não incluir os pinos dos circuitos integrados.

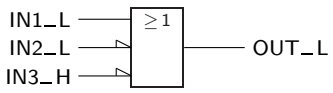


Figura 7.31: Logigrama da função do Exercício 7.6

(∗) 7.8 Pretende-se estabelecer o logigrama e a tabela de verdade física de uma função booleana simples, $ACTUATE\,(IN1, IN2, SEL, DETECT)$, que deve vir activada quando forem satisfeitas uma ou outra mas não ambas as condições que se descrevem a seguir:

1. as entradas $IN1$ ou $IN2$ ou ambas estão activadas;
2. as entradas SEL e $DETECT$ estão activadas.

Admitir:

a) que $IN1$, $IN2$, SEL, $DETECT$ e $ACTUATE$ são todas activas a H;

b) que $IN1$ e $IN2$ são activas a H, e que SEL, $DETECT$ e $ACTUATE$ são activas a L.

7.9 São fornecidas as entradas $A_$L, $B_$H, $C_$L e $D_$H. Usando integrados TTL, desenhe os esquemas eléctricos das seguintes funções,

a) $Z = (\overline{A + B + C})\,(\overline{B} + D)$;

b) $Z = (A + \overline{B} + C)\,(C \odot D)$;

c) $Z = A + B + C + D$,

admitindo que $Z_$L.

(∗) 7.10 Pretende-se implementar um circuito lógico que acende uma luz sob comando dos terminais $IN_$L e $TOL_$L. A função que vai permitir acender a luz deverá ser activa a L e será comandada pelos seguintes sinais:

(1) ligar a luz ($TOL_$H);

(2) inibir ($IN_$L);

(3) emergência ($EMERG_$L); e

(4) a ocasião não é adequada ($TNR_$H).

A luz deverá acender-se desde que a ocasião seja adequada, o comando de luz não seja inibido pela variável IN, e seja dada uma ordem para ligar a luz. Se, contudo, se verificar uma emergência, a luz deverá acender-se, independentemente dos outros comandos. Desenhe um logigrama em lógica de polaridade para o circuito, e estabeleça o correspondente esquema eléctrico em lógica positiva.

7.11 Considere o circuito da Figura 7.32.

Sabendo que as variáveis $INTERRUPTOR.ON$ e $ACCAO1$ estão activadas, e que a linha $DESLIGAR_$L está ao nível H,

a) em que nível de tensão está a linha $CONDICAO_$H? Porquê?

b) a função $RESULTADO$ está ou não activa? Porquê?

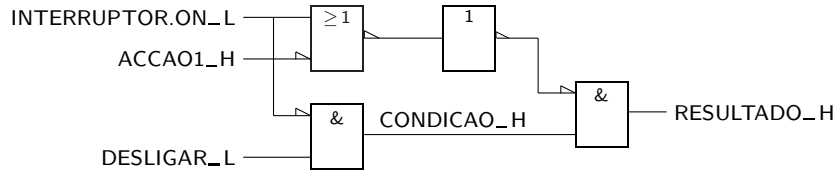


Figura 7.32: Logigrama da função do Exercício 7.11

7.12 Projecte um circuito com 3 entradas onde são aplicadas as variáveis booleanas simples $ACCAO1$, $ACCAO2$ e $ACCAO3$, e uma saída onde vem gerada a função booleana simples $RESULTADO$. $RESULTADO$ deverá estar activa quando $ACCAO1$ estiver activa e uma e só uma das outras duas variáveis estiver activa, ou quando apenas $ACCAO3$ estiver activa. $ACCAO2$ é activa a H, enquanto que as outras duas variáveis e a função são activas a L.

7.13 São dados $A_$L, $B_$H, $C_$L e $D_$H. Utilizando os circuitos integrados existentes na lógica HCT (positiva) desenhe os esquemas eléctricos das seguintes funções, estabelecendo o paralelo entre esses esquemas e os logigramas que obtém em lógica de polaridade directamente das expressões das funções Z, (com $Z_$H).

a) $Z = (A + D)\,(\overline{B} + C)$;

b) $Z = A\,\overline{B}\,\overline{C} + B\,D$;

c) repita as alíneas a) e b) para o caso de ser $Z_$L.

7.14 Considere o circuito com o logigrama da Figura 7.33.

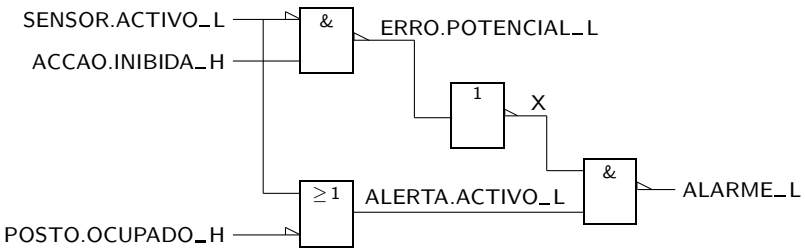


Figura 7.33: Logigrama da função do Exercício 7.14

Supondo que as variáveis $SENSOR.ACTIVO$ e $ACCAO.INIBIDA$ estão activas, e que a linha $POSTO.OCUPADO_$H está no nível L, indique, justificando:

a) se a variável $ERRO.POTENCIAL$ está activa ou inactiva;

b) em que nível está a linha $ALARME_$L;

c) estabeleça uma designação razoável (com conteúdo semântico) para a linha marcada com um X.

7.15 Projecte o circuito combinatório de controlo de um elevador entre dois pisos. O circuito possui as seguintes entradas:

(1) $ELEV.NO.PISO.INFERIOR_$L;

(2) $ELEV.NO.PISO.SUPERIOR_$L;

(3) $ELEV.A.SUBIR_$H;

(4) $ELEV.A.DESCER_$H;

(5) $PEDIDO.DE.SUBIDA_$L;

(6) $PEDIDO.DE.DESCIDA_$L; e

(7) $PORTA.ABERTA_$H,

e as seguintes saídas:

(a) $MARCHA.ASCENDENTE_$H;

(b) $MARCHA.DESCENDENTE_$H;

(c) $PORTA.SUP.BLOQUEADA_$L; e

(d) $PORTA.INF.BLOQUEADA_$L.

7.16 É dado o logigrama da Figura 7.34.

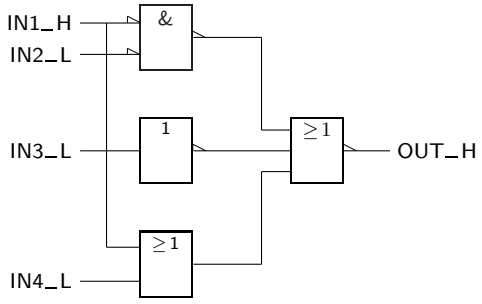


Figura 7.34: Logigrama utilizado no Exercício 7.16

Determine a expressão booleana da função $OUT(IN1, IN2, IN3, IN4)$.

(∗) 7.17 Considere os logigramas da Figura 7.35, todos semelhantes. Estabeleça as tabelas de verdade físicas para as três funções, e deduza directamente as suas expressões lógicas a partir das tabelas. Em seguida, confirme as expressões analisando os logigramas correspondentes.

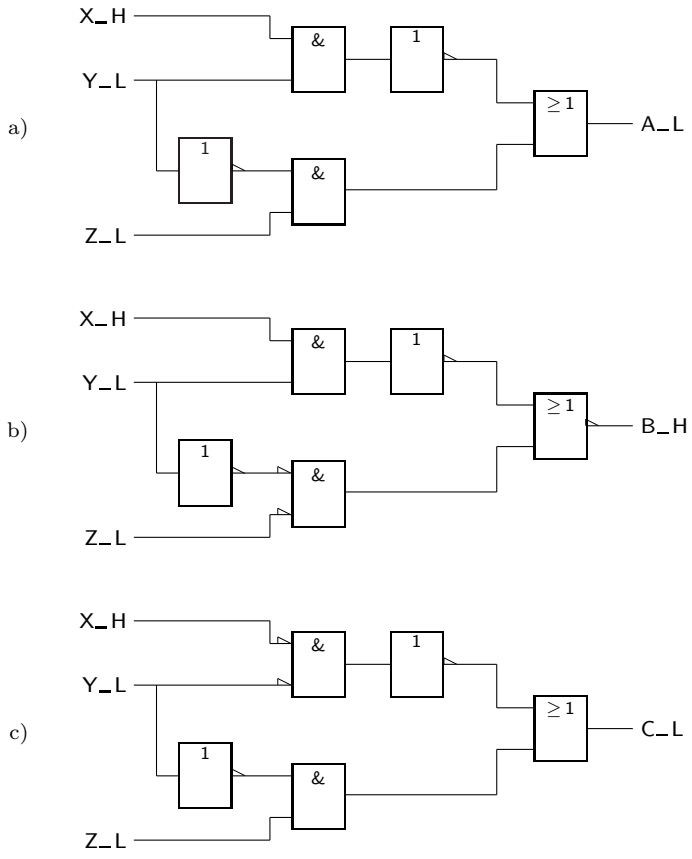


Figura 7.35: Logigramas utilizados no Exercício 7.17

Capítulo 8

Análise e Síntese de Circuitos Combinatórios

8.1 Enquadramento

As matérias anteriores só têm interesse na medida em que permitem gerar recursos e metodologias para resolver problemas concretos reais de automatização, de cálculo, de monitorização ou outros em que os circuitos digitais possam ser utilizados.

Os circuitos digitais são de dois tipos: combinatórios e sequenciais. Ambos possuem um conjunto de entradas por onde os dados são introduzidos no circuito, e um outro de saidas onde são reflectidos os resultados do processamento realizado.

Circuito combinatório

Circuito sequencial

Os **circuitos combinatórios** são aqueles que permitem prever os valores lógicos (ou níveis de tensão) das saídas num determinado instante em função da combinação de valores lógicos (níveis) nas entradas do circuito no mesmo instante. São os que nos vão ocupar de imediato. Os **circuitos sequenciais** são circuitos em que os valores (níveis) das saídas num determinado instante não depende apenas dos valores (níveis) das entradas nesse instante, mas também da sequência desses valores (níveis) ao longo do tempo.

Vamos analisar, para já, os circuitos combinatórios.

Este tipo de circuitos pode ser descrito por um modelo como o que se apresenta na Figura 8.1, em que existem n entradas, p saídas e um conjunto de p funções booleanas simples de, no limite, n variáveis booleanas simples correspondentes às entradas.

Do ponto de vista fisico, o circuito é constituído por um conjunto de portas e pelas suas interligações, que implementam as p funções.

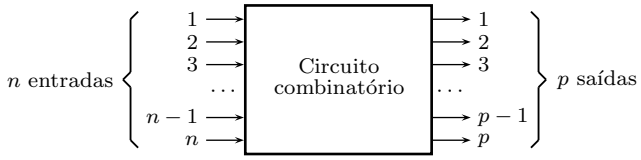


Figura 8.1: Modelo de circuito combinatório

8.2 Análise de Circuitos Combinatórios

Basicamente, a análise de um circuito combinatório é feita a partir do seu logigrama ou esquema eléctrico, e pode ser executada de duas formas: (i) ou se levantam as equaçõess das saídas do circuito; (ii) ou se escreve a tabela de verdade do circuito.

Como é evidente, as duas formas não são conceptualmente diferentes, mas utilizam ferramentas diferentes.

No caso de derivação da equação do circuito, o que há a fazer é:

— criar nomes para todas as funções nas saídas de portas que dependem apenas das entradas, e determinar as suas expressões em função das variáveis de entrada;

— repetir o processo para as portas que dependem das portas já tratadas e das variáveis de entradas;

— aplicar sucessivamente o passo anterior até se terem determinado todas as funções de saída.

Exemplifique-se com o circuito da Figura 8.2.

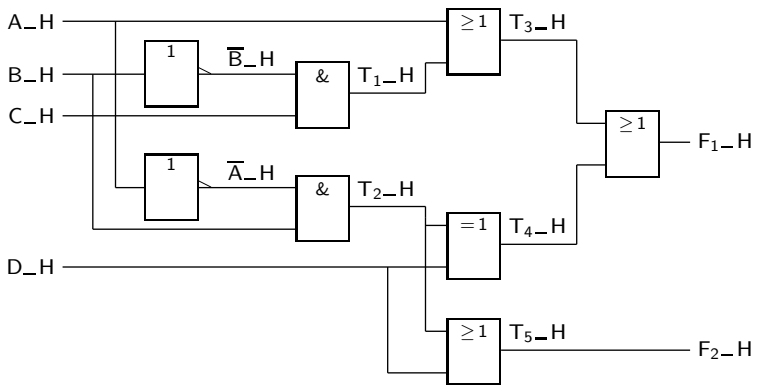


Figura 8.2: Logigrama de um circuito combinatório onde se identificam as funções T_1 a T_5 nas saídas das portas lógicas que interessa analisar para a obtenção das equações de saída do circuito

Após identificação das saídas dos dois ANDs obtemos:

$$T_1 = \overline{B}\, C$$
$$T_2 = \overline{A}\, B\,.$$

Seguidamente procede-se à identificação das saídas das portas do segundo nível, obtendo-se:

$$T_3 = A + T_1 = A + \overline{B}\, C$$
$$T_4 = T_2 \oplus D = (\overline{A}\, B) \oplus D = \overline{A}\, B\, \overline{D} + A\, D + \overline{B}\, D$$
$$T_5 = T_2 + D = \overline{A}\, B + D\,.$$

As funções de saída são, portanto,

$$\begin{aligned} F_1 &= T_3 + T_4 = A + \overline{B}\, C + \overline{A}\, B\, \overline{D} + A\, D + \overline{B}\, D \\ &= A + \overline{B}\, C + B\, \overline{D} + \overline{B}\, D \\ F_2 &= T_5 = \overline{A}\, B + D \end{aligned}$$

Para se obter a tabela de verdade física do circuito, podemos utilizar o procedimento que foi usado na Subsecção 7.3.2, a propósito da construção da Tabela 7.8: procedendo linha a linha (o que pode tornar morosa a sua construção), ou procedendo globalmente em relação à tabela toda.

Contudo, quando o logigrama é relativamente complexo, é útil construir a tabela de verdade física aos poucos, a partir das tabelas de funções intermédias (de maneira análoga à que foi usada para determinar as expressões booleanas das funções nas saídas do logigrama anterior).

Para exemplificar esta metodologia, consideremos o logigrama da Figura 8.3.

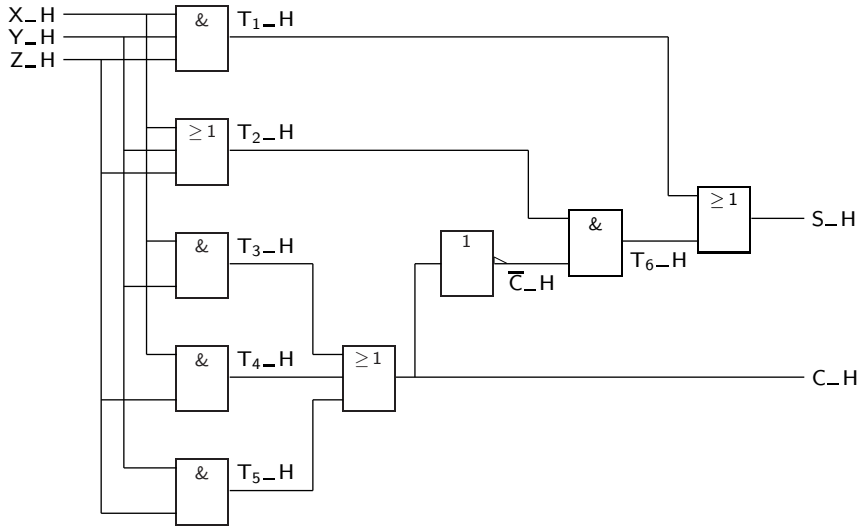


Figura 8.3: Logigrama de outro circuito combinatório onde se identificam as saídas T_1 a T_6 das portas lógicas que interessa analisar para a obtenção da tabela de verdade física do circuito

Tabela 8.1: Construção da tabela de verdade física do circuito com o logigrama na Figura 8.3, com a construção de tabelas para as funções intermédias

X_H	Y_H	Z_H	T_3_H	T_4_H	T_5_H	C_H	$\overline{C}$_H	T_1_H	T_2_H	T_6_H	S_H
L	L	L	L	L	L	L	H	L	L	L	L
L	L	H	L	L	L	L	H	L	H	H	H
L	H	L	L	L	L	L	H	L	H	H	H
L	H	H	L	L	H	H	L	L	H	L	L
H	L	L	L	L	L	L	H	L	H	H	H
H	L	H	L	H	L	H	L	L	H	L	L
H	H	L	H	L	L	H	L	L	H	L	L
H	H	H	H	H	H	H	L	H	H	L	H

A tabela de verdade física do circuito pode obter-se imediatamente (Tabela 8.1).

Por exemplo, a função T_1 vem activa (a H) apenas quando as variáveis X, Y e Z estiverem activas (todas a H), o que apenas ocorre na última linha da tabela.

Ou, tomando ainda outro exemplo, para que a função final C venha activa (a H) basta que uma das funções intermédias, T_3, T_4 ou T_5 esteja activa (a H). Por seu turno,

— T_3 vem activa quando X e Y estão activas (linhas 6 e 7);

— T_4 vem activa quando X e Z estão activas (linhas 5 e 7); e

— T_5 vem activa quando Y e Z estão activas (linhas 3 e 7),

pelo que C fica activa nas linhas 3, 5, 6 e 7.

8.3 Projecto de Circuitos Combinatórios

Para realizar o projecto de um circuito combinatório é possível atacar o problema na sua complexidade total, mas o mais comum é dividir o problema em problemas mais simples, concebendo o circuito como um conjunto de módulos de menor complexidade, devidamente interligados.

Seguidamente analisa-se cada um dos módulos e, se ele jé é suficientemente simples para ser projectado com as ferramentas que temos, projecta-se. Se isso não acontece, continua a divisão em módulos mais simples.

Um dos conceitos importantes é, portanto, o de **hierarquia de projecto**, partindo da descrição mais abstracta do circuito através de um modelo semelhante ao do da Figura 8.1, para módulos de menor complexidade, e assim por diante até chegar às portas lógicas.

Exemplifiquemos com um circuito detector de paridade ímpar com 9 entradas e 1 saída, supostas todas activas a H, com o diagrama de blocos da Figura 8.4.

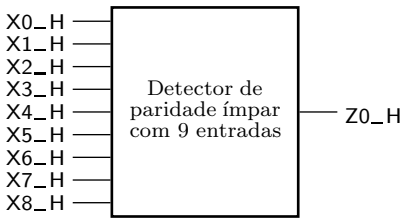


Figura 8.4: Diagrama de blocos de um detector de paridade ímpar com 9 entradas e 1 saída, todas activas a H

Este circuito recebe 9 bits nas entradas e produz um resultado igual a 1 se e só se o número de bits a 1 nas entradas for ímpar. A sua utilidade é a de detectar erros de transmissão de dados com 9 bits em paralelo, garantindo que os dados transmitidos possuem paridade par (a saída vem activa se houver algum bit trocado).

Podemos calcular a função $Z0$ de 9 variáveis e implementar o circuito correspondente, mas tal seria uma tarefa de enorme complexidade. Assim, é preferível dividir o problema em problemas menores, por exemplo em 3 detectores de paridade ímpar, apenas com 3 entradas.

Obtém-se, assim, o diagrama de blocos da Figura 8.5.

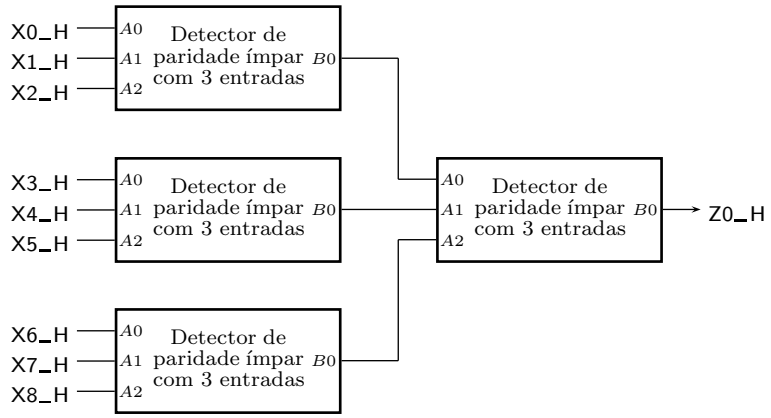


Figura 8.5: Diagrama de blocos de um detector de paridade ímpar com 9 entradas, construído com 3 detectores de paridade ímpar com 3 entradas cada um

Seguidamente pensa-se na constituição de cada um dos blocos (Figura 8.6).

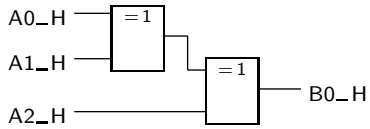


Figura 8.6: Logigrama de um detector de paridade ímpar com 3 entradas activas a H e saída activa a H

Por fim, podem usar-se portas XOR ou, em vez delas, portas de outro tipo. Por exemplo, o XOR pode ser implementado pelo circuito com NANDs (em lógica positiva) da Figura 8.7.

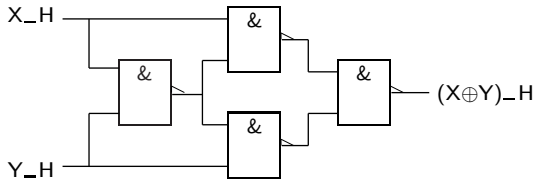


Figura 8.7: Logigrama de implementação de um XOR com 2 entradas, feito à custa de NANDs (em lógica positiva)

A hierarquia de projecto que foi ilustrada é feita com base num sistema que utiliza vários módulos iguais. Mas nem sempre é possível proceder desse modo. Por exemplo, se se pretender obter um circuito combinatório que adiciona ou multiplica dois números de 4 bits, podemos ter como primeira desagregação do modelo inicial algo como se ilustra na Figura 8.8.

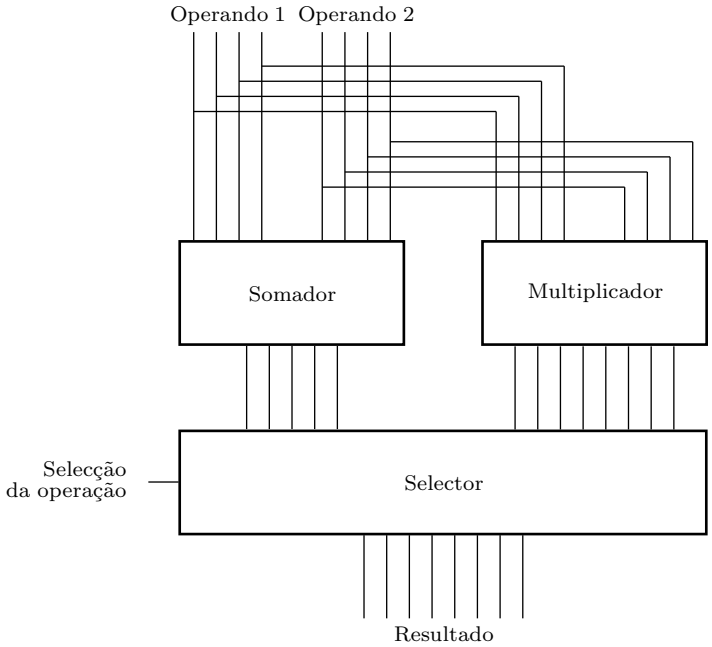


Figura 8.8: Diagrama de blocos de um adicionador/multiplicador de dois números de 4 bits

As vantagens de utilização deste tipo de procedimento com hierarquização são:

— melhor concepção estrutural e visualização da estrutura do circuito;

— em cada nível, há abstracção em relação aos pormenores dos níveis inferiores;

— replicação de módulos com reutilização.

Idealmente, o projecto deveria ser feito segundo uma aproximação **"top-down"**. No entanto, muitas vezes há condicionalismos em relação aos módulos que podemos usar em níveis inferiores, o que leva a que muitas vezes se combine ou até se substitua a estratégia "top-down" por uma estratégia **"bottom-up"**.

Projecto "top-down"

Projecto "bottom-up"

8.4 Síntese de Circuitos Combinatórios

Admitindo que as regras de projecto já apresentadas foram utilizadas, os circuitos combinatórios que se apresentam são, naturalmente, de pequena dimensão. Nessas circunstancias o que há a fazer é:

— definir o número de entradas e saídas e atribuir-lhes nomes;

— obter a tabela de verdade para cada saída ou, em alternativa, obter as suas expressões lógicas;

— obter as expressões lógicas simplificadas das funções de saída;

— desenhar o logigrama do circuito, fazendo eventualmente adaptações para minimizar o número de circuitos integrados a utilizar;

— verificar a correcção do projecto; e

— desenhar o esquema eléctrico do circuito.

O primeiro ponto deve ser evidente nesta fase do projecto.

Ao obter a tabela de verdade de uma função é frequente descobrir que aquilo que era evidente não está completamente esclarecido e necessita de mais profunda análise sobre o comportamento pretendido do circuito em circunstâncias marginais.

A simplificação pode ser feita de qualquer forma, desde a simplificação algébrica à simplificação utilizando pacotes de "software" adequados, passando pela minimização usando quadros de Karnaugh.

O desenho do logigrama e do esquema eléctrico pode ser feito usando um "software" adequado ("schematic capture") ou, simplesmente, à mão. Convém usar a experiência acumulada para minimizar o número de integrados a utilizar.

A verificação de correcção pode ser feita manualmente ou utilizando pacotes de "software" destinados à simulação de circuitos. De qualquer forma, é sempre necessário verificar no fim, com um circuito real, se tudo funciona como previsto. Há pormenores que escapam a algum do "software" de simulação comumente utilizado.

Como exemplo, consideremos o projecto de um circuito de conversão entre um dígito BCD e o dígito correspondente no **código Excesso de 3** (ou **código D+3**). O código Excesso-3 não é já muito utilizado mas, no passado, facilitou a realização de certos circuitos aritméticos. Trata-se de um código decimal-binário e, como tal, possui 10 palavras, numeradas de 0 a 9. A sua construção resulta da do código BCD adicionando 3 unidades a cada palavra homóloga desse código (daí a designação Excesso-3). Vejamos na Tabela 8.2 o código $D + 3$.

Código Excesso de 3 ($D + 3$)

Tabela 8.2: Código Excesso-3 (ou $D + 3$)

Dígito	Código
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Para implementar este circuito precisamos de uma lógica com 4 entradas (por onde se apresentam as palavras do código BCD) e 4 saídas (onde são devolvidas as palavras no código Excesso-3). As entradas serão designadas, por exemplo, por A, B, C e D, sendo A a mais significativa, e as saídas por W, X, Y e Z, sendo W a mais significativa (diagrama de blocos na Figura 8.9).

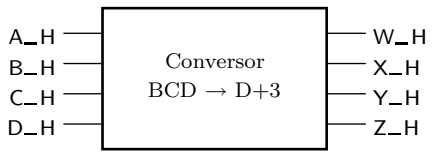


Figura 8.9: Diagrama de blocos de um conversor do código BCD para o código $D + 3$

A conversão entre os dois códigos será, então, a que se ilustra na tabela de verdade lógica da Tabela 8.3.

Para obter agora as funções de saída, usaremos quadros de Karnaugh, tendo em consideração que as posições dos mapas que não correspondem a dígitos BCD são indiferenças (Figura 8.10).

Dos quadros de Karnaugh é possível ler:

$$
\begin{aligned}
W &= A + B\,C + B\,D \\
X &= \overline{B}\,C + \overline{B}\,D + B\,\overline{C}\,\overline{D} \\
Y &= C\,D + \overline{C}\,\overline{D} \\
Z &= \overline{D}
\end{aligned}
$$

A análise destas funções mostra que é possível construir o circuito na tecnologia "Low Power Schottky" TTL usando:

Tabela 8.3: Tabela de verdade lógica do conversor do código BCD para o código Excesso-3

Dígito	Código BCD				Código D+3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

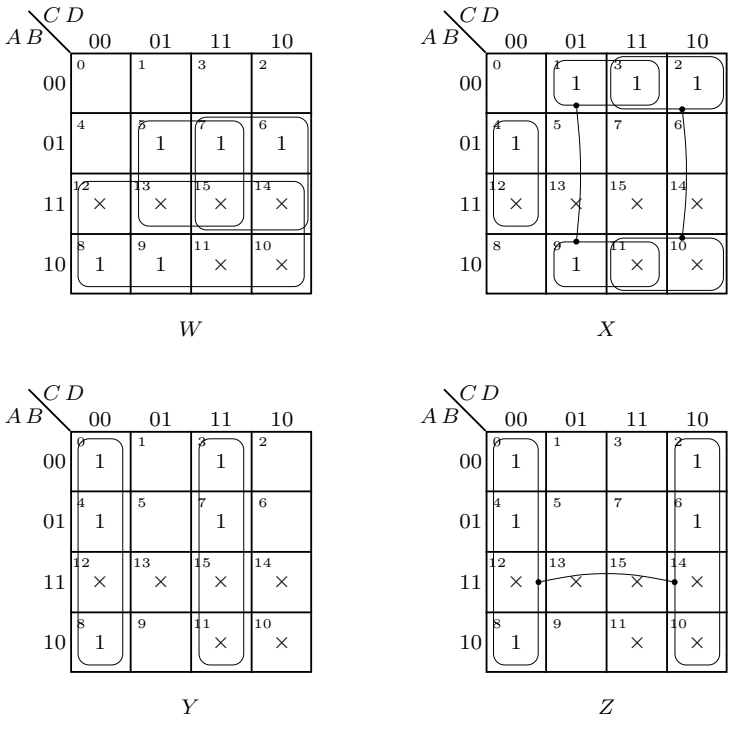


Figura 8.10: Minimização das funções de saída do conversor BCD $\rightarrow D + 3$

3 NOTs	1 IC de negações (74LS04) usado a 50%
6 ANDs de 2 entradas	2 ICs de ANDs de 2 entradas (74LS08), com o segundo IC usado a 50%
1 AND de 3 entradas	1 IC de ANDs de 3 entradas (74LS11) usado a 33%
2 ORs de 3 entradas	que não existem e teriam de ser implementados por ORs de 2 entradas
1 OR de 2 entradas	2 ICs em conjunto com os anteriores (74LS32), com o segundo usado a 25%.

Manipulando um pouco a expressão, e à custa de construir um circuito mais lento (e de alguma experiência), é possível obter o seguinte circuito, que utiliza apenas 1 IC de ANDs de 2 entradas (74LS08) a 25%, 1 IC de ORs de 2 entradas (74LS32) a 50%, e 1 IC de XORs (74LS86) a 75%, como se mostra no esquema eléctrico simplificado da Figura 8.11 (sem a indicação dos pinos dos integrados).

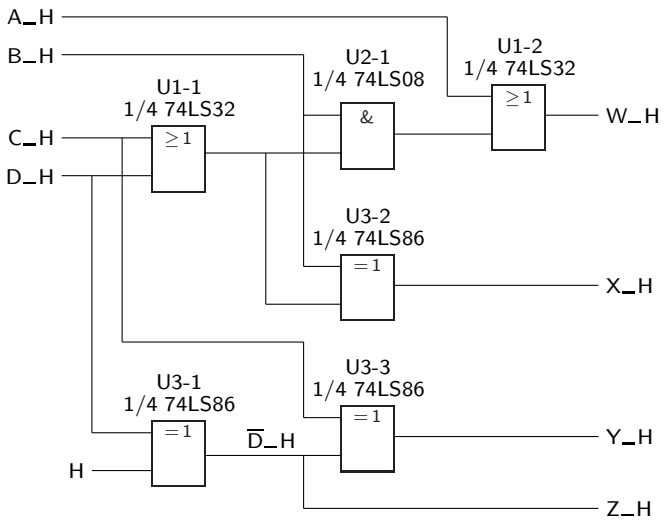


Figura 8.11: Esquema eléctrico do conversor BCD $\rightarrow D+3$

Para se obter este esquema eléctrico repare-se que:

1. combinámos as expressões de W e de X para aproveitar um OR em comum e usar um XOR:

$$W = A + B\,(C + D)$$
$$X = \overline{B}\,(C + D) + B\,\overline{C}\,\overline{D} = B \oplus (C + D)\,;$$

2. construimos a função Y com outro XOR (usámos $\overline{D}$, uma vez que já era necessário para formar Z):

$$Y = C\,D + \overline{C}\,\overline{D} = C \odot D = \overline{C \oplus D} = C \oplus \overline{D}\,;$$

3. usámos um XOR para fazer a negação:

$$Z = \overline{D} = D \oplus 1\,.$$

8.5 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 3.1 a 3.4.

8.6 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

8.1 Porque é que o detector de paridade ímpar com 9 entradas da Figura 8.4 pode ser dividido em 3 detectores de paridade ímpar com 3 entradas, como mostra o diagrama de blocos da Figura 8.5?

8.2 Justifique que o circuito da Figura 8.7 implementa de facto um XOR com 2 entradas.

8.3 Dado um número com 8 bits, $NUM7\_H$ a $NUM0\_H$, desenhar o diagrama de blocos de um circuito combinatório que gera o seu complemento para 2.

8.4 Desenhar o logigrama de um circuito combinatório com três entradas, $X2\_H$ a $X0\_H$, e três saídas, $Y2\_H$ a $Y0\_H$ (que representam números decimais entre 0 e 7), por forma a que o circuito calcule o valor de $\mathbf{Y} = =(3\mathbf{X}) \mod 8$.

Capítulo 9

Codificadores e Descodificadores

No projecto de circuitos digitais, como vimos, é habitual dividir os projectos, organizando-os hierarquicamente e realizando-os a partir de módulos de complexidade inferior ao problema que se quer resolver.

Neste capítulo e nos seguintes estudaremos alguns módulos típicos que são re-utilizados com frequência no projecto de sistemas digitais e que correspondem também a circuitos integrados existentes no mercado.

9.1 Descodificadores

Definição de descodificador

Um **descodificador** é um circuito que permite obter, a partir de um conjunto de bits que constituem uma palavra de um determinado código, a identificação dessa palavra. Para isso, o descodificador tem tantas saídas quanto o número de palavras de código, e activa, em cada momento, a saída correspondente à palavra de código presente nas entradas.

Descodificador binário

Por exemplo um **descodificador binário** de n bits possuiu n entradas onde são aplicadas palavras do CBN (Código Binário Natural) de comprimento n, desde $00\ldots0$ até $11\ldots1$. Este descodificador tem ainda 2^n saídas, numeradas de 0 a 2^n-1, sendo que uma e apenas uma pode vir activa de cada vez.

Em termos de funcionamento, um descodificador binário de 3 bits pode ser descrito pela tabela de verdade lógica da Tabela 9.1, em que Si é a saída correspondente à palavra i do código.

O símbolo deste circuito, de acordo com a norma IEC 60617-12, é o que se representa na Figura 9.1(a), admitindo que todas as entradas e todas as saídas são activas a H. Na Figura 9.1(b) representa-se o mesmo descodificador e ainda 3 variáveis booleanas simples aplicadas às entradas e 8 funções booleanas simples geradas nas saídas, com designações arbitrárias e sem conteúdo semântico significativo.

Qualificador geral BIN/1-OF-8 ou BIN/OCT

De notar o **qualificador geral** BIN/1-OF-8 atribuído ao símbolo (em alternativa podíamos usar o qualificador geral BIN/OCT), significando que aplicamos às

Tabela 9.1: Tabela de verdade lógica de um descodificador binário de 3 bits

Palavra	S0	S1	S2	S3	S4	S5	S6	S7
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

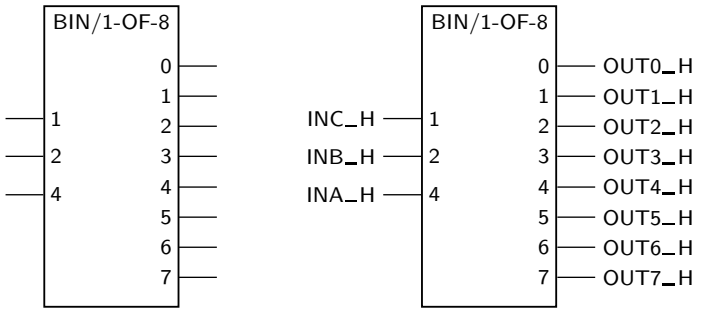


Figura 9.1: (a) Símbolo de um descodificador binário de 3 bits com todas as entradas e saídas activas a H, de acordo com a norma IEC 60617-12; (b) o mesmo descodificador com 3 variáveis booleanas simples aplicadas às entradas (com designações arbitrárias, sem conteúdo semântico), e possíveis designações para as funções booleanas simples de saída que são geradas pelo circuito

entradas palavras do CBN e que obtemos nas saídas palavras de um código especial, da família dos códigos 1–em–n (no caso, do código 1–em–8), em que em cada palavra com 8 bits apenas existe um 1 (no caso do descodificador, apenas uma saída está activa de cada vez).

Pesos das entradas

De notar ainda os pesos relativos das entradas do símbolo da Figura 9.1:

— a entrada 4 tem peso 4;

— a entrada 2 tem peso 2; e

— a entrada 1 tem peso 1,

Saída activa

e o modo como uma e apenas uma saída vem activa de cada vez:

— a função $OUT0$ deve vir activa, a H, se e só se a palavra do código de entrada for (INA, INB, INC)=(L,L,L), ou 0 em decimal;

— a função $OUT1$ deve vir activa, a H, se e só se a palavra do código de entrada for (INA, INB, INC)=(L,L,H), ou 1 em decimal;

— a função $OUT2$ deve vir activa, a H, se e só se a palavra do código de entrada for (INA, INB, INC)=(L,H,L), ou 2 em decimal; etc,

— a função $OUT7$ deve vir activa, a H, se e só se a palavra do código de entrada for (INA, INB, INC)=(H,H,H), ou 7 em decimal.

Ou seja, para determinar a saída que está activa em cada instante, *somamos os pesos das entradas que estão activas* nesse instante. Por exemplo, a saída $OUT3$ vem activa quando (INA, INB, INC)=(L,H,H) porque a entrada INB está activa e possui peso 2 e a entrada INC também está activa e possui peso 1 (a soma dos pesos é igual a 3).

Podemos, então, deduzir a tabela de verdade física para este descodificador, como se ilustra na Tabela 9.2. De notar como uma e só uma saída vem activa de cada vez, como pretendemos.

Tabela 9.2: Tabela de verdade física do descodificador binário da Figura 9.1

IN			OUT							
A_H	B_H	C_H	7_H	6_H	5_H	4_H	3_H	2_H	1_H	0_H
L	L	L	L	L	L	L	L	L	L	H
L	L	H	L	L	L	L	L	L	H	L
L	H	L	L	L	L	L	L	H	L	L
L	H	H	L	L	L	L	H	L	L	L
H	L	L	L	L	L	H	L	L	L	L
H	L	H	L	L	H	L	L	L	L	L
H	H	L	L	H	L	L	L	L	L	L
H	H	H	H	L	L	L	L	L	L	L

Da tabela de verdade física do circuito é fácil obter a sua estrutura interna (Figura 9.2). De notar que o símbolo utilizado na parte direita da figura é um **símbolo composto**.

Símbolo composto

Para além deste tipo de descodificadores (binários de n bits), existem ainda descodificadores específicos para certos códigos, como por exemplo o código BCD. Esse tipo, em particular, tem 4 entradas e 10 saídas, como mostra o símbolo IEC da Figura 9.3, e designa-se por **descodificador BCD**.

Descodificador BCD

De notar o **qualificador geral** BCD/DEC atribuído ao símbolo (em alternativa podíamos usar o qualificador geral BCD/1-OF-10), significando que aplicamos às entradas palavras do código BCD e que obtemos nas saídas palavras do código 1–em–10, em que em cada palavra com 10 bits apenas existe um 1 (no caso do descodificador, apenas uma saída está activa de cada vez).

Qualificador geral BCD/1-OF-10 ou BCD/DEC

É muito frequente que os descodificadores tenham as suas saídas activas a L. Nessas circunstâncias, as portas de saída são NANDs em vez de ANDs (em lógica positiva), e a saída activa passa a ser a que está, de cada vez, a L. O símbolo IEC de um descodificador binário de 3 bits, semelhante ao da Figura 9.1 mas com saídas activas a L, será o que se representa na Figura 9.4.

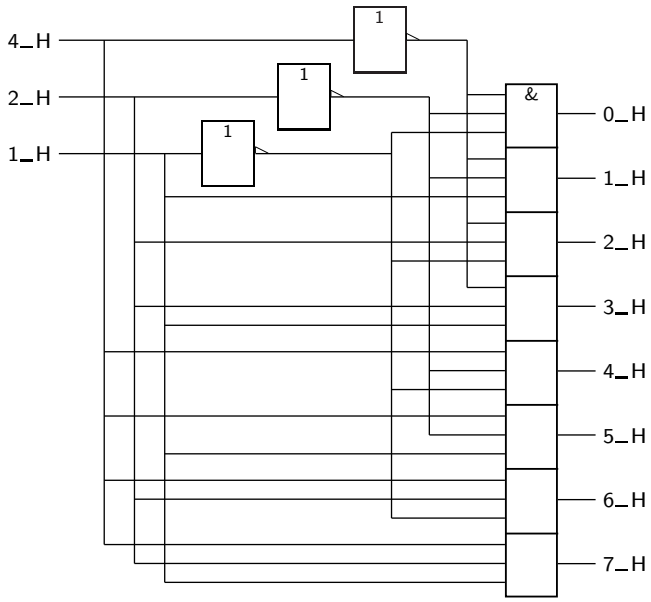


Figura 9.2: Estrutura interna do descodificador binário da Figura 9.1

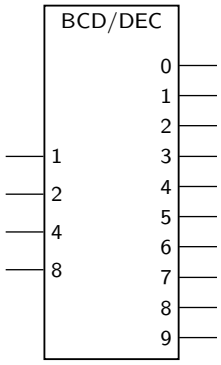


Figura 9.3: Símbolo de um descodificador BCD com entradas e saídas activas a H, de acordo com a norma IEC 60617-12

9.1.1 Expansão de descodificadores

Muitas vezes os descodificadores que existem no mercado não têm o número de entradas (e de saídas) que são necessárias para descodificar um determinado código. É, por isso, necessário expandi-los.

Entrada de Enable
Dependência de Enable (EN)

Para isso, muitos têm uma **entrada de Enable**, geralmente designada por EN, que configura uma **dependência de Enable (EN)**. Esta entrada, quando activada, permite que o descodificador funcione normalmente. Quer isso dizer que, nessas condições, existe sempre uma, mas apenas uma, saída seleccionada.

Se, pelo contrário, EN estiver desactivada, as saídas vêm desactivadas em bloco. Assim, se as saídas forem activas a H, elas virão todas a L. E se forem activas a L, elas virão todas a H. No caso de as saídas serem do tipo "tri-state", a

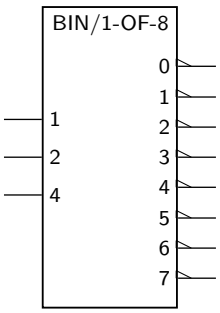


Figura 9.4: Símbolo de um descodificador binário de 3 bits, com entradas activas a H e saídas activas a L, de acordo com a norma IEC 60617-12

desactivação de EN significa que todas as saídas ficam em alta impedância.

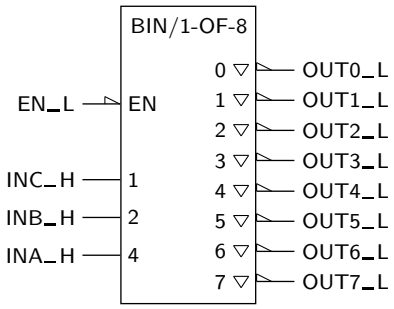


Figura 9.5: Símbolo IEC de um descodificador binário de 3 bits com entradas de dados activas a H e entrada de Enable activa a L, e com saídas "tri-state" activas a L. Ao símbolo adicionaram-se designações com conteúdo semântico para as entradas e saídas

Na Figura 9.5 apresenta-se o símbolo IEC de um descodificador binário de 3 bits com Enable e saídas "tri-state" activas a L, e na Tabela 9.3 mostra-se a correspondente tabela de verdade física.

Usando descodificadores com Enable podemos construir descodificadores de n entradas com descodificadores "menores". Exemplifica-se na Figura 9.6 com um descodificador binário de 4 bits que utiliza descodificadores binários de 2 bits. Naturalmente, os níveis de actividade nas entradas e nas saídas dos descodificadores de 2 bits foram escolhidos arbitrariamente. Contudo, as entradas de Enable *devem ter o mesmo nível de actividade* das saídas, caso contrário teríamos de colocar portas NOT entre as saídas do descodificador da esquerda e as entradas de Enable dos descodificadores da direita.

9.1.2 Utilização de descodificadores na implementação de funções lógicas

Repare-se que um descodificador implementa *todos os mintermos* das variáveis de entrada. Assim sendo, deve ser possível usar o descodificador para implemen- *Gerador de mintermos*

Tabela 9.3: Tabela de verdade física do descodificador binário da Figura 9.5

IN				OUT							
A_H	**B_H**	**C_H**	**EN_L**	**7_L**	**6_L**	**5_L**	**4_L**	**3_L**	**2_L**	**1_L**	**0_L**
×	×	×	H	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z	Hi-Z
L	L	L	L	H	H	H	H	H	H	H	L
L	L	H	L	H	H	H	H	H	H	L	H
L	H	L	L	H	H	H	H	H	L	H	H
L	H	H	L	H	H	H	H	L	H	H	H
H	L	L	L	H	H	H	L	H	H	H	H
H	L	H	L	H	H	L	H	H	H	H	H
H	H	L	L	H	L	H	H	H	H	H	H
H	H	H	L	L	H	H	H	H	H	H	H

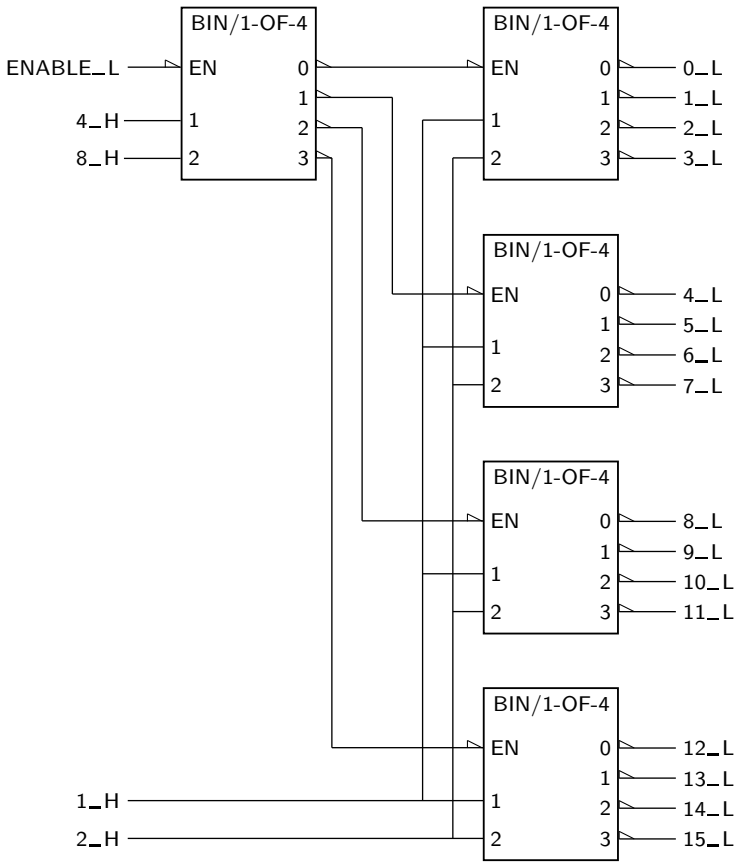


Figura 9.6: Logigrama com a estrutura de um descodificador binário de 4 bits com Enable, construído com descodificadores binários de 2 bits

tar funções na forma de uma soma de mintermos (ou seja, em primeira forma canónica), com auxilio de portas OR, como é óbvio.

Na Figura 9.7 ilustra-se o logigrama de um circuito usando esse princípio, que implementa a função lógica $F(a,b,c) = \sum m(0,2,3,5)$.

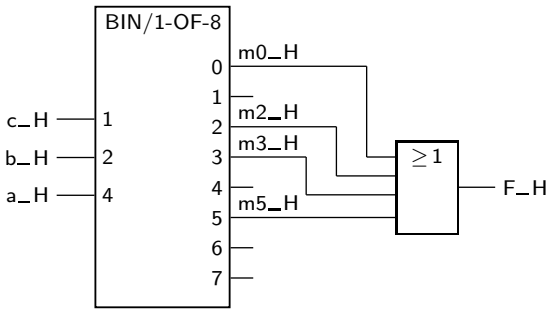


Figura 9.7: Implementação da função booleana simples $F(a,b,c) = \sum m(0,2,3,5)$ que utiliza um descodificador binário de 3 bits e uma porta OR com entradas e saída activas a H

De salientar a geração dos mintermos m_0 a m_7 da função $F(a,b,c)$ nas saídas do descodificador, todos activos a H. Como se pretende construir o logigrama da primeira forma canónica da função, isto é, uma *soma de mintermos*, vamos necessitar de uma porta OR com entradas activas a H. No caso em que se pretende obter $F_$H (como escolhemos fazer, mas podíamos ter tomado a opção de obter $F_$L), a saída da porta também deve ser activa a H.

9.2 Codificadores

Codificador

Um circuito com funcionamento algo inverso é o **codificador**, com tantas entradas quantas as palavras do código 1-em-n e com tantas saídas quanto o número de bits da palavra do código de saída. Ou seja, um codificador é um circuito que converte palavras do código 1-em-n na entrada para palavras de um código arbitrário na saída.

O problema maior com o codificador que acabámos de definir está na descrição do seu comportamento quando *mais que uma entrada está activa*, ou seja, quando deixamos de ter à entrada palavras do código 1-em-n. De facto, nessas circunstancias não e óbvio o que fazer, uma vez que na saída só pode estar presente uma das palavras do código.

Codificador de prioridades

Transcodificador

A solução clássica consiste em atribuir prioridades às entradas e codificar a entrada mais prioritária. Obtém-se, então, um circuito que se designa habitualmente por **codificador de prioridades**, embora uma designação mais correcta seja a de **transcodificador**.

Assim, para um codificador de prioridades com saídas no código BCD, em que se opta por dar às entradas com maior peso a maior prioridade, teremos a tabela de verdade física da Tabela 9.4 admitindo que as entradas e as saídas são todas activas a H.

Repare-se que é também necessário ter uma saída que indique se há alguma entrada a 1, para poder distinguir a situação em que não está nenhuma entrada a 1 da situação em que a entrada menos prioritária, $I0$, está a 1.

Tabela 9.4: Tabela de verdade física de um codificador de prioridades com saídas no código BCD, em que a entrada $I9$ é a mais prioritária

I0_H	I1_H	I2_H	I3_H	I4_H	I5_H	I6_H	I7_H	I8_H	I9_H	BCD	Val_H
L	L	L	L	L	L	L	L	L	L	××××	L
H	L	L	L	L	L	L	L	L	L	LLLL	H
×	H	L	L	L	L	L	L	L	L	LLLH	H
×	×	H	L	L	L	L	L	L	L	LLHL	H
×	×	×	H	L	L	L	L	L	L	LLHH	H
×	×	×	×	H	L	L	L	L	L	LHLL	H
×	×	×	×	×	H	L	L	L	L	LHLH	H
×	×	×	×	×	×	H	L	L	L	LHHL	H
×	×	×	×	×	×	×	H	L	L	LHHH	H
×	×	×	×	×	×	×	×	H	L	HLLL	H
×	×	×	×	×	×	×	×	×	H	HLLH	H

9.3 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 3.5 a 3.8, 3.10 e 3.12.

9.4 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 9.1 Na parte (a) da Figura 9.8 apresenta-se um descodificador binário de 3 bits com entradas activas a H, enquanto que na parte (b) se apresenta um outro descodificador, em tudo idêntico ao anterior excepto pelas entradas, que são agora activa a L. Se a estes descodificadores se aplicar às entradas a quantidade booleana geral (INA, INB, INC) = (H,L,L), qual é a saída que vem activa nos dois casos? Porquê?

9.2 Estabelecer o símbolo IEC, a tabela de verdade física e o logigrama interno de um descodificador binário de 4 bits com entradas activas a L e saídas activas a H. Qual o "fan-in de cada uma das entradas? Como faria para melhorar a situação?

(∗) 9.3 Escrever a tabela de verdade física do descodificador BCD da Figura 9.3.

9.4 Como será o símbolo IEC de um descodificador BCD com saídas activas a L? E com entradas activas a L?

9.5 Construa um logigrama com a estrutura interna de um descodificador BCD com saídas activas a L.

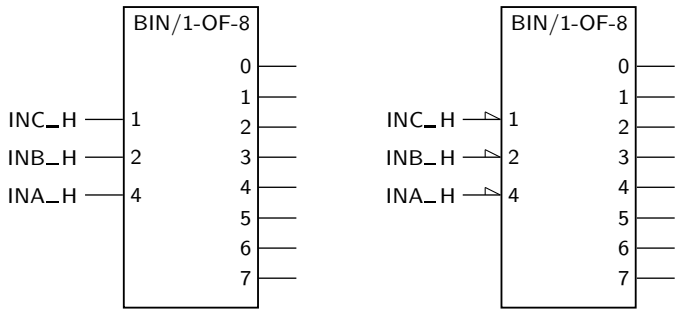


Figura 9.8: (a) Símbolo IEC de um descodificador binário de 3 bits com entradas activas a H; e (b) símbolo de um descodificador idêntico, mas com entradas activas a L

(∗) 9.6 Como será o símbolo IEC do descodificador expandido da Figura 9.6?

9.7 Construa um descodificador binário de 4 bits com Enable, à custa de descodificadores binários de 3 bits.

9.8 Construa um descodificador binário de 5 bits com Enable, à custa de descodificadores binários de 4 bits. Como será o símbolo IEC deste descodificador?

9.9 Construa um descodificador binário de 5 bits com Enable, à custa de descodificadores binários de 3 bits.

9.10 Utilizando descodificadores BCD e o mínimo de lógica adicional, projecte um descodificador binário com 4 bits.

9.11 Utilizando descodificadores BCD e o mínimo de lógica adicional, projecte um descodificador binário com 6 bits.

(∗) 9.12 Diga como poderá utilizar 9 descodificadores do tipo 74x138 para implementar um descodificador com 6 linhas de entrada e 64 linhas de saída. O descodificador 74x138 tem o símbolo IEC que se ilustra na Figura 9.9. *74x138*

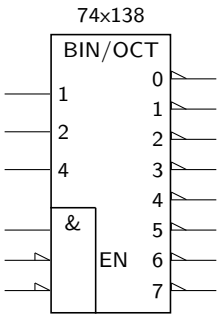


Figura 9.9: Símbolo IEC de um descodificador 74x138

9.13 Tendo-se, na Figura 9.7, ligado a variável A à entrada 4, a variável B à entrada 2 e a variável C à entrada 1, o que é que se pode afirmar em relação aos pesos das variáveis?

9.14 Se na Figura 9.7 fizessemos as ligações das entradas de forma diferente, por exemplo ligando A a 1, B a 2 e C a 4, mas mantendo os pesos das variáveis, qual seria a função implementada nesse caso?

(∗) 9.15 Implementar a função booleana simples $F(a,b,c) = \sum m(1-3,7)$ utilizando um descodificador binário de 3 bits com saídas activas a H, como o da Figura 9.1. Que funções e correspondentes níveis de actividade se obtêm nas saídas do descodificador?

(∗) 9.16 Implementar a função booleana simples $F(a,b,c) = \sum m(1-3,7)$ do exercício anterior, mas utilizando agora um descodificador binário de 3 bits com saídas activas a L, como o 74x138 da Figura 9.9. Que funções e correspondentes níveis de actividade se obtêm nas saídas do descodificador?

(∗) 9.17 Implementar a função booleana simples $F(a,b,c) = \prod M(0,4-6)$ utilizando um descodificador binário de 3 bits com saídas activas a H, como o da Figura 9.1 (de notar que esta função é a mesma dos Exercícios 9.15 e 9.16). Que funções e correspondentes níveis de actividade se obtêm nas saídas do descodificador?

(∗) 9.18 Implementar a função booleana simples $F(a,b,c) = \prod M(0,4-6)$ utilizando um descodificador binário de 3 bits com saídas activas a L, como o 74x138 da Figura 9.9 (de notar que esta função é a mesma dos Exercícios 9.15 e 9.16). Que funções e correspondentes níveis de actividade se obtêm nas saídas do descodificador?

9.19 Repetir os Exercícios 9.15 a 9.18 na implementação da função booleana simples $\overline{F}(a,b,c)$.

74x42 9.20 Dado o descodificador 74x42 da Figura 9.10, diga como faria para gerar, à custa dele e, eventualmente, de portas lógicas suplementares, a função $f(A,B,C,D) = \sum m(1,2,4,8)$. É possível, com este descodificador e lógica suplementar, gerar qualquer função de 4 variáveis? Porquê?

9.21 Usando um descodificador e a lógica suplementar que achar convenientes, construa a função $f = A\,B + B\,\overline{C}$.

9.22 Considere o circuito da Figura 9.11. Qual é a expressão lógica da função que ele implementa?

9.23 É dado o logigrama da Figura 9.12. Utilizando um descodificador à sua escolha, implemente a função realizada pelo circuito dado.

9.24 Desenhe um logigrama com a estrutura interna do codificador BCD da Tabela 9.4.

(∗) 9.25 Traçar o logigrama de um codificador de prioridades com 4 entradas, I0 a I3, e duas saídas, A1 e A0. A entrada I3 deverá ter prioridade sobre I2 que, por sua vez, deverá ter prioridade sobre I1, e esta sobre I0. Prever ainda a existência de uma entrada de Enable e de duas outras saídas,

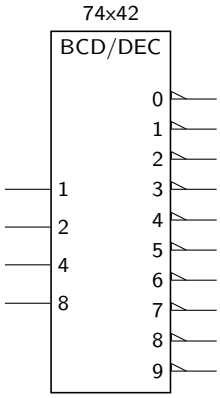


Figura 9.10: Símbolo IEC de um descodificador 74x42

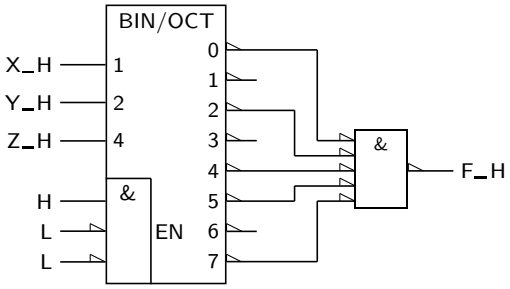


Figura 9.11: Logigrama de um circuito que implementa uma função booleana simples cuja expressão se pretende determinar

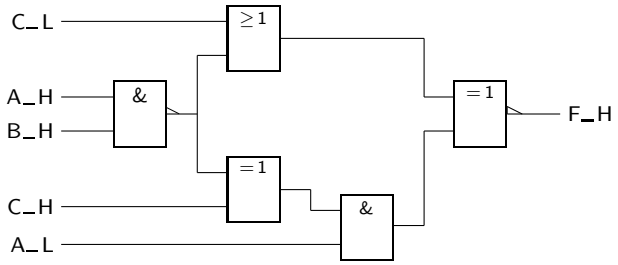


Figura 9.12: Logigrama utilizado no Exercício 9.23

uma de Enable e outra de Grupo, em que esta última indica se, estando o codificador activo, há pelo menos uma entrada activa. Todas as entradas deverão ser activas a H.

9.26 Desenhe um transcodificador que aceita nas suas entradas um dígito BCD e fornece nas suas saídas o código requerido para acender um display de 7 segmentos representativo do dígito na entrada.

9.27 Implemente um transcodificador do CBR (código binário reflectido) para o código de 7 segmentos, utilizando um transcodificador do código BCD para o código de 7 segmentos.

9.28 Repetir o Exercício 8.4, mas usando agora um codificador e um descodificador binários.

(∗) 9.29 Implementar as funções booleanas simples

a) $f(A, B, C, D) = \sum m(0, 3, 5, 10, 11)$;

b) $g(A, B, C, D, E) = \prod M(0, 5, 7, 14, 30, 31)$

usando apenas um descodificador do tipo 74x138 e a lógica suplementar mínima que considerar necessária. Admitir que, nas duas funções, A é a variável booleana simples com maior peso.

Capítulo 10

Multiplexers e Demultiplexers

10.1 Multiplexers

O multiplexer é um circuito combinatório muito comum em sistemas digitais.

A sua funcionalidade básica é a de um circuito que realiza a **função de seleccção de uma entrada**, sugerida pela analogia mecânica da Figura 10.1. Naturalmente, o número de entradas pode ser diferente de quatro mas, para o efeito da função de selecção que pretendemos, admitiremos que ele é sempre uma potência de 2.

Selecção de uma entrada

Analogia mecânica de um multiplexer

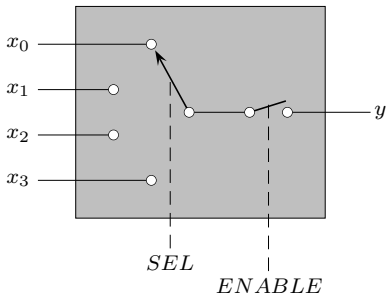


Figura 10.1: Analogia entre um comutador mecânico e a função de selecção de uma entrada realizada por um multiplexer

Um multiplexer tem, portanto, 2^n **entradas de dados** das quais selecciona uma, e **entradas de controlo** ou **de selecção** que permitem escolher a entrada de dados que, em cada momento, vê encaminhado o nível de tensão nela aplicado para a **saída de dados** do multiplexer.

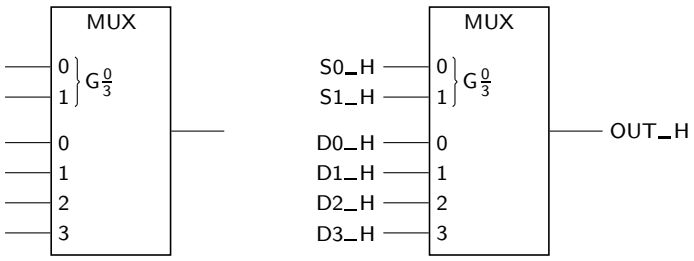
Entradas de dados

Entradas de selecção ou de controlo

Saída de dados

Um multiplexer de 4 entradas de dados e com entradas e saída activas a H, por exemplo, tem o símbolo IEC da Figura 10.2(a).

Na Figura 10.2(b), as entradas Di são de dados, e as entradas Si são de selecção. O número binário colocado no par $(S1, S0)$, sendo $S1$ a variável booleana simples



(a) (b)

Figura 10.2: (a) Símbolo de um multiplexer com 4 entradas de dados, com entradas e saída activas a H, de acordo com a norma IEC 60617-12; (b) o mesmo multiplexer, ao qual se juntaram variáveis de selecção $S0$ e $S1$, de dados $D0$ a $D3$, e uma função de saída OUT

com maior peso (porque está aplicada à entrada de selecção de maior peso), determina a entrada de dados cujo valor é colocado na saída.

Dependência And (G)

O símbolo $\mathsf{G}^{\mathsf{0}}_{\mathsf{3}}$ é uma abreviatura para as **dependências And** G0, G1, G2, G3, aplicadas às entradas 0, 1, 2 e 3, respectivamente.

As entradas de selecção, 0 e 1, indicam os pesos das potências de 2 que ponderam as palavras do CBN aplicadas ás entradas SEL0 e SEL1. Por exemplo, se SEL0 e SEL1 estiverem ambas activadas, os pesos somam $2^0 + 2^1 = 3$ e G3 vem activada, o que faz com que a entrada 3 venha seleccionada. Nessas condições, a quantidade booleana simples aplicada a essa entrada de dados vem reproduzida na saída.

A descrição funcional deste multiplexer será a da Tabela 10.1, e a tabela de verdade física correspondente vem descrita pela Tabela 10.2, admitindo-se que todas as entradas e a saída são activas a H..

Tabela 10.1: Descrição funcional do multiplexer da Figura 10.2(b)

S1_H	S0_H	OUT_H
L	L	$D0$_H
L	H	$D1$_H
H	L	$D2$_H
H	H	$D3$_H

A estrutura interna deste multiplexer vem ilustrada na Figura 10.3, mais uma vez admitindo-se que todas as entradas e a saída são activas a H.

É fácil de explicar o funcionamento do multiplexer a partir da sua estrutura interna. De facto, cada uma das quatro possiveis configurações das variáveis $S0$ e $S1$ fazem que as saídas de três dos ANDs estejam forcosamente a L e que a saída do quarto AND tenha um nível de tensão igual ao da respectiva entrada Di. A saída OUT terá, portanto, o nível $L + L + L + Di = Di$.

Por exemplo, se $S0$ e $S1$ estiverem ambas inactivas as duas entradas inferiores

Tabela 10.2: Tabela de verdade física do multiplexer da Figura 10.2(b)

D3_H	D2_H	D1_H	D0_H	S1_H	S0_H	OUT_H
×	×	×	L	L	L	L
×	×	×	H	L	L	H
×	×	L	×	L	H	L
×	×	H	×	L	H	H
×	L	×	×	H	L	L
×	H	×	×	H	L	H
L	×	×	×	H	H	L
H	×	×	×	H	H	H

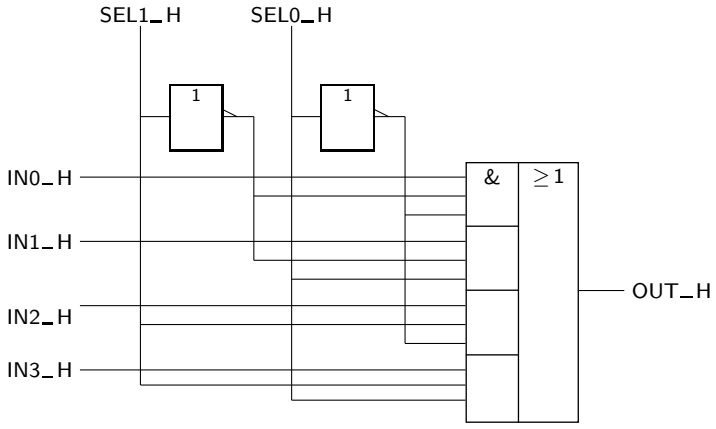


Figura 10.3: Logigrama com a estrutura interna do multiplexer da Figura 10.2(b), com entradas e saída activas a H, de acordo com a norma IEC 60617-12

da porta AND de cima estão activas e, nessas circunstâncias, a saída desse AND vem igual a $D0$. Os outros ANDs têm as saídas inactivas (a L) uma vez que está inactiva pelo menos uma das 2 entradas inferiores de cada uma das portas. A saída OUT é, então, igual a $D0$.

Da mesma forma, as outras configurações possíveis de $S0$ e de $S1$ seleccionam uma das outras 3 entradas, e o nível de tensão na saída vem igual ao dessa entrada.

Por vezes, convém que o multiplexer seja dotado de uma **entrada de Enable** que permita controlar melhor o seu funcionamento. O Enable permite, quando activo, que o multiplexer tenha a funcionalidade descrita. Quando inactivo, faz com que a saída do dispositivo esteja inactiva, independentemente dos valores lógicos aplicados às entradas.

Entrada de Enable

Na Figura 10.4 vem desenhado o símbolo IEC de um multiplexer idêntico ao da Figura 10.2 mas com entrada de Enable activa a L.

Repare-se que este multiplexer foi desenhado com um Enable activo a L (situação

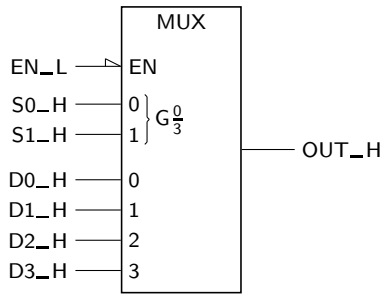


Figura 10.4: Símbolo IEC de um multiplexer em que todas as entradas e a saída são activas a H, com excepção da entrada de Enable que é activa a L

muito frequente), o que quer dizer que a função de selecção só é feita quando tivermos aplicado um L à linha de Enable. Caso contrário, a saída vem inactiva.

Na Tabela 10.3 apresenta-se a tabela de verdade física do multiplexer da Figura 10.4. De notar que esta tabela é idêntica à da Tabela 10.2 para todas as linhas em que a entrada de Enable está activa. Quando esta entrada vem desactivada, a saída do multiplexer vem inactiva.

Tabela 10.3: Tabela de verdade física do multiplexer com Enable da Figura 10.4

D3_H	D2_H	D1_H	D0_H	S1_H	S0_H	EN_L	OUT_H
×	×	×	×	×	×	H	L
×	×	×	L	L	L	L	L
×	×	×	H	L	L	L	H
×	×	L	×	L	H	L	L
×	×	H	×	L	H	L	H
×	L	×	×	H	L	L	L
×	H	×	×	H	L	L	H
L	×	×	×	H	H	L	L
H	×	×	×	H	H	L	H

Quanto à estrutura interna deste multiplexer, podemos vê-la na Figura 10.5.

10.1.1 Simbolos dos multiplexers

Já vimos a subsecção anterior os símbolos IEC 60617-12 de alguns multiplexers. Agora vamos estudar mais alguns símbolos.

74x151

Na Figura 10.6 está representado um multiplexer do tipo 74x151 com 8 entradas de dados. Anote-se que:

Qualificador geral MUX

— o facto de se tratar de um multiplexer é indicado pelo **qualificador geral** MUX na parte superior do rectângulo;

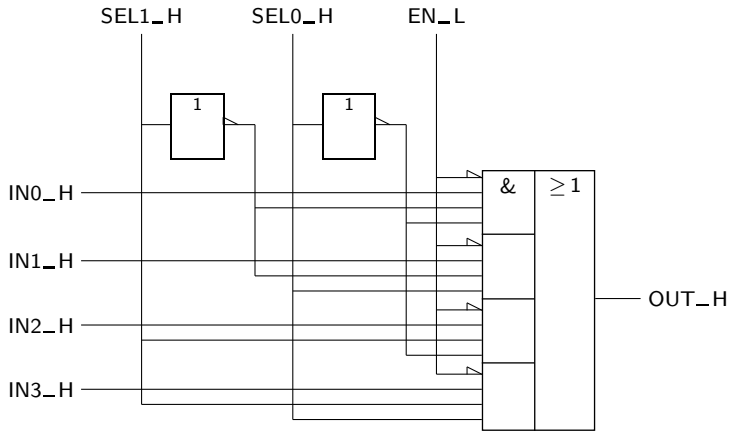


Figura 10.5: Logigrama com a estrutura interna do multiplexer da Figura 10.4

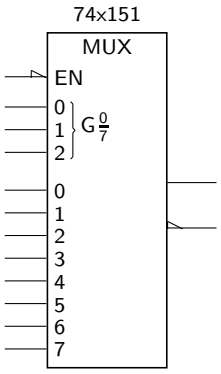


Figura 10.6: Símbolo IEC de um multiplexer do tipo 74x151, com 8 entradas de dados activas a H e Enable activo a L, e com duas saídas complementares

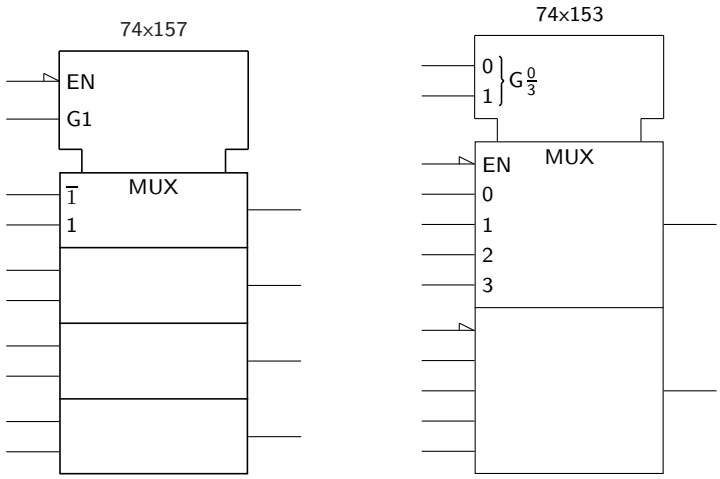
— as três entradas de selecção devem ser entendidas em conjunto (daí a chaveta); quando se aplica a essas entradas uma determinada quantidade booleana geral, o seu equivalente decimal é gerado internamente e vem activada a entrada com ese número (como sabemos, o G representa uma dependência And);

— o Enable é, neste multiplexer, activo a L;

— o circuito representado possui 2 saídas, sendo uma a negação da outra.

O simbolo IEC da Figura 10.7(a) representa um multiplexer quádruplo com a designação 74x157, com 2 entradas de dados em cada multiplexer individual, e com entradas de selecção e de Enable comuns. Por essa razão, estas linhas estão num **bloco de controlo comum** colocado no topo do símbolo. *74x157*

Quanto ao circuito da Figura 10.7(b), trata-se de um multiplexer duplo com 4 entradas de dados e entradas de Enable independentes, e entradas de selecção comuns (o 74x153). *74x153*



(a) (b)

Figura 10.7: (a) Símbolo de um multiplexer quádruplo (o 74x157), de acordo com a norma IEC 60617-12 — cada multiplexer possui 2 entradas de dados, e entradas de selecção e de Enable no bloco de controlo comum; com excepção do Enable, as entradas e as saídas são todas activas a H; (b) um multiplexer duplo (o 74x153), em que cada multiplexer individual possui 4 entradas de dados activas a H e entrada de Enable independente, activa a L, e com entradas de selecção comuns, activas a H

10.1.2 Expansão de multiplexers

A expansão de multiplexeres pode ser feita de duas formas básicas.

Estrutura em árvore

— Uso de camadas sucessivas de multiplexers, numa **estrutura em árvore**. Na Figura 10.8 ilustra-se a construção de um multiplexer com 16 entradas de dados, a partir de multiplexers de 4 entradas de dados.

De notar, em particular, a designação dadas às entradas IN0 a IN15, cuja numeração é função dos pesos relativos dos multiplexers. Por exemplo, o multiplexer superior é o que corresponde às entradas com índices de menor peso (0 a 3), dado que este multiplexer vem seleccionado com $(SEL3, SEL2) = = (L, L)$; o segundo multiplexer corresponde às entradas com índices imediatamente superiores (4 a 7), porque este multiplexer vem seleccionado com $(SEL3, SEL2) = (L, H)$ — reparar que $SEL3$ tem mais peso do que $SEL2$; etc.

— Utilização de um descodificador auxiliar e de uma porta OR. Na Figura 10.9 ilustra-se este tipo de solução para um multiplexer funcionalmente idêntico ao anterior, embora com o dobro das entradas de dados e, naturalmente, mais uma entrada de selecção.

De notar, agora, que o multiplexer U1 é o que corresponde às entradas com índices de menor peso (0 a 7), dado que ele contribui com a única entrada activa (a L) do OR quando o seu Enable está activo, isto é, quando $(SEL4, SEL3) =$ (L,L); da mesma forma, o multiplexer U2 corresponde às entradas com índices imediatamente superiores (8 a 15) porque ele contribui

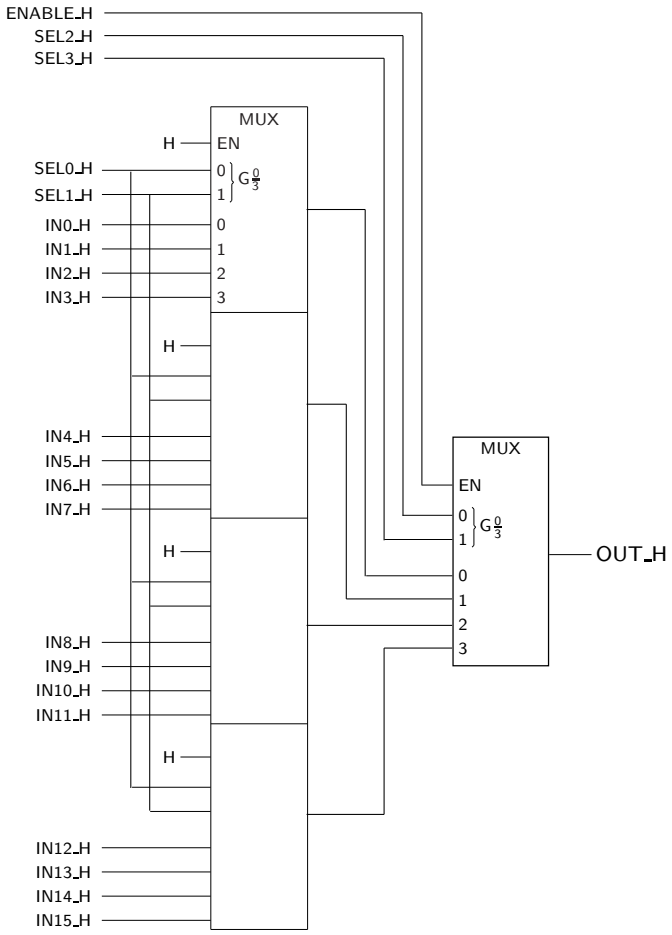


Figura 10.8: Logigrama de um multiplexer com 16 entradas de dados, realizado à custa de multiplexers com 4 entradas de dados, numa estrutura em árvore

com a única entrada activa do OR quando o seu Enable está activo, isto é, quando $(SEL4, SEL3) = (\text{L,H})$ — reparar que $SEL4$ tem mais peso do que $SEL3$; etc.

10.2 Demultiplexers

Selecção de uma saída

A funcionalidade básica de um demultiplexer é oposta à da de um multiplexer.

Analogia mecânica de dum demultiplexer

Nesse sentido, um demultiplexer realiza a **função de seleccção de uma saída**, com a analogia mecânica sugerida pela Figura 10.10.

De forma idêntica à que acontecia para as entradas de dados dos multiplexers, admitiremos que o número de saídas do demultiplexer é sempre uma potência de 2.

Saídas de um demultiplexer

Um demultiplexer tem, portanto, 2^n **saídas**, das quais uma vem, em cada momento, seleccionada por intermédio de n **entradas de controlo** ou **de selecção**, que

Entradas de selecção ou de controlo

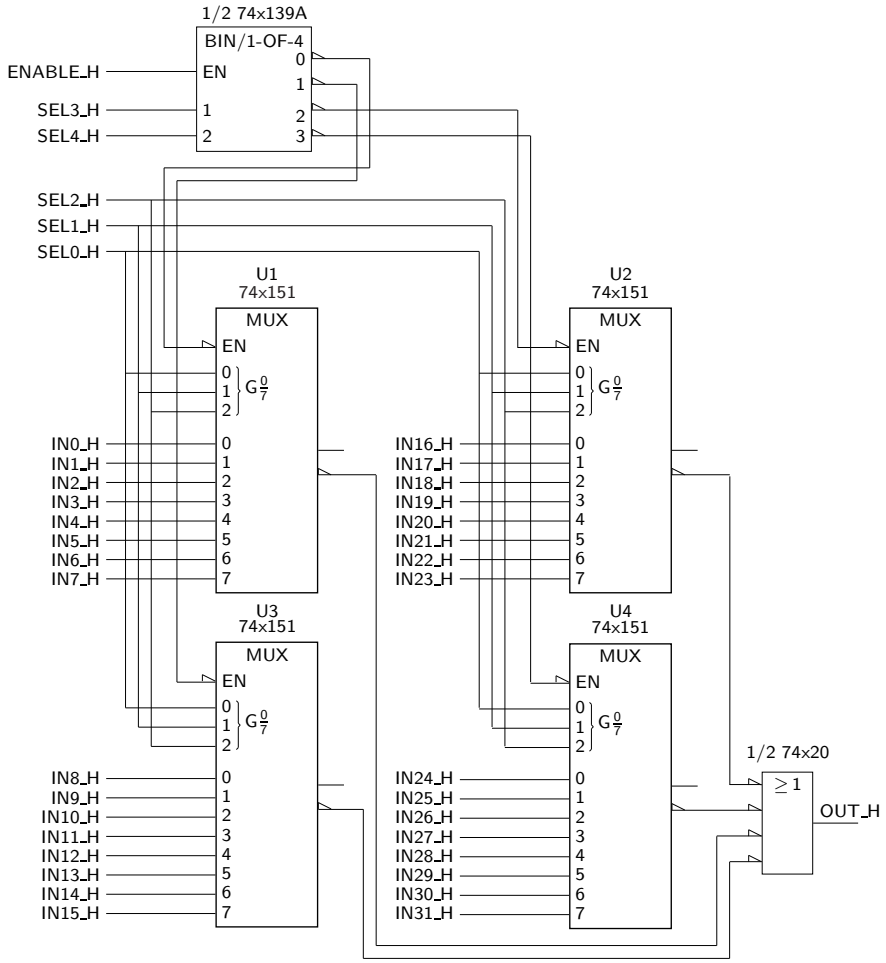


Figura 10.9: Esquema eléctrico de um multiplexer de 32 entradas, realizado à custa de multiplexers de 4 entradas, de um descodificador auxiliar e de uma porta OR

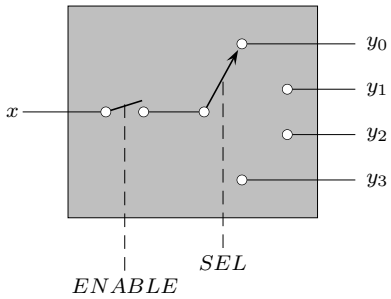


Figura 10.10: Analogia entre um comutador mecânico e a função de selecção de uma saída realizada por um demultiplexer

Entrada de dados

permitem veicular o nível de tensão aplicado à **entrada de dados** para a saída seleccionada.

Na Figura 10.11 apresenta-se o logigrama de um demultiplexer com 4 saídas.

