

### 3. Sistemas de numeração e codificação

Neste capítulo são discutidos aspectos sobre sistema de numeração em representação de quantidades. São discutidas metodologias para representação de quantidades em diferentes bases de numeração. Alguns sistemas de codificação binários são também apresentados.

#### 3.1 Notação posicional

Essa é a notação que é habitualmente usada no dia a dia como forma de escrever quantidades. Dígitos são usados em posições específicas, cada posição contendo um significado na composição das quantidades.

**Definição 3.1.1** (Notação posicional) Método numérico de representação de quantidades no qual a quantidade representada por cada dígito usado na representação depende da sua posição relativa na composição do número. A notação posicional é usada para expressar uma quantidade justapondo uma sequência de símbolos (normalmente números arábicos) lado a lado.

A notação posicional expressa quantidades numéricas usando uma quantidade numérica fundamental denominada de base. Assim um número  $x$  de  $n$  dígitos na base  $r$  é expresso por:

$$(x)_r = [x_{n-1} | x_{n-2} | x_{n-3} | \dots | x_1 | x_0] \quad (3.1)$$

em que,  $x_i | i \in \{0 \dots r - 1\}$  são os dígitos usados na composição do número  $(x)_r$ .

Hoje, a notação posicional é padrão na sociedade para se expressar quantidades. Ela é utilizada no dia-a-dia com valor de base igual a 10. A quantidade expressa pelo número  $(x)_r$  na base 10 é dada por:

$$(x)_{10} = \sum_{i=0}^{n-1} x_i r^i. \quad (3.2)$$

■ **Exemplo 3.1** Qual a quantidade, na base 10, expressa pelo número  $x = (1110)_2$ ?

Solução:

$$(x)_{10} = \sum_{n=0}^3 x_i r^i = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (14)_{10}.$$

■ **Exemplo 3.2** Qual a quantidade, na base 10, expressa pelo número  $x = (56)_7$ ?

Solução:

$$(x)_{10} = \sum_{i=0}^1 x_i 7^i = 5 \times 7^1 + 6 \times 7^0 = (41)_{10}.$$

■ **Exemplo 3.3** Qual a quantidade, na base 10, expressa pelo número  $x = (56)_9$ ?

Solução:

$$(x)_{10} = \sum_{i=0}^1 x_i 9^i = 5 \times 9^1 + 6 \times 9^0 = (51)_{10}.$$

É possível representar quantidades inteiras usando (3.2). No entanto, a notação posicional também pode ser usada para expressar quantidades fracionárias. Ou seja, um número fracionário  $x$  na base  $r$  com  $n$  dígitos inteiros e  $m$  dígitos fracionários pode ser expresso na notação posicional assim:

$$(x)_r = [x_{n-1} | \dots | x_1 | x_0 | , | x_{-1} | x_{-2} | \dots | x_{-(m-1)} | x_{-(m)}] \quad (3.3)$$

A quantidade expressa pelo número fracionário  $(x)_r$  em (3.3) na base 10 é dada por:

$$(x)_{10} = \sum_{i=-m}^{n-1} x_i r^i. \quad (3.4)$$

■ **Exemplo 3.4** Qual a quantidade, na base 10, expressa pelo número  $x = (1110,111)_2$ ?

Solução:

$$(x)_{10} = \sum_{n=-3}^3 x_i 2^i = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = (14,875)_{10}$$

■ **Exemplo 3.5** Qual a quantidade, na base 10, expressa pelo número  $x = (56,35)_7$ ?

Solução:

$$(x)_{10} = \sum_{i=-2}^1 x_i 7^i = 5 \times 7^1 + 6 \times 7^0 + 3 \times 7^{-1} + 5 \times 7^{-2} = (41,5306122\dots)_{10}$$

■ **Exemplo 3.6** Qual a quantidade, na base 10, expressa pelo número  $x = (56,35)_9$ ?

Solução:

$$(x)_{10} = \sum_{i=-2}^1 x_i 9^i = 5 \times 9^1 + 6 \times 9^0 + 3 \times 9^{-1} + 5 \times 9^{-2} + = (51,39506172\dots)_{10}$$

■

O sistema de numeração mais utilizado neste documento é o binário ( $r = 2$ ). Sendo assim será destacada a forma de encontrar a quantidade expressa por um número escrito na base dois. Em decimal, a quantidade expressa por um número binário inteiro  $x$  de  $n$  bits ( $x = x_{n-1}\dots x_1 x_0$ ) é dada por:

$$(x)_{10} = \sum_{i=0}^{n-1} x_i 2^i. \quad (3.5)$$

**Definição 3.1.2** Dado o número binário  $x$  composto por  $n$  bits ( $x = x_{n-1}\dots x_1 x_0$ ), o bit  $x_{n-1}$  é chamado de *bit* mais significativo de  $x$  (ou MSB) enquanto o bit  $x_0$  é chamado de bit menos significativo de  $x$  (ou LSB).

### 3.1.1 Relações entre quantidades e número de dígitos

Para o escopo deste documento duas questões são importantes:

- Qual a máxima quantidade pode ser expressa por número de  $k$  dígitos na base  $r$ ?
- O número na base  $r$  necessita de quantos dígitos para expressar uma determinada quantidade?

Se faz necessário entender como realizar os cálculos para responder essas perguntas.

#### Máxima quantidade que pode ser expressa com um certo número de dígitos

Considere um número  $x$  ( $x = x_{k-1}\dots x_1 x_0$ ) de  $k$  dígitos na base  $r$ . A máxima quantidade que pode ser expressa por esse número (notada aqui como  $x_{MAX}$ ) é obtida quando todos os dígitos do número  $x$  são os maiores valores possíveis. Na base  $r$ , o maior valor possível expresso por um dígito é  $x_i = r - 1$ . Usando essa argumentação e substituindo  $x_i = r - 1$  em (3.2) vem:

$$\begin{aligned} x_{MAX} &= \sum_{i=0}^{k-1} (r-1) r^i \\ &= \sum_{i=0}^{k-1} r^{i+1} - \sum_{i=0}^{k-1} r^i \\ &= r \left( \frac{r^k - 1}{r - 1} \right) - \left( \frac{r^k - 1}{r - 1} \right) \\ &= \frac{r^{k+1} - r - r^k + 1}{r - 1} \\ &= \frac{(r-1)r^k - (r-1)}{r-1} \\ &= r^k - 1. \end{aligned} \quad (3.6)$$

Perceba que os dois somatórios da segunda linha de (3.6) expressam somas de progressões geométricas. Sendo assim, a passagem da segunda para terceira linha é realizada utilizando a fórmula da soma de uma progressão geométrica. Ou seja, a maior quantidade que pode ser expressa por um número na base  $r$  é dada, na base 10, por:

$$(x_{MAX})_{10} = r^k - 1. \quad (3.7)$$

**■ Exemplo 3.7** Qual a quantidade máxima que pode ser expressa por um número binário de 8 bits?  
Resposta: Para esse problema tem-se  $k = 8$  (8 bits) e  $r = 2$  (número binário). Aplicando (3.7) vem:

$$(x_{MAX})_{10} = r^k - 1 = 2^8 - 1 = 255. \quad (3.8)$$

■

#### Número de dígitos necessários para expressar uma determinada quantidade

Como visto na subseção anterior a máxima quantidade que pode ser expressa por um número  $x$  ( $x = x_{k-1} \dots x_1 x_0$ ) na base  $r$  é  $r^k - 1$ . Para que uma quantidade  $z$  possa ser expressa na base  $r$  o número de dígitos  $k$  usado na representação numérica precisa ser grande o suficiente tal que se verifique:

$$z \leq r^k - 1. \quad (3.9)$$

Rearrumando chega-se a:

$$r^k \geq z + 1 \quad (3.10)$$

Como função logaritmo monotônica, é possível aplicar o logaritmo na base  $r$  em ambos os lados de (3.10) para se isolar o valor de  $k$ , chegando-se a:

$$\log_r r^k \geq \log_r (z + 1) \Rightarrow k \geq \log_r (z + 1). \quad (3.11)$$

Note que com a expressão (3.11) é possível se chegar ao valor mínimo de  $k$  necessário para expressar a quantidade  $z$ . Esse valor mínimo é  $\log_r (z + 1)$ . Como obrigatoriamente o valor de  $k$  é inteiro (pois  $k$  é o número de dígitos) é possível usar a função “menor inteiro maior que” para se calcular mais simplesmente o valor de  $k$  necessário para se expressar a quantidade  $z$ :

$$k = \lceil \log_r z \rceil, \quad (3.12)$$

em que  $\lceil y \rceil$  é o menor inteiro maior que  $y$ .

