Capítulo 3

Um Cálculo Axiomático para Bits

Los locos no están locos si uno acepta su razonamiento.
Gabriel García Márquez

Chamamos de *sinais* os valores transportados pelas conexões entre os componentes dos circuitos. Nos interessam os sinais que podem assumir um de dois possíveis valores, 0 e 1, e que variam no tempo. Tais sinais são chamados de *sinais digitais*, em contraste com os *sinais analógicos*, que podem assumir quaisquer níveis de um intervalo limitado por valores máximo e mínimo. Voltagem e temperatura são sinais analógicos enquanto que o estado de uma lâmpada (de LEDs) – acesa ou apagada – ou de uma fechadura – aberta ou fechada – é digital.

Este capítulo[1] define uma abstração para *sinais digitais* que permite tratá-los como se fossem entidades matemáticas simples. A abstração é baseada na Álgebra de Boole e é relevante para projetistas de circuitos porque simplifica enormemente a compreensão dos conceitos, bem como a manipulação algébrica dos operadores que relacionam logicamente os sinais.

A abstração de sinais elétricos como *bits*, para representar o comportamento de circuitos digitais, é útil se encarada como uma simplificação e idealização do comportamento dos sinais e das relações entre eles nos circuitos que implementam as funções lógicas. A abordagem empregada neste texto é inspirada em Sanders [San90] e o formalismo para especificar circuitos é baseado na linguagem formal Z [Spi89, Mou01].

Empregamos os símbolos ' $\wedge$ ' para representar a conjunção e ' $\vee$ ' para representar a disjunção para não sobrecarregar os operadores '$\cdot$' e '+', e mais importante, para diferenciar operações sobre valores lógicos de operações aritméticas sobre valores numéricos.

3.1 Abstração de Sinais como Bits

A Figura 3.1 mostra quatro possíveis representações para um mesmo sinal que varia com o tempo. A *representação concreta* tenta reproduzir o comportamento elétrico de um sinal, como seria observado diretamente num circuito em operação. O sinal analógico varia continuamente entre V_M e V_m, e esta forma de onda aparenta estar contaminada por ruído gerado pelos circuitos próximos e pelo comportamento elétrico real do circuito que gera o sinal.

[1] © Roberto André Hexsel, 2012-2021. Versão de 1 de fevereiro de 2021.

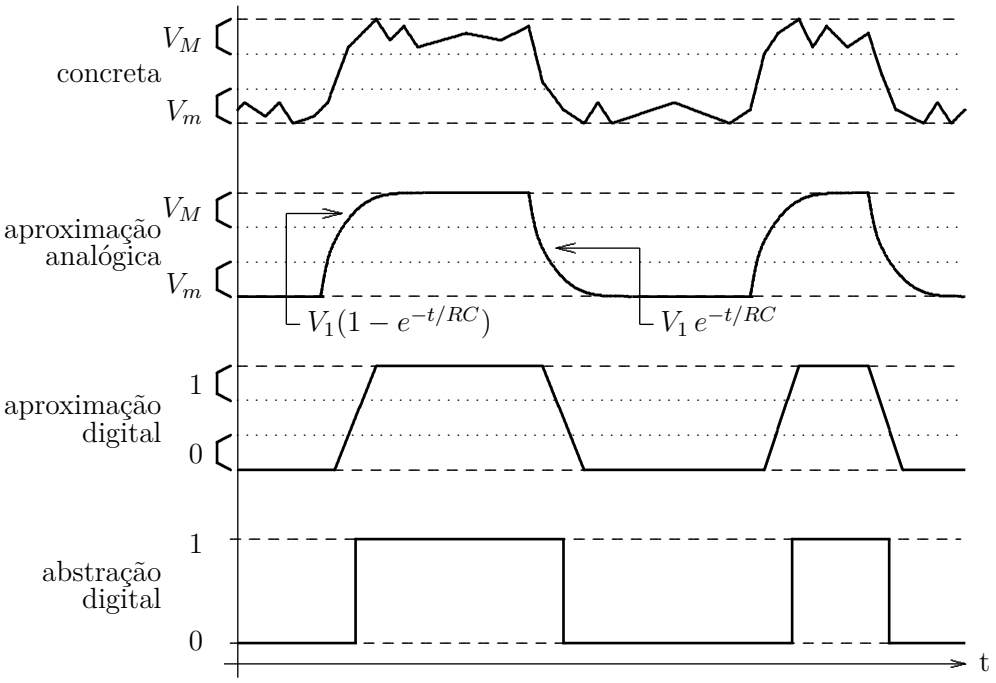

Figura 3.1: Quatro representações para o comportamento de um sinal.

Se o circuito que gera o sinal for simulado num "simulador analógico", como o programa `spice`, o sinal teria uma forma de onda como a mostrada na segunda curva, que é uma *aproximação analógica* para o comportamento real. Neste tipo de simulação, equações diferenciais modelam correntes e tensões no circuito, e as mudanças de nível são descritas por funções do tempo com exponenciais, tais como $V_1(t) = V_1(1 - e^{-t/RC})$, para constantes R e C relacionadas aos parâmetros elétricos de resistência e capacitância dos circuitos nos quais a tensão é avaliada. Voltaremos a este assunto no Capítulo 5.

As duas últimas curvas são idealizações, ou *representações abstratas* para o comportamento real. Na chamada *aproximação digital*, as exponenciais são aproximadas por retas que indicam que a mudança do nível mínimo, que representa 0, para o nível máximo, que representa 1, não é instantânea – o instante do início não é o mesmo do final da transição. Na *abstração digital*, as transições entre 0 e 1 são instantâneas e não há ambiguidade na interpretação dos níveis lógicos do sinal.

Nas representações concreta e analógica, entre as faixas de tensão que correspondem aos níveis lógicos 1 e 0, há uma faixa de transição em que o sinal é indeterminado, porque não é nem 1 nem 0. Os projetistas dos circuitos tentam fazer com que a duração dos intervalos em que os sinais estão indeterminados seja a menor possível.

A aproximação digital é usada em alguns dos diagramas de tempo mostrados ao longo do texto para lembrar ao leitor de que as mudanças de nível dos sinais não são instantâneas. A representação idealizada e abstrata, na qual os sinais elétricos são representados por *bits*, é empregada para descrever o comportamento dos circuitos, quando aqueles estão em estado de repouso, após cessarem os efeitos de todas as transições nos sinais.

Espaço em branco proposital.

3.2 Bits, Operações e Propriedades

Definição 3.1 (Bits 0 e 1) *O conjunto* $\mathbb{B}$ *tem dois elementos distintos 0 e 1 chamados de* bits, $\mathbb{B} = \{0, 1\}$.

Definição 3.2 (Tupla) *Uma* tupla de k bits *é um produto cartesiano com* k *parcelas:*

$$\mathbb{B}^k = \overbrace{\mathbb{B} \times \mathbb{B} \times \cdots \times \mathbb{B}}^{\text{k parcelas}}$$

Uma tupla de bits pode ser representada pela mera justaposição de bits, como 1 ou 1010, ou com constantes ou variáveis agrupadas por colchetes angulares como $\langle a, b, c \rangle$.

Definição 3.3 (Tipo de Função) *O* tipo de uma função *com domínio* $\mathcal{D}$ *e imagem* $\mathcal{I}$ *é denotado por*

$$\mathcal{D} \mapsto \mathcal{I}$$

e o tipo de uma função f *com domínio* $\mathcal{D}$ *e imagem* $\mathcal{I}$ *é especificado por*

$$f : \mathcal{D} \mapsto \mathcal{I}$$

que é lido "f é uma função que mapeia elementos de $\mathcal{D}$ em $\mathcal{I}$*". O operador* $\mapsto$ *indica o mapeamento, ou a associação, entre o conjunto dos argumentos e o conjunto dos valores da função.*

Definição 3.4 (Operador Unário) *Um* operador f_1 *sobre bits é chamado de* unário *se tem um bit como argumento e produz um bit como resultado:*

$$f_1 : \mathbb{B} \mapsto \mathbb{B}$$

Definição 3.5 (Operador Binário) *Um* operador f_2 *sobre bits é chamado de* binário *se tem dois bits como argumento e produz um bit como resultado:*

$$f_2 : \mathbb{B}^2 \mapsto \mathbb{B}$$

O subscrito no nome da função indica a *aridade* de um operador, ou o "número de entradas", é geralmente omitido porque tal informação pode ser facilmente depreendida do contexto no qual o operador é usado.

Espaço em branco proposital.

Definição 3.6 (Conjunção, Disjunção, Complemento) *Três operadores sobre bits, chamados de* conjunção, disjunção *e* complemento*, são definidos na Tabela 3.1, e são denotados por* $\wedge$, $\vee$ *e* $\neg$, *respectivamente.*

Tabela 3.1: Definição dos operadores conjunção, disjunção e complemento.

conjunção

a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

disjunção

c	d	$c \vee d$
0	0	0
0	1	1
1	0	1
1	1	1

complemento

e	$\neg e$
0	1
1	0

O resultado de uma conjunção é 1 se *todos* os argumentos são 1. O resultado de uma disjunção é 1 se *algum* dos argumentos é 1. A disjunção é também chamada de *ou-inclusivo* porque além da condição "um mas não o outro", a condição "um mais o outro" produz também resultado 1. A negação é também representada por $\overline{a} = \neg a$. Os operadores *conjunção*, *disjunção* e *complemento* são funções.

A tabela para o complemento mostra *todas* as combinações dos valores possíveis para o argumento $e \in \mathbb{B}$, quais sejam 0 e 1. As tabelas para a conjunção e a disjunção contém uma linha para cada uma de *todas* as combinações possíveis dos dois argumentos. Para a conjunção, as combinações possíveis para o par $a, b \in \mathbb{B}$ são $\{(0,0),(0,1),(1,0),(1,1)\}$.

Definição 3.7 (Variável em $\mathbb{B}$) *Uma* variável *é um símbolo, ou um nome, que representa um valor em* $\mathbb{B}$.

Definição 3.8 (Expressão em $\mathbb{B}$) *Uma* expressão *representa um valor em* $\mathbb{B}$ *e consiste de variáveis relacionadas por operadores.*

Uma variável pode conter um valor que é 1 ou 0, sendo ela própria uma expressão. A expressão que representa a informação "p é 1 se ambos q e r são também 1" pode ser escrita como $p = q \wedge r$ e é lida como "p é a conjunção de q e de r", ou ainda "p é 1 se todos q e r são 1".

Definição 3.9 (Precedência) *A* precedência *da aplicação dos operadores é* $\neg$, $\wedge$ *e* $\vee$, *e esta ordem pode ser alterada com o uso de parênteses.*

Definição 3.10 (Valor de uma Função) *Quando valores em* $\mathbb{B}$ *são atribuídos a todas as variáveis de uma expressão* $\mathcal{E}$, *o resultado da avaliação desta expressão é chamado de o* valor da função $\mathcal{E}$.

