Capítulo 4

Circuitos Combinacionais

Palavra puxa palavra, uma ideia traz outra, e assim se faz um livro, um governo, ou uma revolução; alguns dizem mesmo que assim é que a natureza compôs as suas espécies.
Machado de Assis, Primas de Sapucaia!

Um *circuito combinacional* produz saídas que dependem exclusivamente dos valores nas suas entradas, e para um mesmo conjunto de entradas o circuito produz sempre o mesmo conjunto de saídas. Circuitos combinacionais não têm *estado* nem *memória*.

Este capítulo[1] define os *circuitos combinacionais* que as empregam[2]. Com essas definições postas, são investigados os circuitos que implementam algumas funções lógicas que são comuns no projeto de sistemas digitais. A Seção 4.2 descreve os circuitos que implementam os operadores lógicos básicos, chamados de *portas lógicas*. A implementação destes circuitos é discutida no Capítulo 5. A Seção 4.3 especifica o comportamento de multiplexadores, demultiplexadores e decodificadores.

Os nomes dos circuitos que implementam os operadores lógicos estão propositalmente em Inglês para evitar confusão com as palavras '*e*' e '*ou*' do Português. Assim, a porta lógica com a função de conjunção, ou *e-lógico*, é chamada de porta *and*. A porta com a função de disjunção, ou *ou-lógico*, é chamada de porta *or*.

Espaço em branco proposital.

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

[2] A definição de *circuito combinacional* apresentada na Seção 4.1 toma emprestado muito dos conteúdos da disciplina 6.004 – *Computation Structures* do MIT, versão de 2013.

4.1 O Que É Um Circuito Combinacional?

Com a abstração para os sinais elétricos definida em termos de *bits*, ou *sinais digitais* que podem assumir um dentre os valores discretos em $\mathbb{B}$, é possível definir o que se entende por *dispositivo combinacional* e por *circuito combinacional*.

Definição 4.1 (Dispositivo Combinacional) *Um* Dispositivo Combinacional *é um dispositivo com as seguintes propriedades:*

1. *uma ou mais* entradas *digitais;*
2. *uma ou mais* saídas *digitais;*
3. *uma* especificação funcional *que determina o valor lógico de cada saída para cada uma das combinações das entradas; e*
4. *uma* especificação temporal *que determina, pelo menos, um limite superior para o tempo de propagação dos sinais que atravessam o dispositivo.*

A especificação funcional do dispositivo pode ser um conjunto de somas de produtos que relacionam todas as combinações de valores nas entradas com valores bem definidos nas saídas, ou ainda uma tabela verdade para cada saída que relaciona entradas com os valores de saída. Uma *especificação funcional* é um contrato entre o projetista de um dispositivo e o usuário deste dispositivo, e garante que *entradas válidas implicam em saídas válidas*.

O comportamento temporal dos dispositivos pode ser mensurado nos terminais do dispositivo, ou computado através de simulações, e o que nos interessa agora é o *tempo de propagação* dos sinais através do dispositivo, ou o *atraso* (*delay*) na propagação dos sinais introduzido pelo dispositivo. O tempo de propagação é o tempo necessário para que o circuito produza uma saída válida estável, a partir do instante em que suas entradas estabilizaram. Voltaremos a este assunto na Seção 5.3.

Definição 4.2 (Circuito Combinacional) *Um* circuito combinacional *pode ser construído pela interligação de dispositivos combinacionais que atendem às seguintes restrições:*

5. *cada componente é um dispositivo combinacional;*
6. *cada uma das entradas está ligada a exatamente uma saída ou às constantes 0 ou 1;*
7. *o circuito não contém circularidade nas ligações* ($\lambda \rightsquigarrow \sigma \rightsquigarrow \pi \not\rightsquigarrow \lambda$).

A restrição 6 proíbe a ligação de duas saídas, o que causa curto-circuitos, e também proíbe entradas desligadas (circuitos abertos). A restrição 7 impede laços que liguem as saídas do componente Λ à entradas do componente Π, que direta ou indiretamente formam um laço que alimenta as entradas de Λ.