**■ Exemplo 3.8** Quantos bits são necessários em um número binário para expressar quantidades que vão até 300?

Resposta: Para esse problema tem-se  $z = 300$  e  $r = 2$  (número binário). Aplicando (3.12) vem:

$$k = \lceil \log_2 z \rceil = \lceil \log_2 300 \rceil = \lceil 8,228818 \rceil = 9. \quad (3.13)$$

Ou seja, é necessário um número binário de pelo menos 9 bits para se representar quantidades até 300.

■

### 3.2 Principais bases usadas em circuitos digitais

Para se trabalhar com circuitos digitais, o principais esquemas de codificação numéricas utilizados são os sistemas de base  $r$  mostrados na Tabela. 3.1.

Tabela 3.1: Principais esquemas de codificação numéricas utilizados em circuitos digitais.

Base	Denominação
$r = 2$	Binário
$r = 8$	Octal
$r = 10$	Decimal
$r = 16$	Hexadecimal

As quatro primeiras colunas da Tabela. 3.2 mostra a codificação padrão usada para as bases elencadas quando é considerado um cenário de números de 4 bits. Perceba que no sistema hexadecimal são necessários 16 símbolos diferentes. Só estão disponíveis 10 símbolos para representação numérica. Sendo assim, O sistema hexadecimal necessita de 6 símbolos extras, que normalmente são escritos usando as letras maiúsculas de A a F conforme mostrado na Tabela. 3.2.

Tabela 3.2: Principais códigos e sistemas de numeração considerados em circuitos digitais.

Decimal	Binário	Octal	Hexadecimal	Gray	Excesso-3	Excesso-5
0	0000	00	0	0000	0011	0101
1	0001	01	1	0001	0100	0110
2	0010	02	2	0011	0101	0111
3	0011	03	3	0010	0110	1000
4	0100	04	4	0110	0111	1001
5	0101	05	5	0111	1000	1010
6	0110	06	6	0101	1001	1011
7	0111	07	7	0100	1010	1100
8	1000	10	8	1100	1011	1101
9	1001	11	9	1101	1100	1110
10	1010	12	A	1111	1101	1111
11	1011	13	B	1110	1110	-
12	1100	14	C	1010	1111	-
13	1101	15	D	1011	-	-
14	1110	16	E	1001	-	-
15	1111	17	F	1000	-	-

#### 3.2.1 Principais conversões entre bases

Nesta seção são mostradas algumas formas de conversão entre números nas principais bases usadas em circuitos digitais.

##### Conversão entre números decimais e binários

A conversão de números binários para decimais pode ser feita usando (3.5).

$$\begin{array}{r}
 N \quad 2 \\
 r_0 \quad q_0 \quad 2 \\
 \quad \quad r_1 \quad q_1 \quad 2 \\
 \quad \quad \quad r_2 \quad \dots \quad 2 \\
 \quad \quad \quad \quad r_{n-2} \quad q_{n-2} \quad 2 \\
 \quad \quad \quad \quad \quad r_{n-1} \quad q_{n-1}
 \end{array}$$

Figura 3.1: Método das divisões sucessivas para conversão decimal-binário.

Já a conversão de números decimais para números binários pode ser feita usando o algoritmo das divisões sucessivas por dois (base do sistema de numeração binário). O resultado da conversão é fornecido pela justaposição ordenada do resultado do último quociente obtido (dígito MSB) seguido pelos restos encontrados sucessivamente na última, penúltima, e assim sucessivamente, divisões realizadas. O método é mostrado na Fig. 3.1. Nela é possível se ver o número  $N$  sendo sucessivamente dividido por 2 e, a cada divisão, gerando um quociente  $q_i$  e um resto  $r_i$ . Após efetuar todas as divisões sucessivas o número  $N$  pode ser escrito em binário por:

$$(N)_{10} = (q_{n-1}r_{n-1}r_{n-2}\dots r_1r_0)_2 \quad (3.14)$$

■ **Exemplo 3.9** Converta o número 12 na base 10 para um número na base 2.

Solução:

$$\begin{array}{r}
 12 \quad 2 \\
 0 \quad 6 \quad 2 \\
 \quad \quad 0 \quad 3 \quad 2 \\
 \quad \quad \quad 1 \quad 1
 \end{array}$$

Figura 3.2: Aplicação do método das divisões sucessivas.

A aplicação do método de divisão sucessivas está mostrado na Fig. 3.2. O número binário deve ser lido na seguinte sequência: último quociente (em vermelho), último resto (em azul), penúltimo resto (em verde) e primeiro resto (em amarelo), ou seja, 1100. ■

### Conversão entre números octais e binários

A conversão entre números octais e binários pode ser feita transformando diretamente cada dígito do número octal em um número binário de 3 bits, justapondo os grupos de 3 bits lado a lado na mesma ordem dos respectivos dígitos octais.

Dado um número octal  $O$  de  $n$  dígitos ( $O = o_{n-1}o_{n-2}\dots o_1o_0$ ) ele pode ser transformado em um número binário  $B$  de  $3n - 1$  dígitos ( $B = b_{3n-1}b_{3n-2}\dots b_2b_1b_0$ ) como mostrado na Tabela. 3.3.

Tabela 3.3: Conversão entre números, binário-octal e octal-binário

octal	$o_{n-1}$	$o_{n-2}$	...	$o_1$	$o_0$
binário	$b_{3n-1}b_{3n-2}b_{3n-3}$	$b_{3n-4}b_{3n-5}b_{3n-6}$	...	$b_5b_4b_3$	$b_2b_1b_0$

Também é possível converter, de forma similar, um número binário em um número octal. Basta fatiar o número binário de 3 em 3 bits da direita para a esquerda (i.e. iniciando a contagem no bit

LSB) e transformar cada grupo de 3 *bits* do número binário em um dígito octal. *Bits* 0 à esquerda do número binário podem ser incluídos para se completar trios, se for necessário.

■ **Exemplo 3.10** Faça as seguintes conversões:

- O número octal  $O = (345)_8$  em um número binário.
- O número binário  $B = (100101011)_2$  em um número octal.

Solução letra a:

Realizando a conversão de cada dígito do número octal  $O = (345)_8$  em números binários de 3 *bits* vem: 3 = 011, 4 = 100 e 5 = 101. Ou seja,  $(345)_8 = (011100101)_2$ .