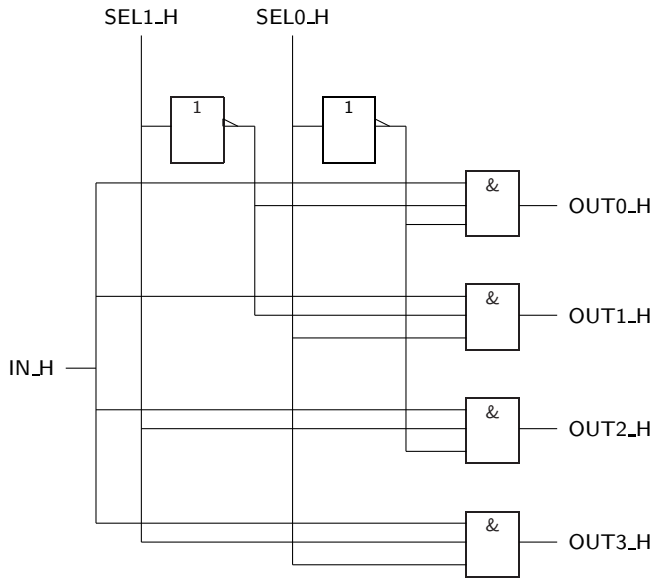


Figura 10.11: Logigrama de um demultiplexer com 4 saídas

É de notar que a estrutura deste demultiplexer é igual à da de um descodificador binário de 2 bits com Enable, de que se apresenta um exemplo na Figura 10.12.

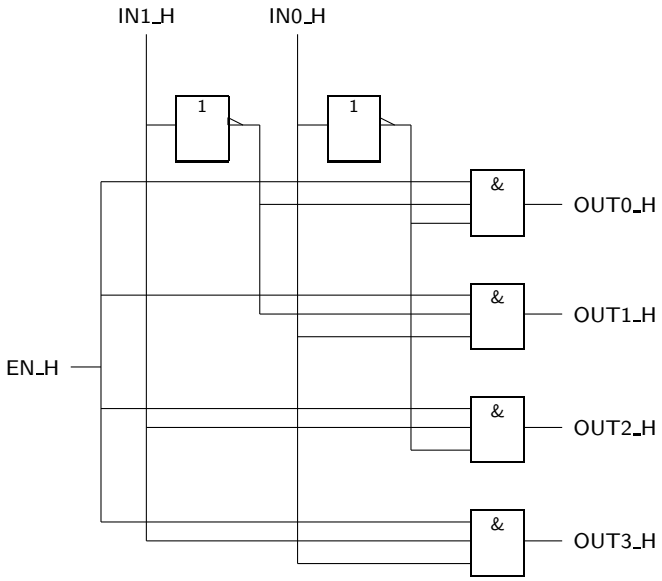


Figura 10.12: Logigrama de um descodificador binário de 2 bits com Enable, que acentua a semelhança com o demultiplexer com 4 saídas da Figura 10.11

Com efeito, basta identificar a entrada de Enable do descodificador com a entrada de dados do demultiplexer e as entradas de dados do descodificador com as entradas de selecção do demultiplexer.

Qualificador geral DMUX ou DX

Não admira, por isso, que os símbolos IEC dos demultiplexers sejam semelhantes aos dos descodificadores com Enable, distinguindo-se destes pelo **qualificador geral** DMUX, ou DX, como mostra a Figura 10.13 para um demultiplexer com 4 saídas.

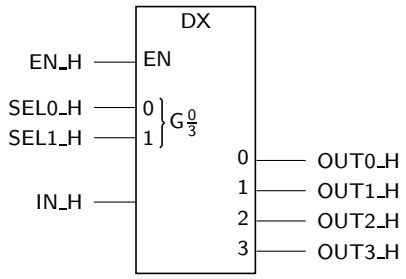


Figura 10.13: Símbolo IEC de um demultiplexer com 4 saídas, a que se juntaram designações possíveis para as entradas e saídas

Os restantes qualificadores têm significados em tudo idênticos aos que foram utilizados a propósito dos multiplexers.

Podemos, então, estabelecer facilmente a tabela de verdade física deste tipo de circuito, como se indica na Tabela 10.4 para um demultiplexer com 4 saídas e Enable, todas activas a H.

Tabela 10.4: Tabela de verdade física compactada de um demultiplexer com 4 saídas e Enable, todas activas a H

EN_H	SEL1_H	SEL0_H	OUT3_H	OUT2_H	OUT1_H	OUT0_H
L	×	×	L	L	L	L
H	L	L	L	L	L	IN
H	L	H	L	L	IN	L
H	H	L	L	IN	L	L
H	H	H	IN	L	L	L

74x138

Para finalizar, na Figura 10.14 apresenta-se o símbolo IEC do 74x138, um demultiplexer com 8 saídas que podemos identificar, em alternativa, com um descodificador binário de 3 bits.

Na interpretação como demultiplexer [Figura 10.14(a)], o 74x138 possui uma única entrada de dados (como seria de esperar) formada pelo AND entre IN0, IN1 e IN2, sendo duas dessas entradas activas a L e a outra a H.

Na interpretação como descodificador [Figura 10.14(b)], o 74x138 possui três entradas de dados, IN0, IN1 e IN2, e uma entrada de Enable formada pelo AND entre EN0, EN1 e EN2, sendo dois dos Enables activos a L e o outro a H.

74x155

Na Figura 10.15, por seu turno, está representado o símbolo IEC do 74x155. Trata-se de um demultiplexer duplo com: (i) entradas de selecção comuns aos

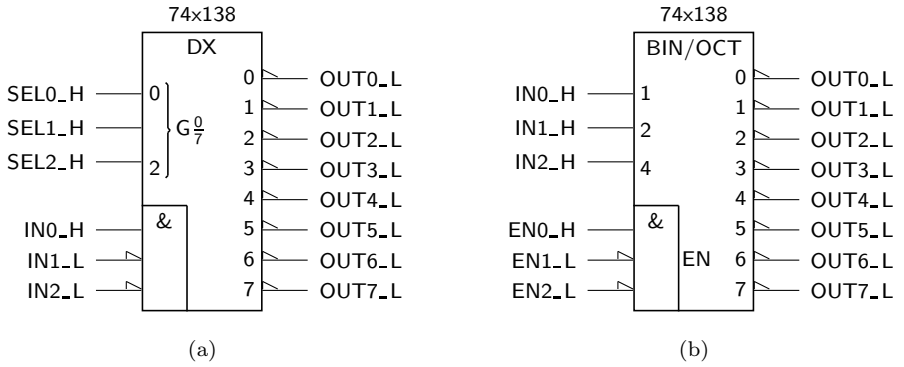


Figura 10.14: (a) Símbolo IEC do 74x138, utilizado como demultiplexer; e (b) símbolo do 74x138, utilizado como descodificador com Enable

dois demultiplexers simples, entradas essas identificadas no bloco de controlo comum; (ii) entradas de Enable independentes e activas a L para cada um dos demultiplexers simples; (iii) entrada de dados activa a H para um dos demultiplexers e activa a L para o outro; e (iv) saídas activas a L.

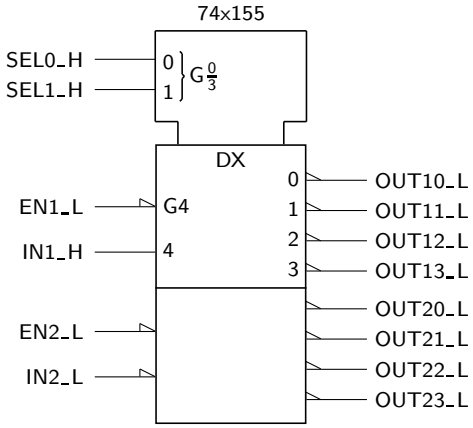


Figura 10.15: Símbolo IEC do 74x155

10.3 Aplicações dos Multiplexers e dos Demultiplexers

Uma das aplicações dos multiplexers é a de realizar a multiplexagem e demultiplexagem de um conjunto de variáveis, como mostra a Figura 10.16.

Uma outra aplicação de um multiplexer é na implementação de funções. Tal torna-se possível porque um multiplexer gera internamente uma soma de produtos de todas as variáveis de selecção, e portanto, uma soma de mintermos.

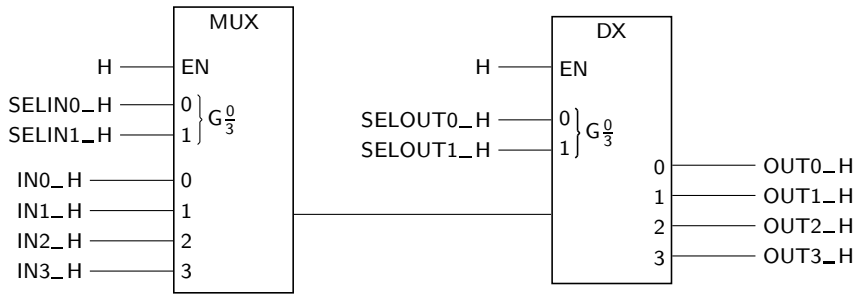


Figura 10.16: Multiplexagem e demultiplexagem de dados, conseguida à custa de um multiplexer e de um demultiplexer

Considere-se, por exemplo, a função descrita pela tabela de verdade lógica da Tabela 10.5.

Tabela 10.5: Tabela de verdade lógica de uma função booleana simples de 2 variáveis booleanas simples

A	**B**	**f(A,B)**
0	0	0
0	1	1
1	0	1
1	1	0

Utilizando um multiplexer de 2 variáveis de selecção e 4 variáveis de entrada de dados (tantas quantas as linhas da tabela), comecemos por ligar as variáveis da função às variáveis de selecção do multiplexer *pela mesma ordem de pesos*, como se indica na Figura 10.17(a).

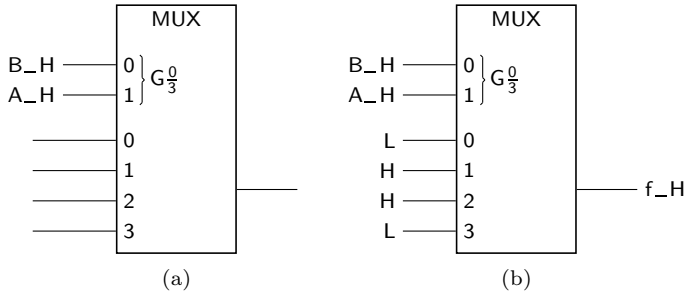


Figura 10.17: (a) Ligação das variáveis booleanas simples A e B às entradas de selecção de um multiplexer com 4 entradas de dados; (b) ligações às entradas de dados

Quando $A = 0$ e $B = 0$, o multiplexer selecciona para a sua saída o valor presente na entrada de dados 0. A função, nessas circunstâncias, deve dar 0 (primeira linha da tabela). Se ligarmos o nível L à entrada 0 do multiplexer, este, quando $A = B = 0$, apresentará na saída o valor da função (nível L).

Se fizermos o mesmo para o resto das entradas do multiplexer e das linhas respectivas da tabela, o multiplexer ficará a representar toda a função, como se indica na Figura 10.17(b).

Isso permite usar um multiplexer de n variáveis de selecção para construir qualquer função de n variáveis.

Mas pode-se ir um pouco mais longe. Observe-se o seguinte exemplo: usando o mesmo multiplexer de duas entradas de selecção, vamos agora implementar uma função booleana simples de 3 variáveis, por exemplo a função da Tabela 10.6.

Tabela 10.6: Tabela de verdade lógica de uma função booleana simples de 3 variáveis booleanas simples

A	B	C	f(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Comecemos por ligar duas das variáveis de entrada às entradas de selecção do multiplexer. Podemos escolher quaisquer duas variáveis, mas será mais óbvio se escolhermos as duas mais significativas, A e B. Ligamo-las de forma idêntica à que se utilizou anteriormente, como mostra a Figura 10.17(a).

Repare-se agora no que acontece quando $A = 0$ e $B = 0$. O multiplexer selecciona a entrada de dados 0. Mas a função tem duas linhas com $A = 0$ e $B = 0$, e para ambas o valor da função é 0. Logo, podemos aplicar um nível L à entrada 0, como sugere a Figura 10.18(a).

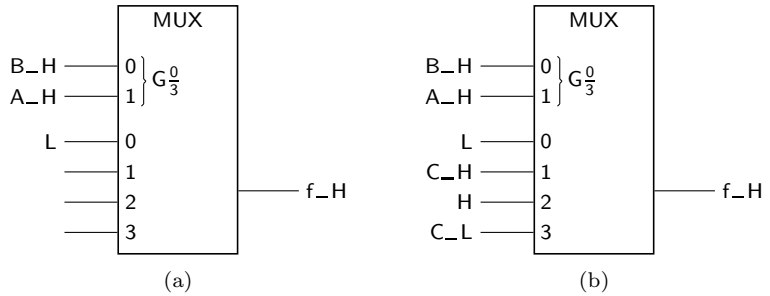


Figura 10.18: (a) Aplicação de um nível L à entrada 0 do multiplexer; (b) de uma maneira geral, temos de aplicar às entradas de dados as funções booleanas simples possíveis da variável booleana simples que sobeja, C, depois de escolhidas A e B para aplicar às entradas de selecção

Na situação seguinte ($A = 0$ e $B = 1$) já a tabela não tem o mesmo valor para a função nas duas linhas. De facto, com $A = 0$ e $B = 1$ a função vale 0 quando $C = 0$ e vale 1 quando $C = 1$. Isto é, *nessa fatia da tabela* a função vale C. Então, a entrada 1 de dados do multiplexer deve vir ligada a C, como se indica na Figura 10.18(b).

Fazendo o mesmo para o resto da tabela, obtém-se a Tabela 10.7.

Tabela 10.7: Tabela de verdade lógica da função de 3 variáveis que temos vindo a estudar, dividida em 4 fatias (porque temos duas das variáveis, A e B, a servir de selecção do multiplexer)

A	B	C	f(A,B,C)	
0	0	0	0	$f = 0$
0	0	1	0	
0	1	0	0	$f = C$
0	1	1	1	
1	0	0	1	$f = 1$
1	0	1	1	
1	1	0	1	$f = \overline{C}$
1	1	1	0	

E o circuito final fica como se indica na Figura 10.18(b).

É possível, como se exemplificou atrás, construir qualquer função f de n variáveis com um multiplexer de $n - 1$ variáveis de selecção e, eventualmente, uma negação.

De uma forma geral, escolhemos $n - 1$ variáveis (arbitrariamente, embora seja mais simples escolher as de maior peso) que aplicamos às entradas de selecção, e a cada uma das entradas de dados aplicaremos uma função da variável que resta (como sabemos, as funções de uma variável X são 0, 1, X e $\overline{X}$), função essa escolhida adequadamente para cada fatia de duas linhas da tabela de verdade da função f a implementar.

10.4 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 3.5 a 3.8, 3.10 e 3.12.

10.5 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 10.1 Desenhar a tabela de verdade física e o logigrama com a estrutura interna de um multiplexer com 4 entradas de dados, admitindo que as entradas de dados e a saída são activas a L, e que as entradas de selecção são activas a H.

(∗) 10.2 Desenhar o símbolo IEC de um multiplexer idêntico ao do da Figura 10.4, mas com saída "tri-state".

(∗) 10.3 Escrever a tabela de verdade física do multiplexer do exercício anterior.

(∗) 10.4 Desenhar o logigrama de um multiplexer com uma estrutura em árvore e com 16 entradas de dados, formado por um primeiro nível com multiplexers de 2 entradas de dados e um segundo nível formado por um multiplexer com 8 entradas de dados. Admitir que as entradas e a saída são todas activas a H.

10.5 Dado o descodificador duplo da Figura 10.19, do tipo 74x139, diga como o pode utilizar para demultiplexar dois bits de dados, A_L e B_L. *74x139*

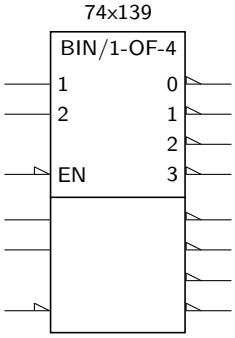


Figura 10.19: Símbolo IEC de um descodificador duplo do tipo 74x139

10.6 Diga como pode ligar dois multiplexers como o da Figura 10.20, do tipo 74x251, de modo a construir um multiplexer com 16 entradas e 1 saída. Use a lógica discreta suplementar que entender necessária. *74x251*

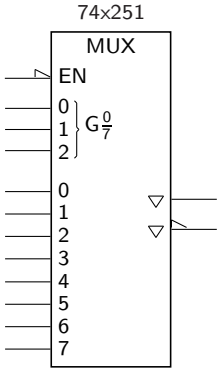


Figura 10.20: Símbolo IEC de um multiplexer 74x251

10.7 Suponha que é dado o multiplexer da Figura 10.20, do tipo 74x251. Pretende-se obter, numa das suas saídas, o nível de tensão oposto ao nível aplicado à entrada 5 de dados. Quais os níveis a aplicar às várias entradas, e qual a saída pretendida?

10.8 Utilize o multiplexer da Figura 10.2(a), na página 170, para gerar a função booleana simples $Z = S_1\,S_0 + S_0\,V + \overline{S_1}\,\overline{S_0}\,V$.

10.9 Diga como pode utilizar o multiplexer da Figura 10.2(a), na página 170, para gerar a função booleana simples $Z = S_1\,\overline{S_0} + S_0\,W + V\,W + S_0\,\overline{W}$.

10.10 Utilize um multiplexer com 8 entradas de dados para gerar a função $Z = \sum m(0, 3, 5, 6, 9, 10, 12, 15)$. Trace um logigrama que implemente a mesma função, mas que utiliza apenas ANDs com entradas activas a H e saída activa a L. Se só utilizar circuitos integrados que contêm, cada um, dois ANDs deste tipo com 4 entradas, compare as duas soluções de implementação contabilizando, em cada caso, o número total de integrados.

10.11 Utilize um multiplexer com 8 entradas de dados para gerar a função $f(A, B, C, D, E) = \sum m(0 - 5, 10, 13, 20 - 25, 30, 31)$.

10.12 Dispõe de 4 multiplexeres com 3 entradas de controlo, e de um descodificador de 2 entradas. Projecte, usando o material referido, um multiplexer com 5 entradas de controlo (e, portanto, com 32 entradas de dados). Especifique as suposições que fez em relação ao material de que dispõe. Se necessitar, pode utilizar portas lógicas suplementares.

10.13 Dispõe de multiplexeres com 3 entradas de controlo, e de descodificadores de 3 entradas. Especificando em pormenor os circuitos que usar, comstruir um multiplexer com 6 entradas de controlo.

10.14 Usando um multiplexer com 3 entradas de controlo e a lógica adicional que entender, sintetize as seguintes funções booleanas simples:

a) $f = (A \oplus B) \oplus (C \oplus D)$;
b) $f = (A \oplus B)\,(C \oplus D)$;
c) $f = A\,B + \overline{A}\,C\,D + A\,\overline{B}\,C\,\overline{D}$;
d) $f = \sum m(0, 2, 3, 5, 7, 12 - 14) + \sum m_d(1, 10, 15)$;
e) $f = A\,B\,(\overline{C} \oplus D) + A\,B\,C + \overline{A}\,\overline{C}\,D + \overline{B}\,C\,D$.

10.15 Considere o circuito da Figura 10.21.

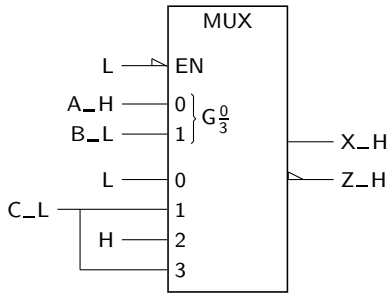


Figura 10.21: Circuito utilizado no Exercício 10.15

a) Qual é a expressão lógica da função booleana simples X, expressa nas variáveis A, B e C?
b) Se as variáveis A, B e C estiverem activas, qual é o nível de tensão na linha Z_H?

10.16 Considere o circuito da Figura 10.22.

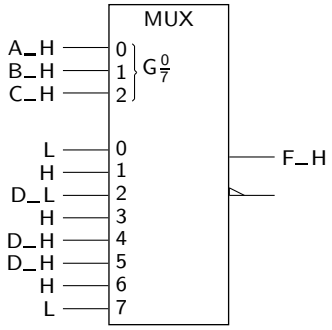


Figura 10.22: Circuito utilizado no Exercício 10.16

a) Sendo A a variável de maior peso e D a de menor peso, diga qual a função $f(A, B, C, D)$ sintetizada pelo multiplexer da figura. Simplifique-a pelo processo que entender.
b) Quando ocorrer uma mudança na configuração das variáveis de controlo, poderão ocorrer picos na saída? Porquê?
c) Como construiria *a mesma função* na saída activa a L do multiplexer?

10.17 Usando um multiplexer de 3 entradas de controlo, construa um circuito que realiza a função construída na Figura 9.11 (Exercício 9.22, na página 166).

10.18 É dada a seguinte função, $f = A\,B + B\,C + \overline{C}\,A + A\,B\,\overline{C}$.

a) Implemente-a usando um multiplexer com 4 entradas de dados.
b) Usando o multiplexer anterior, quantas implementações diferentes da função poderia fazer? Porquê?

10.19 Considere o circuito da Figura 10.23.

a) Qual é a função concretizada pelo circuito?
b) Sintetize de novo a função usando o mínimo de lógica possível.

10.20 Considere que dispõe de multiplexers de 8 linhas de dados e uma linha de Enable activa a L. A saída do multiplexer é activa a L, e é do tipo TTL convencional (totem-pole). Usando o mínimo de lógica adicional, construa um multiplexer com 16 entradas de dados.

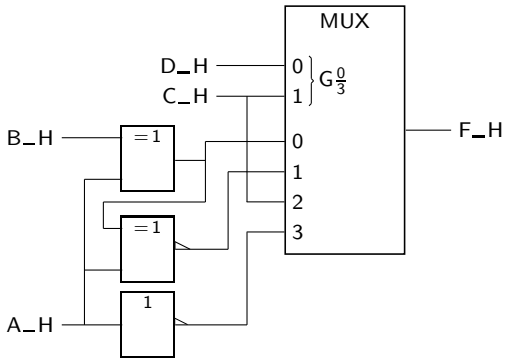


Figura 10.23: Circuito utilizado no Exercício 10.19

Capítulo 11

Circuitos Aritméticos

11.1 Somadores Binários

Circuitos aritmeticos são aqueles que realizam operações aritméticas sobre, em princípio, números binários.

Para realizar circuitos aritméticos teremos que utilizar funções lógicas, uma vez que são estas de que dispomos em termos de circuitos.

O circuito mais simples que podemos considerar é o circuito que implementa a soma de dois algarismos binários.

Recordemos a Tabela 1.3 da soma na base 2, que reproduzimos na Tabela 11.1 sob uma forma ligeiramente diferente, para facilitar a exposição.

Tabela 11.1: Tabela da adição no sistema binário

Adição binária		
$A+B$	$A=0$	$A=1$
$B=0$	0	1
$B=1$	1	10

O circuito somará, portanto, dois bits, e o resultado tem 2 bits, em que um é o resultado da soma propriamente dito e o outro é o transporte para o algarismo seguinte.

Se fizermos corresponder aos dígitos na base 2, isto é, 0 e 1, os valores lógicos correspondentes, respectivamente 0 e 1, podemos construir a tabela de verdade das duas funções lógicas que representam o resultado da soma (Tabela 11.2).

O transporte é habitualmente representado por CO, significando "Carry Out" (transporte para o exterior).

Tabela 11.2: Tabela da adição no sistema binário

A B	Soma	Transporte
00	0	0
01	1	0
10	1	0
11	0	1

Podemos realizar a simplificação das duas funções pelo método de Karnaugh, mas da tabela de verdade conseguimos obter imediatamente

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

O circuito pode, então, ser facilmente desenhado (Figura 11.1).

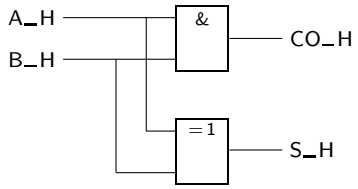


Figura 11.1: Logigrama de um semi-somador

Semi-somador

Este circuito tem o nome de **semi-somador**. A razão para a designação "semi" prende-se com o facto de o circuito não ser capaz de somar os bits dos operandos e ainda o transporte proveniente de uma coluna mais à direita, como é necessário fazer quando somamos dois números binários com n bits, por exemplo em

$$\begin{array}{cccccc} & 3 & 2 & 1 & 0 & \leftarrow \text{Colunas} \\ & 1 & 1 & 0 & 1 & (2) \\ + & & 1 & 0 & 1 & (2) \\ \hline 1 & 0 & 0 & 1 & 0 & (2) \\ & 1 & 1 & 0 & 1 & \end{array}$$

Somador completo

Nesses casos é necessário ir um pouco mais longe. É preciso desenvolver um circuito **somador completo**, capaz de somar os 3 bits que podemos encontrar em cada uma das colunas da soma: os dois bits dos operandos, A e B, e ainda o transporte proveniente da coluna imediatamente à direita, CI (CI tem o significado de "Carry In", ou transporte de entrada). As saídas S e CO do somador completo têm, como tabelas de verdade, as que resultam da adição destes 3 bits, e que se apresentam na Tabela 11.3.

A partir desta tabela podemos gerar os quadros de Karnaugh das funções CO e S e, em seguida, obter as suas expressões mínimas, por exemplo em somas de produtos. Mas porque as tabelas de verdade são muito simples, facilmente

Tabela 11.3: Tabela de verdade de um somador completo

A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

deduzimos directamente as expressões das funções:

$$S = A \oplus B \oplus CI$$
$$CO = A\,B + CI\,(A \oplus B)\,.$$

Com estas expressões podemos agora construir um somador completo à custa de dois semi-somadores, já que elas são, no essencial, extensões das expressões das saídas de um semi-somador (ver atrás). Basta acrescentar uma porta OR no fim, como sugere a Figura 11.2.

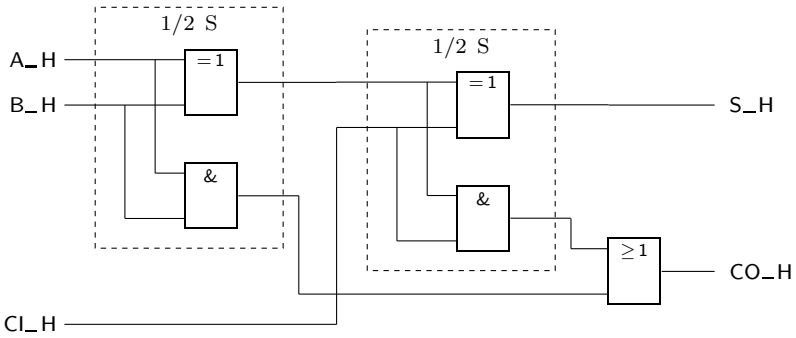


Figura 11.2: Logigrama de um somador completo, formado por dois semi--somadores e uma porta OR

Somador iterativo de n bits

Admitindo a existência de somadores completos, podemos associá-los para realizar **somadores de n bits**, construídos iterativamente a partir de somadores completos. Na Figura 11.3 ilustra-se o diagrama de blocos de um **somador iterativo de 4 bits**.

Repare-se que, nesta figura, o bit mais significativo está colocado à esquerda e o menos significativo à direita, pelo que o fluxo da informação decorre da direita para a esquerda (ao contrário do que é admitido por omissão nos logigramas, diagramas de blocos e esquema eléctricos, daí a inclusão de setas)

Porém, este tipo de representação é mais natural, já que é a habitual quando se fazem adições — ver, por exemplo, a soma anterior, que se reproduz a seguir.

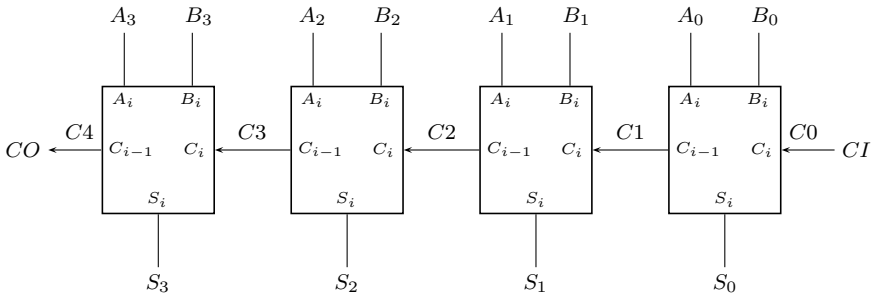
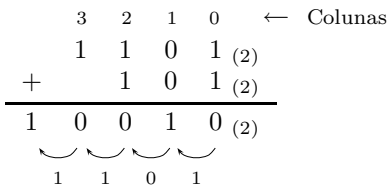


Figura 11.3: Diagrama de blocos de um somador iterativo de 4 bits, formado pela interligação de quatro somadores completos



74x283

Na Figura 11.4 apresenta-se o símbolo IEC de um somador iterativo de 4 bits, o 74x283.

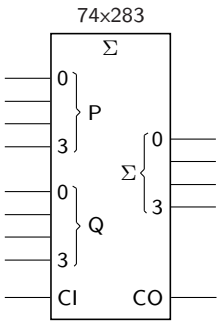


Figura 11.4: Símbolo IEC do 74x283

Qualificador geral Σ

Qualificadores de entrada Pi, Qi e CI

Qualificadores de saída Σ i e CO

De notar o **qualificador geral** Σ que indica um somador. De notar ainda os **qualificadores de entrada** P0 a P3 e Q0 a Q3, e os **qualificadores de saída** Σ0 a Σ3, para além dos qualificadores que identificam os transportes.

11.2 Subtractores Binários

Subtractor completo

Subtractor iterativo de n bits

O problema da subtracção de números binários (sem sinal) pode ser resolvido recorrendo a **subtractores completos** e, a partir destes, formando **subtractores iterativos de n bits**, de forma análoga à que foi utilizada para os adicionadores binários da secção anterior.

Para o fazermos, começamos por apresentar, na Tabela 11.4, a operação de subtracção $A - B$ no sistema binário — admitindo que existe um transporte

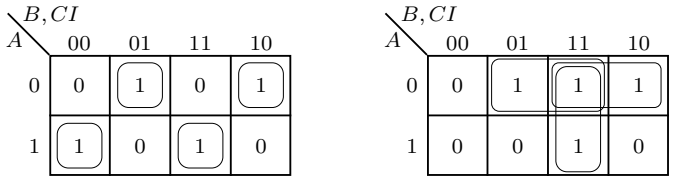
CI da subtracção anterior — o que produz a diferença D e um transporte CO (recordar que A é o **aditivo** e B é o **subtractivo** na operação de subtracção). *Aditivo e subtractivo*

Tabela 11.4: Tabela da subtracção no sistema binário

A	B	CI	D	CO
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Em relação a esta tabela devemos notar como, para cada par de valores nas entradas A e B, se adiciona CI a B e depois se subtrai de A para a obtenção da diferença D e do transporte CO.

Na Figura 11.5 apresentam-se os quadros de Karnaugh destas funções.



$$D = A \oplus B \oplus CI$$
ou
$$\overline{D} = \overline{A} \oplus B \oplus CI$$

$$\begin{aligned} CO &= \overline{A}\,B + \overline{A}{\cdot}CI + B{\cdot}CI \\ &= \overline{A}\,B + CI\,(\overline{A}\,\overline{B} + A\,B) \\ &= \overline{A}\,B + CI\,(\overline{A} \oplus B) \end{aligned}$$

Figura 11.5: Quadros de Karnaugh das funções booleanas simples D e CO

As equações lógicas que se obtêm são, então,

$$\begin{aligned} D &= A \oplus B \oplus CI \\ CO &= \overline{A}\,B + \overline{A}{\cdot}CI + B{\cdot}CI \end{aligned}$$

o que, após manipulação simples, permite obter

$$\begin{aligned} \overline{D} &= \overline{A} \oplus B \oplus CI \\ CO &= \overline{A}\,B + CI\,(\overline{A} \oplus B)\,. \end{aligned}$$

Estas últimas equações sugerem a utilização de dois semi-somadores como os da Figura 11.1 para formar D e CO, como se ilustra na Figura 11.6.

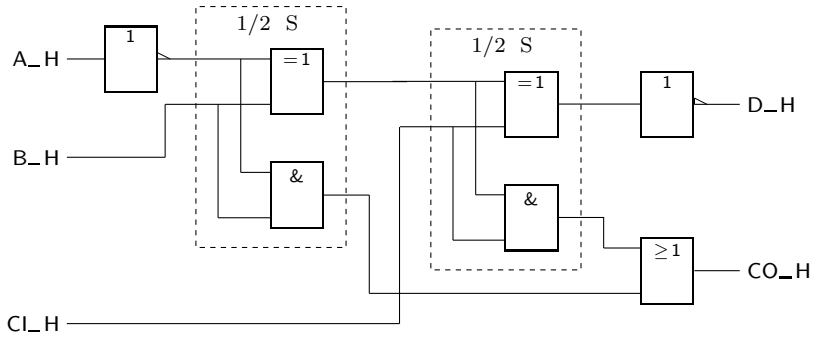


Figura 11.6: Logigrama de um subtractor completo formado por dois semi--somadores e lógica adicional

11.3 Somadores e Subtractores em Complemento para 2

Como sabemos da Subsecção 1.4.2, a soma de dois números representados em notação de complemento para 2 é feita como se de números binários se tratasse.

Ou seja, a soma de números com n bits representados nessa notação usa um somador binário de n bits, e o resultado vem correcto à saída do somador desde que não haja "overflow" na operação.

Por outro lado, como também já sabemos, para detectar um eventual "overflow" na adição em complemento para 2 basta ver se o transporte proveniente do último bit para o exterior do somador é diferente do transporte proveniente do bit anterior e que vai para a posição do bit de sinal, o que pode ser feito simplesmente com uma porta XOR.

Da mesma subsecção, sabemos ainda que a utilização de números representados em notação de complemento para 2 permite realizar subtracções de forma muito simples.

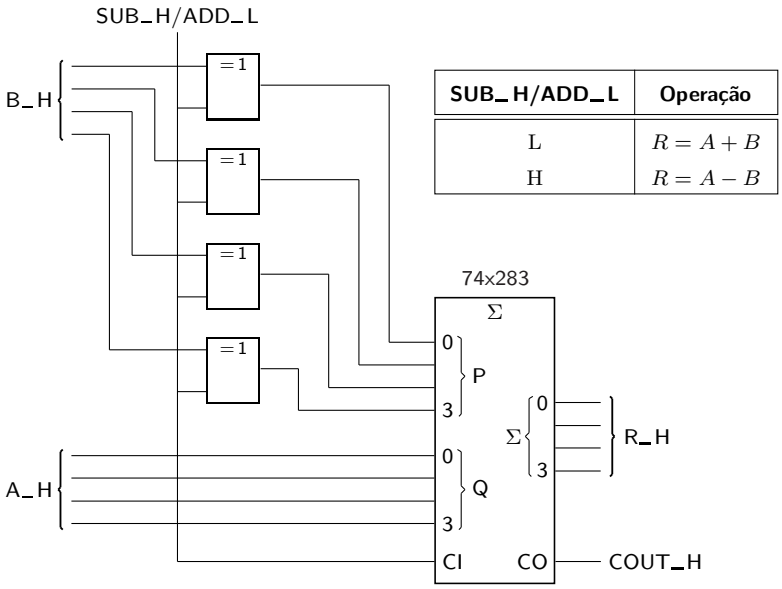
Com efeito, realizar a subtracção $x - y$ é o mesmo que realizar a soma $x + + (-y)$. Por outro lado, trocar o sinal a um número representado em notação de complemento para 2 significa, na prática, obter o complemento para 2 do número. Assim sendo, a subtracção em complemento para 2 pode ser obtida a partir de um somador binário.

Usando as propriedades da função OU-exclusivo a seguir listadas:

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

Somador/subtractor

e ainda a possibilidade de usar a variável CI de um somador para adicionar uma unidade aos dois números, é possível realizar um circuito **somador/subtractor** em complemento para 2 com o logigrama da Figura 11.7.



SUB_H/ADD_L	Operação
L	$R = A + B$
H	$R = A - B$

Figura 11.7: Logigrama de um somador/subtractor em complemento para 2

11.4 Somadores BCD

Como se viu no Capítulo 2, o código BCD é uma forma de representar algarismos decimais. Um número representado em BCD não está representado em base 2. Como é óbvio, tal representação não pretende ter qualquer significado aritmético e, para além disso, não é destinada à realização de operações aritméticas.

No entanto, tal é possível e, por vezes, é interessante. Isso acontece quando temos representações de números em BCD sobre os quais é necessário fazer pequenos cálculos que não justificam, por exemplo, a conversão para binário.

A soma em BCD é feita, naturalmente, dígito decimal a dígito decimal. Por exemplo, se somarmos $3 + 5$ deveremos obter o algarismo 8. Se, para isso usarmos um somador de 4 bits, podemos observar a seguinte operação:

$$\begin{array}{rcccc} & 0 & 0 & 1 & 1 \\ + & 0 & 1 & 0 & 1 \\ \hline & 1 & 0 & 0 & 0 \end{array}$$

As duas parcelas são algarismos BCD, tal como o resultado.

Muitas vezes, porém, o resultado da soma (desde que maior que 9) não pode ser representado por um algarismo BCD. Por exemplo, $7 + 5 = 12$ produz o seguinte resultado:

$$\begin{array}{rcccc} & 0 & 1 & 1 & 1 \\ + & 0 & 1 & 0 & 1 \\ \hline & 1 & 1 & 0 & 0 \end{array}$$

que não é um algarismo BCD.

Em certos casos pode obter-se até um transporte no somador. Por exemplo, se somarmos $8 + 9 = 17$ observa-se:

$$\begin{array}{cccccc} & 1 & 0 & 0 & 0 \\ + & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 \end{array}$$

Repare-se que a sequência de números em BCD que podem ser resultado de uma soma de dois digitos BCD pode ser ilustrada como se indica na Tabela 11.5.

Tabela 11.5: Soma de dois dígitos BCD, interpretada em decimal, em binário e em BCD

Resultado		
em decimal	**em binário**	**em BCD**
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101
16	1 0000	0001 0110
17	1 0001	0001 0111
18	1 0010	0001 1000
19	1 0011	0001 1001

Como estamos a tentar utilizar somadores binários para fazer a soma, o resultado que obtemos é o que corresponde à coluna central da tabela, quando o que queríamos obter seria o da coluna da direita.

Mas repare-se que, a partir do número $10_{(10)}$, o resultado na coluna da direita seria o que se obteria de um somador se somássemos 0110, isto é, $6_{(10)}$, ao resultado da coluna do meio. Por exemplo, se o resultado em binário, à saída do somador, fosse o número $15_{(10)} = 1111_{(2)}$, faríamos

$$\begin{array}{cccccc} & 1 & 1 & 1 & 1 \\ + & 0 & 1 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 & 1 \end{array}$$

Isso quer dizer que é possível corrigir o resultado da soma adicionando $6_{(10)} = = 0110_{(2)}$, de forma a obter-se o resultado certo em BCD, com o dígito correcto (digamos, o dígito das unidades) e ainda um transporte para o algarismo seguinte, interpretado como um segundo dígito BCD (o dígito das dezenas).

A detecção das condições em que se tem de fazer a correcção é fécil de determinar. A correcção tem de ocorrer em dois casos:

1. o resultado é uma configuração superior a $9_{(10)}$; ou

2. existe um transporte da soma.

No primeiro caso, a detecção faz-se por uma função lógica dos bits à saída do somador binário, que facilmente se conclui ser $F = X\,Y + X\,Z$, em que X é o bit de maior peso da soma (de notar que $X\,Y$ detecta uma soma com um resultado igual ou superior a $12_{(10)}$, enquanto que $X\,Z$ detecta um resultado igual a $10_{(10)}$ ou a $11_{(10)}$).

No segundo caso, a detecção faz-se pelo transporte do somador.

O circuito ilustrado na Figura 11.8 mostra como se pode, portanto, fazer um **somador de dois dígitos BCD**, que pode ser ligado em cascata para somar unidades, dezenas, centenas, etc.

Somador de dois dígitos BCD

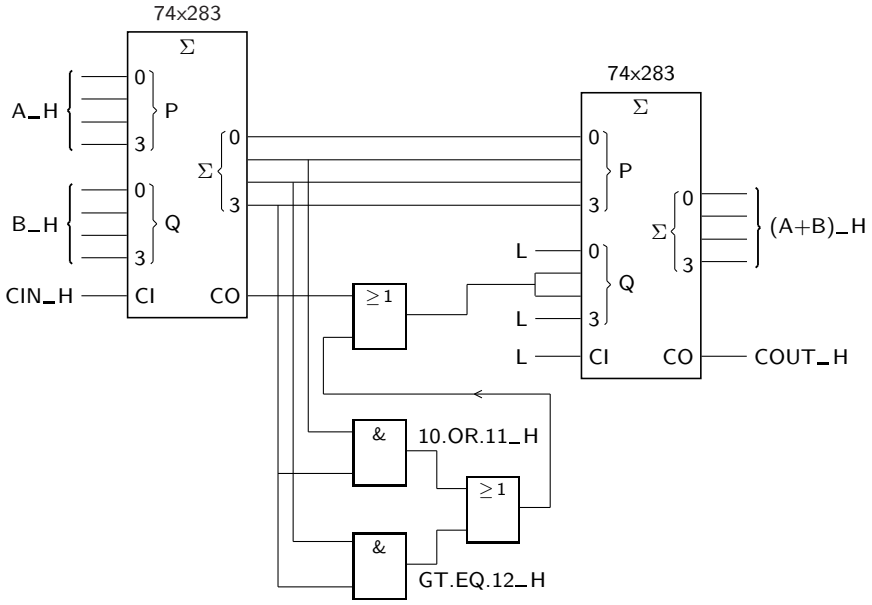


Figura 11.8: Somador de dois dígitos BCD

11.5 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 4.4 e 4.5.

11.6 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

11.1 Provar que um somador completo pode ser construido à custa de dois semi--somadores como os da Figura 11.1, obtendo-se o logigrama da Figura 11.2.

11.2 Utilizar um somador integrado de 4 bits como o 74x283 da Figura 11.4, para implementar um conversor do código BCD para o código Excesso de 3 (para a definição deste último, ver a página 151).

Parte III

CIRCUITOS SEQUENCIAIS

Capítulo 12

Latches

12.1 Latches Simples

Muitas vezes, no projecto de um sistema digital é necessério recorrer a circuitos lógicos cujo comportamento depende não só dos valores nas entradas em cada momento, mas também do comportamento anterior dessas entradas, isto é, a circuitos cujo comportamento é determinado, parcial ou totalmente, pelas entradas que ocorreram no passado. Esses cicuitos designam-se por **circuitos sequenciais**, por oposição aos **circuitos combinatórios** que estudámos nos capítulos anteriores.

Circuito sequencial

Circuito combinatório

Consideremos o circuito da Figura 12.1.

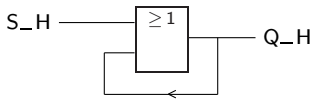


Figura 12.1: Exemplo de circuito sequencial muito simples

Admitamos que os valores iniciais da função Q e da variável de entrada S são iguais a L. Essa situação é estável, como se pode facilmente constatar e, portanto, mantém-se Q_H em L.

Vejamos o que acontece agora se a variável S for activada, isto é, se aplicarmos um nível H no fio S_H. Nesse caso, e como $\mathrm{H} + \mathrm{L} = \mathrm{H}$, o valor da função Q altera-se e fica $Q\_\mathrm{H} = \mathrm{H}$. Como $\mathrm{H} + \mathrm{H} = \mathrm{H}$, a situação $Q\_\mathrm{H} = \mathrm{H}$ mantém-se estável a partir daqui.

Se agora aplicarmos um nível L na linha S_H, a linha Q_H mantém-se activa, isto é, a H.

Este comportamento pode ser descrito sinteticamente pela Tabela 12.1, em que $Q\_\mathrm{H}_{(t)}$ representa o valor, num determinado instante t, da linha de saída do circuito OR, e $Q\_\mathrm{H}_{(t+\Delta t)}$ é o valor da mesma linha um breve intervalo de tempo Δt depois, em que Δt é o tempo de atraso do circuito OR.

Um diagrama temporal ilustrativo do comportamento deste circuito é o que se representa na Figura 12.2.

Tabela 12.1: Tabela de verdade física do circuito sequencial da Figura 12.1

S_H	$Q\_H_{(t)}$	$Q\_H_{(t+\Delta t)}$
L	L	L
L	H	H
H	L	H
H	H	H

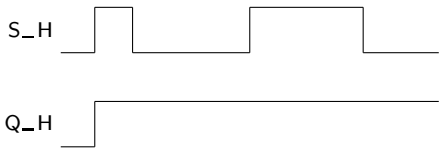


Figura 12.2: Diagrama temporal que ilustra o funcionamento do elemento de memória da Figura 12.1

Memória elementar

Este circuito muito simples constitui, como se pode ver, uma **memória elementar**, capaz de armazenar 1 bit de informação. Com efeito, o nível de tensão em Q_H indica se alguma vez, durante o funcionamento do circuito, a linha S_H foi activada.

O "defeito" do circuito da Figura 12.1 é que, uma vez assumido o nível H na linha de saída, este não pode vir alterado. Assim, como elemento de memória este circuito é pouco interessante, uma vez que só podemos registar nele um tipo de informação (forçar o nível H na linha de saída). Seria mais útil dispormos de uma memória elementar mais completa, ainda capaz de armazenar 1 bit de informação, mas em que conseguíssemos controlar indistintamente os dois níveis L e H na saída (isto é, forçar a linha de saída a L ou a H, conforme quisessemos).

Um circuito que cumpre essa especificação é o da Figura 12.3.

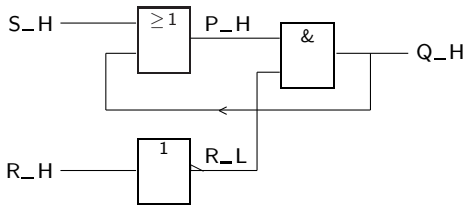


Figura 12.3: Modificação do circuito da Figura 12.1 que permite armazenar um nível L ou um nível H na linha de saída Q_H

Suponhamos que, inicialmente, as linhas Q_H, S_H e R_H estão todas a L. Nesse caso teremos $P\_H = L$ e $R\_L = H$. De onde se conclui que Q_H continua a L e, por consequência, que a situação é estável.

Se, agora, a linha S_H for levada a H, mantendo-se R_H a L, teremos P_H a H e, por conseguinte, Q_H assumirá o valor H.

Da mesma forma que no circuito anterior, se em seguida desactivarmos a variável

S (fazendo $S\_H = L$) a função Q continuará activa ($Q\_H = H$). Até aqui tudo se passa como anteriormente, no circuito da Figura 12.1.

Levemos agora R_H a H. Como R_L fica com o nível L, e independentemente do nível de P_H (que é, por enquanto, H), virá $Q\_H = L$. É claro que, a partir de agora, e enquanto não voltar a ser activada a linha de entrada S_H, a linha de saída Q_H permanecerá a L.

Estamos, pois, na posse de um circuito de memória cujo "conteúdo" pode ser alterado sempre que se queira e que é, por isso, um circuito mais flexível que o anterior.

O comportamento deste circuito pode ser descrito pela Tabela 12.2.

Tabela 12.2: Tabela de verdade física do elemento de memória da Figura 12.3

S_H	R_H	$Q\_H_{(t)}$	$Q\_H_{(t+\Delta t)}$
L	L	L	L
L	L	H	H
L	H	L	L
L	H	H	L
H	L	L	H
H	L	H	H
H	H	L	L
H	H	H	L

Um diagrama temporal ilustrativo do funcionamento desta memória é o da Figura 12.4.

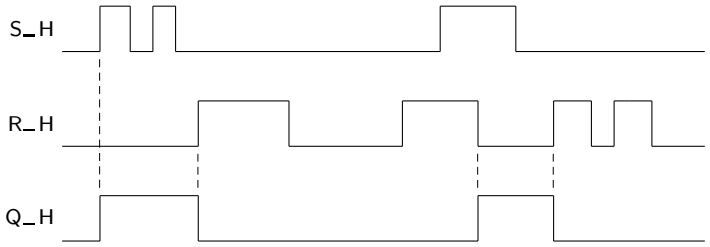


Figura 12.4: Diagrama temporal que ilustra o comportamento do elemento de memória da Figura 12.3

Deve notar-se que, ao contrário do que sucede no diagrama temporal da Figura 6.7, na página 101, o diagrama da Figura 12.4 vem simplificado, não incluindo os tempos de propagação no circuito e admitindo transições instantâneas entre os níveis L e H. Esta é uma situação muito comum, que seguiremos no futuro sempre que não haja possibilidade de confusões devido às simplificações.

Convém, desde já, chamar a atenção para um caso que origina geralmente algumas dificuldades de interpretação. Trata-se da situação em que se encontram activas simultaneamente as linhas S_H e R_H. Repare-se que se está a pedir

ao circuito, de acordo com o que vimos anteriormente, que ele active e desactive simultaneamente a linha de saida Q_H (a ponha simultaneamente a L e a H). Como tais ordens são contraditórias, o circuito não pode cumprir ambas. Dada a estrutura do circuito, o resultado é, como se pode ver na Tabela 12.2, a colocação do valor L na linha Q_H. Esta situação, embora atípica, é possível, não oferecendo quaisquer dificuldades conceptuais e não implicando qualquer actividade não previsível do circuito.

Uma outra situação, que por vezes é confundida com a primeira, é a que a seguir se descreve.

Admita-se que se tem $S\_H = R\_H = H$. Como vimos, virá $Q\_H = L$ neste circuito. Admita-se agora que S_H e R_H transitam simultaneamente para L. Essa simultaneidade não é possível se encararmos escalas de tempo muito finas em que, por exemplo, a diferença entre os tempos de propagação dos sinais nas linhas de entrada seja significativa. De qualquer modo, se a diferença entre uma transição e outra for maior que a ordem de grandeza do tempo de atraso dos circuitos, prevalece, é claro, a combinação intermédia. No caso de ser menor, tudo depende dos atrasos relativos dos circuitos e *o resultado não é previsível*.

Latches
Latch SR

O circuito agora apresentado pertence a uma classe de circuitos que são designados por **latches**. Neste caso concreto trata-se do **latch SR**. Um latch — tal como um *flip-flop*, que estudaremos no capítulo seguinte — pode ser definido como um dispositivo que armazena 1 bit de informação e que pode ser mantido indefinidamente num qualquer de dois estados, comutando (mudando) de um para o outro por activação de determinadas entradas. A diferença entre um latch e um flip-flop será explicada oportunamente, mas podemos adiantar desde já que ela depende das entradas e do modo como as entradas forem actuadas.

Uma análise rudimentar do circuito da Figura 12.5 revela que tem um comportamento exactamente igual ao anterior. Este circuito também se emprega muito. Aliás, tem a vantagem de necessitar apenas de duas portas lógicas e do mesmo tipo.

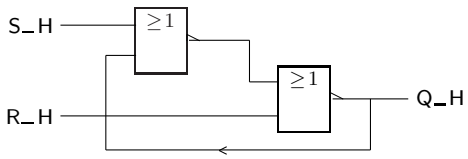


Figura 12.5: Forma alternativa à do latch da Figura 12.3

Uma forma mais usual de desenhar o latch SR é a da Figura 12.6. A saida Q'_H é, normalmente, a negação da saida Q_H excepto na situação particular em que as variáveis de entrada S e R estão simultaneamente activas. Nessa situação teremos $Q\_H = Q'\_H = L$. Esse caso é, porém, de menor interesse, sendo, por isso, frequente que Q'_H seja representada por $\overline{Q}\_H$ ou, o que é o mesmo, por Q_L, como mostra a figura.

Estados de um latch

Os níveis de tensão na saída Q_H definem os **estados** do latch. Dizemos, assim, e de forma simplificada, que o latch se encontra, num determinado momento, no estado $\mathsf{Q} = H$ ou no estado $\mathsf{Q} = L$ (em vez de $Q\_H = H$ ou $Q\_H = L$). Desta forma, a activação da entrada S leva o latch para o estado $\mathsf{Q} = H$, e a activação da entrada R leva o latch para o estado $\mathsf{Q} = L$.

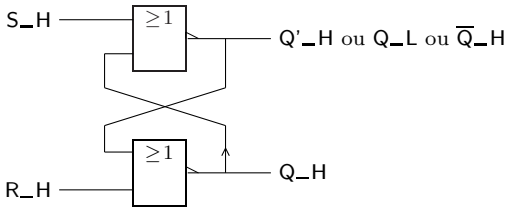


Figura 12.6: Logigrama habitual de um latch SR

Tabela 12.3: Tabela de verdade física do latch SR da Figura 12.6

S_H	R_H	Q_H$_{(t+\Delta t)}$	Q_L$_{(t+\Delta t)}$	Modo
L	L	Q_H$_{(t)}$	Q_L$_{(t)}$	Manutenção
L	H	L	H	Reset
H	L	H	L	Set
H	H	L	L	Força LL

A tabela de verdade física para este circuito é identica à da Tabela 12.2. Na Tabela 12.3 representa-se uma versão abreviada, que tem em conta também a saida Q_L. Notemos que na tabela estão identificadas vários **modos de funcionamento** do latch SR:

Modos de funcionamento de um latch SR

— o **modo de manutenção** do estado do latch, com S e R inactivas, em que o seu estado se mantém e, por conseguinte, se tem Q_H$_{(t+\Delta t)} = Q$_H$_{(t)}$; do mesmo modo, Q_L$_{(t+\Delta t)} = Q$_L$_{(t)}$;

Modo de manutenção

— o **modo de Reset** do latch, com R activa mas S inactiva, em que se faz o Reset da saída Q_H, o que significa que se tem Q_H$_{(t+\Delta t)} =$ L e, por conseguinte, também se tem Q_L$_{(t+\Delta t)} =$ H;

Modo de Reset

— o **modo de Set** do latch, com S activa mas R inactiva, em que se faz o Set da saída Q_H, o que significa que se tem Q_H$_{(t+\Delta t)} =$ H e, por conseguinte, também se tem Q_L$_{(t+\Delta t)} =$ L; e, finalmente,

Modo de Set

— um modo de funcionamento sem nenhuma designação especial, com S e R activas, em que as saída Q_H e Q_L vêem forçadas a L; neste modo de funcionamento, e apenas neste, as saídas não são complementares.

Fazem, por isso, sentido as designações S e R, com o significado de "Set" e "Reset", respectivamente — e que se podem traduzir por "pôr" (ou activar) e "repor" (ou desactivar) a função de saída Q.

A interpretação da Tabela 12.3 deve, então, ser feita nos seguintes moldes:

Funcionamento de um latch SR

1. na primeira linha as variáveis de entrada S e R estão ambas inactivas, ou seja, *não se quer fazer o "Set" nem o "Reset do latch*; nessas condições, as funções de saída Q e $\overline{Q}$ mantêm o seu valor anterior;

2. na segunda linha *faz-se o "Reset" do latch*, já que R está activa (e S está inactiva); então, a função de saída Q vem desactivada (a L) e $\overline{Q}$ vem a H;

3. na terceira linha os papéis das entradas vêm trocados, isto é, agora *faz-se o "Set" do latch*, com S activa (e R inactiva); nessas condições, a função de saída Q vem activada (a H) e $\overline{Q}$ vem a L;

4. finalmente, na quarta linha *tenta fazer-se simultaneamente o "Set" e o "Reset" do latch*; como vimos anteriormente, estas ordens são contraditórias na medida em que o circuito não pode *simultaneamente* activar e desactivar a função Q, e ele responde colocando a L as funções de saída Q e $\overline{Q}$.

Latch $\overline{S}\,\overline{R}$

É também comum utilizar o circuito da Figura 12.7, que é semelhante ao anterior mas com as variáveis de entrada S e R activas no nivel L (por isso, este tipo de latch designa-se habitualmente por latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$).

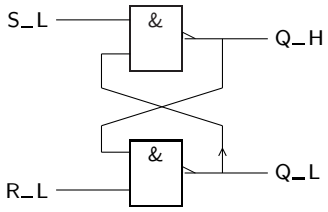


Figura 12.7: Logigrama de um latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$

Modos de funcionamento de um latch $\overline{S}\,\overline{R}$

A tabela de verdade física simplificada para este latch é a da Tabela 12.4, em que podemos constatar os mesmos **modos de funcionamento** do latch SR, excepto pelo facto de os níveis de actividade das entradas serem diferentes nos dois casos.

Tabela 12.4: Tabela de verdade física de um latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$

S_L	R_L	$\mathrm{Q\_H}_{(t+\Delta t)}$	$\mathrm{Q\_L}_{(t+\Delta t)}$	Modo
L	L	H	H	Força HH
L	H	H	L	Set
H	L	L	H	Reset
H	H	$Q\_\mathrm{H}_{(t)}$	$Q\_\mathrm{L}_{(t)}$	Manutenção

As designações $S_$L e $R_$L são, naturalmente, equivalentes a $\overline{S}_$H e $\overline{R}_$H.

Funcionamento de um latch $\overline{S}\,\overline{R}$

A interpretação da Tabela 12.4 deve, então, ser feita nos seguintes moldes:

1. na primeira linha *pretende efectuar-se simultaneamente o "Set" e o "Reset" do latch*, o que o circuito não consegue fazer; a sua resposta consiste em colocar a H as duas funções de saída;

2. na segunda linha *faz-se o "Set" do latch*, já que S está activa e R inactiva; nessas condições, a saída $Q_$H vem activa (a H) e a outra saída a L;

3. na terceira linha *faz-se o "Reset" do latch*, com R activa e S inactiva; então, a saída $Q_$H vem desactivada (a L) e a outra saída vem a H;

4. finalmente, na última linha *não se quer fazer o "Set" nem o "Reset do latch*; nessas condições, as funções de saída Q e $\overline{Q}$ mantêm o seu valor anterior.

Como é óbvio, podem considerar-se latches SR com mais de uma entrada S ou R, como se exemplifica na Figura 12.8 para um latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$.

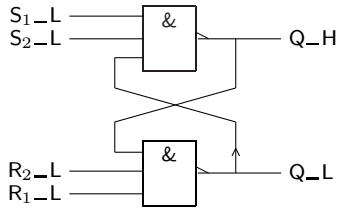


Figura 12.8: Logigrama de um latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$ com duas entradas de "Set" e duas de "Reset"

Neste latch, a activação da saida Q_H é conseguida pela activação de S_1 ou de S_2 (com o nível L), indiferentemente, e a desactivação da saída é conseguida pela activação de R_1 ou de R_2, também indiferentemente.

A representação destes circuitos de acordo com a norma IEC 60617-12 é feita por um simbolo com contorno rectangular, como é habitual, mas sem qualificador geral, situação que é extensiva a todos os elementos bi-estáveis (latches e flip-flops).

Na Figura 12.9 representa-se o símbolo IEC do latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$ da Figura 12.8.

Símbolo do latch $\overline{S}\,\overline{R}$

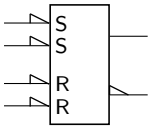


Figura 12.9: Símbolo IEC do latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$ da Figura 12.8

De notar que a norma IEC 60617-12 usa o **qualificador de entrada** S para designar a **dependência Set**, e o **qualificador de entrada** R para designar a **dependência Reset**.

Qualificadores de entrada S e R

Dependências Set (S) e Reset (R)

12.2 Latches Controlados

Os latches até agora estudados respondem a mudanças de níveis nas suas entradas, logo que elas ocorrem. Um latch deste tipo é usualmente designado por **assincrono** porque o seu comportamento não é sincronizado por qualquer sinal.

Latch assíncrono

Mas consideremos o circuito sequencial da Figura 12.10.

Estamos, agora, na presença de um latch SR cujo comportamento depende do nível numa entrada de controlo, EN. Ou seja, temos agora um **latch controlado** do tipo SR, ou **latch SR controlado**, ou **latch SR sincronizado**, ou ainda **latch SRT**.

Latch SR controlado, ou sincronizado ou SRT

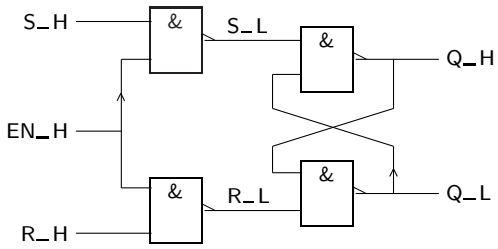


Figura 12.10: Circuito de um latch SR controlado, ou sincronizado

Funcionamento de um latch SR controlado

Como podemos observar, quando a variável de entrada EN está a H as variáveis S e R aparecem complementadas nas linhas S_L e R_L de entrada para o latch $\overline{S}\,\overline{R}$ da direita, e globalmente o circuito comporta-se como um latch SR. Pelo contrário, quando a variável de entrada EN está a L as portas AND da esquerda do logigrama têm pelo menos uma entrada inactiva, e as suas saídas vêm a H, pelo que o latch $\overline{S}\,\overline{R}$ da direita mantém o seu estado.

Ou seja, a linha de entrada EN_H, quando activa, permite que as restantes entradas regulem o funcionamento do latch, e quando inactiva inibe qualquer mudança deste, que mantém o seu estado. Esta entrada tem, portanto, e naturalmente, o nome de "Enable".

Então, a tabela de verdade física que descreve o funcionamento deste latch (Tabela 12.5) é semelhante à que descreve o comportamento de um latch SR assíncrono quando a variável EN está activa. No caso contrário, o circuito não reage às mudancas nas entradas S e R.

Tabela 12.5: Tabela de verdade física de um latch SR controlado

EN_H	S_H	R_H	$Q\_H_{(t+\Delta t)}$	$Q\_L_{(t+\Delta t)}$	Modo
L	×	×	$Q\_H_{(t)}$	$Q\_L_{(t)}$	Manutenção
H	L	L	$Q\_H_{(t)}$	$Q\_L_{(t)}$	Manutenção
H	L	H	L	H	Reset
H	H	L	H	L	Set
H	H	H	H	H	Força HH

No caso de se estabelecer, com EN a L, uma configuração nas entradas S e R que faria o latch mudar de estado se EN estivesse activa, o que se passa é que essa mudanca ocorrerá logo que EN fique activa.

Modos de funcionamento de um latch SR controlado

Podemos, por conseguinte, afirmar que os **modos de funcionamento** deste latch são idênticos ao do latch não controlado quando a entrada de Enable está activa, e que mantém o estado quando ela está inactiva.

É importante chamar a atenção de novo para o que se passa com as variáveis S e R activas. Neste caso, se EN_H estiver a H, as saidas Q_H e Q_L passam a H como se pode observar na última linha da Tabela 12.5. Mas quando EN_H passa a L, mantendo-se as variáveis R e S a H, ocorre à entrada do latch $\overline{S}\,\overline{R}$ simples formado pelos dois ANDs da direita aquilo para que já tinhamos chamado a atenção, isto é, *a passagem "simultânea" de L para H* das suas

duas linhas de entrada, S_L e R_L. Nestas condições é impossível, como se viu, prever a reacção do circuito.

Na Figura 12.11 ilustra-se, com um diagrama temporal, o funcionamento do latch SR controlado.

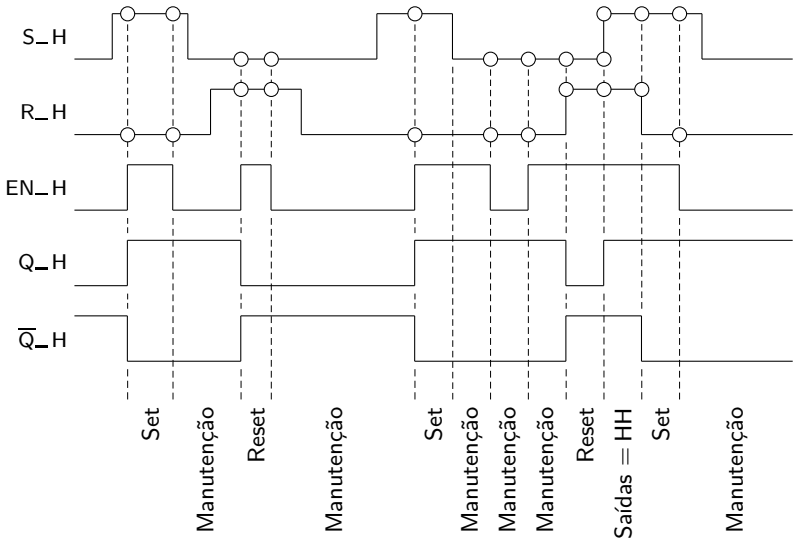


Figura 12.11: Diagrama temporal com o funcionamento de um latch SR controlado, onde se identificam os estados pelos quais o circuito passa

Na Figura 12.12 representa-se o símbolo de um latch SR controlado com entradas activas a H, de acordo com a norma IEC 617-12,

Símbolo de um latch SR controlado

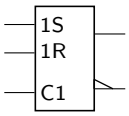


Figura 12.12: Símbolo IEC de um latch SR controlado

O qualificador de entrada C1 é um exemplo de **dependência de Controlo** C, prevista na norma IEC 60617-12, que no essencial afirma que todos os qualificadores de entrada ou de saída com o símbolo m apenas produzem a sua função normal quando Cm está activa.

Qualificadores de entrada C1
Dependência de Controlo (C)

Muitas vezes é útil dispor, nos latches SR, de entradas suplementares que permitam colocá-los num dos seus dois estados possíveis, independentemente dos níveis de tensão nas entradas de Set e de Reset.

Com essas entradas suplementares podemos, então, definir um **estado incial** para o latch, antes do seu funcionamento "normal". Com efeito, não podemos prever o estado de um latch SR — aliás, de qualquer latch ou flip-flop (no capítulo seguinte) — quando inicialmente o ligamos à tensão de alimentação, podendo a saída Q assumir o nível H ou o nível L, e a saída $\overline{Q}$ o seu complemento. Daí a necessidade dessas outras entradas.

Estado inicial

As entradas suplementares, que recebem o nome de **"Preset"** (a que leva o latch

Preset e Clear

Entradas assíncronas ou directas

para o estado $Q\_H = H$) e de **"Clear"** (a que leva o latch para $Q\_H = L$), são **entradas assíncronas** ou **directas**, na medida em que o efeito das suas activações é imediato, não dependendo das entradas S ou R.

Símbolo de um latch SR controlado com Preset e Clear

Na Figura 12.13(a) apresenta-se o logigrama e na parte (b) o símbolo IEC de um latch SR controlado com entradas assíncronas activas a L.

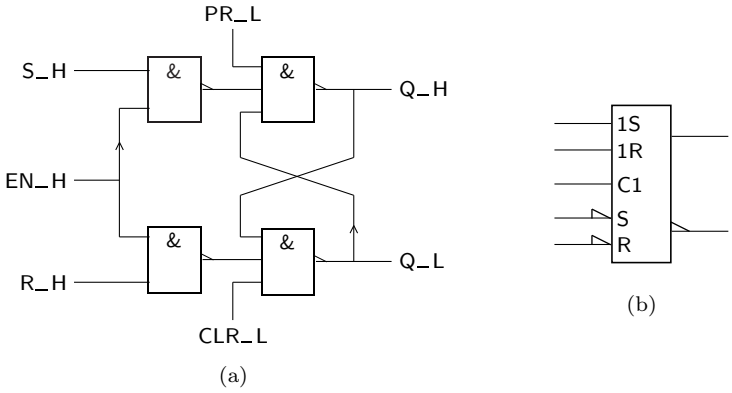


Figura 12.13: (a) Latch SR controlado ao qual se acrescentaram entradas assíncronas de Preset e de Clear activas a L; (b) Símbolo IEC do latch

O funcionamento deste latch é, no essencial, o seguinte: (i) quando as entradas de Preset e de Clear estão inactivas, o latch comporta-se como um latch SR controlado normal; (ii) quando a entrada de Preset está activa e a de Clear inactiva, com a entrada de Enable inactiva, o latch é colocado no estado de Set ($Q\_H = H$); (iii) quando a entrada de Clear está activa e a de Preset inactiva, com a entrada de Enable inactiva, o latch é colocado no estado de Reset ($Q\_H = = L$); e (iv) não devemos activar simultaneamente as entradas de Preset e de Clear (porquê?).

De notar, no símbolo IEC da Figura 12.13(b), a designação S e R dada aos qualificadores das entradas assíncronas, que se referem, respectivamente, às entradas Preset e Clear. Esses qualificadores denotam os comportamentos assíncronos dessas entradas, assim se distinguindo das entradas com qualificadores 1S e 1R que dependem do qualificador de entrada C1 (entrada de Controlo), e que correspondem às entradas de Set e de Reset normais deste latch.

Os latches SR controlados são circuitos largamente usados, mas não constituem os únicos tipos de latch existentes. Um outro tipo é um latch com apenas uma entrada sincronizada e que memoriza o nível de tensão presente nessa entrada em dado momento.

Suponhamos que, num latch SR controlado, obrigamos a entrada R a ser igual à negação da entrada S. Neste caso, na tabela de verdade física do latch SR restariam apenas três linhas, como se sugere na Tabela 12.6.

Tudo se passa como se a função de saída Q seguisse o nível de tensão na variável de entrada S. É claro que a versão assíncrona deste latch tem pouco interesse prático. Com efeito, o valor de Q será, nesse caso, sempre igual ao de S e, portanto, o efeito de memória do circuito não existe. Tal não é o caso na versão de latch controlado que acabámos de estudar, uma vez que aí a função de memória é garantida quando a linha EN_H está inactiva.

Tabela 12.6: Tabela de verdade física do latch SR controlado, modificada por forma a obrigar a entrada R a ser sempre igual à negação da entrada S

EN_H	S_H	R_H	Q_H$_{(t+\Delta t)}$	Q_L$_{(t+\Delta t)}$
L	×	×	Q_H$_{(t)}$	Q_L$_{(t)}$
H	L	H	L	H
H	H	L	H	L

O circuito modificado está representado na Figura 12.14.

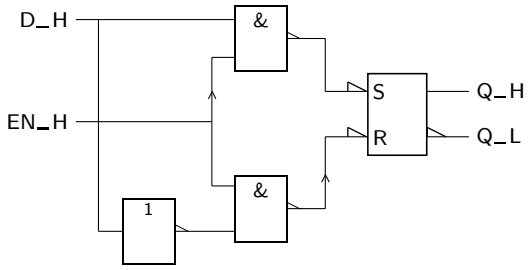


Figura 12.14: Logigrama de um latch D controlado

Note-se que modificámos a designação da entrada S, substituindo-a por D. Esta designação provém da palavra inglesa "Data" (dados), uma vez que este circuito é muito usado para memorizar dados. Este tipo de latch é conhecido por **latch D controlado**, e tem o mesmo comportamento em relação à entrada de Enable que o latch SR controlado.

Latch D controlado

Uma arquitectura alternativa para o latch D controlado vem representada na Figura 12.15, e é fácil a constatação de que o funcionamento do circuito não se altera.

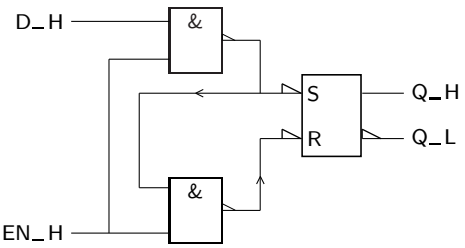


Figura 12.15: Logigrama alternativo para um latch D controlado

Neste tipo de latch, como vimos, quando a variável EN está activa a função Q reproduz a variável de entrada D. É usual designar este tipo de funcionamento por **funcionamento transparente**.

Funcionamento transparente

A tabela de verdade física, será, portanto, a que se apresenta na Tabela 12.7, admitindo que as variáveis D e EN são activas a H.

Tabela 12.7: Tabela de verdade física de um latch D controlado

EN_H	D_H	Q_H$_{(t+\Delta t)}$	Q_L$_{(t+\Delta t)}$	Modo
L	×	Q_H$_{(t)}$	Q_L$_{(t)}$	Manutenção
H	L	L	H	Copia
H	H	H	L	Copia

Modos de funcionamento

Desta tabela podemos deduzir os seguintes **modos de funcionamento** para este latch:

Modo de manutenção

— o **modo de manutenção do estado** do latch, com a entrada de Enable inactiva, em que o seu estado se mantém e, por conseguinte, se tem Q_H$_{(t+\Delta t)} =$ $= Q$_H$_{(t)}$; do mesmo modo, Q_L$_{(t+\Delta t)} = Q$_L$_{(t)}$; e

Modo de cópia

— o **modo de cópia**, com Enable activa, em que vem copiado para a saída Q_H o nível de tensão que se encontrar aplicado à entrada D_H.

Símbolo do latch D controlado

A representação do latch D controlado, de acordo com a norma IEC 60617-12, está ilustrada na Figura 12.16.

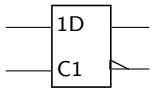


Figura 12.16: Símbolo IEC para um latch D controlado

Qualificador de entrada D

Entrada de dados

O **qualificador de entrada** D siginifica, na norma IEC 60617-12, uma **entrada de dados**.

12.3 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secção 5.2.

12.4 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

12.1 Porque é que o circuito da Figura 12.1, na página 199, é sequencial? Como é que ele se diferencia de um circuito combinatório?

12.2 Admitindo que o circuito da Figura 12.1, na página 199, é sequencial, em que medida é que o seu comportamento depende não só dos valores nas entradas em cada momento, mas também do comportamento passado

dessas mesmas entradas, como resulta da definição de circuito sequencial dada no início deste capítulo? E quão "longe no passado" é que temos que procurar para assegurar o comportamento "presente" do circuito?

12.3 Para cada um dos circuitos representados na Figura 12.17, desenhe um diagrama temporal semelhante ao da Figura 12.2 e responda às seguintes questões:

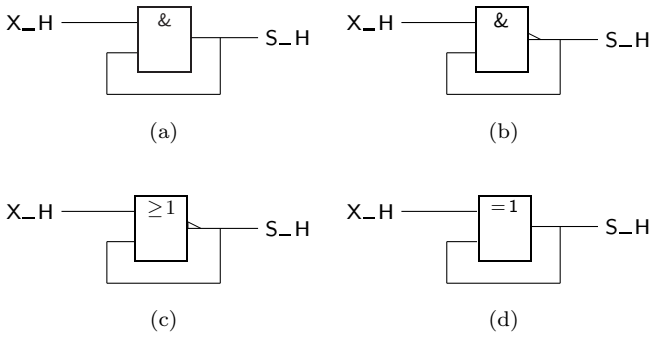


Figura 12.17: Exemplos de 4 circuitos

a) trata-se de um circuito combinatório ou sequencial?
b) poderá servir como elemento de memória elementar?

12.4 Justifique o andamento do diagrama temporal da Figura 12.4, na página 201.

12.5 Para cada um dos circuitos representados na Figura 12.17, responda, justificando, se se trata de um latch.

12.6 Certifique-se que o circuito da Figura 12.5, na página 202, possui o mesmo comportamento do da Figura 12.3, na página 200.

(∗) 12.7 Na Tabela 12.3 da página 203, identificar com setas adequadas a transição ou transições entre linhas da tabela que correspondem à situação em que se tem $S\_H = R\_H = H$, com $Q\_H = L$ e $Q\_L = L$, e em que se provoca a transição simultânea para L de S_H e R_H, admitindo que os tempos de propagação das portas são diferentes. O que podemos deduzir em relação ao estado final do latch?

(∗) 12.8 Completar o diagrama temporal da Figura 12.18, representativo do funcionamento de um latch SR em determminadas condições de níveis de tensão nas entradas, admitindo que incialmente $Q\_H = H$ e $Q\_L = L$, e que o tempo de propagação das duas portas é de 10 ns. Como se comporta o latch nestas condições?

12.9 Complete o diagrama temporal da Figura 12.19, que representa o funcionamento de um latch $\overline{S}\,\overline{R}$ em determminadas condições de níveis de tensão nas entradas. Identifique ao longo do diagrama cada um dos estados por que passa o latch, de forma semelhante à que se apresenta na Figura 12.11, na página 207.

12.10 Desenhe o símbolo IEC dos latches das Figuras 12.6, na página 203, e 12.7, na página 204.

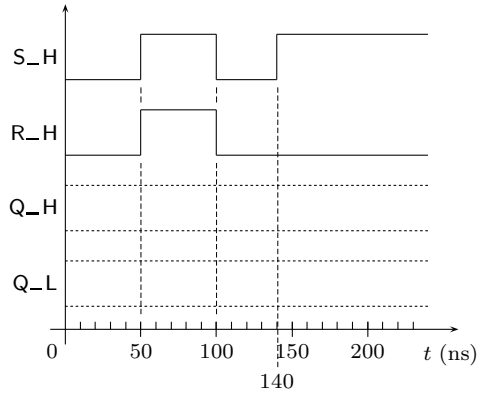


Figura 12.18: Diagrama temporal incompleto que descreve o comportamento de um latch SR em determinadas condições de níveis de tensão nas entradas

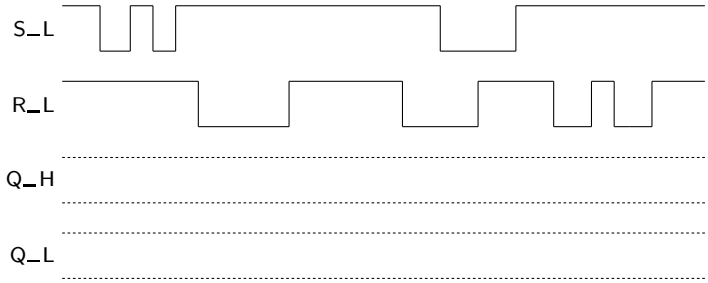
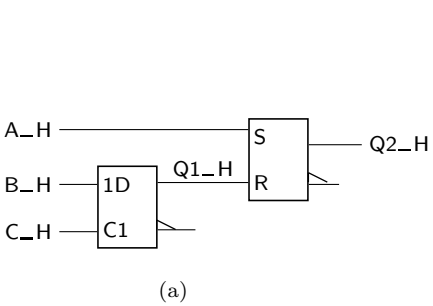
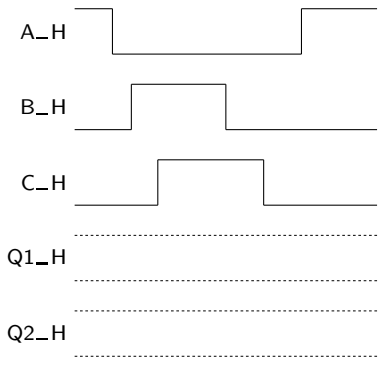


Figura 12.19: Diagrama temporal incompleto que descreve o comportamento de um latch $\overline{\mathrm{S}}\,\overline{\mathrm{R}}$ em determinadas condições de níveis de tensão nas entradas

12.11 Para o circuito da Figura 12.20(a), complete o correspondente diagrama temporal da Figura 12.20(b). Desenhe ainda possíveis logigramas para o latch SR.



(a)



(b)

Figura 12.20: (a) Circuito do Exercício 12.11; (b) diagrama temporal incompleto para o circuito

(∗) 12.12 Considere o latch SR controlado da Figura 12.13, na página 208, com entradas assíncronas de Preset e de Clear activas a L. Mostre que o comportamento deste latch é o que se explicou no texto que acompanha a figura.

(∗) 12.13 Considere a modificação da Figura 12.21, que transforma um latch SR controlado num **latch JK controlado**. A ideia por detrás desta modificação é tentar obviar aos problemas levantados pelos latches SR controlados, apontados na página 206 e nos Exercícios 12.7 e 12.8 (dificuldade em prever o estado final do latch ou o latch entrar em oscilação quando as entradas estão todas a H e se muda a entrada de Enable de H para L). Será que o latch JK controlado resolve esses problemas? Analise o seu funcionamento.

Latch JK controlado

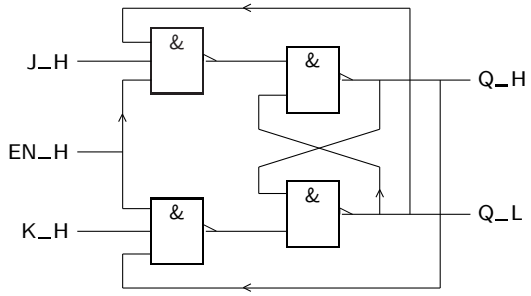


Figura 12.21: Latch JK controlado, obtido por modificação de um latch SR controlado

12.14 O logigrama e o símbolo IEC da Figura 12.22 representam um latch SR com preponderância do Set. Analise o funcionamento do circuito explicando o que se entende por "preponderância do Set", e estabeleça uma tabela de verdade física para o latch. Como poderia modificar o logigrama por forma a obter um latch SR com preponderância do Reset? E qual será o seu símbolo IEC?