Definição 3.11 (Igualdade) *Duas expressões* $\mathcal{E}(a, b, \cdots, z)$ *e* $\mathcal{F}(a, b, \cdots, z)$ *são ditas* iguais *se, ao atribuir todas as combinações de valores em* $\mathbb{B}$ *às variáveis* $a, b, \cdots, z$, *então para todas as combinações das entradas obtém-se* $\mathcal{E}(a, b, \cdots, z) = \mathcal{F}(a, b, \cdots, z)$.

Propriedades dos operadores Os operadores sobre bits exibem as propriedades listadas abaixo. Como definidos na Tabela 3.1, os operadores conjunção e disjunção são funções binárias; o operador complemento é uma função unária.

Identidade (i) A *identidade* com relação a $\vee$ é chamada de 0:

$$\exists\, 0 \in \mathbb{B} \;\bullet\; \forall\, a \in \mathbb{B} \;\bullet\; a \vee 0 = a$$

(ii) A *identidade* com relação a $\wedge$ é chamada de 1:

$$\exists\, 1 \in \mathbb{B} \;\bullet\; \forall\, a \in \mathbb{B} \;\bullet\; a \wedge 1 = a$$

Máximo e Mínimo O *elemento máximo* de $\mathbb{B}$ é 1, e o *elemento mínimo* de $\mathbb{B}$ é 0:

$$\forall\, a \in \mathbb{B} \;\bullet\; \begin{array}{l} a \vee 1 = 1, \\ a \wedge 0 = 0 \end{array}$$

Complemento A conjunção (disjunção) de a com seu complemento resulta no elemento mínimo (máximo) de $\mathbb{B}$:

$$\forall\, a \in \mathbb{B} \;\bullet\; \exists\, \neg a \in \mathbb{B} \bullet \begin{array}{l} a \wedge \neg a = 0, \\ a \vee \neg a = 1 \end{array}$$

Fechamento As operações sobre bits resultam em bits:

$$\forall\, a, b \in \mathbb{B} \;\bullet\; (a \wedge b) \in \mathbb{B}, \quad (a \vee b) \in \mathbb{B}, \quad \neg a \in \mathbb{B}$$

Involução O complemento do complemento de $a \in \mathbb{B}$ é o próprio a:

$$\forall\, a \in \mathbb{B} \;\bullet\; \neg(\neg a) = a$$

Idempotência A conjunção, e a disjunção, de a com a é o próprio a:

$$\forall\, a \in \mathbb{B} \;\bullet\; \begin{array}{l} a \wedge a = a, \\ a \vee a = a \end{array}$$

Comutatividade A ordem dos operandos não altera o resultado:

$$\forall\, a, b \in \mathbb{B} \bullet \begin{array}{l} a \wedge b = b \wedge a, \\ a \vee b = b \vee a \end{array}$$

Associatividade A ordem de aplicação de um mesmo operador não altera o resultado:

$$\forall\, a, b, c \in \mathbb{B} \;\bullet\; \begin{array}{l} (a \wedge b) \wedge c = a \wedge (b \wedge c), \\ (a \vee b) \vee c = a \vee (b \vee c) \end{array}$$

Distributividade O operador conjunção distribui sobre a operação de disjunção, e o operador disjunção distribui sobre a operação de conjunção:

$$\forall\, a, b, c \in \mathbb{B} \;\bullet\; \begin{array}{l} a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c), \\ a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \end{array}$$

Teorema de DeMorgan O complemento de uma conjunção é a disjunção das entradas complementadas, e o complemento de uma disjunção é a conjunção das entradas complementadas.

$$\begin{array}{l} \forall\, a, b \in \mathbb{B} \;\bullet\; \neg(a \wedge b) = (\neg a \vee \neg b) \\ \qquad\qquad\quad \neg(a \vee b) = (\neg a \wedge \neg b) \end{array}$$

Definição 3.12 (Dual) *Define-se o* dual *de uma expressão sobre bits como a expressão que é obtida pela troca simultânea de todas as ocorrências de $\wedge$ por $\vee$, de todas as ocorrências de $\vee$ por $\wedge$, e de todas as ocorrências das constantes 0 por 1, e de 1 por 0.*

Definição 3.13 (Princípio da Dualidade) *O dual de uma propriedade é também uma propriedade.*

O resultado da aplicação de um operador binário a um par de vetores de bits é determinado pela Definição 3.15.

Definição 3.14 (Elemento de Tupla) *Seja uma tupla A com n elementos de $\mathbb{B}$; o i-ésimo elemento da tupla é denotado por a_i, para $0 \leq i < n$.*

Definição 3.15 (Aplicação Bit a Bit) *A aplicação de uma função binária $f_2 : \mathbb{B}^2 \mapsto \mathbb{B}$, bit a bit, aos elementos de um par de tuplas $A, B \in \mathbb{B}^k$ produz a tupla $C \in \mathbb{B}^k$, tal que*

$$\forall i,\, 0 \leq i < k \;\bullet\; c_i = f_2(a_i, b_i)$$

Exemplo 3.1 Considere o vetor $A = \langle 0,0,0,1,1,1,0,0\rangle$, representado mais concisamente por $A =$ `0001.1100`, e o vetor $B = \langle 0,1,0,1,1,0,0,1\rangle$, ou $B =$ `0101.1001`. O "ponto de milhar" é usado para agrupar quartetos de bits.

O vetor $C = A \wedge B$ resulta da aplicação, bit a bit, do operador $\wedge$

$$c_7 = a_7 \wedge b_7,\; c_6 = a_6 \wedge b_6,\; c_5 = a_5 \wedge b_5,\; \cdots,\; c_1 = a_1 \wedge b_1,\; c_0 = a_0 \wedge b_0$$

ou

índice	7	6	5	4	3	2	1	0
A	0	0	0	1	1	1	0	0
	$\wedge$	$\wedge$	$\wedge$	$\wedge$	$\wedge$	$\wedge$	$\wedge$	$\wedge$
B	0	1	0	1	1	0	0	1
$A \wedge B = C$	0	0	0	1	1	0	0	0

e portanto $C =$ `0001.1000`. Frequentemente, operações como esta são usadas para isolar um segmento do vetor de bits. Neste caso, $A \wedge B$ isola a tripla $\langle b_4, b_3, b_2\rangle$ ao fazer com que os demais elementos do vetor B sejam todos convertidos para 0 na atribuição à C. O vetor A é chamado de 'máscara' porque os bits em 0 do vetor A 'mascaram' os valores de B nas posições cujos valores são 0. ◁

A Definição 3.16 permite estender um operador binário para aplicação sobre tuplas com mais do que dois argumentos.

Definição 3.16 (Redução) *Se $f_2 : \mathbb{B}^2 \mapsto \mathbb{B}$ é associativa, então $f_k : \mathbb{B}^k \mapsto \mathbb{B}$ é a extensão de f_2 para k entradas, e é obtida por repetidas aplicações de f_2*

$$f_k(b_0, b_1, \cdots, b_{k-1}) = f_2(b_0, f_{k-1}(b_1, \cdots, b_{k-1}))$$

O resultado da aplicação de f_k à tupla $B = \langle b_0, \cdots, b_{k-1}\rangle$ é chamado de *redução de B por f*. O somatório de uma série de números S é a redução por soma de S. O produtório dos elementos de S é a redução pelo produto.

Exemplo 3.2 Suponha que um vetor de bits é usado para contar a população de uma certa região: se o i-ésimo elemento do vetor é 1, então a i-ésima área está ocupada, do contrário aquela área está despopulada. Podemos usar uma redução para descobrir se a região está vazia: $v = 0$ somente se nenhuma das k áreas estiver ocupada: $v = \vee_k(a_0, a_1, \cdots, a_{k-1})$. ◁

Definição 3.17 (Tabela Verdade) *A* tabela verdade *da expressão $\mathcal{E}(a, b, \cdots, z)$ é construída pela atribuição de valores em $\mathbb{B}$ a todas combinações das variáveis e pela avaliação dos operadores em $\mathcal{E}$.*

Para uma expressão com n variáveis as linhas da tabela são enumeradas de 0 a $2^n - 1$. Por convenção, as variáveis são dispostas em ordem alfabética, de tal forma que a letra mais próxima de 'a' tenha peso de 2^{n-1}, e a mais próxima de 'z' tenha peso 2^0. Para a expressão $\mathcal{E}(a, b, c)$, a vale 2^2, b vale 2^1 e c vale 2^0. Para atribuir valores às combinações de variáveis, quando uma variável está complementada ela é considerada como 0, do contrário ela é considerada como 1. As linhas de número 0, 3 e 5 correspondem às triplas $\langle \overline{a}\overline{b}\overline{c}\rangle = \langle 000\rangle$, $\langle \overline{a}bc\rangle = \langle 011\rangle$ e $\langle a\overline{b}c\rangle = \langle 101\rangle$, respectivamente.

Espaço em branco proposital.

A Tabela 3.1 define o resultado das operações binárias conjunção, disjunção e complemento para todas as combinações possíveis dos argumentos. Uma *tabela verdade* pode ser usada para comprovar as propriedades dos operadores. Vejamos como comprovar a comutatividade da conjunção. A Tabela 3.2 mostra os três passos necessários para a comprovação. A última linha da tabela indica em qual passo os valores de cada coluna foram gerados; a coluna do último passo, com o maior número, contém a comprovação da igualdade, se for toda preenchida com 1, ou sua refutação, se algum dos valores for 0. A tabela verdade de uma expressão com n variáveis tem 2^n linhas.

Tabela 3.2: Construção da tabela verdade para $a \wedge b = b \wedge a$.

(1) atribuição dos valores às variáveis

a	b	a	$\wedge$	b	$=$	b	$\wedge$	a
0	0	0		0		0		0
0	1	0		1		1		0
1	0	1		0		0		1
1	1	1		1		1		1
passo		1		1		1		1

(2) cômputo das conjunções

a	b	a	$\wedge$	b	$=$	b	$\wedge$	a
0	0	0	**0**	0		0	**0**	0
0	1	0	**0**	1		1	**0**	0
1	0	1	**0**	0		0	**0**	1
1	1	1	**1**	1		1	**1**	1
passo		1	2	1		1	2	1

(3) verificação da igualdade

a	b	a	$\wedge$	b	$=$	b	$\wedge$	a
0	0		0		**1**		0	
0	1		0		**1**		0	
1	0		0		**1**		0	
1	1		1		**1**		1	
passo		1	2	1	3	1	2	1

Os valores das variáveis são atribuídos aos argumentos dos operadores no passo (1); no passo (2) os resultados das conjunções são inseridos na coluna dos operadores $\wedge$ (mostrados em negrito). No passo (3), os resultados das conjunções são comparados, linha a linha. Se iguais, **1** é inserido na coluna da igualdade, ou **0** caso sejam diferentes. Uma coluna de igualdade preenchida somente com **1**s comprova a propriedade – se os dois lados da igualdade são idênticos em todas as linhas, então a igualdade é válida para todas as combinações de valores das variáveis. Os valores de a e b foram omitidos no último passo para facilitar a visualização.