Solução letra b: O número binário  $B = (100101011)_2$  precisa ser dividido de trás em três *bits* da direita para a esquerda:

$$100 \mid 101 \mid 011.$$

Cada trio obtido deve ser convertido em um dígito octal:

$$\begin{array}{ccccccc} \text{binário:} & 100 & & 101 & & 011 & \\ & \downarrow & & \downarrow & & \downarrow & \\ \text{octal:} & 4 & & 5 & & 3 & \end{array}$$

Ou seja,  $(100101011)_2 = (453)_8$

■

**Conversão entre entre números hexadecimais e binários**

A conversão entre números hexadecimais e binários pode ser feita transformando diretamente cada dígito do número hexadecimal em um número binário de 4 *bits*, justapondo os grupos de 4 *bits* lado a lado na mesma ordem dos respectivos dígitos hexadecimais.

Dado um número hexadecimal  $H$  de  $n$  dígitos ( $H = h_{n-1}h_{n-2}\dots h_1h_0$ ) ele pode ser transformado em um número binário  $B$  de  $4n - 1$  dígitos ( $B = b_{4n-1}b_{4n-2}\dots b_2b_1b_0$ ) como mostrado na Tabela. 3.4.

Tabela 3.4: Conversão entre números, binário-hexadecimal e hexadecimal-binário

hexadecimal	$h_{n-1}$	$h_{n-2}$	...	$h_1$	$h_0$
binário	$b_{4n-1}b_{4n-2}b_{4n-3}b_{4n-4}$	$b_{4n-5}b_{4n-6}b_{4n-7}b_{4n-8}$	...	$b_7b_6b_5b_4$	$b_3b_2b_1b_0$

Também é possível converter, de forma similar, um número binário em um número hexadecimal. Basta fatiar o número binário de 4 em 4 *bits* da direita para a esquerda (*i.e.* iniciando a contagem no bit LSB) e transformar cada grupo de 4 *bits* do número binário em um dígito hexadecimal. *Bits* 0 à esquerda do número binário podem ser incluídos para se completar quartetos, se for necessário.

■ **Exemplo 3.11** Faça as seguintes conversões:

- O número hexadecimal  $H = (9FA)_{16}$  em um número binário.
- O número binário  $X = (100101011)_2$  em um número hexadecimal.

Solução letra a:

Realizando a conversão de cada dígito do número octal  $H = (9FA)_{16}$  em números binários de 4 *bits* vem: 9 = 1001, F = 1111 e A = 1010. Ou seja,  $(9FA)_{16} = (10011111010)_2$ .

Solução letra b:

O número binário  $X = (100101011)_2$  precisa ser dividido de quatro em quatro *bits* da direita para a esquerda:

$$1 \mid 0010 \mid 1011.$$

Como sobrou um 1 sozinho no terceiro quarteto da direita para a esquerda, completa-se esse quarteto com zeros a esquerda:

$$0001 \mid 0010 \mid 1011.$$

Cada quarteto obtido deve ser convertido em um dígito hexadecimal:

binário:	0001	0010	1011
	↓	↓	↓
hexadecimal:	1	2	B

Ou seja,  $(100101011)_2 = (12B)_{16}$  ■

### 3.3 Principais códigos binários

#### 3.3.1 Código ASCII

O código ASCII (*American Standard Code for Information Interchange*) é um código binário de 7 *bits* geralmente utilizado para codificar caracteres em sistemas computacionais.

O código ASCII é usado para representar caracteres textuais em computadores e equipamentos digitais em geral que necessitam trabalhar com textos. Ele foi introduzido nos anos 1960 e grande parte das codificações de caracteres usadas atualmente usam a ideia (ou herdaram diretamente) da codificação ASII como base.

A Tabela. 3.5 mostra parte da tabela ASCII (do caractere 48 ao 127) que relaciona o código binário respectivo a cada caractere representado pelo código. Por exemplo, o caractere “d” é representado no código pelo número decimal 100 e pelo respectivo binário 01100100.

#### 3.3.2 Código gray

O código de Gray é um sistema de codificação binária introduzido por Frank Gray. Contadores usando válvulas eletrônicas necessitavam de altos valores de potências, e podiam gerar muitos ruídos no momento que vários *bits* se modificavam simultaneamente. Sendo assim, a codificação gray foi pensada para a mudança entre dois números subsequentes de uma contagem apresentarem apenas uma única mudança de bit no respectivo código binário utilizados por esses números subsequentes. Nos dias atuais não há mais a preocupação de se evitar picos/ruídos de energia. No entanto o uso do esquema de codificação gray do evitar situação de corrida em circuitos digitais sequenciais, bem como aumentar a velocidade das transições em contadores binários. A quinta coluna Tabela. 3.2 mostra um exemplo de código binário usando a codificação gray.

**Definição 3.3.1** (Código Gray) Código binário no qual há a mudança de exatamente um único *bit* na representação numérica de números sucessivos.

Tabela 3.5: Tabela parcial da codificação ASC-II, do caractere 48 ao 127. A tabela mostra os números decimais e binários associados a cada caractere.

Dec	Binário	Caractere	Dec	Binário	Caractere	Dec	Binário	Caractere
48	00110000	0	75	01001011	K	102	01100110	f
49	00110001	1	76	01001100	L	103	01100111	g
50	00110010	2	77	01001101	M	104	01101000	h
51	00110011	3	78	01001110	N	105	01101001	i
52	00110100	4	79	01001111	O	106	01101010	j
53	00110101	5	80	01010000	P	107	01101011	k
54	00110110	6	81	01010001	Q	108	01101100	l
55	00110111	7	82	01010010	R	109	01101101	m
56	00111000	8	83	01010011	S	110	01101110	n
57	00111001	9	84	01010100	T	111	01101111	o
58	00111010	:	85	01010101	U	112	01110000	p
59	00111011	;	86	01010110	V	113	01110001	q
60	00111100	<	87	01010111	W	114	01110010	r
61	00111101	=	88	01011000	X	115	01110011	s
62	00111110	>	89	01011001	Y	116	01110100	t
63	00111111	?	90	01011010	Z	117	01110101	u
64	01000000	@	91	01011011	[	118	01110110	v
65	01000001	A	92	01011100	\`	119	01110111	w
66	01000010	B	93	01011101	]	120	01111000	x
67	01000011	C	94	01011110	^	121	01111001	y
68	01000100	D	95	01011111	_	122	01111010	z
69	01000101	E	96	01100000	'	123	01111011	{
70	01000110	F	97	01100001	a	124	01111100	
71	01000111	G	98	01100010	b	125	01111101	}
72	01001000	H	99	01100011	c	126	01111110	~
73	01001001	I	100	01100100	d	127	01111111	espaço
74	01001010	J	101	01100101	e			

### 3.3.3 Código BCD

O código BCD (*Binary-coded decimal*) é um código usado para escrever um número decimal usando código binário. Nesse esquema, de codificação cada dígito decimal é representado por um grupo de quatro bits (*nibble*). Cada dígito de um número decimal é substituído por um número binário de quatro bits. A Tabela 3.6 mostra a codificação BCD/binário para os 10 símbolos numéricos usados no sistema de numeração decimal.

■ **Exemplo 3.12** Escreva o número decimal 1924 usando o código BCD.

Solução: O código BCD codifica cada dígito decimal usando 4 bits. A seguinte correspondência é obtida para o número 1924 do exemplo:

Sendo assim, justapondo lado a lado a codificação binária mostrada acima é possível escrever:

$$(1924)_{10} = (0001100100100100)_{BCD} \quad (3.15)$$

Tabela 3.6: Correspondência entre os dígitos decimais e o código binário BCD.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
dígitos decimais:	1	9	2	4						
	↓	↓	↓	↓						
correspondentes em BCD:	0001	1001	0010	0100						

### 3.3.4 Código excesso- $n$

O código excesso- $n$  é um código binário cuja representação binária é a mesma usada no código binário padrão acrescido de  $n$  unidades. Por exemplo, o código excesso-3 representa o número 5 não pelo número binário 0101, como o código binário padrão faz, mas sim pelo número binário 1000 que é a codificação binária atribuída ao número  $5 + 3 = 8$ . Nas colunas 6 e 7 da Tabela. 3.2 estão mostradas duas formas de codificação excesso- $n$ : excesso-3 e excesso-5.