Considere o circuito combinacional mostrado na Figura 4.1, que consiste de um circuito com três entradas (p, q, r) e dois dispositivos combinacionais, cada um com sua especificação funcional e temporal. Para nos certificarmos de que o circuito na figura é combinacional basta observar que o sinal interno y torna-se válido e bem definido 2ns depois que as entradas p e q estabilizam. A saída s torna-se válida e bem definida $2 + 1$ns depois que as entradas p, q e r estabilizam. O sinal y está ligado a exatamente uma única saída, e não há circularidade porque não existe ligação entre as saídas y e s com as entradas p, q ou r.

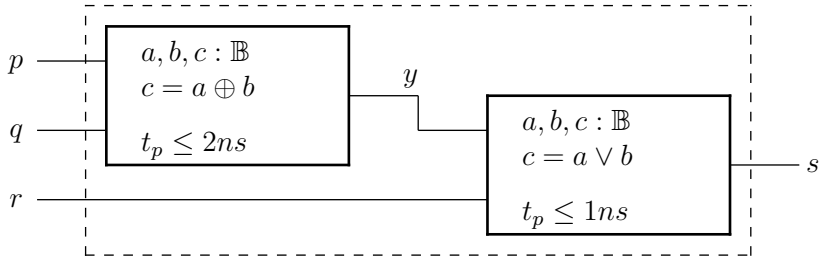


Figura 4.1: Circuito combinacional.

O valor da saída s, seja ele 0 ou 1, pode ser facilmente deduzido pela composição das funções dos dois componentes. Este circuito tem um tempo de propagação de, no máximo, 3ns, e sua saída é uma função dos sinais de entrada, sendo portanto um circuito combinacional. Esta forma de verificação pode ser aplicada a circuitos com um número arbitrário de componentes.

Como é computado o tempo de propagação através de um circuito combinacional? Para que se possa usar um determinado dispositivo com a confiança de que seu tempo de propagação é um limite superior para o intervalo entre o instante em que as entradas ficam estáveis e o instante no qual a saída fica estável, devemos computar o *tempo de propagação de pior caso*. Para cada caminho entre as entradas e a saída computa-se o atraso cumulativo naquele caminho, que é a soma dos atrasos impostos pelos componentes no caminho. O tempo de propagação do dispositivo é o mais longo dentre os tempos de propagação de todos os caminhos. A Seção 5.3 explora este tema em mais detalhe.

Exemplo 4.1 Ignorando sua função, qual é o tempo de propagação do circuito na Figura 4.2?

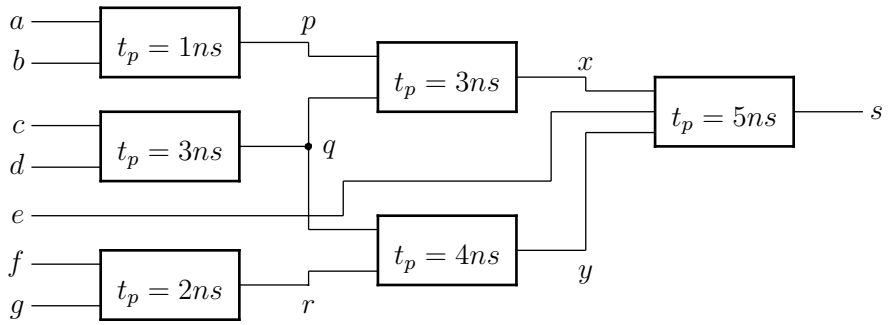


Figura 4.2: Circuito combinacional com atrasos.

1. O caminho mais curto é o da entrada e para a saída, que é de 5ns;
2. o caminho $\{a, b\} \rightsquigarrow p \rightsquigarrow x \rightsquigarrow s$ tem custo de $1 + 3 + 5 = 9$ns;
3. o caminho $\{c, d\} \rightsquigarrow q \rightsquigarrow x \rightsquigarrow s$ tem custo de $3 + 3 + 5 = 11$ns;
4. o caminho $\{c, d\} \rightsquigarrow q \rightsquigarrow y \rightsquigarrow s$ tem custo de $3 + 4 + 5 = 12$ns; e
5. o caminho $\{f, g\} \rightsquigarrow r \rightsquigarrow y \rightsquigarrow s$ tem custo de $2 + 4 + 5 = 11$ns.