Exemplo 3.3 Comprovemos a distributividade: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$. A Tabela 3.3 mostra os três primeiros passos da comprovação. Como a propriedade envolve três variáveis, a tabela tem $2^3 = 8$ linhas.

No passo (2), são computados os valores das operações dentro dos parênteses, e no passo (3) são efetuadas as operações fora dos parênteses. Na terceira tabela, os valores desnecessários foram omitidos para facilitar a visualização. Para comprovar a propriedade, no quarto passo basta verificar que as duas colunas geradas no passo 3 são idênticas. ◁

Tabela 3.3: Tabela verdade para $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

(1)atribuição dos valores às variáveis

a	b	c	a	$\vee$	$(b$	$\wedge$	$c)$	$=$	$(a$	$\vee$	$b)$	$\wedge$	$(a$	$\vee$	$c)$
0	0	0	0		0		0		0		0		0		0
0	0	1	0		0		1		0		0		0		1
0	1	0	0		1		0		0		1		0		0
0	1	1	0		1		1		0		1		0		1
1	0	0	1		0		0		1		0		1		0
1	0	1	1		0		1		1		0		1		1
1	1	0	1		1		0		1		1		1		0
1	1	1	1		1		1		1		1		1		1
	passo		1		1		1		1		1		1		1

(2) operações dentro dos parênteses

a	b	c	a	$\vee$	$(b$	$\wedge$	$c)$	$=$	$(a$	$\vee$	$b)$	$\wedge$	$(a$	$\vee$	$c)$
0	0	0	0		0	**0**	0		0	**0**	0		0	**0**	0
0	0	1	0		0	**0**	1		0	**0**	0		0	**1**	1
0	1	0	0		1	**0**	0		0	**1**	1		0	**0**	0
0	1	1	0		1	**1**	1		0	**1**	1		0	**1**	1
1	0	0	1		0	**0**	0		1	**1**	0		1	**1**	0
1	0	1	1		0	**0**	1		1	**1**	0		1	**1**	1
1	1	0	1		1	**0**	0		1	**1**	1		1	**1**	0
1	1	1	1		1	**1**	1		1	**1**	1		1	**1**	1
	passo		1		1	2	1		1	2	1		1	2	1

(3) operações fora dos parênteses

a	b	c	a	$\vee$	$(b$	$\wedge$	$c)$	$=$	$(a$	$\vee$	$b)$	$\wedge$	$(a$	$\vee$	$c)$
0	0	0	0	**0**		0				0		**0**		0	
0	0	1	0	**0**		0				0		**0**		1	
0	1	0	0	**0**		0				1		**0**		0	
0	1	1	0	**1**		1				1		**1**		1	
1	0	0	1	**1**		0				1		**1**		1	
1	0	1	1	**1**		0				1		**1**		1	
1	1	0	1	**1**		0				1		**1**		1	
1	1	1	1	**1**		1				1		**1**		1	
	passo		1	3	1	2	1	4	1	2	1	3	1	2	1

(4) verificação de igualdade

a	b	c	a	$\vee$	$(b$	$\wedge$	$c)$	$=$	$(a$	$\vee$	$b)$	$\wedge$	$(a$	$\vee$	$c)$
0	0	0		0				**1**				0			
0	0	1		0				**1**				0			
0	1	0		0				**1**				0			
0	1	1		1				**1**				1			
1	0	0		1				**1**				1			
1	0	1		1				**1**				1			
1	1	0		1				**1**				1			
1	1	1		1				**1**				1			
	passo		1	3	1	2	1	4	1	2	1	3	1	2	1

Definição 3.18 (Operadores Adicionais) *Expressões podem ser combinadas pelo uso dos* operadores lógicos *listados nas Tabelas 3.4 e 3.5.*

Tabela 3.4: Operadores Lógicos sobre bits.

símbolo	operador	descrição
$\equiv$	"pode ser reescrito como"	(*meta símbolo*)
$=$	igualdade	
$\neg$	negação	
$\wedge$	conjunção	
$\vee$	disjunção	
$\oplus$	ou-exclusivo	$(a \oplus b \ \equiv \ \neg a \wedge b \ \vee \ a \wedge \neg b)$
$\Rightarrow$	implicação	$(a \Rightarrow b \ \equiv \ \neg a \vee b)$
$\Leftrightarrow$	equivalência	$(a \Leftrightarrow b \ \equiv \ a \Rightarrow b \ \wedge \ b \Rightarrow a)$
$\triangleleft \ \ \triangleright$	condicional	$(a \triangleleft c \triangleright b \ \equiv \ c \wedge a \ \vee \ \neg c \wedge b)$

Tabela 3.5: Definição dos operadores ou-exclusivo, implicação e equivalência.

ou-exclusivo

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

implicação

c	d	$c \Rightarrow d$
0	0	1
0	1	1
1	0	0
1	1	1

equivalência

e	f	$e \Leftrightarrow f$
0	0	1
0	1	0
1	0	0
1	1	1

Exemplo 3.4 O operador ou-exclusivo é usado numa série de aplicações interessantes. Em especial, ele é usado no circuito que efetua somas. Uma propriedade útil deste operador é

$$a \oplus \overline{b} = \overline{a \oplus b}$$

e a Tabela 3.6 a comprova. A operação ou-exclusivo produz 1 se os dois operandos são distintos. A negação do ou-exclusivo testa se os operandos são iguais, e esta é uma operação deveras popular.

Para verificar a propriedade, a coluna 2 do lado esquerdo deve ser comparada com a coluna 3 do lado direito. $\triangleleft$

Tabela 3.6: Tabela verdade para $a \oplus \overline{b} = \overline{a \oplus b}$.

a	b	a	$\oplus$	$\overline{b}$	$=$	$\neg$	$(a$	$\oplus$	$b)$
0	0	0	**1**	1	**1**	**1**	0	0	0
0	1	0	**0**	0	**1**	**0**	0	1	1
1	0	1	**0**	1	**1**	**0**	1	1	0
1	1	1	**1**	0	**1**	**1**	1	0	1
passo		1	2	1	4	3	1	2	1

Exemplo 3.5 Na expressão $a \Rightarrow b$, dizemos que a é o *antecedente* e b é o *consequente*, e ela é interpretada como "se o antecedente a é verdade, então o consequente b é verdade". A expressão $a \Rightarrow (b \Rightarrow c)$ é interpretada como "se a é verdade, então se b é verdade, então c é verdade". Essa afirmação equivale a "se ambos a e b são verdade, então c é verdade". A equivalência entre as duas

$$a \Rightarrow (b \Rightarrow c) \;\Leftrightarrow\; (a \wedge b) \Rightarrow c$$

é demonstrada na Tabela 3.7; deve-se comparar a coluna 3 nos dois lados da equivalência. ◁

Tabela 3.7: Tabela verdade para $[a \Rightarrow (b \Rightarrow c)] \;\Leftrightarrow\; (a \wedge b) \Rightarrow c$.

a	b	c	a	$\Rightarrow$	$(b$	$\Rightarrow$	$c)$	$\Leftrightarrow$	$(a$	$\wedge$	$b)$	$\Rightarrow$	c
0	0	0	0	**1**	0	1	0	**1**	0	0	0	**1**	0
0	0	1	0	**1**	0	1	1	**1**	0	0	0	**1**	1
0	1	0	0	**1**	1	0	0	**1**	0	0	1	**1**	0
0	1	1	0	**1**	1	1	1	**1**	0	0	1	**1**	1
1	0	0	1	**1**	0	1	0	**1**	1	0	0	**1**	0
1	0	1	1	**1**	0	1	1	**1**	1	0	0	**1**	1
1	1	0	1	**0**	1	0	0	**1**	1	1	1	**0**	0
1	1	1	1	**1**	1	1	1	**1**	1	1	1	**1**	1
	passo		1	3	1	2	1	4	1	2	1	3	1

Exemplo 3.6 Vejamos como comprovar a *Lei do Silogismo*

$$[(a \Rightarrow b) \wedge (b \Rightarrow c)] \;\Rightarrow\; (a \Rightarrow c) \tag{3.1}$$

que é empregada em cadeias de deduções: "se a faz com que b seja verdade, e b faz com que c seja verdade, então a faz com que c seja verdade". A Tabela 3.8 a comprova: a coluna 3 do lado esquerdo deve ser comparada à coluna 2 do lado direito.

A Lei do Silogismo nos permite concluir absurdos a partir de premissas corretas, em argumentos ditos *non sequitur*, porque a conclusão não segue das premissas. Por exemplo, *(i) se uma pessoa não usa máscara, então ela contrai COVID-19; e (ii) se uma pessoa tem COVID-19, então ela morre; (iii) logo, quem não usa máscara morre*. Este tipo de raciocínio é encontrado com dolorosa frequência em discussões nas tais "redes sociais". Os argumentos individuais são 'corretos' mas a conclusão não decorre do encadeamento dos antecedentes. ◁

Tabela 3.8: Tabela verdade para Lei do Silogismo.

a	b	c	$(a$	$\Rightarrow$	$b)$	$\wedge$	$(b$	$\Rightarrow$	$c)$	$\Rightarrow$	$(a$	$\Rightarrow$	$c)$
0	0	0	0	1	0	**1**	0	1	0	**1**	0	**1**	0
0	0	1	0	1	0	**1**	0	1	1	**1**	0	**1**	1
0	1	0	0	1	1	**0**	1	0	0	**1**	0	**1**	0
0	1	1	0	1	1	**1**	1	1	1	**1**	0	**1**	1
1	0	0	1	0	0	**0**	0	1	0	**1**	1	**0**	0
1	0	1	1	0	0	**0**	0	1	1	**1**	1	**1**	1
1	1	0	1	1	1	**0**	1	0	0	**1**	1	**0**	0
1	1	1	1	1	1	**1**	1	1	1	**1**	1	**1**	1
	passo		1	2	1	3	1	2	1	4	1	2	1

3.2.1 Simplificação de Expressões

Uma função com variáveis que pertencem a $\mathbb{B}$ é representada da forma usual, como na Equação 3.2. O valor de uma função f de n variáveis pode ser computado, e este valor pode ser atribuído a uma variável.

$$a, b, \ldots y, z \in \mathbb{B},\ f_n : \mathbb{B}^n \mapsto \mathbb{B} \ \bullet \ z = f_n(\overbrace{a, b, \ldots, y}^{n}) \tag{3.2}$$

O resultado da avaliação pode ser atribuído a uma variável que não aparece na expressão, para uso em outras funções ou expressões. A restrição de que a variável a que é atribuído o valor da função não pode estar no corpo da função é uma restrição importante e que somente será levantada no Capítulo 8. O método para a atribuição de valores às variáveis, para o cálculo do valor da função, é aquele usado para construir tabelas verdade.