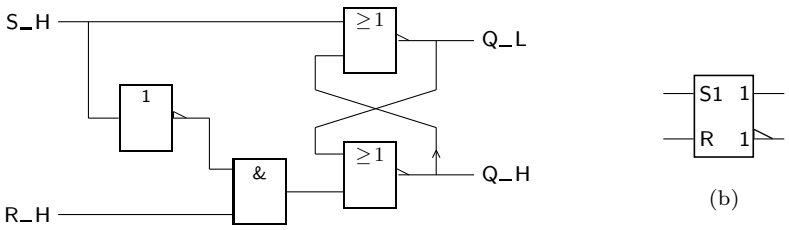
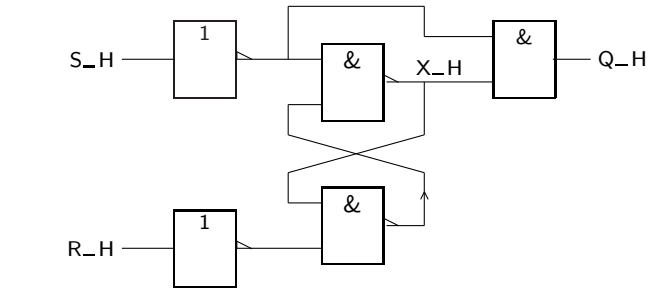


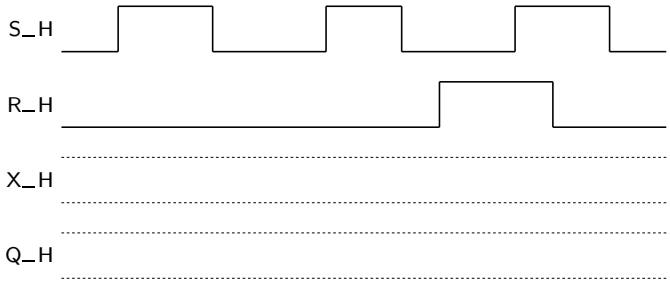
Figura 12.22: (a) Logigrama de um latch SR com preponderância do Set; (b) símbolo IEC correspondente

12.15 Desenhe um circuito que, quando actuado por um botão, muda de estado de cada vez que o botão é sucessivamente activado e desactivado.

12.16 Considere o circuito da Figura 12.23, que corresponde a um latch SR modificado, e complete o diagrama temporal nele incluído.



(a)

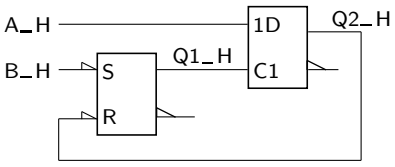


(b)

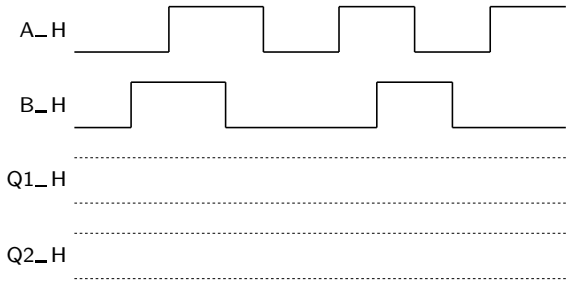
Figura 12.23: (a) Logigrama do latch SR modificado do Exercício 12.16; e (b) diagrama temporal a completar

12.17 Dado o circuito da Figura 12.24(a), complete o correspondente diagrama temporal, na Figura 12.24(b).

12.18 O fabricante de um latch SR controlado como o da Figura 12.10, na página 206, garante que os tempos de propagação t_{pHL} nas saídas Q_H e Q_L são maiores do que os tempos t_{pLH} correspondentes. Construa um diagrama temporal que prove ou não que o fabricante tem razão.



(a)



(b)

Figura 12.24: (a) Circuito do Exercício 12.17; e (b) diagrama temporal a completar

Capítulo 13

Flip-flops

13.1 Flip-flops Master-slave

Em muitas aplicações é inconveniente o uso de elementos de memória com funcionamento transparente. Seria interessante dispor de elementos em que a mudança de estado, a ocorrer, se dê num momento bem determinado e sob o controlo do utilizador, independentemente do momento em que as entradas que provocam a alteração se estabelecem.

A todos os elementos de memória em que não existe transparência e em que as eventuais mudanças de estado ocorrem em flancos de uma linha de sincronização, chamaremos **flip-flops**.

Flip-flop

A linha de sincronização é usualmente designada por **entrada de relógio**, à qual são aplicados **impulsos de relógio**.

Entrada de relógio

Impulsos de relógio

Convém referir aqui que, noutros textos, se chamam indistintamente flip-flops aos elementos que aqui designamos por flip-flops e aos latches. Em qualquer caso, trata-se de uma convenção e nada de fundamental está em jogo nesta contradição.

Consideremos, então, o flip-flop com o circuito da Figura 13.1.

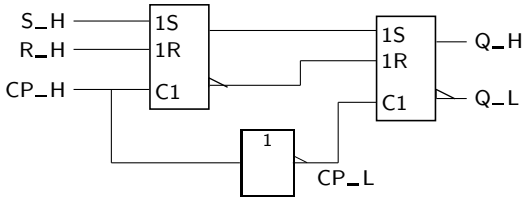


Figura 13.1: Logigrama de um flip-flop SR master-slave que comuta nos flancos descendentes

Este circuito pode ser redesenhado para mostrar a estrutura completa com portas lógicas, como mostra a Figura 13.2.

Funcionamento de um flip-flop SR master-slave

O funcionamento do circuito pode ser seguido com alguma facilidade. Assim, com a variável de entrada CP desactivada, o latch de entrada está num estado

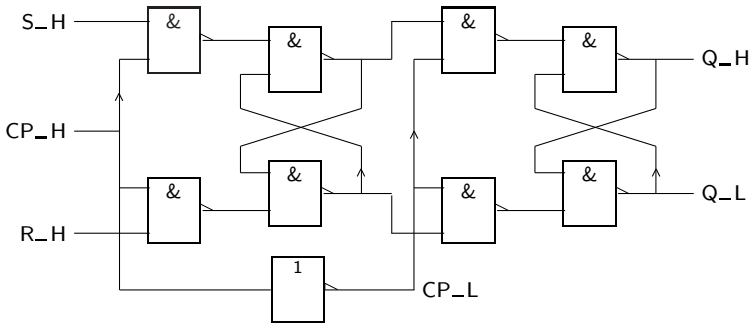


Figura 13.2: Logigrama de um flip-flop SR master-slave que comuta nos flancos descendentes, em que se mostram todas as portas lógicas envolvidas

qualquer e mantém o segundo latch no mesmo estado, uma vez que CP_L está ao nível H e permite que as suas entradas o controlem.

Quando CP_H passa a H, o segundo latch fica isolado, mantendo o estado anterior, uma vez que CP_L está no nível L. O primeiro latch, pelo contrário, está agora aberto à influência das entradas e assume o estado que estas lhe impõem.

Quando CP_H volta a L, temos o primeiro latch de novo isolado das entradas e ligado ao segundo, influenciando-o e fazendo-o assumir o estado que as entradas lhe impuseram.

Portanto, globalmente, o flip-flop fica sensível aos níveis nas entradas durante o tempo em que a variável CP está activa, mas as suas saídas não reagem a eventuais condições nas linhas S_H e R_H que provoquem mudança. A reacção das saídas surge apenas após a passagem da variável CP de activa para inactiva, isto é, no **flanco descendente** da linha CP_H (o outro flanco é designado por **flanco ascendente**).

Flanco descendente e ascendente

O tipo de arquitectura que acabámos de descrever para o flip-flop das Figuras 13.1 e 13.2, e que assenta na existência de um latch de entrada e noutro de saída, denomina-se **master-slave**, sendo o primeiro latch o "master" e o segundo o "slave". O flip-flop em questão designa-se, então, e de forma simplificada, por **flip-flop SR master-slave**.

Flip-flop master-slave

Flip-flop SR master-slave

Por outro lado, este tipo de flip-flop em particular **comuta (ou muda de estado) nos flancos descendentes** dos impulsos de relógio. Diz-se, então, que os flancos descendentes dos sinais de relógio são os **flancos de comutação** do flip-flop.

Comutação nos flancos descendentes

Flanco de comutação

Naturalmente, existem outros flip-flops que comutam nos flancos ascendentes dos impulsos de relógio.

A Tabela 13.1 descreve o funcionamento de um flip-flop SR master-slave pela sua **tabela de verdade física**.

Tabela de verdade física

Desta tabela podemos deduzir os seguintes **modos de funcionamento** para este flip-flop:

Modos de funcionamento

— o **modo de manutenção** do estado do flip-flop, quando não existe flanco de comutação na entrada de relógio ou, existindo flanco, as entradas S e R

Modo de manutenção

Tabela 13.1: Tabela de verdade física de um flip-flop SR master-slave que comuta nos flancos descendentes

$S\_H_{(t)}$	$R\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$	Modo
L	L	⊓	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
L	H	⊓	L	H	Reset
H	L	⊓	H	L	Set
H	H	⊓	?	?	—
×	×	∫	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção

estão inactivas, em que o seu estado se mantém e, por conseguinte, se tem $Q\_H_{(t+1)} = Q\_H_{(t)}$; do mesmo modo, $Q\_L_{(t+1)} = = Q\_L_{(t)}$;

A designação $Q\_H_{(t+1)}$ identifica o nível de tensão na saída Q_H de um flip-flop no instante $t+1$ que se segue a um flanco de comutação (comparar com a designação $Q\_H_{(t+\Delta t)}$ utilizada nos latches para identificar o nível de tensão na saída, decorrido que é o seu tempo de propagação)

— o **modo de Reset** do flip-flop, quando existe flanco de comutação e R está activo e S inactivo, em que se faz o Reset da saída Q_H, o que tem como consequência que $Q\_H_{(t+1)} = L$ e, por conseguinte, $Q\_L_{(t+1)} = H$;

Modo de Reset

— o **modo de Set** do flip-flop, quando existe flanco de comutação e S está activo e R inactivo, em que se faz o Set da saída Q_H, o que tem como consequência que $Q\_H_{(t+1)} = H$ e, por conseguinte, $Q\_L_{(t+1)} = L$; e

Modo de Set

— um modo de funcionamento sem nenhuma designação especial, quando existe flanco de comutação mas S e R estão ambas activas, em que não se pode prever o estado final do flip-flop; neste modo de funcionamento não se garante que as saídas sejam complementares.

O facto de a reacção do flip-flop, no caso descrito na quarta linha da tabela, não ser previsível é resultante de uma situação semelhante, já analisada para o latch SR controlado.

De facto, nas circunstâncias que imediatamente antecedem as que são referidas nessa linha da tabela, existe actividade simultânea das variáveis S e R e um nível H no impulso aplicado à linha CP_H. Quando a linha CP_H vem a L, o latch "master" tem um comportamento que depende dos tempos de propagação das suas portas de entrada, ou seja, um comportamento que não podemos prever. E sendo o seu comportamento imprevisível, o do "slave" também o é quando aparece o flanco de comutação. Logo, a reacção global do flip-flop é imprevisível.

Este facto é suficientemente desagradável para que se tenha procurado obviá-lo. Nessas circunstâncias surgiu um novo tipo de flip-flop, o **flip-flop JK master-slave**, com um comportamento semelhante ao do flip-flop SR master-slave mas em que se impõe que, no caso das duas entradas estarem activas, e perante um flanco de comutação num impulso de relógio, o flip-flop altere o seu estado. Por outras palavras, redefiniu-se a correspondente linha da Tabela 13.1.

Flip-flop JK master-slave

O funcionamento de um flip-flop JK master-slave está, assim, descrito pela **tabela de verdade física** da Tabela 13.2, de onde se destacam os mesmo modos

Tabela de verdade física

Tabela 13.2: Tabela de verdade física de um flip-flop JK master-slave que comuta nos flancos descendentes

J_H$_{(t)}$	K_H$_{(t)}$	CP_H	Q_H$_{(t+1)}$	$\overline{Q}$_H$_{(t+1)}$	Modo
L	L	⎍	Q_H$_{(t)}$	$\overline{Q}$_H$_{(t)}$	Manutenção
L	H	⎍	L	H	Reset
H	L	⎍	H	L	Set
H	H	⎍	$\overline{Q}$_H$_{(t)}$	Q_H$_{(t)}$	Comutação
×	×	⎌	Q_H$_{(t)}$	$\overline{Q}$_H$_{(t)}$	Manutenção
×	×	L	Q_H$_{(t)}$	$\overline{Q}$_H$_{(t)}$	Manutenção
×	×	H	Q_H$_{(t)}$	$\overline{Q}$_H$_{(t)}$	Manutenção

Modo de comutação

de funcionamento do flip-flop SR master-slave, e ainda um **modo de comutação** em que os níveis de tensão nas saídas mudam (comutam).

Na Figura 13.3 ilustra-se a arquitectura de um flip-flop JK master-slave.

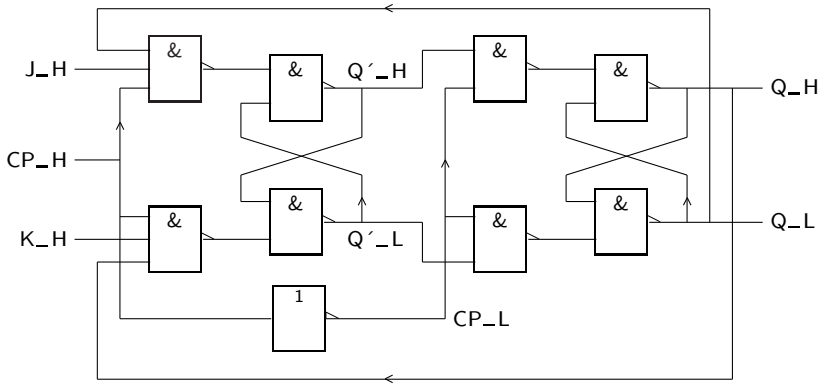


Figura 13.3: Logigrama de um flip-flop JK master-slave que comuta nos flancos descendentes

É necessário, desde já, chamar a atenção para certas particularidades que o funcionamento deste circuito apresenta.

Funcionamento de um flip-flop JK master-slave

Suponhamos, assim, que Q_H está no nível H e que CP_H passa de L para H. Se as variáveis J e K estiverem inactivas, nada se altera.

Mas analise-se, agora, o que se passa quando a variável de entrada K passa a estar activa (K_H = H), mantendo-se CP activa. Então, o primeiro latch muda de estado, isto é, Q'_H, que estava ao nível H, passa, por acção de K, ao nível L. Repare-se, agora, que futuras alterações nos níveis de tensão aplicados às variáveis J e K já não se repercutirão no funcionamento do latch. De facto, se K deixar de estar activa, as entradas do master deixarão também de estar activas, o que, de qualquer forma, não altera o novo nível à saída do latch.

Mais ainda, repare-se que, como a função de saída Q permanece activa, a linha Q_L está ao nível L, *o que bloqueia o AND onde entra a variável J, mantendo a entrada superior do "master" inactiva*. Assim, mesmo que a variável J seja

activada, isso não se reflectirá numa mudança de estado do "master". Portanto, criou-se uma situação em que o flip-flop mudará irremediavelmente de estado quando CP_H voltar ao nível L.

Esta análise mostra, assim, que:

a) com CP_H ao nível L, o primeiro latch não vem influenciado pelas entradas;

b) com CP_H ao nível H, o primeiro latch fica receptivo às entradas e o segundo isolado do primeiro. Se, durante esse intervalo de tempo, alguma vez as entradas forem tais que motivem a alteração de estado do latch master, mesmo que tal configuração não seja a última antes de CP_H passar a L, o flip-flop mudará de estado logo que CP_H volte a L;

c) quando CP_H passa de H a L, isola-se novamente o master das entradas, enquanto o slave copia o estado do master; isto dá-se sempre neste tipo de flip-flops e é, por vezes, fonte de erros de utilização quando não levado em conta. Recomenda-se, por isso, muita atenção a esta característica.

"One's catching"

Este tipo de comportamento tem o nome de **"one's catching"**, que se justifica pelo facto de poder ser usado como detector de "1"s. A título de exemplo mostra-se, na Figura 13.4, um diagrama temporal com uma evolução possível para as formas de onda nas entradas de um flip-flop JK master-slave. De notar que o impulso provocado em K_H provoca o Reset do master, Reset este que se repercute nas saídas do flip-flop assim que aparece o próximo flanco descendente em CP. E mais impulsos que ocorressem em K_H com a entrada CP activa apenas serviriam para confirmar o estado de Reset do master e do flip-flop.

Símbolo do flip-flop JK master-slave

Os flip-flops deste tipo tem o símbolo IEC da Figura 13.5.

Qualificadores de entrada 1J, 1K e C1

Dependência de Controlo (C)

Aqui, o 1 que antecede o J e o K e que se segue ao C nos **qualificadores de entrada** 1J, 1K e C1, indica que existe um efeito de "disparo" pela entrada C1 (**dependência de Controlo**, C), que condiciona a acção das entradas 1J e 1K (os níveis nestas entradas apenas são levados em consideração pelo flip-flop enquanto C1 estiver activada, a H).

Símbolo de atraso

Qualificador de saída ¬

Por outro lado, como sabemos já, a acção destas duas entradas apenas tem efeito nas saídas do flip-flop depois de o "master" ter transferido para o "slave" essa acção (manter o estado, fazer o Reset da saída Q, etc.), isto é, nos flancos descendentes dos impulsos de relógio. A indicação deste modo de funcionamento vem dada pelo **símbolo de atraso** colocado junto às saídas (**qualificador de saída** ¬).

No caso do flip-flop da Figura 13.5, a entrada de relógio vem activada (a H) depois de ocorrer um flanco ascendente num impulso de relógio, pelo que no flanco seguinte, o flanco descendente, as saídas reflectem o funcionamento que resulta dos níveis aplicados às entradas J e K enquanto C1 está activada. O comportamento "one's catching" não é, em si, representado no símbolo.

Preset e Clear

Muitas vezes é útil dispor nos flip-flops, tal como nos latches, de entradas suplementares que permitam colocá-los num dos seus dois estados possíveis, independentemente dos níveis de tensão nas entradas sincronizadas por CP. Essas entradas suplementares recebem, como vimos, o nome de **"Preset"** (a que leva o flip-flop para o estado $Q\_H = H$) e de **"Clear"** (a que leva o dispositivo para

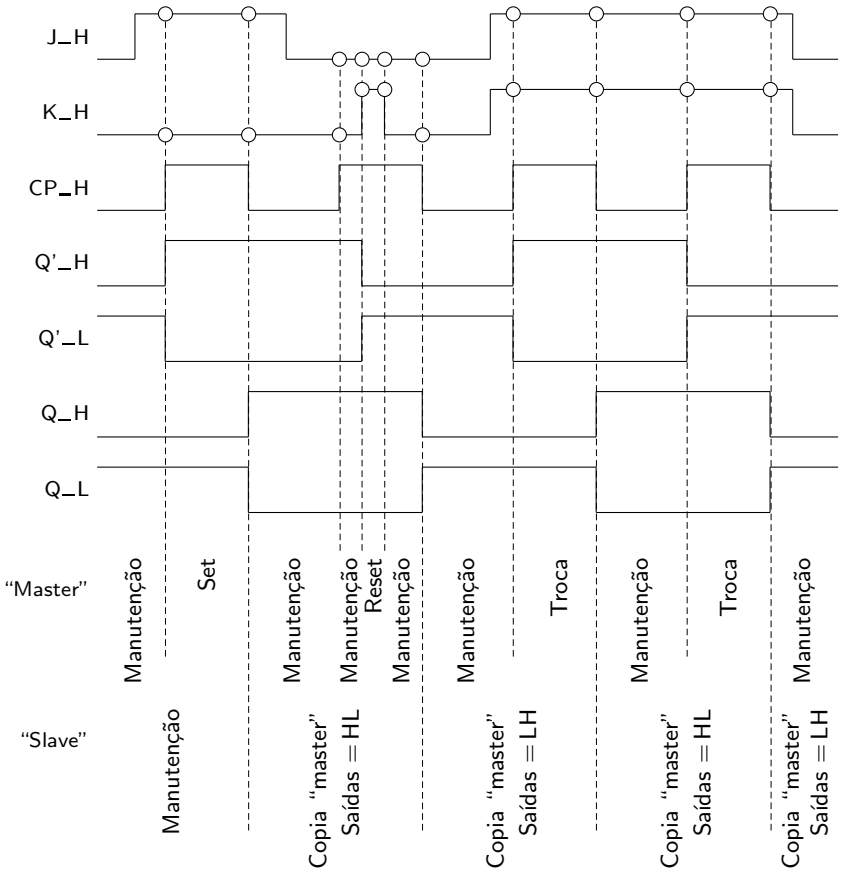


Figura 13.4: Diagrama temporal com o comportamento de um flip-flop JK master-slave que ilustra o fenómeno de "one's catching"

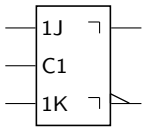


Figura 13.5: Símbolo IEC de um flip-flop JK master-slave que comuta nos flancos descendentes dos impulsos de relógio

Entradas assíncronas ou directas

Entradas síncronas

$Q\_H = L$), e são designadas por **entradas assíncronas** ou **directas**. Neste sentido, essas entradas distinguem-se das outras, aquelas que estudámos anteriormente, que se designam por **entradas síncronas**.

Entradas assíncronas de Set (S) e de Reset (R)

Na Figura 13.6 ilustra-se o logigrama de um flip-flop JK master-slave com **entradas assíncronas de Set (Preset) e de Reset (Clear)** activas a L.

Qualificadores de entrada S e R

E na Figura 13.7 encontra-se o simbolo IEC desse flip-flop. É de notar que os qualificadores de entrada R e S, que se referem, respectivamente, às entradas Clear e Preset, denotam os seus comportamentos assíncronos, não dependentes do qualificador de entrada C1.

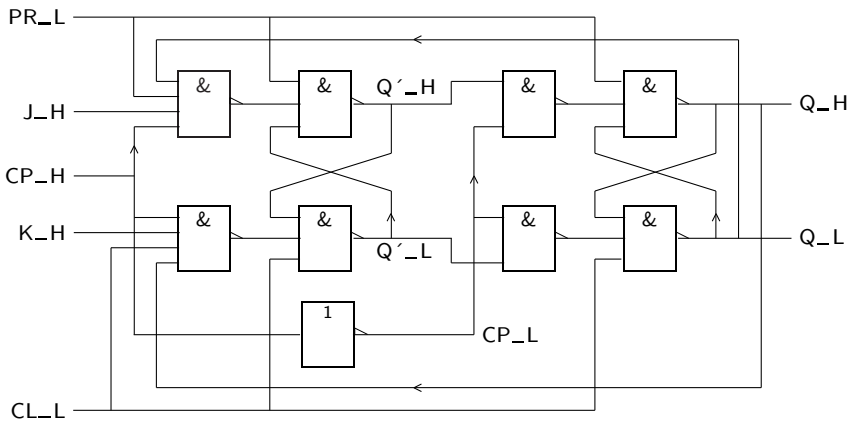


Figura 13.6: Logigrama de um flip-flop JK master-slave com entradas assíncronas de Preset e de Clear activas a L e que comuta nos flancos descendentes dos impulsos de relógio

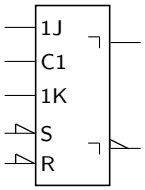


Figura 13.7: Símbolo IEC de um flip-flop JK master-slave com entradas assíncronas de Preset e de Clear activas a L e que comuta nos flancos descendentes dos impulsos de relógio

Flip-flop D master-slave

Da mesma forma que aconteceu anteriormente, é possível definir um **flip-flop D master-slave**, com uma entrada síncrona D. Basta considerar um flip-flop SR master-slave em que a entrada R é feita sempre igual à negação da entrada S, como já foi feito no caso dos latches.

Símbolo do flip-flop D master-slave

O símbolo IEC de um tal flip-flop é o que se representa na Figura 13.8. Notemos que este flip-flop comuta nos flancos ascendentes. o que se traduz na inclusão de um triângulo na entrada de relógio, orientado no sentido do fluxo dos sinais.

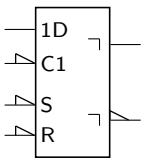


Figura 13.8: Símbolo IEC de um flip-flop D master-slave com entradas assíncronas de Preset e de Clear activas a L e que comuta nos flancos ascendentes dos impulsos de relógio

A inclusão deste triângulo tem o seguinte significado: quando a entrada de relógio está inactiva (a H), os níveis de tensão na entrada síncrona, D, não afectam o funcionamento do flip-flop; quando a entrada de relógio passa a ficar

activa (porque foi gerado um flanco descendente que a levou ao nível L), o master fica receptivo aos níveis na entrada síncrona; e aquando do flanco seguinte (ascendente) os níveis do master são transmitidos ao slave e aparecem na saída do flip-flop. Logo, este flip-flop comuta nos flancos ascendentes dos impulsos de relógio, ao contrário do que sucedia com os flip-flops master-slave anteriores.

13.2 Flip-flops Edge-triggered

Flip-flop edge-triggered

Os flip-flops master-slave que acabámos de analisar não cobrem todos os tipos de comportamento possível em relação à entrada de relógio. Um outro tipo de reacção é a que os flip-flops usualmente chamados **edge-triggered** apresentam. Nestes flip-flops, a reacção, a dar-se, ocorre num determinado flanco do impulso de relogio e é independente dos níveis nas entradas, excepto num curto intervalo de tempo em torno desse flanco.

Flip-flop D edge-triggered

Examinemos o **flip-flop D edge-triggered** da Figura 13.9. (Nota: A descrição seguinte está incluída para satisfazer a curiosidade do estudante. A estrutura interna dos edge-triggered não faz parte da matéria da disciplina; porém, o uso deste tipo de flip-flops, faz!)

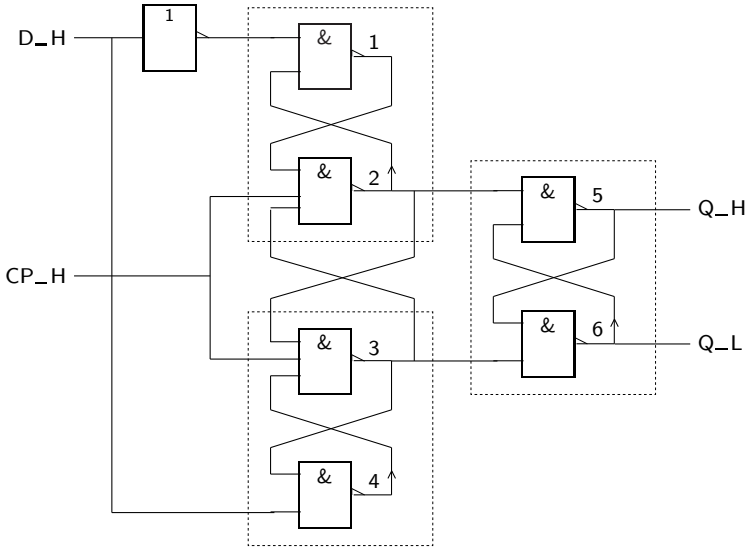


Figura 13.9: Flip-flop D edge-triggered que comuta nos flancos ascendentes dos impulsos de relógio

Funcionamento de um flip-flop D edge-triggered

O funcionamento deste circuito é já um pouco mais difícil de seguir. Para começar, é conveniente encará-lo como sendo constituído por 3 latches SR simples, conforme se indica na figura.

Repare-se, agora, no seguinte: enquanto a linha CP_H se mantém em L, as saídas dos ANDs 2 e 3 estão em H e, por consequência, o latch de saída manter-se-á no estado em que se encontrava anteriormente, independentemente da entrada D. Isto é, o flip-flop mostra na saída o valor anteriormente memorizado.

Esta entrada D tem, por outro lado, influência directa nas saídas dos ANDs 1 e 4. Com efeito, estando as saídas de 2 e 3 em H, a saída do AND 1 é igual a D, enquanto que o AND 4 assume o nível oposto. As variações de nível em D reflectem-se nestes dois pontos mas não avançam mais enquanto os circuitos 2 e 3 estiverem bloqueados pela inactividade da variável CP. Portanto o sinal CP_H está a "aguentar" os dois latches da primeira camada.

Vamos agora supor, para a continuação da nossa análise, que D_H assume o valor H. A análise podia ser identicamente feita para D_H em L.

Teremos, nesse caso, que a saída do AND 1 assume o nível H e a do AND 4 o nível L. Se agora CP passar a estar activa, isto é, se CP_H vier a H, verifica-se o seguinte: as entradas do AND 2 estão todas com o valor H, tal como as do AND 3 com excepção, nesta última, da entrada que provém da saída do AND 4, que está em L. Então, a saída do AND 2 passará a L, originando um estado estável no latch superior. A saída do AND 3 continuará a H, mantendo um estado estável no latch inferior. O latch de saída assume o estado H, isto é, copia o valor da linha D_H.

A partir de agora as alterações na entrada D não afectam o circuito. De facto, estando a saída do AND 2 a L, ficam impostas a H as saídas dos ANDs 1 e 3 impossibilitando, por este meio, qualquer influência de D no latch de saída. Repare-se que, quando o latch superior da primeira camada reagiu, bloqueou qualquer mudança no latch inferior a partir desse momento e até CP_H voltar a L e retomar o nível H.

Quando a linha CP_H volta a assumir o valor L, as saídas dos NANDs 2 e 3 voltam a H e retoma-se a situação inicial. Assim, nesta transição, não ocorre qualquer mudança de estado.

Na Figura 13.10 apresenta-se um diagrama temporal que ilustra o comportamento do flip-flop D edge-triggered da Figura 13.9. Recomenda-se uma observação atenta desse diagrama em conjunção com a logigrama do flip-flop. Assume-se que, inicialmente, a linha Q_H está inactiva.

Recapitulando: no flip-flop D edge-triggered da Figura 13.9 a saída Q_H assume o estado da entrada D_H quando se dá uma transição L → H na entrada CP_H, e mantém o estado, independentemente do nível em D_H, enquanto CP_H está a H, a L, ou ainda durante as transições H → L nessa linha.

O flip-flop só reage, portanto, quando na entrada CP_H ocorrem flancos ascendentes dos impulsos de relógio. Diz-se, por isso, que este flip-flop **comuta nos flancos ascendentes** desses impulsos.

Comutação nos flancos ascendentes

Na Tabela 13.3 descreve-se o funcionamento deste flip-flop através da sua **tabela de verdade física**.

Tabela de verdade física de um flip-flop D edge-triggered

Da tabela deduz-se existirem os seguintes **modos de funcionamento** para este flip-flop:

Modos de funcionamento

— o **modo de manutenção** do estado, quando não existe flanco de comutação, em que o estado do flip-flop se mantém e, por conseguinte, se tem $Q\_H_{(t+1)} = = Q\_H_{(t)}$; do mesmo modo, $Q\_L_{(t+1)} = Q\_L_{(t)}$; e

Modo de manutenção

— o **modo de cópia** do flip-flop, quando existe flanco de comutação, em que se copia para a saída Q_H o nível de tensão que se encontrar aplicado à entrada D_H.

Modo de cópia

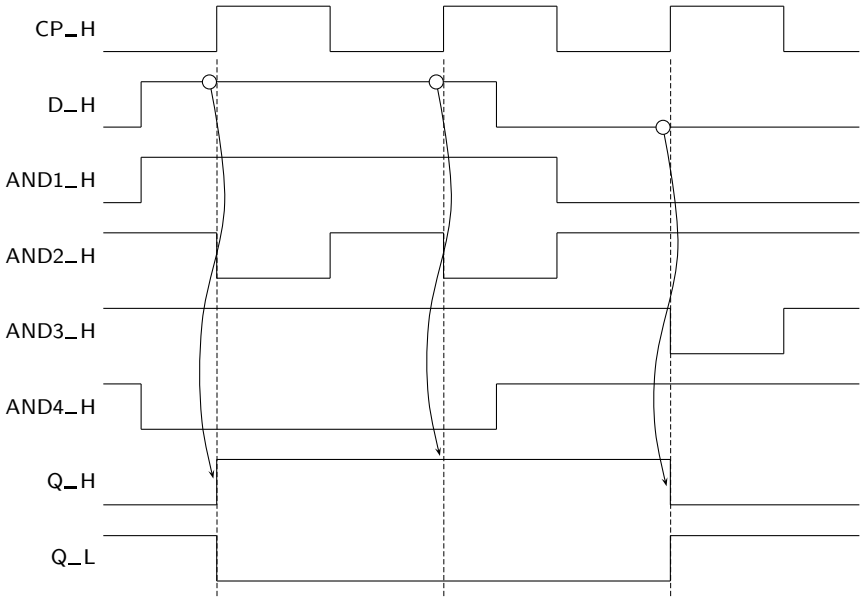


Figura 13.10: Diagrama temporal com o funcionamento do flip-flop D edge--triggered da Figura 13.9

Tabela 13.3: Tabela de verdade física de um flip-flop D edge-triggered que comuta nos flancos ascendentes

$D\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$	Modo
L	↑	L	H	Cópia
H	↑	H	L	Cópia
×	↓	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção

Como é óbvio existem também outros tipos de flip-flops edge-triggered. São comuns os flip-flops JK.

Na Figura 13.11 ilustra-se a representação deste tipo de flip-flops de acordo com a norma IEC 60617-12.

Símbolo do flip-flop D edge-triggered

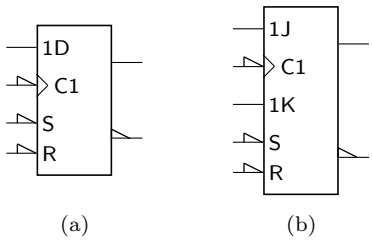
Símbolo do flip-flop JK edge-triggered

Na parte a) da figura representa-se um flip-flop D edge-triggered e na parte b) um flip-flop JK, em ambos os casos que comutam nos flancos descendentes dos impulsos de relógio.

Qualificadores de entrada ▷ e ◺

O triângulo interior ao símbolo (**qualificador de entrada** ▷), na linha de relógio, indica que o flip-flop é do tipo edge-triggered.

A presença de um triângulo exterior à entrada de relógio do flip-flop, orientado no sentido do fluxo dos sinais (**qualificador de entrada** ◺), indica que o flip--flop comuta nos flancos descendentes. Se o triângulo estiver ausente, o flip-flop comuta nos flancos ascendentes do sinal de relógio.



(a) (b)

Figura 13.11: (a) Símbolo IEC de um flip-flop D edge-triggered com entradas assíncronas de Preset e de Clear e que comuta nos flancos descendentes dos impulsos de relógio; (b) símbolo idêntico para um flip-flop JK edge-triggered

Em ambos os casos, representam-se nos símbolos **entradas assíncronas de Set (Preset) e de Reset (Clear)**.

Entradas assíncronas de Set (S) e de Reset (R)

13.3 Temporizações nos Flip-flops

Os flip-flops são caracterizados, do ponto de vista temporal, por vários parâmetros.

Para começar, o tempo de propagação. O **tempo de propagação** (ou **tempo de atraso**) é, definido de forma geral, como o intervalo de tempo que decorre entre o instante em que ocorre o flanco activo do sinal de relógio (flanco de comutação) e o instante em que as saídas vêem actualizadas. É representado por t_{pd}.

Tempo de propagação (atraso) de um flip-flop

t_{pd}

O tempo de propagação deve ser distinguido consoante ocorrer um flanco ascendente ou um flanco descendente numa saída de um flip-flop, em resposta ao aparecimento de um flanco de comutação. Se, por exemplo, na saída Q ocorrer uma transição do nível L para o nível H, o tempo de propagação designa-se por t_{pLH}. No caso contrário, designa-se por t_{pHL}, sendo que t_{pLH} e t_{pHL} podem ter valores diferentes.

t_{pLH} e t_{pHL}

Por outro lado, os tempos de propagação t_{pLH} e t_{pHL} aplicam-se igualmente à saída $\overline{\mathrm{Q}}$ e, em geral, possuem um valor diferente dos valores para a saída Q. Dos 4 valores possíveis, a Figura 13.12 apenas representa dois deles.

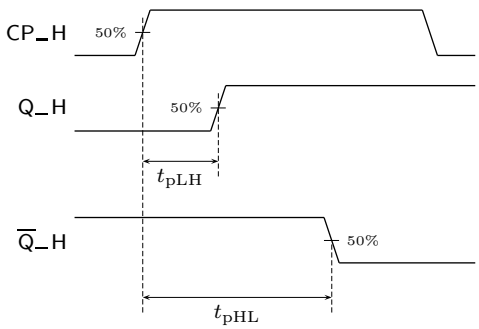


Figura 13.12: Tempos de propagação de um flip-flop

No momento em que o relógio muda de nível, as entradas devem estar estáveis, por forma a evitar uma situação em que a reacção do flip-flop fique dependente de uma corrida entre sinais dentro do circuito e seja, portanto, não previsível.

Isso é evitado, no caso dos flip-flops edge-triggered, evitando mudanças nas entradas num intervalo de tempo que envolve o flanco de comutação.

Tempo de preparação ou "set-up time", t_{su}

O intervalo de tempo que decorre entre o instante em que as entradas têm de estar estáveis e o momento em que ocorre o flanco de comutação denomina-se **tempo de preparação** ou **"set-up time"**, e representa-se por t_{su}.

Tempo de manutenção ou "hold time", t_h

O intervalo de tempo que medeia entre o instante em que ocorre o flanco de comutação e o momento em que as entradas já podem variar é o **tempo de manutenção** ou **"hold time"** e representa-se por t_h.

Na Figura 13.13 ilustram-se esses tempos.

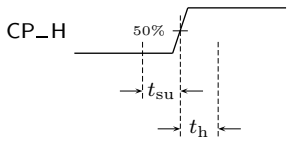


Figura 13.13: Tempos de preparação e de manutenção de um flip-flop

13.4 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secção 5.3.

13.5 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 13.1 Identificar, na Figura 13.4, os instantes ou os intervalos de tempo em que ocorre o fenómeno de "one's catching".

(∗) 13.2 Desenhar o símbolo IEC de um flip-flop SR master-slave sem entradas assíncronas de Set e de Reset.

(∗) 13.3 Desenhar o símbolo IEC de um flip-flop SR master-slave com entradas assíncronas de Set e de Reset (ou de Preset e de Clear). Todas as entradas, síncronas e assíncronas, devem ser activas a L. Descrever o funcionamento deste flip-flop com um diagrama temporal onde se realce o efeito das entradas síncronas e assíncronas.

(∗) 13.4 Construir um flip-flop JK master-slave a partir de:

a) um flip-flop D master-slave;
b) um latch D controlado.

13.5 Construir um flip-flop T edge-triggered a partir de um flip-flop JK, também edge-triggered. Um flip-flop T tem a **tabela de verdade física** que se indica na Tabela 13.4, para o caso em que comuta nos flancos ascendentes. De notar que o comportamento deste flip-flop garante que, a cada flanco de comutação, há troca do seu estado se a entrada síncrona T estiver activa. Em todas as outras situações, o flip-flop mantém o estado.

Tabela de verdade física de um flip-flop T edge-triggered

Tabela 13.4: Tabela de verdade física de um flip-flop T edge-triggered que comuta nos flancos ascendentes

$T\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$
L	↑	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
H	↑	$\overline{Q}\_H_{(t)}$	$Q\_H_{(t)}$
×	↓	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$

(∗) 13.6 O flip-flop hipotético A, do tipo edge-triggered, é obtido por transformação de um flip-flop JK do mesmo tipo como mostra a Figura 13.14. Será que o flip-flop A é facilmente utilizável na prática, ou apresenta problemas?

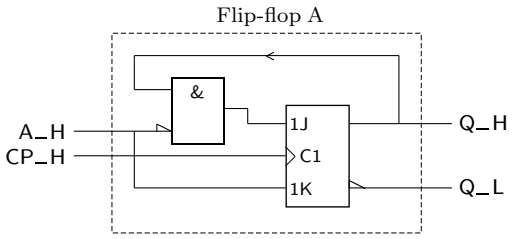


Figura 13.14: Flip-flop hipotético, do tipo edge-triggered, com a designação de flip-flop A

(∗) 13.7 Para o circuito representado na Figura 13.15, estabelecer o diagrama temporal da saída S entre t_0 e t_1, admitindo que em t_0 se tem $(Q1, Q2, Q3) = = (\mathrm{L}, \mathrm{H}, \mathrm{H})$.

(∗) 13.8 Considere o circuito representado na Figura 13.16 e admita que os flip-flops utilizados possuem $t_\mathrm{h} = 5$ ns e $t_\mathrm{su} = 4$ ns, e que as portas lógicas possuem o mesmo tempo de atraso, $t_\mathrm{pd} = 10$ ns.

Analisando o circuito apresentado, e tendo em consideração as características indicadas, responda às seguintes perguntas.

a) Qual o tipo de flip-flop utilizado?

b) Qual o tempo de atraso mínimo de um flip-flop para que o circuito funcione correctamente? E qual é a frequência máxima de funcionamento do circuito nessas circunstâncias?

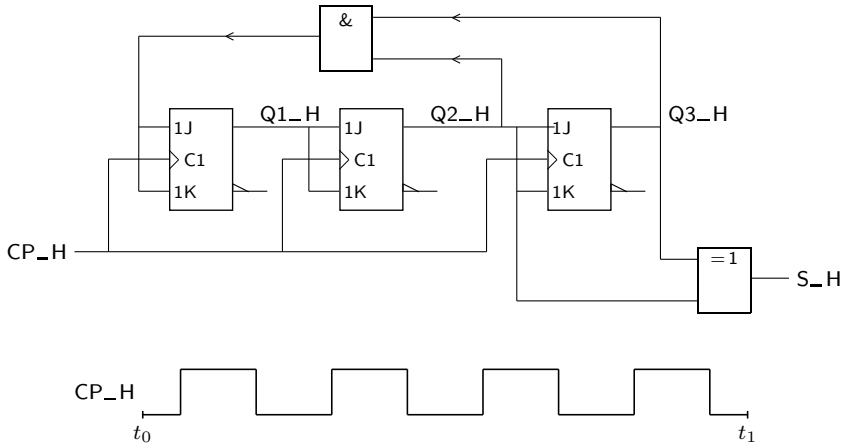


Figura 13.15: Exemplo de circuito sequencial síncrono

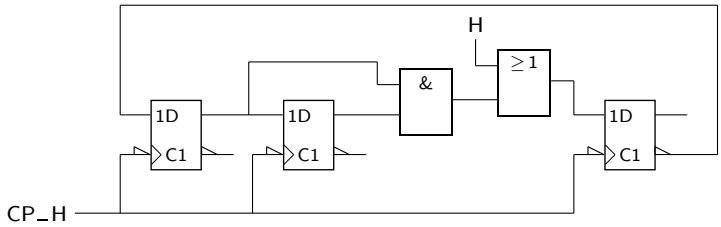


Figura 13.16: Exemplo de circuito sequencial síncrono

13.9 Usando um flip-flop RS, construa um flip-flop XY edge-triggered que comuta nos flancos ascendentes e que possui a tabela de verdade física da Tabela 13.5.

Tabela 13.5: Tabela de verdade física de um flip-flop XY edge-triggered que comuta nos flancos ascendentes

$X\_H_{(t)}$	$Y\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$
L	L	⎍	$\overline{Q}\_H_{(t)}$	$Q\_H_{(t)}$
L	H	⎍	H	L
H	L	⎍	L	H
H	H	⎍	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
×	×	⎎	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
×	×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$
×	×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$

13.10 Considere um flip-flop D edge-triggered. Transforme-o, usando lógica exterior, num JK edge-triggered.

13.11 Dispõe de latches D controlados. Construa com eles e com a lógica com-

binatória que entender um flip-flop JK master-slave.

13.12 Transforme um flip-flop D num flip-flop RS.

13.13 Mostre que, no circuito da Figura 13.17, que representa um flip-flop master-slave, as entradas directas (assíncronas) S e R se sobrepõem às entradas síncronas S e R.

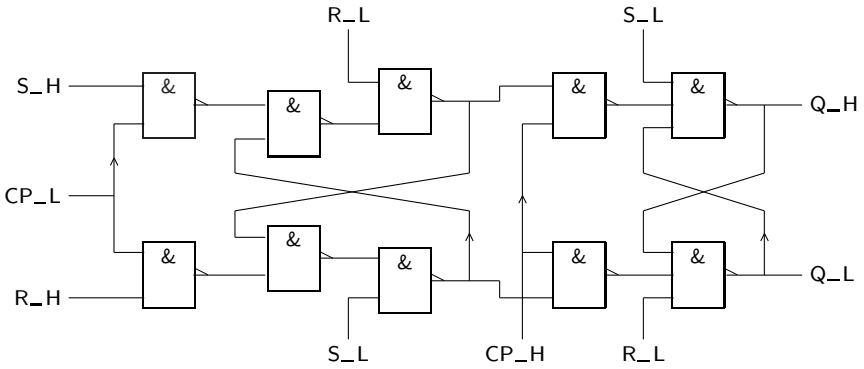
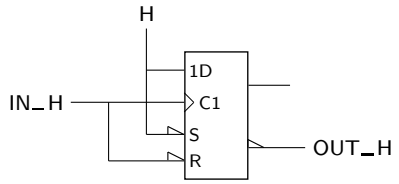


Figura 13.17: Flip-flop RS master-slave com entradas assíncronas de Set e de Reset

(∗) 13.14 O circuito da Figura 13.18 é baseado num flip-flop D edge-triggered e constitui uma proposta de aproveitamento deste circuito para substituir uma porta NOT (!).



$t_{\mathrm{su,D\ max}} = 20$ ns
$t_{\mathrm{su,RS\ max}} = 6$ ns
$t_{\mathrm{h}} = 0$ ns
$t_{\mathrm{pd,RS\ max}} = 50$ ns
$t_{\mathrm{pd,C\ max}} = 40$ ns

$S\_L_{(t)}$	$R\_L_{(t)}$	$D\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$	Modo
L	H	×	×	H	L	Set
H	L	×	×	L	H	Reset
L	L	×	×	H	H	Força HH
H	H	H	↑	H	L	Cópia
H	H	L	↑	L	H	Cópia
H	H	×	↓	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
H	H	×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
H	H	×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção

Figura 13.18: Circuito do Exercício 13.14

Com efeito, enquanto a variável IN de entrada está inactiva (a L), o Reset assíncrono do flip-flop actua e OUT_L fica a H. Por outro lado, quando IN muda de L para H, vem aplicado à entrada de relógio do flip-flop um flanco ascendente e, como a entrada D está activa, a função OUT de saída passa a L. Ou seja, aparentemente $OUT = \overline{IN}$.

Os parâmetros temporais do flip-flop estão indicados na figura: $t_{\mathrm{su,D\ max}}$ é o tempo máximo de preparação da entrada D, $t_{\mathrm{su,RS\ max}}$ é o tempo máximo de preparação das entradas R e S, t_{h} é o tempo de manutenção das entradas, $t_{\mathrm{pd,RS-\overline{Q}\ max}}$ é o tempo máximo de propagação desde as entradas R e S até à saída $\overline{\mathsf{Q}}$, e $t_{\mathrm{pd,C-\overline{Q}\ max}}$ é o tempo máximo de propagação desde a entrada de relógio até à saída $\overline{\mathsf{Q}}$.

Explique porque é que o circuito não funciona.

13.15 Considere o circuito da Figura 13.19, em que B é um botão que, quando premido, estabelece o contacto a tracejado e, quando não premido, estabelece o contacto a cheio.

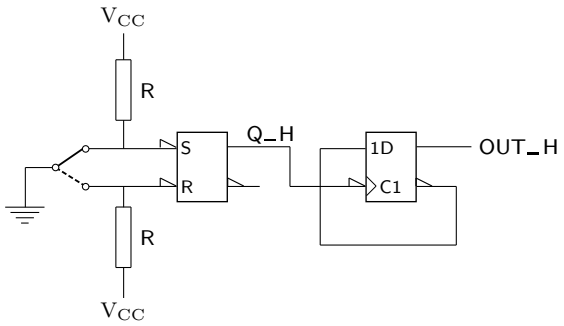


Figura 13.19: Circuito do Exercício 13.15

a) Qual é a utilidade do circuito?

b) Utilize um flip-flop JK para realizar, no circuito, a mesma função que a que está a ser realizada pelo flip-flop D.

13.16 Verifique que o circuito da Figura 13.20 funciona como um flip-flop que opera correctamente se se considerar o tempo de propagação nas portas OR da entrada.

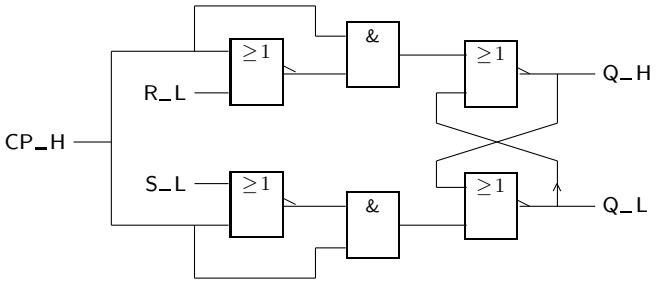


Figura 13.20: Circuito do Exercício 13.16

Capítulo 14

Contadores

14.1 Contadores Assíncronos

Uma aplicação clássica dos flip-flops é na realização de contadores assíncronos.

Esses contadores são circuitos sequenciais constituidos por vários flip-flops interligados, o primeiro dos quais recebe impulsos de relógio pela **linha de relógio** do contador, que está ligada à sua entrada de relógio. Uma das saídas deste flip-flop, por sua vez, vai servir de entrada de relógio ao flip-flop seguinte, este ao seguinte, e por aí fora, até ao último flip-flop do contador.

Linha de relógio de um contador assíncrono

Cada impulso de relógio faz evoluir a configuração dos flip-flops, que passam a contar segundo um determinado código com palavras com um comprimento que é igual ao número de flip-flops do contador.

Ou seja, o que os contadores assíncronos contam são os impulsos de relógio. Mais exactamente, a evolução através da **sequência de estados de contagem** é feita pelas transições entre **estados de contagem**, que são consequência do aparecimento de **flancos activos** ou **flancos de comutação** aplicados às entradas de relógio dos flip-flops.

Sequência de estados (de contagem)

Estados de contagem

Flanco activo (de comutação)

Por exemplo, um **contador binário de 3 bits** é um circuito com três flip-flops que evolui ao longo da sequência de (estados de) contagem da Tabela 14.1, correspondente ao CBN com palavras de comprimento 3.

Contador binário de 3 bits

É fácil de perceber que o flip-flop cuja saída foi denominada $Q0$ (o que tem menor peso) muda de estado cada vez que o contador recebe um impulso de relógio. Um flip-flop JK com ambas as entradas a H tem exactamente esse comportamento (modo de comutação). Usaremos um flip-flop edge-triggered que comuta nos flancos descendentes dos impulsos aplicados à sua entrada de relógio, como sugere a Figura 14.1. A escolha entre edge-triggered e master-slave é irrelevante, mas a definição do flanco de comutação não é, como veremos.

Se observarmos, agora, de novo a sequência de contagem, podemos observar que o flip-flop $Q1$ muda de estado sempre que o flip-flop $Q0$ transita de H para L. É o que acontece nas transições LLH → LHL, LHH → HLL, etc.

Como estamos a usar flip-flops que reagem nos flancos descendentes dos impulsos de relógio, podemos, então usar, precisamente a transição de $Q0$ para atacar a

Tabela 14.1: Sequência de contagem para um contador binário de 3 bits, que utiliza o CBN com palavras de comprimento 3

Sequência de contagem		
Q2_H	Q1_H	Q0_H
L	L	L
L	L	H
L	H	L
L	H	H
H	L	L
H	L	H
H	H	L
H	H	H
L	L	L
. . .		

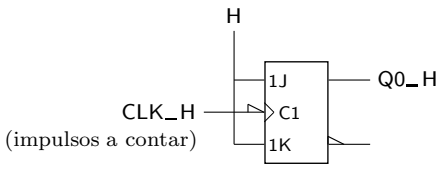


Figura 14.1: Andar $Q0$ (de menor peso) de um contador binário de 3 bits, que conta segundo o CBN

entrada de relógio de $Q1$. As entradas J e K do flip-flop $Q1$ terão de estar igualmente a H para que, sempre que surge a transição referida, o flip-flop mude de estado. Obtemos, assim, o logigrama parcial da Figura 14.2.

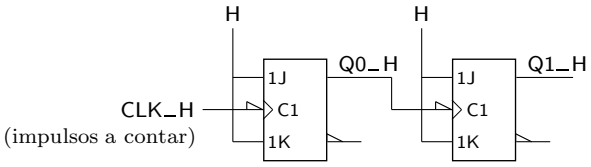


Figura 14.2: Os andares $Q0$ (de menor peso) e $Q1$ (intermédio) de um contador binário de 3 bits, que conta segundo o CBN

Do mesmo modo, o terceiro flip-flop deverá mudar de estado quando o segundo transita de H para L. Logo, o circuito completo será o que se indica na Figura 14.3.

Neste contador, como se pode perceber, os flip-flops nunca reagem simultaneamente. É a mudança de um que pode provocar a mudança do outro. Neste sentido, não há qualquer sincronismo entre os diversos flip-flops. O circuito é

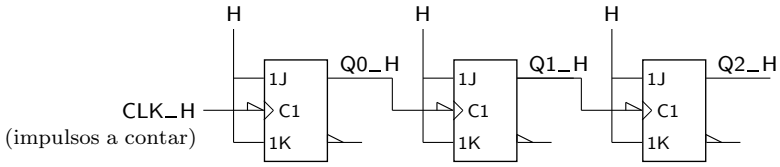


Figura 14.3: Os três andares de um contador binário de 3 bits, que conta segundo o CBN

assíncrono, por isso, e o contador é um **contador assíncrono**.

Contador assíncrono

Repare-se que é importante serem descendentes os flancos de relógio em que os flip-flops reagem. Algumas questões se podem colocar. Que aconteceria se os flip-flops reagissem nos flancos ascendentes? E, se isso acontecesse, como se poderia, apesar de tudo, com alterações ao circuito, obter o mesmo tipo de comportamento? São perguntas interessantes.

Este contador conta segundo uma sequência de 8 estados. Por isso se designa por **contador módulo 8**. Repare-se que, atingido o último estado de contagem (o estado 7) e recebido um novo impulso de relógio, o contador evolui de novo para o primeiro estado (o estado 0) e recomeça a contagem.

Contador módulo 8

Os contadores assíncronos podem ainda incluir uma entrada de Reset assíncrona que permite iniciá-los no estado 0. Por exemplo, o contador da Figura 14.3 pode ser acrescentado com a entrada $RESET\_L$ da Figura 14.4 que, quando activa, actua de imediato as entradas assíncronas dos flip-flops, levando o contador para o estado LLL.

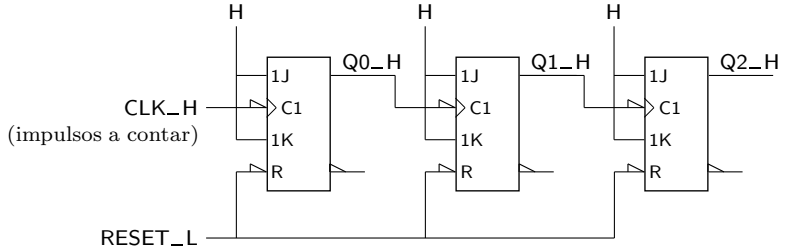


Figura 14.4: Contador asssíncrono módulo 8 que conta segundo o CBN, com entrada assíncrona de Reset

14.1.1 Flip-flops T

Como se pode ver pela aplicação feita, os flip-flops JK estão subaproveitados, uma vez que, dos seus quatro modos de funcionamento, apenas um (com $J\_H = = K\_H = H$, o modo de comutação) está a ser usado.

Por isso, surge frequentemente, no contexto dos contadores, um novo tipo de flip-flop que é uma simplificação do JK. Trata-se de um flip-flop apenas com uma entrada síncrona, designada por T_H, tal que quando $T\_H = L$ mantém o estado, e quando $T\_H = H$ complementa o estado. A tabela de verdade física

Os fabricantes de circuitos integrados raramente disponibilizam este tipo de flip-flop. Na prática, os flip-flops T são substituídos por flip-flops JK com as entradas síncronas ligadas a H.

de um flip-flop T edge-triggered será, assim, a da Tabela 14.2, se admitirmos que o flip-flop comuta nos flancos ascendentes dos impulsos de relógio.

Tabela 14.2: Tabela de verdade física de um flip-flop T edge-triggered que comuta nos flancos ascendentes dos impulsos de relógio

$T\_H_{(t)}$	CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$	Modo
L	⎍	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
H	⎍	$\overline{Q}\_H_{(t)}$	$Q\_H_{(t)}$	Comutação
×	⎎	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
×	H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção

Modos de funcionamento

Note-se como este flip-flop apenas possui dois **modos de funcionamento**:

Modo de manutenção

— o **modo de manutenção** do estado, quando não existe flanco de comutação ou quando existe flanco e $T\_H = L$, em que o estado do flip-flop se mantém e, por conseguinte, se tem $Q\_H_{(t+1)} = Q\_H_{(t)}$; do mesmo modo, $Q\_L_{(t+1)} = = Q\_L_{(t)}$; e

Modo de comutação

— o **modo de comutação**, quando existe flanco de comutação e $T\_H = H$, em que os níveis de tensão nas saídas mudam (comutam).

Símbolo do flip-flop T edge-triggered

O símbolo IEC deste flip-flop é o que se apresenta na Figura 14.5, admitindo que ele é do tipo edge-triggered e que comuta nos flancos ascendentes (dos impulsos de relógio).

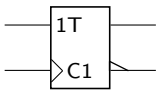


Figura 14.5: Símbolo IEC de um flip-flop T edge-triggered que comuta nos flancos ascendentes

Uma versão ainda mais simples prescinde da entrada T_H e tem apenas a entrada de relógio, como mostra a Tabela 14.3.

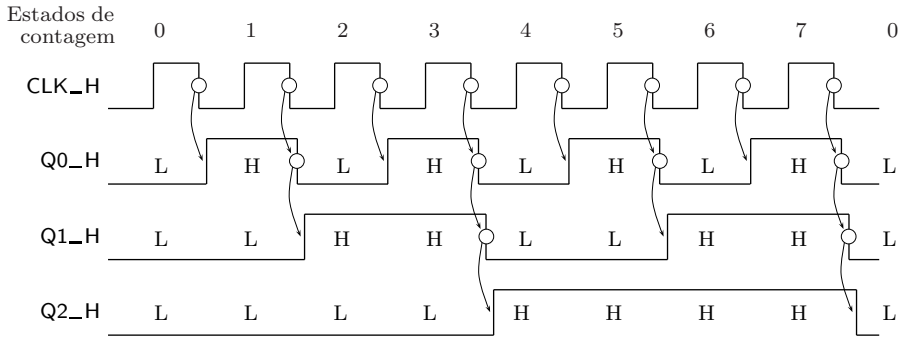
Tabela 14.3: Tabela de verdade física de um flip-flop T edge-triggered simplificado que comuta nos flancos ascendentes

CP_H	$Q\_H_{(t+1)}$	$\overline{Q}\_H_{(t+1)}$	Modo
⎍	$\overline{Q}\_H_{(t)}$	$Q\_H_{(t)}$	Comutação
⎎	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
L	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção
H	$Q\_H_{(t)}$	$\overline{Q}\_H_{(t)}$	Manutenção

Este tipo de flip-flop permite construir contadores do tipo anteriormente ilustrado de forma mais simples mas com a mesma estrutura.

14.1.2 Diagrama temporal

Como se viu atrás, a evolução dos estados dos contadores assíncronos tem a particularidade de, para além de não ser síncrona, se realizar com as transições de estado a ocorrerem na sequência de outra transição. Ilustremos isso no diagrama temporal da Figura 14.6, em que representamos (um pouco exageradamente para a escala) os tempos de atraso dos flip-flops envolvidos no contador binário de módulo 8 da Figura 14.3.



Nota: As curvas com setas indicam transições nas saídas dos flip-flops que são consequência de flancos descendentes (flancos de comutação) dos flip-flops.

Figura 14.6: Diagrama temporal que explicita as transições de estados nas saídas dos flip-flops do contador binário de módulo 8 da Figura 14.3, com o consequente aparecimento de estados instáveis em algumas das transições

Neste diagrama podemos observar facilmente a evolução dos estados de contagem. Mas também se pode ver que existem estados intermédios, que são **estados instáveis** (ou **transitórios**) que correspondem a fases de transição entre dois **estados estáveis**. São uma consequência directa do mecanismo de contagem. No diagrama temporal que se repete na Figura 14.7 podem ser vistos os estados de contagem, estáveis (a negrito) e instáveis (em itálico).

Estados estáveis e instáveis (transitórios)

Os estados instáveis assinalados existem naturalmente devido ao mecanismo de contagem e são perfeitamente previsíveis, devido exactamente a esse facto. A sua duração é da ordem do tempo de propagação dos flip-flops utilizados.

14.1.3 Contadores assíncronos com módulos arbitrários

É fácil conceber um contador assíncrono para contar em qualquer módulo que seja potência de 2. No entanto, por vezes são precisos contadores que contem noutros módulos. A solução clássica consiste em utilizar um contador com um módulo que é a potência de 2 imediatamente superior, e utilizar a entrada assíncrona de Reset do contador para o levar para o estado 0, após o último estado de contagem.

Para isso, realiza-se esse Reset com a *descodificação do estado a seguir ao último*

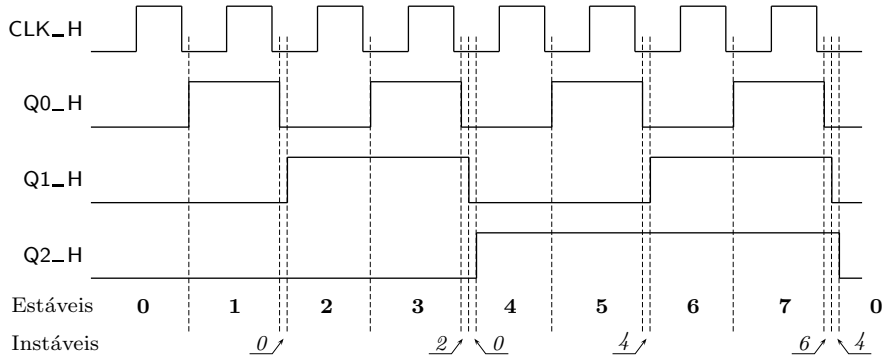


Figura 14.7: Repetição do diagrama temporal da Figura 14.6 que identifica os estados estáveis de contagem (a negrito) e os estados instáveis que ocorrem nalgumas transições (em itálico)

estado de contagem. É usado esse estado e não o último porque o último estado tem de ser estável até vir o impulso de relógio a contar. Assim, por exemplo, para realizar um contador binário de módulo 6 que vai contar de 0 a 5 no CBN, utiliza-se um contador módulo 8, descodifica-se o estado 6 e aplica-se o sinal resultante da descodificação ao Reset assíncrono dos flip-flops, como mostra a Figura 14.8.

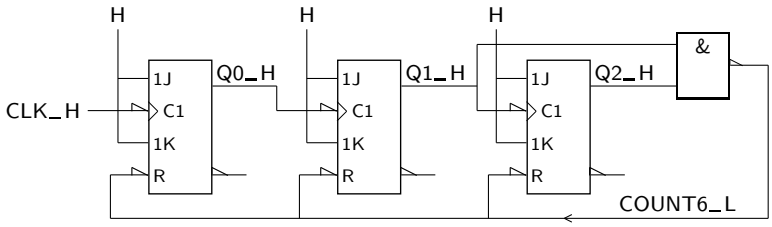


Figura 14.8: O contador assíncrono binário de 3 bits da Figura 14.3, que conta segundo o CBN, vem modificado para contar apenas de 0 a 5

Note-se que, para descodificar o estado 6, apenas se tiveram em consideração os níveis H de saída dos dois últimos flip-flops. Isso resulta do facto de, para qualquer sequência de contagem, o conjunto dos bits a 1 de um número surgir simultaneamente a 1 pela primeira vez justamente nesse número. Daí que não valha a pena ter os "0"s em consideração. De facto, no nosso caso o estado $(Q2\_\mathrm{H}, Q1\_\mathrm{H}, Q0\_\mathrm{H}) = (\mathrm{H,H,L})$ é o primeiro que tem os dois níveis mais significativos a H. O outro é o estado $(Q2\_\mathrm{H}, Q1\_\mathrm{H}, Q0\_\mathrm{H}) = (\mathrm{H,H,H})$, que apenas surgiria após o estado $(Q2\_\mathrm{H}, Q1\_\mathrm{H}, Q0\_\mathrm{H}) = (\mathrm{H,H,L})$ se não fizessemos o Reset dos flip-flops.

Note-se, em segundo lugar, que criámos mais um estado instável, o 6. Mas isso é habitual nos contadores assíncronos.

Por fim, há que ter em conta que o conjunto dos tempos de atraso dos dispositivos utilizados tem de ser tal que garanta que todos os flip-flops são postos a L antes do sinal de Reset ser desactivado.

14.1.4 Símbolos dos contadores assíncronos

A Figura 14.9 ilustra o símbolo IEC de um contador assíncrono em tecnologia TTL do tipo 74LS293.

74LS293

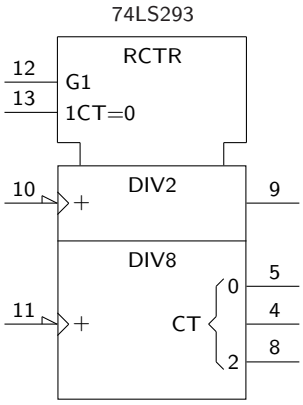


Figura 14.9: Símbolo do contador assíncrono 74LS293, de acordo com a norma IEC

O símbolo possui três blocos distintos. O bloco superior é designado, como já sabemos, por **bloco de controlo comum**, e contém as dependências que são comuns aos blocos inferiores.

Bloco de controlo comum

O **qualificador geral** RCTR no bloco de controlo comum identifica um contador assíncrono ("Ripple Counter"). Os dois blocos inferiores, com os qualificadores gerais DIV2 e DIV8, significam que o contador é, na realidade, formado por dois contadores (que funcionam como **divisores de frequência**, um divisor por 2 e um divisor por 8).

Qualificador geral RCTR

Divisor de frequência

Se considerarmos um diagrama temporal que envolva a entrada de relógio do contador e as saídas CT0 a CT2 do divisor por 8, constatamos que essas saídas possuem, respectivamente, frequências que são 1/2, 1/4 e 1/8 da frequência de relógio (daí o nome de divisor de frequência por 2, por 4 ou por 8).

As entradas que fazem parte do bloco de controlo comum comandam os dois blocos inferiores. A entrada superior estabelece uma dependência And (ou dependência G) sobre a outra entrada, conferida pelos **qualificadores de entrada** G1 e 1.

Qualificadores de entrada G1 e 1

O **qualificador de entrada** CT=m significa que, quando a entrada correspondente estiver activa, o contador vem carregado internamente com o valor m. No caso do 74LS293, CT=0 significa o carregamento de 0 no contador, o que é o mesmo que dizer que se faz o seu Reset. Da conjugação das funções das duas entradas do bloco de controlo comum podemos então concluir que, se ambas as entradas estiverem activas (ambas a H), se fará o Reset assíncrono dos dois contadores/divisores de frequência.

Qualificador de entrada CT=m

Por seu turno, o **qualificador de saída** CTm vem colocado em múltiplas saídas agrupadas por uma chaveta, o que significa que o estado de contagem do contador vem dado pela soma das potências de 2 correspondentes às saídas activas. Por exemplo, se o divisor de frequência por 8 se encontrar no estado de contagem 6, as saídas identificadas por CT1 e por CT2 estão activas (isto é, a H), enquanto que a saída identifcada por CT0 está inactiva (isto é, a L).

Qualificador de saída CTm

Dado que o bloco intermédio do 74LS293 constitui um divisor por 2, será naturalmente composto por um único flip-flop, enquanto que o divisor por 8 do bloco

Qualificador de entrada +

inferior será composto por três flip-flops. As entradas de relógio para ambos os divisores tornam-se activas nos flancos descendentes dos impulsos a elas aplicados. O **qualificador de entrada** + que lhes está afecta significa que os contadores são ambos do tipo ascendente, incrementando de uma unidade a cada flanco descendente dos impulsos de relógio respectivos.

14.2 Contadores Síncronos

Estes contadores são ainda circuitos sequenciais constituidos por vários flip-flops interligados, tal como sucede com os contadores assíncronos. E tal como eles, cada impulso de relógio aplicado à linha de relógio faz evoluir a sequência de estados do contador, que passa a contar segundo um determinado código pré-estabelecido.

Linha de relógio de um contador síncrono

Porém, ao contrário dos contadores assíncronos, a interligação entre os flip-flops é completamente diferente. Agora, todos os flip-flops recebem em simultâneo impulsos de relógio pela **linha de relógio** do contador, que está ligada às entradas de relógio de *todos* os flip-flops. Isso significa que *todos os flip-flops são actuados no mesmo flanco* dos impulsos de relógio.

Tal como no caso dos contadores assíncronos, o que os contadores síncronos contam são os impulsos de relógio. Mais exactamente, contam o número de flancos activos aplicados à linha de relógio.

14.2.1 Concepção heurística de um contador síncrono

A concepção de um contador síncrono pode ser feita aplicando os conceitos de síntese de circuitos sequenciais síncronos que serão apresentados no Capítulo 16, partindo de um diagrama de estados que reflicta a sequência de contagem pretendida.

Em certos casos, porém, é possível realizar uma concepção mais heurística. Retomemos o exemplo de um contador binário de 3 bits. Pretende-se um contador que conte segundo a sequência da Tabela 14.1 da página 234, que repetimos na Tabela 14.4 por facilidade de exposição.

Serão necessários três flip-flops. Utilizaremos, arbitrariamente, flip-flops JK edge-triggered a comutarem nos flancos ascendentes. Como o circuito é síncrono, teremos as linhas de relógio interligadas entre si e à linha de relógio do contador. As três saidas serão denominadas Q_2_H, Q_1_H e Q_0_H, sendo Q_2_H a que vai corresponder ao bit mais significativo da contagem. Podemos, então, começar a construir o contador, como mostra a Figura 14.10.

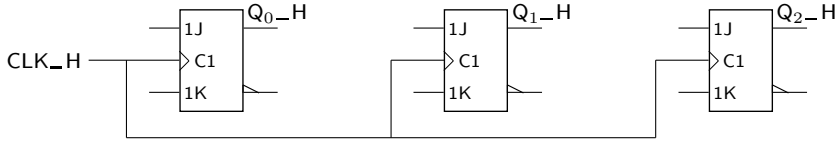


Figura 14.10: Primeira fase de construção do contador síncrono módulo 8, que conta segundo o CBN

Tabela 14.4: Sequência de contagem para um contador binário de 3 bits, que utiliza o CBN com palavras de comprimento 3

Sequência de contagem		
Q2_H	Q1_H	Q0_H
L	L	L
L	L	H
L	H	L
L	H	H
H	L	L
H	L	H
H	H	L
H	H	H
L	L	L
	. . .	

Repare-se que, tal como no caso do contador assíncrono, o flip-flop Q_0 muda de estado em todas as transições, isto é, muda de estado sempre que surge um flanco activo na linha de relógio. Isso consegue-se, tal como no caso referido, colocando as linhas J e K do flip-flop sempre a H.

No caso do segundo flip-flop, ele deve mudar de estado, como se pode concluir por inspecção da sequência de estados pretendida, sempre que no bit menos significativo se dá a transição de H para L. Aqui não podemos utilizar essa transição para actuar o relógio do flip-flop, como fizemos no caso assíncrono. Assim, temos de reformular a nossa táctica. Tudo o que podemos controlar são as linhas J e K. E essas devem ser postas a H (para o flip-flop mudar de estado) quando Q_0_H está a H. Passamos, assim a utilizar os *níveis* de tensão dos sinais e não as *transições* dos mesmos para condicionar as mudanças de estado.

O circuito evolui, então, para o logigrama parcial da Figura 14.11.

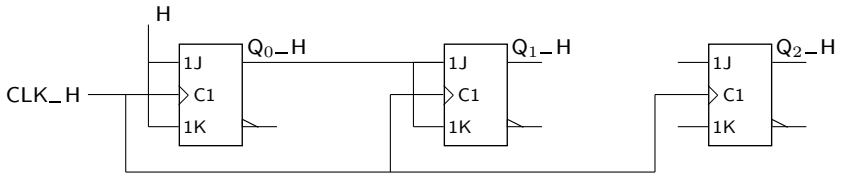


Figura 14.11: Fase intermédia de construção do contador síncrono módulo 8, que conta segundo o CBN

O controlo das entradas do terceiro flip-flop é mais complexo. De facto, a ideia rápida que pode surgir de que as entradas do terceiro devem ser ligadas à saída do segundo flip-flop está errada. De facto, não basta que a saída Q_1_H esteja activa (a H) para que Q_2_H mude. Por exemplo, se $(Q_2\_\mathrm{H}, Q_1\_\mathrm{H}, Q_0\_\mathrm{H}) = (\mathrm{L}, \mathrm{H}, \mathrm{L})$, o estado seguinte de contagem é (L,H,H) e, contudo, Q_2_H não se

altera. De facto, para Q_2_H se alterar terá de se ter $Q_1\_H = Q_0\_H = H$, como se pode observar na sequência de contagem proposta.

O circuito virá, então, com o logigrama da Figura 14.12.

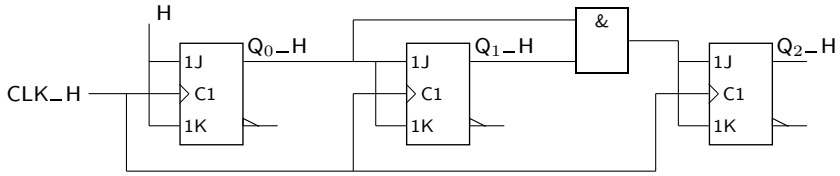


Figura 14.12: Logigrama final do contador síncrono módulo 8, que conta segundo o CBN

Repare-se que, *no caso dos contadores síncronos é irrelevante o flanco activo de relógio, ao contrário do que sucede com os contadores assíncronos*. O circuito é um contador ascendente qualquer que seja o flanco activo. Na realidade, o facto de o controlo da mudança dos flip-flops ser feito pelas entradas síncronas e não pelo relógio, leva a que o flanco deste perca relevância (excepto para definir o instante em que se dá a mudança).

E óbvio que, para um número maior de bits de contagem, para contagens em binário natural, se terá sempre $J_i = K_i = Q_0\, Q_1 \cdots Q_{i-1}$. Este produto lógico pode ser implementado de duas formas: (i) como produto de i entradas; ou (ii) como um produto de duas entradas, com $J_i = K_i = J_{i-1} \cdot Q_{i-1}$. Neste último caso usam-se ANDs apenas com 2 entradas mas, em contrapartida, o circuito fica mais lento, uma vez que as mudanças nas saídas dos flip-flops têm, no pior caso, de se propagar por $i-1$ portas AND para as entradas ficarem estabilizadas.

Na Figura 14.13 ilustra-se esta solução alternativa para um contador síncrono binário de módulo 16.

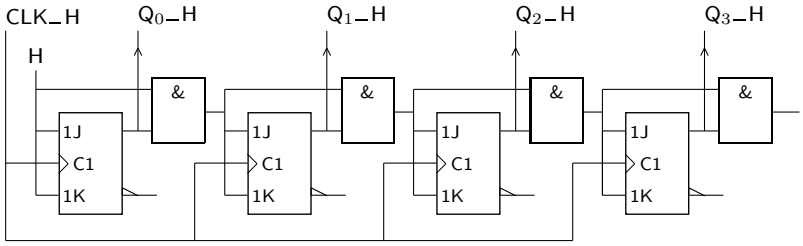


Figura 14.13: Um contador síncrono binário de módulo 16, que utiliza portas AND com 2 entradas para formar as equações lógicas dos J_i e K_i (excepto, naturalmente para J_0 e K_0, que são ligadas a H). Note-se que a porta AND que gera J_1 e K_1 não era estritamente necessária, já que $J_1 = K_1 = Q_0$, mas foi incluída para realçar o padrão de formação dos J_i e dos K_i

Os contadores síncronos (como, aliás, também os assíncronos, como vimos atrás), podem ainda contar com uma entrada de Reset assíncrona que permite iniciá-los no estado 0, utilizando para tanto as entradas assíncronas dos flip-flops (Figura 14.14).

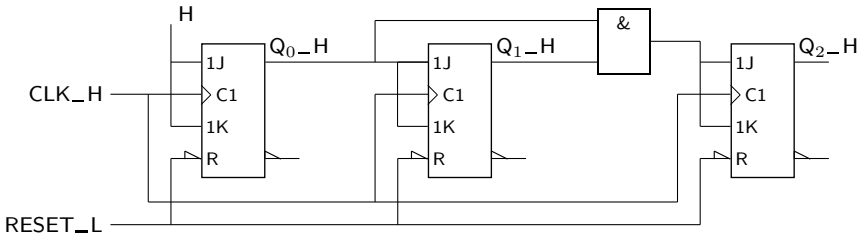


Figura 14.14: Contador síncrono binário de módulo 8, com entrada de Reset assíncrona

14.2.2 Contadores síncronos com entrada de Enable

Muitas vezes são necessários contadores com uma **entrada de Enable** ou de **Modo** que controla o contador, impondo-lhe um modo de contagem quando recebe impulsos de relógio ou, pelo contrário, mantendo o estado sem alteração. Isso é fácil de concretizar alterando um pouco a lógica das entradas J e K, como ilustra a Figura 14.15.

Entrada de Enable (Modo)

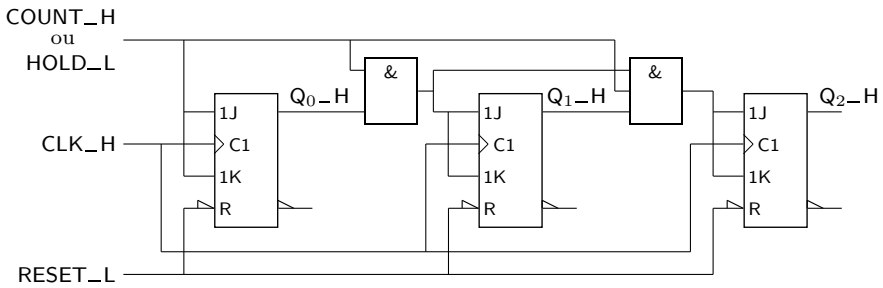


Figura 14.15: Contador síncrono binário de módulo 8, com entrada de Reset assíncrona e entrada síncrona de Enable (Modo), $COUNT\_H$ ou $HOLD\_L$

Repare-se que, se a variável $COUNT$ estiver activa (a H), a funcionalidade do contador é a que temos vindo a descrever. Se, pelo contrário, $COUNT$ estiver inactiva, todas as entradas J e K ficam a L, e os flip-flops não alteram o seu estado, ficando portanto bloqueado o valor da contagem.

Por esta razão, podemos designar essa entrada por $HOLD$ em vez de $COUNT$, desde que simultaneamente lhe troquemos a polaridade. Ou seja, em vez de $COUNT\_H$ podemos ter $HOLD\_L$.

14.2.3 Concepção de contadores síncronos de módulo qualquer

A concepção de qualquer contador síncrono pode sempre ser feita utilizando um procedimento sistemático de síntese, que será mais tarde retomado no estudo dos circuitos sequenciais síncronos em geral.

Para exemplificar, aborda-se aqui o problema de conceber um contador de módulo 6 que conta segundo o CBN e que utiliza palavras de comprimento 3, contando de 0 a 5.

Tabela de transições

Começa-se por definir, numa **tabela de transições**, o estado para que evolui o contador a partir de um outro estado.

Estado actual (presente) e estado seguinte

Teremos assim, a tabela de transições da Tabela 14.5, em que a coluna da esquerda designa o **estado actual** ou **estado presente** do contador, EA, e a coluna da direita identifica o seu **estado seguinte**, ES.

Tabela de transições de um contador síncrono módulo 6

Tabela 14.5: Tabela de transições de um contador de módulo 6 que utiliza o CBN com palavras de comprimento 3, contando de 0 a 5

EA			**ES**		
$Q2\_H_{(t)}$	$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$Q2\_H_{(t+1)}$	$Q1\_H_{(t+1)}$	$Q0\_H_{(t+1)}$
L	L	L	L	L	H
L	L	H	L	H	L
L	H	L	L	H	H
L	H	H	H	L	L
H	L	L	H	L	H
H	L	H	L	L	L

O facto da síntese com flip-flops JK produzir, muitas vezes, um logigrama mais simples, reside no facto de estes flip-flops terem 4 modos de funcionamento (contra 2 ou 3 para os outros flip-flops que estudámos), o que em geral permite simplificar as equações lógicas das entradas síncronas. Contudo, os flip-flops JK necessitam de equações de excitação para as entradas J e K, enquanto que, por exemplo, os flip-flops D apenas necessitam de equações de excitação para uma entrada D.

Repare-se que só colocamos na tabela os estados correspondentes à contagem pretendida para o contador. De facto, com três flip-flops podia-se codificar, como aliás se viu atrás, os estados 6 e 7. Mas como esses valores ficam fora do ciclo de contagem do contador, não são utilizados.

O contador, como qualquer circuito síncrono, pode ser implementado usando flip-flops de qualquer tipo. Usualmente á mais fácil trabalhar com flip-flops JK a este nível. O circuito usará, assim, três flip-flops do tipo JK, um para cada bit, e começará, como anteriormente, pelo pré-esquema da Figura 14.16.

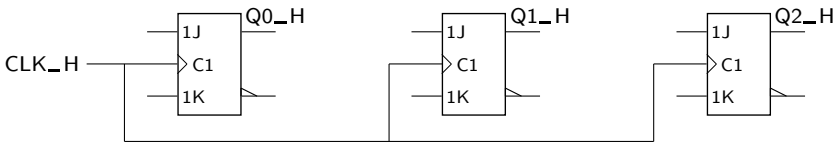


Figura 14.16: Primeira fase de síntese de um contador síncrono de módulo 6 que conta de 0 a 5 no CBN

Equações de excitação

A tarefa de projectar o circuito resume-se a decidir que funções lógicas utilizar para atacar as entradas J_H e K_H de cada um dos flip-flops (as chamadas **equações de excitação dos flip-flops**).

Repare-se que, se se pretender que um flip-flop JK mantenha o estado L ao receber um impulso de relógio, temos duas hipóteses: ou se faz $J\_H = K\_H = L$ e o flip-flop não muda de estado (modo de manutenção do estado), ou se coloca $J\_H = L$ e $K\_H = H$, que é uma ordem para assumir o estado L (modo de Reset), o que tem o mesmo efeito. Portanto, só é necessario garantir que $J\_H = L$, sendo indiferente o valor da entrada K_H.