O tempo de propagação do circuito é o pior caso, que é 12ns. ◁

4.2 Portas Lógicas

A abstração de sinais como bits, definida na Seção 3.1, é um modelo abstrato para o comportamento dos circuitos eletrônicos que são usados para implementar sistemas digitais. Se o que ocorre durante as transições entre os dois níveis lógicos pode ser ignorado, então o modelo abstrato é uma descrição perfeitamente razoável para o comportamento da implementação das portas lógicas na tecnologia CMOS, que são o objeto do Capítulo 5.

Os operadores conjunção ($\wedge$), disjunção ($\vee$) e complemento ($\neg$), podem ser representados pelos símbolos gráficos das portas lógicas *and*, *or* e *not*, respectivamente. Os símbolos destes circuitos são mostrados na Figura 4.3, e suas funções são definidas na Equação 4.1. As portas *and* e *or* são definidas para duas entradas; estas definições podem ser entendidas para qualquer número de entradas, empregando-se a Definição 3.16.

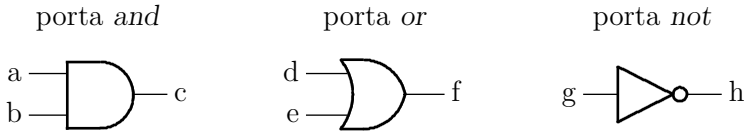


Figura 4.3: Símbolos das portas lógicas *and*, *or* e *not*.

$$
\begin{array}{l}
a, b, c, d, e, f, g, h : \mathbb{B} \\
\\
\mathit{and\text{-}2} : (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\mathit{and\text{-}2}(a, b, c) \;\equiv\; c = a \wedge b \\
\\
\mathit{or\text{-}2} : (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\mathit{or\text{-}2}(d, e, f) \;\equiv\; f = d \vee e \\
\\
\mathit{not} : \mathbb{B} \mapsto \mathbb{B} \\
\mathit{not}(g, h) \;\equiv\; h = \neg g
\end{array}
\tag{4.1}
$$

O comportamento das portas lógicas também pode ser descrito por *tabelas verdade*, e a Tabela 4.1 define o comportamento dos operadores $\wedge$, $\vee$ e $\neg$, e também das portas *and*, *or* e *not*. Note que as Tabelas 3.1 e 4.1 são idênticas.

Tabela 4.1: Tabelas Verdade das portas *and*, *or* e *not*.

and

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

or

d	e	f
0	0	0
0	1	1
1	0	1
1	1	1

not

g	h
0	1
1	0

O comportamento da porta *or* difere do senso comum porque normalmente a palavra 'ou' é entendida como "ou exclusivo" e portanto uma alternativa exclui a outra, enquanto que a função $\vee$ define um "ou inclusivo" que admite uma das alternativas, além de ambas as alternativas. O operador "ou exclusivo", que somente admite uma das alternativas é definido pelo operador $\oplus$ e implementado pela porta lógica *xor*.

Como indicado na Seção 3.2 e nos Exercícios 4.2 e 4.4, as funções $\wedge$ e $\vee$ são associativas. Se a implementação destas funções é correta, então construção de portas com um número arbitrário de entradas é trivial.

As portas lógicas *and* e *or* correspondem às funções lógicas $\wedge$ e $\vee$. Do ponto de vista de implementação de sistemas digitais nas tecnologias disponíveis em 2021, os circuitos menores, portanto mais rápidos e que dissipam menos energia, são aqueles que implementam as portas lógicas *nand* e *nor*. A estrutura interna destas portas lógicas na tecnologia CMOS é apresentada no Capítulo 5. A Equação 4.2 e a Tabela 4.2 definem seus comportamentos, e a Figura 4.4 mostra os símbolos das portas *nand*, *nor* e *xor*.

$$
\begin{array}{l}
a, b, c, d, e, f, g, h, i : \mathbb{B} \\
\\
\textit{nand-2} : (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\textit{nand-2}(a, b, c) \;\equiv\; c = \neg(a \wedge b) \\
\\
\textit{nor-2} : (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\textit{nor-2}(d, e, f) \;\equiv\; f = \neg(d \vee e) \\
\\
\textit{xor-2} : (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\textit{xor-2}(g, h, i) \;\equiv\; i = \neg g \wedge h \;\vee\; g \wedge \neg h
\end{array}
\tag{4.2}
$$