Definição 3.19 (Literal) *Um* literal *numa expressão corresponde a uma variável ou o ao seu complemento.*

Definição 3.20 (Simplificação) *A expressão $\mathcal{E}'(a, \ldots, z)$ para $a, \ldots, z \in \mathbb{B}$ é uma* versão simplificada *da expressão $\mathcal{E}(a, \ldots, z)$ se, para todas as combinações possíveis dos valores de $a, \ldots, z$, a avaliação de $\mathcal{E}'$ produz os mesmos valores que a avaliação de $\mathcal{E}$, e $\mathcal{E}'$ tem um menor número de operadores e/ou de literais do que $\mathcal{E}$.*

O objetivo da simplificação de expressões é obter uma expressão mais simples, possivelmente com menos literais, e com menor número de operadores, mas que representa a mesma função dos argumentos que a expressão original. A implementação da função simplificada resulta num circuito mais simples, de menor custo, e possivelmente com maior velocidade de operação.

Exemplo 3.7 Simplifiquemos a expressão abaixo, empregando as propriedades dos operadores em $\mathbb{B}$. A cada passo da simplificação, a coluna da direita indica a propriedade que o justifica.

	$(a \vee b) \wedge \neg a$	distributiva
$\Leftrightarrow$	$(a \wedge \neg a) \vee (b \wedge \neg a)$	complemento $\wedge$
$\Leftrightarrow$	$0 \vee (b \wedge \neg a)$	identidade $\vee$
$\Leftrightarrow$	$b \wedge \neg a$	

A expressão original é expandida pela distributividade para que então a subexpressão $(a \wedge \neg a)$ possa ser eliminada. Isso é possível poque $(a \wedge \neg a) = 0$ e $(0 \vee x) = x$. A expressão original tem três literais e três operadores enquanto a simplificada tem dois literais e dois operadores. ◁

Espaço em branco proposital.

Exemplo 3.8 Simplifiquemos a expressão abaixo.

$$\begin{array}{lll} & a \vee (a \wedge b) & \text{identidade } \wedge \\ \Leftrightarrow & (a \wedge 1) \vee (a \wedge b) & \text{distributiva} \\ \Leftrightarrow & a \wedge (1 \vee b) & \text{elem. máximo } \vee \\ \Leftrightarrow & a \wedge 1 & \text{identidade } \wedge \\ \Leftrightarrow & a & \end{array}$$

A expressão original é expandida com a identidade do $\wedge$ ($a \wedge 1 = a$) para então aplicarmos a distributividade, de onde obtemos a subexpressão $(1 \vee b) = 1$. A subexpressão $(a \wedge 1)$ pode ser simplificada com outra aplicação da identidade do $\wedge$. A expressão original tem três literais e dois operandos; a simplificada é somente o literal a. ◁

Exemplo 3.9 Simplifiquemos a expressão abaixo, eliminando todos os parênteses desnecessários.

$$\begin{array}{lll} & (((\neg a) \wedge b) \vee (c \wedge (\neg d)) \vee (b \wedge (\neg d)) \vee (a \wedge (\neg b))) & \text{precedência } \neg \\ \Leftrightarrow & ((\neg a \wedge b) \vee (c \wedge \neg d) \vee (b \wedge \neg d) \vee (a \wedge \neg b)) & \text{precedência } \wedge \\ \Leftrightarrow & \neg a \wedge b \vee c \wedge \neg d \vee b \wedge \neg d \vee a \wedge \neg b & \end{array}$$

A primeira regra elimina os parênteses à volta das negações porque o operador $\neg$ é o que se liga mais fortemente às variáveis ou subexpressões. A segunda regra elimina os parênteses à volta das conjunções porque o operador $\wedge$ liga mais fortemente variáveis ou subexpressões do que o operador $\vee$. Em que pese a simplificação, a segunda linha parece ser mais gentil aos (meus) olhos do que a terceira. ◁

Exemplo 3.10 Simplifiquemos a expressão abaixo usando as propriedades das operações sobre os elementos de $\mathbb{B}$.

$$\begin{array}{lll} & \overline{a} \wedge b \vee c \wedge \overline{d} \vee b \wedge \overline{d} \vee a \wedge b & \text{precedência } \wedge \\ \Leftrightarrow & (\overline{a} \wedge b) \vee (c \wedge \overline{d}) \vee (b \wedge \overline{d}) \vee (a \wedge b) & \text{comutativa} \\ \Leftrightarrow & [(\overline{a} \wedge b) \vee (a \wedge b)] \vee (c \wedge \overline{d}) \vee (b \wedge \overline{d}) & \text{distributiva (b)} \\ \Leftrightarrow & [(\overline{a} \vee a) \wedge b] \vee (c \wedge \overline{d}) \vee (b \wedge \overline{d}) & \text{complemento } (a \vee \overline{a}) \\ \Leftrightarrow & [1 \wedge b] \vee (c \wedge \overline{d}) \vee (b \wedge \overline{d}) & \text{identidade } \wedge, \text{comutativa} \\ \Leftrightarrow & b \vee (b \wedge \overline{d}) \vee (c \wedge \overline{d}) & \text{identidade } \wedge \\ \Leftrightarrow & [(b \wedge 1) \vee (b \wedge \overline{d})] \vee (c \wedge \overline{d}) & \text{distributiva } (b) \\ \Leftrightarrow & [b \wedge (1 \vee \overline{d})] \vee (c \wedge \overline{d}) & \text{identidade } \vee \\ \Leftrightarrow & [b \wedge 1] \vee (c \wedge \overline{d}) & \text{identidade } \wedge \\ \Leftrightarrow & b \vee (c \wedge \overline{d}) & \end{array}$$

A primeira alteração agrupa as subexpressões, em função da precedência dos operadores, para simplificar as próximas manipulações. A aplicação da comutatividade aproxima as duas subexpressões em que aparecem os literais a e b. A distributividade separa b e faz $a \wedge \overline{a}$, o que permite eliminar os literais em a. A identidade $b = b \wedge 1$ e introduzida para permitir a eliminação do literal $\overline{d}$. A expressão da primeira linha tem dez operadores, enquanto a da última tem somente três. O número de literais cai de cinco para três. ◁

A uma primeira vista, a escolha de quais regras aplicar pode parecer feitiçaria, e em alguma medida o é. Há esperança e é possível acelerar o aprendizado. Tente repetir os exemplos, copiando somente a expressão por simplificar e a coluna das propriedades, e então efetue as transformações indicadas.

Exemplo 3.11 Simplifiquemos a expressão abaixo.

$$
\begin{array}{lll}
 & \neg a \wedge b \wedge \neg c \vee \neg a \wedge \neg b \wedge \neg c \vee a \wedge b \wedge \neg c \vee a \wedge \neg b \wedge \neg c & \text{precedência } \wedge \\
\Leftrightarrow & (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) & \text{comutativa } 2\times \\
\Leftrightarrow & (\neg a \wedge \neg c \wedge b) \vee (\neg a \wedge \neg c \wedge \neg b) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) & \text{associativa } 2\times \\
\Leftrightarrow & [(\neg a \wedge \neg c) \wedge b] \vee [(\neg a \wedge \neg c) \wedge \neg b] \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) & \text{distributiva } (\neg a \wedge \neg c) \\
\Leftrightarrow & [(\neg a \wedge \neg c) \wedge (b \vee \neg b)] \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) & \text{compl } \vee,\ \text{ident } \wedge \\
\Leftrightarrow & [\neg a \wedge \neg c] \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge \neg c) & \text{comut } 2\times, \text{assoc } 2\times \\
\Leftrightarrow & [\neg a \wedge \neg c] \vee [(a \wedge \neg c) \wedge b] \vee [(a \wedge \neg c) \wedge \neg b] & \text{distributiva } (a \wedge \neg c) \\
\Leftrightarrow & [\neg a \wedge \neg c] \vee [(a \wedge \neg c) \wedge (b \vee \neg b)] & \text{compl } \vee,\ \text{ident } \wedge \\
\Leftrightarrow & [\neg a \wedge \neg c] \vee [a \wedge \neg c] & \text{distributiva } \neg c \\
\Leftrightarrow & (\neg a \vee a) \wedge \neg c & \text{compl } \vee,\ \text{ident } \wedge \\
\Leftrightarrow & \neg c &
\end{array}
$$

A expressão contém ocorrências de a e $\neg a$, e de b e $\neg b$. A estratégia para simplificação é agrupar os literais com seus complementos e então eliminá-los. Veremos adiante o Teorema 3.21, que permite reduzir substancialmente o número de linhas desta simplificação. ◁

Exemplo 3.12 Simplifiquemos a expressão abaixo usando as propriedades das operações sobre os elementos de $\mathbb{B}$.

$$
\begin{array}{lll}
 & \neg(a \Rightarrow b) \vee \neg(\neg b \Rightarrow \neg a) & \text{definição } \Rightarrow \\
\Leftrightarrow & \neg(\neg a \vee b) \vee \neg(\neg(\neg b) \vee \neg a) & \text{involução} \\
\Leftrightarrow & \neg(\neg a \vee b) \vee \neg(b \vee \neg a) & \text{DeMorgan } 2\times \\
\Leftrightarrow & [\neg(\neg a) \wedge \neg b\,] \vee [\neg b \wedge \neg(\neg a)\,] & \text{involução } 2\times \\
\Leftrightarrow & a \wedge \neg b \vee \neg b \wedge a & \text{comutativa } \wedge \\
\Leftrightarrow & a \wedge \neg b \vee a \wedge \neg b & \text{idempotência } \vee \\
\Leftrightarrow & a \wedge \neg b &
\end{array}
$$

A expansão da definição de $\Rightarrow$ nos permite aplicar as propriedades do complemento e reduzir o número de literais e de operadores. ◁

Espaço em branco proposital.

Além da simplificação de expressões, podemos usar a manipulação algébrica para provar propriedades dos operadores sobre bits.