O uso da codificação excesso-3 pode facilitar a implementação de alguns circuitos somadores culminando inclusive na redução da quantidade de *hardware* necessário para sua implementação.

### 3.3.5 Códigos ponderados

São códigos nos quais é possível saber a quantidade expressa por um número escrito no código por meio de operações de multiplicação em soma conforme a definição 3.3.2.

**Definição 3.3.2** (Códigos ponderados) Um código  $C$  é dito ponderado se a quantidade  $Q$  representada por um número  $d$  ( $d = d_{n-1} \dots d_1 d_0$ ), escrito usando código  $C$ , pode ser calculada por:

$$Q = \sum_{i=0}^{n-1} d_i w_i, \quad (3.16)$$

em que  $d_i$  ( $i \in \{0, 1, \dots, n-1\}$ ) são os dígitos usados pelo código  $C$  e  $w_i$  é o peso do dígito  $d_i$  na composição das quantidades expressas pelo esquema de codificação  $C$ .

A notação posicional na base  $r$  qualquer é um exemplo de código ponderado. Nela, se tem  $w_i = r^i$ .

■ **Exemplo 3.13** Considere o código ponderado em que  $d_i \in \{0, 1\}$  e  $w_5 w_4 w_3 w_2 w_1 w_0 = 764311$ . Quais as quantidades expressas por:

- a)  $N_1 = 001101$
- b)  $N_2 = 001110$
- c)  $N_3 = 110000$

Solução: A quantidade  $Q_1$  expressa por  $N_1$  é tal que

$$Q_1 = \sum_{i=0}^5 x_i w_i = 0 \times 7 + 0 \times 6 + 1 \times 4 + 1 \times 3 + 0 \times 1 + 1 \times 1 = 8.$$

A quantidade  $Q_2$  expressa por  $N_2$  é tal que

$$Q_2 = \sum_{i=0}^5 x_i w_i = 0 \times 7 + 0 \times 6 + 1 \times 4 + 1 \times 3 + 1 \times 1 + 0 \times 1 = 8.$$

A quantidade  $Q_3$  expressa por  $N_3$  é tal que

$$Q_3 = \sum_{i=0}^5 x_i w_i = 1 \times 7 + 1 \times 6 + 0 \times 4 + 0 \times 3 + 0 \times 1 + 0 \times 1 = 13.$$

■

### 3.4 Problemas propostos

**Problema 3.1** Encontre, na base 10, a quantidade expressa por cada um dos números:

- a)  $(1011)_2$
- b)  $(678)_9$
- c)  $(A9CD)_{16}$
- d)  $(1010, 101)_2$
- e)  $(256)_7$
- f)  $(743)_8$

**Problema 3.2** Qual a quantidade máxima, na base 10, pode ser expressa em cada situação a seguir:

- a) Número binário de 12 bits.
- b) Número hexadecimal de 3 dígitos.
- c) Número binário de 4 bits.
- d) Número com 7 dígitos na base 5.
- e) Número com 3 dígitos na base 9.

**Problema 3.3** Determine o número mínimo de dígitos, na base indicada, necessários para representar a quantidade mostrada em cada item:

- a) Quantidade  $(11)_{10}$ , na base 2.
- b) Quantidade  $(250)_{10}$ , na base 2.
- c) Quantidade  $(1024)_{10}$ , na base 2.
- d) Quantidade  $(1023)_{10}$ , na base 2.
- e) Quantidade  $(2000)_{10}$ , na base 16.
- f) Quantidade  $(500)_{10}$ , na base 7.

**Problema 3.4** Realize as conversões numéricas indicadas:

- a)  $(1FFA)_{16}$ , na base 2.
- b)  $(10100011)_2$ , na base 16.
- c)  $(1234)_{16}$ , na base 2.
- d)  $(100011)_2$ , na base 16.
- e)  $(100011)_8$ , na base 2.
- f)  $(1011)_8$ , na base 2.

**Problema 3.5** Escreva seu nome em um código hexadecimal utilizando o esquema de codificação ASCII para representar os caracteres textuais.

**Problema 3.6** Escreva os números indicados usando o código BCD.

- a)  $(81)_{10}$
- b)  $(250)_{10}$
- c)  $(1024)_{10}$

**Problema 3.7** Considere os códigos ponderados binários definidos em cada item e determine a quantidade relacionada com cada número binário.

- a) Pesos do código  $w_3w_2w_1w_0 = 8421$ , número 1010.
- b) Pesos do código  $w_3w_2w_1w_0 = 7511$ , número 1010.
- c) Pesos do código  $w_3w_2w_1w_0 = 7511$ , número 1001.

## 4. Funções de chaveamento e portas lógicas

Neste capítulo é discutido como aplicar a álgebra de boole para realizar a descrição de circuitos digitais. São abordadas as expressões e funções de chaveamento bem como formas de representar funções e expressões de chaveamento. São apresentadas as portas lógicas e a representação de sistemas combinacionais usando funções de chaveamento.

### 4.1 Álgebra de boole usada na descrição de circuitos digitais

É possível descrever circuitos digitais utilizando uma instância específica de uma álgebra de boole binária. Essa consiste em um conjunto  $B$  de dois elementos,  $B = \{0, 1\}$  e três operadores básicos que implementam as operações definidas na álgebra de boole. Esses operadores são, o OU lógico, o E lógico e a NEGAÇÃO lógica, que correspondem, respectivamente, às operações  $+$ ,  $\cdot$  e  $(\ )'$  discutidas no Capítulo 2. A partir daqui que é usada a nomenclatura OR para se referir a operação lógica OU, AND para se referir a operação lógica E e NOT para se referir ao complemento. A Tabela 4.1 mostra esses operadores sendo utilizados nas variáveis binárias  $X$  e  $Y$  (*i.e.*  $X, Y \in \{0, 1\}$ ) e a simbologia matemática que é utilizada neste documento para representar essas operações.

Tabela 4.1: Operações OR, AND e NOT aplicadas às variáveis  $X$  e  $Y$

	OR	$X + Y$
	AND	$X \cdot Y$
	NOT	$\bar{X}$

Esses três operadores podem ser definidos por meio das tabelas verdade mostradas na Tabela 4.2.

### 4.2 Álgebra de chaveamento binária

Utilizando os operadores discutidos na Seção 4.1 juntamente com o conjunto binário de elementos é possível definir uma álgebra especial para descrição/análise de circuitos digitais, a álgebra de

Tabela 4.2: Tabelas verdade binárias para os operadores: (a) NOT, (b) OR, (c) AND.

$X$	$\bar{X}$	$X$	$Y$	$\bar{Y}$	$X \cdot Y$	$X + Y$
0	1	0	0	1	0	0
1	0	1	1	0	1	1
(a)		(b)		(c)		

chaveamento (Definição 4.2.1).

**Definição 4.2.1** (Álgebra de chaveamento binária ou Álgebra de chaveamento) - É uma álgebra de boole que possui dois elementos, 0 e 1 e implementa os operadores da álgebra de boole usando os operadores lógicos OR (+), AND (.) e NOT ( $\bar{()}$ ).

As descrições e análises dos circuitos digitais neste documento fazem uso das expressões de chaveamento.