Fazendo o mesmo tipo de raciocínio para todas as outras transições possíveis, obtém-se a Tabela 14.6, denominada **tabela de excitações de um flip-flop JK**.

Tabela de excitações de um flip-flop JK

Tabela 14.6: Tabela de excitações de um flip-flop JK

$Q\_H_{(t)}$	→	$Q\_H_{(t+1)}$	$J_{(t)}\,K_{(t)}$
L	→	L	L ×
L	→	H	H ×
H	→	L	× H
H	→	H	× L

Podemos em seguida obter uma **tabela de excitações do contador** substituindo, em cada ponto da tabela de estados, o nível de tensão que pretendemos que as saídas dos flip-flops assumam pelo nível de tensão a colocar nas suas entradas de excitação, ou entradas síncronas (Tabela 14.7).

Tabela de excitações do contador módulo 6

Tabela 14.7: Tabela de excitações do contador de módulo 6

Estado Actual					
$Q2\_H_{(t)}$	$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$J2\_H_{(t)}\,K2\_H_{(t)}$	$J1\_H_{(t)}\,K1\_H_{(t)}$	$J0\_H_{(t)}\,K0\_H_{(t)}$
L	L	L	L ×	L ×	H ×
L	L	H	L ×	H ×	× H
L	H	L	L ×	× L	H ×
L	H	H	H ×	× H	× H
H	L	L	× L	L ×	H ×
H	L	H	× H	L ×	× H

Reparemos que as tabelas de excitações do contador representam circuitos combinatórios, já que as excitações e as saídas dos flip-flops estão definidas no mesmo instante t.

Da tabela de excitações do contador podemos obter directamente os mapas de Karnaugh que definirão as expressões lógicas das entradas síncronas dos flip-flops.

Convém, contudo, notar-se que, por observação da tabela de excitações, se conclui directamente que $J0\_H = K0\_H = H$.

Por outro lado, nas posições correspondentes a valores não existentes da sequência de contagem são colocadas indiferenças nos mapas de Karnaugh, como é óbvio. Ou seja, os quadros de Karnaugh que irão permitir obter as equações de excitação dos flip-flops possuem indiferenças por dois tipos de razões: (i) as que resultam da tabela de excitações dos flip-flops; e (ii) as que resultam de estados de contagem não utilizados.

Obtemos, assim, os mapas da Figura 14.17, de onde se podem obter as seguintes equações de excitação dos flip-flops do contador,

$$\begin{array}{lll} J2 = Q1\,Q0 & J1 = Q0\,\overline{Q2} & J0 = 1 \\ K2 = Q0 & K1 = Q0 & K0 = 1\,, \end{array}$$

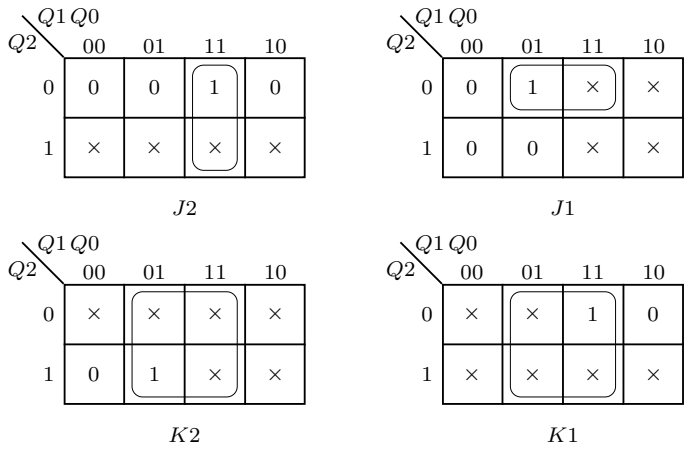


Figura 14.17: Quadros de Karnaugh de onde se podem extrair as equações de excitação dos flip-flops do contador síncrono de módulo 6

e o circuito final terá o logigrama da Figura 14.18, admitindo uma implementação com flip-flops edge-triggered que comutam no flanco ascendente.

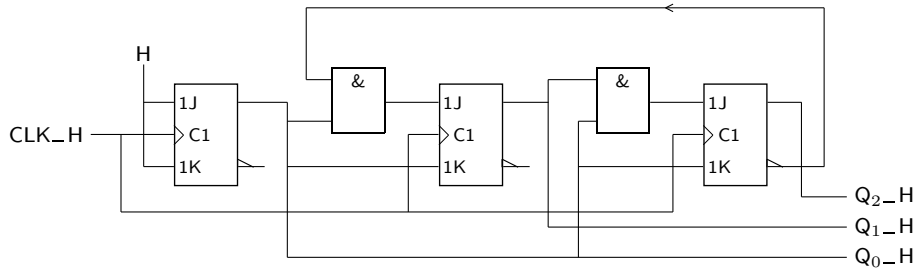


Figura 14.18: Logigrama do contador síncrono de módulo 6, que conta de 0 a 5 no CBN

14.2.4 Contadores síncronos com vários modos de funcionamento

É possível projectar contadores com vários modos de funcionamento. Já se viu anteriormente o caso de um contador com uma entrada de Modo que permite que ele conte ou que mantenha o estado. O método geral de projectar contadores desse tipo continua a ser a utilização das metodologias gerais de síntese dos circuitos sequenciais síncronos.

Contadores bidireccionais (ou ascendentes/descendentes ou "up/down")

Um exemplo comum é o dos contadores **bidireccionais**, também chamados **ascendentes/descendentes** ou **"up/down"**, que podem contar ascendentemente ou descendentemente, consoante o nível de uma variável de Modo. Na Tabela 14.8 apresenta-se a tabela de transições de um contador desse tipo de módulo 10.

A síntese do circuito seria feita sem dificuldade, com as adaptações necessárias para ter em conta que os "J"s e os "K"s são agora funções, não só das saídas dos flip-flops, como também da variável $UP/DOWN$.

Tabela 14.8: Tabela de transições de um contador binário ascendente/descendente de módulo 10 que conta de 0 a 9 ou de 9 a 0, consoante o nível de tensão numa entrada de Modo designada por $UP\_\mathrm{L}/DOWN\_\mathrm{H}$

EA				**ES**							
				UP_L/DOWN_H = L				UP_L/DOWN_H = H			
Q3_H	Q2_H	Q1_H	Q0_H	Q3_H	Q2_H	Q1_H	Q0_H	Q3_H	Q2_H	Q1_H	Q0_H
L	L	L	L	L	L	L	H	H	L	L	H
L	L	L	H	L	L	H	L	L	L	L	L
L	L	H	L	L	L	H	H	L	L	L	H
L	L	H	H	L	H	L	L	L	L	H	L
L	H	L	L	L	H	L	H	L	L	H	H
L	H	L	H	L	H	H	L	L	H	L	L
L	H	H	L	L	H	H	H	L	H	L	H
L	H	H	H	H	L	L	L	L	H	H	L
H	L	L	L	H	L	L	H	L	H	H	H
H	L	L	H	L	L	L	L	H	L	L	L

14.2.5 Contadores síncronos com carregamento em paralelo

Em muitas aplicações é importante utilizar contadores com a capacidade de realizar o carregamento de um valor pré-determinado e arbitrário, antes de proceder à contagem, de forma a iniciá-lo com esse valor.

Tal pode ser realizado se multiplexarmos a lógica normal de contagem com a lógica destinada a proceder a esse carregamento.

Exemplifica-se este procedimento na Figura 14.19 com a adição dessa possibilidade ao contador binário de módulo 8 com Enable anteriormente projectado na Figura 14.15, na página 243.

Para clareza da figura, rearrumam-se os flip-flops e adiciona-se uma linha de controlo, $LOAD\_\mathrm{H}$, que procede ao carregamento quando está activa e que permite os outros modos quando inactiva.

A lógica de $J0$ e $K0$ será determinada pela necessidade de fazer aparecer na saída $Q0\_\mathrm{H}$ o nível de tensão que aplicarmos à entrada $I0\_\mathrm{H}$ (ou seja, assegurar que $Q0\_\mathrm{H} = I0\_\mathrm{H}$) sempre que a linha $LOAD\_\mathrm{H}$ estiver a H, e por assegurar em $Q0\_\mathrm{H}$ o nível H (o da linha $COUNT\_\mathrm{H}$) no caso oposto.

$$J0 = LOAD{\cdot}I0 + COUNT$$
$$K0 = LOAD{\cdot}\overline{I0} + COUNT\,.$$

E de forma semelhante com as outras excitações dos flip-flops:

$$J1 = LOAD{\cdot}I1 + COUNT{\cdot}Q0$$
$$K1 = LOAD{\cdot}\overline{I1} + COUNT{\cdot}Q0$$

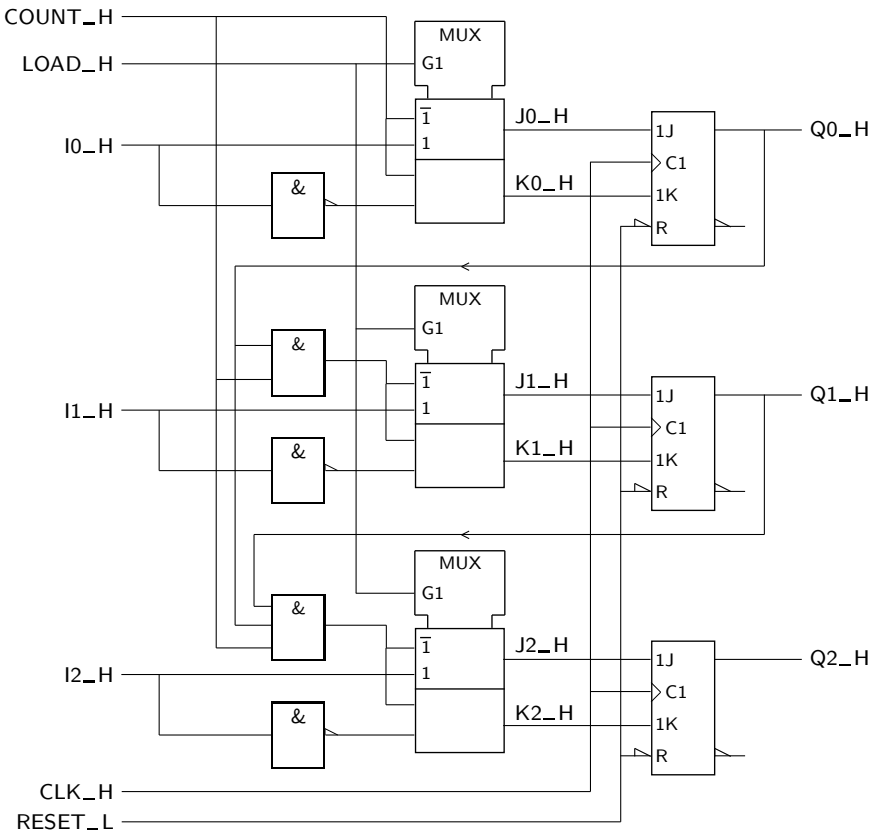


Figura 14.19: Circuito com três modos de funcionamento síncrono (carregamento em paralelo, contagem binária ascendente de módulo 8, e manutenção do estado), e um modo de funcionamento assíncrono (Reset)

e

$$J2 = LOAD{\cdot}I2 + COUNT{\cdot}Q1{\cdot}Q0$$
$$K2 = LOAD{\cdot}\overline{I2} + COUNT{\cdot}Q1{\cdot}Q0\,,$$

o que se pode obter, de forma compacta, usando 3 multiplexers.

Repare-se que um flip-flop JK, para ser carregado com o nível de tensão de uma linha $Ii_$H, terá de ter $Ji = Ii$ e $Ki = \overline{Ii}$.

Deve notar-se que as entradas de controlo não devem vir simultaneamente activas. Por isso, este contador possui 4 modos de funcionamento, a saber:

Modo de carregamento em paralelo

Entradas de carregamento em paralelo

— um **modo de carregamento em paralelo**, síncrono, do valor que estiver presente nas **entradas de carregamento em paralelo**, $I2_$H, $I1_$H e $I0_$H, se a entrada de Modo $LOAD_$H, e só ela, estiver activa (neste modo são escolhidas as entradas 1 dos multiplexers, ligadas a $I_i_$H e a $\overline{I_i}_$H); sendo o carregamento síncrono, ele só é efectivado quando ocorre um flanco de relógio ascendente, que é o flanco de comutação dos flip-flops utilizados;

Modo de contagem ascendente

— um **modo de contagem ascendente**, também síncrono, se a entrada de Modo

$COUNT_$H, e só ela, estiver activa [neste modo são escolhidas as entradas $\overline{1}$ dos multiplexers porque $LOAD_$H está inactiva, pelo que $(J0, K0)$ = (H,H), $(J1, K1) = (Q0, Q0)$ e $(J2, K2) = (Q1 \cdot Q0, Q1 \cdot Q0)$, como acontece para um contador binário síncrono de módulo 8]; como este modo é síncrono, a contagem só é efectivada quando ocorrem flancos de comutação dos flip-flops;

— um **modo de Reset**, assíncrono, quando a entrada de Modo $RESET_$L estiver activa (sendo assíncrono, o Reset é efectivado assim que a variável $RESET$ vem activa); e

Modo de Reset

— um **modo de manutenção do estado**, síncrono, quando nenhuma das entradas de Modo estiver activa [neste modo são escolhidas as entradas $\overline{1}$ dos multiplexers porque $LOAD_$H está inactiva, pelo que (Ji, Ki) = (L,L)].

Modo de manutenção do estado

14.3 Símbolos dos Contadores

A Figura 14.20 ilustra o símbolo IEC de um contador do tipo 74LS161A. Este símbolo é relativamente complexo, porém representativo dos contadores síncronos integrados que existem.

74LS161A

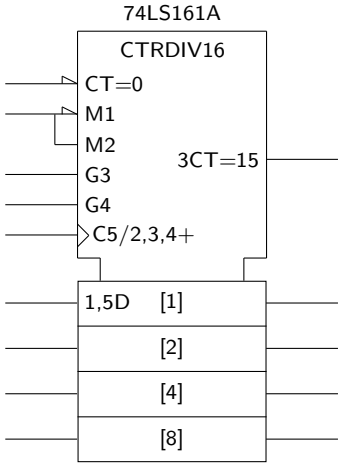


Figura 14.20: Símbolo IEC de um contador 74LS161A

Passemos à explicação do símbolo.

— CTRDIV16: é o **qualificador geral** do símbolo. Trata-se de um contador de 4 bits (CTR) que pode ser entendido como um divisor de frequência por 16 (DIV16), na medida em que conta segundo o CBN de $0_{(10)}$ a $15_{(10)}$.

Qualificador geral CTRDIV16

— Modos de funcionamento: há uma linha de Modo que define dois modos de funcionamento do contador (há outros modos, por exemplo de Reset, que não são controlados pelo nível de tensão nesta linha): o modo M1 ocorre quando a linha de Modo está a L, e o modo M2 quando a linha está a H (os qualificadores 1 e 2 são convencionais, e apenas servem para "ligar" a linha de Modo a outras linhas que utilizem estes qualificadores e que, por

Modos de funcionamento

esse facto, dependem desta, como é o caso das linhas com os qualificadores de entrada C5/2,3,4+ e 1,5D).

Modo M1 (carregamento em paralelo)

— O **modo** M1 é o **modo de carregamento em paralelo** e corresponde ao carregamento em paralelo síncrono do contador, como veremos já a seguir.

Modo M2 (contagem ascendente)

— O **modo** M2 é o **modo de contagem** e corresponde, como também veremos mais à frente, à contagem normal, ascendente, do contador.

Qualificador de entrada C5

Dependência de Controlo (C)

Qualificadores de entrada ▷ e ⌙

— Linha de relógio: o qualificador de entrada C5 configura uma **dependência de controlo**, C, que condiciona a acção das entradas que possuem o qualificador 5. Associado a C5 existe um qualificador de entrada ▷ na linha de relógio que indica, como habitualmente, que os flip-flops do contador possuem uma estrutura do tipo edge-triggered. Por seu turno, a ausência de um triângulo exterior à entrada de relógio, orientado no sentido do fluxo dos sinais (qualificador de entrada ⌙), indica que os flip-flops comutam nos flancos ascendentes dos impulsos de relógio.

Qualificadores de entrada G3 e G4

Dependência And (G)

— G3 e G4: são qualificadores de entrada que designam dependências And com efeito sobre as entradas ou saídas que possuam os qualificadores 3 e 4 (é o caso da entrada de relógio com o qualificador 2,3,4+, e da saída com o qualificador 3CT=15). Trata-se, naturalmente, de dois de Enables de contagem e de um Enable da saída 3CT=15.

Qualificador de entrada 2,3,4+

— 2,3,4+: é um qualificador de entrada que indica que, ao satisfazerem-se as condições 2, 3 e 4, o contador conta ascendentemente (o qualificador + significa contagem ascendente e o qualificador −, se existisse, significaria contagem descendente). Como podemos constatar, este qualificador está associado ao qualificador C5 (com efeito, ter C5/2,3,4+ é o mesmo que ter uma linha de entrada com dois qualificadores, o qualificador C5 e o qualificador 2,3,4+). Por sua vez, C5 está ligado ao qualificador ▷, como vimos anteriormente. Logo, existe contagem ascendente se ocorrer um flanco ascendente na linha de relógio, se o modo M2 estiver activo (entrada de Modo a H) e se as linhas de Enable G3 e G4 estiverem ambas activas.

Qualificador de entrada CT=0

Modo de Reset

Reset assíncrono

— CT=0: é um qualificador de entrada que identifica o **Modo de Reset** assíncrono do contador (CT significa "Count"). Quando a linha vem activada (a L), o contador vem de imediato para o estado $0_{(10)}$, sem auxílio de um flanco de relógio (por isso o Reset é assíncrono). O Reset seria síncrono se, em vez de CT=0, tivessemos este qualificador a depender de C5 na forma 5CT=0.

Qualificador de entrada 1,5D

Valor inicial

Carregamento em paralelo síncrono

— 1,5D: é um qualificador de entrada que identifica em que condições se faz o carregamento em paralelo dos 4 flip-flops do contador (como sabemos, o carregamento em paralelo do contador permite fornecer-lhe um **valor inicial** de contagem). O carregamento dá-se quando a linha de Modo está a L (Modo M1) e quando surge um flanco activo na entrada de relógio (C5). Trata-se, portanto, de um **carregamento em paralelo síncrono** (se o contador tivesse, por exemplo, o qualificador de entrada 1D em vez do qualificador 1,5D, isso significava que o carregamento seria **assíncrono**, isto é, que se verificava logo que a linha de modo viesse activada, independentemente da existência ou não de flancos de comutação na linha de relógio; esse tipo de carregamento vem implementado em vários contadores, síncronos e assíncronos, mas não no 74LS161A).

— 3CT=15: trata-se de um qualificador de saída que indica que a contagem do contador atingiu o estado 15 (relembrar que CT significa "Count"). Esta saída vem activada durante todo o estado 15 de contagem se o Enable G3 estiver activo (dependência And). *Qualificador de saída 3CT=15*

— [1], [2], [4] e [8]: os qualificadores entre parêntesis rectos indicam **comentários** que, não sendo estritamente necessários, ajudam contudo a interpretar o símbolo. Neste caso indicam os pesos dos diversos flip-flops no processo de contagem. *Comentários* *Pesos dos flip-flops*

14.4 Estados Instáveis

Quando se estudaram os contadores assíncronos chamou-se a atenção para a existência de estados instáveis que resultam do funcionamento dessas estruturas. Como sabemos, esses estados evoluem entre si até se atingir um estado estável, e não podem ser evitados uma vez que resultam, como se disse, do próprio mecanismo de funcionamento daqueles contadores.

Numa primeira análise ao caso dos contadores síncronos, pode resultar a ideia errada que não existem, neste caso, estados instáveis. De facto, como todos os flip-flops recebem simultaneamente o impulso de relógio que provoca uma dada transição, e como cada flip-flop só tem, no máximo, uma transição (ou muda de estado ou mantém o estado) para cada impulso, pode ser dado o passo suplementar de admitir que todos os flip-flops que alteram o estado o fazem simultaneamente. Não é assim, uma vez que os tempos de propagação de cada flip-flop dependem de muitos factores e, portanto, não são determináveis em absoluto. Sabemos os intervalos de variação, mas não os valores absolutos dos tempos de propagação.

Daqui resulta que os diversos flip-flops podem reagir em tempos diferentes, desde que dentro das suas especificações. Por exemplo, no flip-flop 74LS76A os fabricantes especificam um **tempo de propagação máximo** de 20 ns, sendo 15 ns o **tempo de propagação típico**. É, assim, possível que num determinado contador haja flip-flops a reagir em 20 ns e outros em 15 ns, e até alguns em menos tempo que isso. Daí resulta o aparecimento de estados instáveis. *74LS76A* *Tempos máximo e típico de propagação*

Na Figura 14.21 ilustra-se um exemplo dessa situação, para um contador síncrono com 4 flip-flops, na passagem do estado estável **7** para o estado estável **8** (QA_H é a saída do flip-flop menos significativo).

Como se pode observar, no processo de transitar do estado **7** para o estado **8** o contador vai passar, neste exemplo, pelos estados instáveis *7*, *15*, *13*, *12* e, finalmente, *8*. É evidente que seria possível obter-se uma qualquer outra sequência de estados instáveis, obtidos por transição de apenas um flip-flop de cada vez por uma ordem diferente da que se apresenta na figura. Existem, portanto, estados instáveis também em contadores síncronos. Há, contudo, diferenças entre os estados instáveis nos contadores assíncronos e nos contadores síncronos:

— *duração*: os estados instáveis em contadores síncronos têm durações da ordem de grandeza da *diferença* entre os tempos de propagação dos flip-flops, en-

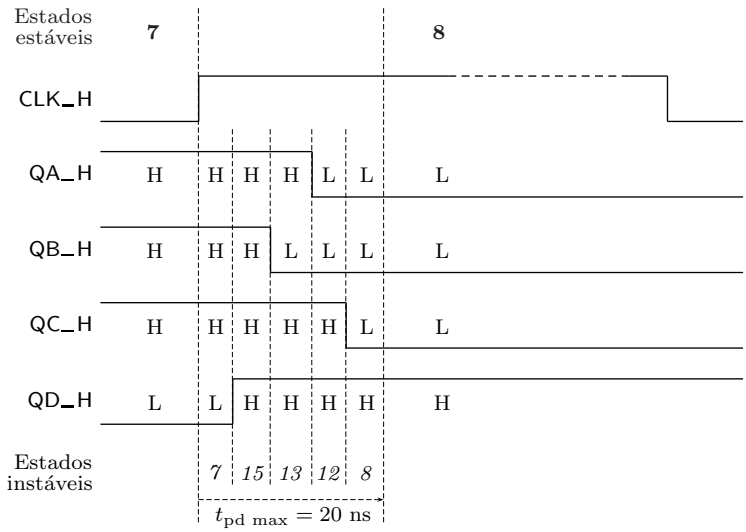


Figura 14.21: Transição entre o estado estável de contagem **7** e o estado estável **8**, num contador síncrono

quanto que nos contadores assíncronos essa duração é da ordem de grandeza dos próprios tempos de de propagação.

— *previsibilidade*: nos contadores assíncronos pode-se prever a exacta sequência de estados instáveis, uma vez que essa sequência está relacionada exactamente com o mecanismo de contagem. No caso dos contadores síncronos, tal não pode ser feito. Sabemos que existem estados instáveis, mas não sabemos a ordem por que aparecem.

— *tipo de estados*: nos contadores assíncronos ascendentes, os estados instáveis são sempre estados de contagem com valores inferiores ao do do estado de partida. Por exemplo, na transição de **7** para **8** os estados instáveis são *6*, *4* e *0*, por esta ordem. No caso dos contadores síncronos os estados instáveis, como se pode ver na Figura 14.21, *podem* ter valores de contagem superiores ao do estado de partida.

14.5 Interligação de Contadores

Muitas vezes é necessário proceder à interligação entre vários contadores para aumentar o módulo de contagem. Uma solução óbvia consiste em interligá-los de modo a que um dos contadores seja o menos significativo e os seguintes contem apenas quando o primeiro chega ao último estado de contagem simultaneamente com a passagem deste para o primeiro estado.

A solução, para manter o carácter síncrono da interligação, passa por ligar os contadores de forma a que o relógio seja comum e a que o Enable de contagem de cada um dependa do conjunto dos contadores menos significativos terem chegado ao último estado de contagem. Um exemplo desse tipo de ligação encontra-se na Figura 14.22 para um um contador síncrono de módulo 256.

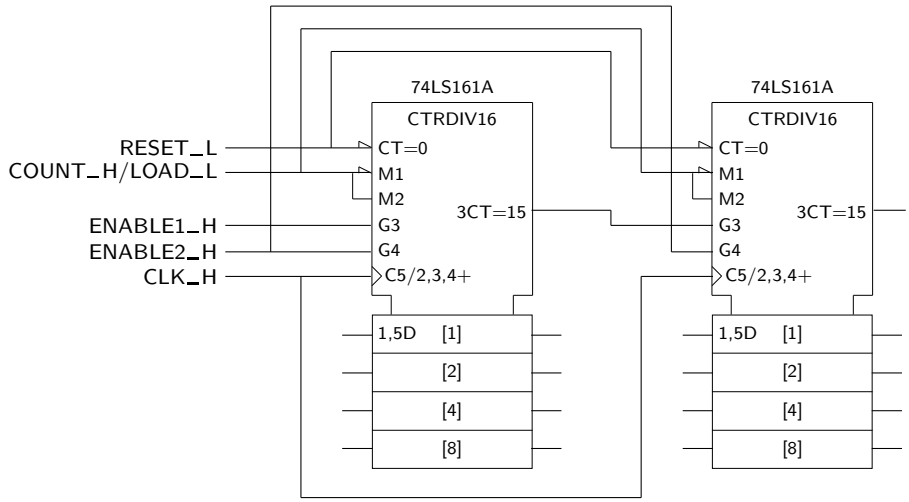


Figura 14.22: Contador síncrono de módulo 256 implementado por ligação síncrona de dois contadores síncronos de módulo 16

14.6 Utilização do Carregamento em Paralelo na Realização de Diferentes Módulos de Contagem

Naturalmente, podemos sempre sintetizar este tipo de contadores da forma que aprendemos na Subsecção 14.2.3.

O carregamento em paralelo dos contadores tem outra utilidade, que é a de permitir modificar os módulos de contagem.

Década

Suponhamos que pretendemos utilizar um contador de módulo 16, como o que se utilizou no exemplo anterior, e transformá-lo num contador de módulo 10 (**divisor de frequência por 10** ou **década**). A solução consiste em detectar o último estado de contagem pretendido e usar essa linha para activar o carregamento em paralelo do contador. As linhas de carregamento em paralelo serão utilizadas para impor o estado de contagem $0_{(10)}$. Note-se que não pode ser usado o Reset, uma vez que ele é assíncrono e, supostamente, pretendemos manter o contador síncrono.

Na Figura 14.23 ilustra-se o logigrama deste contador modificado.

Repare-se que não é necessário que a porta lógica que descodifica (ou detecta) o estado 9 tome em linha de conta os níveis a L à saída do contador. De facto, a primeira vez que, na sequência de contagem, se encontram os dois níveis dos extremos a H, é quando o contador chega ao estado $9_{(10)}$. Assim, se detectarmos apenas essa situação, detectamos o estado $9_{(10)}$, uma vez que os estados seguintes não vão aparecer.

De igual modo se poderia ter usado outro valor para carregar nas entradas de carregamento em paralelo, se a sequência pretendida a isso obrigasse. Se se pretendesse, por exemplo, a sequência de contagem decimal

$$\cdots, 4, 5, 6, 7, 8, 9, 4, \cdots,$$

então detectar-se-ia o estado $9_{(10)}$ da mesma forma, mas em vez de carregar o número $0_{(10)}$ carregar-se-ia o número $4_{(10)}$.

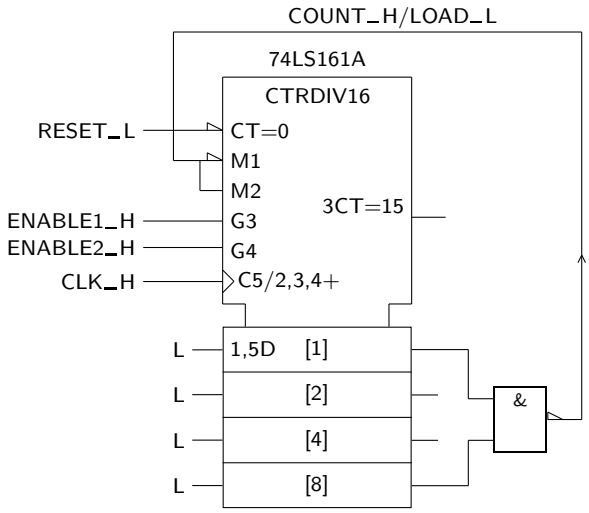


Figura 14.23: Contador síncrono de módulo 10 construído com base num contador do mesmo tipo mas de módulo 16

14.7 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secções 5.4, 5.7 e 6.3 a 6.5.

14.8 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

14.1 a) Construir um contador assíncrono *descendente* de módulo 8 usando os flip-flops e portas que entender.

b) Diga quais são os estados instáveis por que o contador passa na transição do estado estável **4** para o estado estável **3**.

c) A solução de ligação entre os flip-flops que escolheu é a única? No caso contrário, diga quais as formas alternativas de efectuar essas ligações.

(∗) 14.2 Considere o logigrama da Figura 14.24, constituído por um contador binário assíncrono e por um multiplexer com 8 entradas de dados.

a) Desenhe o diagrama temporal da saída F_H quando uma sequência de 10 impulsos é aplicada à entrada CLK_H. Suponha que inicialmente o contador tem o valor de contagem $0_{(10)}$. Não dê relevo à existência de estados instáveis nem de atrasos nos circuitos.

b) Entrando agora em conta com a existência de estados instáveis e de atrasos nos circuitos, desenhe o diagrama temporal pormenorizado — que inclua as variáveis $I2$, $I1$, $I0$ e a função F — das transições resultantes da passagem do estado de contagem $3_{(10)}$ para o estado de contagem $4_{(10)}$.

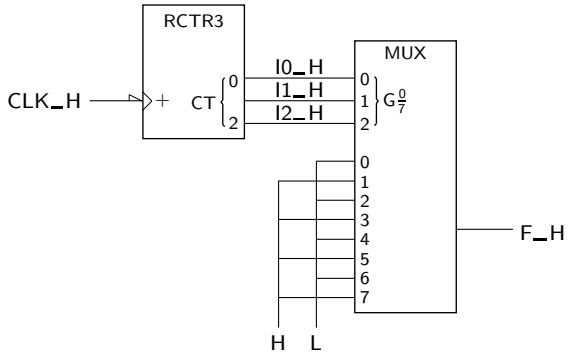


Figura 14.24: Logigrama do circuito do Exercício 14.2

14.3 Considere o contador assíncrono da Figura 14.25.

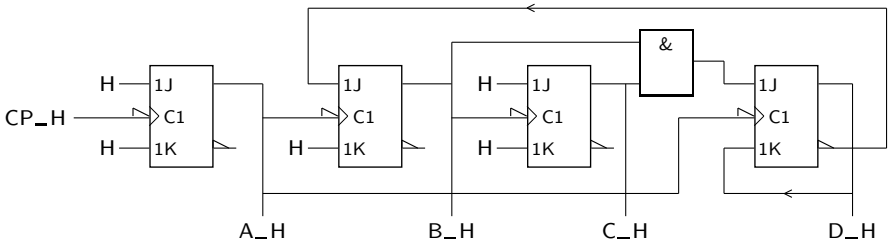


Figura 14.25: Logigrama do contador assíncrono do Exercício 14.3

a) Determine o módulo de contagem e desenhe o diagrama temporal das saídas A_H, B_H, C_H e D_H, em função da linha CP_H.

b) Qual é a frequência máxima de funcionamento do contador, descrita em função dos tempos relevantes dos componentes utilizados?

(∗) 14.4 Estabelecer o símbolo IEC do contador assíncrono integrado 74HCT393, fabricado em tecnologia HCMOS compatível com TTL. Esse contador é constituído por dois contadores/divisores de frequência independentes e idênticos, cada um de módulo 16 e com uma entrada de Reset activa a H. Os contadores contam ascendentemente a cada flanco descendente dos impulsos aplicados às suas entradas de relógio. Os flip-flops utilizadas nos contadores são do tipo edge-triggered. *74HCT393*

(∗) 14.5 Estabelecer o símbolo IEC do contador assíncrono integrado 74HC4024, fabricado em tecnologia HCMOS. Trata-se de um contador/divisor de frequência por 128, com uma entrada de Reset activa a H. O contador conta ascendentemente a cada flanco descendente dos impulsos aplicados à entrada de relógio. Os flip-flops utilizadas nos contadores são do tipo edge-triggered. *74HC4024*

14.6 a) Utilizando flip-flops JK edge-triggered que comutam nos flancos descendentes, desenhe o logigrama de um contador síncrono módulo 8 ou módulo 6, consoante o valor de uma variável de controlo, C_H, tenha o nível H ou L, respectivamente.

b) Modifique o circuito da alínea anterior por forma a que, para além do funcionamento aí descrito, o contador possa também aceitar um carregamento em paralelo do exterior quando uma segunda variável de controlo, PE_H, estiver activa.

14.7 Projecte um contador síncrono módulo 7, usando os flip-flops que entender.

14.8 Considere o circuito da Figura 14.26.

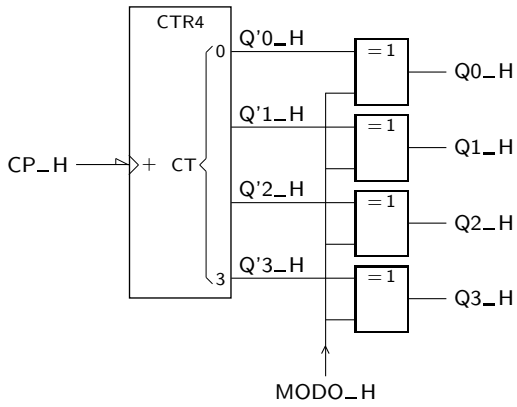


Figura 14.26: Circuito do Exercício 14.8

a) Estabeleça as sequências de contagem nas saídas $Q3\_H$ a $Q0\_H$ que se obtêm quando a variável $MODO$ está activa e inactiva, respectivamente.

b) Como poderia designar este circuito? Essa designação está inteiramente correcta?

14.9 Utilizando flip-flops do tipo D, projecte um contador "up/down" síncrono de módulo 6 cuja contagem varie entre $1_{(10)}$ e $6_{(10)}$ do CBN.

14.10 Utilizando o método clássico de síntese da Subsecção 14.2.3, projecte um contador síncrono com a seguinte sequência de contagem:

$$\cdots, 0, 3, 7, 2, 5, 0, \cdots.$$

Pode utilizar os flip-flops que desejar.

14.11 a) Utilizando o método clássico de síntese da Subsecção 14.2.3, projecte um contador síncrono com a seguinte sequência de contagem:

$$\cdots, 6, 0, 2, 3, 5, 6, \cdots.$$

O referido contador deve ter a possibilidade de ser incializado assincronamente com o número 6.

b) Quais os factores que limitam a frequência máxima de contagem do contador que acabou de projectar?

14.12 Utilizando o método clássico de síntese da Subsecção 14.2.3, projecte um contador síncrono com a seguinte sequência de contagem:

$$\cdots, 0, 1, 2, 3, 7, 9, 10, 12, 0, \cdots.$$

Pode utilizar os flip-flops que desejar. O contador deve ainda possuir uma linha PE_H que, quando activada, permite o carregamento em paralelo do contador e, quando inactiva, permite a contagem. Indique se optou por um carregamento em paralelo síncrono ou assíncrono, e justifique.

14.13 a) Utilizando flip-flops JK edge-triggered que comutam nos flancos descendentes, desenhe o logigrama de um contador síncrono módulo 8 ou 6, conforme o valor de uma variável de controlo, C, estiver activa ou inactiva, respectivamente. Justifique a escolha que fez do nível de actividade desta variável.

b) Modifique o circuito da alínea anterior de forma a que, para além do funcionamento lá descrito, o contador possa também fazer o carregamento em paralelo do exterior quando uma segunda variável de controlo, PE, estiver activa. Justifique a escolha que fez do nível de actividade desta variável.

14.14 Desenhe o logigrama de um contador síncrono que possa contar módulo 16, módulo 8 ou carregar em paralelo do exterior. Dispõe, para o efeito, de flip-flops D edge-triggered que comutam nos flancos ascendentes. Os modos de funcionamento deste contador são controlados pelas variáveis $C1$ e $C2$ da forma que se indica na Tabela 14.9.

Tabela 14.9: Tabela com os modos de funcionamento do contador do Exercício 14.14

C1_H	C0_H	Modo
L	L	Carregamento em paralelo
L	H	Contagem módulo 8
H	L	Contagem módulo 16

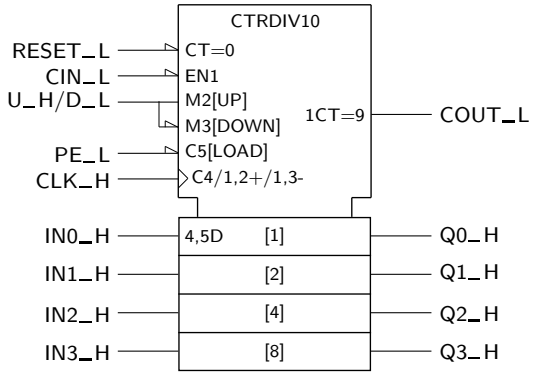
14.15 Desenhar, justificando, o símbolo IEC do 74LS192, um contador síncrono integrado com as seguintes características: trata-se de uma década (divisor de frequência por 10) que conta ascendente (de 0 a 9) ou descendentemente (de 9 a 0) no CBN, com entradas de relógio independentes (a selecção do tipo de contagem é feita pela aplicação de impulsos de relógio à entrada de relógio adequada). Os flip-flops são do tipo edge-triggered e comutam nos flancos ascendentes. O contador permite carregamento em paralelo assíncrono com uma entrada de carregamento activa a L dedicada a esse efeito. Existe ainda uma entrada de Reset assíncrona activa a H. Finalmente, o contador possui duas saídas activas a L que vêm activadas nas seguintes condições: uma delas quando o contador está a contar ascendentemente e passa pelo estado 9; a outra quando o contador está a contar descendentemente e passa pelo estado 0. Em qualquer dos casos, estas saídas só vêm activadas durante a fracção de cada impulso de relógio em que estas condições estão activas e o impulso está a L. *74LS192*

14.16 Dispondo de contadores módulo 8 com entrada de carregamento em paralelo PE_H (que permite seleccionar entre carregamento em paralelo quando activa, e contagem quando inactiva), construa:

a) um contador módulo 10;

b) um contador módulo 100.

14.17 São dados contadores do tipo ilustrado na Figura 14.27.



CIN_L	U_H/D_L	PE_L	Modo
H	×	H	Manutenção
L	H	H	Contagem ascendente
L	L	H	Contagem descendente
×	×	L	Carregamento em paralelo

Figura 14.27: Contador utilizado no Exercício 14.17

a) Utilizando 3 destes contadores e mais a lógica adicional que entender necessária, projecte um contador decimal (década ou divisor de frequência por 10) ascendente de modo programável (até 999).

b) Diga que valores colocaria nas entradas de carregamento em paralelo dos contadores para que eles contassem módulo 123.

14.18 Repita o Exercício 14.10, mas utilizando agora um contador integrado 74LS161A (Figura 14.20).

14.19 Repita o Exercício 14.12, mas utilizando agora um contador integrado 74LS161A (Figura 14.20).

(∗) 14.20 Utilizando um contador como o da Figura 14.28, projecte um contador com a seguinte sequência de contagem:

$$\cdots, 0, 1, 2, 3, 4, 5, 6, 7, 0, \cdots.$$

Indique ainda se optou por um carregamento em paralelo síncrono ou assíncrono, e justifique.

14.21 Considere o contador da Figura 14.29(a).

a) Complete o diagrama temporal na parte (b) da figura.

b) Qual o tempo mínimo entre os dois primeiros flancos descendentes dos impulsos de relógio do diagrama temporal, em função dos parâmetros temporais que considerar relevantes?

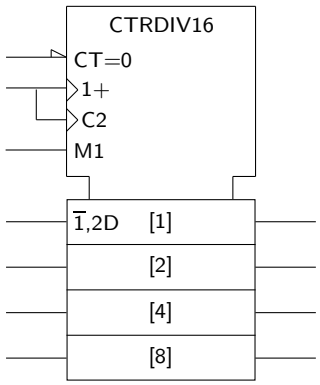


Figura 14.28: Contador utilizado no Exercício 14.20

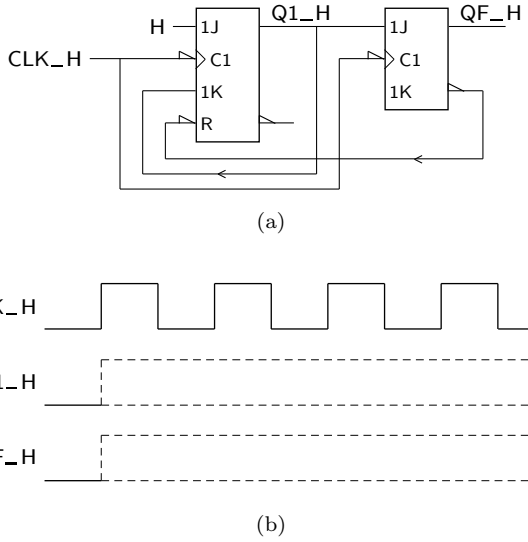


Figura 14.29: (a) Contador utilizado no Exercício 14.21; (b) diagrama temporal a completar

14.22 a) Usando dois contadores síncronos de 4 bits que especificará como achar conveniente, construa um contador síncrono módulo 200.

b) Refira qual a frequência máxima de contagem do novo contador, em função dos parâmetros temporais que achar conveniente.

14.23 Dispõe de dois contadores decimais síncronos como o da Figura 14.27.

a) Ligue 3 destes contadores para formar um contador módulo 1000.

b) Calcule a frequência máxima de funcionamento do contador módulo

1000 sabendo que:

$$t_{\text{pd,CLK}-\text{Q}} = 15 \text{ ns}$$
$$t_{\text{pd,CLK}-\text{COUT}} = 25 \text{ ns}$$
$$t_{\text{pd,CIN}-\text{COUT}} = 8 \text{ ns}$$
$$t_{\text{su,CIN}} = 15 \text{ ns}$$

14.24 São dados os seguintes tipos de flip-flops:

1. master-slave;
2. edge-triggered a comutar nos flancos ascendentes;
3. edge-triggered a comutar nos flancos descendentes, com

$$t_{\text{su}} = 10 \text{ ns}$$
$$t_{\text{h}} = 10 \text{ ns}$$
$$t_{\text{pd}} = 9 \text{ ns}$$

4. edge-triggered a comutar nos flancos descendentes, com

$$t_{\text{su}} = 10 \text{ ns}$$
$$t_{\text{h}} = 10 \text{ ns}$$
$$t_{\text{pd}} = 12 \text{ ns}$$

Considerando apenas os tipos de flip-flops indicados, quais de entre eles é que:

a) poderão ser utilizados no circuito da Figura 14.30?

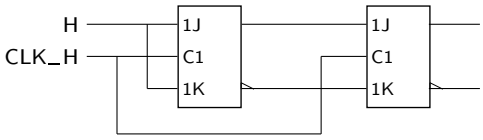


Figura 14.30: Circuito utilizado no Exercício 14.24

b) poderão ser utilizados no circuito, operando este à frequência de 50 MHz?

(∗) 14.25 Utilize um contador binário síncrono de módulo 16 com Reset e carregamento em paralelo síncronos, para realizar um contador com a sequência de contagem

$$\cdots, 0, 1, 2, 3, 4, 5, 0, 1, \cdots,$$

isto é, com módulo 6. Desenhe um diagrama temporal que mostre o funcionamento do contador. Modifique o logigrama que obteve para acomodar um contador integrado do tipo 74LS163A como o da Figura 14.31.

74LS163A

(∗) 14.26 Repetir o Exercício 14.25 mas utilizando agora um contador com Reset assíncrono.

(∗) 14.27 Utilizar um contador integrado do tipo 74LS163A para implementar uma década (contador com 10 estados), com sequência de contagem

$$\cdots, 6, 7, 8, 9, \cdots, 14, 15, 6, 7, \cdots.$$

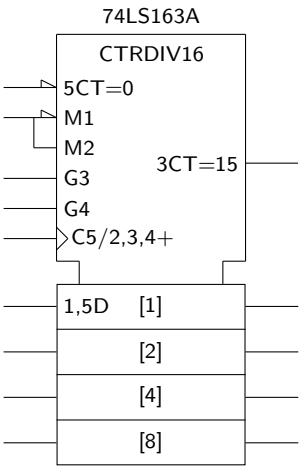


Figura 14.31: Símbolo IEC de um contador 74LS163A

(∗) 14.28 Utilizar um contador integrado 74LS163A para implementar a sequência de contagem

$$\cdots, 1, 2, 3, 6, 7, 8, 12, 13, 1, 2, \cdots .$$

Recorrer ao menor número possível de integrados para desenhar o esquema eléctrico do circuito que obtiver.

(∗) 14.29 Desenhar o diagrama de estados completo do contador do Exercício 14.25, que utiliza o contador integrado 74LS163A. O que acontece ao contador se ele for inicializado no estado $12_{(10)}$ quando o circuito vem alimentado electricamente?

14.30 Utilizar um contador integrado 74LS163A para implementar um contador síncrono que conte segundo o código Excesso de 3 (para a definição deste código, ver a página 151).

14.31 Utilizar dois contadores integrados do tipo 74LS163A para construir um contador síncrono de módulo 193 que conte segundo o CBN desde 63 até 255.

14.32 É dado o contador da Figura 14.32. Qual é a sua sequência de contagem?

14.33 Admita que, no exercício anterior, substituia o contador 74LS163A por um contador integrado 74LS161A como o da Figura 14.20. O que aconteceria ao contador?

(∗) 14.34 O contador da Figura 14.33 usa um contador integrado do tipo 74x169. Qual é a sequência de contagem do contador? *74x169*

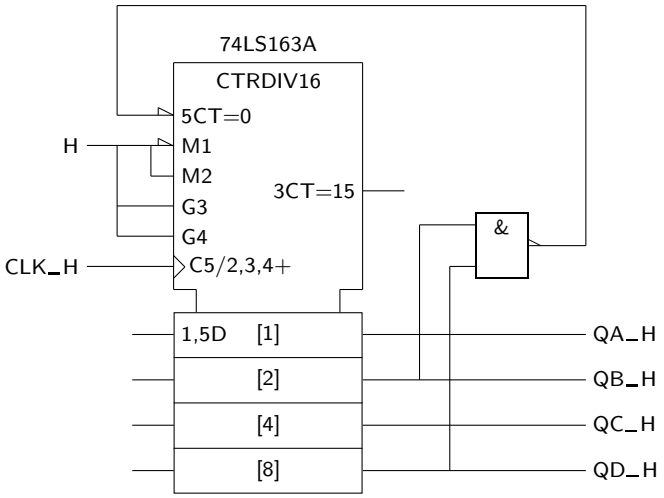


Figura 14.32: Contador utilizado no Exercício 14.32

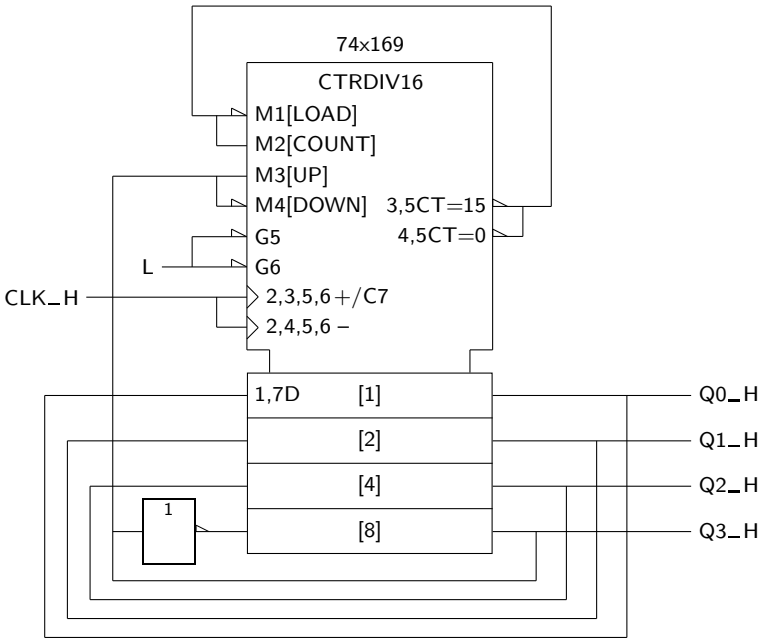


Figura 14.33: Contador utilizado no Exercício 14.34

Capítulo 15

Registos

15.1 Conceito de Registo

Um registo é um circuito capaz de memorizar um certo número de bits, o que constitui uma unidade de informação em si: por exemplo um carácter ASCII com 7 bits, um número inteiro binário de 16 bits, ou qualquer outro tipo de informação.

Basicamente, o registo deve poder tratar essa informação globalmente, isto é, como um todo, e não bit a bit.

15.2 Registos Simples

Um registo simples de n bits é constituído por n flip-flops, em geral do tipo D, com uma entrada de relógio comum. Cada um dos flip-flops constitui 1 **andar** do registo. Nessas condições, as entradas do registo são lidas e registadas simultaneamente, e apresentadas globalmente à saída, como mostra a Figura 15.1.

Andar (de um registo)

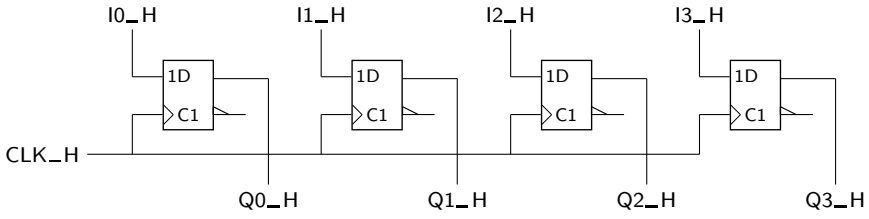


Figura 15.1: Registo simples com 4 andares

Repare-se que a estrutura é muito simples, sendo a interligação entre os diversos flip-flops (andares) apenas garantida pela linha de relógio. Como se pode perceber, trata-se de um circuito sequencial síncrono, o que é comum a todos os tipos de registo que iremos considerar.

A palavra $I0\,I1\,I2\,I3$ é tratada globalmente, sem nenhuma especificidade ligada a cada bit.

Uma variante comum deste registo assenta na inclusão de um controlo de apagamento (Reset ou Clear) da informação registada, utilizando, para isso, as linhas de Reset assíncronas dos flip-flops, como se mostra na Figura 15.2.

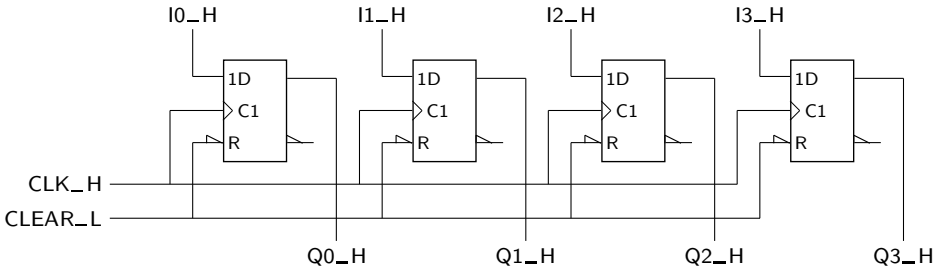


Figura 15.2: Registo simples com 4 andares e Reset assíncrono

Símbolo de um registo simples

O símbolo IEC deste registo vem ilustrado na Figura 15.3(a).

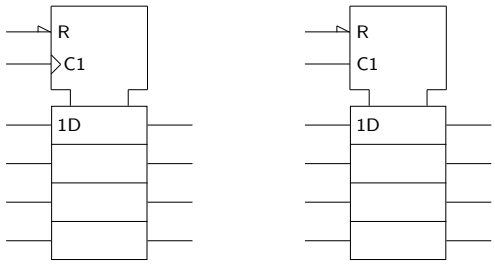


Figura 15.3: (a) Símbolo IEC de um registo simples com 4 andares formados por flip-flops edge-triggered e Reset assíncrono; e (b) símbolo IEC de um registo idêntico mas formado por latches D

Qualificadores de entrada R e C1

Como cada um dos 4 andares tem um tratamento semelhante, o simbolo exibe quatro módulos iguais que correspondem aos 4 flip-flops. No bloco de controlo comum colocado no topo incluem-se dois qualificadores de entrada já conhecidos, o qualificador R, que indica uma dependência Reset, e o qualificador C1, que é um exemplo de dependência de Controlo.

Repare-se que os qualificadores de entrada dos 4 flip-flops são iguais, mas que apenas se ilustra o qualificador do flip-flop superior, admitindo-se que o flip-flop por debaixo dele "herda" os qualificadores do primeiro flip-flop, que o flip-flop que se encontra na terceira posição "herda" os qualificadores do segundo, etc. Com isto a norma IEC pretende melhorar a legibilidade do símbolo.

De notar que o carregamento em paralelo do registo é síncrono, uma vez que os qualificadores 1D de entrada dos flip-flops dependem, através do símbolo 1, dos impulsos de relógio aplicados a C1.

Uma questão que se pode levantar é a de saber se os registos têm de ser construídos com flip-flops edge-triggered ou se podem ser construidos com outro tipo de dispositivos de memória (latches ou flip-flops). Desde logo não há qualquer problema na utilização de flip-flops master-slave, para além dos cuidados a ter em qualquer utilização desse tipo de dispositivo.

O caso dos latches D é mais delicado. Para circuitos do tipo do ilustrado e do seguinte, não há qualquer problema. Existem registos formados por latches, como o da Figura 15.3(b), que são dispositivos bastante úteis. Se, porém, pensarmos nos registos de deslocamento que a seguir se apresentarão, o uso de latches D é absolutamente de excluir devido às suas características de transparência.

Uma outra questão é a de considerar a utilização de flip-flops JK. Nada há a opor, excepto o facto de que os circuitos ficarão porventura mais complexos, uma vez que um flip-flop desse tipo não permite guardar directamente o nível de tensão presente numa linha, como sucede com os flip-flops D.

15.3 Registos com Enable

Por vezes temos necessidade de utilizar registos com Enable, isto é, com uma entrada que possibilite, quando ocorre um determinado flanco de um impulso de relógio, optar entre manter a informação armazenada no registo ou registar novos dados.

Tal faz-se de forma muito óbvia recorrendo a um multiplexer à entrada de cada flip-flop, que permita escolher se o que se carrega em cada flip-flop é um dado vindo do exterior ou, pelo contrário, o seu próprio conteúdo, mantendo, assim a informação inalterada. Ou seja, obtém-se a solução descrita na Figura 15.4.

Entrada de Modo

Notemos que a entrada de Enable pode ser apropriadamente designada por **entrada de Modo**, já que permite dois **modos de funcionamento** do registo:

Modos de funcionamento

— o **modo de carregamento em paralelo** do registo que permite, como o nome indica, o carregamento em paralelo da quantidade booleana geral $(I3, I2, I1, I0)$; ou

Modo de carregamento em paralelo

— o **modo de manutenção** do estado do registo.

Modo de manutenção

O simbolo IEC deste registo será o que se ilustra na Figura 15.5.

O qualificador de entrada M2 está relacionado, por intermédio do símbolo 2, com as entradas de carregamento em paralelo (entradas D dos flip-flops) através dos qualificadores de entrada 1,2D.

Este modo realiza-se apenas se a entrada M2 estiver activa, isto é, a H, e se aparecer um flanco ascendente de relógio (o que é indicado pelo símbolo 1 ligado ao qualificador de entrada de relógio, C1). Ou seja, o carregamento em paralelo é síncrono (se fosse assíncrono, as entradas D dos flip-flops teriam qualificadores 2D em vez de 1,2D), isto é, ocorre imediatamente após um flanco de comutação dos flip-flops que apareça quando a entrada de Modo estiver activa.

Quando a entrada de Modo está inactiva, a L, não se faz carregamento em paralelo. Por omissão, supõe-se que o registo mantém o seu estado. Contudo, esta operação é ainda síncrona, como se percebe pelo logigrama do registo, pelo que sucessivos flancos de comutação com a entrada de Modo inactiva apenas confirmam a manutenção do estado do registo.

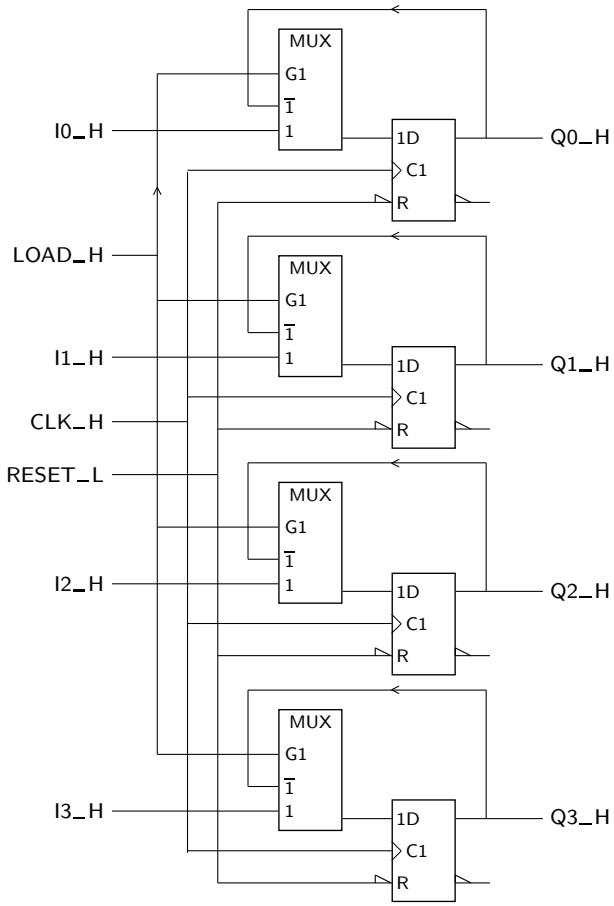


Figura 15.4: Registo de 4 andares com Reset assíncrono e uma entrada de Modo, $LOAD\_H$, que permite o carregamento em paralelo síncrono se estiver activada, ou a manutenção do valor carregado quando inactiva

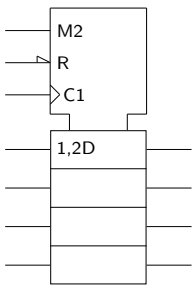


Figura 15.5: Símbolo IEC do registo da Figura 15.4

15.4 Registos de Deslocamento

Um outro tipo de registo muito importante é o registo de deslocamento. Na sua versão mais simples, um registo de deslocamento é um registo em que os dados,

em vez de entrarem em paralelo como no caso dos exemplos anteriores, surgem em série, isto é, bit a bit, por uma única entrada de dados. Como mostra a Figura 15.6, o circuito base é muito simples de desenhar.

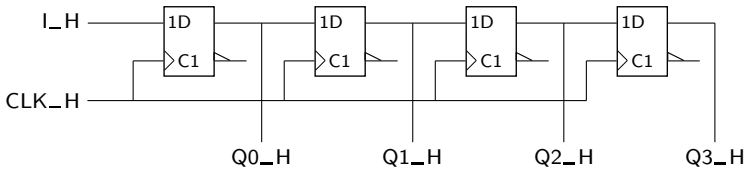


Figura 15.6: Registo de deslocamento para a direita com 4 andares formados por flip-flops edge-triggered do tipo D a comutar nos flancos ascendentes

Neste caso usaram-se 4 flip-flops D edge-triggered a comutar nos flancos ascendentes, mas é óbvio que se podiam ter usado mais ou menos andares, com outros tipos de flip-flops — contudo, *ver a observação da página 265 no que diz respeito à impossibilidade de usar latches D neste tipo de circuitos, devido às suas características de transparência.*

Quanto ao símbolo IEC deste registo, podemos constatar na Figura 15.7 a existência de um qualificador de entrada da forma C1 →, em que a seta para a direita significa que o registo desloca para a direita, como mostra o logigrama da Figura 15.6, o que no símbolo significa *de cima para baixo.* Com efeito, notar que a entrada de dados está ligada à entrada D do primeiro flip-flop (o de cima, no símbolo).

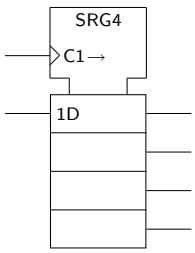


Figura 15.7: Símbolo IEC do registo de deslocamento para a direita da Figura 15.6

Naturalmente, o deslocamento é síncrono (quando houver um flanco de comutação), uma vez que a seta está precedida do qualificador C1).

Uma designação comum para este tipo de registos é **SIPO ("Serial-In, Parallel-Out")**, por oposição aos registos **PIPO ("Parallel-In, Parallel-Out")** estudados nas Secções 15.2 e 15.3. De notar que os primeiros fazem deslocamento, enquanto que os últimos não o podem fazer.

SIPO ("Serial-In, Parallel-Out")

PIPO ("Parallel-In, Parallel-Out")

Naturalmente, podemos ainda imaginar registos de deslocamento do tipo **SISO ("Serial-In, Serial-Out")** ou do tipo **PISO ("Parallel-In, Serial-Out")**. Na Figura 15.8 ilustram-se os símbolos IEC destes registos.

SISO ("Serial-In, Serial-Out")

PISO ("Parallel-In, Serial-Out")

É de realçar que se trata, nos dois casos, de registos de deslocamento, sendo que os registos PISO necessitam de dois modos de funcionamento, um para carregar

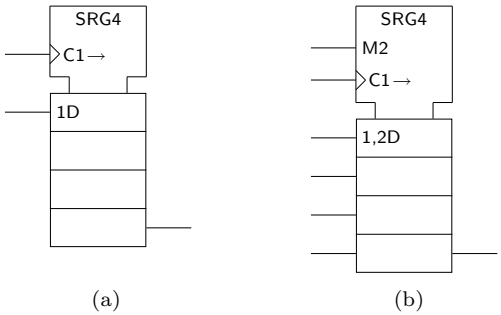


Figura 15.8: Símbolos IEC de registos do tipo: (a) SISO; e (b) PISO

em paralelo os dados de entrada, e outro para o subsequente deslocamento (no caso, para a direita).

Por outro lado, acentua-se que tudo o que foi dito para os registos de deslocamento para a direita pode igualmente ser aplicado a registos de deslocamento para a esquerda, como o da Figura 15.9.

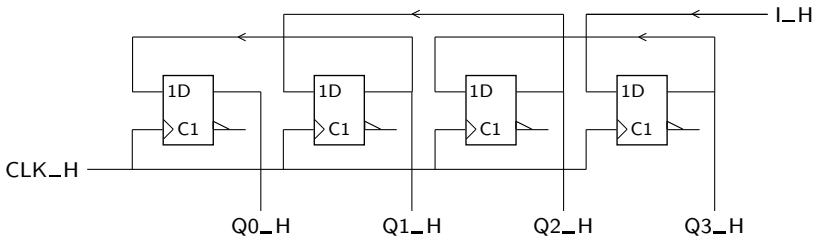


Figura 15.9: Registo de deslocamento para a esquerda com 4 andares

Neste caso o símbolo IEC (Figura 15.10) deve ter uma seta orientada para a esquerda, indicando um deslocamento *de baixo para cima* no símbolo.

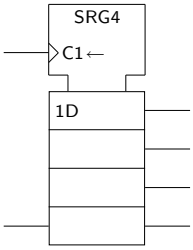


Figura 15.10: Símbolo IEC de um registo de deslocamento para a esquerda, com 4 andares formados por flip-flops D edge-triggered

15.5 Registos Multimodo

E possível conceber registos multimodo, usando a ideia de incluir um multiplexer antes de cada uma das entradas D dos flip-flops. Nesse caso, cada modo de

funcionamento corresponde a uma das entradas do multiplexer que selecciona a origem dos dados para cada entrada D.

Registo de deslocamento universal

O **registo de deslocamento universal** é um registo com 4 modos de funcionamento:

Modos de funcionamento

— o **modo de carregamento em paralelo** que permite, como o nome indica, o carregamento em paralelo de uma quantidade booleana geral com um número de bits igual ao número de andares do registo;

Modo de carregamento em paralelo

— o **modo de deslocamento para a direita**;

Modo de deslocamento para a direita

— o **modo de deslocamento para a esquerda**; e

Modo de deslocamento para a esquerda

— o **modo de manutenção** (síncrono) do estado do registo.

Modo de manutenção

Na Figura 15.11 ilustra-se um andar genérico de um registo de deslocamento universal.

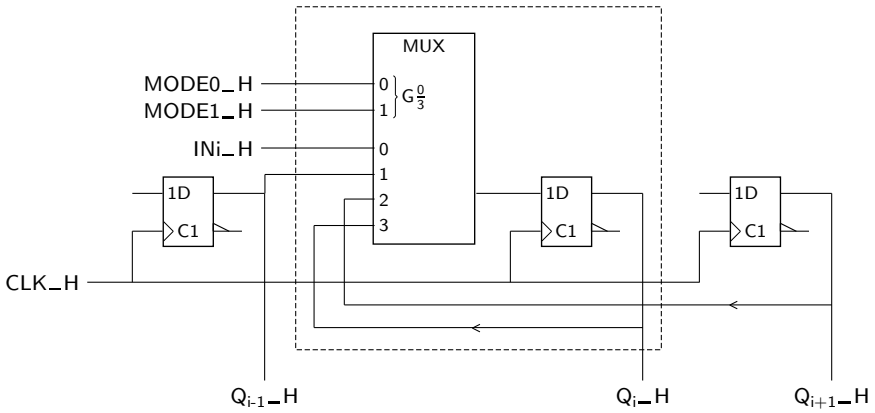


Figura 15.11: Andar genérico, i, de um registo de deslocamento universal

É de notar que os 4 modos são assegurados através de 4 quantidades booleanas gerais aplicadas às entradas de Modo, $MODE0\_H$ e $MODE1\_H$, em que a variável $MODE1$ tem mais peso do que $MODE0$. Isso resulta de as entradas de Modo terem sido ligadas às entradas de selecção do multiplexer da forma indicada na figura.

Na Figura 15.12 apresenta-se o símbolo IEC de um registo de deslocamento universal, o 74x194.

74x194

Repare-se que existem no símbolo entradas série nos dois flip-flops extremos do registo. Essas entradas têm qualificadores 1,4D no caso do deslocamento para a direita, porque associados à seta com o qualificador 1. E têm qualificadores 2,4D no caso do deslocamento para a esquerda, porque associados à seta com o qualificador 2.

Por outro lado, existem ainda entradas separadas de carregamento em paralelo (com qualificadores 3,4D) em todos os andares do registo.

Assim sendo, o modo M3 é o de carregamento em paralelo síncrono, o modo M1 é o de deslocamento para a direita (naturalmente, síncrono), o modo M2 é

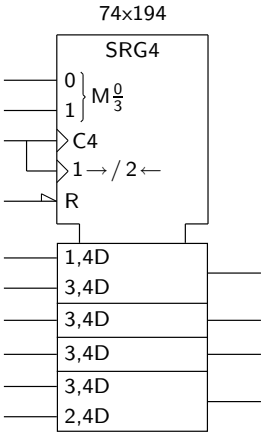


Figura 15.12: Símbolo IEC de um registo de deslocamento universal, o 74x194

o de deslocamento para a esquerda (mais uma vez síncrono), e o modo M0, por omissão, é o modo de manutenção do estado do registo (também síncrono).

De notar ainda que a entrada de relógio, com o qualificador C4, é apenas uma (um e apenas um pino do integrado), embora esteja cindida em duas partes para não sobrecarregar o simbolo. Contudo, era perfeitamente legítimo indicar apenas uma entrada no símbolo com o qualificador C4 / 1→ / 2←.

15.6 Transferências entre Registos

15.6.1 Interligação de registos

Muitas aplicações em Sistemas Digitais, nomeadamente no que diz respeito à Arquitectura de Computadores, assentam na existência de um certo número de registos interligados que permitem transferir informação entre si, directamente ou através de blocos de lógica combinatória que processam a informação contida nos registos. No imediato, iremos estudar apenas soluções para possibilitar a troca de informação entre um conjunto de registos.

Barramento

A interligação de registos será feita interligando, através de outros dispositivos, o conjunto das suas saídas e entradas. É oportuno introduzir aqui a noção de **barramento** ("bus" em inglês). Um barramento é um conjunto de linhas que transportam sinais do mesmo tipo e que devem ser tratados de forma semelhante. Por exemplo o conjunto de linhas de saída de um registo constitui um barramento.

15.6.2 Interligação entre registos utilizando multiplexers

Supondo que temos um certo número de registos que devem trocar informação entre si, a solução mais evidente consiste na utilização de multiplexers nas entradas dos registos, multiplexers esses que permitem seleccionar a origem da informação que vai ser armazenada em cada registo.

Suponhamos uma situação em que há 4 registos de 5 bits cada um, que pretendemos interligar (Figura 15.13).

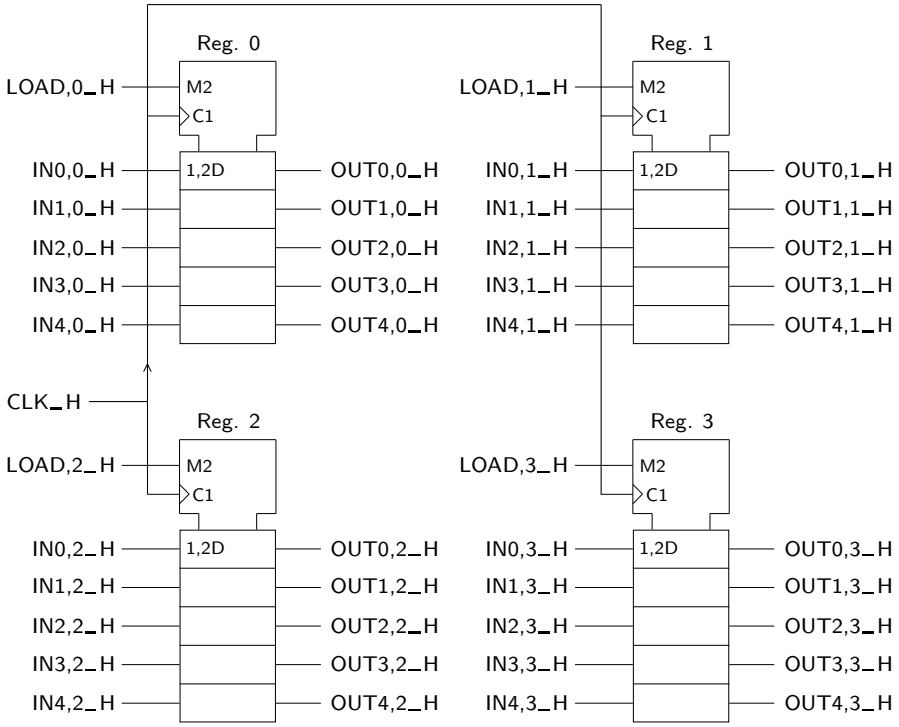


Figura 15.13: Quatro registos de 5 bits cada um, que pretendemos ver interligados

A saída i do registo j vem designada por OUTi,j. Do mesmo modo, a entrada i do registo j vem designada por INi,j.

Para os interligarmos, cada registo vai precisar de ter, em cada entrada, um multiplexer de 4 entradas de dados, o que significa a utilização de 20 multiplexers de 4 entradas. Deste modo consegue-se, para cada registo, multiplexar em cada uma das suas entradas os barramentos de saída de todos os registos, incluindo ele próprio.

Escolhendo um dos registos para ilustrar a solução, teremos a situação descrita na Figura 15.14.

Esta solução é muito complexa, mas tem a vantagem de permitir, em cada momento, transferir dados para os n registos simultaneamente.

Uma solução mais interessante do ponto de vista de complexidade consiste em usar, para todos os registos, apenas um conjunto de multiplexers, em vez de um para cada registo. Isso leva a que todos os registos vejam na sua entrada a mesma informação, e só aquele em que se pretende carregar a informação de saída de um deles é que deve ver activado o seu modo de carregamento em paralelo.

Perde-se, é claro, a hipótese de fazer várias transferências simultaneamente, uma vez que o recurso crítico (os multiplexers) passa a estar partilhado.

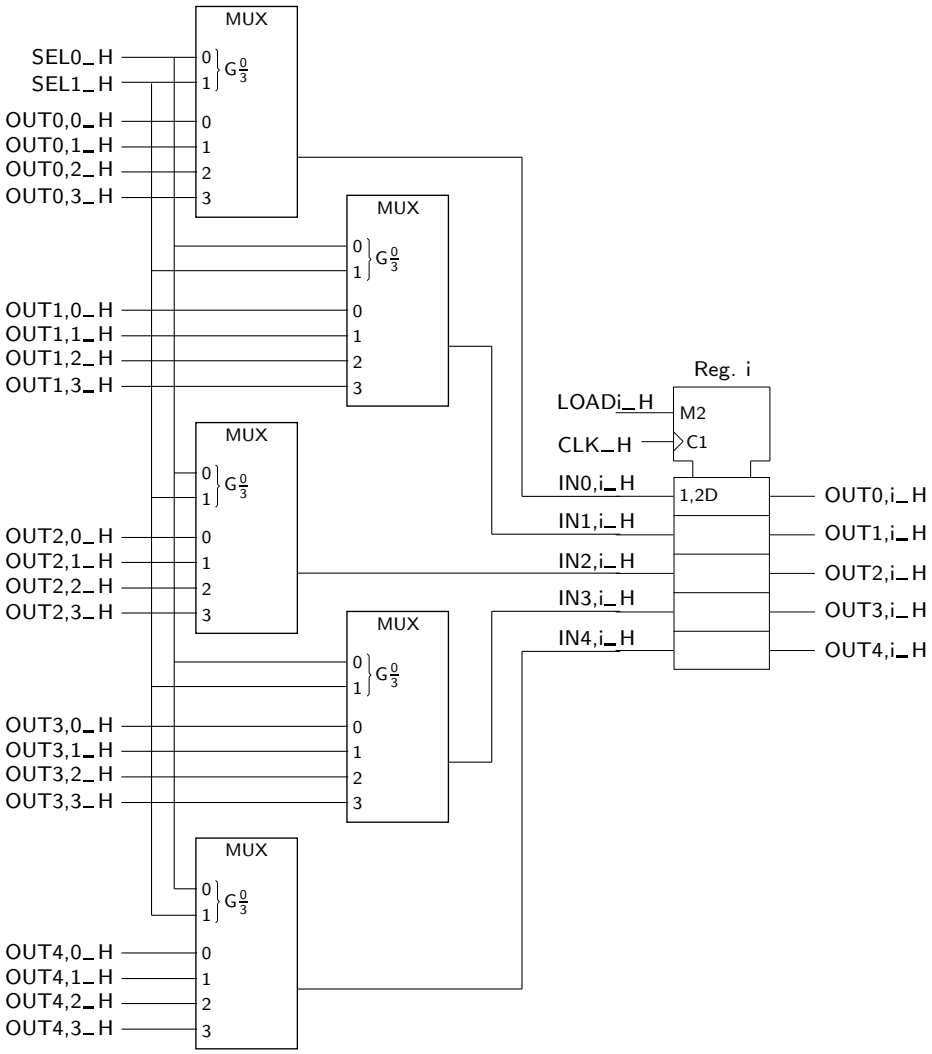


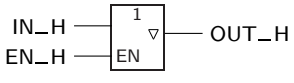
Figura 15.14: A interligação dos 4 registos da Figura 15.14 pode ser feita com multiplexers, mas a solução é muito complexa. Nesta figura apenas se apresenta a ligação a *um* dos registos

15.6.3 Buffers tri-state

Buffer tri-state

Antes de avancar com uma solução mais simples, há que apresentar um novo dispositivo — o **Buffer tri-state**. Trata-se um dispositivo com uma entrada e uma saída de dados com capacidade tri-state, e uma entrada de Enable que controla essa capacidade tri-state de acordo com a tabela de verdade física da Figura 15.15(b), que mais não é do que uma extensão da tabela de verdade genérica da Tabela 7.5, na página 118.

Ou seja, quando a entrada de Enable está activa, a saída assume um nível que é uma cópia do nível que estiver presenta à entrada de dados do Buffer. Quando, pelo contrário, a entrada de Enable está inactiva, então a saída fica em alta impedância [o que se representa por Hi-Z na Figura 15.15(b)].



(a)

EN_H	IN_H	OUT_H
L	×	Hi-Z
H	L	L
H	H	H

(b)

Figura 15.15: Símbolo IEC de um Buffer tri-state com entradas de dados e de Enable activas a H, e com a saída também activa a H; e (b) tabela de verdade física deste Buffer

Quanto ao símbolo IEC deste Buffer, ver a Figura 15.15(a).

Muitas vezes estes Buffers não aparecem de forma isolada nos circuitos integrados, mas em conjuntos de 4 ou de 8. A Figura 15.16 apresenta o símbolo de um Buffer quádruplo do tipo 74HCT244. *74HCT244*

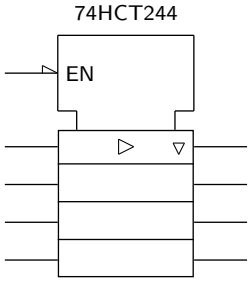


Figura 15.16: Símbolo IEC de um Buffer tri-state quádruplo do tipo 74HCT244

No símbolo, a única particularidade digna de realce é a existência dos qualificadores gerais ▷ colocados em cada um dos 4 Buffers e orientados no sentido das entradas para as saídas, que indicam a existência de amplificação dos sinais de entrada (a designação Buffer vem desta característica de amplificação), para além de identificarem o sentido do fluxo da informação. *Qualificador geral ▷*

Há duas aplicações fundamentais para os Buffers tri-state. Por um lado permitem simplificar a complexidade das operações de multiplexagem, sem o recurso a multiplexers. Por outro lado, potenciam a utilização de linhas bidireccionais.

A utilização de Buffers tri-state para multiplexar temporalmente vários dados é já conhecida da Subsecção 6.4.2, e pode ser traduzida pelo logigrama da Figura 15.17 no caso de o número de dados a multiplexar ser igual a dois.

Considere-se a linha $SEL.IN1\_H = H$. Nestas circunstâncias, o Buffer superior vem inibido (a sua saída vem em alta impedância) e o inferior activado, colocando o nível de tensão da linha de entrada $IN1\_H$ na linha de saída OUT_H. Inversamente, se $SEL.IN1\_H = L$ teremos o Buffer inferior inibido e o superior activado, obtendo-se $OUT\_H = IN0\_H$.

A estrutura apresentada tem, assim, o comportamento de um multiplexer de duas entradas de dados e uma de selecção. Com um descodificador de n entradas

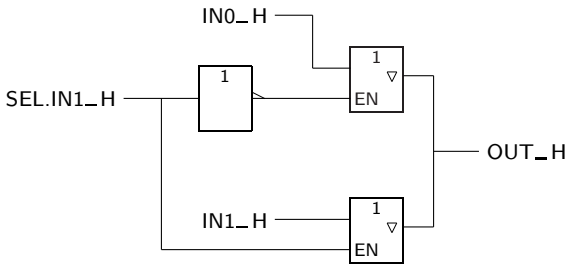


Figura 15.17: Multiplexagem temporal de dois dados, obtida à custa de dois Buffers tri-state

e um conjunto de Buffers tri-state é possível contruir um multiplexer de 2^n entradas de dados.

Consideremos agora a questão da bidireccionalidade, observando o logigrama da Figura 15.18.

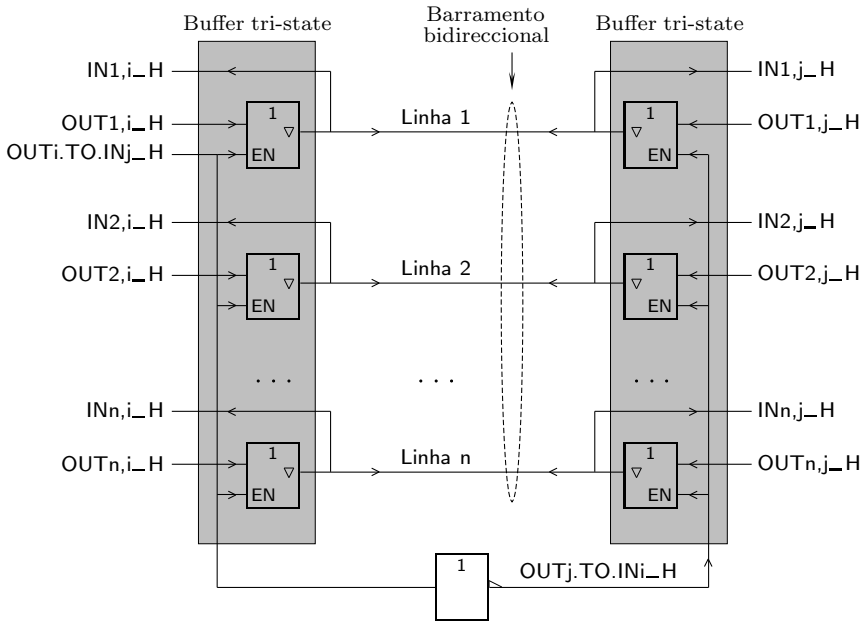


Figura 15.18: Barramento bidireccional com n linhas, controlado por Buffers tri-state unidireccionais

Buffer tri-state unidireccional

Neste logigrama é possível observar as várias linhas do barramento, com cada uma delas a ligar dois **Buffers tri-state unidireccionais**, um à direita e outro à esquerda no desenho.