Exemplo 3.13 Vejamos como provar a propriedade chamada de *contra-positiva*

$$(a \Rightarrow b) \;\Leftrightarrow\; (\overline{b} \Rightarrow \overline{a})\,. \tag{3.3}$$

A prova é construída da mesma forma que numa simplificação e cada passo da cadeia de transformações é justificado por uma propriedade válida.

$$\begin{array}{rcll}
a \Rightarrow b & \Leftrightarrow & (\overline{a} \vee b) & \text{definição de } \Rightarrow \\
& \Leftrightarrow & \overline{\overline{\overline{a} \vee b}} & \text{involução} \\
& \Leftrightarrow & \overline{(\overline{\overline{a} \vee b})} & \text{DeMorgan} \\
& \Leftrightarrow & \overline{(\overline{\overline{a}} \wedge \overline{b})} & \text{involução} \\
& \Leftrightarrow & \overline{(a \wedge \overline{b})} & \text{DeMorgan} \\
& \Leftrightarrow & \overline{a} \vee \overline{\overline{b}} & \text{comutativa} \\
& \Leftrightarrow & \overline{\overline{b}} \vee \overline{a} & \text{definição de } \Rightarrow \\
& \Leftrightarrow & \overline{b} \Rightarrow \overline{a} &
\end{array}$$

A estratégia de prova é, a partir da *forma* do lado esquerdo (LE) da equivalência, tentar aplicar as propriedades que transformem a expressão do LE para que ela fique com a *forma* da expressão do lado direito (LD). Como o LD da equivalência contém os complementos dos literais do lado esquerdo, a aplicação da involução e DeMorgan nos ajudam a comprovar a equivalência. ◁

Exemplo 3.14 O Teorema de DeMorgan é excepcionalmente útil e foi definido para duas variáveis na Página 32. Vejamos como provar a sua validade para três variáveis.

$$\begin{array}{rcll}
\overline{a \wedge b \wedge c} & \Leftrightarrow & \overline{a \wedge k} & \text{substitui } (b \wedge c) \text{ por } k \\
& \Leftrightarrow & \overline{a} \vee \overline{k} & \text{DeMorgan} \\
& \Leftrightarrow & \overline{a} \vee \overline{(b \wedge c)} & \text{substitui } k \text{ por } (b \wedge c) \\
& \Leftrightarrow & \overline{a} \vee \overline{(b \wedge c)} & \text{DeMorgan em } \overline{(b \wedge c)} \\
& \Leftrightarrow & \overline{a} \vee (\overline{b} \vee \overline{c}) & \text{associatividade} \\
& \Leftrightarrow & \overline{a} \vee \overline{b} \vee \overline{c} &
\end{array}$$

A propriedade do fechamento nos garante que o resultado da conjunção de duas variáveis em $\mathbb{B}$ também pertence a $\mathbb{B}$. O primeiro passo da prova substitui a conjunção $b \wedge c$ pela variável $k \in \mathbb{B}$, para então aplicar DeMorgan nas duas variáveis. O terceiro passo desfaz a substituição e mais uma aplicação de DeMorgan nos leva ao resultado que desejávamos provar. A dualidade nos garante que

$$\overline{a \vee b \vee c} \Leftrightarrow \overline{a} \wedge \overline{b} \wedge \overline{c}$$

é também uma propriedade. Este teorema terá sua utilidade comprovada no Capítulo 5. ◁

3.3 Tipos e Especificações

A ideia de *tipos* não é novidade para aqueles que tiveram a sorte de encontrar uma competente professora – ou professor – de física no Ensino Médio. Estes os terão ensinado a *nunca* misturar voltagem com temperatura, nem comprimento com velocidade. A definição popular para tipos é "laranjas com laranjas, maçãs com maçãs".

A maioria das especificações e descrições de circuitos neste texto indicam o *tipo* dos operandos e operadores. Os tipos básicos são definidos na Tabela 3.9. O uso de tipos para os sinais é útil porque obriga o projetista a atentar para as interfaces entre os vários componentes de um circuito. Quando um sinal de tipo $\mathcal{T}_1$ é ligado a uma entrada com tipo $\mathcal{T}_2$, com $\mathcal{T}_1 \neq \mathcal{T}_2$, é muito provável que haja um erro de projeto ou de concepção. A linguagem de descrição de circuitos VHDL, introduzida no Capítulo 9, é dita *fortemente tipada* porque o compilador VHDL verifica os tipos de todos os sinais e indica como um erro a ligação entre sinais de tipos distintos e/ou incompatíveis. Em VHDL, dá-se o nome de *sinal* a uma variável lógica ou a uma tupla de variáveis lógicas. Usaremos este nome no que se segue.

Tabela 3.9: Tipos Básicos.

tipo	descrição
$\mathbb{B}$	Conjunto dos Bits
$\mathbb{N}$	Conjunto dos Números Naturais
$\mathbb{Z}$	Conjunto dos Números Inteiros
$\mathbb{R}$	Conjunto dos Números Reais
$\mathbb{B}^n$	Tupla, ou Vetor de Bits com largura n
$(\mathbb{B} \times \cdots \times \mathbb{B})$	Tupla de valores do tipo $\mathbb{B}$
$\Delta \mapsto \Theta$	Função com domínio em Δ e imagem em Θ

A Equação 3.4 contém a definição completa do operador condicional, que é a versão compacta e infixada do comando `if c then a else b` com operandos `a`, `b` e `c` de tipo $\mathbb{B}$.

$$\begin{array}{l} a, b, c : \mathbb{B} \\ \lhd \ \ \rhd : \mathbb{B} \times (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\ a \lhd c \rhd b \ \equiv \ (c \wedge a) \vee (\neg c \wedge b) \end{array} \tag{3.4}$$

Esta equação pode ser usada para *especificar* o comportamento de um circuito, ou para *documentar* uma implementação. Nos dois casos, o comportamento é definido com precisão e sem ambiguidade, e pode ser usado como um contrato entre quem especifica o circuito e quem o implementa, ou entre quem implementa e quem usa o circuito.

O tipo dos três operandos a, b, c é declarado, explicitando que estes são do tipo *bit*.

$$a, b, c : \mathbb{B}$$

O operador condicional $\lhd \ \rhd$ tem tipo

$$\lhd \ \ \rhd : \mathbb{B} \times (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B}$$

que é lido como "o condicional é uma *função* com dois argumentos, o primeiro do tipo $\mathbb{B}$, e o segundo é um *par de operandos* $(\mathbb{B} \times \mathbb{B})$, e produz um *resultado* do tipo $\mathbb{B}$". Finalmente, a expressão

$$a \lhd c \rhd b \ \equiv \ (c \wedge a) \vee (\neg c \wedge b)$$

define as relações válidas entre as variáveis declaradas anteriormente. Veremos adiante que o circuito que implementa o operador condicional é tão útil quanto o comando **`if then else`**.

3.4 Tuplas de Bits Que Representam Números

Uma tupla com k bits pode representar 2^k padrões distintos. Por exemplo, uma tupla com $k = 1$ representa o conjunto de padrões $\{\langle 0\rangle, \langle 1\rangle\}$; uma tupla com $k = 2$ representa o conjunto $\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$; para $k = 3$, o conjunto é

$$\{\langle 0,0,0\rangle, \langle 0,0,1\rangle, \langle 0,1,0\rangle, \langle 0,1,1\rangle, \langle 1,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,0\rangle, \langle 1,1,1\rangle\}\,.$$

As potências de 2 são números deveras populares em Computação. A n-ésima potência de 2 é definida por

$$2^n = \overbrace{2 \times 2 \times \cdots \times 2}^{\text{n vezes}}\,.$$

Por exemplo, $2^2 = 4 = 2 \times 2$, enquanto que $2^5 = 32 = 2 \times 2 \times 2 \times 2 \times 2$.

Tuplas de bits podem ser usadas para representar números na base 2. O número N é representado pelo vetor de bits $\langle n_{k-1}, n_{k-2}, \cdots, n_1, n_0\rangle$ e seu valor é definido, em notação posicional, pela soma de potências de 2 da Equação 3.5.

$$N = \sum_{i=0}^{k-1} n_i \times 2^i \tag{3.5}$$

Se o j-ésimo elemento da tupla é zero, a parcela correspondente da soma é zero. O número cinco é representado por $101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1$. O subscrito em '$101_2$' indica que o número está representado na base 2 e não na base 10.

A operação inversa da exponenciação é o logaritmo, e estamos interessados no logaritmo da base 2. Para um número N, o *logaritmo de N na base 2* é

$$2^{\log N} = N.$$

O logaritmo de um número indica a largura mínima da tupla de bits necessária para representar aquele número. Se $M = 2^m$, então uma tupla de m bits pode representar todos os números entre 0 e $2^m - 1$. Considere $M = 16$; temos que $\log 16 = 4$ porque $16 = 2 \times 2 \times 2 \times 2$, e na base 2, $16 = 1.0000_2$. É necessária uma tupla de quatro bits para representar todos os dezesseis padrões de bits entre $0000_2 = 0$ e $1111_2 = 15$. Para facilitar a leitura, o ponto agrupador de milhares é usado para agrupar quartetos de bits, e por isso escrevemos 1.0000_2 ao invés de 10.000_2.

No que se segue estão especificadas formalmente a função que converte uma tupla de bits para o número natural que aquela tupla representa, e o circuito que efetua a divisão inteira de dois números inteiros. A especificação não contém *nenhuma* indicação de como o circuito deva ser implementado, e esta "liberdade de implementação" é uma das liberdades fundamentais da, ou do, projetista.

Exemplo 3.15 A Equação 3.6 define a função que converte um vetor de bits para o número natural correspondente. A função num recebe como argumento um vetor V de n bits, de tipo $\mathbb{B}^n$, e retorna um inteiro N, que é o valor representado pelo vetor V. O i-ésimo elemento de V é denotado por v_i.

$$\begin{array}{l} V : \mathbb{B}^n \\ N : \mathbb{N} \\ num : \mathbb{B}^n \mapsto \mathbb{N} \\ num(V) = N \;\equiv\; N = \sum_{i=0}^{n-1} v_i \cdot 2^i \end{array} \tag{3.6}$$

O valor representado pelo vetor de bits é a soma dos produtos do valor do bit v_i pela i-ésima potência de dois, que é a formulação da representação posicional para a base-2. ◁

Exemplo 3.16 A Equação 3.7 especifica um circuito que efetua a divisão inteira de dois números representados em 8 bits. O circuito tem três entradas e três saídas: o sinal *inicia* carrega os valores do dividendo e do divisor nas entradas *ddndo* e *dvsor*, respectivamente, e dispara a computação do resultado; a saída *pronto* indica que as saídas *quoc* e *resto* contêm os valores do quociente e do resto da divisão inteira.

$$\begin{array}{l} inicia, pronto : \mathbb{B} \\ ddndo, dvsor : \mathbb{B}^8 \\ quoc, resto : \mathbb{B}^8 \\ \textit{div-8} : \mathbb{B} \times (\mathbb{B}^8 \times \mathbb{B}^8) \mapsto (\mathbb{B}^8 \times \mathbb{B}^8) \times \mathbb{B} \\ \textit{div-8}(inicia, ddndo, dvsor, quoc, resto, pronto) \;\equiv \\ \qquad inicia \Rightarrow [\, pronto \Rightarrow \\ \qquad\quad (num(ddndo) = (num(dvsor) \cdot num(quoc)) + num(resto))\,] \end{array} \tag{3.7}$$

Note que a especificação não determina a maneira de implementar o circuito mas somente define o resultado a ser produzido em função das entradas que lhe sejam apresentadas. Esta forma de especificar o comportamento desejado para um circuito, indicando-se apenas sua funcionalidade, é empregada na *modelagem funcional* de circuitos com VHDL. O material deste e dos próximos capítulos contém vários exemplos de especificações funcionais e de implementações que as satisfazem. ◁

Especificação com Pré-condições O uso de implicação lógica na Equação 3.7 merece uma explicação. Intuitivamente, a expressão $a \Rightarrow b$ significa que "*se* $a = 1$ *então* $b = 1$". A definição do operador $\Rightarrow$ é

$$a \Rightarrow b \equiv \neg a \vee b$$

e contra intuitivamente significa também que, "*se* $a = 0$ *então não se pode afirmar nada sobre o valor de* b". Esta interpretação é usada nas especificações para garantir que as condições necessárias para a correta operação do circuito são válidas: se as entradas estão dentro do esperado, então o circuito operará como o desejado.