## 4.3 Funções de chaveamento

É possível se relacionar variáveis binárias por meio de funções. Essas funções são denominadas de funções de chaveamento (Definição 4.3.1).

**Definição 4.3.1** (função de chaveamento binária ou função de chaveamento) - É uma correspondência entre o conjunto  $\{0, 1\}^n$  ( $n$  variáveis binárias) e o conjunto  $\{0, 1\}$  (uma variável binária). Uma Função de chaveamento pode ser escrita algebraicamente utilizando a álgebra de chaveamento.

Um função de chaveamento  $f(x_0, x_1, \dots, x_{n-1})$  de  $n$  varáveis relaciona um conjunto de  $n$  variáveis binárias independentes a uma variável binária dependente. Há várias formas de se representar uma função de chaveamento, entre elas está o uso de expressões de chaveamento.

#### 4.3.1 Representação de funções de chaveamento

Nesta seção são mostradas 4 formas de se representar uma função de chaveamento:

- Representação tabular;
  - Representação por diagrama;
  - Representação por conjunto zero e conjunto um;
  - Representação algébrica.

A representação mais intuitiva para uma função de chaveamento é a representação tabular. A Fig. 4.1a mostra um exemplo arbitrário de uma função de chaveamento de três variáveis  $f(x_2, x_1, x_0)$  e os respectivos valores da função para cada um dos possíveis valores das variáveis independentes  $x_2$ ,  $x_1$  e  $x_0$ . Outra representação bem simples é a representação por diagrama. Na Fig. 4.1b a mesma função mostrada na Fig. 4.1a é representada usando a representação por diagrama.

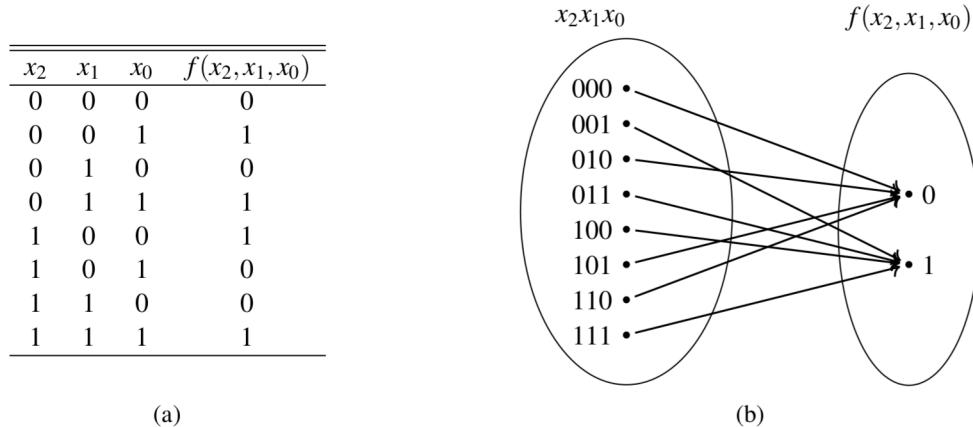


Figura 4.1: Exemplo de representação de funções de chaveamento: (a) representação tabular (b) representação por diagrama.

Como o conjunto imagem e uma função de chaveamento só possui dois elementos (0 e 1) é possível representar a função indicando para quais valores das variáveis de entrada a função tem valor 1 ou, alternativamente, indicando para quais valores das variáveis de entrada a função tem valor 0. Essa é a representação usando o Conjunto-UM() ou o Conjunto-ZERO(). Há duas representações possíveis para a função de 3 variáveis  $f(x_2, x_1, x_0)$  mostrada na Fig. 4.1a. A primeira é utilizando o Conjunto-UM():

$$f(x_2, x_1, x_0) = \text{Conjunto-UM}(001, 011, 100, 111). \quad (4.1)$$

Essa representação informa que se os valores para as variáveis  $x_2x_1x_0$  são iguais a 001, 011, 100 e 111 o valor de  $f(x_2, x_1, x_0)$  é 1. Como  $f(x_2, x_1, x_0)$  é um número binário, os valores de  $x_2x_1x_0$  não informados automaticamente resultam em  $f(x_2, x_1, x_0) = 0$ , i.e.  $f(x_2, x_1, x_0) = 0$  para  $x_2x_1x_0 = 000, 010, 101$  e 110.

Alternativamente, pode ser usando o Conjunto-ZERO() para se descrever a função exemplo mostrada na Fig. 4.1a:

$$f(x_2, x_1, x_0) = \text{Conjunto-ZERO}(000, 010, 101, 110). \quad (4.2)$$

As atribuições de variáveis indicadas dentro dos parêntesis resultam em  $f(x_2, x_1, x_0) = 0$  e as não indicadas resultam em  $f(x_2, x_1, x_0) = 1$ .

A quarta forma de representação de funções de chaveamento é utilizando expressões de chaveamentos. Essa forma é discutida com profundidade nos próximos capítulos deste documento. A título de exemplo, uma possível descrição algébrica para a função  $f(x_2, x_1, x_0)$  mostrada na Fig. 4.1a é:

$$f(x_2, x_1, x_0) = \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot x_1 \cdot x_0 \quad (4.3)$$

#### 4.3.2 Exemplos de funções de chaveamento

Devido a natureza discreta nos conjuntos imagem e domínio de uma função de chaveamento, existem um número limitado dessas funções. A título de exemplo são analisados aqui dois casos:

Tabela 4.3: Todas as possíveis funções de chaveamento de uma variável.

$x$	$f_a(x)$	$f_b(x)$	$f_c(x)$	$f_d(x)$
0	0	0	1	1
1	0	1	0	1
Nome	constante 0	identidade	NOT	constante 1

funções de chaveamento de uma e de duas variáveis. Isso é, respectivamente funções do tipo  $y = f(x)$  e  $y = f(x_1, x_0)$ .

Existem exatamente quatro possíveis funções de chaveamento binárias de uma variável ( $f_a(x)$ ,  $f_b(x)$ ,  $f_c(x)$  e  $f_d(x)$ ). Todas estão mostradas na Tabela 4.3. Na primeira coluna da tabela estão os possíveis valores para  $x$  e nas colunas 2, 3, 4 e 5 estão as quatro combinações possíveis de resultados de funções. Na última linha da tabela está o nome de cada uma das funções.

Existem exatamente 16 possíveis funções de chaveamento binárias de duas variáveis. Todas estão mostradas na Tabela 4.4. Na primeira linha da tabela estão as quatro possíveis combinações de atribuição para as variáveis de  $x_1$  e  $x_0$ : 00, 01, 10 e 11. Cada linha da tabela representa uma função diferente, ou seja, um conjunto de valores de função diferente para cada uma das atribuições mencionadas. Há uma coluna mencionando o nome da função lógica implementada (coluna nome) e uma que mostra a expressão algébrica da respectiva função (coluna expressão algébrica).

Tabela 4.4: Todas as possíveis funções de chaveamento de duas variável.

$f(x_1, x_0)$	$x_1$				Nome	Expressão algébrica
	00	01	10	11		
$f_0$	0	0	0	0		
$f_1$	0	0	0	1	AND	$x_1 \cdot x_0$
$f_2$	0	0	1	0		
$f_3$	0	0	1	1		
$f_4$	0	1	0	0		
$f_5$	0	1	0	1		
$f_6$	0	1	1	0	XOR	$x_1 \oplus x_0 = \overline{x_1} \cdot x_0 + x_1 \cdot \overline{x_0}$
$f_7$	0	1	1	1	OR	$\frac{x_1 + x_0}{x_1 + x_0}$
$f_8$	1	0	0	0	NOR	$\overline{x_1 + x_0}$
$f_9$	1	0	0	1	XNOR	$\overline{x_1 \oplus x_0} = \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0$
$f_{10}$	1	0	1	0		
$f_{11}$	1	0	1	1		
$f_{12}$	1	1	0	0		
$f_{13}$	1	1	0	1		
$f_{14}$	1	1	1	0	NAND	$\overline{x_1 \cdot x_0}$
$f_{15}$	1	1	1	1		

As seis funções lógicas nomeadas na tabela são:

- AND - função E lógico.
- NAND - Complemento da função E, ou seja, NÃO E.
- OR - função OU lógico.
- NOR - Complemento da função OU, ou seja, NÃO OU.