Admite-se que as entradas de dados dos Buffers estão ligadas às saídas $OUTi$_H e $OUTj$_H de registos apropriados, e que as linhas do barramento estão ligadas às entradas INi_H e INj_H dos mesmos registos.

Se a linha $OUTi.TO.INj$_H estiver activa, a H, então os buffers da direita estão inibidos, as suas saídas estão em alta impedância e, consequentemente,

portam-se como se não estivessem ligados ao barramento. Em contrapartida, os Buffers da esquerda estão activos e a informação que se encontra nas linhas $OUTi$ de saída do registo i é transmitida através dos Buffers e pelo barramento para a parte direita, onde pode ser lida ou armazenada no registo j.

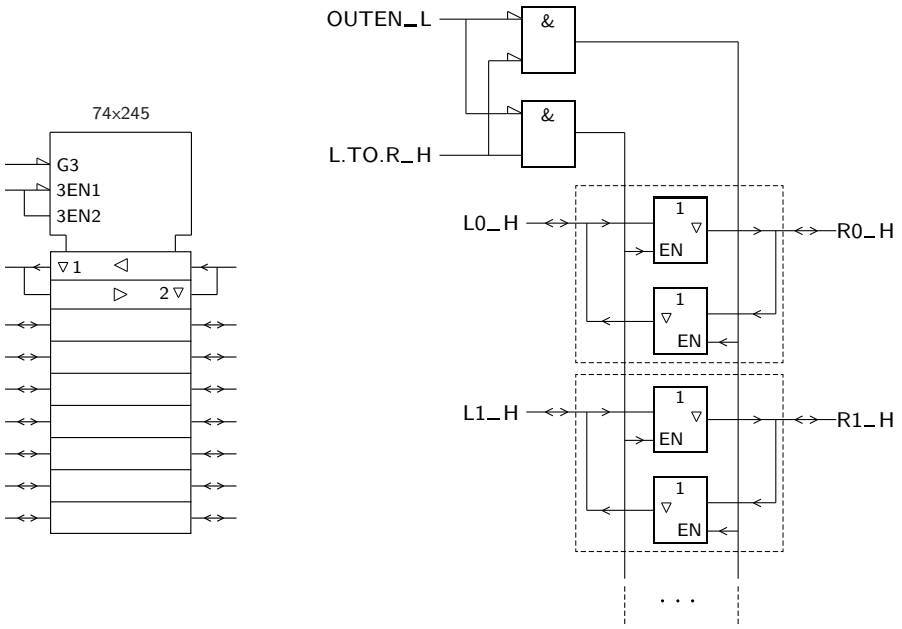
Se a linha $OUTi.TO.INj\_H$ estiver inactiva, a L, observa-se exactamente o contrário, com a transferência a ser efectuada da saída do registo j, através dos Buffers da direita, pelo barramento e daí para a entrada do registo i, onde pode ser aramazenada.

Um barramento bidireccional como este permite então trocar, sobre as mesmas linhas, informação nas duas direcções (uma de cada vez, claro). Este tipo de aplicação é muito importante, como veremos em Arquitectura de Computadores.

Na Figura 15.19(a) apresenta-se o símbolo IEC do 74x245, um **Buffer tri-state bidireccional** típico, com 8 linhas. E na Figura 15.19(b) ilustram-se dois andares do Buffer (de entre os 8 que ele possui), mais os respectivos sinais de controlo.

74x245

Buffer tri-state bidireccional



(a) (b)

Figura 15.19: (a) Símbolo IEC de um Buffer tri-state bidireccional com 8 linhas, o 74x245; (b) logigrama de dois dos 8 andares do Buffer, e respectivos sinais de controlo

Realçam-se os qualificadores gerais ▷ colocados nos dois sentidos em cada um dos 8 Buffers, o que ilustra claramente o sentido do fluxo da informação.

De notar que, se a entrada de Enable das saídas, OUTEN_L, estiver inactiva, todos os Buffers ficam com as suas saídas em alta impedância e o barramento da esquerda fica separado do barramento da direita. Quando OUTEN_L está activa, a transferência dá-se da esquerda para a direita se L.TO.R_H estiver activa, e da direita para a esquerda se estiver inactiva.

15.6.4 Interligação entre registos utilizando barramentos tri-state

Os Buffers tri-state unidireccionais permitem simplificar consideravelmente o problema da interligação de registos. Se admitirmos que os registos possuem uma capacidade tri-state nas saídas, controlada por uma linha OE_L (OE de Output Enable) — o que é comum verificar-se — podemos organizar o logigrama da Figura 15.20 em que se interligam quatro registo de 5 bits, tal como se fez para a Figura 15.13.

Repare-se que desapareceram os circuitos em volta dos registos da Figura 15.14, ficando apenas presente dois descodificadores.

O descodificador superior faz o Enable da saída de um e apenas de um registo de cada vez (aquele que se pretende que seja a fonte da informação a transferir). Pelo barramento passa, então, o conteúdo desse registo, conteúdo esse que é apresentado simultaneamente à entrada de todos os registos. Por sua vez, o descodificador inferior escolhe o registo que vem com o modo de carregamento em paralelo activo e, portanto, o registo que armazena o que está presente no barramento, quando surgir o flanco de comutação no relógio.

Dada a sua simplicidade e capacidade de modularização, esta solução com barramentos tri-state é muito comum neste tipo de problemas, em que se pretendem interligar vários registos com idênticas dimensões.

15.7 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secção 5.1 a 5.3, 7.4 e 7.5.

15.8 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

(∗) 15.1 Recorrendo a 4 flip-flops JK, implemente um registo com 4 andares que permita fazer deslocamento à direita, deslocamento à esquerda, deslocamento circular à direita, e memorizar em paralelo.

(∗) 15.2 a) Construa um registo com 4 flip-flops do tipo D capaz de memorizar em paralelo do exterior, de efectuar a divisão do seu conteúdo por dois, e de duplicar o seu conteúdo (desde que o resultado da duplicação continue a poder ser representado com 4 bits).

b) Amplie o circuito anterior, de forma a ligar quatro registos idênticos aos pedidos na alínea anterior a um barramento comum.

15.3 Suponha que dispõe de um registo de deslocamento à direita com 4 bits e carregamento em paralelo síncrono. Construa, usando o material que

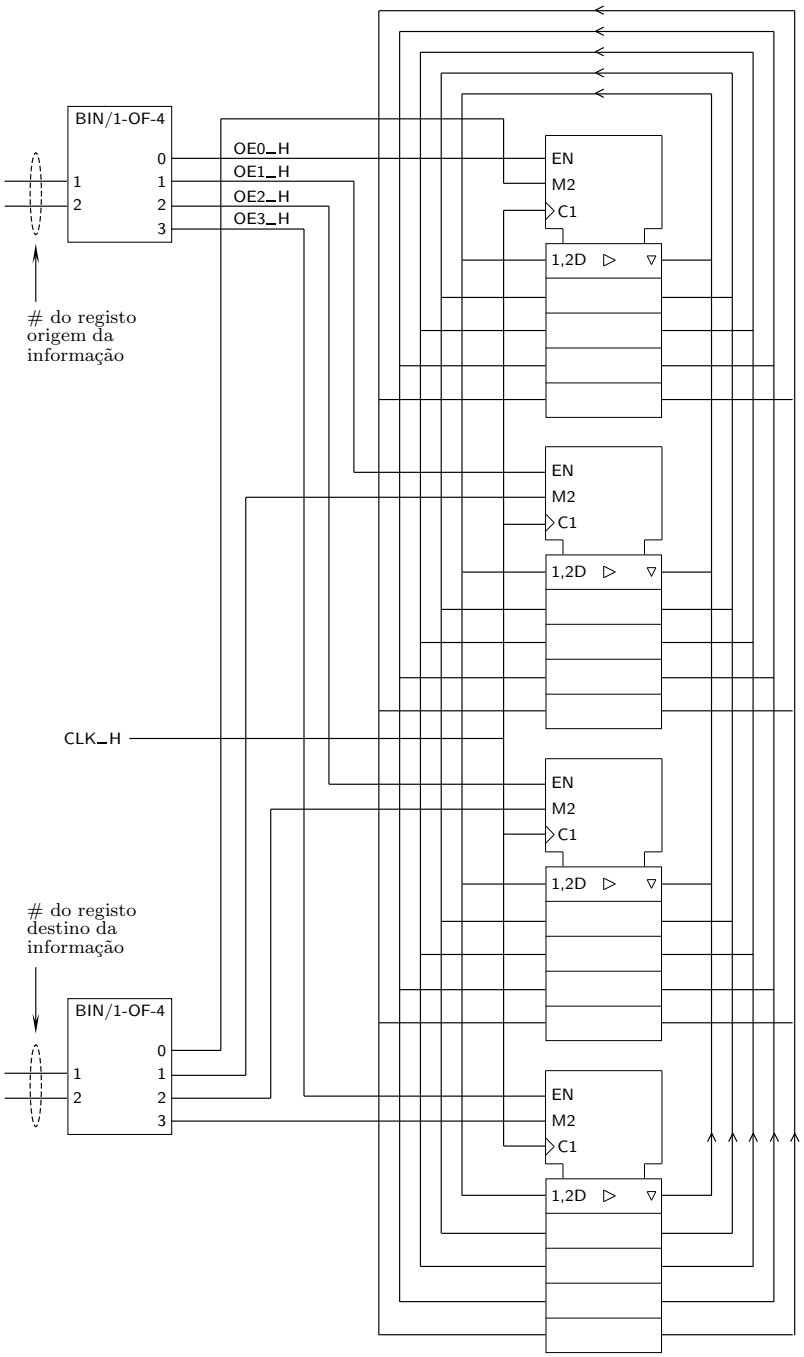


Figura 15.20: Logigrama de um sistema formado por 4 registos interligados com saídas tri-state

achar necessário e que especificará como entender conveniente, um circuito com os seguintes modos de funcionamento: (i) deslocamento de um bit à direita; (ii) deslocamento de um bit à esquerda; (iii) carregamento de

dados em paralelo; (iv) complementação do seu conteúdo; e (v) não fazer nada (apesar de receber impulsos de relógio).

15.4 Construa um registo de 6 bits que possa operar nos seguintes modos: (i) memorizar em paralelo do exterior; (ii) multiplicar por 4 o seu conteúdo (até 15); (iii) multiplicar por 8 o seu conteúdo (até 7; e (v) efectuar a divisão inteira do seu conteúdo por 4.

15.5 Construa um registo de deslocamento com 4 bits com os seguintes modos: (i) deslocamento de uma posição para a direita; (ii) deslocamento de uma posição para a esquerda; (iii) complementação do conteúdo; e (iv) adição de uma unidade ao seu conteúdo (por exemplo, se o conteúdo for 0101 passará a 0110).

15.6 Construa, usando os flip-flops que achar conveniente e o mínimo possível de lógica adicional, um registo de 4 bits com os seguintes modos de funcionamento: (i) carregamento em paralelo; (ii) contagem descendente; (iii) reset síncrono; e (iv) deslocamento à direita (no sentido do bit menos significativo).

(∗) 15.7 Um registo de deslocamento de comprimento variável é um registo SISO com um número variável de bits. Este tipo de registo é utilizado para provocar um número variável de ciclos (de relógio) de atraso da entrada para a saída. O comprimento vem especificado por uma variável booleana geral de controlo. Desenhe o logigrama de um registo de deslocamento com comprimento variável entre 1 e 8 bits. Use, para tanto, um registo SIPO e módulos combinatórios.

15.8 Projecte um circuito capaz de reconhecer os seguintes padrões binários: 0101 e 0110. Utilize um registo de deslocamento do tipo SIPO e portas lógicas.

(∗) 15.9 Como projectaria um andar (genérico) de um registo de deslocamento com carregamento em paralelo assíncrono? E se fosse com carregamento em paralelo síncrono?

15.10 Dispõe de dois registos, A e B, com 4 bits cada um, que permitem, a cada impulso de relógio, a memorização em paralelo dos valores que se lhes apresentam às entradas.

a) Ligue-os sobre um barramento comum, por forma a que a informação possa ser transferida de A para B ou de B para A.

b) Ligue-os sobre um terceiro registo (Buffer C), também com 4 bits. A informação a ser transferida de A par B ou de B para A deverá ser memorizada temporariamente no Buffer. Neste caso, a transferência da informação deverá ocorrer em dois impulsos de relógio consecutivos.

15.11 Dispõe de 3 registos idênticos, $R1$, $R2$ e $R3$, cada um com 4 andares e constituídos por flip-flops do tipo D e saídas tri-state. Desenhe as ligações internas entre os flip-flops de um dos registos, e as ligações externas entre os registos, de modo a que a informação de qualquer dos registos possa ser transferida para outro ou outros registos, de acordo com as tabelas da Tabela 15.1.

Tabela 15.1: Tabelas de verdade físicas com os modos de funcionamento do circuito do Exercício 15.11

ORG1_H	ORG0_H	Modo
L	L	—
L	H	Origem em $R1$
H	L	Origem em $R2$
H	H	Origem em $R3$

DEST2_H	DEST1_H	DEST0_H	Modo
L	L	L	Destino é $R1$
L	L	H	Destino é $R2$
L	H	L	Destino é $R3$
L	H	H	Destinos são $R1$ e $R2$
H	L	L	Destinos são $R1$ e $R3$
H	L	H	Destinos são $R2$ e $R3$

15.12 Dispõe de um registo que não é de deslocamento, com 4 bits, e de um circuito somador completo de 4 bits. Projecte, usando os multiplexers e as portas que achar conveniente, um circuito que permita realizar a seguinte operação: $((A \times 2) + B)/2)$. As operações parciais decorrerão sequencialmente: o valor de A será inicialmente carregado em paralelo no registo e depois multiplicado por 2; em seguida o resultado dessa operação deverá ser somado com B e, após isso, o novo resultado deverá vir dividido por 2.

Não se pretende o projecto do circuito de controlo (isso constituirá matéria para os capítulos seguintes), mas apenas do circuito controlado (o circuito que é pedido, com o registo de deslocamento, os multiplexers e as portas).

15.13 O registo de deslocamento universal 74x194 da Figura 15.12 é utilizado no circuito da Figura 15.21 para formar um **contador em anel**, com apenas uma saída activa de cada vez. *Contador em anel*

Qual é a sequência de contagem do contador? Desenhar um diagrama de estados que justifique a sua resposta.

15.14 O contador em anel do exercício anterior tem dois problemas: (i) se, como consequência de uma falha no circuito ou de ruído, a sua única saída activa vier a L, o contador passa ao estado 0000 e fica nesse estado por tempo indeterminado; e (ii) se um 1 extra vier, pelos mesmos motivos, inserido na sequência de contagem (por exemplo, criando o estado de contagem 0101), o contador segue um ciclo de contagem incorrecto (deixa de funcionar como contador em anel) e fica nesse ciclo por tempo indeterminado. Desenhar um **contador em anel auto-corrector** que não tenha estes problemas. *Contador em anel auto-corrector*

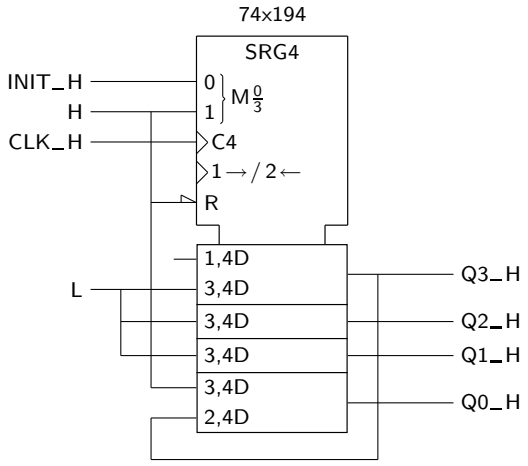


Figura 15.21: Contador em anel que utiliza um registo de deslocamento universal

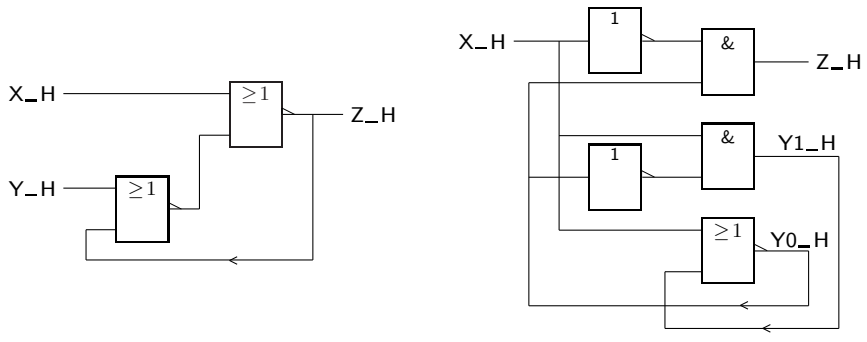
Capítulo 16

Circuitos Sequenciais Síncronos

16.1 Circuitos Síncronos e Assíncronos

Relembremos, dos capítulos anteriores, que os circuitos sequenciais (ao contrário do que sucede com os circuitos combinatórios) apresentam níveis de tensão (valores lógicos) nas saídas que podem ser diferentes para níveis de tensão (valores lógicos) iguais nas entradas.

Existem várias formas de realizar circuitos sequenciais. Desde logo é possível realizar circuitos sequenciais sem utilizar flip-flops, apenas construindo realimentações entre as saídas de circuitos combinatórios e algumas entradas, como acontece com o circuito da Figura 16.1(a) — um latch RS como o da Figura 12.5, redesenhado para fazer salientar a (única) linha de realimentação — ou o circuito da Figura 16.1(b), com duas linhas de realimentação.



(a) (b)

Figura 16.1: Logigramas de circuitos sequenciais que não utilizam flip-flops (circuitos assíncronos). O circuito da parte (a) é um latch RS, já conhecido da Figura 12.5

Não se deve, contudo, inferir que todos os circuitos formados por circuitos com-

binatórios com realimentações entre as saídas e as entradas são, necessariamente, circuitos sequenciais.

Circuitos sequenciais assíncronos

Este tipo de circuitos possui vantagens e inconvenientes: é de mais difícil projecto, e os circuitos são mais delicados de interligar. Como não há qualquer sinal que coordene o momento em que as diversas variáveis de realimentação são actualizadas, estes circuitos são denominados de **sequenciais assíncronos**.

O facto de um circuito ser construído em torno de flip-flops (ou latches) não garante, por si só, que o circuito não seja assíncrono. Por exemplo, o circuito da Figura 16.2 é, também, um circuito assíncrono e, contudo, possui dois flip-flops edge-triggered.

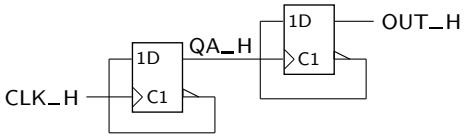


Figura 16.2: Logigrama de um circuito sequencial assíncrono que utiliza flip-flops edge-triggered do tipo D

Os flip-flops do circuito estão individualmente organizados como divisores de frequência por 2 pelo que, no conjunto, formam um contador assíncrono de módulo 4.

Com efeito, uma mudança de estado do segundo flip-flop apenas pode ser consequência de uma mudança (um flanco ascendente) na saída do primeiro e ocorre, portanto assincronamente em relação a ele. Mais uma vez se trata de um circuito sequencial assíncrono, uma vez que não há um sinal que coordene a mudança *dos dois* flip-flops.

Circuitos sequenciais síncronos

Os circuitos que nos vão ocupar daqui para a frente são designados por **circuitos sequenciais síncronos**. Um circuito sequencial síncrono é um circuito que é baseado em flip-flops (ou em outros elementos de memória adequados) que reagem ao mesmo flanco do relógio, ou seja, com interligação das linhas de relógio de todos eles. O circuito da Figura 16.3 é exemplo de um circuito sequencial síncrono.

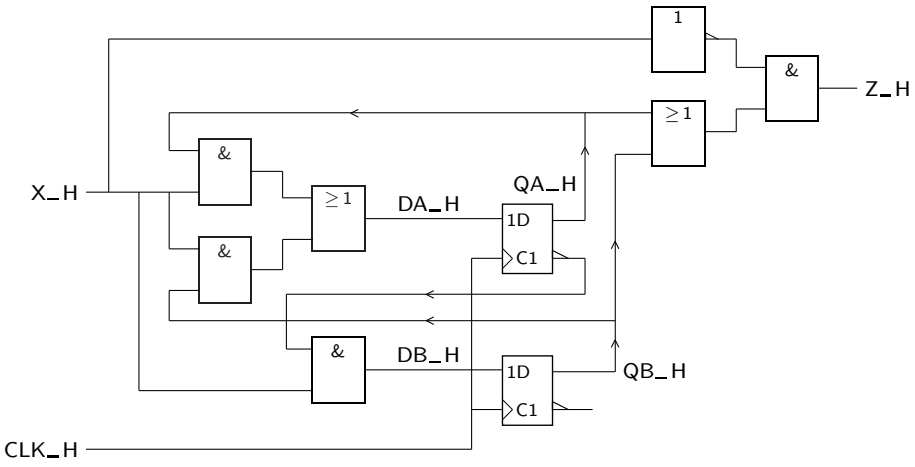


Figura 16.3: Exemplo de circuito sequencial síncrono

16.2 Modelo de um Circuito Sequencial Síncrono

Como vimos anteriormente, os circuitos síncronos podem ser caracterizados, em primeira análise, por possuírem um conjunto de flip-flops (ou outros elementos de memória com as mesmas características — sem transparência) que reagem sincronamente aos flancos de comutação dos impulsos de relógio aplicados simultaneamente a todos eles.

Função de memória

Desta forma, os flip-flops garantem uma **função de memória** da sequência de valores lógicos (ou níveis de tensão eléctrica) aplicados às suas entradas externas, até um determinado instante. No fundo, os flip-flops descrevem, a cada flanco de comutação do relógio, um novo **estado do circuito** — que pode, eventualmente, ser igual ao anterior.

Estado de um circuito síncrono

Estado actual (presente)

Terá de existir também um módulo de lógica combinatória que, em função dos níveis de tensão (ou valores lógicos) presentes nas entradas externas e do **estado actual** ou **estado presente** do circuito, permite determinar e apresentar aos flip-flops, para armazenagem, o próximo estado do circuito — o seu **estado seguinte**. A esta lógica combinatória dá-se o nome de **lógica do estado seguinte**.

Estado seguinte

Lógica do estado seguinte

Por fim, é necessário um outro bloco de lógica combinatória que, também em função do estado actual do circuito e dos níveis (valores lógicos) presentes nas entradas externas, determina as saídas do circuito — é a chamada **lógica de saída**.

Lógica de saída

O modelo descrito na Figura 16.4 descreve a estrutura genérica de um circuito sequencial síncrono.

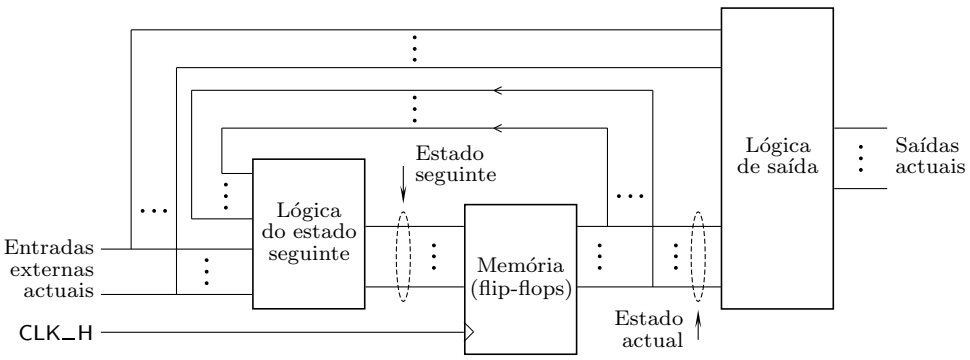


Figura 16.4: Modelo de um circuito sequencial síncrono genérico, onde se pode observar um conjunto de flip-flops sincronizados pelos mesmos flancos de comutação, uma lógica do estado seguinte e uma lógica de saída. As saídas dos flip-flops definem, em cada instante, o estado actual do circuito sequencial. As entradas dos flip-flops definem, directa ou indirectamente, o estado seguinte do circuito

16.3 Análise dos Circuitos Sequenciais Síncronos

Para a análise de um circuito sequencial síncrono parte-se do esquema eléctrico ou do logigrama do circuito e obtêm-se descrições de maior nível de abstracção,

que permitam concluir do comportamento do circuito em vez da sua estrutura, tal qual ela vem dada pelo logigrama ou pelo esquema eléctrico.

O procedimento a seguir é relativamente linear:

Equações de excitação
Equações de saída

1. levantam-se do circuito as **equações de excitação** dos flip-flops e as **equações da saída** do circuito;

Tabela de excitações do circuito

2. com as equações de excitação dos flip-flops estabelece-se uma **tabela de excitações do circuito** onde, para cada estado actual e para cada congiguração das entradas externas, se identificam os níveis de tensão aplicados aos flip--flops;

Tabela de transições e de saídas

3. a partir da tabela de excitações do circuito elabora-se uma outra tabela onde, para cada estado actual e para cada configuração de entradas externas, se identifica o estado seguinte do circuito e o nível correspondente das saídas; essa tabela designa-se por **tabela de transições e de saídas**;

Codificação dos estados
Tabela de estados e de saída

4. a partir da tabela de transições do circuito podemos, por **codificação dos estados**, obter uma **tabela de estados e de saídas** do circuito;

Diagrama de estados

5. em alternativa à tabela de estados e de saídas, pode construir-se um grafo orientado representando os estados, as transições entre estados e as saídas, a que se chama **diagrama de estados**.

Vamos exemplificar a aplicação destas regras para o circuito anteriormente representado na Figura 16.3.

1. Equações de excitação dos flip-flops e equações de saída

É fácil de ver, a partir do logigrama do circuito, que se verificam as seguintes equações de excitação e de saída:

$$\begin{aligned} DA &= QA \cdot X + QB \cdot X \\ DB &= \overline{QA} \cdot X \\ Z &= (QA + QB)\,\overline{X} \end{aligned}$$

2. Tabela de excitações do circuito

O circuito possui dois flip-flops, pelo que poderá ter 4 estados. Na tabela de excitações da Tabela 16.1 apresentam-se todos os estados do circuito e, para cada nível de tensão na entrada externa, representam-se os níveis de tensão em DA_H e em DB_H.

Notar como esta tabela traduz os circuitos combinatórios de excitação dos flip--flops, já que toda ela é definida no instante t.

3. Tabela de transições e de saídas

Como os flip-flops D assumem nas saídas os níveis de tensão que têm presentes nas entradas síncronas quando recebem o flanco activo do impulso de relógio, é fácil perceber que a tabela de transições é igual, para estes flip-flops, à tabela de excitações. Obtemos, então, a tabela de transições e de saídas do circuito na Tabela 16.2.

Tabela 16.1: Tabela de excitações para o circuito da Figura 16.3

Estado actual		**Níveis em DA e em DB**			
		$X\_H_{(t)} = L$		$X\_H_{(t)} = H$	
$QA\_H_{(t)}$	$QB\_H_{(t)}$	$DA\_H_{(t)}$	$DB\_H_{(t)}$	$DA\_H_{(t)}$	$DB\_H_{(t)}$
L	L	L	L	L	H
L	H	L	L	H	H
H	L	L	L	H	L
H	H	L	L	H	L

Tabela 16.2: Tabela de transições e de saídas para o circuito da Figura 16.3

Estado actual		**Estado seguinte**				**Saída** $Z\_H_{(t)}$	
		$X\_H_{(t)} = L$		$X\_H_{(t)} = H$		$X\_H_{(t)} = L$	$X\_H_{(t)} = H$
$QA\_H_{(t)}$	$QB\_H_{(t)}$	$QA\_H_{(t+1)}$	$QB\_H_{(t+1)}$	$QA\_H_{(t+1)}$	$QB\_H_{(t+1)}$		
L	L	L	L	L	H	L	L
L	H	L	L	H	H	H	L
H	L	L	L	H	L	H	L
H	H	L	L	H	L	H	L

Notar como os estados actuais e as saídas são definidas no instante t, e como os estados seguintes são definidos no instante $t + 1$. Por outro lado, reparar como as excitações $D_{(t)}$ em cada flip-flop (na Tabela 16.1) coincidem com as suas saídas $Q_{(t+1)}$ — os estados seguintes da Tabela 16.2.

Esta última tabela já constitui uma descrição comportamental do circuito ao longo do tempo, na medida em que apresenta a evolução estado actual → estado seguinte.

Uma máquina sequencial é uma abstracção de um circuito sequencial síncrono. Geralmente é descrita por um diagrama de estados ou por uma tabela de estados e de saídas.

Contudo, é ainda possível obter uma tabela de estados para o circuito ou ainda o seu diagrama de estados que, de forma mais abstracta e mais "visual", permitem percepcionar o comportamento da **máquina sequencial** que lhe corresponde.

Máquina sequencial

4. Tabela de estados

Tabela de estados

A partir da tabela de transições e de saída podemos agora estabelecer uma tabela de estados e de saídas que, por vezes, e de forma abreviada, se designa apenas por **tabela de estados** da máquina sequencial.

Para tanto necessitamos de codificar previamente os estados da máquina (e do circuito sequencial). Ou seja, vamos, de forma abstracta, designar cada configuração de estados dos flip-flops, no caso (QA_H, QB_H), por um nome arbitrário.

Naturalmente, a codificação dos estados é, nesta fase de análise, completamente arbitrária. É usual designar os estados pelas primeiras letras do alfabeto latino ("A", "B", etc.), mas podemos dar-lhes quaisquer outras designações. Por exemplo, podemos fazer a codificação de estados expressa pela Tabela 16.3.

Tabela 16.3: Tabela com a codificação de estados da máquina sequencial que está a ser analisada

Estado	**QA_H**	**QB_H**
A	L	L
B	L	H
C	H	L
D	H	H

Nesse caso obtém-se a tabela de estados da máquina sequencial que vem descrita pela Tabela 16.4.

Tabela 16.4: Tabela de estados para a máquina sequencial que vem implementada pelo circuito da Figura 16.3

Estado	**Estado seguinte**		**Z**	
actual	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	A	D	1	0
C	A	C	1	0
D	A	C	1	0

De notar que a tabela de saídas está representada em lógica positiva (com valores lógicos em vez de níveis de tensão), dado estarmos agora no domínio algébrico, onde nos manteremos com o diagrama de estados que se segue.

De notar ainda que a tabela de estados representa, de facto, uma máquina sequencial abstracta, que pode ser implementada pelo circuito da Figura 16.3 ou por muitos outros circuitos sequenciais diferentes (por exemplo, que usem flip-flops JK em vez de flip-flops D, ou que usem ainda flip-flops D edge-triggered mas que comutem nos flancos descendentes).

5. Diagrama de estados

O diagrama de estados da máquina sequencial constrói-se formando um grafo orientado com círculos representativos dos estados da máquina e com setas a ligar estados, representando as transições entre os estados.

Cada círculo representativo de um estado conterá a inscrição desse estado. As transições entre estados são representadas por setas que vão dos estados actuais para os estados seguintes. Junto a cada seta coloca-se uma indicação dos valores lógicos (porque estamos no domínio algébrico) das entradas externas que ocasionam essa transição.

Quanto aos valores lógicos das saídas, vêm associados aos valores lógicos das entradas externas que ocasionam as transições, e separados deles por um símbolo "/".

Na Figura 16.5 ilustra-se o diagrama de estados da máquina sequencial que foi implementada pelo circuito da Figura 16.3.

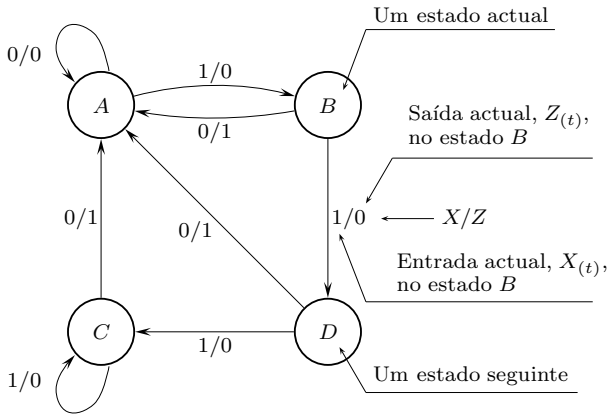


Figura 16.5: Diagrama de estados do circuito sequencial síncrono da Figura 16.3

Repare-se que, quer na tabela de transições, quer no diagrama de estados, a indicação do estado seguinte corresponde a um futuro que se concretizará quando houver impulso de relógio (flanco de comutação), enquanto que a saída se refere à situação presente (actual).

Por exemplo, se o circuito se encontrar no estado actual B com a entrada externa (actual) a 1, a saída (actual) é 0 *enquanto o circuito se mantiver no estado B*. Quando vier o flanco de comutação, o circuito passa para o estado seguinte D e a saída deixa de ser 0.

De forma idêntica, estando o circuito no estado actual B mas agora com a entrada externa (actual) a 0, a saída (actual) é 1 enquanto o circuito se mantiver no estado B. Quando vier o flanco de comutação, o circuito passa para o estado seguinte A e a saída deixa de ser 1.

Ou seja, e em resumo, *enquanto o circuito se encontrar no estado B, a saída actual $Z_{(t)}$ é sempre o complemento da entrada actual $X_{(t)}$. De forma semelhante, concluiríamos que outro tanto se passa quando o circuito está nos estados C e D, mas que no estado A a saída actual é sempre 0, independentemente do valor lógico aplicado à entrada actual.*

16.4 Modelos de Mealy e de Moore

O modelo da Figura 16.4 não é o único que existe para estruturar circuitos sequenciais síncronos. Este modelo foi apresentado por Mealy e, por isso, é conhecido por **modelo de Mealy**. Repete-se na Figura 16.6 por comodidade. *Modelo de Mealy*

Com efeito, há um outro modelo que, se bem que diferindo apenas num pequeno pormenor, dá origem a circuitos com um comportamento sensivelmente

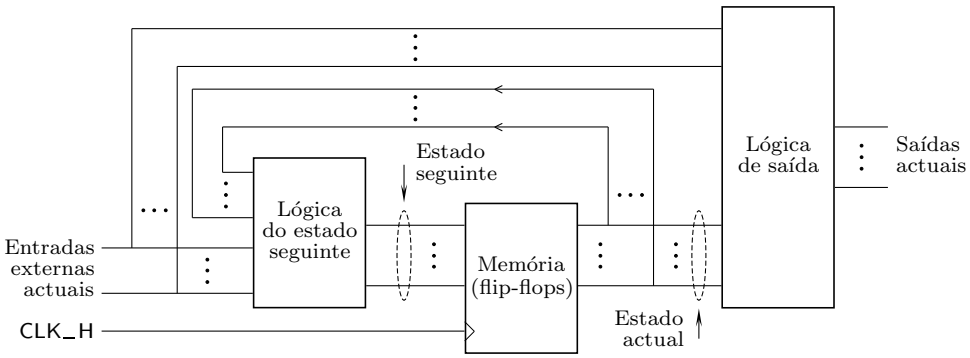


Figura 16.6: Modelo de Mealy um circuito sequencial síncrono

Modelo de Moore

diferente dos concebidos em torno do modelo de Mealy. Esse modelo alternativo, representado na Figura 16.7, é designado por **modelo de Moore**, e a única diferença consiste na lógica de saída que é apenas função dos estados do circuito e não das entradas externas.

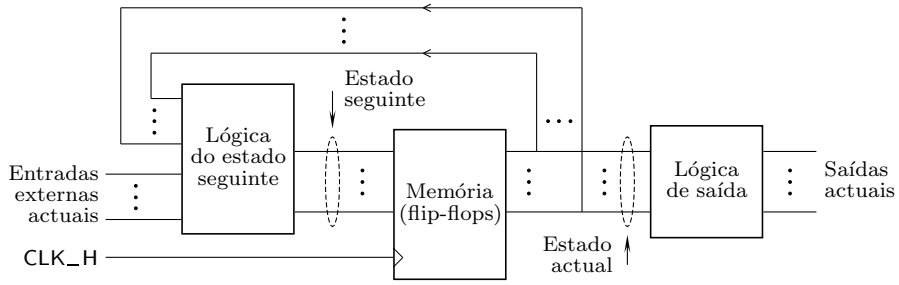


Figura 16.7: Modelo de Moore um circuito sequencial síncrono

Isso quer dizer que, no modelo de Moore, as saídas num dado instante não são sensíveis aos valores lógicos presentes nas entradas externas nesse instante, e o circuito na saída reage apenas ao seu passado e não ao seu presente. Isso tem consequências interessantes, que conduzem a diferenças entre os circuitos feitos segundo os dois modelos.

16.5 Síntese de Circuitos Sequenciais Síncronos

A síntese de circuitos sequenciais síncronos é feita de forma aproximadamente inversa à metodologia de análise dos circuitos.

O processo inicia-se pela formalização de um problema em termos de um diagrama ou de uma tabela de estados. A este nível sabemos pouco da estrutura do circuito, e apenas nos interessa estabelecer o comportamento pretendido. Assim, o nível de abstracção do diagrama ou da tabela é adequado. Em geral prefere-se começar pelo diagrama, por ser mais intuitivo na fase de construção que a tabela.

Obtido o diagrama passa-se à tabela de estados com o fim de obter uma forma adequada à determinação das equações do circuito.

Voltaremos a esta questão com mais pormenor mais à frente neste texto. Para já, na Secção 16.6 procuraremos mostrar como conceber diagramas de estados.

16.6 Exemplo de Concepção de Diagramas de Estados

Exemplifiquemos o processo de construção de um diagrama de estados com o seguinte problema: pretende-se obter um circuito que identifique a ocorrência da sequência binária 0101 na sua (única) entrada. Quando isso ocorrer, e só nessas circunstâncias, a sua saída deve exibir o valor 1.

Temos, por conseguinte, um circuito sequencial síncrono — que podemos designar por **detector da sequência 0101** — para o qual sabemos que existe uma entrada (chamemos-lhe X) e uma saída (digamos, Z), para além, naturalmente, da entrada de relógio (Figura 16.8).

Detector da sequência 0101

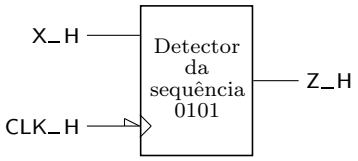


Figura 16.8: Diagrama de blocos do circuito sequencial síncrono que detecta a sequência 0101

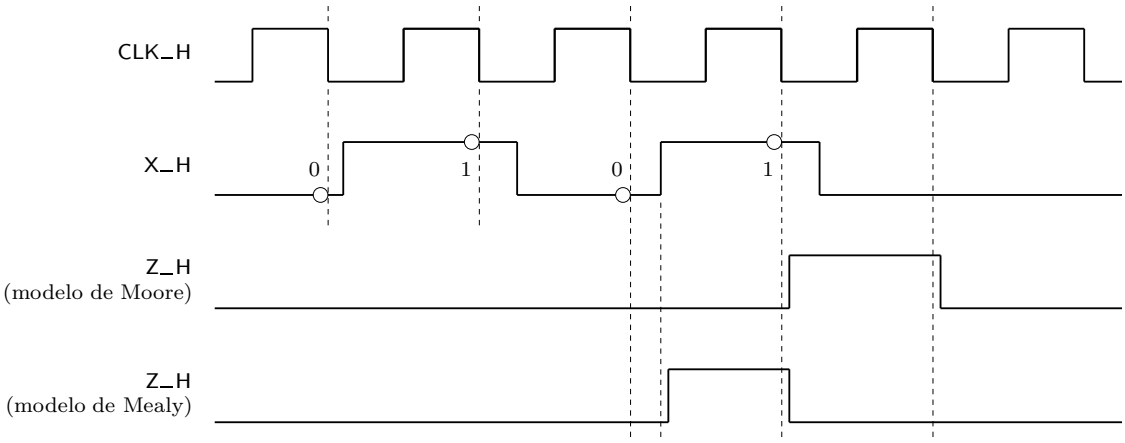
Em primeiro lugar, há que optar por construir uma máquina segundo o modelo de Mealy ou de Moore. À primeira vista pode não se vislumbrar qualquer diferença, mas uma análise um pouco mais profunda mostra-a.

Se estivermos perante um modelo de Moore, *a saída da máquina só passará a* 1 *após o flanco de relógio que surge quando está presente o último bit da sequência, e esta tiver sido completamente identificada*. Se optarmos por um modelo de Mealy, *a saída surge logo que aparece aquele último bit*.

Por outro lado, *se o último bit da sequência tiver uma duração encurtada, então a saída da máquina de Mealy também terá uma duração encurtada, enquanto que a máquina de Moore terá uma saída com a duração exacta de um período de relógio*.

Admitindo que o circuito é realizado com flip-flops que reagem nos flancos descendentes, teremos, então, diagramas temporais diferentes consoante se opte por um ou outro modelo, como mostra a Figura 16.9.

Na figura mostram-se as duas reacções. No modelo de Moore, apesar do quarto bit da sequência estar presente quase desde o fim do terceiro impulso de relógio apresentado, a reacção só surge no quarto impulso. A saída mantém-se a H enquanto o circuito se encontrar no estado para que transitou, isto é, mantém-se a H durante um período de relógio.



Notas:
(i) no caso de se optar por uma máquina de Moore, a saída Z_H vem a H depois de aparecer o último bit da sequência, porém vem com a duração de um período de relógio;
(ii) no caso de se optar por uma máquina de Mealy, a saída Z_H vem encurtada (com uma duração inferior a um período de relógio) se a entrada X_H vier a H muito tarde nesse período de relógio, porém vem coincidente com o último bit da sequência.

Figura 16.9: Diagramas temporais que ilustram a diferença de resultados obtidos na saída do detector da sequência 0101 se se optar por um modelo de Mealy ou por um modelo de Moore. Admite-se que o circuito sequencial vem implementado com flip-flops edge-triggered que comutam nos flancos descendentes

No modelo de Mealy, a saída vem actualizada logo que surge o quarto bit da sequência (ou seja, a saída vem a H praticamente em coincidência com o último bit da sequência). E no próximo impulso o circuito inicia já a busca de nova sequência. No entanto, a duração da saída a H depende da duração do último bit da sequência. E se este bit vier encurtado em relação ao período de relógio, então a saída a H também vem encurtada.

16.6.1 Concepção de diagramas de estados: modelo de Moore

Comecemos, então, por usar o modelo de Moore. Para construir um diagrama de estados há que perceber inicialmente quantas entradas e saídas tem o circuito. Neste caso é fácil. Temos uma entrada e uma saída. Isso quer dizer que cada estado terá, para além da sua designação específica, também a indicação da sua saída associada.

De cada estado sairão dois arcos, que correspondem aos dois valores lógicos da entrada (no caso das duas transições irem para o mesmo estado, pode usar-se apenas um arco do grafo).

A concepção do diagrama de estados de uma máquina sequencial síncrona assenta apenas na análise do comportamento pretendido. E vai sendo estruturado ao longo do processo. Aqui, por exemplo, iniciaremos o diagrama com um estado (Figura 16.10) que corresponde a ainda não ter surgido nenhum bit à entrada do circuito e que se chama, por isso, **estado inicial**. O estado inicial vem geralmente

Estado inicial

assinalado com uma seta que converge para ele. No nosso caso, o estado inicial terá saída 0, uma vez que não se verificam ainda as condições para a saída vir a 1 (ainda não surgiu a sequência 0101).

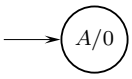


Figura 16.10: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Moore. Começa-se pelo estado inicial, designado arbitrariamente por estado A

Repare-se que, neste momento, não temos qualquer informação sobre o número de flip-flops que vão ser usados no circuito e, portanto, não podemos senão usar uma designação abstracta para cada um dos estados. Usámos a letra A para o estado inicial, mas podíamos, por exemplo, usar "Espera" ou qualquer outra designação.

No estado inicial podem ocorrer duas situações: ou surge um bit 0 na entrada ou surge um bit 1. Se surgir o bit 0, isso pode significar o início da sequência que se pretende identificar, e terá de ser memorizado. Logo, haverá que transitar para um estado (B ou "Primeiro_Bit", por exemplo). A saída de B será 0, uma vez que ainda não se verificou a ocorrência da sequência completa. No caso de surgir 1 no estado A, esse bit não é o início da sequência pretendida e, portanto, a máquina vai continuar à espera do próximo 0, que pode ser o início de uma sequência. Com entrada 1 manter-nos-emos, assim, no estado A.

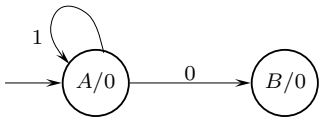


Figura 16.11: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Moore. Agora acrescentou-se um estado B e estabeleceram-se as transições que partem do estado A

No estado B (Figura 16.12), se a entrada for 1 estamos no bom caminho para detectar a sequência e passamos para um estado C que significa que, até esse estado, foram detectados os dois primeiros bits da sequência. Se, porém, a entrada for 0 no estado B, isso significa que esse 0 não é o segundo bit da sequência e, portanto, que o anterior 0 não era o primeiro bit da sequência. No entanto, pode acontecer que este 0 seja, ele próprio, o primeiro bit de uma sequência, como se pode ver no exemplo 00101 . . . Assim, nesse caso o circuito vai manter-se no estado B.

No estado C, uma entrada a 0 conduz-nos ao estado D, onde ainda não se detectou completamente a sequência mas onde se detectaram já 3 dos 4 bits. Uma entrada a 1 no estado C, porém, significa que não estamos a detectar qualquer sequência. Ao contrário do que acontece com o 0 no estado anterior, este 1 não tem qualquer possibilidade de estar de alguma forma relacionado com a sequência. Portanto, recebido este 1, só resta ao circuito voltar ao estado inicial e iniciar nova espera (Figura 16.13).

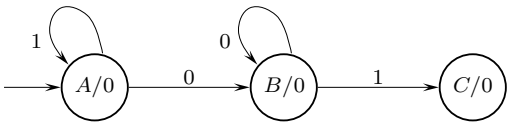


Figura 16.12: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Moore. Aos estados A e B foi acrescentado um terceiro estado, C, bem como as transições que partem de B

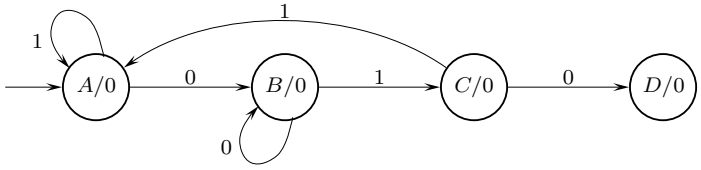


Figura 16.13: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Moore. Aos estados A, B e C foi acrescentado um quarto estado, D, bem como as transições que partem de C

Se, agora, em D entrar um 1, a sequência é detectada e avançamos para um estado E (Figura 16.14) em que, finalmente, a saída é 1. Se a entrada for 0 no estado D, a hipótese de se tratar da sequência não se verifica e transita-se para o estado B pelas razões já atrás expostas.

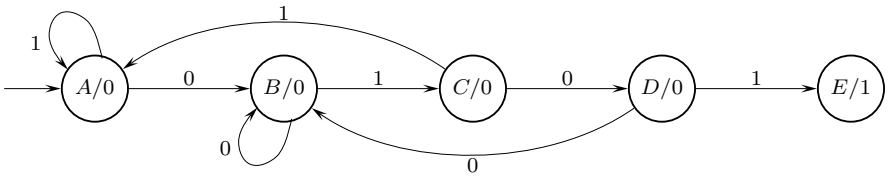
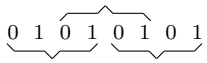


Figura 16.14: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Moore. Ainda precisamos de um estado E

O estado E não é o fim das entradas no circuito. Novos bits vão ser presentes à entrada, pelo que é necessário prever a evolução a partir do estado E.

Mas agora temos um pequeno problema devido à ambiguidade do enunciado. De facto, o que admitir quando entra um novo bit a 0? Uma primeira interpretação é que, tendo acabado a sequência anterior, este bit pode ser o primeiro da próxima sequência. Nesse caso, transitar-se-ia para o estado B.

Mas outra interpretação surge se pretendermos considerar sequências sobrepostas. Nesse caso, como se ilustra a seguir,



podemos considerar que o bit 0 entrado no estado E é, não o primeiro, mas o

terceiro bit de uma nova sequência, o que nos faria transitar para D. Optaremos pela segunda hipótese, como mostra a Figura 16.15.

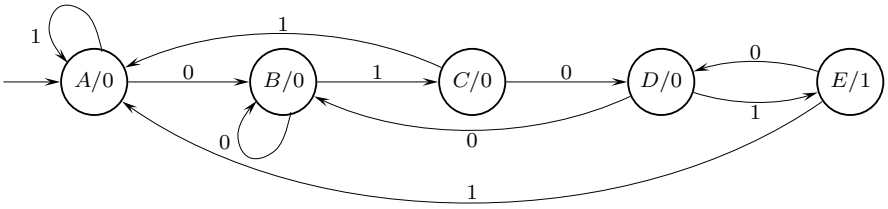


Figura 16.15: Versão final do diagrama de estados do detector da sequência 0101, usando um modelo de Moore e admitindo sequências sobrepostas

Repare-se que problemas de ambiguidade do tipo do exposto (e muito mais graves) são frequentes com descrições dos problemas em linguagem corrente e, portanto, informal. É, por isso, muito importante que, em qualquer problema de engenharia, se comece por formalizar o que se pretende. Aqui, isso é feito usando o formalismo dos diagramas de estado.

16.6.2 Concepção de diagramas de estados: modelo de Mealy

Vamos agora refazer o exemplo anterior para um modelo de Mealy. Relembre-se que se pretende obter um circuito que identifique a ocorrência da sequência 0101 na sua entrada. Quando isso ocorrer, e só nessas circunstâncias, a sua saída deve exibir o valor 1.

No modelo de Mealy a saída depende não só do estado mas também dos valores lógicos aplicados à entrada da máquina. Assim, cada estado, ao contrário do que acontece no modelo de Moore, não tem a saída associada apenas a si. A saída depende do valor na entrada. Por isso, o valor da saída é colocado junto aos arcos de transição, onde estão as entradas.

O estado inicial será, de novo, o estado A, como ilustra a Figura 16.16.

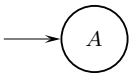


Figura 16.16: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Mealy. Começa-se pelo estado inicial, designado por estado A

Nesse estado podem surgir dois valores da entrada. Se a entrada for 0, evolui-se para o estado B pelas razões apontadas para a máquina de Moore. Se a entrada for 1, mantém-se o circuito no estado A. Em qualquer dos casos a saída é 0, como mostra a Figura 16.17.

Até se atingir o estado D o raciocínio é semelhante ao anteriormente realizado (Figura 16.18).

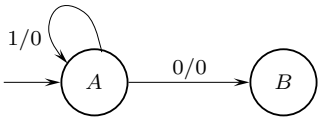


Figura 16.17: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Mealy. Acrescentou-se um estado B e as transições que partem do estado A

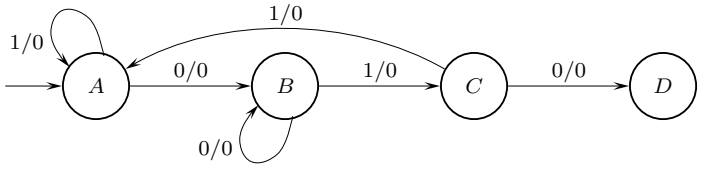


Figura 16.18: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Mealy. Acrescentaram-se os estados C e D, e as transições que partem de B e de C

Agora, como se está num modelo de Mealy, basta aparecer o valor 1 na entrada para a máquina dar imediatamente saída 1. Claro que a entrada 0 provoca a saída 0 e faz o circuito evoluir de novo para B, admitindo que esse 0 possa ser o início de uma sequência (Figura 16.19).

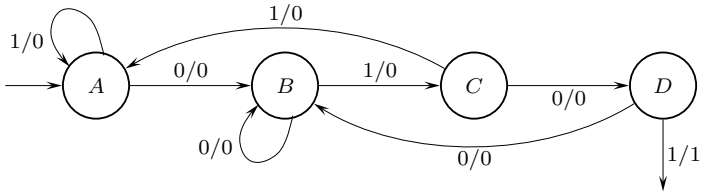


Figura 16.19: Evolução na construção do diagrama de estados do detector da sequência 0101, usando um modelo de Mealy. Mantendo os estados A a D (não são necessários mais estados), acrescentam-se mais transições admitindo sequências sobrepostas

A transição do estado D, neste modelo, é também diferente. Repare-se que a saída 1 já foi considerada. Assim sendo, não é necessário criar um estado apenas para garantir essa saída. Do estado D pode, então, transitar-se (admitindo sobreposição de sequências) directamente para o estado C, que é caracterizado por já ter entrado a sequência 01 (Figura 16.20).

Voltando agora à análise comparativa do comportamento dos circuitos projectados segundo os dois modelos referidos, detalha-se na Figura 16.21 um pouco mais o diagrama temporal da Figura 16.9, incluindo os estados que os circuitos vão assumir.

Como se vê, há realmente um comportamento diferente dos dois circuitos, embora globalmente tenham a mesma funcionalidade.

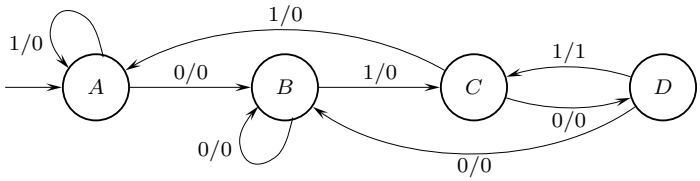


Figura 16.20: Versão final do diagrama de estados do detector da sequência 0101, usando um modelo de Mealy e admitindo sequências sobrepostas

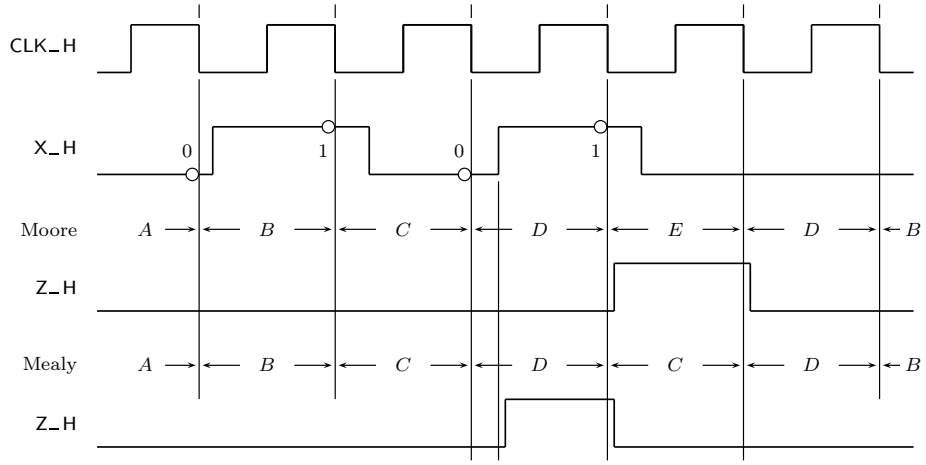


Figura 16.21: Diagramas temporais que ilustram a diferença de resultados obtidos na saída do detector da sequência 0101 se se optar por um modelo de Mealy ou por um modelo de Moore. Nesta figura detalha-se a Figura 16.9, por inclusão dos estados pelos quais as duas máquinas passam

16.7 Síntese Clássica

A síntese de circuitos sequenciais síncronos passa por uma série de passos que permitem passar de uma descrição informal de um problema para a implementação física de um circuito que tem o comportamento pretendido.

Há vários métodos para realizar essa passagem. O mais clássico descreve-se e exemplifica-se de seguida.

1. *Diagrama de estados do circuito.* Trata-se de estabelecer o comportamento pretendido num modelo formal, não ambíguo, que permite validar o que se pretende como comportamento. Trata-se de um passo opcional, uma vez que se pode, em alternativa, obter directamente a tabela de estados (ver a seguir). Mas é mais frequente e muito mais intuitivo construir o diagrama do que a tabela.

2. *Tabela de estados e de saídas do circuito.* A tabela de estados e de saídas obtém-se a partir do diagrama de estados, se o processo foi iniciado pelo diagrama de estados. Continuamos ao nivel comportamental, mas a tabela adapta-se perfeitamente à obtenção do logigrama final para o circuito.

3. *Escolha dos flip-flops.* Trata-se de determinar o número de flip-flops necessário para suportar os estados da máquina, e de escolher o seu tipo. Como é sabido, o número mínimo de flip-flops que é necessário prever é o menor inteiro que é maior ou igual ao logaritmo na base 2 do número de estados da máquina. Quanto ao tipo de flip-flops a utilizar no circuito, não existe nenhuma maneira de garantir que um determinado tipo de flip-flop conduz às equações de excitação mais simples (ver, contudo, o comentário da página 244). É nesta etapa que se gera a tabela de excitações dos flip-flops escolhidos.

4. *Codificação dos estados.* Em princípio, qualquer codificação serve. A cada codificação corresponderá, contudo, um circuito diferente e, naturalmente, algumas codificações vão gerar circuitos mais simples, enquanto outras darão origem a circuitos mais complexos. Contudo, não se dispõe de um algoritmo simples que permita determinar a configuração que conduz ao circuito mínimo. Nesta etapa obtemos a tabela de transições e de saídas do circuito.

5. *Tabela de excitações do circuito.* Depois de codificados os estados, obtemos uma tabela de excitações do circuito a partir da tabela de transições anterior e da tabela de excitações dos flip-flops escolhidos. Trata-se de uma tabela que descreve os níveis de tensão a aplicar às entradas síncronas dos flip-flops para que o comportamento temporal do circuito seja o que se estabeleceu na tabela de transições. Trata-se, pois, de uma tabela que descreve os circuitos combinatórios de excitação dos flip-flops, uma vez que na tabela todas as funções são descritas no mesmo instante t.

6. *Equações de excitação dos flip-flops.* Uma vez que a tabela de excitações do circuito descreve as excitações a aplicar num instante t genérico aos flip-flops em função dos seus estados no mesmo instante, podemos assim desenhar quadros de Karnaugh para as excitações e obter as correspondentes equações lógicas.

7. *Equações de saída do circuito.* Obtêm-se as equações das saídas do circuito a partir da tabela de transições e de saída (de notar que, nessa tabela, as saídas num instante t são definidas em função dos estados dos flip-flops no mesmo instante).

8. *Logigrama ou esquema eléctrico.* Desenha-se em seguida o logigrama do circuito, se necessário o seu esquema eléctrico.

Para exemplificar o processo vai-se obter o circuito correspondente a um dos diagramas de estados obtidos na secção anterior. Ilustrar-se-à a construção utilizando flip-flops D e flip-flops JK. O diagrama de estados que se vai utilizar é o da máquina de Mealy da Figura 16.20, que por comodidade se repete na Figura 16.22.

16.7.1 Síntese Clássica com Flip-flops D

A tabela de estados e de saídas que se obtém do diagrama de estados é lida directamente do diagrama (Tabela 16.5). O único pormenor a ter em conta é que as saídas são vistas na tabela como correspondentes a determinados pares

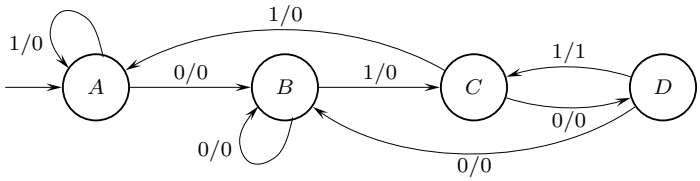


Figura 16.22: Diagrama de estados de Mealy do detector da sequência 0101 com sequências sobrepostas, a implementar com flip-flops do tipo D e do tipo JK

(estado actual, entrada actual), isto é, com tudo definido no mesmo instante, t. Pelo contrário, para os mesmos pares (estado actual, entrada actual) no instante t, os estados seguintes são definidos no instante $t + 1$.

Tabela 16.5: Tabela de estados e de saídas para o detector de sequências de Mealy da Figura 16.22

EA	ES/Z	
	X = 0	X = 1
A	$B/0$	$A/0$
B	$B/0$	$C/0$
C	$D/0$	$A/0$
D	$B/0$	$C/1$

De notar, nesta tabela, como a saída vem a 0 nos estados actuais A, B e C, e como ela vem igual a X no estado D, algo a que já tinhamos aludido anteriormente a propósito da leitura dos diagramas de estado das máquinas de Mealy (ver, por exemplo, os comentários finais da Secção 16.3, tecidos a propósito do diagrama de estados da Figura 16.5).

Como o circuito tem quatro estados, precisamos de dois flip-flops ($2^2 = 4$), que serão designados por $Q1$ e por $Q0$.

A determinação das configurações dos estados dos flip-flops que suportam cada estado do circuito (a codificação dos estados) não tem, como se disse atrás, nenhuma metodologia de resultados garantidos para a obtenção do circuito mais simples. O uso de algum bom senso pode, contudo, ajudar.

Por exemplo, no nosso caso o estado A é o estado inicial. O estado inicial terá de ser alcançado no início do funcionamento do circuito, actuando as entradas directas (assíncronas) dos flip-flops. Porque alguns flip-flops comerciais apresentam apenas entrada de CLEAR, parece razoável escolher, para o estado A, a configuração $Q1 = 0$ e $Q0 = 0$.

Por outro lado, a saída vale 1 apenas num lugar da tabela, quando a máquina está no estado actual D e $X = 1$. A função mais simples que se pode conceber é, então, o produto lógico entre as saídas dos flip-flops e a variável de entrada. Para isso, conviria que o estado D fosse codificado com $Q1 = 1$ e $Q0 = 1$, o que permite gerar uma função muito simples de implementar, $Z = X\,Q1\,Q0$.

As restantes configuração são irrelevantes com o nosso conhecimento actual do circuito. Assim sendo, escolhe-se, por exemplo, a codificação dos estados que se apresenta na Tabela 16.6 (às saídas dos flip-flops é usual chamar **variáveis de estado** do circuito).

Variáveis de estado

Tabela 16.6: Tabela com a codificação dos estados do detector de sequências de Mealy da Figura 16.22

Estado	**Q1_H**	**Q0_H**
A	L	L
B	L	H
C	H	L
D	H	H

Substituindo, na anterior tabela de estados do circuito, os estados do circuito pelos estados dos flip-flops obtém-se a tabela de transições e de saídas da Tabela 16.7.

Tabela 16.7: Tabela de transições e de saídas do detector de sequências de Mealy da Figura 16.22

EA		**ES/Z**					
		$X\_H_{(t)} = L$			$X\_H_{(t)} = H$		
$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$Q1\_H_{(t+1)}$	$Q0\_H_{(t+1)}$	/ $Z\_H_{(t)}$	$Q1\_H_{(t+1)}$	$Q0\_H_{(t+1)}$	/ $Z\_H_{(t)}$
L	L	L	H	/ L	L	L	/ L
L	H	L	H	/ L	H	L	/ L
H	L	H	H	/ L	L	L	/ L
H	H	L	H	/ L	H	L	/ H

Esta tabela mostra-nos como deverão evoluir as variáveis de estado para satisfazer o comportamento pedido para o circuito.

Para em seguida obtermos a tabela de excitações do circuito, consideremos o seguinte: como sabemos da Secção 13.2 (ver a Tabela 13.3), para conseguir que um flip-flop D assuma um determinado nível de tensão na saída Q_H depois de ocorrido um flanco de comutação, basta colocar na entrada síncrona esse nível de tensão antes de ocorrer o flanco. Por isso, para passar da tabela de transições anterior para a tabela de excitações do circuito, basta mudar o nome das colunas, como se faz na Tabela 16.8.

De notar que se retiraram desta tabela as saídas do circuito, dado que as podemos gerar a partir da tabela de transições e de saídas anteriormente obtida.

Podemos agora obter as equações lógicas de $D1$, de $D0$ e de Z em função de

Tabela 16.8: Tabela de excitações do detector de sequências de Mealy da Figura 16.22 que usa flip-flops do tipo D

EA		**Excitações dos flip-flops**			
		$X\_H_{(t)} = L$		$X\_H_{(t)} = H$	
$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$D1\_H_{(t)}$	$D0\_H_{(t)}$	$D1\_H_{(t)}$	$D0\_H_{(t)}$
L	L	L	H	L	L
L	H	L	H	H	L
H	L	H	H	L	L
H	H	L	H	H	L

$Q1$, de $Q0$ e de X recorrendo aos quadros de Karnaugh da Figura 16.23:

$$D1 = X\,Q0 + \overline{X}\,Q1\,\overline{Q0}$$
$$D0 = \overline{X}$$
$$Z = X\,Q1\,Q0$$

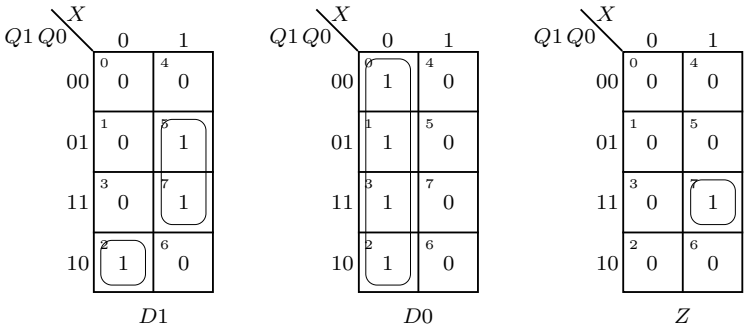


Figura 16.23: Quadros de Karnaugh para as excitações dos flip-flops D e para a saída do detector de sequências de Mealy da Figura 16.22

Finalmente, podemos desenhar na Figura 16.24 o logigrama do detector de sequências que temos vindo a sintetisar.

Notemos que *guardámos até esta altura a decisão sobre o modo de actuação dos flip-flops cujo tipo foi anteriormente escolhido. Ou seja, todo o processo de síntese anterior apenas necessitou de saber que usámos flip-flops do tipo D. Só quando queremos desenhar o logigrama é que temos de decidir se são flip-flops master-slave ou edge-triggered, a comutar nos flancos ascendentes ou descendentes.*

No logigrama do nosso detector de sequências escolhemos, arbitrariamente, utilizar flip-flops D edge-triggered a comutar nos flancos descendentes.

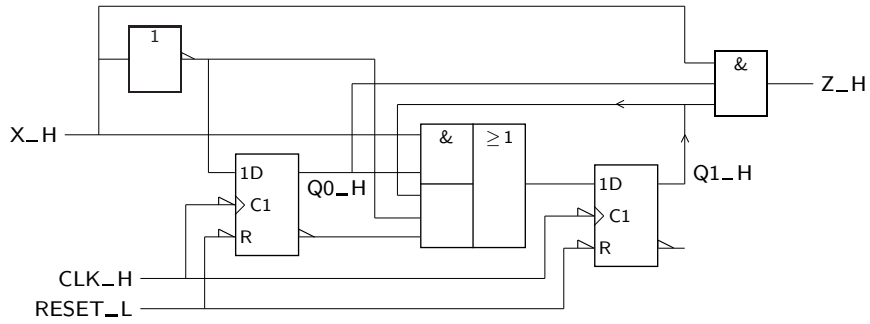


Figura 16.24: Logigrama do detector de sequências de Mealy que utiliza flip-flops D

16.7.2 Síntese Clássica com Flip-flops JK

Como é evidente, todos os passos até à definição da tabela de transições e de saídas do circuito (Tabela 16.7) são idênticos ao caso em que se trabalha com flip-flops D. Naturalmente, agora precisamos de utilizar a tabela de excitações dos flip-flops JK (Tabela 14.6 na página 245), em vez da tabela correspondente dos flip-flops D.

Combinando a tabela de transições e a tabela de excitações dos flip-flops JK, obtém-se uma tabela de excitações do circuito idêntica à da Tabela 16.8 em que se especifica o nível de tensões a aplicar às entradas dos flip-flops para obter a evolução pretendida.

Por exemplo para o quadrado assinalado a **negrito** na Tabela 16.9, indicam-se os níveis de tensão a colocar nas entradas síncronas J_H e K_H dos dois flip-flops para que o estado do primeiro se mantenha a L e o do segundo evolua de L para H, como especificado na parte esquerda da tabela para o estado actual (L,L) e entrada actual $X\_H = L$.

Obtemos, assim, uma tabela em que temos a epecificação dos níveis a aplicar a J e a K em função dos estados actuais e da entrada actual.

Tal como anteriormente, podemos obter os mapas de Karnaugh para os J e K na Figura 16.25 e as equações

$$\begin{aligned} J1 &= X\,Q0 \\ K1 &= \overline{X}\,Q0 + X\,\overline{Q0} = X \oplus Q0 \\ J0 &= \overline{X} \\ K0 &= X\,. \end{aligned}$$

A equação da saída Z é idêntica à obtida para os flip-flops D, uma vez que a função de saída não se altera com o tipo de flip-flops.

O circuito terá, portanto, o logigrama da Figura 16.26.

Como se pode observar, o circuito total possui complexidade idêntica à do circuito obtido com flip-flops D. Deve, contudo, recordar-se a nota à margem da página 244, que afirma que é mais provável obter circuitos de excitação mais simples se se utilizarem flip-flops JK.

Tabela 16.9: Construção da tabela de excitações do detector de sequências de Mealy da Figura 16.22, com a utilização de flip-flops JK, a partir da tabela de transições do circuito

EA		**ES**			
		$X\_H_{(t)} = L$		$X\_H_{(t)} = H$	
$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$Q1\_H_{(t+1)}$	$Q0\_H_{(t+1)}$	$Q1\_H_{(t+1)}$	$Q0\_H_{(t+1)}$
L	**L**	**L**	**H**	L	L
L	H	L	H	H	L
H	L	H	H	L	L
H	H	L	H	H	L

← Tabela de transições

Tabela de excitações ↓

EA		**Excitações dos flip-flops**							
		$X\_H_{(t)} = L$				$X\_H_{(t)} = H$			
$Q1\_H_{(t)}$	$Q0\_H_{(t)}$	$J1\_H_{(t)}$	$K1\_H_{(t)}$	$J0\_H_{(t)}$	$K0\_H_{(t)}$	$J1\_H_{(t)}$	$K1\_H_{(t)}$	$J0\_H_{(t)}$	$K0\_H_{(t)}$
L	**L**	**L**	**X**	**H**	**X**	L	×	L	×
L	H	L	×	×	L	H	×	×	H
H	L	×	L	H	×	×	H	L	×
H	H	×	H	×	L	×	L	×	H

16.8 Síntese com um Flip-flop por Estado

Uma forma alternativa de implementar um circuito sequencial síncrono consiste em utilizar um flip-flop D por cada estado do circuito. Trata-se de um método que não garante a simplificação do logigrama do circuito, mas que é bastante estruturado e permite realizações interessantes com os componentes programáveis actualmente disponíveis.

Por outro lado, com circuitos que tenham muitas entradas ou estados, pode mesmo ser a metodologia mais adequada por não obrigar a uma visão global do circuito na fase de concepção, e por conduzir a uma síntese muito simplificada — não há, neste método, que estabelecer as tabelas de transições e de saídas, nem as tabelas de excitação do circuito, nem os mapas de Karnaugh para as excitações e saídas, que podem ter dimensões muito grandes se o número de entradas e de estados for elevado.

A existência de um flip-flop por cada estado do circuito, é claro, conduz a mais flip-flops do que os que são necessários com as metodologias anteriores. Em cada momento, só um dos flip-flops vê a sua saída Q_H activada, e esse flip-flop determina o estado em que o circuito se encontra. No estado inicial há que activar a saída do respectivo flip-flop (o flip-flop que corresponde ao estado inicial) e desactivar os restantes.

Se olharmos para as saídas dos n flip-flops do circuito, apenas com uma activa e todas as outras inactivas em cada impulso de relógio, é como se estivessemos a descrever as diversas palavras de um código 1-em-n.

A simplicidade de síntese que este método permite resulta de podermos pôr em

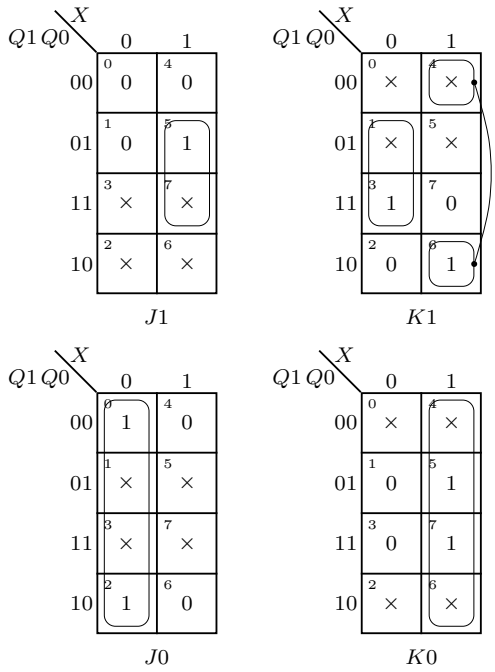


Figura 16.25: Quadros de Karnaugh para as excitações dos flip-flops JK e para a saída do detector de sequências de Mealy da Figura 16.22

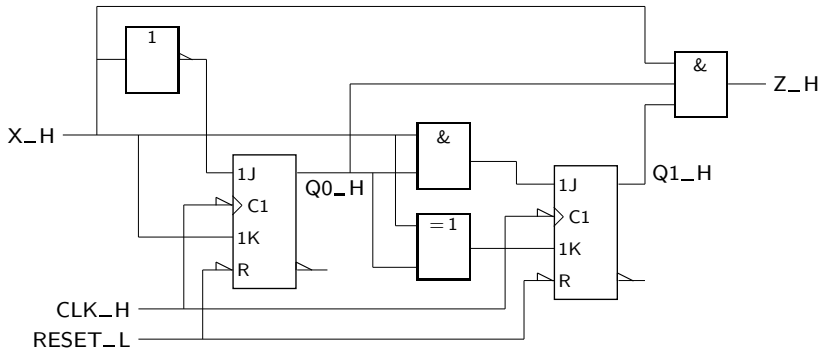


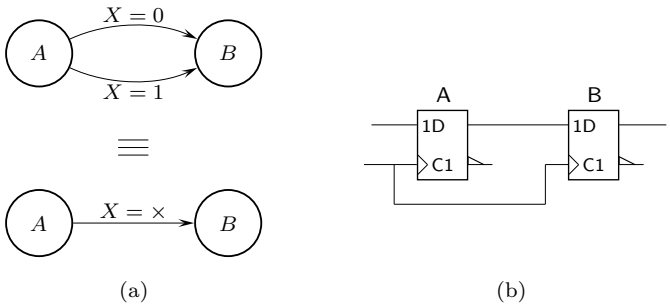
Figura 16.26: Logigrama do detector de sequências de Mealy que utiliza flip-flops JK

correspondência biunívoca determinadas partes do diagrama de estados e partes do logigrama final do circuito. Vamos examinar as principais.

Transição incondicional

Na Figura 16.27(a) representa-se uma **transição incondicional** do estado A para o estado B (por transição incondicional entende-se uma transição que não é condicionada pelos valores lógicos nas entradas externas).

A figura representa as duas notações possíveis para este tipo de transição, admitindo que apenas uma entrada externa, X, está em jogo. A Figura 16.27(b), por seu lado, ilustra o troço de logigrama equivalente à transição, com uma ligação directa do flip-flop A ao flip-flop B.



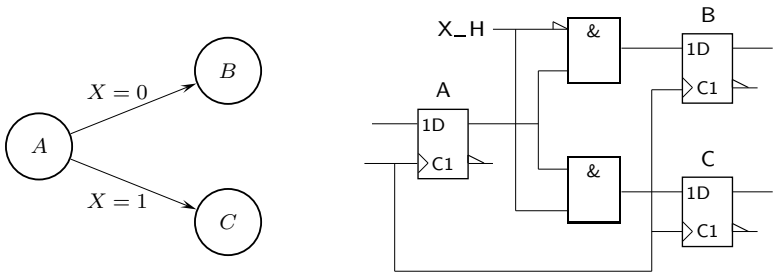
(a) (b)

Figura 16.27: (a) Uma transição incondicional do estado A para o estado B no diagrama de estados (isto é, uma transição que não depende dos valores lógicos na entrada X), tem por correspondência (b) um troço do logigrama (com um flip-flop por estado) que se traduz pela ligação do flip-flop A ao flip-flop B

Relembrando que, numa implementação com um flip-flop por estado, apenas um flip-flop de cada vez tem a sua saída Q_H activada, constatamos que, se num determinado impulso de relógio é o flip-flop A que tem a saída activa e, portanto, a saída de B está inactiva, no impulso de relógio seguinte os flip-flops vêem as suas saídas com as actividades respectivas trocadas.

Transição condicionada ou "Fork"

A Figura 16.28 ilustra um outro tipo de transição, designada por **transição condicionada** ou **"Fork"**.



(a) (b)

Figura 16.28: (a) Uma transição do estado A para um dos estados B ou C condicionada aos valores lógicos numa entrada, X, tem por correspondência (b) um troço do logigrama (com um flip-flop por estado) que se traduz pela ligação do flip-flop A aos flip-flops B ou C consoante o valor lógico aplicado a X

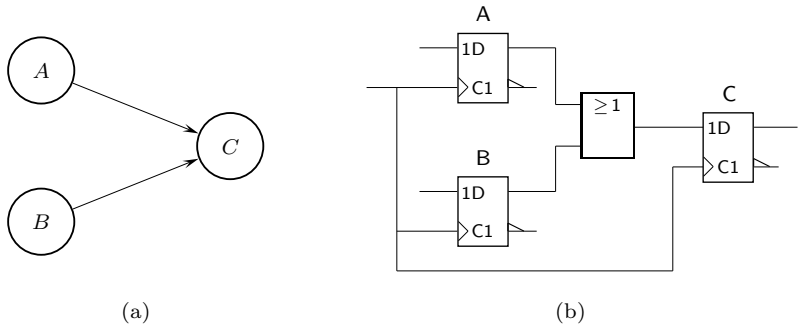
Trata-se de uma transição que depende dos valores lógicos aplicados às entradas externas num determinado estado. Na figura apresenta-se a transição do estado A para os estados B ou C, condicionada ao valor lógico aplicado numa única entrada, X, quando o circuito está no estado A.

De notar que era possível estender este conceito a mais do que uma entrada, porém *tendo em atenção que k entradas externas dão origem a 2^k transições a partir do estado de origem: nesse caso podíamos usar um demultiplexer controlado pelas entradas para encaminhar a saída do estado de origem para um dos 2^k estados de destino*.

Relembrando, mais uma vez, que numa implementação com um flip-flop por estado, apenas um flip-flop de cada vez tem a sua saída Q_H activada, constatamos que, se num determinado impulso de relógio é o flip-flop A que tem a saída activa e, portanto, as saídas de B e de C estão inactivas, no impulso de relógio seguinte um dos flip-flops B ou C (mas apenas um deles) vê a sua saída activada, consoante o valor da entrada X no estado A.

Convergência ou "Join"

A Figura 16.29 ilustra um terceiro tipo de transição, designada por **convergência** ou **"Join"**.



(a) (b)

Figura 16.29: (a) Uma transição do estado A ou do estado B para o estado C (convergência), tem por correspondência (b) um troço do logigrama (com um flip-flop por estado) que se traduz pela ligação do flip-flop A *ou* do flip-flop B ao flip-flop C

Trata-se de transições alternativas que não dependem dos valores lógicos aplicados às entradas externas. Na figura apresenta-se a transição do estado A ou do estado B para o estado C, pelo que se utiliza uma porta OR para assegurar a convergência.

Dado que numa implementação com um flip-flop por estado, como sabemos, apenas um flip-flop de cada vez tem a sua saída Q_H activada, verificamos que, se num determinado impulso de relógio é o flip-flop A (o flip-flop B) que tem a saída activada e, portanto, as saídas de B (de A) e de C estão inactivas, no impulso de relógio seguinte será o flip-flop C que vê a sua saída activada. Em qualquer dos casos, será sempre a vez de C vir activado depois de A ou de B.

Finalmente, consideremos a geração das saídas, em que temos quatro casos, ilustrados na Figura 16.30.

Saídas de Mealy

Nas Figuras 16.30(a) e (b) representam-se **saídas de Mealy** ou **saídas condicionadas**, porque *dependem* dos valores lógicos aplicados às entradas (apenas uma entrada e uma saída estão representadas nessas figuras). Assim, na Figura 16.30(a) temos $Z = X$ no estado A, enquanto que na Figura 16.30(b) temos $Z = \overline{X}$ no mesmo estado A.

Saídas de Moore

Por seu turno, nas Figuras 16.30(c) e (d) representam-se **saídas de Moore** ou **saídas incondicionais**, porque *não dependem* dos valores lógicos aplicados às entradas. Por exemplo, na Figura 16.30(c) temos $Z = 0$ no estado A, enquanto que na Figura 16.30(d) temos $Z = 1$ no estado A, em ambos os casos de forma independente do valor na entrada X nesse estado.

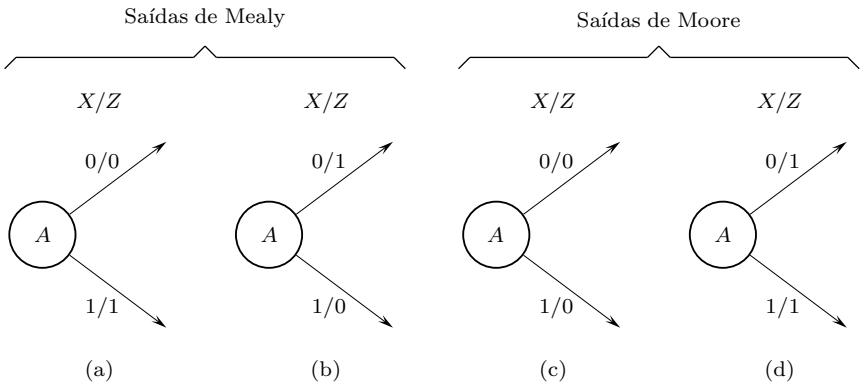


Figura 16.30: (a) e (b) Saídas de Mealy, porque os seus valores dependem dos valores nas entradas (apenas uma entrada e uma saída se encontram representadas); e (c) e (d) saídas de Moore, que não dependem dos valores nas entradas

A geração das saídas é diferente para os quatro casos, como mostra a Figura 16.31.