Uma especificação é um contrato entre vendedor e comprador do circuito. Como parte do contrato, consta que se as entradas não são válidas, então nada se pode dizer sobre o comportamento do circuito. No caso do circuito divisor, se o circuito externo ao divisor não disponibilizar os valores de dividendo e divisor, e então ativar o sinal *inicia*, não se pode esperar uma resposta correta. Ademais, no contrato consta que, se as entradas estão corretas, então o resultado está disponível somente se o sinal *pronto* estiver ativo.

3.5 Tabela Verdade e Soma de Produtos

Expressões complexas podem ser construídas pela combinação de expressões simples, e o valor da expressão complexa pode ser derivado dos valores das expressões simples. Uma expressão pode ser avaliada através de uma tabela verdade, na qual são listadas todas as combinações dos valores das variáveis da expressão.

A tabela verdade de uma expressão com n variáveis contém 2^n linhas, uma linha para cada uma das combinações das variáveis. Considere a função

$$p = q \wedge (r \vee s)\,.$$

Sua tabela verdade (TV) tem 8 linhas para cobrir todas as 2^3 combinações para os valores de q, r e s, e é mostrada na Tabela 3.10. O *índice* das linhas da tabela, que normalmente é omitido, é o número representado pela atribuição de valores às variáveis em cada linha na tabela: $num(\langle q, r, s\rangle) =$ índice. Os operadores foram propositalmente omitidos e nos interessa mostrar somente o valor de p para todas as combinações das entradas.

Tabela 3.10: Tabela verdade da função $p = q \wedge (r \vee s)$**.**

índice	q	r	s	p
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Qualquer função em $\mathbb{B}^n$ pode ser representada por uma *soma de produtos*. A *soma* é uma disjunção de termos, e cada termo corresponde a uma linha da tabela verdade cujo valor seja 1. Cada *produto* é a conjunção de variáveis, ou de seus complementos – a própria variável se na linha da tabela seu valor é 1, ou seu complemento, se 0. A soma de produtos de p, segundo a Tabela 3.10 é

$$p = (q \wedge \overline{r} \wedge s) \;\vee\; (q \wedge r \wedge \overline{s}) \;\vee\; (q \wedge r \wedge s)\,,$$

porque os termos de p que são 1 correspondem às linhas 5 ($(q\wedge\overline{r}\wedge s) \leftrightarrow 101$), 6 ($(q\wedge r\wedge\overline{s}) \leftrightarrow 110$) e 7 ($(q \wedge r \wedge s) \leftrightarrow 111$).

Uma soma de produtos *completa* é aquela em que todas as variáveis da função, ou seus complementos, aparecem em todos os termos da soma. Uma soma de produtos completa é representada por uma *disjunção de conjunções*, com uma conjunção para cada linha da TV na qual a função seja 1. Cada conjunção é chamada de *mintermo* e cada mintermo deve conter todas as variáveis da função ou seus complementos.

Um mintermo é a conjunção de todas as variáveis da função, ou seus complementos, e o i-ésimo mintermo é representado por m_i. Uma função $\mathcal{F}$ pode ser representada pela soma dos mintermos para os quais $\mathcal{F} = 1$. A representação com mintermos de $p(q, r, s)$ é

$$p(q, r, s) = m_5 \vee m_6 \vee m_7\,.$$

Dependendo da função, pode ser mais interessante empregarmos uma representação alternativa, que é um *produto de somas*, ou uma *conjunção de disjunções*, porque esta representação pode ser menor do que a soma de produtos que lhe corresponde. A disjunção de *maxtermos* deve conter uma disjunção para cada linha da TV na qual a função é 0.

O *maxtermo* M_i é o complemento do mintermo m_i, no qual todas as variáveis são os respectivos complementos daquelas do mintermo. Por exemplo, ao mintermo $m_3 = \overline{a} \wedge b \wedge c$ corresponde o maxtermo $M_3 = a \vee \overline{b} \vee \overline{c}$. Assim como para os mintermos, cada maxtermo deve ser completo e conter todas as variáveis da função ou seus complementos. A representação $\overline{p(q,r,s)}$ com maxtermos de é a conjunção dos maxtermos que fazem $p = 0$

$$\overline{p(q,r,s)} = M_0 \wedge M_1 \wedge M_2 \wedge M_3 \wedge M_4 \,.$$

A Tabela 3.11 mostra todos os mintermos e maxtermos para uma função de três variáveis.

Tabela 3.11: Mintermos e maxtermos para uma função de três variáveis.

				mintermos		maxtermos	
índice	a	b	c	*produto*	*repr.*	*soma*	*repr.*
0	0	0	0	$\overline{a} \wedge \overline{b} \wedge \overline{c}$	m_0	$a \vee b \vee c$	M_0
1	0	0	1	$\overline{a} \wedge \overline{b} \wedge c$	m_1	$a \vee b \vee \overline{c}$	M_1
2	0	1	0	$\overline{a} \wedge b \wedge \overline{c}$	m_2	$a \vee \overline{b} \vee c$	M_2
3	0	1	1	$\overline{a} \wedge b \wedge c$	m_3	$a \vee \overline{b} \vee \overline{c}$	M_3
4	1	0	0	$a \wedge \overline{b} \wedge \overline{c}$	m_4	$\overline{a} \vee b \vee c$	M_4
5	1	0	1	$a \wedge \overline{b} \wedge c$	m_5	$\overline{a} \vee b \vee \overline{c}$	M_5
6	1	1	0	$a \wedge b \wedge \overline{c}$	m_6	$\overline{a} \vee \overline{b} \vee c$	M_6
7	1	1	1	$a \wedge b \wedge c$	m_7	$\overline{a} \vee \overline{b} \vee \overline{c}$	M_7

Exemplo 3.17 Vejamos como obter uma representação em maxtermos para a função

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

Como a expressão para $\overline{f(a,b,c)}$ não é completa, devemos iniciar pela obtenção da forma completa. O termo $(\overline{b} \vee c)$ deve ser completado pela inclusão da variável a, e o termo $(a \vee c)$ deve ser completado com a variável b:

$$\begin{array}{rcll} \overline{b} \vee c & \Leftrightarrow & (\overline{b} \vee c) \wedge (a \vee \overline{a}) & a \vee a = 1,\ 1 \wedge x = x \\ & \Leftrightarrow & (a \vee \overline{b} \vee c) \wedge (\overline{a} \vee \overline{b} \vee c) & \text{distributiva, comutativa} \end{array}$$

$$\begin{array}{rcll} a \vee c & \Leftrightarrow & (a \vee c) \wedge (b \vee \overline{b}) & a \vee a = 1,\ 1 \wedge x = x \\ & \Leftrightarrow & (a \vee b \vee c) \wedge (a \vee \overline{b} \vee c) & \text{distributiva, comutativa} \end{array}$$

Ao substituir aqueles termos, obtemos a representação completa para $\overline{f(a,b,c)}$,

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

Consultando a Tabela 3.11, obtemos a representação de $\overline{f(a,b,c)}$ com maxtermos

$$\begin{array}{rcl} \overline{f} & = & M_2 \wedge M_2 \wedge M_6 \wedge M_3 \wedge M_0 \wedge M_2 \\ & = & M_0 \wedge M_2 \wedge M_3 \wedge M_6 \,. \end{array}$$

A tabela verdade de $f(a,b,c)$ é mostrada na Tabela 3.12. Os maxtermos correspondem às linhas da TV nas quais $f(a,b,c)=0$. A TV está construída de forma a minimizar o trabalho de preenchimento; quando um termo é completamente determinado por uma variável ($0 \wedge x$ ou $1 \vee y$), as colunas das outras variáveis não estão preenchidas por serem redundantes. Somente uma coluna das conjunções ou disjunções com 2, 3 ou 4 variáveis é preenchida.

Tabela 3.12: Tabela verdade para $f(a,b,c)$.

a	b	c	a	$\vee$	$\overline{b}$	$\vee$	c	$\wedge$	$\overline{b}$	$\vee$	c	$\wedge$	a	$\vee$	$\overline{b}$	$\vee$	$\overline{c}$	$\wedge$	a	$\vee$	c	
0	0	0		**1**	1				1	**1**		**0**		**1**	1					**0**	0	M_0
0	0	1		**1**	1				1	**1**		**1**		**1**	1					**1**	1	m_1
0	1	0		**0**			0			**0**	0	**0**		**1**			1			**0**	0	M_2
0	1	1		**1**			1			**1**	1	**0**		**0**			0			**1**	1	M_3
1	0	0	1	**1**					1	**1**		**1**	1	**1**					1	**1**		m_4
1	0	1	1	**1**					1	**1**		**1**	1	**1**					1	**1**		m_5
1	1	0	1	**1**						**0**	0	**0**	1	**1**					1	**1**		M_6
1	1	1	1	**1**						**1**	1	**1**	1	**1**					1	**1**		m_7
	passo		1	3	2		1		2	3	1	4	1	3	2		2		1	2	1	

Consultando novamente a Tabela 3.11, obtemos $f(a,b,c)$ representada com mintermos

$$f = m_1 \vee m_4 \vee m_5 \vee m_7 \,.$$

A representação com mintermos é o complemento daquela com maxtermos porque a função com maxtermos representa os termos em que $f = 0$, enquanto a representação com mintermos representa os termos com $f = 1$.

Neste exemplo, as duas representações, com maxtermos e com mintermos, têm complexidade similar, cada uma com quatro termos de três variáveis. A verificação de qual forma simplificada é mais simples fica como um exercício. ◁

A soma (disjunção) dos mintermos é uma representação compacta para a tabela verdade de qualquer função $\mathcal{G}$. Empregamos três variáveis para simplificar a representação.

$$\mathcal{G}(a,b,c) = m_i \vee m_j \vee \ldots \vee m_k \,, \quad i,j,k \in [0,7] \,. \tag{3.8}$$

A representação em termos de produto (conjunção) de maxtermos é

$$\overline{\mathcal{G}(a,b,c)} = M_l \wedge M_m \wedge \ldots \wedge M_n \,, \quad l,n,m \in [0,7] \,. \tag{3.9}$$

Podemos também fazer uso de uma abreviação similar ao somatório e ao produtório para representar funções nas formas *soma de produtos* e o *produto de somas*. Empregando os símbolos $\bigvee$ e $\bigwedge$ para a disjunção e conjunção prefixadas, podemos representar a soma de produtos como

$$\mathcal{G}(a,b,c) = \bigvee(i,j,\ldots,k), \quad i,j,k \in [0,7] \,. \tag{3.10}$$

e o produto de somas como

$$\overline{\mathcal{G}(a,b,c)} = \bigwedge(l,m,\ldots,n), \quad l,m,n \in [0,7] \,. \tag{3.11}$$

Estas representações para funções em termos de soma de produtos completas e produto de somas completos são chamadas de *formas canônicas* das funções.