Tabela 4.2: Tabelas Verdade das portas *nand*, *nor* e *xor*.

nand

a	b	c
0	0	1
0	1	1
1	0	1
1	1	0

nor

d	e	f
0	0	1
0	1	0
1	0	0
1	1	0

xor

g	h	i
0	0	0
0	1	1
1	0	1
1	1	0

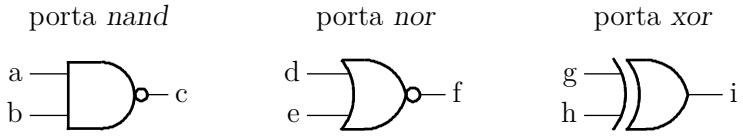


Figura 4.4: Símbolos das portas lógicas *nand*, *nor* e *xor*.

Um ponto importante e que merece ênfase é o seguinte: as definições da Equação 4.2 e da Tabela 4.2 são rigorosamente equivalentes e expressam a mesma informação sobre o comportamento dos circuitos. Da mesma forma, os símbolos gráficos das portas lógicas são apenas uma outra forma de representar aquela informação.

O *Teorema de DeMorgan* é útil na implementação e simplificação de circuitos, e é reescrito na Equação 4.3. Este teorema permite implementar funções lógicas mesmo que as portas lógicas disponíveis não sejam as 'necessárias' – por 'necessárias' entenda-se aquelas portas que correspondem fielmente às equações que descrevem o comportamento do circuito.

$$
\overline{x \wedge y} \;=\; \overline{x} \vee \overline{y} \qquad\qquad \overline{x \vee y} \;=\; \overline{x} \wedge \overline{y} \tag{4.3}
$$

DeMorgan nos permite expressar as funções das portas lógicas de mais de uma maneira, e estas equivalências podem ser muito úteis se a tecnologia de implementação impõe restrições quanto à variedades das portas disponíveis, como veremos na Seção 5.2.2. A Figura 4.5 mostra as equivalências entre as funções e as portas lógicas que as implementam.

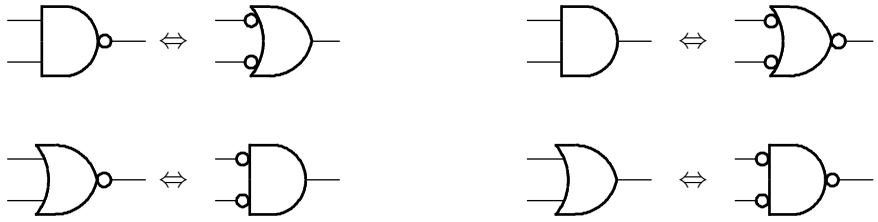
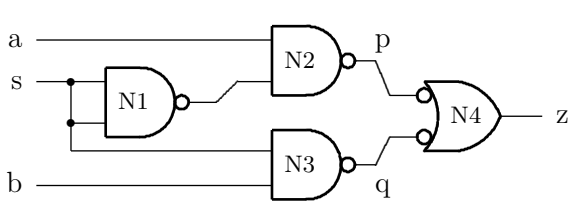


Figura 4.5: Equivalências entre funções lógicas ao aplicar DeMorgan.

As duas equivalências à esquerda são aplicações diretas da Equação 4.3: a porta *nand* equivale a um *or* com as entradas negadas; a porta *nor* equivale a um *and* com as entradas negadas. As equivalências à direita têm uma negação adicional nas saídas.

A Figura 4.6 mostra a implementação de um circuito multiplexador com quatro portas *nand*. A porta N1 é usada como inversor e a porta N4 é usada como *or*. As inversões nas saídas das portas N2 e N3 cancelam as inversões nas entradas da porta N4.



$$p = \overline{a \wedge \overline{s}}$$
$$q = \overline{b \wedge s}$$
$$z = \overline{p} \vee \overline{q}$$
$$z = \overline{\overline{a \wedge \overline{s}}} \vee \overline{\overline{b \wedge s}}$$
$$z = a \wedge \overline{s} \vee b \wedge s$$

Figura 4.6: Multiplexador implementado com quatro portas *nand*.