Retomemos agora na Figura 16.32 o diagrama de estados do detector de sequências de Mealy da Figura 16.22, e vamos sintetizá-lo directamente (com um flip-flop por estado), utilizando os conhecimentos entretanto adquiridos sobre a geração de partes do logigrama do circuito.

Começamos por considerar, na Figura 16.33, os 4 flip-flops correspondentes aos 4 estados. Na figura inclui-se já a inicialização do circuito, fazendo o Set do flip-flop A e o Reset dos outros.

Para começarmos a construir o logigrama, pensemos no seguinte. Para além da inicialização, entra-se no estado A por dois caminhos possiveis, como se pode ver do facto de para ele convergirem duas setas no diagrama de estados: vai-se para A se o circuito estiver no estado A e a entrada X for 1, ou se estiver no estado C e $X = 1$. Esta dupla condição pode"colocar-se" directamente no logigrama do circuito, como mostra a Figura 16.34.

Por outro lado, ao estado B chega-se por três caminhos possíveis, como ilustrado na Figura 16.35: do estado A com $X = 0$, do estado B com $X = 0$, ou do estado D com $X = 0$.

O resto do circuito é, agora, simples de obter (Figura 16.36).

16.9 Fluxogramas

Uma forma alternativa de especificar circuitos sequenciais utiliza **fluxogramas** que, no fundo, são formas alteradas de diagramas de estados. Os fluxogramas são representações mais compactas que os diagramas de estados, mas contêm a mesma informação.

Com os fluxogramas procura-se, em cada situação, apresentar apenas a informação relevante, o que os torna particularmente úteis quando o número de entradas e de saídas é elevado.

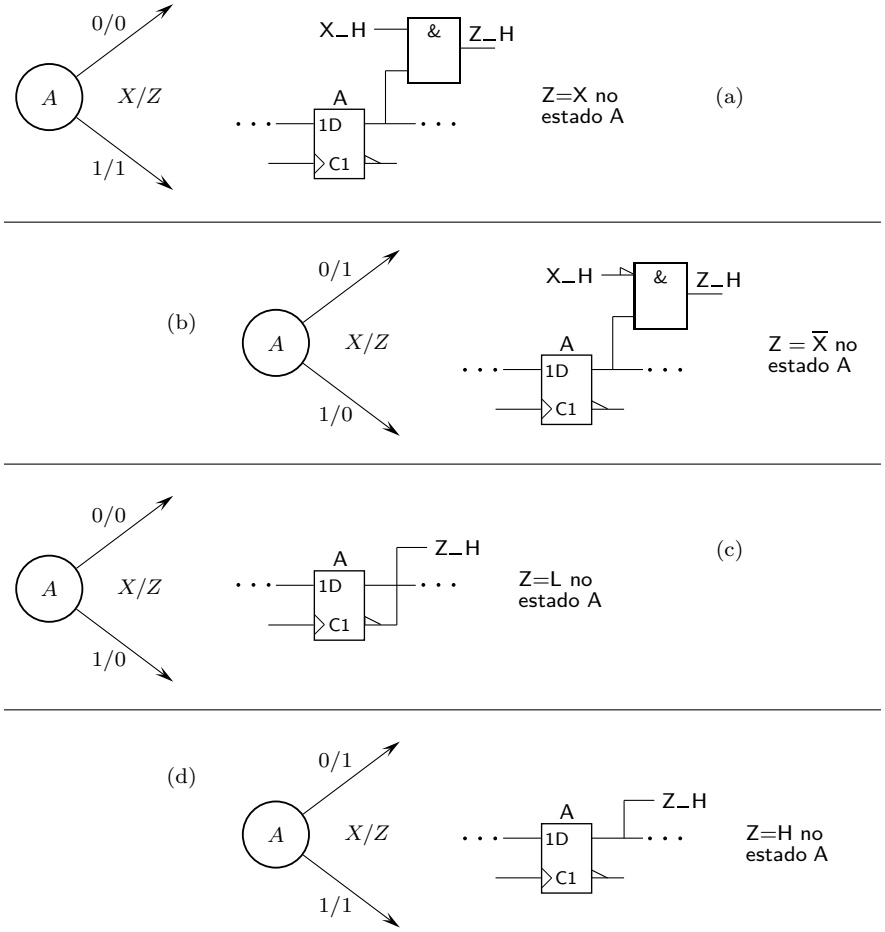


Figura 16.31: (a) e (b) A geração de uma saída de Mealy necessita de uma porta AND com a polaridade adequada na entrada que está ligada à entrada externa; porém, (c) e (d) as saídas de Moore dispensam a existência de portas AND

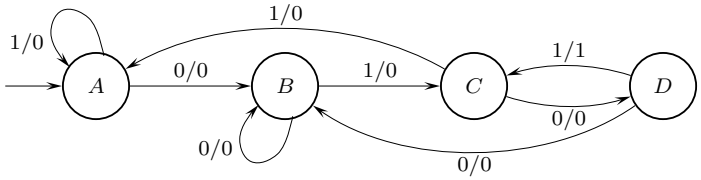


Figura 16.32: Diagrama de estados de Mealy do detector da sequência 0101 com sequências sobrepostas, a implementar usando um flip-flop por estado

Com efeito, como já sabemos, os diagramas de estado aumentam exponencialmente as suas dimensões com o aumento do número de entradas externas, visto que, de cada estado actual, devem partir 2^k transições para os estados seguintes, se k for o número de entradas externas. Por outro lado, precisamos de indicar, para cada transição (num diagrama de Mealy) ou para cada estado (num dia-

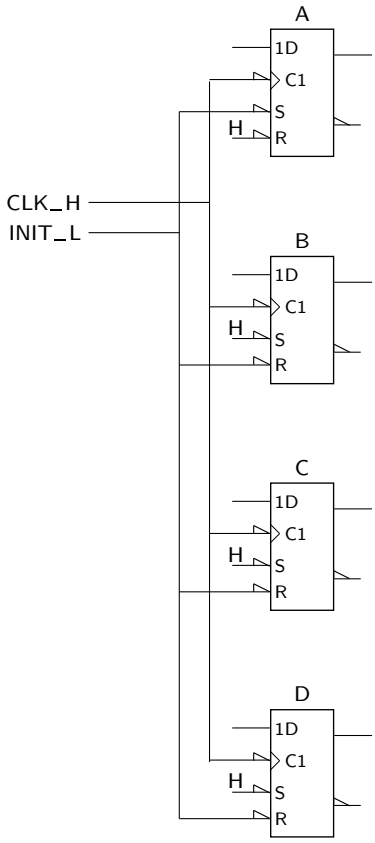


Figura 16.33: Construção do logigrama do detector da sequências de Mealy da Figura 16.32, usando um flip-flop por estado. Nesta fase, apenas se incluem os flip-flops e a inicialização do circuito

grama de Moore), os níveis de tensão (ou valores lógicos) em todas as saídas, o que torna difícil a leitura do diagrama.

Num fluxograma, pelo contrário, não se indicam todas as entradas (e transições) a partir de um dado estado actual, mas apenas as que são relevantes. Segue-se que o número total de transições vem, com um fluxograma, substancialmente reduzido. Identicamente, em cada estado actual ou transição a partir desse estado apenas se indicam, num fluxograma, as saídas que estão activas, o que facilita consideravelmente a sua leitura.

Num fluxograma um estado é representado por um rectângulo, e as entradas surgem em losangos que representam decisões. As saídas, no caso dos modelos de Moore, são representadas dentro dos estados. No caso dos modelos de Mealy, são representadas por um símbolo próprio formado por um rectângulo com os lados substituídos por arcos de circunferência.

Exemplifiquemos com o seguinte caso: considere-se uma estrada de montanha tão estreita que apenas deixa passar uma viatura. A estrada é, como quase todas as estradas, de sentido duplo na planície, mas é de sentido único na zona de montanha (Figura 16.37, na página 310).

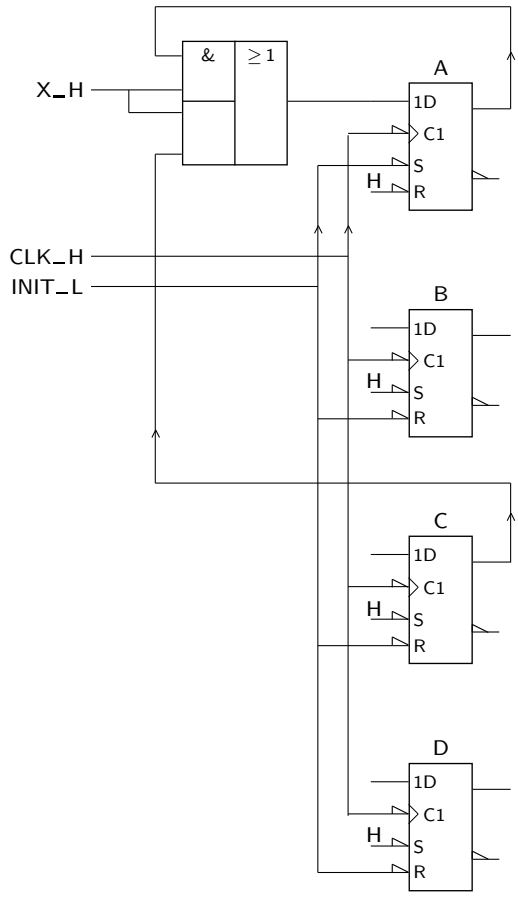


Figura 16.34: Continuação da construção do logigrama anterior, agora com as excitações no estado A

Para resolver o problema instalaram-se duas cancelas nos dois extremos do troço estreito. Há seis detectores (sensores) na estrada nos locais assinalados por $D1$ a $D6$, e dois semáforos designados por $S1$ e $S2$.

O funcionamento pretendido para o circuito de controlo das cancelas é o seguinte: as cancelas estão normalmente fechadas, e os semáforos $S1$ e $S2$ normalmente em vermelho.

Quando surge uma viatura, por exemplo do lado esquerdo, pisa o detector $D1$. Se não houver nenhum carro a deslocar-se no troço estreito, o semáforo $S1$ passa a verde, a cancela $C1$ abre, e a viatura entra. Logo que passa no detector $D2$, essa barreira é fechada, o semáforo volta a vermelho e a situação fica estável neste estado até a viatura sair do sistema. Quando a viatura chega ao detector $D3$, a cancela $C2$ abre e permanece aberta até a viatura pisar o detector $D4$. Então, a cancela fecha.

Se entretanto chegar uma viatura a qualquer dos lados, espera que a primeira saia e só então se inicia de novo o processo no mesmo sentido ou no sentido inverso, conforme o sentido de chegada da viatura. No caso de chegarem duas viaturas ao mesmo tempo, dá-se prioridade ao sentido da esquerda para a direita.

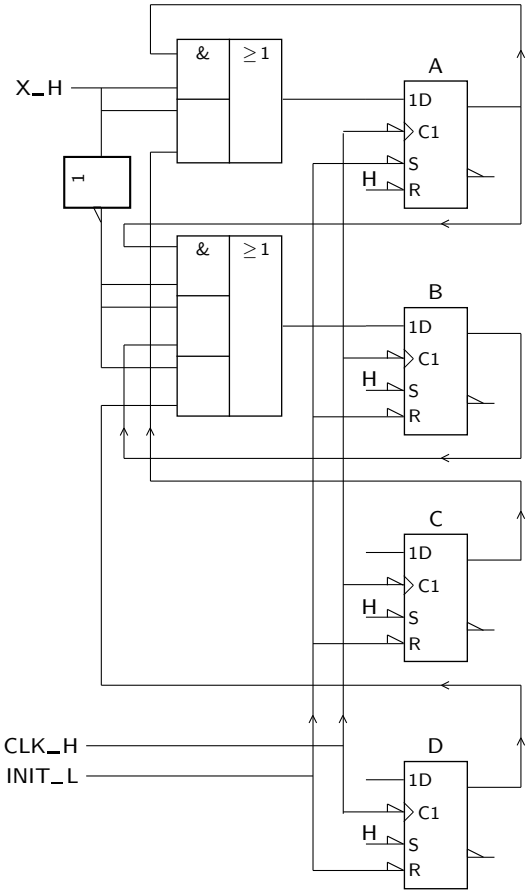


Figura 16.35: Continuação da construção do logigrama anterior, com as excitações no estado B

Esta última especificação serve para simplificar o fluxograma (depois de perceber como funciona o método, tente refazer o problema dando prioridade ao sentido inverso ao da última passagem).

Como é evidente, é difícil desenhar um diagrama de estados para este circuito, uma vez que o circuito tem 6 entradas (os detectores, já que as cancelas e os semáforos são saídas) e, por conseguinte, $2^6 = 64$ transições a partir de cada um dos estados. Por isso, *o fluxograma é mais interessante do que o diagrama de estados, uma vez que em cada estado se vai ter em conta apenas a ou as entradas relevantes para a evolução a partir desse estado*.

Aos estados podem ser dadas designações apropriadas. No caso de um modelo de Moore, as saídas e os correspondentes níveis de actividade são indicadas nos estados. Porém, *em cada estado apenas se indicam as saídas que estão activas*, o que permite simplificar consideravelmente a leitura do fluxograma.

Assim, parte-se de um estado inicial, $E0$, em que o circuito espera que surja uma viatura. A partir desse estado há três hipóteses de evolução:

— não surge viatura alguma e continua-se no mesmo estado; ou

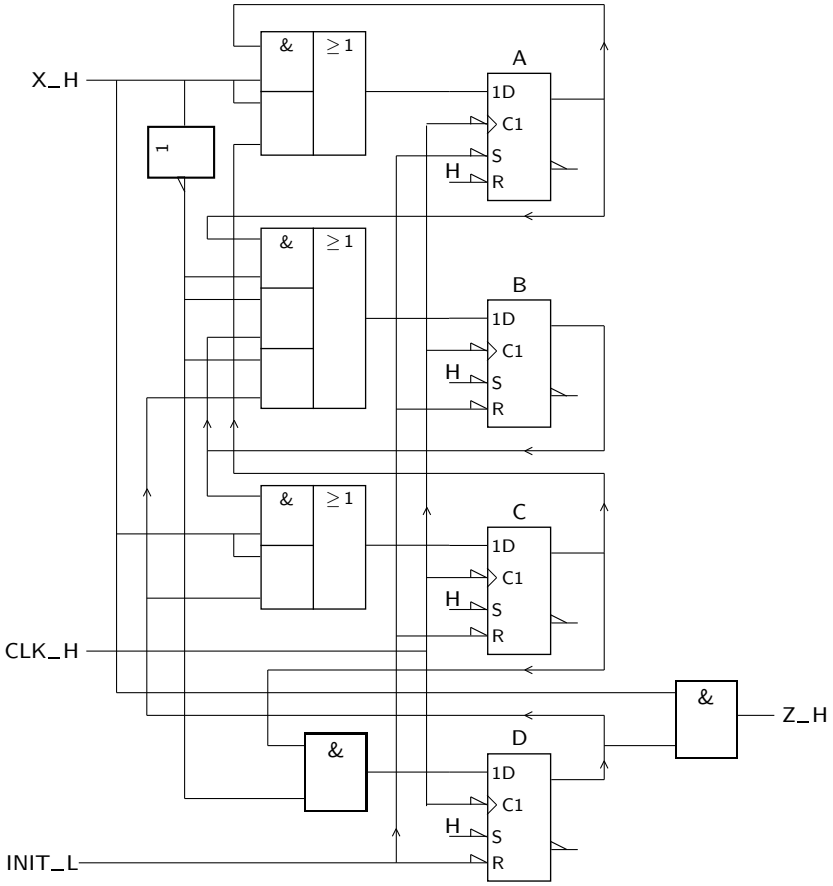


Figura 16.36: Logigrama final do detector da sequências de Mealy da Figura 16.32, implementado com um flip-flop por estado

Figura 16.37: Uma estrada de montanha com duas cancelas, $C1$ e $C2$, seis sensores (detectores), $D1$ a $D6$, e dois semáforos, $S1$ e $S2$

— surge uma viatura detectada por $D1$ do lado esquerdo — inicia-se o processo de atravessamento da esquerda para a direita; ou

— surge uma viatura detectada por $D5$, e não há viatura em $D1$ — inicia-se o

processo de atravessamento da direita para a esquerda.

Este início pode ser, assim, especificado pela parte do fluxograma da Figura 16.38.

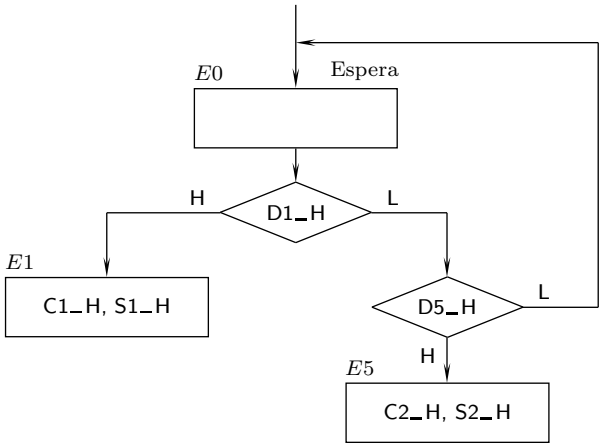


Figura 16.38: Início do fluxograma de Moore de controlo do sistema da Figura 16.37

Admite-se que todas as entradas e saídas são activas a H e que: (i) a activação de uma cancela significa levantá-la; e que (ii) a activação de um semáforo significa pô-lo verde.

Repare-se que, no estado $E0$, se colocou um comentário indicando que o estado é o de Espera inicial. Por outro lado, as saídas nesse estado estão todas desactivadas, o que significa que as duas cancelas estão baixadas e que os dois semáforos estão vermelhos. É de facto, a situação inicial do sistema.

No estado $E1$ a saída $C1$ vem activada (o que corresponde a levantar a cancela) e a saída $S1$ também (o que corresponde ao semáforo verde). Por outro lado, neste estado $C2$ e $S2$ estão inactivas, o que significa que a cancela da direita na Figura 16.38 está baixada e que o semáforo correspondente está vermelho.

Em $E5$ são as saídas $C2$ e $S2$ que vêm activas e $C1$ e $S1$ que vêm inactivas, e os papéis das cancelas e dos semáforos vêm trocados em relação aos do estado $E1$.

O estado $E1$ corresponde ao início do atravessamento da esquerda para a direita, e $E5$ ao início do atravessamento da direita para a esquerda.

O resto do fluxograma deve ser mais ou menos evidente (Figura 16.39).

De realçar o seguinte em relação à parte esquerda do fluxograma (conclusões semelhantes podem ser extraídas da análise da parte direita):

— no estado $E2$ a viatura já entrou na estrada de montanha, as cancelas estão fechadas e os semáforos estão vermelhos;

— no estado $E3$ a viatura ainda está no troço de montanha, mas já pisou o sensor $D3$, pelo que a cancela $C2$ abre;

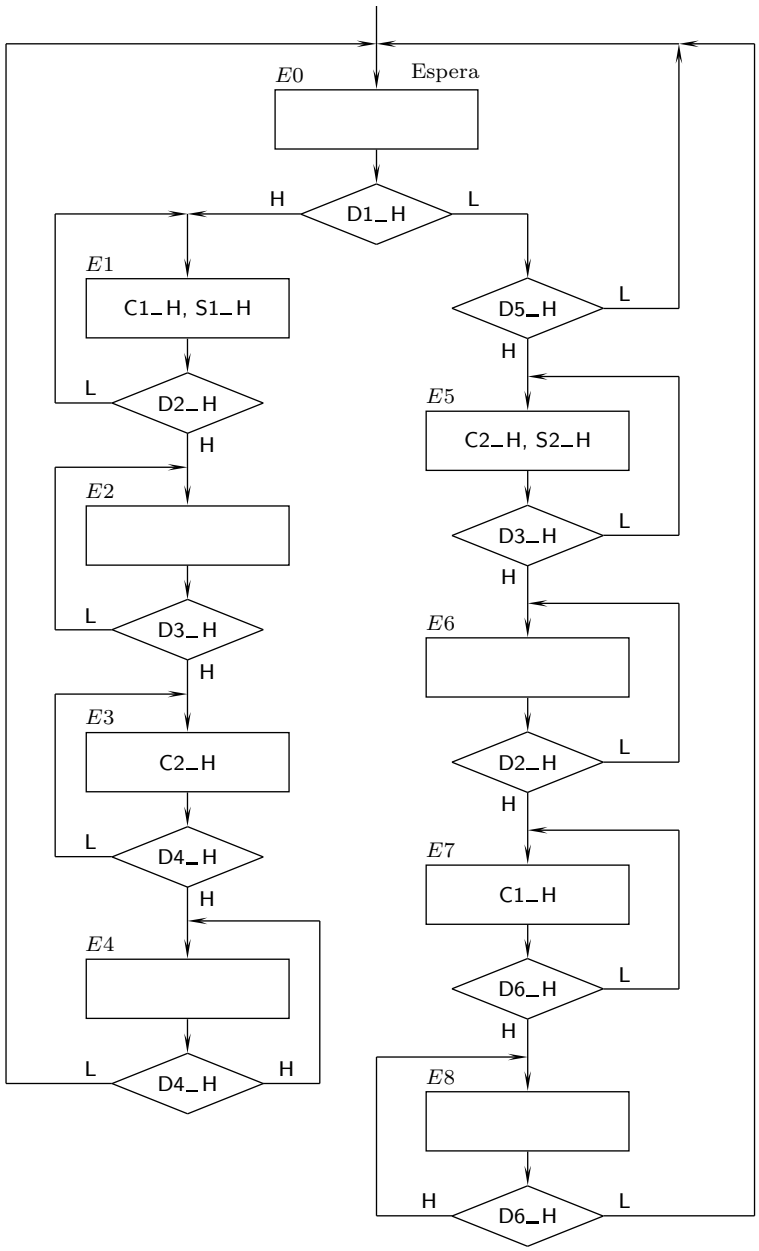


Figura 16.39: Fluxograma de Moore de controlo das cancelas e dos semáforos da Figura 16.37

Por exemplo, se a frequência de relógio for de 1 MHz, ou seja, se o período de relógio for de 1 µs, o circuito dá 10^6 voltas ao estado $E4$ num segundo.

— no estado $E4$ a viatura está prestes a sair do troço de montanha para a estrada com duas vias, porque já pisou o sensor $D4$; a cancela $C2$ fecha, o semáforo $S2$ continua vermelho e as condições na cancela e no semáforo da esquerda da Figura 16.37 mantêm-se; neste estado espera-se que a viatura deixe de pisar o sensor $D4$ — enquanto estiver a pisá-lo, o que pode demorar vários ciclos de relógio, o circuito mantém-se no estado $E4$, e só quando deixar de o pisar é que o circuito se reinicia, voltando ao estado $E0$ e atendendo a

eventuais pedidos em $D1$ ou em $D5$, sendo o primeiro prioritário.

Como se viu, o fluxograma anterior é um fluxograma que representa um circuito de Moore. As saídas estão dependentes, em exclusivo, dos estados.

Exemplifiquemos agora com um fluxograma com saídas segundo o modelo de Mealy. Neste caso, vamos repetir o exemplo anterior mas com uma pequena alteração: o semáforo $S1$ fica activo (verde) apenas até o veículo abandonar o detector $D1$. Logo que isso acontece, o sinal volta a vermelho para impedir que um segundo veículo siga o primeiro.

A alteração consiste em tornar a saída $S1$ dependente não só do estado $E1$ como da entrada $D1$. Trata-se de uma saída de Mealy, por oposição às do fluxograma da Figura 16.39, que são saídas de Moore.

O símbolo de uma saída de Mealy num fluxograma é o que se indica na Figura 16.40(a). Este símbolo deve ser utilizado sempre que, num determinado estado, A por exemplo, se testa um certo número de condições (variáveis de entrada) e se decide que, quando as condições vêm satisfeitas, se devem activar determinadas funções de saída. Naturalmente, tal pressupõe que, se as condições de entrada não vierem satisfeitas, as saídas não vêm activadas nesse estado [Figura 16.40(b)].

Por contraste, as saídas de Moore [Figura 16.40(c)] virão activadas no próprio estado, independentemente dos testes que se fizerem, nesse estado, às entradas.

O resultado final para o circuito dá o fluxograma da Figura 16.41.

No caso do funcionamento do circuito ser representado por um fluxograma, é bastante fácil a sua implementação usando um flip-flop por estado. Basta, para tanto, obter as relações entre partes do fluxograma e partes do logigrama final, tal como se fez anteriormente para o caso dos diagramas de estado.

Por exemplo, as transições incondicionais são implementadas como se ilustra na Figura 16.42(a), enquanto que as transições condicionadas (ou "Forks") e as convergências (ou "Joins") são implementadas com indicam as Figuras 16.42(b) e (c).

Quanto às saídas de Moore e de Mealy, serão implementadas como se ilustra nas Figuras 16.43 e 16.44, respectivamente.

Agora, é fácil desenhar o logigrama do circuito. Por exemplo, a parte do logigrama que tem a ver com a transição do estado $E1$ para o estado $E2$ e consequente activação das funções de saída $C1$ e $S1$ em função das variáveis de entrada $D1$ e $D2$ deve ser feita como mostra a Figura 16.45.

Por exemplo, é de notar que a transição de $E1$ para $E2$ se faz desde que a variável $D2$ esteja activada, independentemente do nível de tensão em $D1$ (se $D2$ estiver desactivada, vai-se de $E1$ para $E1$).

Por outro lado, a função de saída $S1$ vem activada desde que se esteja no estado $E1$ e $D1$ esteja activada, ficando o semáforo 1 verde. De notar que, quando o circuito está noutro estado *que não o estado* $E1$, o flip-flop E1 tem a saída a L e a função $S1$ vem desactivada, o que faz com que o semáforo 1 fique vermelho.

Finalmente, a função de saída $C1$ vem activada se se estiver no estado $E1$ ou no estado $E7$ (e a cancela respectiva abre). Desde que o circuito esteja num estado *diferente* de $E1$ ou de $E7$, a função $C1$ vem desactivada e a cancela fecha.

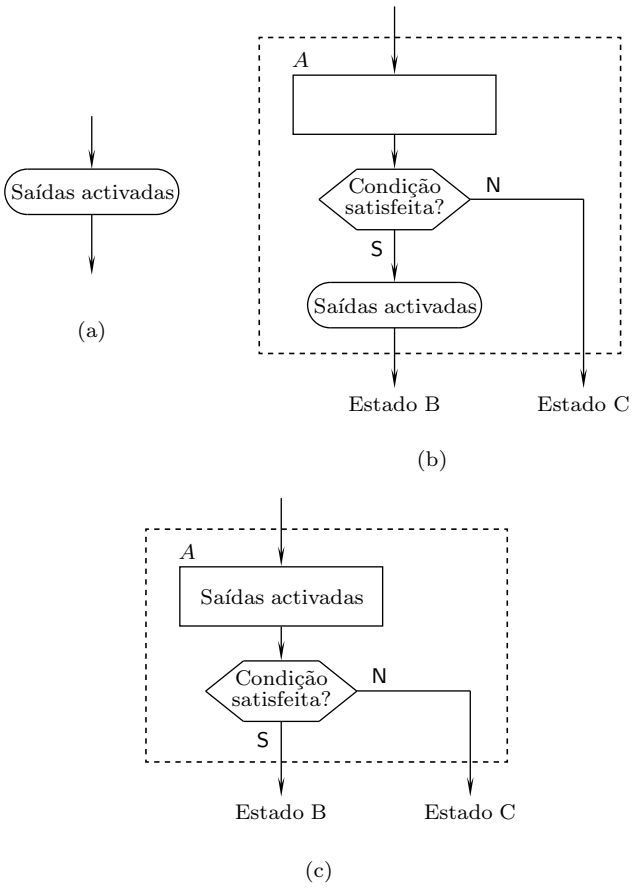


Figura 16.40: (a) Símbolo de uma saída de Mealy. Contraste entre uma saída de Mealy (b) e uma saída de Moore (c) num fluxograma. Enquanto a primeira identifica, no interior do seu símbolo, a saída ou saídas que vêm activadas num determinado estado, A por exemplo, quando uma determinada condição vem satisfeita na entrada ou entradas testadas, a segunda, porque não depende das entradas, vê a saída ou saídas activadas no símbolo do próprio estado

16.10 Referências Bibliográficas

Mano, Morris M., and Kime, Charles R. — *Logic and Computer Design Fundamentals*, 2nd ed., Prentice Hall International, Inc., New Jersey, USA, 2000, Secção 4.4.

16.11 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

16.1 Prove que os circuitos da Figura 16.1, na página 281, são sequenciais.

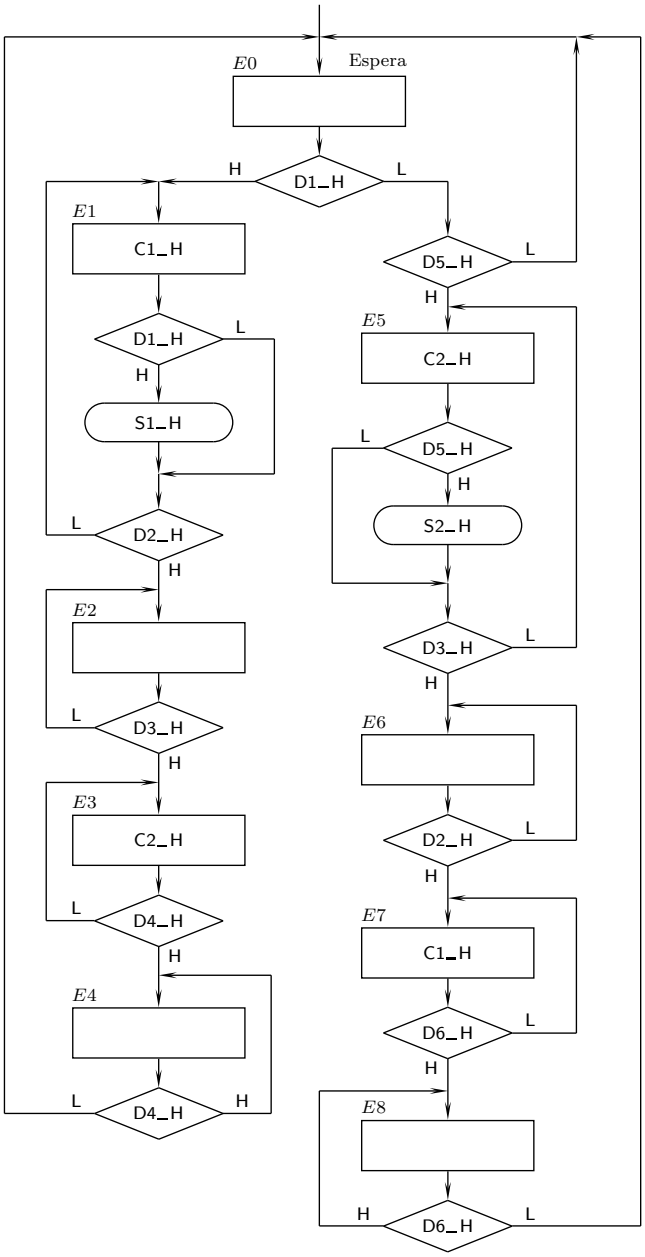


Figura 16.41: Fluxograma de Mealy de controlo das cancelas e dos semáforos da Figura 16.37

16.2 Considere o logigrama da Figura 16.46 e desenhe:

a) o diagrama de estados do circuito;

b) um diagrama temporal que ilustre o funcionamento da saída do circuito entre dois instantes, t_0 e t_1, sabendo que em t_0 se tem $Q0\_H = L$, $Q1\_H = = H$ e $Q2\_H = H$, e em t_1 se tem $Q0\_H = L$, $Q1\_H = L$ e $Q2\_H = H$.

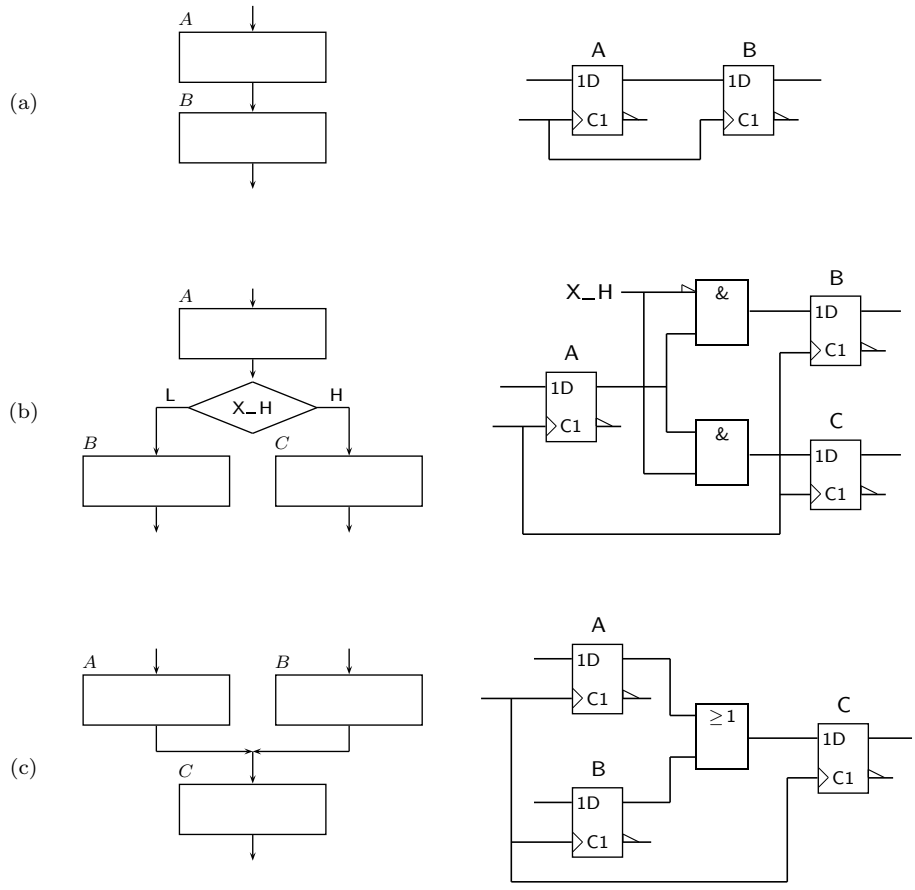


Figura 16.42: (a) Implementação, com um flip-flop por estado, de uma transição incondicional, (b) de uma transição condicionada ou "Fork", e (c) de uma convergência ou "Join" de um fluxograma

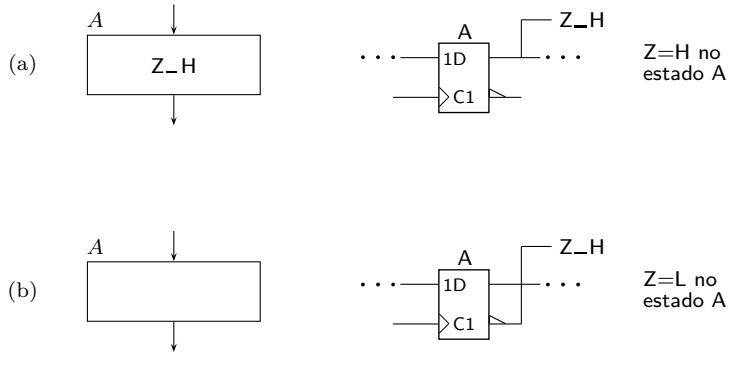


Figura 16.43: Implementação de uma saída de Moore de um fluxograma, com um flip-flop por estado

16.3 Analise o circuito sequencial síncrono da Figura 16.47, construindo a res-

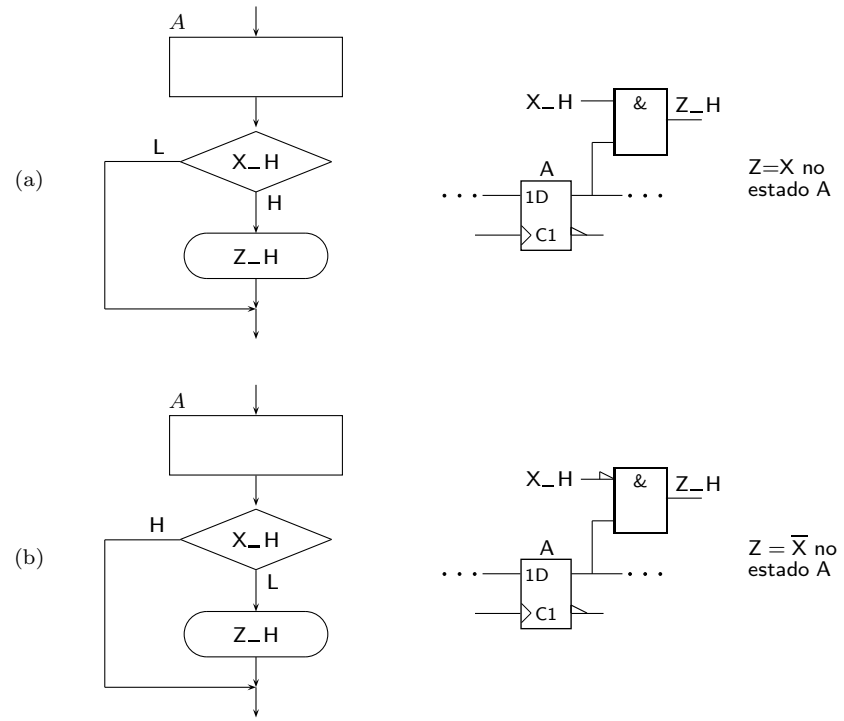


Figura 16.44: Implementação de uma saída de Mealy de um fluxograma, com um flip-flop por estado

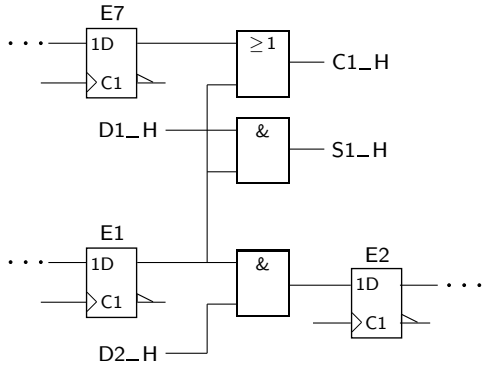


Figura 16.45: Implementação da transição do estado $E1$ para o estado $E2$ e das funções de saída $C1\_H$ e $S1\_H$ em função das variáveis de entrada $D1\_H$ e $D2\_H$ do circuito da Figura 16.37, usando um flip-flop por estado

pectiva tabela de estados e de saídas.

16.4 Analise o circuito sequencial síncrono da Figura 16.48. Para tanto,

a) desenhe o diagrama de estados do circuito;

b) desenhe o diagrama temporal da saída Z_H durante 5 impulsos de relógio, admitindo que os dois flip-flops se encontram inicialmente no es-

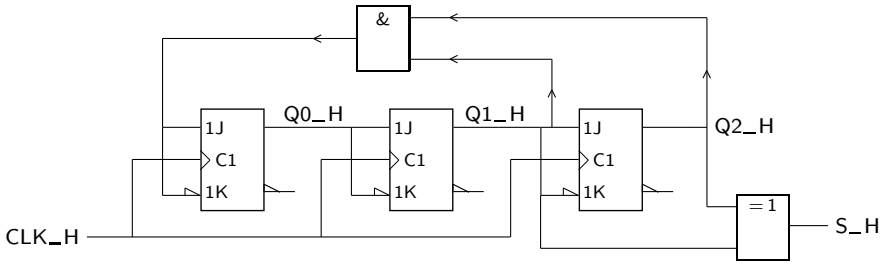


Figura 16.46: Logigrama do circuito sequencial síncrono do Exercício 16.2

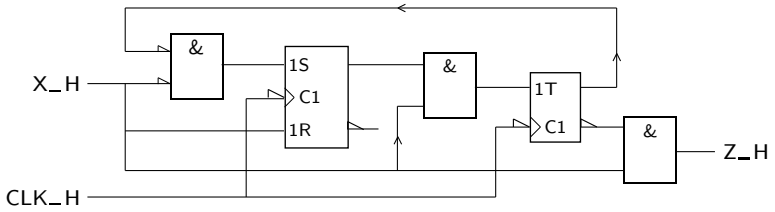


Figura 16.47: Circuito sequencial síncrono do Exercício 16.3

tado $(QA, QB) = (\mathrm{L}, \mathrm{L})$.

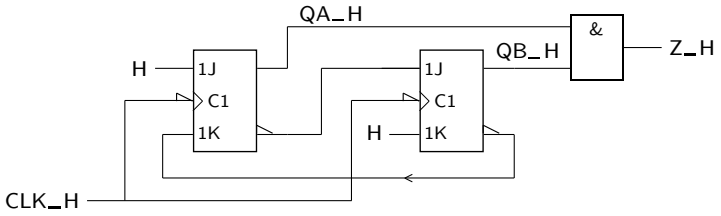


Figura 16.48: Circuito sequencial síncrono do Exercício 16.4

16.5 Considere o circuito da Figura 16.49.

a) Trata-se de um circuito concebido segundo o modelo de Moore ou de Mealy? Justifique brevemente.

b) Determine a sua tabela de estados.

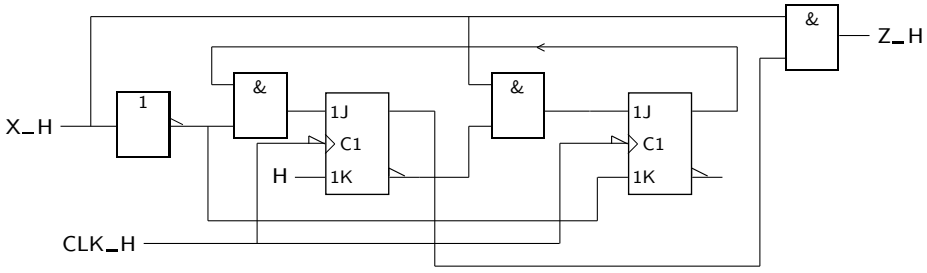


Figura 16.49: Circuito sequencial síncrono do Exercício 16.5

(∗) 16.6 Para uma determinada máquina síncrona com entrada X_H e saída Z_H obteve-se o comportamento temporal descrito na Figura 16.50. A máquina foi construída segundo o modelo de Moore ou de Mealy? Justifique.

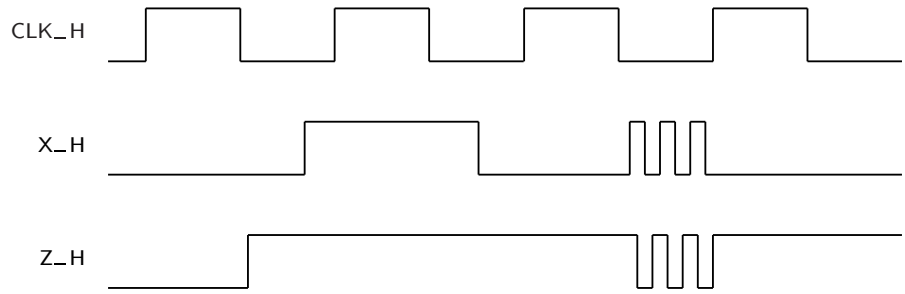


Figura 16.50: Comportamento temporal de uma máquina síncrona

16.7 Desenhe o diagrama de estados de uma máquina sequencial síncrona com duas entradas, X e Y, por onde surgem, em série, os bits de dois números binários sem sinal para serem comparados. Admita que os bits de menor peso dos números surgem em primeiro lugar. As saídas do circuito (quantas?) devem indicar, em cada momento, qual dos dois números é maior, ou se são iguais.

16.8 A saída S de um circuito sequencial é periódica, de período 4, e apresenta a sequência de três "1"s e um 0. É, portanto, do tipo

$$\underbrace{\ldots 1\ 0}\ \underbrace{1\ 1\ 1\ 0}\ \underbrace{1\ 1\ 1\ 0}\ \underbrace{1\ 1\ldots}$$

Desenhe um diagrama de estados para uma máquina de teste de Mealy, T, que analise, momento a momento, a saída S do circuito, e que dê saída 1 sempre que ocorrer uma alteração da sequência de saída em S. O circuito T deve poder ser ligado em qualquer instante, desconhecendo-se o estado de S nesse momento.

(∗) 16.9 Desenhe o diagrama de estados de uma máquina sequencial síncrona cuja função é gerar um bit de paridade para as palavras analizadas. As palavras têm comprimento 3, mas o circuito usa 4 impulsos de relógio para analizar cada palavra: os primeiros 3 impulsos são para os bits da palavra e o quarto impulso serve para a geração do bit de paridade. O bit de paridade deve vir igual a 1 se a paridade da palavra recebida for par, e a 0 se for ímpar. Enquanto o quarto impulso não ocorrer, o valor na saída é indiferente. Desenhe o circuito: (a) como uma máquina de Moore; e (b) como uma máquina de Mealy.

(∗) 16.10 Desenhe um diagrama de estados para uma máquina de Mealy com uma entrada série e uma saída que repete a sequência de entrada com dois períodos de relógio de desfasamento. Os dois primeiros valores na saída devem ser iguais a 0.

16.11 Desenhe o diagrama de estados de uma máquina sequencial síncrona que gera uma das seguintes sequências: 1 1 0 0 ou 0 1 0 1, conforme uma variável

de controlo está a 0 ou a 1, respectivamente. A variável de controlo pode mudar em qualquer momento e a máquina nunca deve, na transição, enviar mais do que dois bits iguais.

(∗) 16.12 Desenhe o diagrama de estados ou o fluxograma de uma máquina sequencial síncrona com uma entrada, X, e duas saídas, $Z0$ e $Z1$, com o comportamento que se descreve em seguida: $Z0 = 1$ apenas quando, na entrada, se verifica a sequência 1 1 0 1, e $Z0 = 0$ em todos os restantes casos; $Z1 = 1$ apenas quando, na entrada, se verifica a sequência 1 0 1 1, e $Z1 = 0$ em todos os restantes casos. A máquina deve detectar sequências com sobreposição.

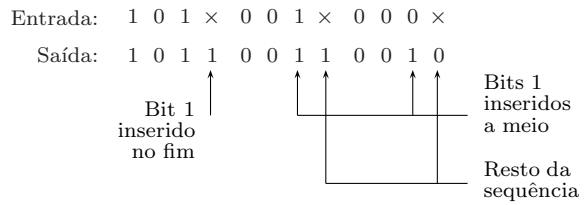
16.13 Desenhe o diagrama de estados de uma máquina sequencial síncrona de Moore com duas entradas, X e Y, e uma saída, Z, rais que: (i) se $Y = 0$, então $Z = X$; e (ii) se $Y = 1$, então Z é igual ao valor de X no período de relógio anterior. Admita que $Y = 0$ no instante inicial.

16.14 Desenhe o diagrama de estados de uma máquina sequencial síncrona com uma entrada e uma saída e com o seguinte comportamento.

(i) A máquina recebe pela entrada sequências de 4 bits, mas desses bits apenas considera os 3 primeiros. Como resposta, a máquina fornece uma sequência de 4 bits.

(ii) A sequência de saída é semelhante à sequência de entrada, mas inclui um 1 suplementar. Esse 1 segue-se imediatamente à primeira sequência de dois "0"s na entrada. Se tal sequência não ocorrer na entrada, o 1 será acrescentado no fim.

Segue-se um exemplo de comportamento da máquina.



16.15 Considere uma máquina sequencial síncrona com duas entradas, $X1$ e $X2$, e uma saída, Z. A saída deve permanecer a 0 até que as entradas sejam iguais uma à outra durante, pelo menos, 3 impulsos de relógio. Quando isso acontece, a saída deverá gerar a sequência 1 0 1 0.

Exemplo de funcionamento:

$X1$:	0	1	0	0	0	1	1	0	1	1	1
$X2$:	1	0	1	0	0	1	0	1	1	0	1
Z:	0	0	0	0	0	0	1	0	1	0	0

Desenhar o diagrama de estados da máquina.

16.16 Desenhe o diagrama de estados de uma máquina sequencial síncrona de Moore com uma entrada, X, e uma saída, Z, com o seguinte funcionamento: a saída só vai a 1 durante um período de relógio após a entrada ter ido a 1 e regressado a 0.

16.17 Desenhe o diagrama de estados de uma máquina sequencial síncrona com uma entrada e uma saída com o seguinte comportamento:

— a máquina possui dois modos de funcionamento;

— no Modo 1 a saída copia a entrada; no Modo 2 a saída é permanentemente 0;

— a máquina passa do Modo 1 para o Modo 2 quando recebe a sequência 1111 na entrada; e

— a máquina passa do Modo 2 para o Modo 1 quando recebe a sequência 0101 na entrada.

16.18 Desenhe o diagrama de estados de uma máquina sequencial síncrona com uma entrada e uma saída, normalmente a 0, que, ao identificar a sequência 101 na entrada, gera na saída a sequência 1010.

16.19 Desenhe o diagrama de estados ou o fluxograma de uma máquina sequencial síncrona com uma entrada e três saídas, com o seguinte comportamento.

a) A máquina recebe sequências sucessivas de 4 bits, que interpreta como dígitos BCD, com o bit mais significativo de cada dígito a entrar em primeiro lugar. A máquina apresentará em duas das saídas, e simultaneamente com o último bit que entra, o resultado da divisão inteira do dígito BCD por 3.

b) Se a sequência de entrada não for um dígito BCD, a terceira saída da máquina assumirá o valor 1, também em simultâneo com o último bit entrado.

c) Consideram-se as sequências de entrada sem sobreposição, isto é, como grupos sucessivos e independentes de 4 bits.

16.20 Construa o diagrama de estados de uma máquina sequencial síncrona com uma entrada e uma saída com o seguinte comportamento: a saída vem a 1 durante um impulso de relógio sempre que as últimas cinco entradas começarem por dois "1"s seguidos e tenham exactamente três "1"s. Por exemplo,

Entrada:	0	0	1	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1
Saída:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1

16.21 Desenhe o diagrama de estados (ou, se preferir, a tabela de estados ou o fluxograma) de uma máquina sequencial síncrona com uma entrada e uma saída com o seguinte funcionamento: nos dois primeiros impulsos de relógio a saída vem a 0; a partir daí, a saída vem a 1 se a entrada, em cada instante, é igual à entrada dois impulsos de relógio atrás e 0 nos restantes casos.

16.22 Desenhe o diagrama de estados de uma máquina sequencial síncrona com duas entradas, $X0$ e $X1$, e duas saídas, Y e Z. Às entradas são apresentados números binários com 2 bits, tendo $X1$ o bit mais significativo e $X0$ o menos significativo. A máquina analisa sequência sucessivas de três números. Se a sequência for $2, 1, 2$, a função Y vem a 1 e a máquina recomeça a análise de outra sequência. No caso de a sequência não ser a

prevista, Z vem a 1 e a máquina recomeça a análise de outra sequência. As saídas mantêm o valor 1 durante um impulso de relógio.

16.23 Desenhe o diagrama de estados de uma máquina sequencial síncrona que faça as funções de divisor de frequência por 2 ou por 3 consoante o valor lógico numa linha de controlo, M.

(∗) 16.24 Elabore um diagrama de estados para uma máquina de Mealy que recebe em série, na sua única entrada, uma palavra qualquer do código BCD (entra em primeiro lugar o bit de maior peso), e cuja saída só vem a 1 se a palavra que entrou for inferior a $4_{(\mathrm{BCD})}$ ou superior a $7_{(\mathrm{BCD})}$. O valor na saída não deve vir especificado para os três primeiros bits da palavra.

(∗) 16.25 Desenhe o diagrama de estados de uma máquina de Mealy que detecte palavras do código BCD. As palavras, com 4 bits, entram em série por uma entrada única, começando pelo bit de maior peso. Ao fim de quatro impulsos de relógio o circuito deve vir reiniciado, preparado para detectar uma nova palavra. Para além da entrada, o circuito tem duas saídas, X e Y, tais que: (i) se $X = Y = 0$, então a palavra recebida não pertence ao código BCD; (ii) se $X = Y = 1$ a palavra pertence ao código; e (iii) se $X = 1$ e $Y = 0$ é porque não foi possível, até ao momento, saber se a palavra pertence ou não ao código.

16.26 Desenhe o diagrama de estados de uma máquina sequencial síncrona com uma entrada e uma saída, com esta última a assumir o valor 1 se os últimos 4 bits forem um número par em BCD. Se o número for maior do que $9_{(10)}$, a saída deverá vir a 0. O bit que entra primeiro é o bit mais significativo. Por outro lado, apenas interessa o valor da saída quando é recebido o quarto bit.

(∗) 16.27 Obtenha o diagrama de estados (ou o fluxograma) de uma máquina sequencial síncrona com duas entradas e duas saídas que compara dois dígitos BCD, A e B. Cada dígito é apresentado em série por uma das entradas, começando pelo bit de menor peso. Nas entradas são presentes sequências sucessivas de 4 bits. A saída deve indicar permanentemente se $A > B$, se $A < B$ ou se $A = B$.

16.28 Desenhe o diagrama de estados de uma máquina sequencial síncrona com duas entradas e duas saídas. A máquina recebe, pelas suas entradas, sequências sucessivas de 4 bits. No fim de cada sequência a máquina deve indicar se as duas sequências que entraram são ou não dígitos BCD. Se forem, a máquina deve ainda indicar se eles são ou não iguais.

(∗) 16.29 Determinar um diagrama ou uma tabela de estados de uma máquina sequencial síncrona que recebe dígitos BCD a começar pelo bit menos significativo. A máquina dará saída 1 se o dígito for múltiplo de 4.

16.30 Desenhe o logigrama de uma máquina sequencial síncrona com uma entrada X que condiciona o comportamento do circuito da seguinte forma: se $X = 0$, a máquina possui o diagrama de estados da Figura 16.51(a); se $X = 1$, o diagrama de estados é o da Figura 16.51(b).

Indique a evolução do circuito se, antes de receber o primeiro impulso de relógio após ligar o sistema, o circuito estiver no estado B e a variável de entrada estiver em 1.

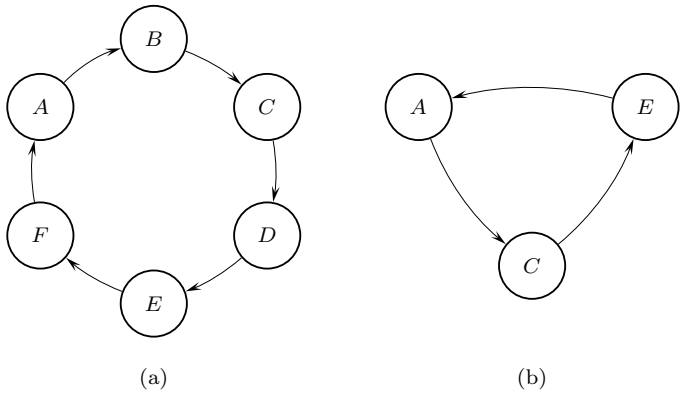


Figura 16.51: (a) Diagrama de estados da máquina síncrona do Exercício 16.30 no caso em que $X = 0$; e (b) no caso em que $X = 1$

16.31 a) Projecte um circuito sequencial síncrono de Mealy que soma a constante 2 em binário a um número com 3 bits entrado em série, ficando preparado para repetir a operação sucessivamente com outros números na entrada. Caso haja transporte no fim, despreze-o. Entram primeiro os bits de menor peso do número.

b) Transforme o diagrama de estados obtido na alínea anterior no pressuposto de o circuito representar uma máquina de Moore.

16.32 Considerando o diagrama de estados da Figura 16.52, e sabendo que dispõe de om oscilador de relógio CLK com uma frequência de 1 Hz,

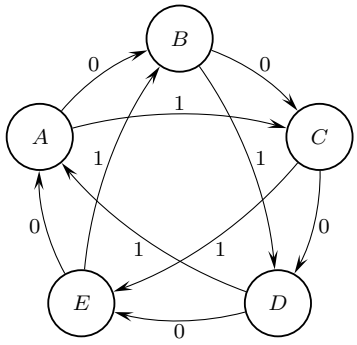


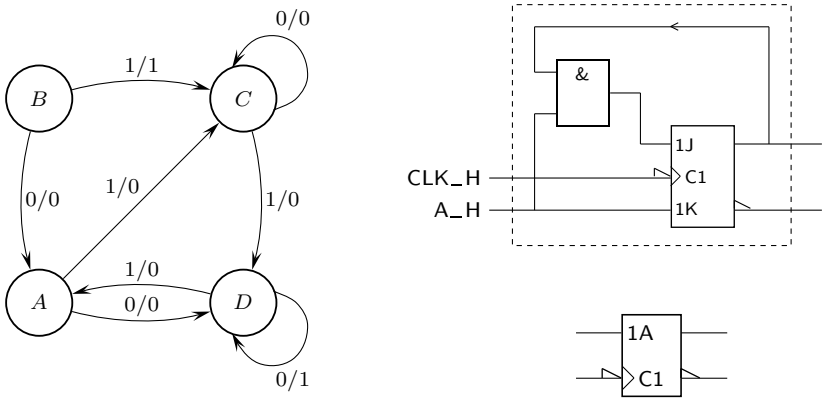
Figura 16.52: Diagrama de estados da máquina síncrona do Exercício 16.32

a) sintetize o respectivo circuito sequencial síncrono usando flip-flops D edge-triggered que comutem nos flancos descendentes de CLK;

b) utilizando o circuito sintetizado sem qualquer alteração, diga como poderia obter uma onda de relógio CLK' sincronizada com CLK mas de período $T = 3\,\mathrm{s}$.

16.33 Partindo do diagrama de estados apresentado na Figura 16.53(a), desenhe o logigrama do respectivo circuito sequencial síncrono, utilizando como

elementos de memória os flip-flops hipotéticos do tipo A representados na Figura 16.53(b).



(a) (b)

Figura 16.53: (a) Diagrama de estados da máquina síncrona do Exercício 16.33; (b) flip-flop hipotético do tipo A, a usar na implementação da máquina

16.34 Implemente, utilizando flip-flops JK e NANDs em lógica positiva, o circuito sequencial síncrono representativo da máquina com a tabela de estados e de saídas da Tabela 16.10. Trata-se de um circuito de Mealy ou de Moore?

Tabela 16.10: Tabela de estados e de saídas da máquina sequencial síncrona do Exercício 16.34

EA	ES		Z
	X = 0	X = 1	
A	A	B	0
B	C	D	0
C	A	C	1
D	B	D	1

16.35 Projecte um contador que conte segundo o código $\cdots, 0, 3, 7, 2, 5, 0, \cdots$. Utilize os flip-flops que entender.

16.36 Projecte um circuito sequencial síncrono descrito pela tabela de estados e de saídas da Tabela 16.11. Trata-se de um circuito de Mealy ou de Moore?

16.37 Projecte o circuito sequencial síncrono especificado pelo diagrama de estados da Figura 16.54. Use os flip-flops que desejar. Não é necessário desenhar o logigrama.

16.38 Utilizando flip-flops JK, realize o projecto da máquina sequencial síncrona com a tabela de estados e de saídas da Tabela 16.12.

Tabela 16.11: Tabela de estados e de saídas do circuito sequencial síncrono do Exercício 16.35

EA	ES/Z	
	X = 0	X = 1
A	$B/0$	$D/0$
B	$A/0$	$G/0$
C	$B/0$	$G/0$
D	$E/0$	$G/1$
E	$B/1$	$A/1$
F	$G/0$	$B/0$
G	$E/0$	$D/1$

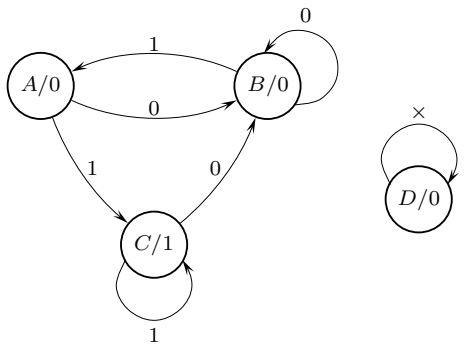


Figura 16.54: Diagrama de estados do circuito sequencial síncrono do Exercício 16.37

Tabela 16.12: Tabela de estados e de saídas da máquina sequencial síncrona do Exercício 16.38

EA	ES/Z	
	X = 0	X = 1
A	$A/0$	$B/0$
B	$C/1$	$D/0$
C	$E/0$	$A/1$
D	$B/1$	$E/1$
E	$D/0$	$A/0$

16.39 Um "watchdog" é um circuito que vigia o bom funcionamento de outro circuito. Para tal, o circuito vigiado deve enviar periodicamente impulsos ao "watchdog". Se o circuito vigiado não enviar um ou mais desses impulsos periódicos, o "watchdog" activa uma saída de alarme. Desenhe o diagrama de estados de um "watchdog" síncrono em que o impulso que *"Watchdog"*

vem do circuito vigiado surge com um atraso máximo de 6 impulsos de relógio, dura 1 período de relógio, e é síncrono com este.

Máquina incompletamente especificada (incompleta)

16.40 Desenhe o logigrama da **máquina sequencial síncrona incompletamente especificada** (ou, mais simplesmente, **máquina incompleta**) descrita pela tabela de estados e de saídas da Tabela 16.13.

Tabela 16.13: Tabela de estados e de saídas da máquina sequencial síncrona incompleta do Exercício 16.40

EA	ES/Z			
	(X2,X1)			
	(0,0)	(0,1)	(1,0)	(1,1)
A	$A/0$	$B/1$	$C/0$	$-/-$
B	$A/1$	$C/1$	$-/-$	$D/0$
C	$C/1$	$A/0$	$B/0$	$-/-$
D	$B/0$	$D/0$	$-/-$	$A/-$

16.41 Projecte, utilizando flip-flops do tipo T, um contador binário Up/Down síncrono de módulo 3.

16.42 Utilizando flip-flops do tipo D, projecte um contador binário de módulo 8, com a possibilidade de ser inicializado sincronamente a 6.

16.43 Projecte um contador binário de módulo 3, usando um flip-flop D para o bit menos significativo e um JK para o mais significativo.

16.44 Considere a entrada de um parque de estacionamento. Para um carro ser admitido no parque, tem de introduzir uma moeda na ranhura de uma caixa, moeda essa que activa um sensor M durante um impulso de relógio. Após essa ocorrência, uma cancela levanta e o carro pode prosseguir com o estacionamento. Durante todo o tempo em que o carro está a entrar, um detector D fica activo. A cancela fecha após o carro ter saido da frente do detector. Imediatamente depois do detector, existe uma banda de borracha que deve ser pisada duas vezes, uma por cada eixo de um carro. No caso de ser pisada três vezes, isso significa que uma camioneta está a tentar entrar (o que é proibido), ou que há fraude. Nesse caso a cancela deverá fechar, caindo em cima do infractor (como solução, não é lá muito aconselhável, mas enfim ...).

Desenhe um fluxograma de uma máquina sequencial síncrona que implemente o controlo da cancela.

16.45 Utilizando flip-flops JK, projecte um circuito sequencial síncrono descrito pelo fluxograma da Figura 16.55.

16.46 Repetir o Exercício 16.9, mas admitindo agora que na saída se reproduzem os 3 bits da palavra de entrada, nos 3 ciclos de relógio que antecedem a geração do bit de paridade.

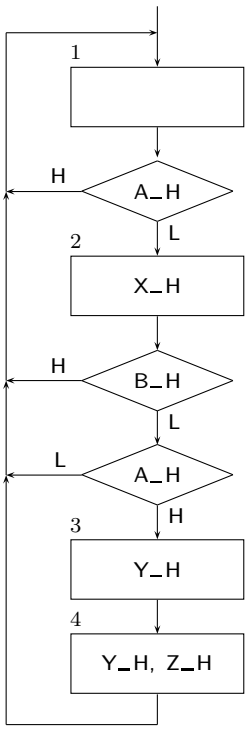


Figura 16.55: Fluxograma do circuito sequencial síncrono do Exercício 16.45

(∗) 16.47 Utilize um contador integrado do tipo 74LS161A para implementar a máquina sequencial síncrona representada pelo diagrama de estados da Figura 16.56.

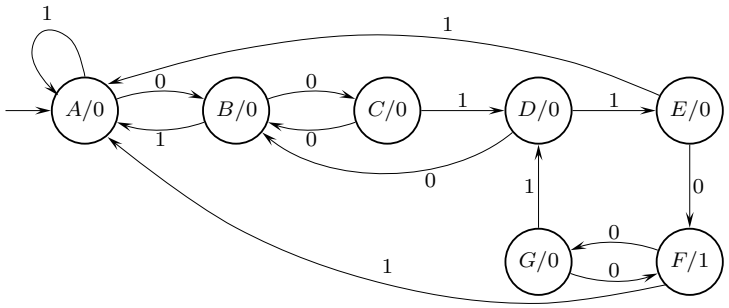


Figura 16.56: Diagrama de estados da máquina sequencial síncrona do Exercício 16.47

Capítulo 17

Memórias

Os modernos computadores — sejam eles sistemas embebidos autónomos, computadores de uso geral ou servidores — são constituídos basicamente por 5 unidades: a **Unidade Central de Processamento** ou **CPU**, a **Unidade de Memória**, a **Unidade de Controlo** e as **Unidades de Entrada e de Saída**. Destas 5 unidades vamos, neste capítulo, estar interessados apenas na Unidade de Memória, constituída por um conjunto de circuitos integrados de memória.

Unidade Central de Processamento (CPU)

Unidade de Memória

Unidade de Controlo

Unidades de Entrada e de Saída

Para além da Unidade de Memória, interna aos computadores existem ainda memórias (discos, diskettes, fitas magnéticas, CDs, DVDs, etc.) que fazem parte da memória externa dos computadores e que comunicam com a CPU através das Unidades de Entrada e de Saída. Nestas memórias externas a informação é gravada em suportes magnéticos ou ópticos. Não iremos aqui considerar estes tipos de memória.

As memórias que nos interessam arrumam-se em dois grandes grupos: o das **RAMs** (**"Random Access Memories"** ou **memórias de leitura/escrita**), e o das **ROMs**, (**"Read Only Memories"** ou **memórias de leitura**).

"Random Access Memory" (RAM)

"Read Only Memory" (ROM)

As RAMs são memórias onde é possível efectuar **operações de escrita** de dados e subsequentes **operações de leitura** dos dados entretanto escritos. As operações de escrita e de leitura decorrem na sequência normal de funcionamento dos sistemas em que as RAMs se inserem.

Operações de escrita e de leitura

Pelo contrário, as ROMs são memórias que podem ser programadas (numa operação de escrita) uma única vez, em alguns tipos de ROM mais do que uma vez (mas sempre poucas vezes). Ou seja, a operação corrente nas ROMs é de leitura, durante o funcionamento normal dos sistemas em que estas memórias se inserem, pelo que a operação de escrita é única ou muito pouco frequente.

A designação "Random Access Memory" dada às RAMs é, hoje pouco clara, mas tem a ver, na origem, com o facto de o tempo de acesso à informação nesse tipo de memória ser igual para todas as palavras, independentemente do endereço aleatório que se lhe aplicasse. Antes do aparecimento deste tipo de dispositivo existiam, basicamente, **memórias série**, semelhantes a registos de deslocamento. Nesses dispositivos o tempo de acesso à informação dependia da distância a que ela se encontrava da saída, quando se pretendia aceder-lhe. Para contrapor a este tipo de memórias, as memórias RAM passaram a ter "acesso aleatório".

Memórias série

17.1 "Read Only Memories" (ROMs)

17.1.1 Tipos de ROMs

A designação "Read Only Memory" ou ROM aplica-se, na realidade, a um conjunto diversificado de memórias, em que se incluem vários tipos, como mostra a Figura 17.1.

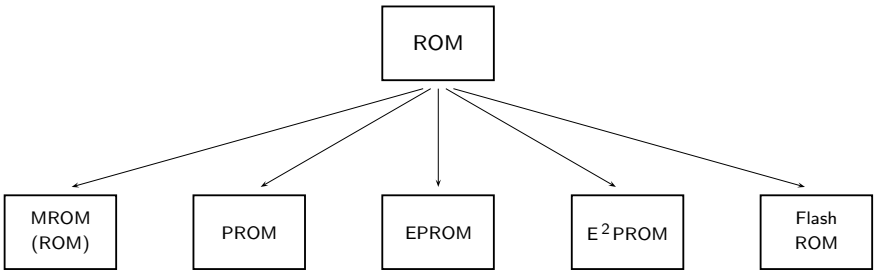


Figura 17.1: Principais tipos de ROMs

À medida que caminhamos da esquerda para a direita na figura obtemos memórias que são progressivamente mais flexíveis em matéria de programação (escrita), aproximando-se cada vez mais das características das RAMs ou memórias de leitura/escrita. Por outro lado, obtemos memórias com um custo por bit em geral mais elevado.

"Mask-Programmed ROM" (MROM)

Assim, as **MROMs** ou **Mask-programmed ROMs** (por vezes também designadas, incorrectamente, simplesmente por ROMs) são circuitos integrados projectados de origem pelo fabricante com o conteúdo pretendido pelo utilizador. Não podem ser programadas pelo utilizador, e o seu conteúdo não é alterável.

"Programmable ROM" (PROM)

Programador de PROM

As **PROMs** (ou **"Programmable ROMs"**) são variantes de ROMs que podem ser programadas uma única vez pelo utilizador, por fusão de pequenos fusíveis. São compradas com todos os fusíveis intactos, e o utilizador recorre a um **programador de PROMs** para fundir alguns dos fusíveis e deixar outros intactos, ficando assim a memória programada com "1"s e com "0"s. No caso de ser necessário alterar o conteúdo da PROM, há que programar um novo dispositivo e o anterior fica inutilizado.

"Erasable PROM" (EPROM)

As **EPROMs** (ou **"Erasable PROMs"**) são memórias que podem ser programadas diversas vezes pelo utilizador (com recurso ainda a um programador de PROMs), e que podem ser apagadas submetendo-as a radiação ultra-violeta durante um intervalo de tempo relativamente prolongado. As EPROMs podem ser regravadas um certo número de vezes (dezenas a milhares).

"Electrically Erasable PROM" (EEPROM ou E^2PROM)

As **EEPROMs** (**"Electrically Erasable PROMs"**, ou ainda **E^2PROMs**) são EPROMs que podem ser apagadas e reprogramadas electricamente, uma palavra de cada vez (em geral, uma palavra tem a dimensão de um byte), sem necessidade de as retirar do circuito em que estão inseridas. Do mesmo modo, são alteráveis um número limitado de vezes.

Memórias Flash

As **memórias Flash** ou **Flash ROMs** são variantes de EEPROMs que podem ser apagadas ou reprogramadas electricamente, em blocos de palavras (um bloco de

cada vez). Tal como acontece com as memórias anteriores, também as memórias Flash são alteráveis um número limitado de vezes. São as memórias ROM que mais se aproximam das memórias RAM sem, contudo, possuirem a mesma facilidade de leitura e de escrita que são características das RAMs.

17.1.2 Utilização das ROMs

As ROMs podem servir para varias aplicações:

— suporte a programas em sistemas embebidos, programas esses que não irão ser alterados (uso de ROMs ou de PROMs) ou que podem ser actualizados um número pequeno de vezes (uso de EPROMS, de EEPROMS ou de memórias Flash);

— memorização de tabelas — como veremos, por exemplo, no Capítulo 19, quando estudarmos as máquinas de estados, ou ainda em aplicações em que se pretende fazer uma conversão de determinados dados (por exemplo, conversões de códigos) ou em que se querem gerar certas funções analógicas (formas de onda sinusoidais, triangulares, em dentes de serra, etc.);

— implementação de lógica combinatória (como veremos no Capítulo 18).

17.1.3 Estrutura de uma ROM

Uma **ROM** pode ser modelada por uma matriz com 2^n linhas (**palavras**) e p colunas (p **bits de dados** por palavra), conjuntamente com um descodificador binário de n bits e saídas activas a H (Figura 17.2).

Palavra (de uma ROM)

Bits de dados

De forma simplificada, diz-se que estamos em presença de uma **ROM de 2^n por p**, ou de uma **ROM de $2^n \times p$**, porque possui $2^n \times p$ **células**, uma por cada intersecção linha–coluna.

ROM de 2^n por p ($2^n \times p$)

Células

Recorrendo a uma máscara fotográfica, o fabricante da ROM insere um nível H ou um nível L (respectivamente, um 1 ou um 0 em lógica positiva) em cada uma das células, no processo de fabrico da ROM.

As n linhas de entrada do descodificador designam-se colectivamente por **barramento de endereços** da ROM, às quais se aplicam **endereços** (quantidades booleanas gerais).

Barramento de endereços (de uma ROM)

Endereço

Para cada endereço aplicado às entradas vai ser activada uma e só uma saída do descodificador, isto é, uma e só uma palavra da ROM. Apenas a palavra que foi endereçada pode debitar para as p saídas do dispositivo (o **barramento de dados**) o que nessa palavra tiver sido programado pelo fabricante.

Barramento de dados (de uma ROM)

Dizemos, então, que se leu para as saídas o conteúdo da palavra da ROM com esse endereço, ou que foi feita a **leitura de uma palavra da ROM**, ou ainda, e mais simplesmente, que foi feita uma **leitura da ROM**.

Leitura (de uma palavra) da ROM

Como à aplicação de um endereço vai corresponder uma operação de leitura da palavra correspondente, com o consequente envio para o exterior dos bits programados nessa palavra (naturalmente, após um certo tempo de propagação), segue-se que o comportamento da ROM é combinatório.

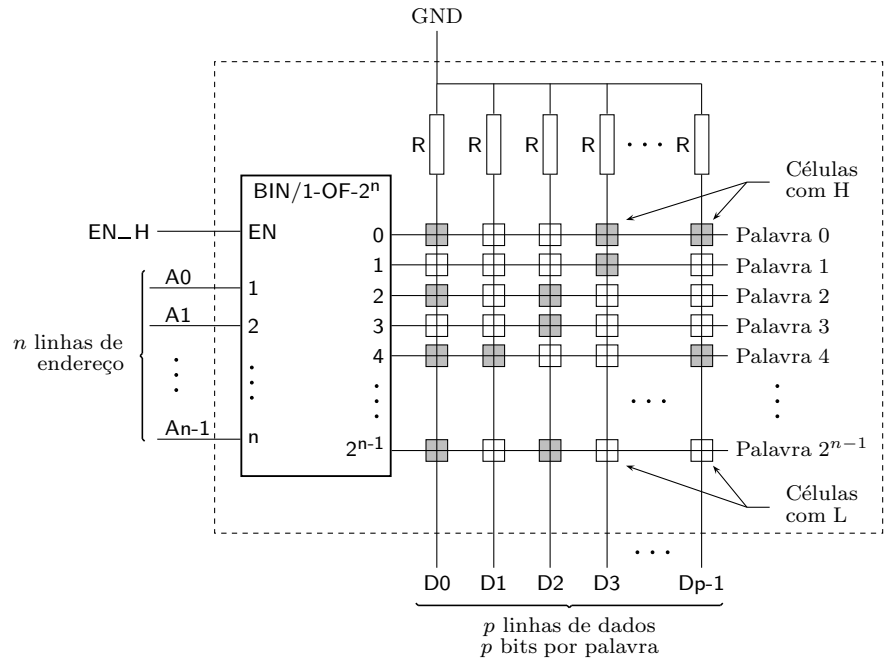


Figura 17.2: Modelo matricial de uma ROM com n linhas de endereço e p linhas de dados, ou ROM de $2^n \times p$. Esta ROM possui $2^n \times p$ células, umas programadas com níveis H (a cinzento), outras a L (a branco)

Assim sendo, não é de estranhar que se possam utilizar ROMs na implementação de funções booleanas gerais (uma função booleana simples por saída) das variáveis booleanas gerais aplicadas às entradas de endereço, como faremos no capítulo seguinte.

17.1.4 Funcionamento de uma MROM

Vamos agora ver como é constituída cada uma das células de uma MROM (Figura 17.3).

Como podemos observar na figura, uma célula é formada por um transistor nMOS (se a MROM for unipolar, como é o caso; no caso de uma MROM bipolar teremos um transistor bipolar por célula) e por uma ligação que, ou é deixada, ou é retirada no processo de geração da máscara da MROM (no caso de uma PROM, a ligação é substituída por um fusível, e a ausência de ligação significa que o programador da PROM fundiu o fusível). O transistor está ligado à tensão de alimentação, V_{dd}, e à linha correspondente.

Admitamos, então que, na célula que se encontra na intersecção da linha i com a coluna j, o fabricante da MROM deixou ficar a ligação [Figura 17.3(a)]. Quando a palavra i for seleccionada pelo endereço aplicado às entradas, o nível H gerado na saída i do descodificador binário torna o transistor condutor, pelo que a tensão V_{dd} (aproximadamente) aparece na coluna j. Ou seja, obtemos um nível H nessa coluna.

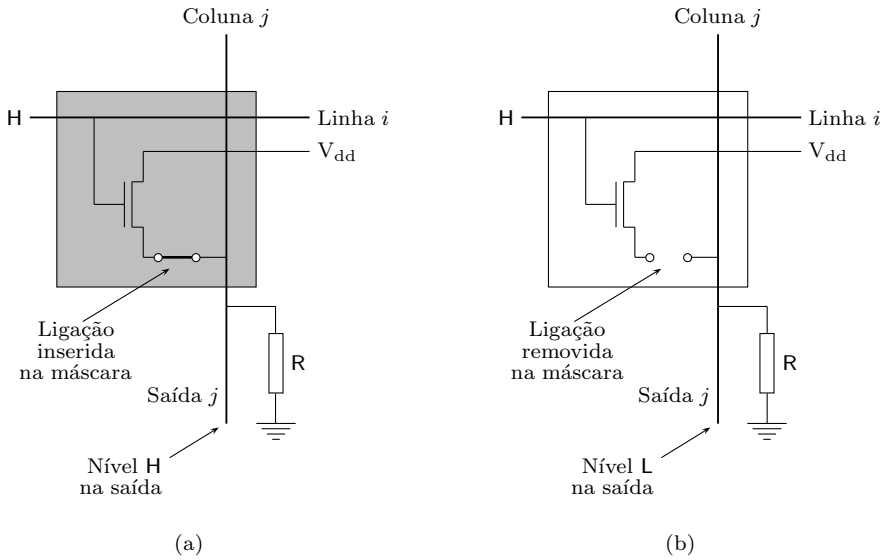


Figura 17.3: Estrutura de cada uma das células de uma MROM em tecnologia nMOS, admitindo que a palavra (linha) i está seleccionada. (a) Caso em que a ligação é deixada na máscara da MROM, obtendo-se um H na coluna j. (b) Caso em que a ligação é removida na máscara da MROM, obtendo-se um L na coluna j

Pelo contrário, admitamos agora que na intersecção da linha i com a coluna j o fabricante da MROM removeu a ligação [Figura 17.3(b)]. Nessas condições, quando a palavra i for seleccionada o transistor conduz, porém a ausência de ligação significa que a tensão V_{dd} não pode passar para a coluna j. Então, o que aparece na coluna j é o nível L proveniente da ligação da coluna à massa, GND, através da correspondente resistência de "pull-down", R. Ou seja, obtemos um nível L (praticamente 0 V) na coluna j.

Se a linha i não tiver sido seleccionada pelo endereço presente às entradas, a saída correspondente do descodificador está a L e *nenhum dos transistores nessa linha conduz*, pelo que as correspondentes contribuições se traduzem por níveis L em todas as colunas.

Consideremos agora uma coluna j qualquer. Duas situações podem ocorrer.

1. Na saída j apenas temos contribuições de níveis L dos diversos transistores ligados à coluna com o mesmo número: (i) porque existe um nível L proveniente do transistor na palavra i que foi endereçada, estando o transistor desligado da linha; e (ii) porque existem níveis L provenientes dos restantes transistores nessa coluna, pertencentes a palavras não endereçadas (relembremos que apenas uma palavra vem endereçada, ou seleccionada, de cada vez). Neste caso a saída j vem a L, nível esse que resulta da leitura do bit a L armazenado na célula que se encontra na intersecção da coluna j com a linha i que foi endereçada.

2. Na saída j temos contribuições de níveis L e um (e só um) nível H, proveniente do transistor condutor na linha i que estiver, nesse momento, a ser endereçada. Neste caso a saída j vem a H, sendo este nível resultante da leitura do bit a H armazenado na célula que está na intersecção da coluna j com a linha i que foi

endereçada.

Em resumo, uma ligação deixada na máscara, na intersecção de uma linha com uma coluna, significa um nível H na coluna e na saída, enquanto que uma ligação removida significa um nível L.

Finalmente, se a entrada de Enable da MROM (*vd.* a Figura 17.2) estiver inactiva, então todas as saídas do descodificador estão inactivas (a L) e *todos os transistores da MROM estão em não condução*, pelo que as saídas vêm todas a L. Podemos, então, afirmar que a ROM da Figura 17.2 tem saídas activas a H, que vêm desactivadas se fizermos o Disable do dispositivo.

17.1.5 Descodificação coincidente

O grande inconveniente da estrutura das ROMs na Figura 17.2 é a dimensão do descodificador utilizado na selecção de uma linha (palavra).

Por exemplo, se pensarmos numa ROM de 256×4, esse descodificador possuirá 256 saídas, uma por cada palavra, e necessitará de 256 ANDs de 8 entradas (mais uma entrada para o Enable). O descodificador vai, nessas circunstâncias, ocupar um espaço considerável no circuito integrado, porque necessita de muitas portas e porque cada uma delas possui um elevado número de entradas.