Exemplo 3.18 Considere a função h, representada por

$$h(a,b,c) = \bigvee(1,3,6,7) = m_1 \vee m_3 \vee m_6 \vee m_7 \,.$$

Seu complemento pode ser facilmente obtido com a Tabela 3.11

$$\overline{h(a,b,c)} = \bigwedge(0,2,4,5) = M_0 \wedge M_2 \wedge M_4 \wedge M_5 \,.$$

A Tabela 3.11 mostra que o maxtermo M_i é o complemento do mintermo m_i, e assim obtemos facilmente o produto de somas que representa $\overline{h(a,b,c)}$. ◁

Considere uma função com k variáveis e m mintermos. Para converter a representação de soma de produtos para produto de somas basta trocar os operadores $\bigvee$ para $\bigwedge$ e listar os $2^k - m$ maxtermos que não estão listados na soma de produtos. A conversão no sentido inverso é similar.

Exemplo 3.19 Para exemplificar a conversão, as funções, $s(a,b,c)$ e $v(a,b,c)$, são representadas nas suas formas canônicas.

$$\begin{aligned} s(a,b,c) &= \bigvee(1,2,4,7) = m_1 \vee m_2 \vee m_4 \vee m_7 \\ \overline{s(a,b,c)} &= \bigwedge(0,3,5,6) = M_0 \wedge M_3 \wedge M_5 \wedge M_6 \,, \end{aligned}$$

$$\begin{aligned} v(a,b,c) &= \bigvee(3,7) = m_3 \vee m_7 \\ \overline{v(a,b,c)} &= \bigwedge(0,1,2,4,5,6) = M_0 \wedge M_1 \wedge M_2 \wedge M_4 \wedge M_5 \wedge M_6 \,. \end{aligned}$$

Esta conversão é um tanto mais simples do que sucessivas aplicações de DeMorgan. A representação em mintermos de $v(a,b,c)$ é algo mais econômica do que com maxtermos. ◁

Vejamos três exemplos de funções de tipo $\mathbb{B}^3 \mapsto \mathbb{B}$, definidas na Tabela 3.13.

Exemplo 3.20 A função $t(a,b,c)$ indica se a tripla de bits representa um número que é múltiplo de três. A função pode ser representada na forma de soma de produtos, ou de forma mais concisa como uma conjunção de mintermos:

$$t = (\overline{a} \wedge b \wedge c) \;\vee\; (a \wedge b \wedge \overline{c}) = m_3 \vee m_6 \,.$$

Da tabela verdade vemos que só duas linhas fazem $t = 1$, as linhas de índice 3 e 6, que são aquelas das conjunções $(\overline{a} \wedge b \wedge c)$ e $(a \wedge b \wedge \overline{c})$. Isso porque a função $t(a,b,c)$ é 1 somente quando os argumentos são tais que fazem aquelas conjunções iguais a 1, a saber $\langle$0,1,1$\rangle$ e $\langle$1,1,0$\rangle$. ◁

Espaço em branco proposital.

Tabela 3.13: Tabelas verdade para múltiplos de três, paridade par e ímpar.

índice	a	b	c	t
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

j	k	l	i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

r	s	t	p
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Exemplo 3.21 O segundo exemplo de função $\mathbb{B}^3 \mapsto \mathbb{B}$ é a função que indica se a contagem de bits em 1 na tripla é ímpar. Esta função é tão útil que recebe nome próprio: *paridade ímpar*. A função $i(j,k,l)$ é 1 quando a quantidade de 1s na tripla de argumentos é ímpar. $i(j,k,l)$ é representada em forma de soma de produtos por

$$i = (\overline{j} \wedge \overline{k} \wedge l) \vee (\overline{j} \wedge k \wedge \overline{l}) \vee (j \wedge \overline{k} \wedge \overline{l}) \vee (j \wedge k \wedge l) = m_1 \vee m_2 \vee m_4 \vee m_7$$

Na Tabela 3.13 a função $i()$ está dividida entre as metades em que $j = 0$ e $j = 1$. Na metade de cima, temos que $i = k \oplus l$ e na metade de baixo temos que $i = \overline{k \oplus l}$. Podemos usar esta informação para simplificar $i(j,k,l)$:

$$\begin{array}{rcll} i & \Leftrightarrow & \overline{j} \wedge (k \oplus l) \vee j \wedge \overline{(k \oplus l)} & \text{substitui } (k \oplus l) \text{ por } y \\ & \Leftrightarrow & \overline{j} \wedge y \vee j \wedge \overline{y} & \text{definição de } \oplus \\ & \Leftrightarrow & j \oplus y & \text{substitui } y \text{ por } (k \oplus l) \\ & \Leftrightarrow & j \oplus (k \oplus l) & \text{Exercício 3.7} \\ & \Leftrightarrow & j \oplus k \oplus l & \end{array}$$

A paridade *ímpar* de n bits é seu ou-exclusivo. ◁

Exemplo 3.22 O terceiro exemplo de função $\mathbb{B}^3 \mapsto \mathbb{B}$ é a função que indica se a contagem de bits em 1 na tripla é par. Esta função é chamada de *paridade par*. A função $p(r,s,t)$ é 1 quando a quantidade de 1s na tripla de argumentos é par. $p(r,s,t)$ é representada em forma de soma de produtos por

$$p = (\overline{r} \wedge \overline{s} \wedge \overline{t}) \vee (\overline{r} \wedge s \wedge t) \vee (r \wedge \overline{s} \wedge t) \vee (r \wedge s \wedge \overline{t}) = r \oplus s \oplus \overline{t}$$

A forma com $\oplus$ é obtida se usarmos o raciocínio do Exemplo 3.21 mais o Exemplo 3.4. ◁

3.6 Mapas de Karnaugh

Um teorema útil no tratamento de expressões com operadores e elementos de $\mathbb{B}$ é o Teorema 3.21, conhecido como o *Teorema da Simplificação*. Este teorema permite a eliminação de variáveis em expressões. As equivalências (a) permitem a construção de Mapas de Karnaugh, que são apresentados no que se segue. O Teorema da Simplificação para duas variáveis pode ser provado, facilmente, com três tabelas verdade e Dualidade. O Teorema 3.21 (a) também é chamado de *Teorema da Absorção*.

Definição 3.21 (Teorema da Simplificação)

$$\begin{array}{cc} (x \wedge y) \vee (x \wedge \neg y) = x & (x \vee y) \wedge (x \vee \neg y) = x \quad (a) \\ x \vee (x \wedge y) = x & x \wedge (x \vee y) = x \quad (b) \\ (x \vee \neg y) \wedge y = x \wedge y & (x \wedge \neg y) \vee y = x \vee y \quad (c) \end{array}$$

Exemplo 3.23 Simplifiquemos a expressão abaixo usando as propriedades dos operadores sobre $\mathbb{B}$, mais o Teorema da Simplificação.

$$\begin{array}{lll} & [w \wedge (x \vee y)] \vee (\neg w \wedge \neg x) \vee y & \text{distributiva} \\ \Leftrightarrow & (w \wedge x) \vee (w \wedge y) \vee (\neg w \wedge \neg x) \vee y & \text{comutativa} \\ \Leftrightarrow & (w \wedge x) \vee (\neg w \wedge \neg x) \vee [(w \wedge y) \vee y] & \text{simplificação (b)} \\ \Leftrightarrow & [(w \wedge x) \vee (\neg w \wedge \neg x)] \vee y & \text{definição } \oplus \\ \Leftrightarrow & (w \oplus \neg x) \vee y & \end{array}$$

A primeira regra expande a expressão original para permitir a simplificação. A segunda aproxima os termos em w e x, e a terceira elimina a subexpressão $(w \wedge y)$. A quarta regra reescreve os dois termos com a definição do ou-exclusivo. ◁

O Teorema da Simplificação é empregado implicitamente na construção de *Mapas de Karnaugh* para permitir a simplificação de expressões pelo agrupamento de células. A Figura 3.2 mostra o mapa de Karnaugh para a expressão $p = q \wedge (r \vee s)$. Cada célula no mapa é identificada por um número no canto superior direito, que é justamente o índice na tabela verdade correspondente à célula. Os pares de números acima do mapa indicam os valores que correspondem às células abaixo do par, para as variáveis r e s. As colunas em que $r = 1$ são aquelas 'cobertas' pela linha horizontal, acima do mapa, marcada com r, e as colunas com $s = 1$ são 'cobertas' pela linha marcada com s. A linha vertical marcada com q indica a linha com $q = 1$.

índice	p	q	r	s
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	1	0	1
6	1	1	1	0
7	1	1	1	1

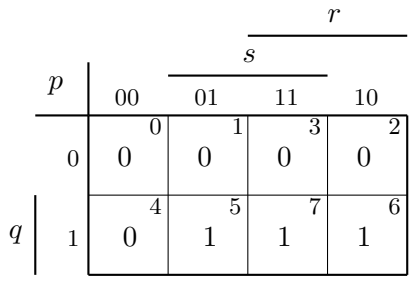

Figura 3.2: Mapa de Karnaugh para $p = q \wedge (r \vee s)$.

A i-ésima célula do mapa é preenchida com o valor para p da linha da tabela verdade de índice i. Assim, a célula de número 0 corresponde à primeira linha da tabela, com $\langle q, r, s \rangle = 000_2$, enquanto que a célula de número 5 corresponde à linha com $\langle q, r, s \rangle = 101_2$.

A disposição das células é tal que a representação em binário dos índices de duas células vizinhas difere exatamente em uma posição. Considere duas células vizinhas i e j. O mapa é organizado para que os índices destas células sejam tais que somente um dos bits na representação binária dos dois índices difere, o que nos permite usar a primeira versão do Teorema da Simplificação, e eliminar a variável que muda. Isso permite agregar e simplificar os mintermos m_i e m_j na soma de produtos.

O mapa é desenhado como um retângulo, mas seu formato é o de um toroide[2] porque as células dos lados menores são adjacentes – as células 0 e 2 são vizinhas – assim como as células dos lados maiores, embora isso seja mais fácil de perceber num mapa para quatro variáveis.

São possíveis duas simplificações no mapa de $p = q \wedge (r \vee s)$, e estes são indicados pelas elipses na Figura 3.3. A elipse da esquerda representa o agrupamento $q \wedge s$ e a elipse da direita representa o agrupamento $q \wedge r$.

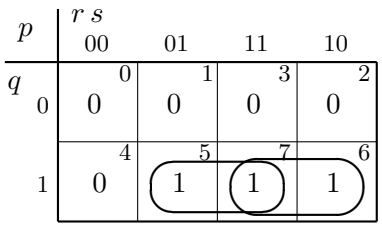

Figura 3.3: Mapa de Karnaugh com os agrupamentos de $p = q \wedge (r \vee s)$.

A elipse da esquerda elimina r porque as células 5 e 7 representam os mintermos m_5 (índice $\langle$q,r,s$\rangle = 101$) com $r = 0$, e m_7 (índice $\langle$q,r,s$\rangle = 111$) com $r = 1$. A elipse da direita elimina s porque as células 6 e 7 representam os mintermos m_6 (índice $\langle$q,r,s$\rangle = 110$) com $s = 0$ e m_7 com $s = 1$.