- XOR - função OU EXCLUSIVO lógico.
- XNOR - Complemento da função OU EXCLUSIVO, ou seja, NÃO OU EXCLUSIVO.

Note que o símbolo  $\oplus$  é usado para representar algebricamente o operador XOR. As seis funções nomeadas na tabela também podem ser funções de  $n$  variáveis: AND (4.4), OR (4.5), XOR (4.6), NAND (4.7), NOR (4.8) e XNOR (4.9).

$$x_{n-1} \cdot x_{n-2} \cdot \dots \cdot x_0 = \begin{cases} 0, & \text{se } \exists i \mid x_i = 0 \\ 1, & \text{caso contrário.} \end{cases} \quad (4.4)$$

$$x_{n-1} + x_{n-2} + \dots + x_0 = \begin{cases} 1, & \text{se } \exists i \mid x_i = 1 \\ 0, & \text{caso contrário.} \end{cases} \quad (4.5)$$

$$x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0 = \begin{cases} 1, & \text{se há um número ímpar de } i \mid x_i = 1 \\ 0, & \text{caso contrário.} \end{cases} \quad (4.6)$$

$$\overline{x_{n-1} \cdot x_{n-2} \cdot \dots \cdot x_0} = \begin{cases} 1, & \text{se } \exists i \mid x_i = 0 \\ 0, & \text{caso contrário.} \end{cases} \quad (4.7)$$

$$\overline{x_{n-1} + x_{n-2} + \dots + x_0} = \begin{cases} 0, & \text{se } \exists i \mid x_i = 1 \\ 1, & \text{caso contrário.} \end{cases} \quad (4.8)$$

$$\overline{x_{n-1} \oplus x_{n-2} \oplus \dots \oplus x_0} = \begin{cases} 0, & \text{se há um número ímpar de } i \mid x_i = 1 \\ 1, & \text{caso contrário.} \end{cases} \quad (4.9)$$

## 4.4 Portas lógicas

Em circuitos eletrônicos digitais os principais operadores lógicos listados nas seções anteriores deste capítulo são implementados em *hardware* usando portas lógicas. As portas lógicas são circuitos eletrônicos físicos implementados em circuitos integrados. A Fig. 4.2 ilustra duas aparências típicas de um circuito integrado. Na Fig. 4.3 está mostrada uma placa mãe de um dos primeiros computadores eletrônicos fabricados com dezenas de circuitos integrados que implementam portas lógicas.

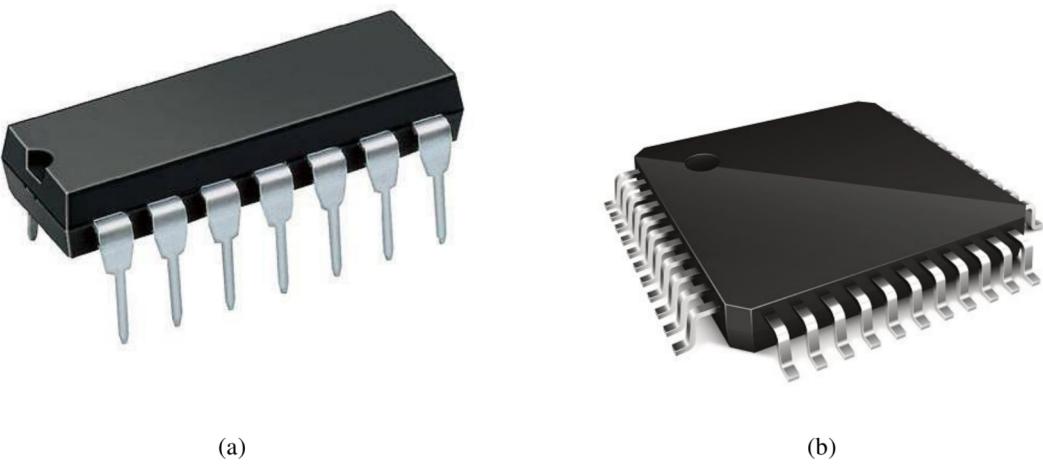


Figura 4.2: Exemplos de circuitos integrados em dois tipos de encapsulamento: (a) encapsulamento DIP (b) encapsulamento SMD.

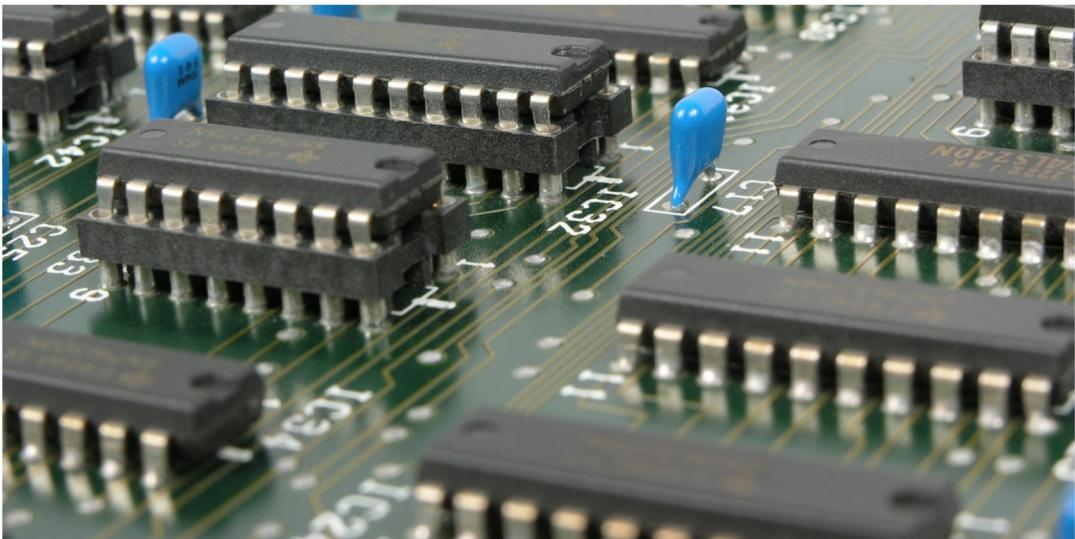


Figura 4.3: Exemplo de uma placa mãe de computador usando circuitos digitais discretos.

Dependendo da tecnologia empregada, um circuito integrado pode conter entre algumas unidades de portas lógicas até dezenas de milhões delas.

Cada operador implementado por uma porta lógica tem um símbolo específico. Em esquemas elétricos os símbolos usados para as portas lógicas estão mostrados na Tabela 4.5. Na tabela estão mostrados os símbolos para operadores de uma e duas variáveis (ou  $x$  ou  $x_0$  e  $x_1$ ). No entanto, cada uma das portas lógicas mostradas (excetuando-se a porta NOT) pode conter duas ou mais entradas.

Tabela 4.5: Listagem das portas lógicas que implementam funções básicas de chaveamento.