Uma aplicação importante, do ponto de vista prático, do Teorema de DeMorgan é facilitar a compreensão da função dos circuitos. A funcionalidade do circuito da Figura 4.6 é facilmente apreendida porque a porta lógica mais à direita, que tem a função de disjunção, está desenhada como uma porta *or*, com suas entradas invertidas. As duas portas no centro do circuito têm a função de conjunção, e estão desenhadas como portas *and* com as saídas invertidas. Como mencionado acima, as inversões nas saídas são canceladas pelas inversões nas entradas, pela propriedade da involução.

Exemplo 4.2 Supondo que o tempo de propagação de uma porta *nand* seja de 2ns (2×10^{-9}s), qual é o tempo de propagação do circuito da Figura 4.6?

O caminho mais longo atravessa as portas N1 $\rightsquigarrow$ N2 $\rightsquigarrow$ N4, e os sinais demoram 2ns para atravessar cada porta. Se uma das três entradas se altera, de 1 para 0 ou de 0 para 1, essa alteração será refletida de forma estável na saída após 6ns.

A palavra '*estável*' na frase anterior é importante: se todas as três entradas mudarem no mesmo instante, é possível que a saída z apresente um valor transitório após 4ns (N2 $\rightsquigarrow$ N4 ou N3 $\rightsquigarrow$ N4), que depois de mais 2ns sofreria nova alteração, para então permanecer estável. A saída do multiplexador somente pode ser usada com a confiança de que ela reflete os valores *estáveis* nas suas entradas depois de decorrido o tempo de propagação de 6ns. ◁

Exemplo 4.3 Frequentemente é necessário verificar se dois números são iguais, e para tanto emprega-se um *comparador de igualdade*. Iniciemos com números de dois bits; a Figura 4.7 mostra a tabela verdade para um comparador de números de dois bits, $A = \langle a_1 a_0 \rangle$ e $B = \langle b_1 b_0 \rangle$.

índice	$a_1 a_0$	$b_1 b_0$	ig
0	0 0	0 0	1
1	0 0	0 1	0
2	0 0	1 0	0
3	0 0	1 1	0
4	0 1	0 0	0
5	0 1	0 1	1
6	0 1	1 0	0
7	0 1	1 1	0
8	1 0	0 0	0
9	1 0	0 1	0
10	1 0	1 0	1
11	1 0	1 1	0
12	1 1	0 0	0
13	1 1	0 1	0
14	1 1	1 0	0
15	1 1	1 1	1

ig	$b_1\,b_0$ 00	01	11	10
$a_1\,a_0$ 00	0: 1	1: 0	3: 0	2: 0
01	4: 0	5: 1	7: 0	6: 0
11	12: 0	13: 0	15: 1	14: 0
10	8: 0	9: 0	11: 0	10: 1

Figura 4.7: Tabela verdade e Mapa de Karnaugh para o comparador.

Não é possível simplificar esta função por agrupamentos no Mapa de Karnaugh, mas é possível simplificá-la algebricamente.