A expressão, na forma de *soma de produtos*, resultante da simplificação é

$$p' = (q \wedge s) \vee (q \wedge r)\,,$$

que é uma expressão 'maior' que aquela de p. A expressão original pode ser obtida de p' pela aplicação de distributividade.

Exemplo 3.24 Vejamos como preencher o Mapa de Karnaugh para $g(a, b, c) = \bigvee(0, 1, 2, 3, 5, 7)$, e então deduzir uma simplificação para g.

O preenchimento é direto: as células com os números dos mintermos são preenchidas com 1, e as demais com 0.

O mapa na Figura 3.4 mostra os dois agrupamentos que resultam na máxima simplificação. O agrupamento da linha de cima simplifica as variáveis b e c porque as células adjacentes de ambas trocam de valor na linha. O termo simplificado corresponde a $\overline{a}$.

O agrupamento no centro do mapa corresponde a c porque há troca de valor tanto nas células em que $a = 1$, quanto nas em que $b = 1$. A função simplificada é portanto $g' = \overline{a} \vee c$.

Outra possibilidade seria aproveitar que o agrupamento do centro cobre os mintermos m_1, m_3, m_5 e m_7, e agrupar os mintermos m_0 e m_2, que são vizinhos "por fora" do mapa. Neste caso, a função (menos) simplificada é $g'' = (\overline{a} \wedge \overline{c}) \vee c$. ◁

[2] Mesmo formato que um pneu de automóvel.

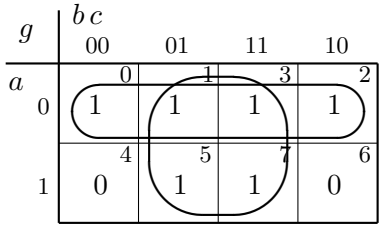

Figura 3.4: Mapa de Karnaugh para $g(a, b, c) = \bigvee(0, 1, 2, 3, 5, 7)$**.**

Exemplo 3.25 A Figura 3.5 mostra a tabela verdade e o Mapa de Karnaugh para uma função de quatro variáveis, $u(x, y, z, w) = \bigvee(1, 4, 6, 9, 12, 14)$. Neste caso, os dois agrupamentos possíveis são nas bordas do mapa. As células 1 e 9 são adjacentes porque x é a única variável que muda entre as duas células, e este agrupamento corresponde ao termo $\overline{y} \wedge \overline{z} \wedge w$. As células 4 e 6 são adjacentes porque z muda, assim como as células 12 e 14. Este agrupamento resulta no termo $y \wedge \overline{w}$. A função simplificada é portanto $u(x, y, z, w) = \overline{y} \wedge \overline{z} \wedge w \vee y \wedge \overline{w}$. ◁

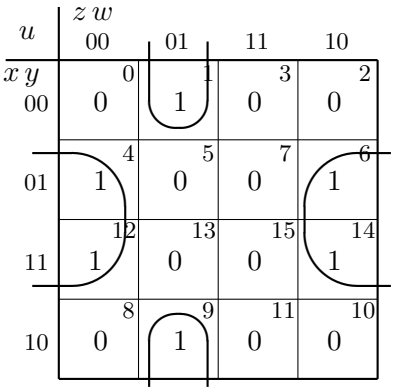

Figura 3.5: Mapa de Karnaugh para $u(x, y, z, w) = \bigvee(1, 4, 6, 9, 12, 14)$**.**

Espaço em branco proposital.

Exemplo 3.26 A Figura 3.6 mostra a tabela verdade e o Mapa de Karnaugh para uma função de quatro variáveis. Para esta função em particular, não há como agrupar células para obter alguma simplificação de t. As células preenchidas com 1 formam diagonais no mapa, e isso indica que a função é o ou-exclusivo das entradas – a tabela verdade mostra que a função t é 1 somente nas linhas com um número ímpar de 1's nos valores de $\langle x, y, z, w\rangle$, e portanto $t = x \oplus y \oplus z \oplus w$. ◁

índice	t	x	y	z	w
0	0	0	0	0	0
1	1	0	0	0	1
2	1	0	0	1	0
3	0	0	0	1	1
4	1	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	1	0	1	1	1
8	1	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	1	1	0	1	1
12	0	1	1	0	0
13	1	1	1	0	1
14	1	1	1	1	0
15	0	1	1	1	1

t	$z\,w$ 00	01	11	10
$x\,y$ 00	0	1	3	2
	0	1	0	1
01	4	5	7	6
	1	0	1	0
11	12	13	15	14
	0	1	0	1
10	8	9	11	10
	1	0	1	0

Figura 3.6: Tabela verdade e mapa de Karnaugh para $t(x, y, z, w)$.

O mapa de quatro variáveis das Figuras 3.5 e 3.6 é desenhado como um quadrado apenas para facilitar o desenho em duas dimensões. Uma representação mais precisa é aquela de um toroide – as células da primeira linha da figura são adjacentes às da última linha (0 e 8, 1 e 9, 3 e 11, e 2 e 10), e as células da esquerda são adjacentes àquelas da direita (0 e 2, 4 e 6, 12 e 14, e 8 e 10). Estas células são adjacentes porque há somente a troca de x para $\overline{x}$ nas linhas do topo e base, e de z para $\overline{z}$ nas linhas da esquerda e da direita, e portanto podem ocorrer simplificações que envolvem células em "margens opostas" do mapa.

Exemplo 3.27 A Figura 3.7 mostra a tabela verdade e a função simplificada para uma função que identifica números primos codificados em 4 bits – a função $p(a, b, c, d)$ é 1 sempre que o número representado pelos argumentos for primo.

A figura também mostra a soma dos seis produtos e a expressão simplificada através de cinco aplicações do teorema da simplificação (a). Para tal, basta olhar na tabela, nas linhas em que a função é 1, e verificar em quais pares de linhas ocorre uma variável e seu complemento, desde que os outros literais sejam idênticos. O termo marcado com uma estrela, é redundante e pode ser eliminado. ◁

Os Mapas de Karnaugh são uma ferramenta interessante e tiveram seu período de glória nas décadas de 1960 e 1970, quando o custo de uma porta lógica individual tendia a ser uma parcela não trivial do custo do projeto inteiro. Atualmente estes mapas têm aplicação reduzida porque o ganho pela eliminação de algumas portas lógicas é irrelevante frente ao custo da hora de trabalho da projetista. Em circuitos com milhares, ou milhões, de portas, o custo da eliminação de uma porta raramente compensa o tempo e o esforço. Além disso, a

índice	a	b	c	d	p
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

$$
\begin{array}{rcll}
p & = & (\overline{a} \wedge \overline{b} \wedge c \wedge \overline{d}) \vee & m_2 \\
 & & (\overline{a} \wedge \overline{b} \wedge c \wedge d) \vee & m_3 \\
 & & (\overline{a} \wedge b \wedge \overline{c} \wedge d) \vee & m_5 \\
 & & (\overline{a} \wedge b \wedge c \wedge d) \vee & m_7 \\
 & & (a \wedge \overline{b} \wedge c \wedge d) \vee & m_{11} \\
 & & (a \wedge b \wedge \overline{c} \wedge d) & m_{13} \\
 & & \text{elimina } d \text{ de } m_2 \text{ e } m_3 & \\
 & & \text{elimina } b \text{ de } m_3 \text{ e } m_7 & \\
 & \Leftrightarrow & \text{elimina } c \text{ de } m_5 \text{ e } m_7 * & \\
 & & \text{elimina } a \text{ de } m_5 \text{ e } m_{13} & \\
 & & \text{elimina } a \text{ de } m_3 \text{ e } m_{11} & \\
 & & (\overline{a} \wedge \overline{b} \wedge c) \vee & \\
 & & (\overline{a} \wedge c \wedge d) \vee & \\
 & & (\overline{a} \wedge b \wedge d) \vee * & \\
 & & (\overline{b} \wedge c \wedge d) \vee & \\
 & & (b \wedge \overline{c} \wedge d) &
\end{array}
$$

Figura 3.7: Tabela verdade para a função primos em 4 bits.

maioria das funções emprega mais do que quatro variáveis, o que torna o desenho dos mapas extremamente monótono e propício a erros. Os "compiladores de *hardware*" disponíveis têm capacidade para efetuar as otimizações necessárias.

Exercícios

Ex. 3.1 Prove que o operador $\bigwedge$ de três entradas $\bigwedge(a, b, c)$ é equivalente a duas aplicações do operador $\wedge$ de duas entradas: $\bigwedge(a, b, c) \equiv (a \wedge b) \wedge c$.

Ex. 3.2 Estenda o resultado do Ex. 3.1 para qualquer número de entradas.

Ex. 3.3 Repita o Ex. 3.1 para os operadores $\bigvee$ and $\vee$.

Ex. 3.4 Estenda o resultado do Ex. 3.3 para qualquer número de entradas.

Ex. 3.5 Com base nos resultados dos Ex. 3.2 e 3.4, defina os operadores $\bigwedge$ e $\bigvee$ de uma maneira análoga à definição de somatório para inteiros.

Ex. 3.6 Com base no Ex. 3.5, mostre que $\bigwedge$ equivale ao quantificador universal $\forall$, e que $\bigvee$ equivale ao quantificador existencial $\exists$.

Ex. 3.7 Prove que o operador $\oplus$ é associativo. Prove que t, na Figura 3.6, é $t = \bigoplus(x, y, z, w)$. *Pista*: inicie provando que $\bigoplus(x, y, z)$ é associativo.

Ex. 3.8 Prove que o Teorema de DeMorgan é válido para qualquer número de variáveis.

Use as expressões abaixo para resolver os Exercícios 3.9 a 3.12.

$$a, b, c, d, e, p, q, r, s, t, u : \mathbb{B}$$

$$\begin{array}{ll} b \vee (c \wedge (d \vee e)) & (i) \\ (q \wedge (\neg t \vee u)) \vee ((q \vee s) \wedge \neg(t \vee \neg s)) & (ii) \\ a \vee \neg b \vee (\neg c \wedge d \wedge (\neg b \vee \neg a)) & (iii) \\ (\neg p \wedge \neg q \wedge r) \vee \neg(s \wedge (p \vee r \vee \neg q)) & (iv) \end{array}$$

Ex. 3.9 Preencha as tabelas verdade das expressões (i) a (iv). Enumere os mintermos com as variáveis na ordem alfabética.

Ex. 3.10 Use as propriedades dos bits para simplificar algebricamente as expressões (i) a (iv).

Ex. 3.11 Simplifique as expressões (i) a (iv) com mapas de Karnaugh.

Ex. 3.12 Escreva as duas formas canônicas para as expressões (i) a (iv).

Ex. 3.13 Com base na especificação do divisor (Eq. 3.7), especifique um circuito que efetua a multiplicação inteira.

Ex. 3.14 Defina a inversa da função num (Eq. 3.6). A função num^{-1} tem como argumento um número natural e produz uma tupla de bits que representa o argumento na base dois.

Índice Remissivo