Função	Símbolo	Expressão
NOT		$y = x$
AND		$y = x_0 \cdot x_1$
OR		$y = x_0 + x_1$
XOR		$y = x_0 \oplus x_1$
NAND		$y = \overline{x_0 \cdot x_1}$
NOR		$y = \overline{x_0 + x_1}$
XNOR		$y = \overline{x_0 \oplus x_1}$

■ **Exemplo 4.1** Implemente a função:

$$f(x_2, x_1, x_0) = \overline{x_2} \cdot x_1 \cdot \overline{x_0} + x_2 \cdot x_1 \cdot \overline{x_0} + \overline{x_2} \cdot \overline{x_1} \cdot x_0 + \overline{x_2} \cdot x_1 \cdot x_0 \quad (4.10)$$

usando portas lógicas.

A ideia é escrever a expressão algébrica em termos de circuitos eletrônicos representados por portas lógicas. Substituindo os ORs, ANDs e NOTs da expressão fornecida chega-se a implementação mostrada na Fig. 4.4.

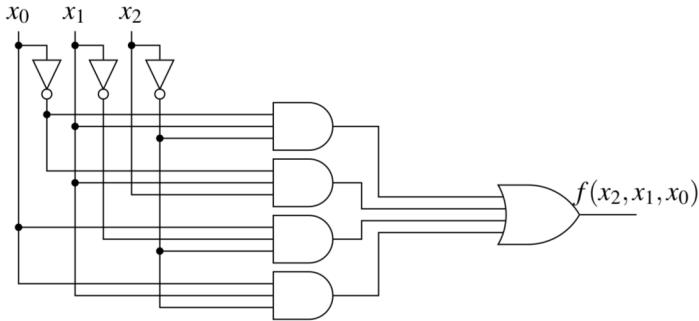


Figura 4.4: Implementação de (4.10) usando portas lógicas.

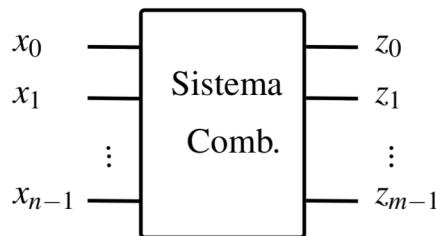


Figura 4.5: Diagrama de blocos de um sistema combinacional com  $n$  entradas  $x_0, x_1, \dots, x_{n-1}$  e  $m$  saídas  $z_0, z_1, \dots, z_{m-1}$ .

#### 4.5 Modelando circuitos combinacionais com funções de chaveamento

Um sistema combinacional genérico pode ser representado pelo bloco mostrado na Fig. 4.5. A representação mostrada também pode ser referida como representação em diagrama de bloco do sistema. Nele, há um conjunto de  $n$  entradas e de  $m$  saídas binárias. A funcionalidade implementada pelo sistema pode ser modelada matematicamente por um conjunto de  $m$  funções de chaveamento binárias. Cada uma das  $m$  funções possuem  $n$  variáveis independentes. Essas funções podem ser implementadas em *hardware* usando portas lógicas.

Escrevendo matematicamente, a  $i$ -ésima saída do sistema ( $z_i$ ) pode ser representada algebraicamente por uma função chaveamento de  $n$  variáveis do tipo:

$$z_i = f_i(x_{n-1}, x_{n-2}, \dots, x_0) \quad i \in \{0, 1, \dots, m-1\}. \quad (4.11)$$

Um dos principais temas abordados neste documento é a responder a questão: Dados os requisitos de um circuito digital (*i.e.* tarefa que o circuito deve realizar), como obter as funções  $f_i(x_{n-1}, x_{n-2}, \dots, x_0)$  de forma que elas atendam aos requisitos solicitados usando a menor quantidade possível de portas lógicas?

#### 4.6 Funções não completamente especificadas

Como discutido na Seção 4.3 o conjunto imagem de um função de chaveamento possui apenas dois valores 0 e 1. Então, o valor de uma função  $f_i(x_{n-1}, x_{n-2}, \dots, x_0)$  de  $n$  variáveis possui, para cada atribuição das variáveis independentes, um valor determinado, 0 ou 1. No entanto, algumas vezes é conveniente deixar um ou mais desses valores de resposta da função em aberto, não especificado.

Considere dois sistemas que trabalham com números binários de três bits: o sistema  $S_1$  que gera números primos em sua saída, e o sistema  $S_2$  que verifica se o valor constante na entrada é um número par (saída igual a 1) ou um número ímpar (saída igual a 0). Note que sistemas binários de três bits trabalham com números desde 0 (000) até 7 (111). Imagine que a saída de  $S_1$  se liga a entrada de  $S_2$ . Nesse cenário, fica claro que  $S_2$  nunca receberá em suas entradas os valores não primos 0, 1, 4 e 6. Sendo assim, como essas entradas são impossíveis de acontecer para  $S_2$  no cenário montado é possível deixar sem especificar os valores da função de chaveamento que rege  $S_2$  para esse valores conforme mostrado na Tabela 4.6. Deixar valores não especificados pode gerar redução no hardware necessário para implementar uma função.

Se problema tratado permitir saídas (valor da função de chaveamento) não especificadas, essas DEVEM ser mencionadas na tabela verdade/especificação do sistema e devem ser expressas usando

Tabela 4.6: Exemplo de uma função não completamente especificada.

$x_2$	$x_1$	$x_0$	$f(x_2, x_1, x_0)$
0	0	0	-
0	0	1	-
0	1	0	1
0	1	1	0
1	0	0	-
1	0	1	0
1	1	0	-
1	1	1	0

“-” ou “x” na tabela verdade. Esses valores não especificados também são conhecidas como valores “*don't care*” (ou DC).

## 4.7 Expressões equivalentes

A principal aplicação das funções de chaveamento do escopo desse documento é prover uma descrição matemática que expresse corretamente a relação entrada e saída desejada para um sistema, no caso, um circuito eletrônico.

Ao se elencar os requisitos para essa relação entrada/saída do sistema chega-se a expressões algébricas e funções de chaveamento que modelam essas relações. Uma preocupação relevante é implementar essas funções como menor número possível de elementos de *hardware*.

A aplicação das regras da álgebra de boole pode ser uma ferramenta muito eficaz nesse sentido e conseguir, em muitos casos, reduzir bastante o quantidade/complexidade do *hardware* necessário para se implementar um circuito eletrônico.

Essas preocupações são relevantes por que, numa álgebra booleana binária a mesma função pode ser escrita por várias expressões distintas algebraicamente falando mas que implementam a mesma tabela verdade. Sendo assim define-se:

**Definição 4.7.1** (Expressões equivalentes) Duas expressões  $E_x$  e  $E_y$  são ditas equivalentes se ambas podem ser expressas pela mesma tabela verdade.

O Exemplo 4.2 ilustra a idéia das expressões equivalentes.

■ **Exemplo 4.2** Imagine as seguintes expressões:

$$E_1(x_2, x_1, x_0) = x_2 \cdot x_1 + x_2 \cdot \overline{x_1} + x_2 \cdot x_0 \quad (4.12)$$

e

$$E_2(x_2, x_1, x_0) = x_2. \quad (4.13)$$

Como fica claro,  $E_1$  e  $E_2$  possuem expressões algébricas diferentes. No entanto, aplicando as regras

da álgebra de boole na expressão  $E_2$  obtém-se:

$$\begin{aligned}
 E_1(x_2, x_1, x_0) &= x_2 \cdot x_1 + x_2 \cdot \bar{x}_1 + x_2 \cdot x_0 \\
 &= x_2 \cdot (x_1 + \bar{x}_1) + x_2 \cdot x_0 \\
 &= x_2 + x_2 \cdot x_0 \\
 &= x_2.
 \end{aligned} \tag{4.14}$$

Ou seja, ambas  $E_1$  e  $E_2$  possuem a mesma tabela verdade porém expressões algébricas diferentes. Ao se implementar com portas lógicas a expressão  $E_1$  obtém-se o circuito mostrado na Fig. 4.6a enquanto para se implementar a expressão  $E_2$  obtém-se o circuito mostrado na Fig. 4.6b. Como ambas os circuitos implementam a mesma função lógica, *i.e.* a mesma tabela verdade, claramente a implementação mostrada na Fig. 4.6b é muito mais conveniente pois é muito mais simples.