$$
\begin{aligned}
ig \quad = \quad & (\overline{a_1} \wedge \overline{a_0} \wedge \overline{b_1} \wedge \overline{b_0}) \vee (\overline{a_1} \wedge a_0 \wedge \overline{b_1} \wedge b_0) \vee \\
& (a_1 \wedge \overline{a_0} \wedge b_1 \wedge \overline{b_0}) \vee (a_1 \wedge a_0 \wedge b_1 \wedge b_0) \\
\Leftrightarrow \quad & \text{associatividade, } 2\times \\
& \overline{a_1} \wedge (\overline{a_0} \wedge \overline{b_1} \wedge \overline{b_0}) \vee \overline{a_1} \wedge (a_0 \wedge \overline{b_1} \wedge b_0) \vee \\
& a_1 \wedge (\overline{a_0} \wedge b_1 \wedge \overline{b_0}) \vee a_1 \wedge (a_0 \wedge b_1 \wedge b_0) \\
\Leftrightarrow \quad & \text{distributividade, } 2\times \\
& \overline{a_1} \wedge [\, (\overline{a_0} \wedge \overline{b_1} \wedge \overline{b_0}) \vee (a_0 \wedge \overline{b_1} \wedge b_0) \,] \vee \\
& a_1 \wedge [\, (\overline{a_0} \wedge b_1 \wedge \overline{b_0}) \vee (a_0 \wedge b_1 \wedge b_0) \,] \\
\Leftrightarrow \quad & \text{associatividade, } 2\times \\
& \overline{a_1} \wedge [\, \overline{b_1} \wedge (\overline{a_0} \wedge \overline{b_0}) \vee \overline{b_1} \wedge (a_0 \wedge b_0) \,] \vee \\
& a_1 \wedge [\, b_1 \wedge (\overline{a_0} \wedge \overline{b_0}) \vee b_1 \wedge (a_0 \wedge b_0) \,] \\
\Leftrightarrow \quad & \text{distributividade, } 2\times \\
& \overline{a_1} \wedge [\, \overline{b_1} \wedge [\, (\overline{a_0} \wedge \overline{b_0}) \vee (a_0 \wedge b_0) \,] \,] \vee \\
& a_1 \wedge [\, b_1 \wedge [\, (\overline{a_0} \wedge \overline{b_0}) \vee (a_0 \wedge b_0) \,] \,] \\
\Leftrightarrow \quad & \text{definição } \oplus, \, 2\times \\
& \overline{a_1} \wedge [\, \overline{b_1} \wedge (a_0 \oplus \overline{b_0}) \,] \vee a_1 \wedge [\, b_1 \wedge (a_0 \oplus \overline{b_0}) \,] \\
\Leftrightarrow \quad & \text{associatividade, } 4\times \\
& (\overline{a_1} \wedge \overline{b_1}) \wedge (a_0 \oplus \overline{b_0}) \vee (a_1 \wedge b_1) \wedge (a_0 \oplus \overline{b_0}) \\
\Leftrightarrow \quad & \text{distributividade} \\
& (a_0 \oplus \overline{b_0}) \wedge [\, (\overline{a_1} \wedge \overline{b_1}) \vee (a_1 \wedge b_1) \,] \\
\Leftrightarrow \quad & \text{definição } \oplus \\
& (a_0 \oplus \overline{b_0}) \wedge (a_1 \oplus \overline{b_1}) \\
\Leftrightarrow \quad & \text{definição } \oplus \\
& \overline{(a_0 \oplus b_0)} \wedge \overline{(a_1 \oplus b_1)}
\end{aligned}
$$

Esta função pode ser implementada com duas portas *xnor* e a uma porta *and*, e o circuito é mostrado na Figura 4.8. Este circuito pode ser estendido da maneira óbvia para a comparação de vetores de bits com qualquer largura. ◁

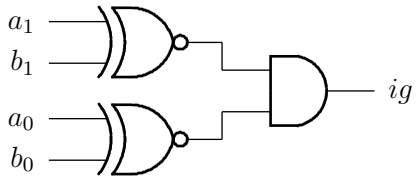


Figura 4.8: Comparador de igualdade para números de dois bits.

Exemplo 4.4 Os *Códigos de Gray* são usados em sistemas que devem operar de forma segura, para alguma definição de 'seguro'. Considere, por exemplo, o eixo que controla a posição vertical de um canhão. A medida da posição angular do eixo é indicada por uma série de furos ao longo do raio de um disco que é rigidamente fixado ao eixo. A cada intervalo, um conjunto radial de furos indica o ângulo do eixo com relação à horizontal, de forma que cada combinação de furos indica o ângulo absoluto do eixo. O disco do medidor de posição angular é mostrado na Figura 4.9.

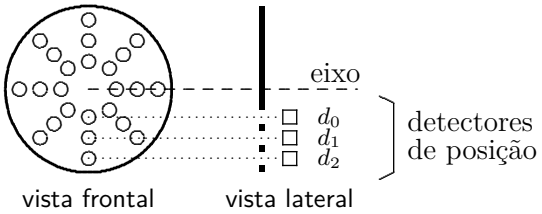


Figura 4.9: Disco do detector de posição angular.