Por essa razão, os fabricantes de ROMs utilizam um outro tipo de descodificação que recorre a dois descodificadores mais pequenos, numa estrutura designada por *descodificação coincidente*. Desta forma ganham em área ocupada no circuito integrado, o que lhes permite incluir ROMs de maiores dimensões para a mesma área.

Por exemplo, a ROM anterior, de 256×4, em vez de vir organizada como uma matriz de 256 linhas por 4 colunas, vem, na prática, organizada com uma matriz de 32×32, sendo cada linha constituída por 8 palavras de 4 bits, como mostra a Figura 17.4.

Descodificador de linha

Um **descodificador de linha** de 5 bits selecciona, de cada vez, uma das 32 linhas (8 palavras) da matriz — à custa das linhas de endereço de menor peso, $A0$ a $A4$.

Descodificador de coluna

Por seu turno, um **descodificador de coluna** de 3 bits utiliza as outras 3 linhas de endereço — as de maior peso, $A5$ a $A7$ — para seleccionar uma de entre as 8 palavras de 4 bits que estão disponíveis na linha da matriz seleccionada pelo descodificador de linha. Os 4 bits dessa palavra são lidos, então, para o exterior da matriz e, daí, para as 4 linhas de saída, se as entradas de controlo permitirem.

Apesar de esta solução necessitar de dois descodificadores, eles são consideravelmente mais pequenos do que o descodificador único: o descodificador de linha necessita de 32 ANDs de 5 entradas, e cada um dos 4 descodificadores de coluna requere 8 ANDs de 3 entradas.

17.1.6 Símbolos das ROMs

Os símbolos das ROMs são relativamente simples. Ilustremos com o símbolo IEC simplificado de uma ROM típica, de 256×4 como a anterior, na Figura 17.5.

*Qualificador geral ROM **

O **qualificador geral ROM *** identifica uma ROM, com o asterisco substituído pela

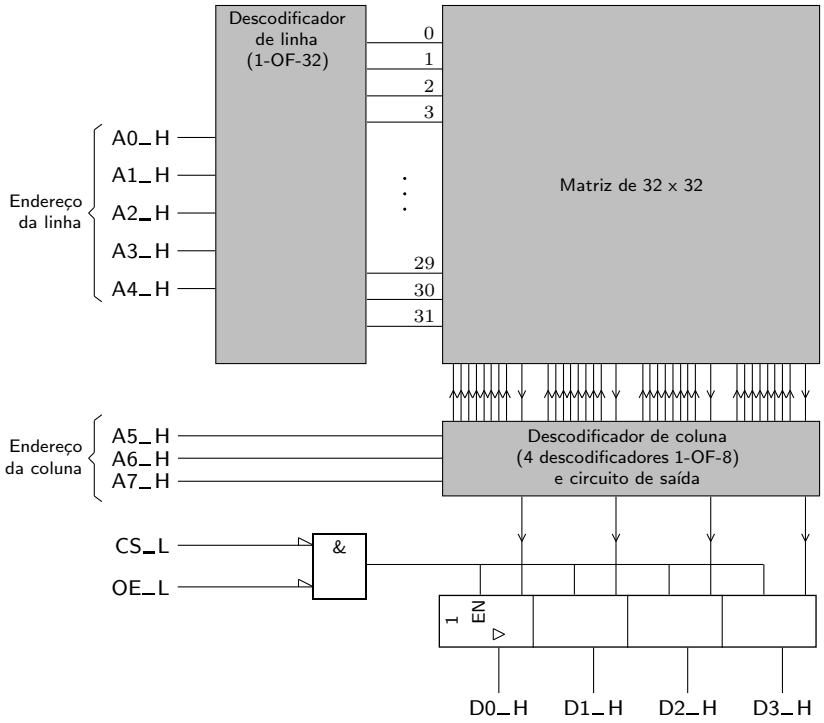


Figura 17.4: Uma ROM de 256×4 vem, na prática, organizada como uma matriz de 32×32, com um descodificador de coluna e um descodificador de linha, para além do circuito de saída controlado por duas linhas, $CS_$L e $OE_$L

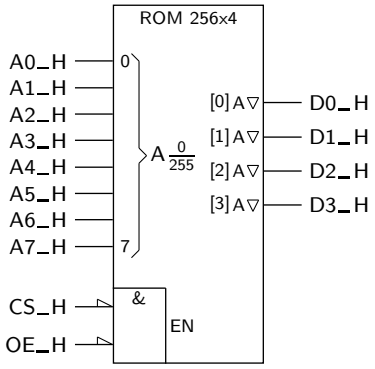


Figura 17.5: Símbolo IEC simplificado de uma ROM típica de 256×4, com saídas "tri-state" e duas entradas de controlo activas a L

correspondente dimensão.

As 8 linhas de endereço, $A0_$H a $A7_$H, permitem endereçar (seleccionar) uma das 256 palavras para ser lida, desde que estejam activadas as entradas de controlo — uma delas designada por Chip Select ($CS_$L) ou, em alternativa, por Chip Enable ($CE_$L), e a outra designada por Output Enable ($OE_$L). Decorrido um certo tempo de propagação, nas saídas $D0_$H a $D3_$H aparece

o conteúdo dessa palavra (a questão dos tempos de propagação será examinada na Subsecção 17.1.7).

Qualificadores de entrada Am

No símbolo, a chaveta indica que as linhas de endereço devem ser tomados em conjunto, com os qualificadores de entrada A0 a A255 a configurar uma **dependência de Endereço** (A significa Address ou endereço).

Dependência de Endereço (A)

Qualificadores de saída A

Esta dependência vem manifestada pelos qualificadores A0 a A255, que afectam as saídas com o **qualificador de saída** A. A palavra seleccionada numa operação de leitura da ROM, por aplicação de certos níveis de tensão às entradas de endereço, será aquela que tem o endereço decimal que resulta da soma das potências de 2 das entradas activas (desde que as entradas de controlo permitam).

Ora como vimos atrás, esta ROM possui duas entradas de controlo que permitem a leitura de uma palavra. Quando uma delas, ou as duas, vêm desactivadas, as saídas da ROM ficam todas em alta impedância. O facto de as saídas serem do tipo "tri-state" permite ligar múltiplas ROMs, ou ainda ROMs e outros dispositivos com saídas "tri-state" (RAMs, microprocessadores, etc.), a barramentos comuns de transporte de dados entre os diversos dispositivos.

Qualificadores de saída [i]

Em cada saída aparece ainda um **qualificador de saída** [i], opcional, que identifica o peso dessa saída relativamente às outras.

17.1.7 Temporizações na leitura de uma ROM

Antes de passarmos ao estudo das temporizações que podemos observar na leitura de uma palavra de ROM, vamos considerar com algum cuidado os significados das variáveis de controlo CS e OE que encontrámos nas Figuras 17.4 e 17.5.

Chip Select, CS

A entrada de **Chip Select** permite seleccionar uma ROM em particular de entre um conjunto de memórias (ROMs ou RAMs), em esquemas de memória do género dos que estudaremos na Subsecção 17.1.8 em que co-existem, no mesmo sistema, diversos circuitos de memória.

Descodificador de endereços

Espaço de endereçamento

Zona de memória

Nesses casos, a activação ou inactivação de CS será controlada por um **descodificador de endereços** externo, cuja função é seleccionar um circuito de memória de cada vez, consoante o endereço pretendido. Existirá, assim, um **espaço de endereçamento** global, com cada circuito de memória afectado a uma certa **zona de memória**, que é subconjunto do espaço de endereçamento.

Se o CS_L da ROM estiver activa, é porque queremos aceder a uma palavra com um endereço afectado à zona de memória ocupada pela memória. Nesse caso queremos que a palavra lida da ROM seja colocada no barramento de dados comum aos diversos circuitos de memória, a fim de poder ser lida por outro circuito qualquer.

Pelo contrário, se CS_L estiver inactiva queremos retirar a ROM do barramento de dados, porque queremos endereçar outro circuito de memória que não esta ROM em particular. Isto é, queremos, neste caso, aceder a uma palavra contida noutro circuito de memória, ou seja, a uma palavra com um endereço pertencente à zona de memória que foi afectada a esse outro circuito.

Output Enable, OE

Quanto à entrada de **Output Enable**, OE_L, permite colocar a saída em alta impedância (no caso de não querermos ler uma palavra da ROM), impedindo a

memória de enviar dados para o barramento de dados, ou, pelo contrário, deixar passar para o barramento o conteúdo de uma palavra lida da ROM.

Passemos, então, à questão das temporizações num acesso à ROM.

Uma ROM possui um certo tempo de propagação (ou de atraso), desde o instante em que se aplica um novo endereço às entradas e o instante em que, nas saídas, aparece o conteúdo da palavra lida desse endereço. Este tempo de propagação toma a designação específica de **tempo de acesso a partir do endereço, t_{AA}**, tal como descreve a Figura 17.6 para uma ROM típica como a da Figura 17.5.

Tempo de acesso a partir do endereço, t_{AA}

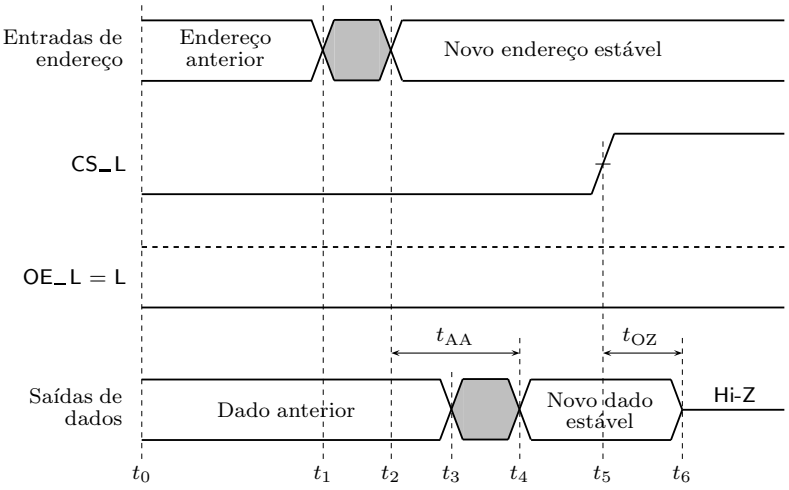


Figura 17.6: Parte das temporizações na operação de leitura de uma palavra de uma ROM típica

A forma de onda de cima representa os sinais aplicados às entradas de endereço (independentemente do número de linhas de endereço), a segunda representa a linha de Chip Select, $CE_$L, a terceira apresenta a linha de Output Enable, $OE_$L, activada em permanência, e a de baixo representa os dados de saída (independentemente do número de linhas de saída).

No instante t_0 as linhas de endereço estão com determinados níveis de tensão, umas a H e outras a L (notar a forma de representação desses níveis). Nesse instante as entradas de Chip Select e de Output Enable estão activas, pelo que as saídas da ROM exibem um dado anterior, resultante de uma leitura prévia da ROM (o dado correspondente ao endereço anterior).

Em t_1 são aplicados às linhas de endereço novos níveis de tensão, que correspondem a um novo endereço. Algumas dessas linhas mudam de nível, e outras não. Contudo, é impossível garantir que as linhas que mudam de nível o fazem em simultâneo. Ou seja, durante um intervalo de tempo representado a cinzento na figura, as linhas de endereço estão instáveis, a ajustar-se ao novo endereço a partir do endereço anterior. Até que em t_2 o novo endereço fica estável. A partir desse instante a ROM começa a descodificar o novo endereço e a seleccionar a palavra correspondente.

Em t_3 os dados de saída começam a mudar para reflectir o novo dado que é lido da ROM. Naturalmente, as mudanças nas linhas de dados não ocorrem

simultaneamente, pelo que entre t_3 e t_4 as saídas da ROM mantêm-se instáveis. Finalmente, em t_4 as saídas passam a reflectir de forma estável o dado que estava memorizado na palavra que acabou de ser endereçada.

O intervalo de tempo entre t_2 e t_4, desde que o novo endereço está estável até que aparece de forma estável nas saídas o dado correspondente, é o tempo de acesso a partir do endereço, t_{AA}. Para uma ROM bipolar t_{AA} é da ordem de grandeza de 30 a 90 ns, para uma ROM nMOS é da ordem de 35 a 500 ns, e para uma ROM CMOS é da ordem de 20 a 60 ns. Quando se afirma que temos uma ROM de 100 ns, por exemplo, estamos a referir-nos a este tempo.

Tempo de Output Disable, t_{OZ}

Outro parâmetro temporal importante é o **tempo de Output Disable, t_{OZ}**, medido entre o instante em que o Chip Select fica inactivo e o instante em que as saídas entram em alta impedância, entre t_5 e t_6. Notemos como se representa, no diagrama temporal, uma situação de alta impedância nas saídas (nem a H nem a L).

Na Figura 17.7 apresenta-se outra situação possível e diferente da anterior em que se activa o Chip Select depois de um endereço ficar estável, admitindo ainda que o Output Enable se mantém activado em permanência.

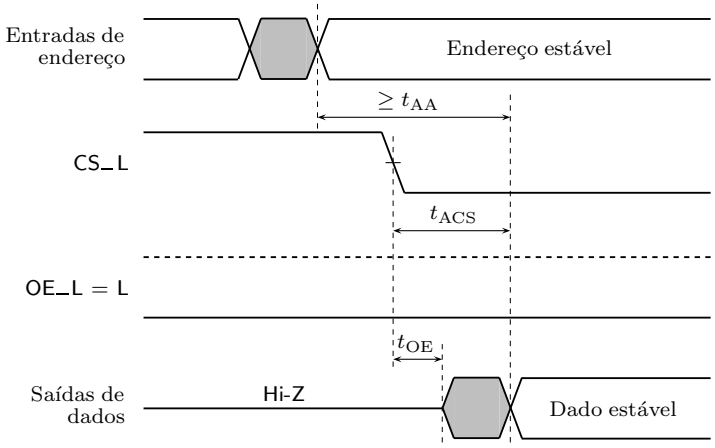


Figura 17.7: Outra parte das temporizações na operação de leitura de uma palavra de uma ROM típica

Tempo de Output Enable, t_{OE}

Neste caso, as saídas que estavam em alta impedância pelo facto de o CS estar desactivado, saem da situação de alta impedância depois de decorrido um **tempo de Output Enable, t_{OE}**. Se fosse a linha $OE_$L a ficar activada em vez da linha $CS_$L, teríamos a mesma situação. No fundo, t_{OE} mede o intervalo de tempo que decorre desde que $CS_$L ou $OE_$L ficam activos até que as saídas saem da situação de alta impedância. Se os endereços estiverem estáveis há tempo suficiente, o dado que se obtém nas saídas é estável. Caso contrário, é instável.

Tempo de acesso a partir do CS, t_{ACS}

O **tempo de acesso a partir do CS, t_{ACS}** mede o tempo de acesso a partir do instante em que o CS fica activo até que as linhas de saída ficam estáveis (em algumas ROMs o CS tem um efeito ligeiramente diferente do de OE, ao contrário do que sugere a Figura 17.4, e nesses casos o t_{ACS} é diferente do t_{OE}, como mostra a figura anterior.)

Enquanto os tempos de acesso t_{ACS} e t_{AA} não forem cumpridos, não podemos

esperar no barramento de dados o dado correspondente à palavra endereçada, mesmo que as saídas da ROM tenham deixado de estar na situação de alta impedância por já ter decorrido o tempo t_{OE}. Ou seja, depois de decorrido t_{OE} mas antes de t_{ACS} e t_{AA} terem terminado, obtemos no barramento de dados algo que não tem a ver com o dado pretendido (daí estar indicado a cinzento).

17.1.8 Expansão de ROMs

Quando se pretendem obter ROMs com dimensões maiores do que as ROMs de que dispomos, podemos distinguir dois problemas fundamentais: (i) queremos aumentar a dimensão das palavras lidas para cada endereço, mantendo o número de palavras; ou (ii) queremos aumentar o número de palavras, mantendo a dimensão das palavras; ou (iii) queremos fazer as duas coisas.

A expansão do número de bits por palavra realiza-se com grande simplicidade, por simples justaposição de ROMs. Exemplifica-se na Figura 17.8 com um sistema de 1k × 16 bits, formado a partir de duas ROMs de 1k × 8. Ou seja, formamos uma ROM que, globalmente, possui 1k palavras de 16 bits.

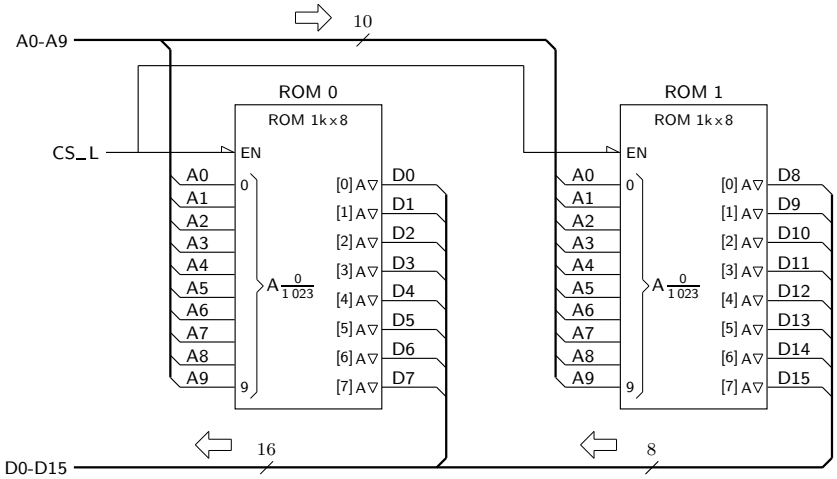


Figura 17.8: Expansão de ROMs que duplica a dimensão de cada palavra, de 8 para 16, mantendo o número de palavras em 1k

Como podemos verificar pela figura, a ROM 0 contém a parte menos significativa de uma palavra, $D0 - D7$, enquanto que a ROM 1 contém a parte mais significativa, $D8 - D15$ — mas, obviamente, podíamos ter escolhido colocar na ROM 0 as partes altas das palavras e na ROM 1 as partes baixas.

A operação de leitura é feita simultaneamente nas duas ROMs (o CS é comum às duas) e para o mesmo endereço (as linhas de endereço são comuns). Nessas condições, cada uma das ROMs envia para o barramento de dados a metade da palavra que nela foi previamente escrita.

Na figura é ilustrada também uma representação simplificada para os barramentos de dados, $D0-D15$, e de endereços, $A0-A9$, e correspondentes interligações às ROMs.

Para a expansão do número de palavras da memória, há que colocar os diversos integrados em paralelo e prolongar para o exterior da memória a função de descodificação através de um descodificador. Esse descodificador interage com os diversos integrados através das respectivas linhas de Enable.

O exemplo que se segue corresponde à implementação de um sistema de memória ROM com a dimensão de 4k × 8, usando os integrados anteriores e um descodificador binário de 2 bits (Figura 17.9).

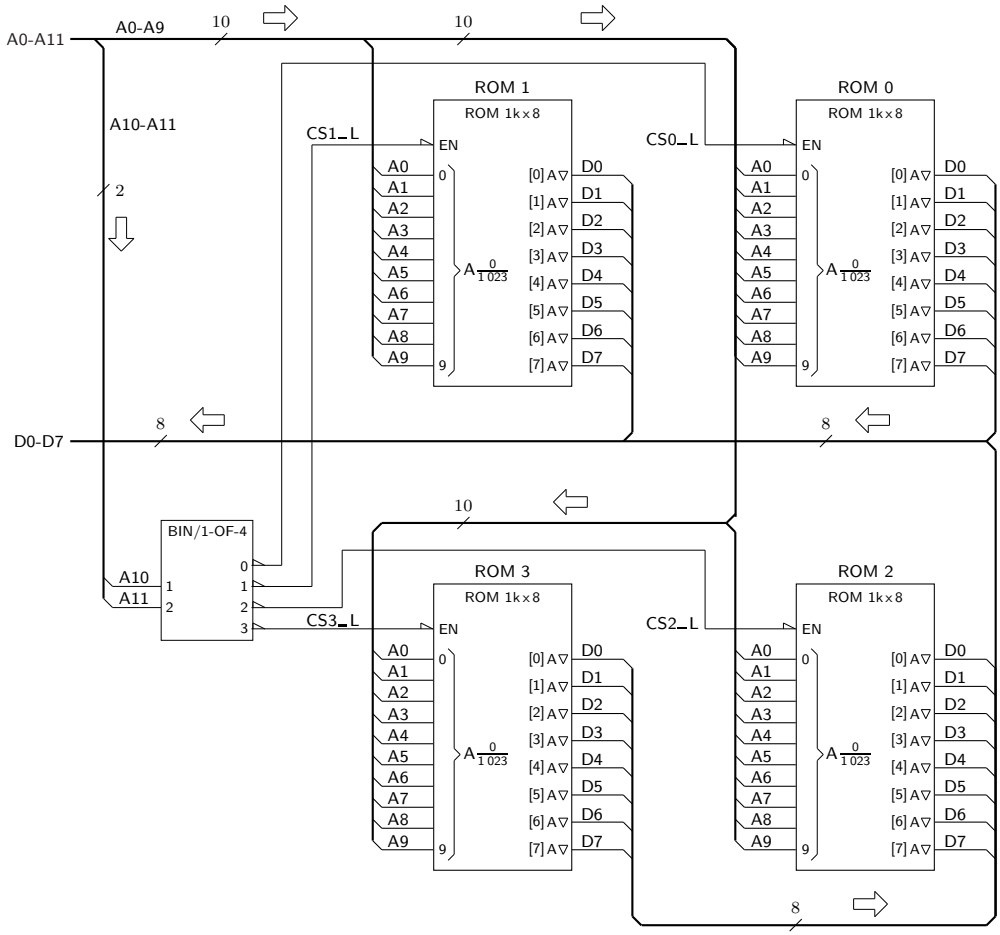


Figura 17.9: Expansão de ROMs que quadruplica o número de palavras, de 1k para 4k, mantendo a dimensão de cada palavra em 8 bits

Porque vamos usar 4k endereços, precisamos de 12 linhas de endereço, $A0$ a $A11$. Os bits de endereço que não podem ir para as ROMs, $A10$ e $A11$, são utilizados pelo descodificador externo, que activa a entrada de CS (Chip Select) de cada uma das ROMs presentes no sistema, obviamente uma de cada vez. Por exemplo, se $A10$ = H e $A11$ = L, a ROM 1 vem activada. Mas se $A10$ = H e $A11$ = H, então é a ROM 3 que vem activada.

Neste sistema de memória as diversas ROMs ocupam os endereços (em hexadecimal) que se indicam na Tabela 17.1.

Tabela 17.1: Tabela com os endereços ocupados por cada uma das ROMs da Figura 17.9

ROM	Endereços
0	000h a 3FFh
1	400h a 7FFh
2	800h a BFFh
3	C00h a FFFh

Naturalmente, podíamos, se assim entendessemos, aumentar o número de palavras e a dimensão de cada uma delas, conjugando as expansões das Figura 17.8 e 17.9.

17.2 "Random Access Memories" (RAMs)

17.2.1 Tipos de RAMs

As RAMS são memórias de leitura/escrita, isto é, memórias em que as operações de leitura e de escrita são igualmente frequentes do ponto de vista estatístico e que, por essa razão, são efectuadas aproximadamente com os mesmos tempos de atraso.

Desta forma as RAMs distinguem-se das ROMs, já que para estas últimas temos uma das seguintes situações:

— a operação de escrita não pode ser de todo efectuada pelo utilizador (como acontece com as MROMs);

— a operação de escrita só pode ser efectuada uma única vez (é o que sucede com as PROMs);

— a operação de escrita pode ser efectuada mais do que uma vez, mas torna-se necessário retirar a memória do circuito para a reprogramar externamente (no caso das EPROMs);

— a operação de escrita pode ser efectuada múltiplas vezes sem retirar a memória do circuito, mas é uma operação pouco eficiente do ponto de vista temporal (comparativamente com a operação de leitura), para além de necessitar de circuitos especiais; ou seja, as operações de escrita apenas ocorrem em circunstâncias muito especiais — como acontece, por exemplo, nas memórias Flash e nas EEPROMs — quando se pretende substituir um programa armazenado na memória.

Memórias RAM estáticas (SRAMs)

Memórias RAM dinâmicas (DRAMs)

Existem RAMs de dois tipos: **estáticas**, ou **SRAMS**, e **dinâmicas**, ou **DRAMs**. As RAMs estáticas são dispositivos em que os diversos bits são armazenados em

dispositivos do tipo latch controlado (ainda que estruturalmente muito simplificados) que podem manter indefinidamente o seu conteúdo enquanto estiverem alimentados electricamente.

As RAMs dinâmicas são dispositivos em que cada bit é representado pela carga eléctrica de um pequeno condensador. Como todos os condensadores, estes têm fugas, pelo que apenas conseguem manter a carga durante um intervalo de tempo muito limitado. Daí que seja preciso, quando se utiliza uma RAM dinâmica, incluir um **circuito de refrescamento** de constante reescrita do conteúdo da RAM, de forma a que todos os condensadores vejam periodicamente reposta a respectiva carga.

Circuito de refrescamento

Neste curso preocupar-nos-emos apenas com as RAMs estáticas.

17.2.2 Símbolos das RAMs

Uma RAM é um dispositivo de memória com uma estrutura idêntica à de uma ROM (ver a Figura 17.2 na página 332), excepto que as células são agora constituídas por latches modificados.

As RAMs são, então, organizadas em palavras de um certo comprimento (número de bits), que podem ser acedidas em operações de escrita ou de leitura.

A palavra pretendida é referenciada por um endereço aplicado a um **barramento de endereços**. A operação de **leitura** de uma palavra da RAM ou de **escrita** numa palavra da RAM é referenciada por duas linhas separadas ou, noutros casos, por uma única linha.

Barramento de endereços (de uma RAM)

Leitura (de uma palavra) da RAM

Escrita (numa palavra) da RAM

Os dados a escrever são introduzidos por um **barramento de entrada** e os dados lidos podem ser acedidos num **barramento de saída**. Em alguns casos apenas existe um único **barramento de dados**, naturalmente bidireccional.

Barramento de entrada

Barramento de saída

Barramento de dados

Consideremos, na Figura 17.10, o símbolo IEC de uma RAM típica de $1\text{k} \times 8$ (isto é, com 1 024 palavras de 8 bits), com barramentos separados para a entrada e para a saída de dados.

O simbolo é semelhante ao de uma ROM, pelo que a seguir apenas mencionaremos as principais diferenças.

A primeira tem a ver com a existência de um bloco de controlo comum, que o símbolo simplificado da ROM da Figura 17.5 não possui (por isso, o símbolo da ROM é simplificado).

Neste caso existem dois barramentos de dados, um de dados de entrada designado por $(DATAIN0-7)\_\text{H}$, e outro de dados de saída, $(DATAOUT0-7)\_\text{H}$. Trata-se, em ambos os casos, de barramentos de 8 bits. Quanto ao barramento de endereços, tem as linhas necessárias para aceder a todas as palavras de memória — no caso, 10 linhas de endereços, $(ADDR0-9)\_\text{H}$.

A linha $READ\_\text{H}$ permite efectuar uma operação de leitura da RAM, enquanto que a linha $WRITE\_\text{H}$ permite realizar uma operação de escrita na RAM. Naturalmente, nestes casos é da responsabilidade do utilizador da memória garantir que apenas uma das operações vem activada de cada vez (mas podem estar as duas inactivas, e então não se faz nem a leitura nem a escrita numa palavra da memória).

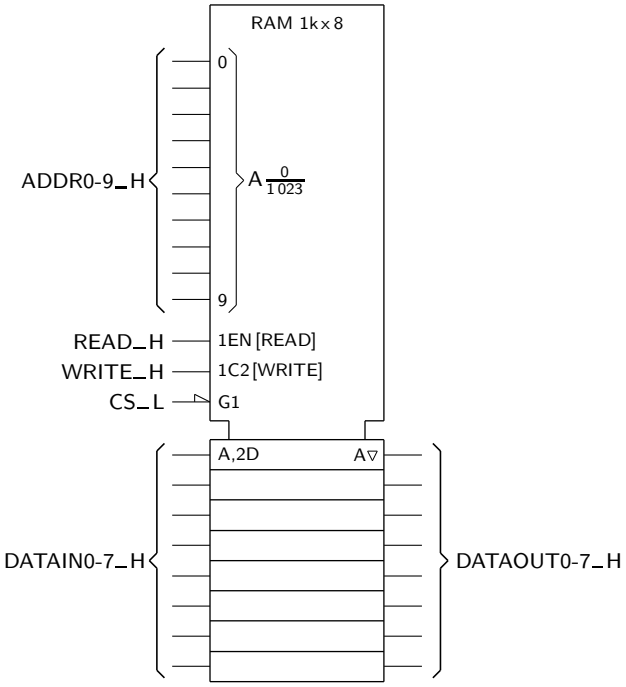


Figura 17.10: Símbolo IEC de uma RAM de 1k × 8 com dois barramentos de dados unidireccionais, um de entrada e um de saída, e saídas tri-state

De notar o qualificador de entrada G1 que, quando activo, permite que essas operações possam ser efectuadas, através do qualificador 1 (dependência And).

A esta entrada ligamos, habitualmente, uma linha com a designação $CS_$L (CS significa Chip Select). Quando esta entrada vem inactiva, inibe o funcionamento da RAM, colocando as saídas em alta impedância através do qualificador EN (impedindo, deste modo, a memória de enviar dados para o barramento de dados de saída numa operação de leitura), e impedindo as operações de escrita por meio do qualificador 2.

Existem várias estruturas alternativas à ilustrada. A variante mais comum utiliza um único barramento de dados, bidireccional. Naturalmente, nestes casos não há necessidade de controlar independentemente a escrita e a leitura. Por isso, existe apenas uma linha de controlo que, ora promove a operação de leitura, ora promove a operação de escrita.

É comum designar a linha de controlo por $READ_$H/$WRITE_$L, sendo que existirá uma operação de leitura se se aplicar um nível H à linha, e uma operação de escrita se se aplicar um nível L. Notemos que, neste caso, não conseguimos controlar esta linha por forma a impedir uma operação de eleitura ou de escrita, ao contrário do que sucede quando as variáveis de $READ$ e de $WRITE$ são separadas. Para que não se efectue nenhuma das operações, teremos de controlar o Chip Select da memória.

A Figura 17.11 ilustra uma memória RAM com essa arquitectura e com a dimensão da memória anterior.

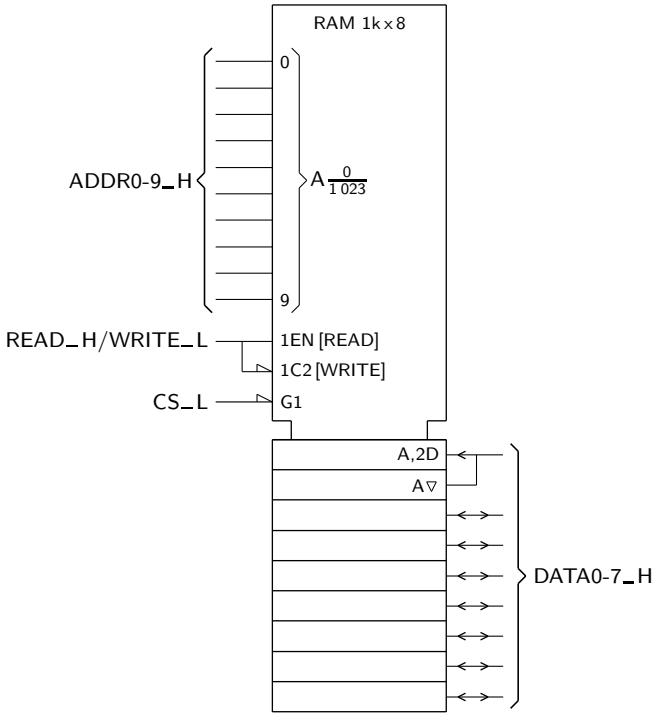


Figura 17.11: Símbolo IEC de uma RAM de 1k $\times$ 8 com um único barramento de dados, bidireccional, e com saídas tri-state

17.2.3 Estrutura de uma RAM estática (SRAM)

Na prática, uma célula de memória estática em tecnologia MOS (mas também há memórias bipolares TTL e BiCMOS) é constituída por um latch SR simplificado, formado por dois inversores entrecruzados, e por dois transistores nMOS, ligados como mostra a Figura 17.12.

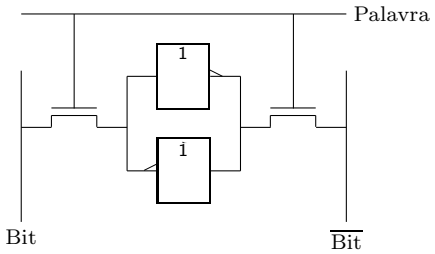


Figura 17.12: Estrutura de uma celula de memória estática em tecnologia MOS, com um total de 6 transistores

Às células encontram-se ainda associados outros circuitos para as operações de escrita e de leitura, para além dos descodificadores de linhas e de colunas.

Como, porém, os alunos não possuem os conhecimentos necessários de Electrónica Digital para podermos analisar o comportamento da célula e dos cir-

cuitos auxiliares, vamos recorrer ao seu equivalente funcional, apresentado na Figura 17.13.

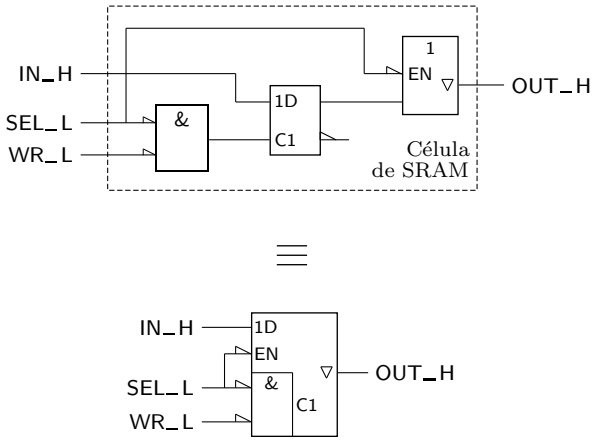


Figura 17.13: Equivalente funcional de uma célula de memória SRAM. De notar a existência de um latch D controlado, precedido por uma lógica de controlo da sua entrada de Enable e seguido por uma lógica de controlo da sua saída Q_H

O bit a escrever é apresentado à entrada D de um latch controlado, mas só é escrito se a sua linha de Enable estiver activa, o que quer dizer que a linha de Selecção da célula, SEL_L, deve estar activa, e a linha de Escrita, WR_L, também deve estar activa. A linha SEL_L controla não apenas a escrita no latch, mas também a leitura do seu conteúdo. Se na linha SEL_L aplicarmos um nível H, a saída da célula vem em alta impedância.

Estrutura de uma SRAM

As SRAMs utilizam células deste tipo numa estrutura matricial idêntica à da Figura 17.2, como se afirmou atrás. A Figura 17.14 ilustra, de forma simplificada, a estrutura interna de uma SRAM com 4 palavras de 2 bits, no caso em que se usam dois barramentos unidireccionais e independentes para a entrada e saída da dados.

A utilização de Buffers tri-state nas saídas permite que várias destas memórias (ou ainda ROMs, microprocessadores, etc, com capacidade tri-state) possam partilhar o mesmo barramento de saída.

Naturalmente, podemos de maneira fácil expandir esta estrutura para SRAMs de maiores dimensões. Por outro lado, veremos mais à frente como modificar a estrutura desta SRAM para o caso em que se usa um único barramento bidireccional para a entrada e saída de dados.

Como se pode ver na Figura 17.14, a selecção de uma palavra da SRAM é feita à custa de um descodificador, que verá uma das suas linhas de saída activada (a L) de cada vez, consoante o endereço aplicado à memória (não devemos, contudo, esquecer que as RAMs reais utilizam descodificadores de linha e de coluna em descodificação coincidente, em vez de um descodificador único).

Repare-se que, para que haja escrita de um bit na célula que se encontra na intersecção de uma dada linha e coluna, terá de vir seleccionada a palavra correspondente (activando-se a entrada SEL_L da célula), e também a coluna correspondente (activando-se a entrada WR_L da célula).

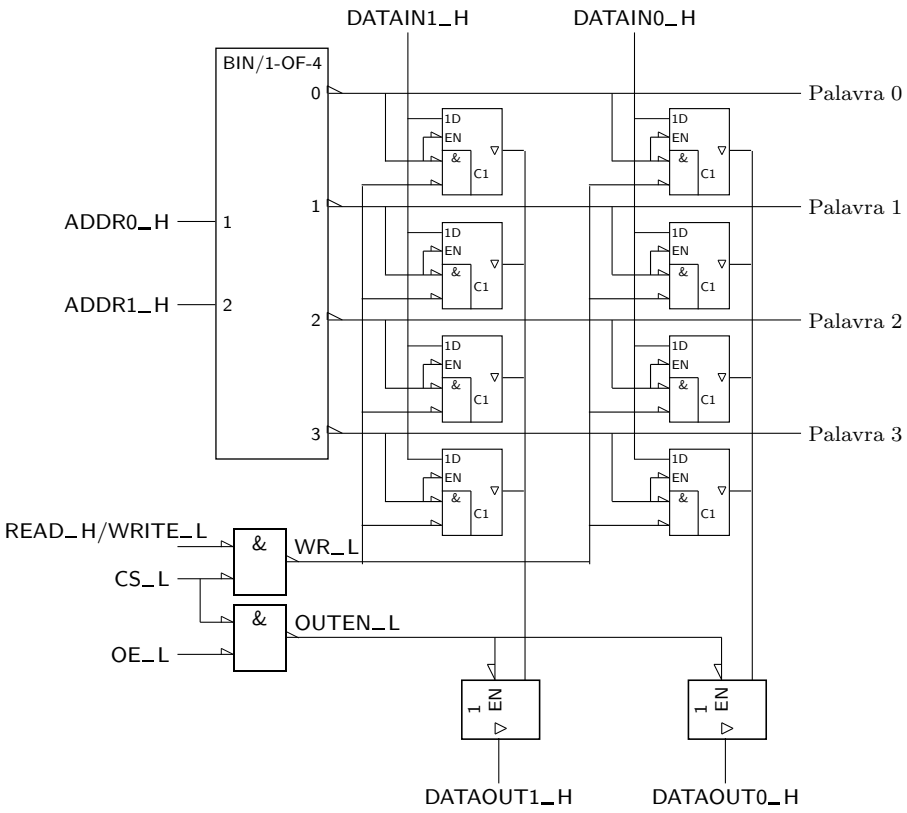


Figura 17.14: Estrutura de uma SRAM de 4×2 com barramentos independentes e unidireccionais para a entrada e saída de dados

Deve também notar-se que as células das palavras que não são seleccionadas pelo descodificador (todas as palavras excepto a que é endereçada) ficam com as suas saídas em alta impedância, ou seja, não interferem com as saídas das células da palavra seleccionada.

Operação de escrita numa SRAM

Consideremos uma **operação de escrita** na SRAM.

Quando a linha $READ\_H/WRITE\_L$ vem a L (e desde que a memória tenha sido seleccionada pela activação do seu Chip Select, CS_L), as entradas WR_L de todas as células vêm activadas, mas apenas é operada uma escrita na palavra endereçada, já que apenas as células dessa linha possuem as suas entradas SEL_L activas. Nestas condições, escrevem-se nessa palavra, e apenas nessa, os bits presentes no barramento $DATAIN\_H$.

Operação de leitura de uma SRAM

Consideremos agora uma **operação de leitura** da SRAM.

Neste caso a linha $READ\_H/WRITE\_L$ vem a H, o que inactiva todas as entradas WR_L das células, pelo que não pode escrever-se nos latches (naturalmente). A palavra seleccionada pela linha SEL_L debita para o barramento de saída o seu conteúdo (se a entrada de Output Enable, OE_L, estiver activa e o Chip Select também estiver activo), e as restantes palavras, não seleccionadas, têm as suas saídas em alta impedância, como vimos anteriormente.

Finalmente, constatemos que, para impedir que numa operação de escrita se leia

igualmente para o exterior a palavra seleccionada (porque, estando SEL_H activo, os bits da palavra seleccionada aparecem nas saídas das células da palavra), devemos assegurar-nos que o Output Enable está inactivo durante a escrita.

Vamos considerar em seguida as modificações a fazer ao circuito de entrada/saída para acomodar um barramento tri-state bidireccional.

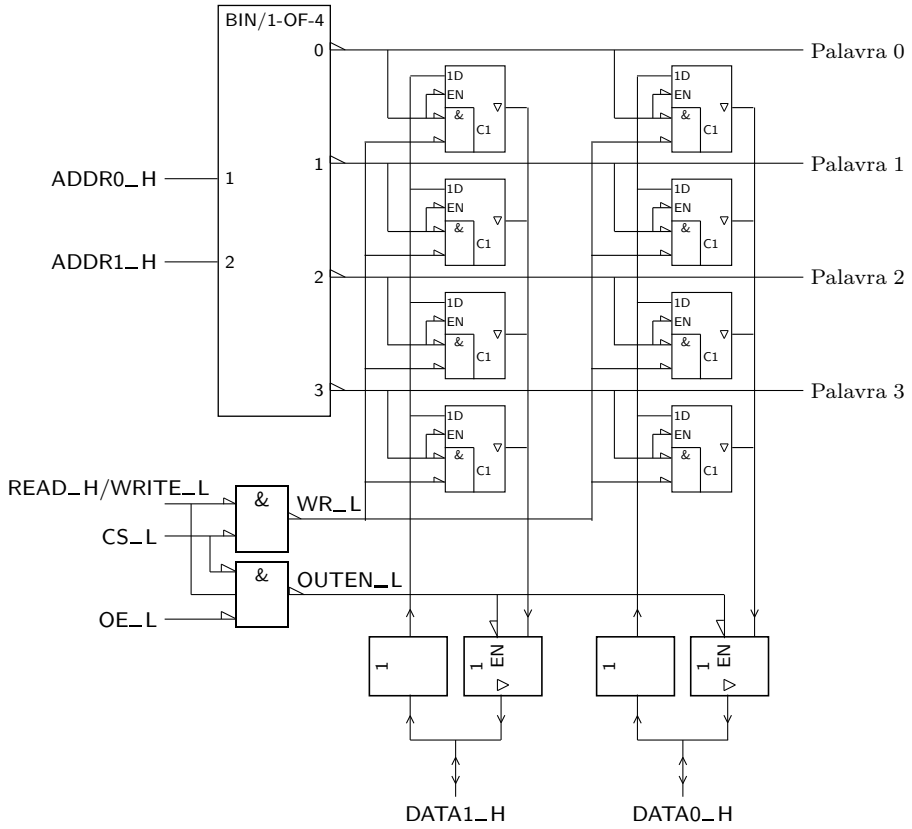


Figura 17.15: Uma SRAM com um barramento tri-state bidireccional usa um circuito de entrada/saída com um Buffer bidireccional comandado pelas 3 linhas de controlo

Buffer de saída

Buffer de entrada

A inclusão dos Buffers de entrada permite diminuir significativamente o fan-out dos circuitos que atacam as linhas $DATA1\_H$ e $DATA0\_H$.

Como podemos observar na Figura 17.15, colocam-se **Buffers de saída**, do tipo tri-state, no caminho das linhas de saída das células de memória (OUT_H) para o barramento de dados, e **Buffers de entrada** simples no caminho do barramento de dados para as linhas de entrada das células da memória (IN_H).

A entrada de Enable dos Buffers de saída vem comandada por CS_L e por OE_L, como no caso anterior (com dois barramentos diferenciados), e ainda pela linha $READ\_H/WRITE\_L$.

Quando é aplicado um L à linha $READ\_H/WRITE\_L$ (operação de $WRITE$, se CS_L e OE_L estiverem activos), a saída $OUTEN\_L$ da porta AND inferior vem inactiva, e é feito o Disable das saídas dos Buffers de saída, o que impede a SRAM de enviar dados para o barramento. Ou seja, nestas condições apenas pode ser feita uma escrita na SRAM do dado proveniente do barramento.

Quando se aplica um H à linha $READ\_H/WRITE\_L$ (operação de $READ$, se CS_L e OE_L estiverem activos), a saída da porta inferior, $OUTEN\_L$, vem activa e é feito o Enable das saídas dos Buffers de saída, o que permite que o dado contido na palavra seleccionada seja lido para o barramento.

Se CS_L ou OE_L ou ambos estiverem inactivos, não pode haver leitura da nem escrita na SRAM. Com efeito, embora o dado que estiver no barramento passe através dos Buffers de entrada e esteja presente às entradas de todas as células, ele não podem ser escrito nas células porque a linha WR_L está inactiva. Por outro lado, porque $OUTEN\_L$ também está inactiva, a SRAM não envia nada para o barramento.

17.2.4 Ciclos de leitura e de escrita numa SRAM

Ciclo de escrita
Ciclo de leitura

A execução da escrita numa palavra em memória, faz-se num **ciclo de escrita** e, do mesmo modo, a leitura é feita num **ciclo de leitura**.

Um ciclo de leitura de uma RAM estática é igual à operação de leitura de uma ROM, que discutimos na Subsecção 17.1.7. Por essa razão não apresentaremos aqui esse ciclo.

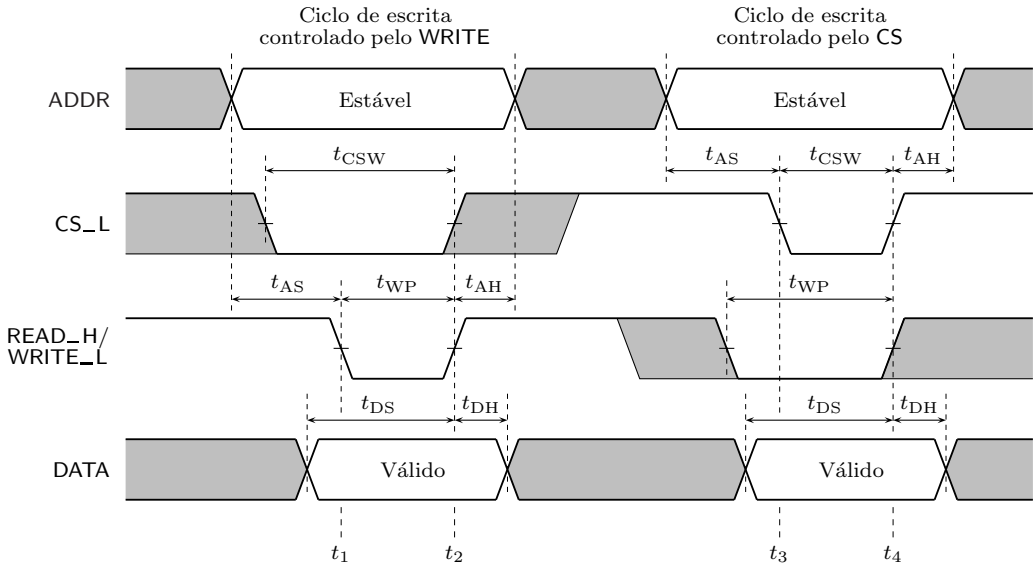


Figura 17.16: Ciclo de escrita numa palavra de uma SRAM

O ciclo de escrita na SRAM está dependente da activação simultânea dos sinais de controlo CS e $WRITE$ (como sabemos, este último é conseguido colocando a linha $READ\_H/WRITE\_L$ a L). Ora estes dois sinais não têm que ficar activos em simultâneo e ficar inactivos em simultâneo.

Assim, na Figura 17.16 apresentam-se duas situações distintas, a primeira em que o $WRITE$ fica activo depois do CS, e a segunda em que o CS fica activo depois do $WRITE$.

Ciclo de escrita controlado pelo WRITE

No primeiro caso o ciclo de escrita começa a partir do instante t_1 em que os dois sinais vêm activos, e termina quando os dois ficam inactivos, em t_2. Trata-se

de um **ciclo de escrita controlado pelo WRITE** porque, depois do CS ficar activo, o ciclo só se inicia quando o $WRITE$ vem activo.

No segundo caso temos a situação contrária: o ciclo de escrita começa a partir do instante t_3 em que os dois sinais vêm activos, e termina quando os dois ficam inactivos, em t_4. Trata-se de um **ciclo de escrita controlado pelo CS** porque, depois do $WRITE$ ficar activo, o ciclo só se inicia quando o CS vem activo.

Ciclo de escrita controlado pelo CS

Consideremos então um ciclo de escrita controlado pelo $WRITE$, com a seguinte sequência de operações:

— o endereço onde se quer escrever uma palavra é colocado no barramento de endereços; antes, as linhas de endereço estão instáveis porque mudaram para o novo endereço;

— o CS vem activado; como os endereços foram alterados há pouco tempo, o que está no barramento de dados não é garantidamente válido (pode ser o conteúdo do endereço anterior, ou podem ser dados sem sentido, provocados pela transição de endereços);

— actua-se em seguida o $WRITE$; a partir daqui inicia-se o ciclo de escrita;

— há que esperar o tempo necessário para que a escrita se efectue com sucesso;

— desactiva-se o $WRITE$ e, simultaneamente ou não, também o CS; em todo o caso, o ciclo de escrita termina;

— após algum tempo, podem ser retirados os dados do barramento;

— a operação é completada com a mudança do endereço para um novo ciclo, seja ele de leitura ou de escrita.

Num ciclo de escrita controlado pelo CS temos uma situação semelhante, também descrita na Figura 17.16.

Quanto às temporizações nos ciclos de escrita, devemos ter em atenção que as células de memória da SRAM são constituídas por latches. Logo, devemos assegurar os correspondentes tempos de preparação ou de set-up, t_{su}, e de manutenção ou de hold, t_{h}, quer em relação aos endereços, quer em relação aos dados.

Os tempos de preparação e de manutenção dos endereços são definidos relativamente aos inícios e aos fins dos ciclos de escrita. São eles o **tempo de preparação** ou **tempo de set-up do endereço, $\mathbf{t_{AS}}$**, antes de se dar início ao ciclo, e o **tempo de manutenção** ou **tempo de hold do endereço, $\mathbf{t_{AH}}$**, depois de terminar o ciclo.

Tempo de preparação (set-up) do endereço, t_{AS}

Tempo de manutenção (hold) do endereço, t_{AH}

Quanto aos dados, os tempos de preparação e de manutenção são definidos relativamente aos fins dos ciclos. Ou seja, há que assegurar que um determinado dado está estável no barramento antes e depois de terminar o ciclo que o escreve na memória.

Temos, por conseguinte, um **tempo de preparação** ou **tempo de set-up do dado, $\mathbf{t_{DS}}$**, antes de terminar o ciclo, e um **tempo de manutenção** ou **tempo de hold do dado, $\mathbf{t_{DH}}$**, depois de terminar o ciclo.

Tempo de preparação (set-up) do dado, t_{DS}

Tempo de manutenção (hold) do dado, t_{DH}

Finalmente, os impulsos de $WRITE$ e de CS devem ter uma duração mínima, respectivamente o **tempo de duração do WRITE, $\mathbf{t_{WP}}$**, e o **tempo de duração do CS, $\mathbf{t_{CSW}}$**.

Tempo de duração do WRITE, t_{WP}

Tempo de duração do CS, t_{CSW}

17.2.5 Expansão de RAMs

A questão da expansão de RAMs segue a par e passo com o processo de expansão das ROMs.

Ou seja, podemos expandir a dimensão de cada palavra colocando RAMs em paralelo, alimentadas por um barramento de endereços comum e com cada uma dss RAMs a recolher ou a enviar uma parte dos dados para o barramento de dados comum.

Por exemplo, se quisermos formar palavras de 16 bits com RAMs de 4 bits, ligamos 4 RAMs em paralelo, com uma delas ligada a $D0-D3$, outra a $D4-D7$, outra a $D8-D11$, e a última a $D12-D15$ do barramento de dados $D0-D15$.

Para expandir o número de palavras, arranjamos um descodificador externo que actua os CE (ou CS) de cada uma delas à custa das linhas de endereço suplementares necessárias ao número total de palavras. Ou seja, todas as RAMs são alimentadas em paralelo pelas linhas de endereço comum, e as linhas de endereço suplementares constituem as entradas do descodificador.

Por exemplo, se possuirmos RAMs de 1k×8 e quisermos ocupar uma zona de memória com 4k, usamos 4 RAMs, com cada uma delas a ocupar uma zona de 1k. Como as RAMs possuem 10 entradas de endereço, aplicamos às 4 RAMs as linhas de endereço $A0-A9$. Mas como a zona total a descodificar possui 4k, são precisas mais 2 linhas de endereço, $A10$ e $A11$, que são as entradas de um descodificador binário de 2 bits. As 4 saídas do descodificador constituem os 4 sinais de CS, um por cada RAM.

Finalmente, podemos aumentar simultaneamente a dimensão e o número de palavras, utilizando as duas técnicas anteriores, tal como fazemos com as ROMs.

Capítulo 18

Lógica Programável

18.1 "Read Only Memories" (ROMs)

Uma ROM é um dispositivo de memória e, como tal, foi estudado no Capítulo 17, conjuntamente com outros tipos de memórias (RAMs). Mas uma das principais aplicações das ROMs, como vimos, é na implementação de lógica combinatória, quando se pretende realizar uma função booleana geral de uma variável booleana geral. Neste sentido, uma ROM pode ser considerada como um dispositivo lógico programável, ou **PLD** (**"Programmable Logic Device"**), e é nesse contexto que a iremos estudar no presente capítulo.

"Programmable Logic Device" (PLD)

Consideremos na Figura 18.1 uma ROM (hipotética) de 16×1, na qual queremos programar a tabela de verdade física que se encontra representada na Tabela 18.1 (na realidade teremos que enviar a tabela de verdade para o fabricante da ROM, para ele gerar a máscara com as ligações nos sítios onde queremos níveis H e ausências de ligações onde queremos níveis L).

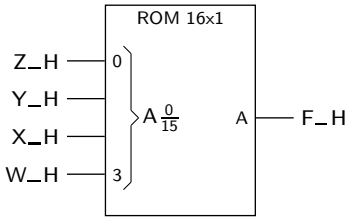


Figura 18.1: Símbolo IEC de uma ROM hipotética de 16×1

Do símbolo da ROM concluímos que esperamos obter a função booleana simples $F(W, X, Y, Z)$, em que W é a variável booleana simples com maior peso (porque está ligada à entrada de endereço com maior peso) e Z a de menor peso.

A forma mais simples que temos de representar graficamente o conteúdo desta ROM é como se apresenta na Figura 18.2: a única saída é constituída por um OR fictício com 16 entradas (uma por cada palavra da ROM), que recolhe os níveis H ou L previamente programados em cada uma das palavras, para essa saída.

Tabela 18.1: Tabela de verdade física que queremos programar na ROM da Figura 18.1

Palavra #	W_H A3	X_H A2	Y_H A1	Z_H A0	Dados
0	L	L	L	L	H
1	L	L	L	H	L
2	L	L	H	L	H
3	L	L	H	H	H
4	L	H	L	L	L
5	L	H	L	H	L
6	L	H	H	L	L
7	L	H	H	H	H
8	H	L	L	L	H
9	H	L	L	H	H
10	H	L	H	L	H
11	H	L	H	H	L
12	H	H	L	L	H
13	H	H	L	H	L
14	H	H	H	L	L
15	H	H	H	H	H

Porque o OR possui um elevado número de entradas, é usual representá-lo de forma simplificada, com as 16 entradas reduzidas a uma única linha, na qual se identificam as ligações aos 16 bits das 16 palavras.

Como de cada vez apenas uma palavra vem endereçada, segue-se que apenas uma entrada do OR vem com o nível H ou L que tiver sido programado para essa palavra. As restantes entradas vêm a L. Logo, do ponto de vista lógico, estamos efectivamente a implementar uma função OR com 15 entradas inactivas (a L) e a restante entrada activa ou inactiva, consoante o que tiver sido programado na palavra seleccionada.

É claro que as ROMs comerciais têm muito mais palavras e bits por palavra possibilitando, assim, a implementação de várias funções booleanas simples de um número de variáveis booleanas simples igual ao número de linhas de endereço. Por exemplo, uma ROM de $8\,\mathrm{k} \times 8$ pode implementar, no máximo, 8 funções booleanas simples (uma por cada saída) de 13 variáveis booleanas simples (porque $2^{13} = 8\,\mathrm{k}$).

A ROM da Figura 18.2 pode, então, ser encarada como uma matriz de ANDs fixos (os ANDs do descodificador binário) seguida de uma matriz de ORs programáveis (os ORs que acabámos de mencionar, um por cada saída).

Na Figura 18.3 sugere-se, então, uma representação gráfica para uma ROM, onde se salienta a matriz de ANDs fixos e a matriz de ORs programáveis.

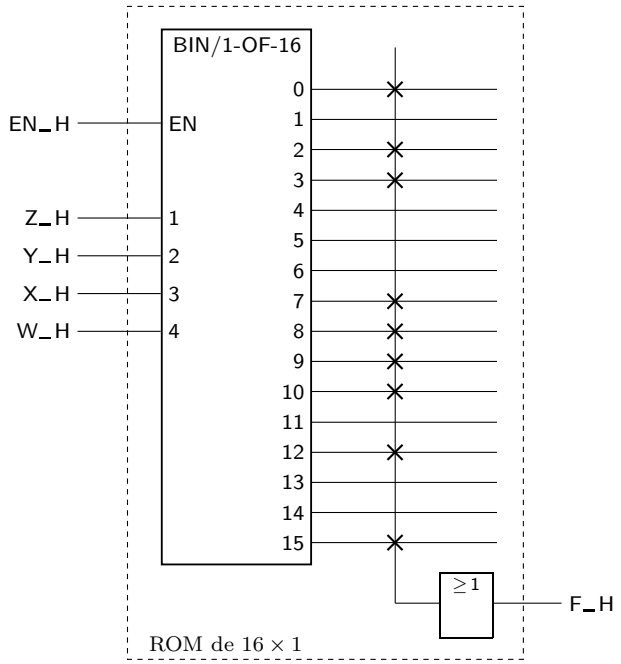


Figura 18.2: Uma forma de representar graficamente a ROM da Figura 18.1 considera a (única) saída como sendo a saída de um OR fictício com 16 entradas, umas programadas com o nível H (aquelas onde se incluiu uma cruz), e outras programadas com o nível L (onde não se colocou uma cruz)

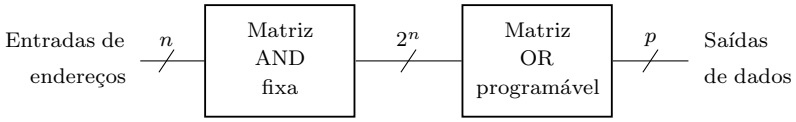


Figura 18.3: Uma ROM pode ser considerada como uma matriz de ANDs fixos (não programáveis), seguida de uma matriz de ORs programáveis

18.2 "Programmable Logic Arrays" (PLAs)

As ROMs que estudámos atrás constituem, como vimos, o primeiro exemplo de um PLD (dispositivo lógico programável), na medida em que permitem gerar funções booleanas simples de complexidade arbitrária, uma função por saída. Tal deve-se ao facto de o descodificador interno da ROM gerar todos os mintermos das funções, e de cada saída poder ser entendida como a soma lógica de um subconjunto qualquer desses mintermos (Figura 18.2).

A popularidade destes dispositivos na implementação de funções booleanas deve-se aos seguintes factores:

— como vimos atrás, é fácil e rápido determinar o conteúdo a inserir numa ROM para cada uma das funções de saída;

— existem linguagens e dispositivos que programam automaticamente uma ROM,

a partir das expressões booleanas das funções a implementar; torna-se, por isso, fácil alterar o conteúdo de uma ROM (se ela permitir que o seu conteúdo seja reprogramado); e

— os preços das ROMs (como, aliás, de outros PLDs) estão em baixa contínua, tornando-as cada vez mais económicas.

Contudo, as ROMs apresentam dois inconvenientes:

— uma ROM pode ser mais dispendiosa, consumir mais potência, ou ser mais lenta do que um circuito que use dispositivos SSI e MSI para implementar as mesmas funções, ou que um circuito que use outros dispositivos lógicos programáveis (como é o caso das PLAs e PALs que estudaremos de seguida); e

— quando o número de variáveis é muito elevado, um circuito baseado numa ROM pode tornar-se impraticável devido à limitação do número de entradas disponível nas ROMs comerciais.

Entradas e saídas de dados (de um PLD)

Estes inconvenientes foram explorados pelos fabricantes de circuitos integrados, levando à concepção de PLDs em que existe um certo número n de **entradas de dados**, um número p de ANDs e um número q de ORs e correspondentes **saídas de dados**, podendo ser programadas as ligações entre os ANDs e as entradas, por um lado, e entre os ORs e os ANDs, por outro.

Ou seja, estamos agora em presença de dispositivos formados por uma matriz de ANDs programáveis, seguida de uma matriz de ORs também programáveis, como mostra a Figura 18.4 (comparar com a estrutura de ANDs e de ORs de uma ROM, na Figura 18.3).

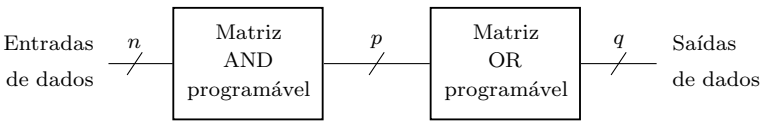


Figura 18.4: Uma PLA pode ser encarada como uma matriz de ANDs programáveis seguida de uma matriz de ORs programáveis

"Programmable Logic Array" (PLA)

Estas estruturas têm a designação de **PLAs**, ou **"Programmable Logic Arrays"**.

Naturalmente, estes dispositivos impõem restrições à estrutura das expressões booleanas a implementar, já que cada uma das q funções deve vir expressa numa soma de produtos, e o número total de implicantes disponíveis não pode ultrapassar p.

Por comparação, a implementação das mesmas funções com uma ROM não sofre qualquer espécie de restrições relativamente às suas expressões booleanas, já que o que fazemos, nesse caso, é somar lógicamente os mintermos necessários à primeira forma canónica de cada função, e todos os mintermos estão disponíveis nas saídas do descodificador interno à ROM.

Consideremos a estrutura de uma PLA muito simples, na Figura 18.5.

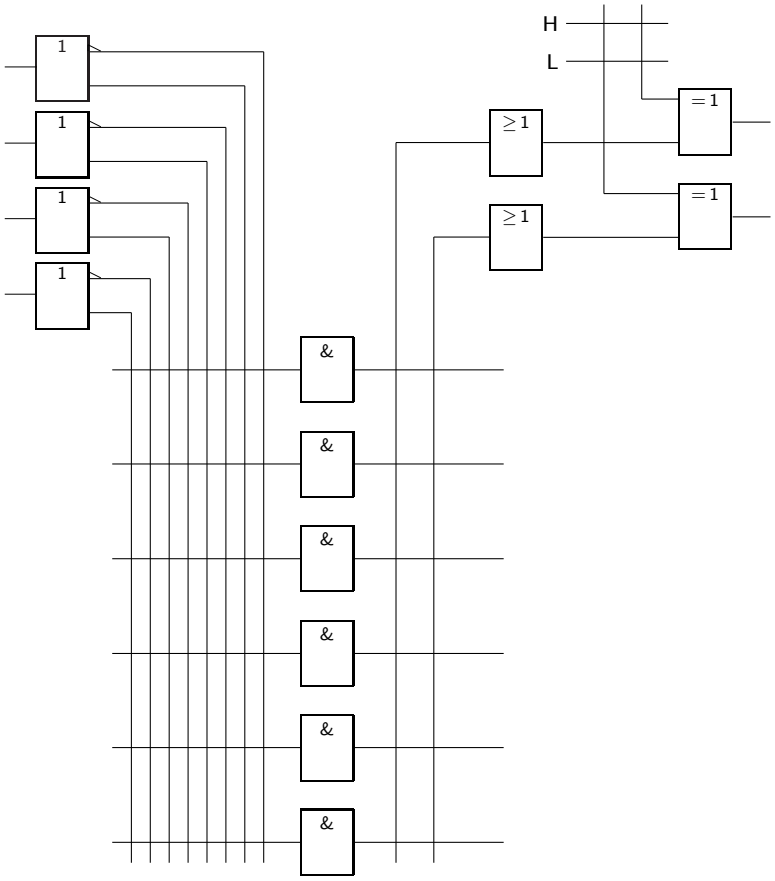


Figura 18.5: Estrutura de uma PLA com $n = 4$ entradas de dados, $p = 6$ ANDs, e $q = 2$ ORs. Este dispositivo permite implementar duas funções booleanas simples de 4 variáveis que possuam expressões booleanas em forma de somas de produtos limitadas a um máximo de 6 implicantes

Repare-se que a PLA tem 2 saídas, 4 entradas e 6 ANDs, o que permite implementar 2 funções booleanas simples das 4 variáveis de entrada, com expressões em somas de produtos limitadas a 6 implicantes.

Repare-se ainda que, nas saidas, as funções podem ser negadas ou não, conforme o que se programar no controlo das portas XOR finais.

Como exemplo, utilizemos a PLA anterior para construir as funções $F1$ e $F2$ dadas pelas tabelas de verdade físicas da Tabela 18.2.

Para implementar estas funções em ROM haveria apenas que gravar as tabelas na memória. Usando como exemplo uma ROM de 16×8, isso significaria realizar a programação expressa na Figura 18.6.

Trata-se de um processo simples mas que acarreta um grande desperdício de hardware, dado apenas aproveitarmos duas das oito possibilidades de geração de funções que a ROM permite.

No caso da PLA o processo será um pouco mais complexo, mas o resultado con-

Tabela 18.2: Tabelas de verdade físicas de duas funções booleanas simples que queremos implementar com a PLA da Figura 18.5

A_H	B_H	C_H	D_H	F1_H	F2_H
L	L	L	L	L	H
L	L	L	H	H	H
L	L	H	L	L	L
L	L	H	H	H	L
L	H	L	L	L	H
L	H	L	H	H	H
L	H	H	L	L	L
L	H	H	H	H	H
H	L	L	L	H	L
H	L	L	H	L	H
H	L	H	L	H	L
H	L	H	H	H	L
H	H	L	L	L	H
H	H	L	H	L	H
H	H	H	L	L	L
H	H	H	H	H	H

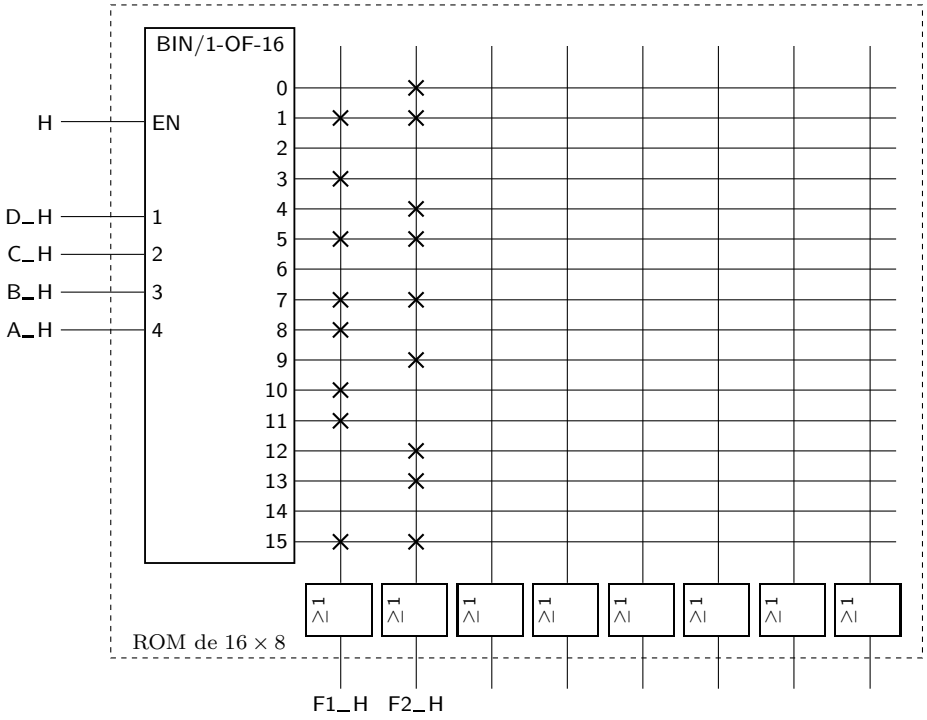
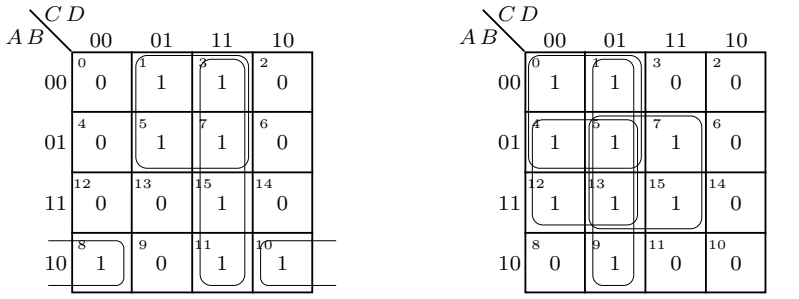


Figura 18.6: A implementação das funções booleanas simples da Tabela 18.2 com uma ROM de 16×8 significa a programação da ROM na forma que aqui se indica

some menos recursos, como veremos. Uma vez que necessitamos de implementar o circuito de tal modo que as funções vão estar em soma de produtos, é conveniente proceder à minimização das expressões booleanas para ver se conseguimos implementar as expressões com o número de ANDs limitados de que dispomos na PLA. Utilizaremos o método de Karnaugh, como mostra a Figura 18.7.



$F1 = \overline{A}\,D + C\,D + A\,\overline{B}\,\overline{D}$ $F2 = \overline{A}\,\overline{C} + B\,\overline{C} + \overline{C}\,D + B\,D$

Figura 18.7: Mapas de Karnaugh para a implementação das funções booleanas simples da Tabela 18.2

Obtemos:

$$F1 = \overline{A}\,D + C\,D + A\,\overline{B}\,\overline{D}$$
$$F2 = \overline{A}\,\overline{C} + B\,\overline{C} + \overline{C}\,D + B\,D\,.$$

A colocação de $F1$ na PLA não oferece qualquer problema: são utilizados três ANDs com as entradas ligadas de forma a conseguir os três produtos necessários, e utiliza-se um dos ORs para somar esses ANDs. O XOR final é programado para não negar a função (Figura 18.8).

Surge um problema, contudo, quando queremos implementar $F2$. É que são precisos 4 produtos, todos diferentes dos três já usados, e a PLA já só tem três.

Podemos, porém, reparar que, no mapa de Karnaugh, se agruparmos os "0"s em vez dos "1"s, obteremos uma expressão mais simples. Contudo, a PLA não tem uma estrutura que facilite o uso de produtos de somas.

Por isso recorremos a um estratagema, que é o de obter a expressão da negação de $F2$ e, depois, usar a possibilidade oferecida pela PLA para negar essa negação, obtendo-se a função $F2$ pretendida. Nesse caso, o mapa de Karnaugh para $F2$ será o que se ilustra na Figura 18.9.

E a expressão obtida para o complemento de $F2$ é

$$\overline{F2} = \overline{B}\,C + C\,\overline{D} + A\,\overline{B}\,\overline{D}\,.$$

Repare-se que um dos produtos necessários, o produto $A\,\overline{B}\,\overline{D}$, já foi programado na PLA por causa de $F1$. O resultado final será, então, o que se ilustra na Figura 18.10.

As PLAs comerciais possuem, naturalmente, tamanhos diversos e diferentes do da PLA da Figura 18.5. Um exemplo é o da PLS100/101 da Philips. Trata-se de uma PLA com 16 entradas, 48 ANDs e 8 ORs.

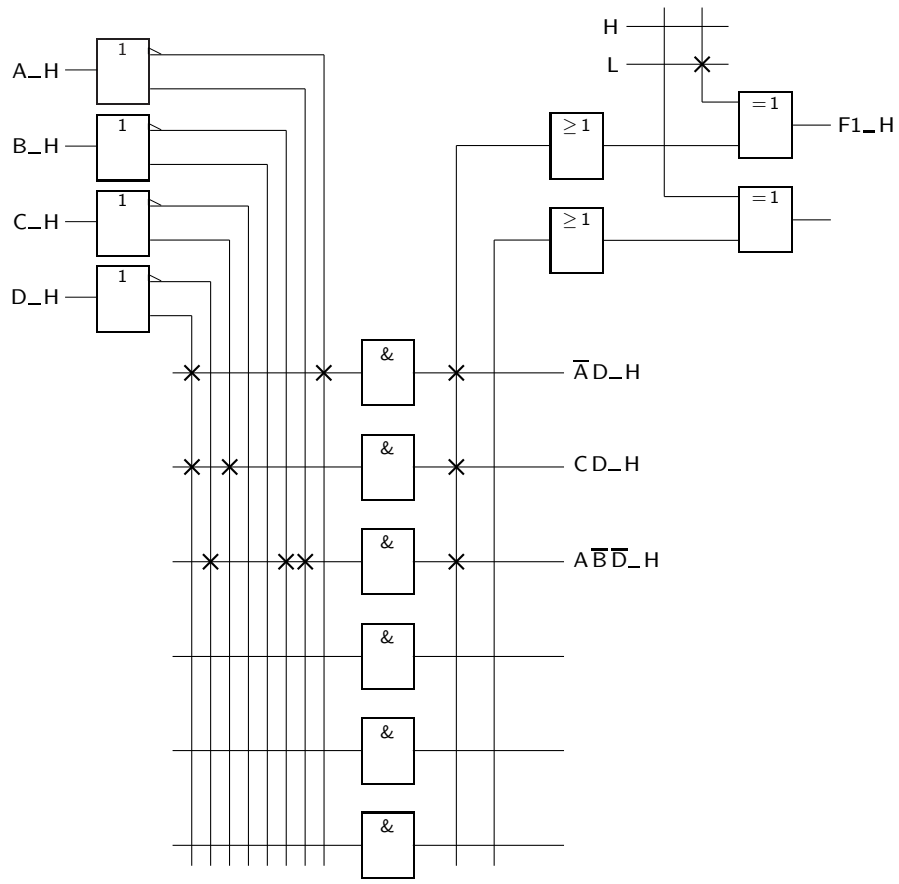


Figura 18.8: Uma PLA como a da Figura 18.5 permite implementar directamente a função $F1$ da Tabela 18.2

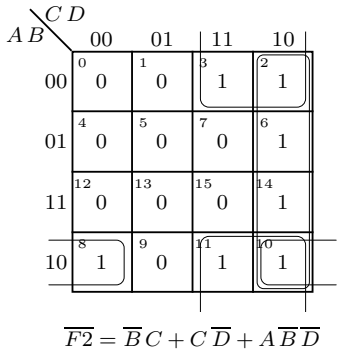


Figura 18.9: Mapa de Karnaugh para a função $\overline{F2}$ da Tabela 18.2

18.3 "Programmable Array Logic" (PALs)

No caso das ROMs, como se viu, as ligações dos ANDs estão fixas e é possível programar as ligações dos ORs. No caso das PLAs é possível programar ambas

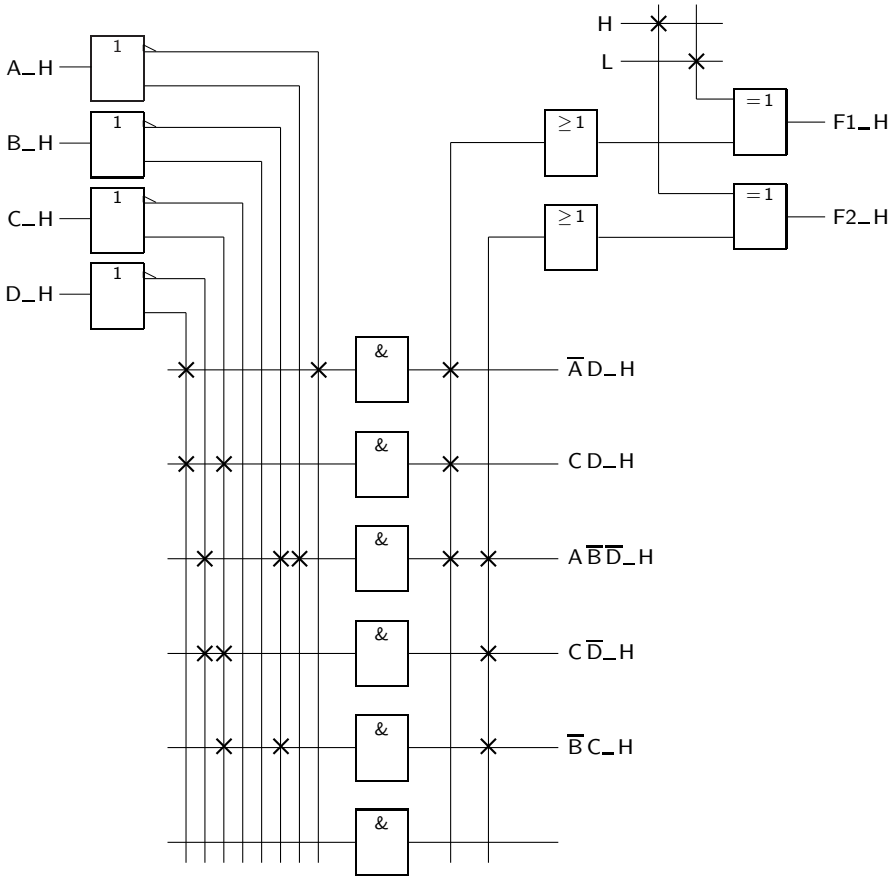


Figura 18.10: A PLA da Figura 18.5 permite implementar directamente a função $F1$ e, indirectamente, também a função $F2$ da Tabela 18.2

as ligações.

Verificou-se, contudo, que as potencialidades mais interessantes advinham da capacidade de programar os ANDs. Assim, desenvolveu-se um novo tipo de dispositivo em que as ligações entre os ANDs e os ORs estão fixas e apenas é possível programar as ligações dos ANDs às entradas, como mostra a Figura 18.11.

"Programmable Logic Array" (PLA)

A essa estrutura passou a chamar-se **PAL**, ou **"Programmable Array Logic"**.

Também uma PAL apresenta restrições quanto às expressões booleanas das q funções a implementar, já que cada uma delas deve ser representada em soma de produtos, e o número de implicantes da soma não pode exceder p por função (por comparação, numa PLA o número de implicantes disponíveis é igual a p para todas as funções).

Um exemplo de uma possível PAL, com 4 entradas, 12 ANDs e 4 saídas, é o que se apresenta na Figura 18.12.

Repare-se que uma das linhas de saida é realimentada para o interior da PAL (acontece isto com alguma frequência) para permitir construir funções que necessitem de um maior número de ANDs.

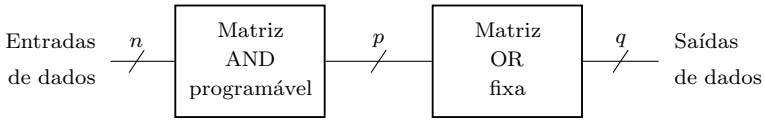


Figura 18.11: Uma PAL pode ser encarada como uma matriz de ANDs programáveis seguida de uma matriz de ORs fixos (não programáveis)

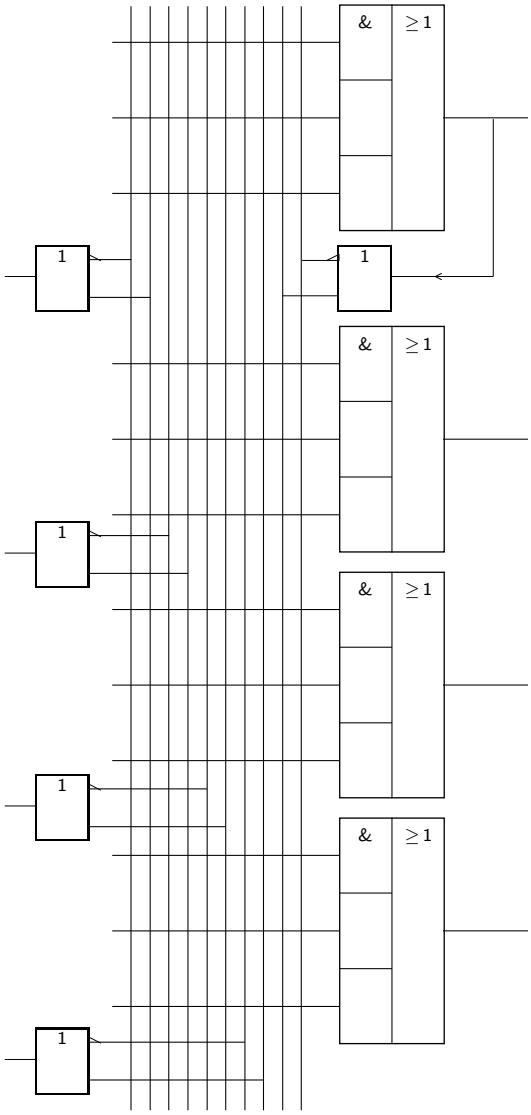


Figura 18.12: Estrutura de uma PAL com $n = 4$ entradas, $p = 12$ ANDs programáveis, e $q = 4$ saídas

O exemplo seguinte, para além de ilustrar a programação desta PAL, exemplifica também este aspecto.

Admitamos, então, que se pretende programar na PAL a seguinte função booleana geral:

$$\begin{aligned} W &= A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C\,\overline{D} \\ X &= A + B\,C\,D \\ Y &= \overline{A}\,B + C\,D + \overline{B}\,\overline{D} \\ Z &= A\,B\,\overline{C} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D \\ &= W + A\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D\,. \end{aligned} \tag{18.1}$$

O resultado da programação será, então, o que se indica na Figura 18.13.