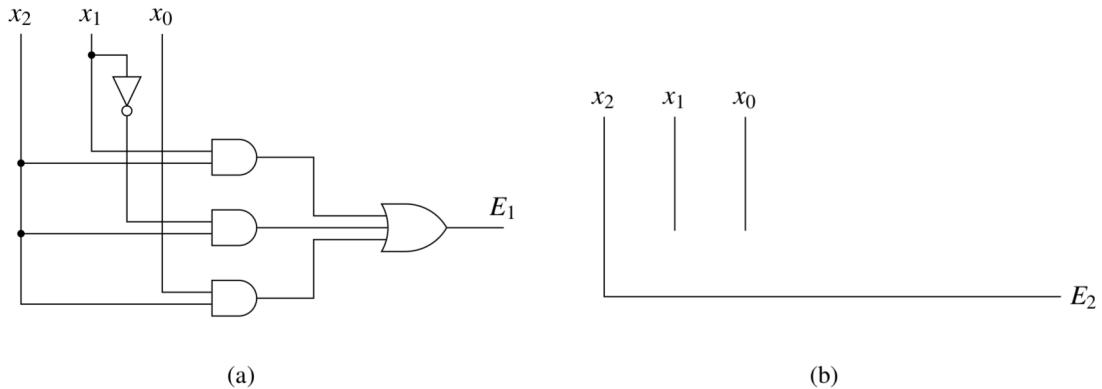


Figura 4.6: Exemplo expressões de chaveamento equivalentes: (a) implementação com portas lógicas da função  $E_1(x_2, x_1, x_0) = x_2 \cdot x_1 + x_2 \cdot \bar{x}_1 + x_2 \cdot x_0$  (b) implementação da função  $E_2(x_2, x_1, x_0)$  equivalente à  $E_1(x_2, x_1, x_0)$ .

## 4.8 Problemas propostos

**Problema 4.1** Dadas as representações algébricas das funções de chaveamento encontre suas representações usando as formas tabular, por Conjunto-UM() e por Conjunto-ZERO().

- a)  $f(x_1, x_0) = \bar{x}_1 \cdot x_0 + x_1 \cdot \bar{x}_0$
- b)  $f(x_2, x_1, x_0) = x_2 \cdot x_1 \cdot x_0 + \bar{x}_1 \cdot \bar{x}_0$
- c)  $f(x_3, x_2, x_1, x_0) = (x_3 + x_2 + x_1 + x_0) \cdot (\bar{x}_3 + x_2 + x_1 + \bar{x}_0)$

**Problema 4.2** Implemente as funções de chaveamento usando portas lógicas. Lembre que portas lógicas com múltiplas entradas podem ser usadas.

- a)  $f(x_1, x_0) = \bar{x}_1 \cdot x_0 + x_1 \cdot \bar{x}_0 + x_1 \cdot x_0$
- b)  $f(x_2, x_1, x_0) = (\bar{x}_2 + x_1 + \bar{x}_0) \cdot (x_2 + x_1 + x_0) \cdot (x_2 + \bar{x}_1 + \bar{x}_0)$
- c)  $f(x_3, x_2, x_1, x_0) = \bar{x}_3 \cdot x_2 \cdot x_1 \cdot x_0 + x_3 \cdot \bar{x}_2 \cdot x_1 \cdot x_0 + x_3 \cdot x_2 \cdot \bar{x}_1 \cdot x_0 + x_3 \cdot x_2 \cdot x_1 \cdot x_0$

**Problema 4.3** Considere o sistema  $S$  como um *bit* de saída  $z$  e 4 bits de entrada  $\underline{x}$ . O sistema  $S$  é um sistema de cálculo de paridade que fornece em sua saída  $z$ :

$$z = \begin{cases} 0, & \text{se } \underline{x} \text{ é par} \\ 1, & \text{se } \underline{x} \text{ é ímpar.} \end{cases} \quad (4.15)$$

Sobre o sistema  $S$  faça o que se pede:

- a) Esquematize em termos de diagrama de blocos o sistema  $S$ .
- b) Encontre a representação da função de chaveamento que modela o funcionamento do sistema usando as representações de tabela verdade, Conjunto-UM() e Conjunto-ZERO().
- c) É possível implementar  $S$  usando portas lógicas do tipo XOR de 4 entradas? Caso positivo forneça a implementação.
- d) É possível implementar  $S$  usando portas lógicas do tipo XOR de 2 entradas? Caso positivo forneça a implementação.

**Problema 4.4** Considere o sistema binário que realiza soma  $Z$  de dois operandos  $X$  e  $Y$ . Considere que  $X, Y \in \{0, 1, 2, 3\}$ .

- a) Esquematize em termos de diagrama de blocos o sistema.
- b) Encontre a representação da função de chaveamento que modela o funcionamento do sistema usando as representações de tabela verdade, Conjunto-UM() e Conjunto-ZERO().

**Problema 4.5** Implemente cada uma das funções listadas usando portas lógicas (implementação 1). Em seguida, aplique as regras da álgebra de boole, estudadas no Capítulo 2, nas funções e obtenha expressões equivalentes reduzidas. Refaça a implementação (implementação 2) das expressões reduzidas usando portas lógicas. Faça a contabilidade e compare os resultados de quantas portas lógicas foram usadas nas implementações 1 e 2.

- a)  $f(x_1, x_0) = (\overline{x_1} + x_0) \cdot (x_1 + x_0)$
- b)  $f(x_2, x_1, x_0) = \overline{x_2} \cdot x_1 \cdot x_0 + x_2 \cdot x_1 \cdot x_0 + x_1 \cdot \overline{x_0} + x_2 \cdot x_1$
- c)  $f(x_3, x_2, x_1, x_0) = x_3 \cdot \overline{x_2} \cdot x_1 \cdot \overline{x_0} + x_3 \cdot \overline{x_2} \cdot x_1 \cdot x_0 + x_3 \cdot \overline{x_2} \cdot x_1 + x_3 \cdot \overline{x_2} \cdot \overline{x_1}$

**Problema 4.6** Considere o sistema  $S$  que controla a abertura automática de portas de um trem de metrô. Esse sistema de metrô possui 3 estações. Os sensores  $E_1, E_2$  e  $E_3$  são colocados, respectivamente nas estações 1, 2 e 3. De tal forma que:

$$E_n = \begin{cases} 1, & \text{se o trem está perfeitamente alinhado na estação } n \\ 0, & \text{caso contrário.} \end{cases} \quad (4.16)$$

No trem está instalado um sensor de movimento  $M$  de tal forma que:

$$M = \begin{cases} 1, & \text{se o trem está em movimento} \\ 0, & \text{se o trem está parado.} \end{cases} \quad (4.17)$$

O sinal de comando  $P$  para a abertura de portas do trem é tal que:

$$P = \begin{cases} 1, & \text{porta deve abrir} \\ 0, & \text{porta deve fechar} \end{cases} \quad (4.18)$$

Sobre o sistema  $S$  faça o que se pede:

- a) Esquematize em termos de diagrama de blocos o sistema  $S$ .
- b) Encontre a tabela verdade do sistema  $S$ .