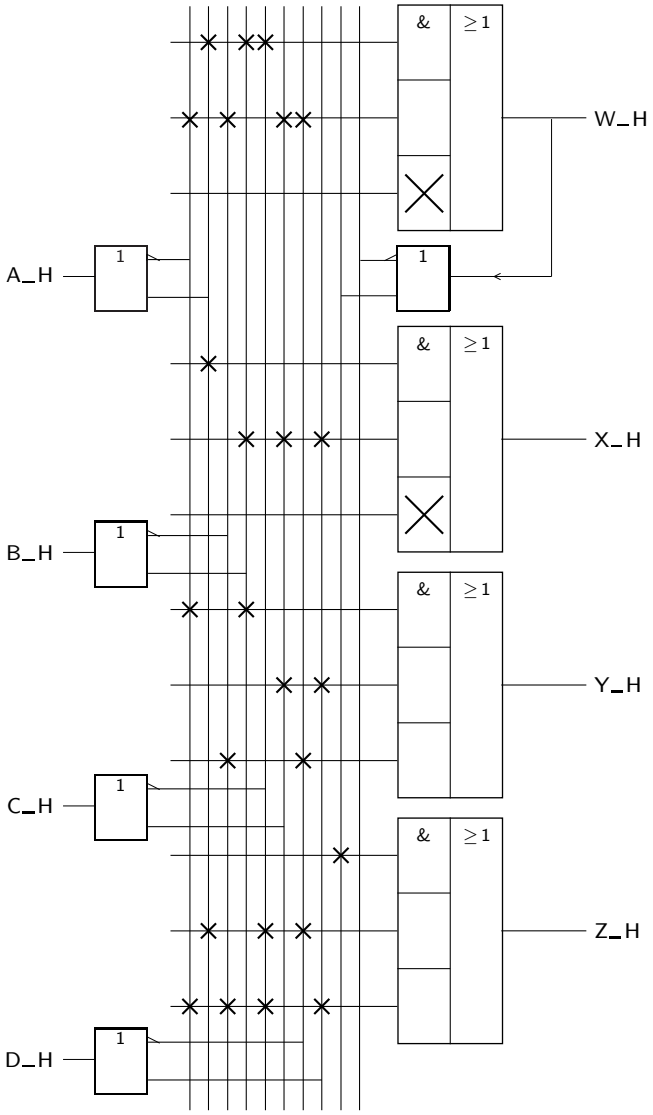


Figura 18.13: A PAL da Figura 18.12 é aqui utilizada para programar a função booleana geral (18.1)

Este tipo de dispositivos pode também incluir flip-flops nas saidas, de modo a

permitir realizar circuitos sequenciais.

As folhas de especificação de um conjunto de dispositivos reais que existem disponíveis no mercado dão exemplos dos dois tipos de PAL. A 16L8 é um caso de uma PAL puramente combinatória, do tipo da ilustrada anteriormente. As 16R4, 16R6 e 16R8 são PALs com uma componente sequencial formada por um registo com saídas "tri-state" controladas por uma entrada de Enable suplementar.

Capítulo 19

Máquinas de Estados

19.1 Circuito Controlado e Circuito de Controlo

Os sistemas digitais com alguma complexidade não podem ser projectados como vulgares máquinas sequenciais síncronas, porque os seus diagramas de estados, ou, em alternativa, as tabela de estados, são de grande dimensão, com um elevado número de entradas e de saídas, para além de um grande número de estados.

Prefere-se, então, organizar esses sistemas hierarquicamente, como foi sugerido no Capítulo 8, estabelecendo uma divisão clara entre o circuito que dá suporte ao fluxo e à manipulação de dados (o chamado **circuito controlado**, ou **circuito de dados**), por um lado, e um outro circuito que controla o primeiro (o **circuito de controlo**).

Circuito de controlo e circuito controlado (de dados)

O circuito controlado é, em geral, formado por um conjunto de módulos simples tais como contadores, registos, multiplexeres, somadores, comparadores, memórias, algumas portas lógicas, etc, podendo ser combinatório ou sequencial. Pelo contrário, o circuito de controlo é sempre um circuito sequencial síncrono.

Existem sistemas de controlo e sistemas controlados que são circuitos sequenciais assíncronos, mas não os estudamos nesta cadeira.

Projectar um sistema digital complexo consiste, então, em projectar o circuito de controlo e o circuito controlado correspondente.

Na Figura 19.1 apresenta-se um diagrama de blocos possível para este sistema hierárquico.

Contudo, o diagrama não é único. Por exemplo, se o sistema controlado for síncrono, precisa de uma ou mais entradas de relógio, que normalmente são obtidas de CLK_H ou da sua negação. Noutros sistemas, o sinal ou sinais de relógio do circuito controlado são gerados pelo circuito de controlo. Naturalmente, se o circuito controlado for combinatório, não necessita de entrada de relógio.

O circuito de controlo de um sistema complexo designa-se abstractamente por **máquina de estados** ou **máquina algorítmica**, mais simplesmente pela sigla **ASM**, que significa "Algorithmic State Machine".

Máquina de estados (algorítmica) ou ASM

O que distingue uma máquina de estados de uma vulgar máquina sequencial síncrona, como as que estudámos anteriormente, é a maneira como ela é pro-

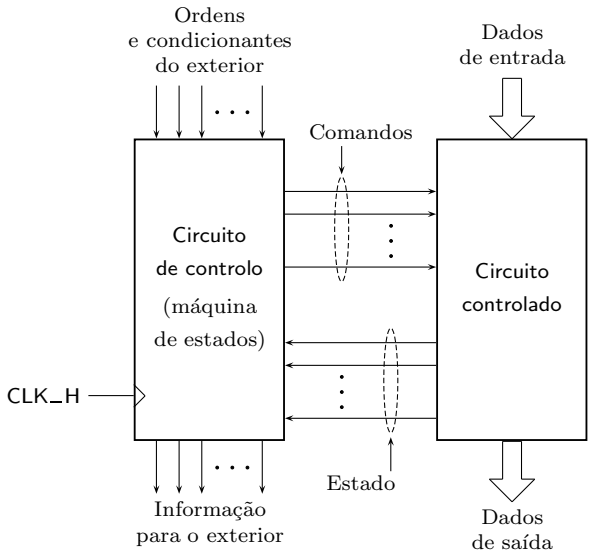


Figura 19.1: Diagrama de blocos de um sistema digital complexo formado por um circuito de controlo e por um circuito controlado

jectada para se inserir no sistemna global, controlando-o. Dado que o número de entradas e de saídas é, para uma máquina de estados, geralmente elevado, na prática somos forçados a usar fluxogramas na sua especificação e desenvolvimento.

Unidade de Controlo
Unidade Central de Processamento (CPU)

Um exemplo de máquina de estados é o da **Unidade de Controlo** de um (micro)processador, que comanda a **Unidade Central de Processamento** ou **CPU** ("Central Processing Unit").

Banco de registos ("Register File")
Memória Central

Na CPU são guardados, num **banco de registos** ou **"Register File"**, as instruções e os operandos que se obtêm da **Memória Central**, são efectuadas as operações aritméticas e lógicas sobre os conteúdos dos operandos, e os resultados das operações são guardados no banco de registo antes de os enviar à Memória Central. Finalmente, a CPU actualiza um contador especial, designado por **Contador de Programa**, que contém o endereço de memória onde se encontra a próxima instrução a ser operada.

Contador de Programa

Naturalmente, esta máquina de estados é complexa e difícil de projectar, até mesmo porque se pretende que ela seja extremamente rápida (vejam-se as elevadas frequências de relógio dos microprocessadores actuais). O projecto destas ASMs é, por essa razão, estudada em cadeiras de Arquitecturas de Computadores.

Neste capítulo vamos considerar um exemplo muito mais simples, de acesso a um parque de estacionamento típico. O acesso faz-se por uma via de sentido único, controlada na entrada e na saída pelas cancelas $C1$ a $C3$, pelos semáforos $S1$ a $S4$, e pelos sensores $D1$ a $D5$.

Como mostra a Figura 19.2, a máquina de estados há-de controlar o acesso ao parque lendo os níveis de tensão às saídas dos sensores e gerando os sinais de controlo das cancelas e dos semáforos, e os sinais de comando para o circuito

controlado. Por outro lado, o circuito controlado há-de indicar quando o parque se encontra cheio, enviando para o circuito de controlo um sinal de estado adequado.

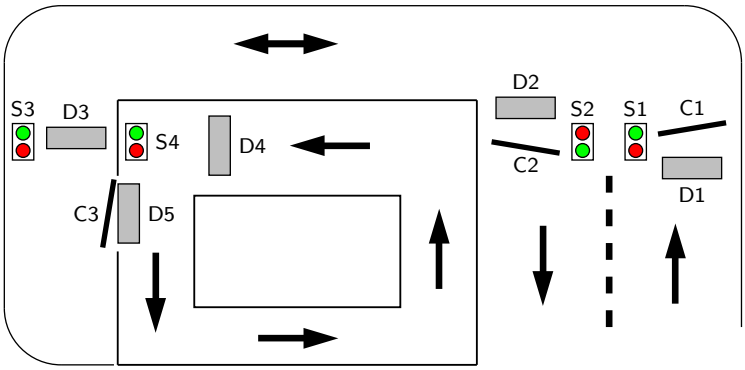


Figura 19.2: O controlo do acesso ao parque de estacionamento deverá abrir ou fechar as cancelas de acesso $C1$ a $C3$ — consoante os valores lidos nos sensores $D1$ a $D5$ — e controlar os semáforos $S1$ a $S4$, para além de gerar os sinais de comando (no caso, dois sinais de relógio) necessários ao funcionamento do circuito controlado, recolhendo deste o sinal de estado que indica quando o parque está cheio

Quanto ao circuito controlado, vai ser constituído por um contador ascendente/descendente, que guarda a informação sobre o número de carros estacionados no parque (Figura 19.3).

O contador possui duas entradas de relógio, uma designada por $CP.UP\_H$, que incrementa o contador de uma unidade, e outra designada por $CP.DOWN\_H$, que decrementa o contador de uma unidade (este é um exemplo em que os sinais de relógio necessários ao funcionamento do circuito controlado são gerados pelo circuito de controlo). Quando o contador atinge um valor de contagem igual à capacidade máxima do parque, gera um sinal de estado, $FULL\_H$, para o circuito de controlo.

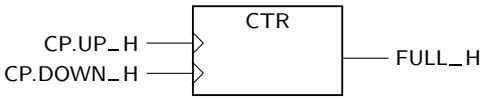


Figura 19.3: O circuito controlado é formado por um contador ascendente/descendente que indica, em cada instante, o número de carros estacionados no parque

O funcionamento do sistema de acesso ao parque é o que se descreve a seguir.

— Quando o parque está cheio, só podem sair carros. Quando não está cheio, podem entrar ou sair. Porque a rua de acesso é estreita, só pode passar um carro de cada vez.

— A saída é detectada pela presença de um carro que pisa $D4$. Se não há entrada em curso, o semáforo $S4$ fica verde e a cancela $C3$ abre. Em seguida

espera-se que o carro pise $D5$ e saia, para se fechar a cancela e colocar o semáforo $S4$ em vermelho. Entretanto, coloca-se o semáforo $S2$ a verde. Quando o carro pisa $D2$, abre-se $C2$, que se mantém aberta enquanto a viatura estiver a pisar $D2$. Quando o carro deixar de pisar $D2$, o semáforo $S2$ passa a vermelho e $C2$ fecha. Nessa altura desconta-se uma unidade no contador de lugares ocupados no parque.

— A entrada começa com um carro a pisar $D1$. Se não há saída em curso, o semáforo $S1$ fica verde e a cancela $C1$ abre, ficando aberta enquanto o carro é detectado por $D1$. Quando o carro deixa $D1$, $S3$ fica a verde, e quando chega a $D3$ a cancela $C3$ é aberta e o carro entra, passando $S3$ a vermelho e ficando o circuito à espera que $D5$ venha pisado (em rigor, podíamos passar sem $S3$. Porquê?). Só depois de $D5$ deixar de ser pisado é que $C3$ fecha. Nessa altura, o contador é incrementado.

Tendo em atenção o modelo geral da Figura 19.1, para este caso particular a situação é a que descreve na Figura 19.4.

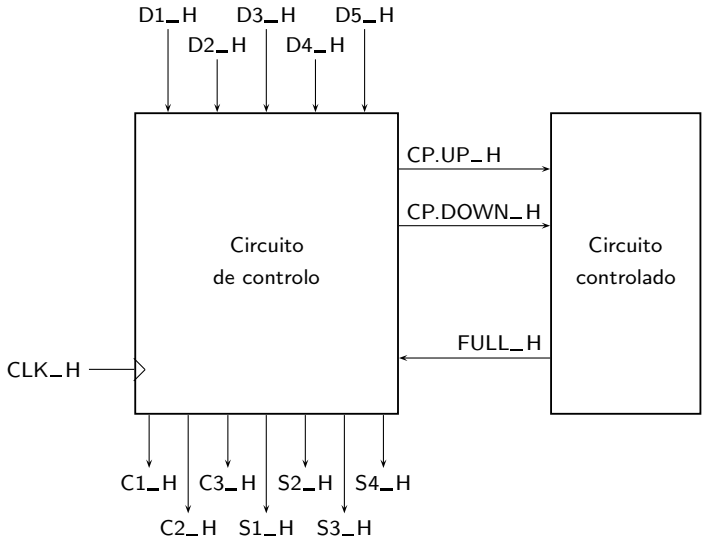


Figura 19.4: Modelo da Figura 19.1, adaptado ao controlo do acesso ao parque de estacionamento da Figura 19.2

19.2 Máquinas de Estados e Fluxogramas

Naturalmente, podia-se tentar obter em seguida um diagrama de estados para a máquina de estados que controla o acesso ao parque de estacionamento. Porém, como este circuito tem 6 entradas e 9 saídas, o diagrama ficaria muito confuso. Daí que se recorra a um fluxograma, por ser mais compacto. Esta é, aliás, a forma mais comum de especificar e documentar as ASMs, dadas as suas complexidades habituais.

O fluxograma de controlo de acesso ao parque de estacionamento encontra-se ilustrado nas Figuras 19.5 e 19.6, dada a sua complexidade.

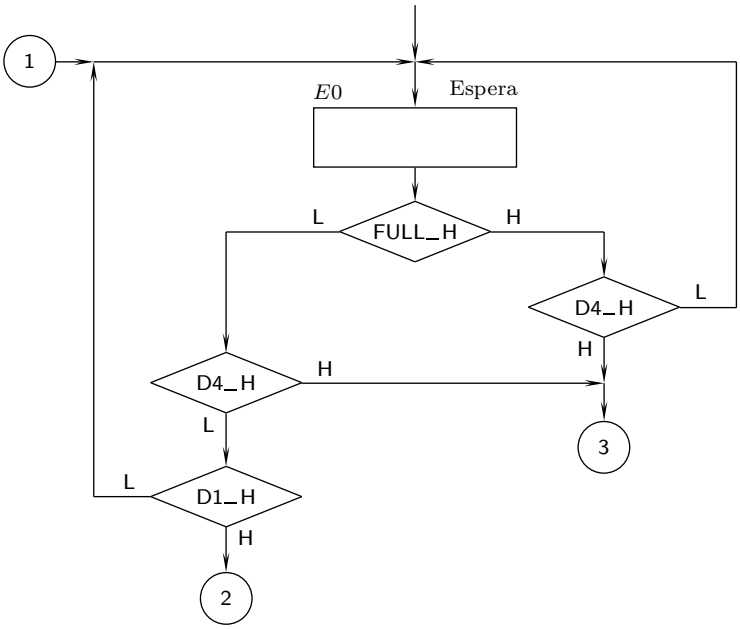


Figura 19.5: Parte inicial do fluxograma da ASM da Figura 19.4

Dado encontrar-se repartido por duas figuras, o fluxograma obriga à inclusão de indicadores de mudança de figura, representados por círculos numerados.

A explicação do fluxograma é óbvia, pelo que dispensamos os respectivos comentários.

Frisa-se, contudo, que se utilizaram as mesmas convenções que foram usadas na Secção 16.9 a propósito das Figuras 16.37 a 16.39 (páginas 310 a 312):

1. admite-se que todas as entradas e saídas são activas a H;
2. admite-se que a activação de uma cancela significa levantá-la; e
3. admite-se que a activação de um semáforo significa colocá-lo a verde.

19.3 Implementação com ROMs

Como se viu até agora, existem dois problemas sérios que dificultam a implementação de circuitos sequenciais com um elevado número de estados, de entradas e de saídas, como acontece com as máquinas de estados.

Por um lado, necessita-se de uma grande quantidade de lógica diversa para gerar as excitações dos flip-flops e as saídas dos circuitos. E, por outro, é em geral difícil sintetisar o circuito, dada a dimensão do fluxograma (ou diagrama de estados, ou tabela de estados) a que muitas vezes se chega.

O último aspecto foi facilitado com a síntese com um flip-flop por estado, muito utilizada nas implementações com dispositivos lógicos programáveis devido à

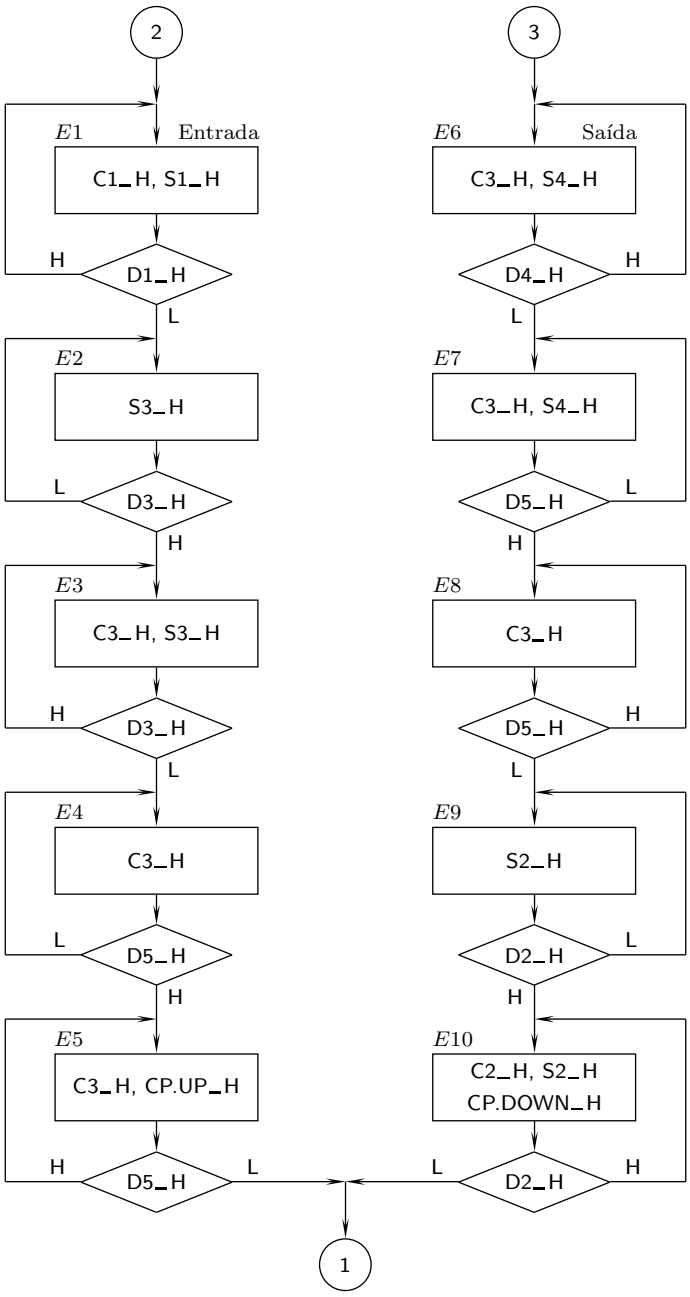


Figura 19.6: Continuação do fluxograma da Figura 19.5, correspondente às situações de entrada ou de saída de um carro no parque de estacionamento

existência de programas que geram automaticamente soluções prontas a serem transferidas para alguns desses dispositivos, nomeadamente CPLDs e FPGAs.

Naturalmente, existem sempre limitações quanto às dimensões desses dispositivos, pelo que se procurou obter outro modo de implementação de circuitos de controlo que usasse pouca lógica discreta e que permitisse a implementação de

muitos estados, entradas e saídas.

Controlo microprogramado (microprogramação)

Processadores CISC

Microprocessadores RISC

O uso de ROMs permitiu caminhar nesse sentido. Permitiu mesmo dar um passo muito significativo na evolução dos circuitos de controlo complexos, com o aparecimento do **controlo microprogramado** ou **microprogramação**, característico dos **processadores CISC** das ´´mainframes" das décadas de 60 a 80.

O aparecimento dos modernos **microprocessadores RISC** fez perder importância a este tipo de controlo, que é lento se precisarmos de usar ROMs de grandes dimensões, em tecnologia nMOS. A alternativa consistiu em implementações com PALs ou PLAs (muito rápidas) e sínteses com um flip-flop por estado, gerando implementações de dimensões muito consideráveis.

Nesta secção iremos estudar as implementações por ROM, nas suas versões mais simples. Contudo, deve ficar claro que não vamos abordar os problemas da microprogramação que, por serem muito variados e complexos, deixaremos para uma cadeira posterior de Arquitectura de Computadores.

Limitar-nos-emos, assim, a abordar os princípios básicos de implementação de máquinas de estado usando um controlo por ROM, em diversas versões sucessivamente mais complexas, sendo que a última contém já os fundamentos do controlo microprogramado.

19.3.1 Estrutura básica de controlo por ROM

Uma máquina de estados pode ser implementada com uma estrutura suportada numa ROM, como a que vem ilustrada na Figura 19.7.

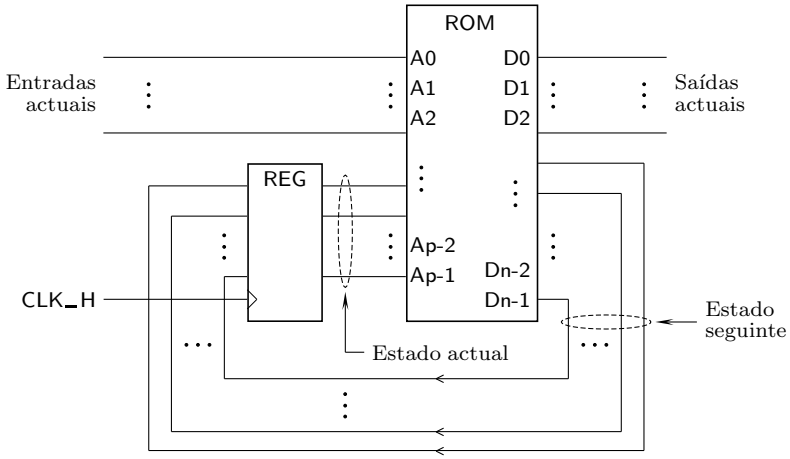


Figura 19.7: Diagrama de blocos de uma máquina de estados com controlo por ROM que utiliza uma estrutura básica

Se compararmos este diagrama de blocos com o da Figura 16.4, na página 283, compreendemos que a ROM que agora vamos usar substitui a lógica do estado seguinte e a lógica de saída do circuito.

Se, por outro lado, compararmos este diagrama de blocos com o da Figura 19.1, verificamos que as entradas da Figura 19.7, sendo, naturalmente, as entradas externas da máquina de estados, são agora compostas pelas entradas externas do

circuito de controlo (as ordens e condicionantes da Figura 19.1) e pelas entradas de estado que provêm do circuito controlado.

Identicamente, as saídas da Figura 19.7, isto é, as saídas externas à máquina de estados, são compostas pelas saídas para o exterior do circuito de controlo (a informação para o exterior) e pelas saídas (comandos) para o circuito controlado.

Como podemos constatar no diagrama de blocos anterior, as entradas da máquina e os saídas dos flip-flops (o seu estado actual, EA) constituem o barramento de endereços da ROM, A_0 a A_{p-1}.

Quanto ao barramento de dados, D_0 a D_{n-1}, está dividido em duas partes: (i) as entradas de excitação dos flip-flops, que decidem o estado seguinte, ES, da máquina; e (ii) as saídas para o exterior da máquina de estados.

Quando, num período de relógio t, aplicamos ao barramento de endereços um determinado endereço, isso significa que a máquina se encontra num determinado estado (actual, no perído t) e tem certos níveis de tensão aplicados às entradas (actuais). O conteúdo da palavra que é lida da ROM nesse endereço, e que foi previamente programado, identifica as saída (actuais) a gerar para o exterior da máquina, e o estado (seguinte, no período de relógio $t + 1$) para o qual a máquina irá evoluir.

Campos

Tudo se passa como se a palavra lida da ROM (o conteúdo desse endereço) se encontrasse formatada em dois **campos**, um para o estado seguinte e outro para as saídas actuais, como mostra a Figura 19.8.

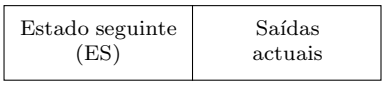


Figura 19.8: Formato das palavras quando se usa uma estrutura básica de controlo por ROM

Para entendermos como programar a ROM para o correcto funcionamento deste tipo de controlo, consideremos na Figura 19.9 um fluxograma muito simples de uma máquina de Mealy.

Neste caso temos:

— 3 estados, A, B e C; podemos, portanto, usar 2 variáveis de estado, digamos $Q1\_H$ e $Q0\_H$;

— duas variáveis de entrada, $I0\_H$ e $I1\_H$; e

— duas variáveis de saída: X_H, de Mealy, gerada no estado A se a entrada $I1$ estiver inactiva, e Y_H, de Moore, gerada no estado C.

O diagrama de blocos da Figura 19.7 fica, com esta máquina, estruturado como se indica na Figura 19.10.

O formato de cada palavra de ROM é o que se indica na Figura 19.11.

A codificação dos estados é arbitrária, pelo que usaremos a atribuição de variáveis de estado que consta da Tabela 19.1. Porque vamos trabalhar no contexto

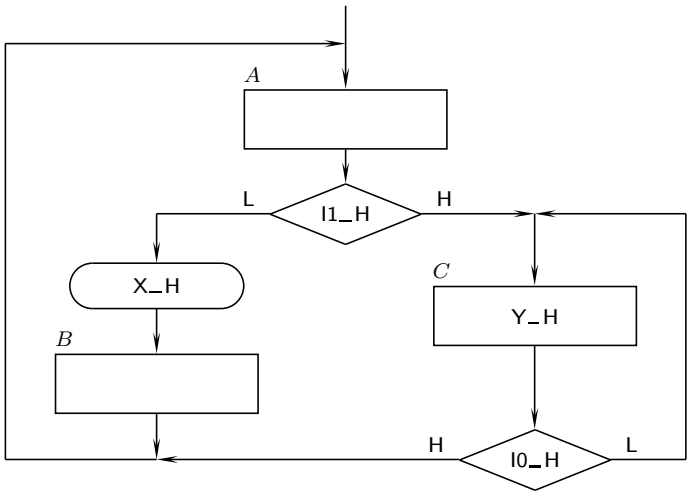


Figura 19.9: Fluxograma de uma máquina de estados de Mealy muito simples que serve de exemplo para uma implementação por ROM, na sua estrutura básica

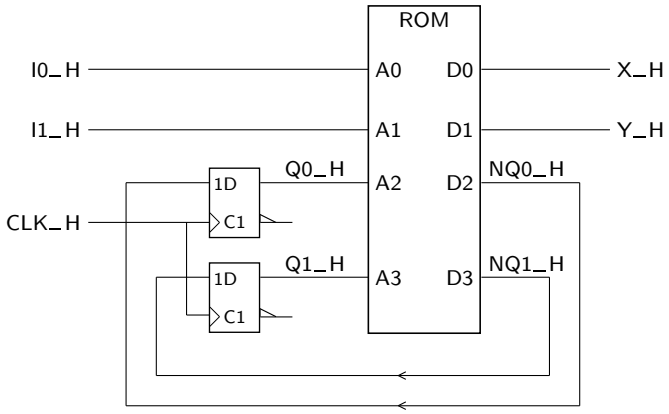


Figura 19.10: Diagrama de blocos com a estrutura básica de controlo por ROM para a máquina de estados da Figura 19.9

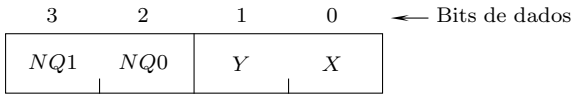


Figura 19.11: Formato das palavras quando se usa um controlo por ROM básico na implementação da máquina de estados da Figura 19.9

algébrico quando lidamos com as ROMs e com os seus conteúdos, todas as tabelas que iremos usar neste capítulo envolvem exclusivamente "0"s e "1"s.

O conteúdo da ROM vai ser definido por uma tabela de estados lógica como a da Tabela 19.2.

Como se observou anteriormente, os conteúdos das palavras de ROM contêm

Tabela 19.1: Tabela com a codificação de estados para o circuito de controlo da Figura 19.9, com controlo por ROM básico

Estado actual	$Q1\_H_{(t)}$	$Q0\_H_{(t)}$
A	0	0
B	0	1
C	1	0

Tabela 19.2: Tabela de estados lógica com o conteúdo da ROM a programar para a máquina de estados da Figura 19.9, quando se utiliza uma estrutura básica de controlo

Estado	**Endereço**	Q1 A3	Q0 A2	I1 A1	I0 A0	NQ1 D3	NQ0 D2	Y D1	X D0
A	0	0	0	0	0	0	1	0	1
A	1	0	0	0	1	0	1	0	1
A	2	0	0	1	0	1	0	0	0
A	3	0	0	1	1	1	0	0	0
B	4	0	1	0	0	0	0	0	0
B	5	0	1	0	1	0	0	0	0
B	6	0	1	1	0	0	0	0	0
B	7	0	1	1	1	0	0	0	0
C	8	1	0	0	0	1	0	1	0
C	9	1	0	0	1	0	0	1	0
C	10	1	0	1	0	1	0	1	0
C	11	1	0	1	1	0	0	1	0
	12	1	1	0	0	×	×	×	×
	13	1	1	0	1	×	×	×	×
	14	1	1	1	0	×	×	×	×
	15	1	1	1	1	×	×	×	×

"0"s e "1"s (em vez de "H"s e de "L"s) porque estamos a trabalhar no contexto algébrico com uma tabela de estados lógica.

É de notar ainda que os 16 endereços de ROM (porque existem 4 linhas de endereço) são repartidos por 4 zonas com 4 endereços cada uma: uma zona para o estado A, outra para o estado B, outra para o estado C e, finalmente, uma quarta zona que não é utilizada. Nesta última zona colocaram-se indiferenças nos conteúdos das palavras (porque não são utilizados), embora *a programação da ROM obrigue a especificar "0"s e "1"s. Isto é, a programação da ROM exige a colocação de "0"s ou de "1"s no lugar das indiferenças.*

Vejamos como se constrói uma linha qualquer da ROM, por exemplo a linha com endereço 0.

Essa linha corresponde ao estado actual A (porque $Q1 = Q0 = 0$) com as entradas actuais $I1 = I0 = 0$. Como $I1 = 0$ neste estado, queremos ir para o estado B, pelo que $NQ1 = 0$ e $NQ0 = 1$. Por outro lado, no estado A queremos activar X e desactivar Y, pelo que programamos $X = 1$ e $Y = 0$.

19.3.2 Controlo por ROM com endereçamento explícito

Para construir o circuito correspondente ao exemplo do parque de estacionamento com a metodologia anterior, a ROM deveria ter dimensões consideráveis:

— 10 linhas de endereço, sendo 6 correspondentes às entradas — 5 externas ao sistema e uma de estado — e 4 correspondentes às variáveis de estado; e

— 12 linhas de saída, sendo 4 no campo "Estado seguinte" e 9 no campo "Saídas" — 7 para o exterior do sistema e duas de comandos para o circuito controlado.

Tratar-se-ia de uma ROM com 1 024 palavras de 12 bits, o que, não levantando qualquer problema do ponto de vista prático, exige uma ROM já com uma dimensão considerável.

Porém, na maior parte dos circuitos reais, com um elevado número de variáveis de entrada, de saída e de estado, esta metodologia pode ficar comprometida.

Felizmente, há alternativas. Na primeira, que iremos estudar em seguida, as variáveis de entrada são retiradas do barramento de endereços da ROM, o que permite diminuir consideravelmente o número de palavras — relembremos que cada linha de endereço a menos numa memória significa a redução do número de palavras para metade.

Como não se pode prescindir da influência das variáveis de entrada no funcionamento da máquina de estados, algo tem que ser feito. A ideia básica consiste em:

1. eliminar as suas acções nas saídas, transformando as saídas de Mealy em saídas de Moore; e
2. limitar as acções das variáveis de entrada nas mudanças de estado, por forma a que, de cada estado actual, só se possa evoluir para um de dois estados seguintes (incluindo, eventualmente, o próprio).

Analisemos a estrutura e dois exemplos. O novo diagrama de blocos é agora o que se ilustra na Figura 19.12.

Notemos como cada palavra da ROM vem agora formatada em quatro campos, sendo dois para o estado seguinte (ES0 e ES1), um para o teste das entradas, e um quarto campo para as saídas actuais (Figura 19.13).

Porque cada palavra de ROM indica explicitamente dois estados seguintes em

Endereçamento explícito

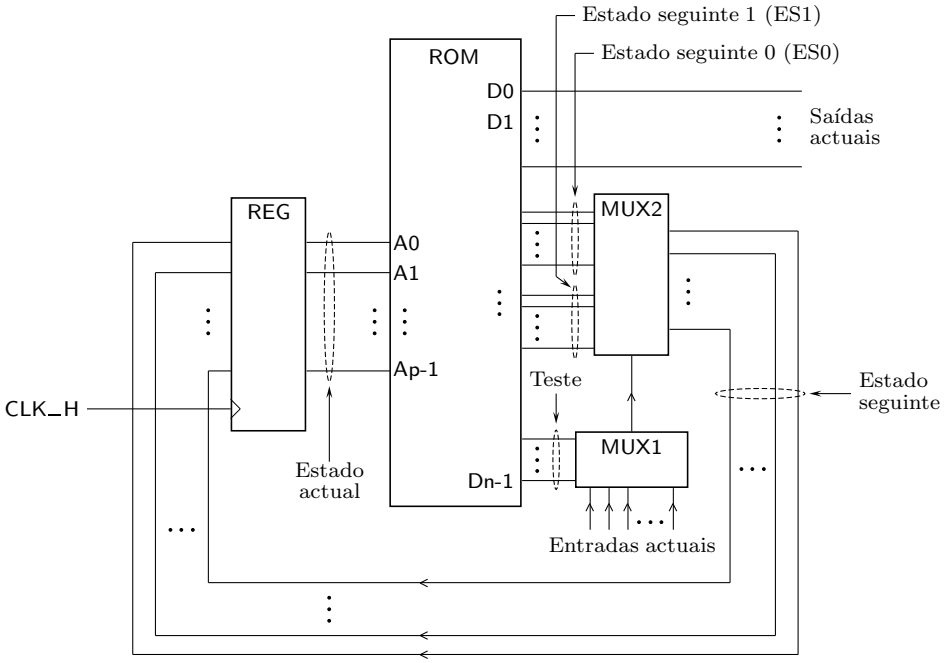


Figura 19.12: Diagrama de blocos de uma máquina de estados que utiliza um controlo por ROM com endereçamento explícito

Teste	ES1	ES0	Saídas actuais

Figura 19.13: Formato das palavras de ROM quando se usa um controlo com endereçamento explícito

alternativa, esta estrutura é designada por **controlo por ROM com endereçamento explícito**.

O MUX1 tem as suas linhas de entradas de dados ligadas às linhas de entrada do circuito de controlo. As linhas do campo de teste da ROM permitem, para cada estado actual, escolher a entrada ou combinações de entradas a testar. Se a entrada seleccionada tiver o valor 0, por exemplo, então o estado seguinte escolhido é o que vier indicado no campo ES0. No caso contrário, será o estado seguinte ES1 a ser escolhido.

Notemos que, desta forma, podemos facilmente implementar toda e qualquer transição condicionada que encontremos num fluxograma. As transições incondicionais, que não envolvem testes a variáveis de entrada, são igualmente fáceis de implementar: basta que programemos o mesmo estado seguinte nos campos ES0 e ES1.

Voltemos, então, à nossa máquina de estados simples, com o fluxograma da Figura 19.9. Para utilizar este tipo de estrutura, o fluxograma vai ter que ser transformado. Se tal não fôr possível, então será necessária alguma forma de lógica adicional.

Neste caso a saída X é de Mealy, pelo que precisamos de transformar a máquina de estados numa máquina de Moore.

Por outro lado, de cada estado actual só se vai, no máximo, para dois estados seguintes, pelo que, a esse nível, não são necessárias alterações.

Para passar a saída X a saída de Moore, temos duas hipóteses: (i) inclui-se X no estado B; ou (ii) acrescenta-se um estado antes de B. Em qualquer dos casos X vem activada mais tarde do que previsto, dependendo a escolha das temporizações específicas da aplicação. Vamos admitir que X pode ser incluída no estado B, obtendo-se o fluxograma da Figura 19.14.

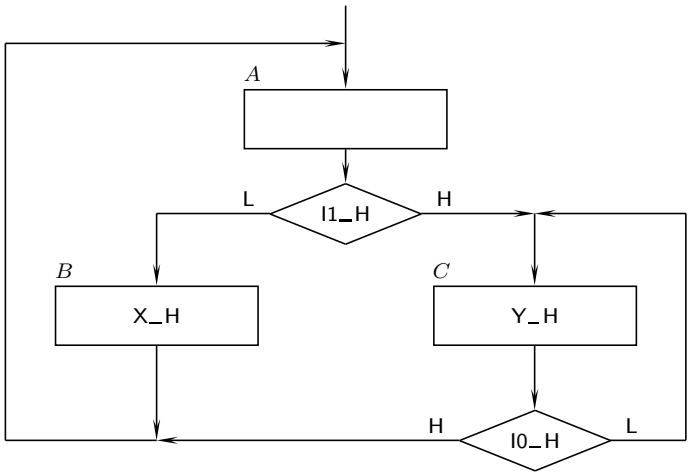


Figura 19.14: A máquina de estados de Mealy da Figura 19.9 tem de ser transformada numa máquina de Moore para se poder utilizar um controlo por ROM com endereçamento explícito. Neste caso, admitiu-se que a saída X podia vir activada no estado B

Nessa hipótese, o diagrama de blocos da máquina de estados será o que se ilustra na Figura 19.15.

A ROM passou a ser constituída por 4 palavras de 7 bits, em vez das 16 palavras de 4 bits necessárias para a estrutura básica de controlo.

De notar que as linhas $NQ \times 0$ são os bits de estado seguinte 0 (ES0) quando o resultado da variável testada der 0, e as linhas $NQ \times 1$ são os bits de estado seguinte 1 (ES1) quando a variável testada der 1.

O formato de cada palavra de ROM é o que se indica na Figura 19.16.

Para a mesma codificação de estados da Tabela 19.1, o conteúdo da ROM é, com esta estrutura, a que se indica na Tabela 19.3.

Em relação a esta tabela, tecem-se alguns comentários:

1. como se afirmou anteriormente, as linhas $NQ00$ e $NQ10$ são os bits de ES0 quando o resultado da variável testada for 0, e as linhas $NQ01$ e $NQ11$ são os bits de ES1 quando a variável testada tiver o valor 1; e
2. na linha correspondente ao estado A testa-se a entrada $I1$ — trata-se de uma transição condicionada ao valor de $I1$ no estado A;

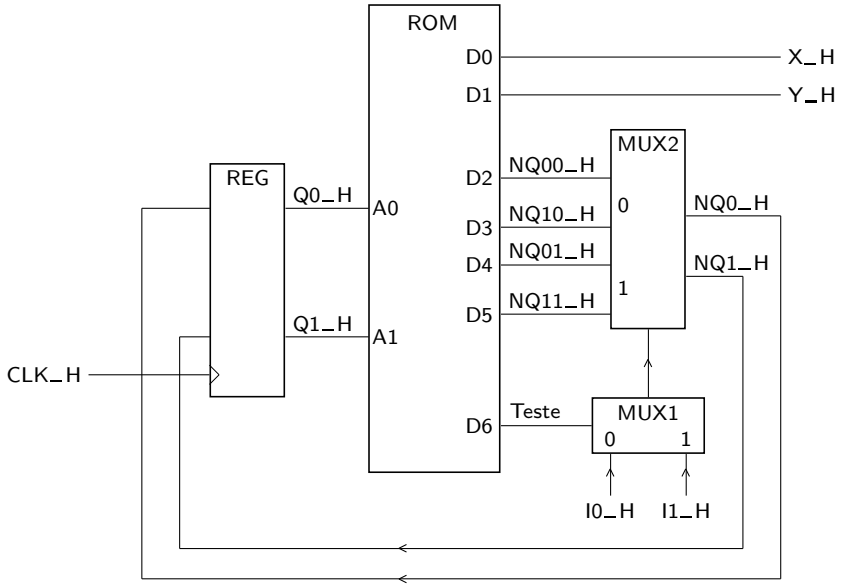


Figura 19.15: Diagrama de blocos para a máquina de estados da Figura 19.14, com controlo por ROM com endereçamento explícito

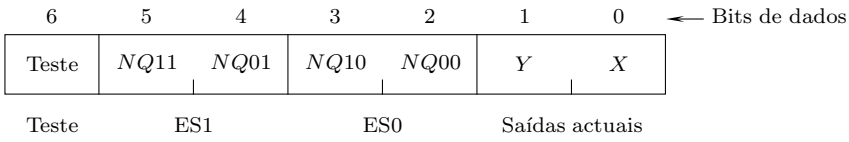


Figura 19.16: Formato das palavras num controlo por ROM com endereçamento explícito para a máquina de estados da Figura 19.14

Tabela 19.3: Tabela de estados lógica com o conteúdo da ROM a programar para a máquina de estados da Figura 19.14, quando se utiliza uma estrutura de controlo com endereçamento explícito

	Q1	Q0	Teste	NQ11	NQ01	NQ10	NQ00	Y	X
Estado	A1	A0	D6	D5	D4	D3	D2	D1	D0
A	0	0	1	1	0	0	1	0	0
B	0	1	×	0	0	0	0	0	1
C	1	0	0	0	0	1	0	1	0
	1	1	×	×	×	×	×	×	×

3. na linha correspondente ao estado B não é feito qualquer teste, pelo que é indiferente o valor a inserir no campo com o mesmo nome (contudo, ver a observação anterior que afirma que não podemos programar indiferenças na tabela);

4. a transição a partir do estado B é incondicional, pelo que os dois campos de estado seguinte apontam ambos para o estado A;

5. finalmente, no estado C testa-se $I0$ (outra transição condicionada, desta feita ao valor de $I0$).

Consideremos agora o caso do parque de estacionamento e os fluxogramas das Figuras 19.5 e 19.6.

Neste caso não é preciso alterar as saídas, porque são todas de Moore. Mas é preciso acrescentar alguns estados para garantir que, de qualquer um estado actual, apenas se prossegue para um de dois estados seguintes, no máximo.

Tal torna-se necessário apenas nas transições que partem do estado $E0$, pelo que apenas a parte do fluxograma da Figura 19.5 precisa de ser redesenhada, como se ilustra na Figuras 19.17.

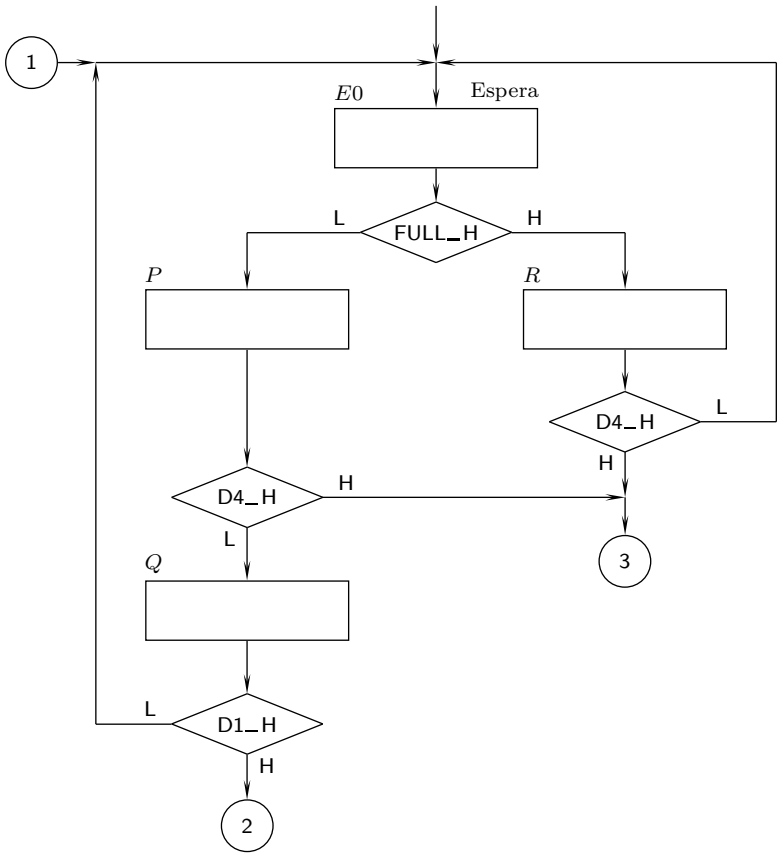


Figura 19.17: Parte inicial do fluxograma das Figuras 19.5 e 19.6, modificado para permitir um controlo por ROM com endereçamento explícito

Por comodidade, reproduz-se na Figura 19.18 o resto do fluxograma de controlo.

Agora vamos necessitar de 4 flip-flops, porque precisamos de codificar 14 estados. Sendo indiferente a codificação das variáveis de estado, podemos usar a codificação da Tabela 19.4.

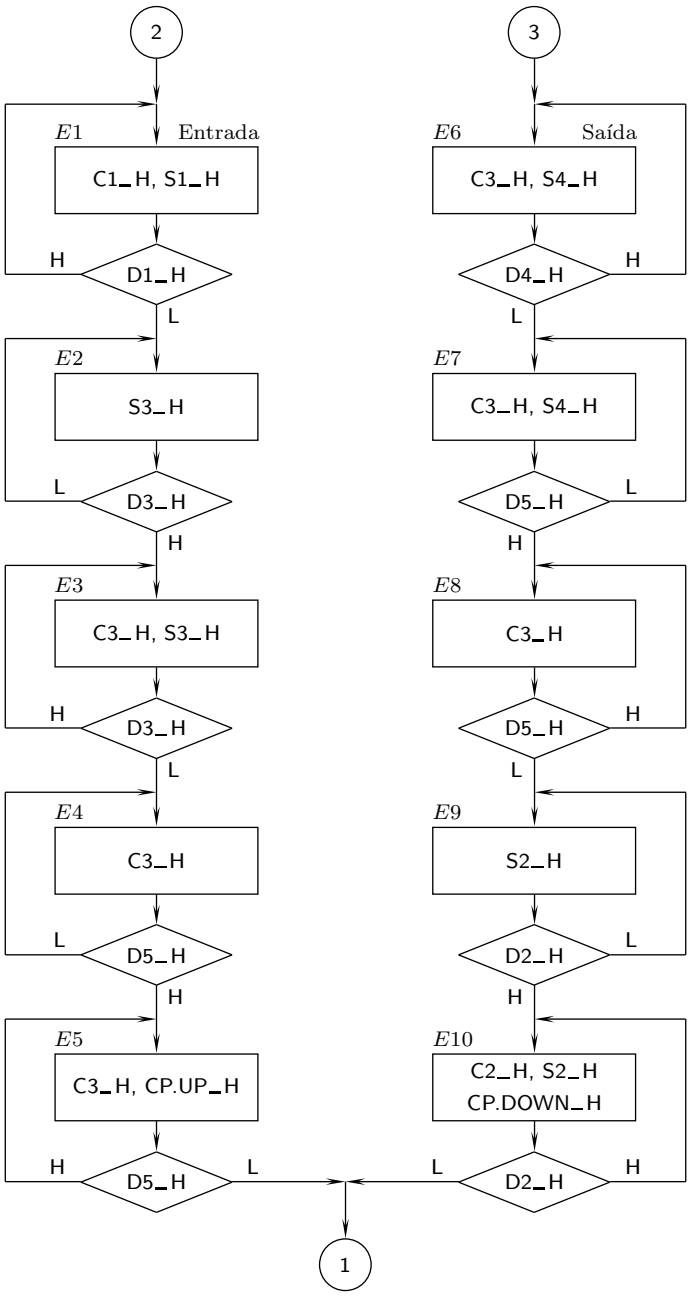


Figura 19.18: Continuação do fluxograma da Figura 19.17

O diagrama de blocos da máquina de estados será, por sua vez, o que se indica na Figura 19.19.

De notar que as linhas $NQ00\_H$ a $NQ30\_H$ designam os bits do estado seguinte 0 (ES0) quando o resultado da variável testada for o valor 0, e que as linhas $NQ01\_H$ a $NQ31\_H$ designam os bits do estado seguinte 1 (ES1) quando a variável testada tiver o valor 1.

Tabela 19.4: Tabela de codificação de estados para a máquina de estados das Figuras 19.17 e 19.18, com controlo por ROM com endereçamento explícito

Estado	Q3	Q2	Q1	Q0
$E0$	0	0	0	0
$E1$	0	0	0	1
$E2$	0	0	1	0
$E3$	0	0	1	1
$E4$	0	1	0	0
$E5$	0	1	0	1
$E6$	0	1	1	0
$E7$	0	1	1	1
$E8$	1	0	0	0
$E9$	1	0	0	1
$E10$	1	0	1	0
P	1	0	1	1
Q	1	1	0	0
R	1	1	0	1
	1	1	1	0
	1	1	1	1

A ROM terá, agora 16 palavras de 20 bits, com o formato que se ilustra na Figura 19.20.

Utilizando a codificação de estados da Tabela 19.4, o conteúdo da ROM para esta máquina de estados é, com uma arquitectura com endereçamento explícito, a que se indica na Tabela 19.5.

Por exemplo, a primeira linha, com endereço 0 e com

$$(Q3, Q2, Q1, Q0) = (0, 0, 0, 0)\,,$$

corresponde ao estado $E0$, de acordo com a codificação de estados da Tabela 19.4.

Ora, neste estado testa-se a variável de entrada $FULL$, que foi ligada à entrada 0 do MUX1 do diagrama de blocos da Figura 19.19. Segue-se que

$$(T2, T1, T0) = (0, 0, 0)$$

nesta linha da ROM.

Se $FULL = 1$ deve escolher-se o estado seguinte 1 (isto é, ES1), que se encontra codificado nos bits $D16$ a $D13$ da palavra da ROM com as designações $NQ31$ a $NQ01$, respectivamente. Nesta situação (com $FULL = 1$), vai-se do estado actual $E0$ para o estado seguinte R, de acordo com o fluxograma da Figura 19.19. Ora o estado R foi codificado com

$$(Q3, Q2, Q1, Q0) = (1, 1, 0, 1)\,,$$

Tabela 19.5: Tabela de estados lógica com o conteúdo da ROM a programar para a máquina de estados das Figuras 19.17 e 19.18, quando se utiliza uma estrutura de controlo com endereçamento explícito

					Teste			**ES1**				**ES0**				**Saídas actuais**								
	Q3	Q2	Q1	Q0	T2	T1	T0	NQ31	NQ21	NQ11	NQ01	NQ30	NQ20	NQ10	NQ00	CP.DOWN	CP.UP	S4	S3	S2	S1	C3	C2	C1
Estado	A3	A2	A1	A0	D19	D18	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
$E0$	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0
$E1$	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1
$E2$	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0
$E3$	0	0	1	1	0	1	1	0	0	1	1	0	1	0	0	0	0	0	1	0	0	1	0	0
$E4$	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0
$E5$	0	1	0	1	1	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
$E6$	0	1	1	0	1	0	0	0	1	1	0	0	1	1	1	0	0	1	0	0	0	1	0	0
$E7$	0	1	1	1	1	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0	0
$E8$	1	0	0	0	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$E9$	1	0	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0
$E10$	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0
P	1	0	1	1	1	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
Q	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
R	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
–	1	1	1	0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
–	1	1	1	1	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

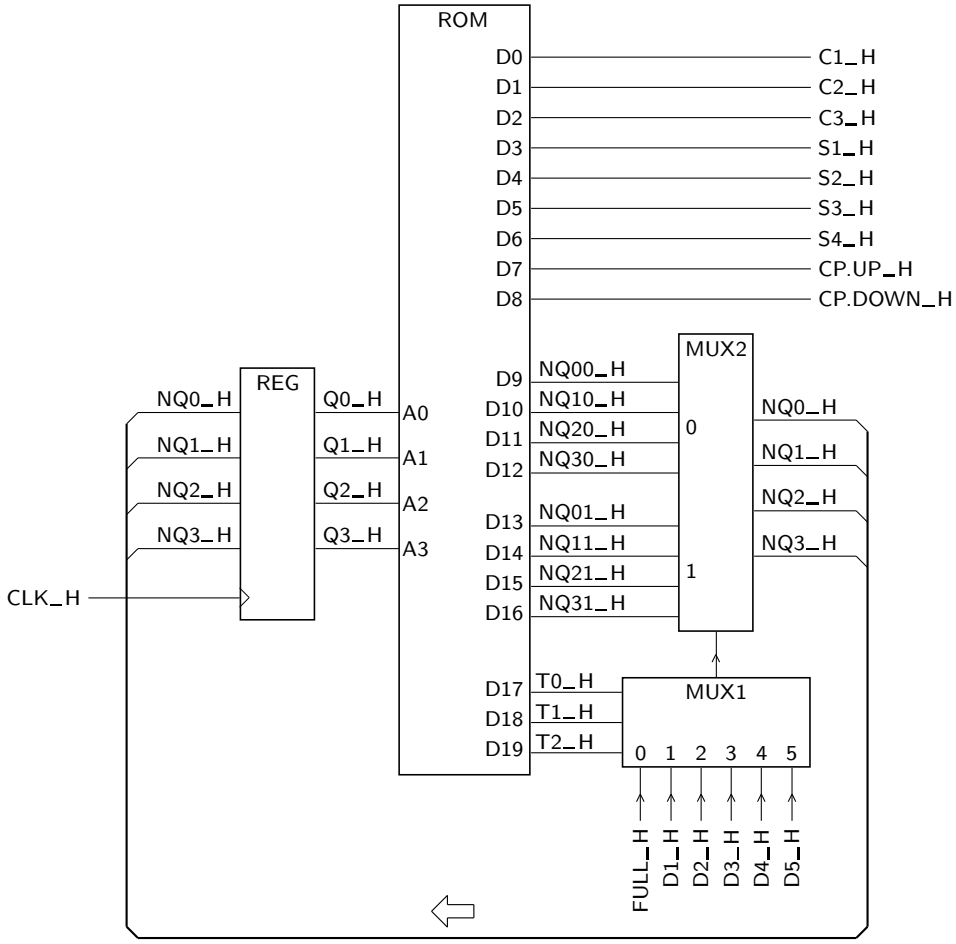


Figura 19.19: Diagrama de blocos da máquina de estados das Figuras 19.17 e 19.18, com controlo por ROM com endereçamento explícito

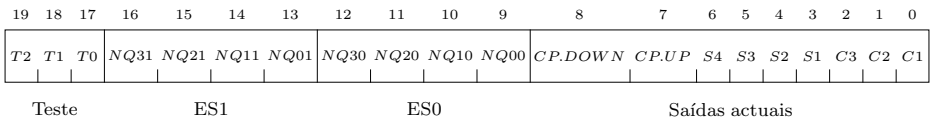


Figura 19.20: Formato das palavras num controlo por ROM com endereçamento implícito para a máquina de estados das Figuras 19.17 e 19.18

pelo que, nesta linha da ROM, o campo ES1 vem preenchido com essa codificação. Quanto às saídas actuais da máquina de estados, ficam todas desactivadas no estado $E0$.

Se, pelo contrário, $FULL = 0$, deve escolher-se o estado seguinte 0 (isto é, ES0), que se encontra codificado nos bits $D12$ a $D9$ da palavra da ROM com as designações $NQ30$ a $NQ00$, respectivamente. Neste caso vai-se para o estado

seguinte P, que foi codificado com

$$(Q3, Q2, Q1, Q0) = (1, 0, 1, 1)\,,$$

continuando a desactivar-se todas as saídas no estado $E0$.

De forma semelhante podíamos preencher as restantes linhas com a programação da ROM.

19.3.3 Controlo por ROM com endereçamento implícito

Endereçamento implícito

Uma variante da estrutura anterior usa implicitamente um dos dois endereços de estado seguinte, permitindo reduzir ainda mais as dimensões da ROM. Por essa razão, este tipo de controlo designa-se por **controlo por ROM com endereçamento implícito**.

Nessa estrutura, substitui-se o registo por um contador com carregamento em paralelo (Figura 19.21), e rearruma-se o fluxograma para que, de cada estado actual, se possa evoluir para o estado seguinte de contagem, ou então saltar para um outro estado qualquer, não colocado imediatamente a seguir em termos de contagem.

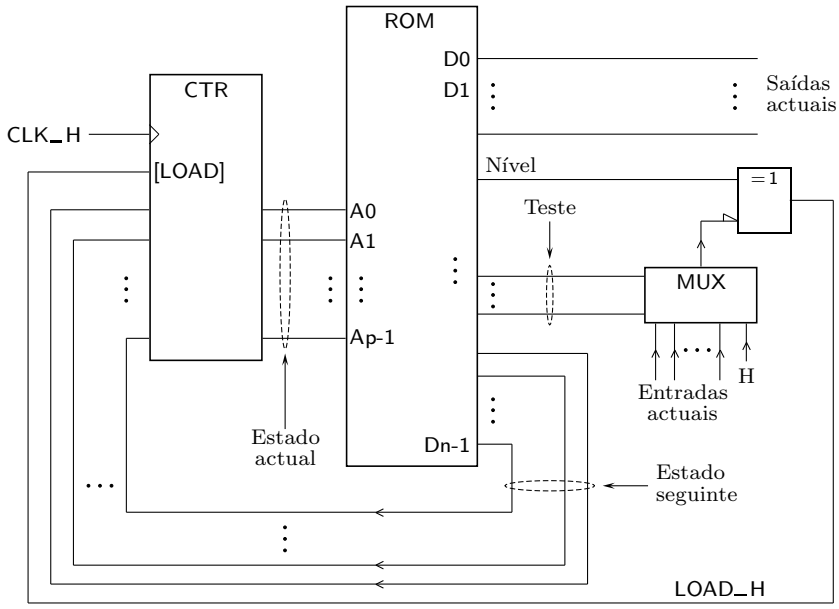


Figura 19.21: Diagrama de blocos de uma máquina de estados com controlo por ROM com endereçamento implícito

Agora não é preciso ter dois campos de estado seguinte na ROM, como acontecia com o endereçamento explícito, e a poupança em matéria de número total de palavras provém desse facto. De notar que a poupança é na *dimensão de cada palavra*, e não no número de palavras (que até pode ser ligeiramente superior ao do controlo com endereçamento explícito).

Usando esta estrutura, há três hipóteses a partir de cada estado actual:

Contagem

— ou se continua a contar para o estado seguinte (**contagem**);

— ou se salta para um estado que não é o estado seguinte de contagem, consoante o valor de uma variável de entrada (**transição condicionada** ou **salto condicionado**); para isso, existe um campo Teste que seleciona a variável de entrada, e um campo Nível onde se decide se o salto se deve efectuar quando ela tiver o valor 1 ou o valor 0;

Transição condicionada (salto condicionado)

— ou se salta incondicionalmente (independentemente do valor de qualquer variável de entrada) para um estado que não é o estado seguinte de contagem (**transição ou salto incondicional**), quando o estado seguinte corresponde obrigatoriamente a um salto na contagem; nesse caso selecciona-se a entrada 1 do MUX, e coloca-se o campo Nível a 1 (isso significa, de acordo com a mnemónica anterior, "salta se 1 fôr 1", o que é, obviamente, verdadeiro).

Transição (salto) incondicional

Pode-se resumir a funcionalidade dos campos Teste e Nível com a seguinte mnemónica: *salta-se se o valor da variável de entrada seleccionada pelo campo Teste fôr igual ao do campo Nível*.

O XOR com níveis de actividade diferentes nas entradas assegura essa funcionalidade, fazendo o carregamento em paralelo do contador em todas as situações de salto, e deixando contar no caso contrário (relembrar que um XOR tem a saída activa se uma e apenas uma das suas entradas estiver activa).

No exemplo do parque de estacionamento temos ainda 4 linhas de endereço, cada uma correspondendo a um estado actual (tal como sucedia com o controlo por ROM com endereçamento explícito), mas as palavras agora vão ter menos bits:

— 9 bits de saída;

— 1 bit de nível;

— 3 bits de teste; e

— 4 bits de estado seguinte,

num total de 17 bits por palavra. Cada palavra possui, então, o formato que se ilustra na Figura 19.22.

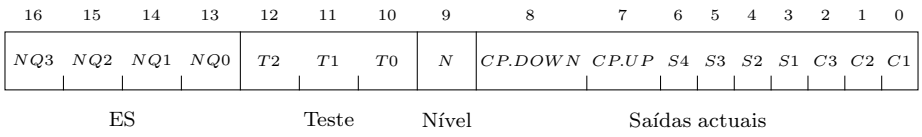


Figura 19.22: Formato das palavras num controlo por ROM com endereçamento implícito para a máquina de estados que controla o acesso ao parque de estacionamento da Figura 19.2

Com podermos usar esta estrutura de controlo, agora temos que ter cuidado com a codificação das variáveis de estado, ao contrário do que sucedia com as arquitecturas anteriores, em que as codificações eram arbitrárias.

Com efeito, agora temos de garantir que o estado seguinte a cada estado actual possui, tanto quanto possível, a configuração "que se segue" à do estado actual, ou então codificamo-lo com uma configuração que corresponde a um salto.

Ora, se examinarmos o fluxograma das Figuras 19.17 e 19.18, constatamos que esse tipo de codificações é simples de atribuir aos diversos estados, com excepção dos estados $E5$ e $E10$. Com efeito, $E5$ evolui para $E5$ ou para $E0$, e $E10$ evolui para $E10$ ou para $E0$, isto é, $E5$ e $E10$ evoluem, cada um, para um de dois estados em que nenhum deles é o "estado seguinte" na sequência natural de contagem

Daí que tenhamos que fazer uma nova alteração no fluxograma. Há que criar dois novos estados seguintes a $E5$ e a $E10$ que evoluam depois com saltos incondicionais para o estado inicial, $E0$.

Para tanto, desenhamos nas Figuras 19.23 e 19.24 o novo fluxograma para a máquina de estados que controla o acesso ao parque de estacionamento, necessário para este tipo de controlo.

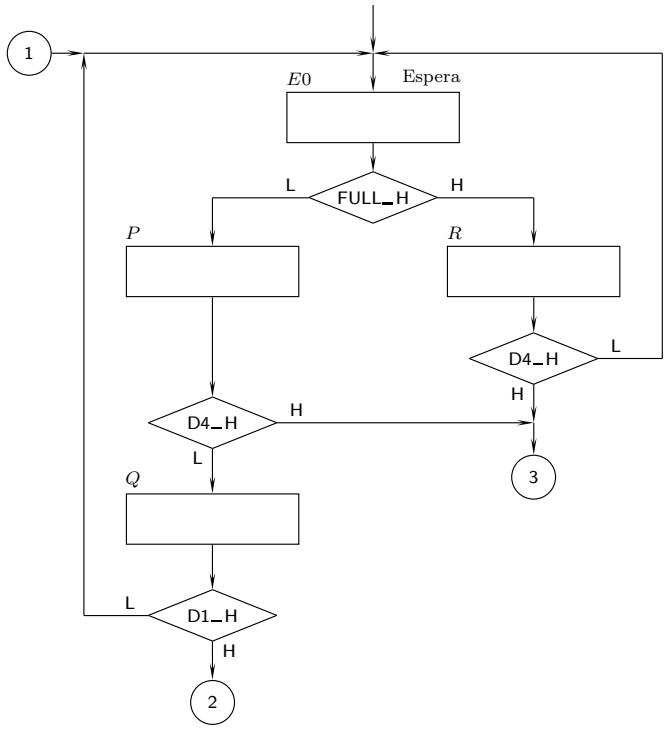


Figura 19.23: Parte inicial do fluxograma da máquina de estados que controla o acesso ao parque de estacionamento, quando se utiliza um controlo por ROM com endereçamento implícito

Reparemos que a Figura 19.23, com o início do fluxograma, é igual à Figura 19.17 porque esta parte do fluxograma não precisa de ser alterada.

Por outro lado, a Figura 19.24 é semelhante à Figura 19.18, excepto que possui os dois estados suplementares, S e T, que se acabaram de referir.

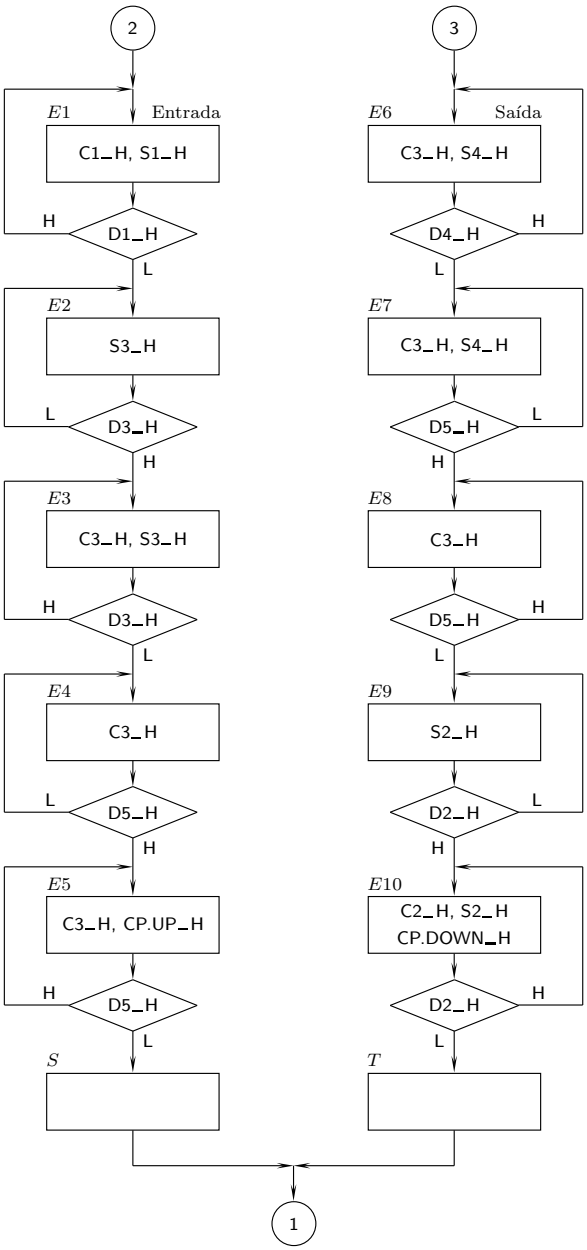


Figura 19.24: Continuação do fluxograma da Figura 19.23

A nova codificação de variáveis de estado será, então, a que se indica na Tabela 19.6.

Repare-se que as codificações dos estados S e T foram colocadas na sequência de $E5$ e de $E10$, respectivamente.

As variáveis de entrada vão ser seleccionadas da mesma forma que no exemplo anterior, com excepção do facto de se incluir o nível H na última entrada (que corresponde ao salto incondicional), seleccionável com o valor 111 no campo de

Tabela 19.6: Tabela de codificação de estados para a máquina de estados das Figuras 19.23 e 19.24, com controlo por ROM com endereçamento implícito

Estado	Q3	Q2	Q1	Q0
$E0$	0	0	0	0
P	0	0	0	1
Q	0	0	1	0
$E1$	0	0	1	1
$E2$	0	1	0	0
$E3$	0	1	0	1
$E4$	0	1	1	0
$E5$	0	1	1	1
S	1	0	0	0
R	1	0	0	1
$E6$	1	0	1	0
$E7$	1	0	1	1
$E8$	1	1	0	0
$E9$	1	1	0	1
$E10$	1	1	1	0
T	1	1	1	1

teste.

Obtém-se, assim, o diagrama de blocos da Figura 19.25 para a máquina de estados que opera o controlo de acessos ao parque de estacionamento, quando se usa um controlo por ROM com endereçamento implícito.

Neste caso, a tabela de verdade lógica para a ROM fica preenchida como se indica na Tabela 19.7.

Notemos como, em cada uma das linhas da Tabela 19.7, vem sempre identificada uma situação de salto. A progressão para o estado seguinte de contagem é efectuada pela desactivação da linha $LOAD_$H e corresponde a uma situação em que não se verifica a condição de salto descrita nessa linha.

Por exemplo, do estado actual $E0$ salta-se para o estado seguinte R se $FULL = = 1$. No caso contrário, prossegue-se com a contagem para o estado seguinte a $E0$, isto é, P (que não vem identificado na tabela, por não ser necessário). Esta é uma situação de salto condicionado.

Pelo contrário, do estado actual S salta-se sempre para o estado seguinte $E0$, independentemente dos valores das variáveis de entrada. Neste caso não há transição para um estado seguinte de contagem, pelo que estamos na presença de um salto incondicional.

Tabela 19.7: Tabela de estados lógica com o conteúdo da ROM a programar para a máquina de estados das Figuras 19.23 e 19.24, quando se utiliza uma estrutura de controlo por ROM com endereçamento implícito

					ES				**Teste**			**Nível**	**Saídas actuais**								
	Q3	Q2	Q1	Q0	NQ3	NQ2	NQ1	NQ0	T2	T1	T0	N	CP.DOWN	CP.UP	S4	S3	S2	S1	C3	C2	C1
Estado	A3	A2	A1	A0	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
$E0$	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0
Q	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
$E1$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	1	1	0	1
$E2$	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	1	0	0
$E3$	0	1	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0	0	1	0	0
$E4$	0	1	1	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0
$E5$	0	1	1	1	0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0
S	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
R	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$E6$	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	0
$E7$	1	0	1	1	1	0	1	1	1	0	1	1	0	0	1	0	0	0	1	0	0
$E8$	1	1	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	0	1	0	0
$E9$	1	1	0	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0
$E10$	1	1	1	0	1	1	1	0	0	1	0	1	1	0	0	0	1	0	0	1	0
T	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0

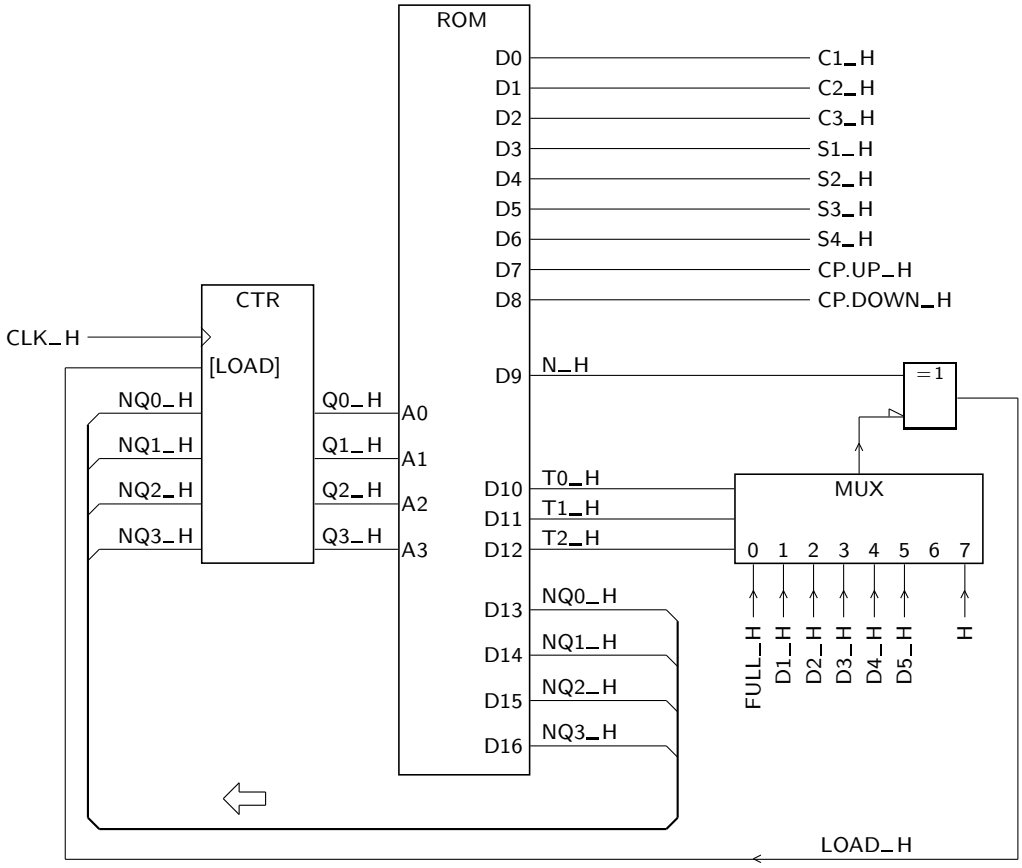


Figura 19.25: Diagrama de blocos da máquina de estados das Figuras 19.23 e 19.24, com controlo por ROM com endereçamento implícito

19.4 Exercícios

Nota: os exercícios identificados com um asterisco (∗) estão resolvidos em SD:ER.

19.1 Pretende-se desenhar uma máquina de estados que implemente um conversor pararelo-série para números com 8 bits. Cada número é identificado por $NUM7$ a $NUM0$, sendo $NUM7$ o bit mais significativo e $NUM0$ o menos significativo. A máquina deve possuir uma entrada $START_$H que, quando activada, desencadeia o processo de conversão, e uma saída $DONE_$H que, quando activada, indica o fim do processo.

Para facilitar o processo de conversão, o circuito a controlar deverá possuir: (i) um registo PISO com 8 bits do tipo indicado na Figura 19.26; e (ii) um contador (à sua escolha) que contabilize o número de deslocamentos efectuados. O registo e o contador devem possuir entradas de relógio que são o complemento da entrada de relógio do circuito de controlo.

(a) Desenhar um diagrama de blocos que contenha o circuito de controlo e o circuito controlado, identificando claramente as entradas e as saídas

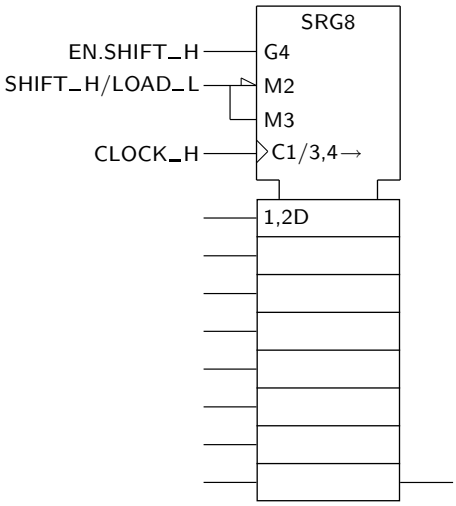


Figura 19.26: Registo de deslocamento do tipo PISO utilizado no Exercício 19.1

de cada um deles.

(b) Desenhar o fluxograma da ASM, justificando todas as opções que tomou, nomeadamente as que resultam de se pretender que a entrada de relógio do registo e do contador sejam o complemento da entrada de relógio do circuito de controlo.

19.2 Uma solução alternativa à do Exercício anterior põe o circuito de controlo a gerar os impulsos de relógio para o registo de deslocamento e para o contador. Redesenhe o fluxograma da máquina de estado nestas condições.

19.3 Para simplificar o circuito controlado do Exercício 19.1, foi decidido remover o contador de deslocamentos. Redesenhe o fluxograma da ASM para este caso.

19.4 Repita os três exercícios anteriores, mas agora para um conversor série-paralelo. Utilize o registo SIPO de 8 bits que entender.

19.5 No Exercício 8.3 desenhou-se um circuito complementador para 2 puramente combinatório. Agora pretende-se desenhar uma solução sequencial síncrona para o mesmo problema, sob a forma de um circuito de controlo e de um circuito controlado como o da Figura 19.27.

O número de 8 bits a complementar é designado por $DAT7$ a $DAT0$, sendo $DAT7$ o bit mais significativo e $DAT0$ o menos significativo. Esse número deve vir inicialmente carregado em paralelo num registo de deslocamento do tipo PISO, e o seu complemento para 2 deve ser obtido em série na saída OUT_H, começando pelo bit de menor peso.

A máquina de estados que controla o circuito da Figura 19.27 deve possuir uma entrada $START\_H$ que, quando activada, desencadeia o processo de obtenção do complemento para 2 do número, e uma saída $DONE\_H$ que, quando activada, indica que a conversão terminou.

Desenhar: (i) um diagrama de blocos para a totalidade do circuito; (ii) o fluxograma da ASM; e (iii) um diagrama temporal que explicite as opções

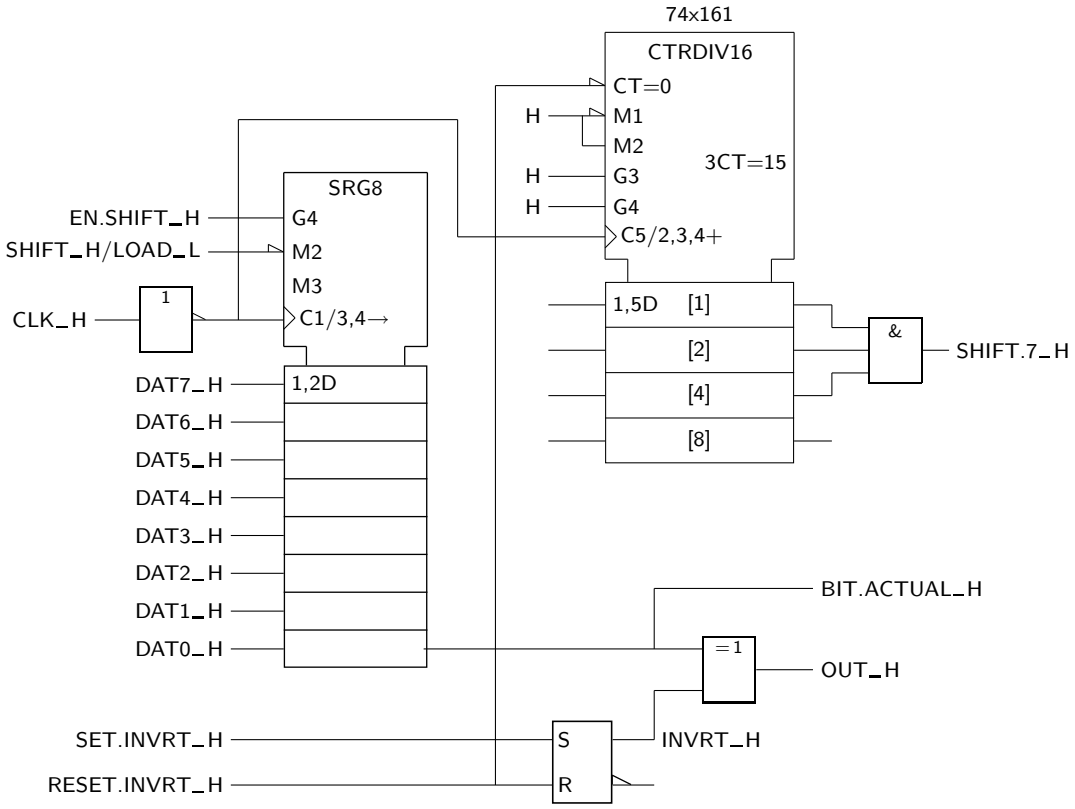


Figura 19.27: Circuito controlado utilizado no Exercício 19.5

que tomou e que mostre a geração de, pelo menos, dois bits na saída OUT_H.

19.6 Repita o exercício anterior para o caso em que os impulsos de relógio para o circuito controlado são gerados no circuito de controlo.

19.7 Repita o exercício anterior para o caso em que os bits que constiturm o complemento para 2 do número entrado são reinseridos no registo de deslocamento por forma a que, no fim do processo de conversão, este contenha o complemento para 2 do número inicial.

19.8 Para simplificar o circuito controlado do exercício 19.6, eliminaram-se o contador de deslocamentos e o flip-flop que força (ou não) a complementação do bit proveniente do registo de deslocamento. Obteve-se, assim, o circuito da Figura 19.28.

Redesenhar o fluxograma da ASM de controlo.

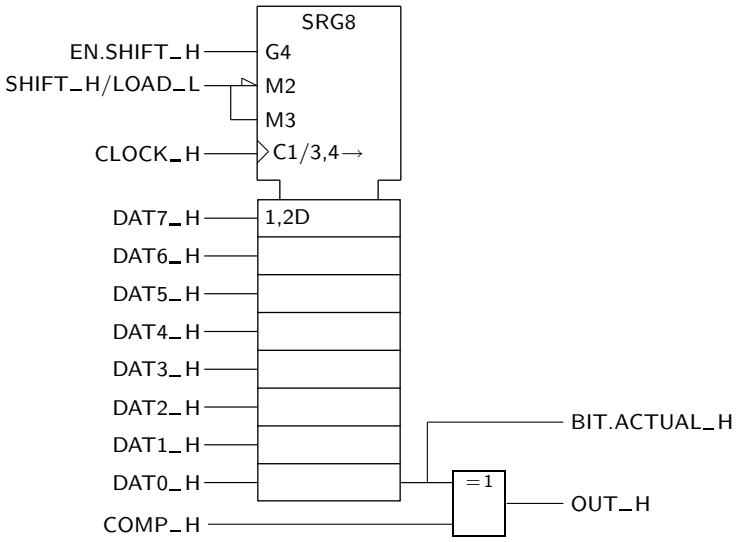


Figura 19.28: Circuito controlado utilizado no Exercício 19.8

Índice Remissivo