Suponha ainda que o sistema de numeração binária com três bits é usado para marcar os ângulos. O que pode ocorrer quando o eixo está mudando de posição, de 011 para 100? Por problemas de construção, ou desgaste, o ângulo poderia ser medido pelos detectores na sequência $011 \rightarrow 001 \rightarrow 000 \rightarrow 100$, cuja correspondente em decimal é $3 \rightarrow 1 \rightarrow 0 \rightarrow 4$. Esta sequência de ângulos certamente provocaria uma pane no acionador vertical do canhão.

Uma solução para este problema consiste em usar uma codificação na qual só ocorre uma única troca de bit entre cada dois vizinhos na sequência – só uma posição troca de 1 para 0 ou de 0 para 1. A Tabela 4.3 mostra a geração de um Código de Gray em três bits.

Tabela 4.3: Geração de um Código de Gray em três bits.

1 bit	refl.	2 bits	refl.	3 bits
0	0	**0** 0	00	**0** 00
1	1	**0** 1	01	**0** 01
	1	**1** 1	11	**0** 11
	0	**1** 0	10	**0** 10
			10	**1** 10
			11	**1** 11
			01	**1** 01
			00	**1** 00

O código de um bit consiste de dois casos, 1 e 0, e é mostrado na coluna da esquerda. A construção do código de dois bits se dá em dois passos: (i) o código de um bit é copiado de forma refletida, como mostra a segunda coluna; (ii) à metade original é acrescentado um 0 à esquerda, e à parte refletida é acrescentado 1 à esquerda. As duas colunas da direita repetem estes dois passos na construção do código de três bits. Note que o 100 da última linha é adjacente ao 000 da primeira linha.

A Tabela 4.4 associa o código de três bits aos índices da Tabela 4.3. Esta é também a tabela verdade para as três funções, g_2, g_1, g_0, que geram o dígito apropriado a partir das entradas $\langle d_2 d_1 d_0 \rangle$.

Tabela 4.4: Tabela verdade para o Código de Gray em três bits.

índice	$d_2 d_1 d_0$	$g_2 g_1 g_0$
0	0 0 0	0 0 0
1	0 0 1	0 0 1
2	0 1 0	0 1 1
3	0 1 1	0 1 0
4	1 0 0	1 1 0
5	1 0 1	1 1 1
6	1 1 0	1 0 1
7	1 1 1	1 0 0

Ao inspecionar a Tabela 4.4, percebe-se que $g_2 = d_2$, e a implementação de g_2 é apenas um fio ligado a d_2. Da tabela verdade obtemos

$$g_2 = d_2, \qquad g_1 = d_2 \wedge \overline{d_1} \vee \overline{d_2} \wedge d_1 \quad \text{e} \quad g_0 = \overline{d_1} \wedge d_0 \vee d_1 \wedge \overline{d_0}.$$

As funções g_1 e g_0 podem ser implementadas com portas *xor*. ◁

Exemplo 4.5 Supondo que os tempos de propagação de portas *and* e *or* sejam de 2ns, e o de um inversor de 1ns, qual é o tempo de propagação do circuito que gera Código de Gray do Exemplo 4.4?

O caminho mais longo atravessa um inversor, uma porta *and* e uma porta *or*. Logo, o tempo de propagação, no caminho mais longo, é $1 + 2 + 2 = 5$ns. ◁

Exemplo 4.6 Considere a função $\wedge$ de 4 entradas, implementada com três portas de duas entradas *and*-2: $c = \bigwedge(x, y, z, w) = x \wedge (y \wedge (z \wedge w))$. Este circuito é mostrado no lado esquerdo da Figura 4.10 e seu tempo de propagação é dado pelo caminho que atravessa as três portas *and* encadeadas: $T_{\text{cadeia}} = 3 \times T_A$.

Outra implementação para $\wedge$ de 4 entradas é obtida se tirarmos proveito da associatividade da conjunção: $a = \bigwedge(x, y, z, w) = (x \wedge y) \wedge (z \wedge w)$. Este circuito é mostrado à direita na Figura 4.10, e seu tempo de propagação é menor do que o da cadeia de *and*s: $T_{\text{árvore}} = 2 \times T_A$.

Se um operador é associativo, o número de entradas N é uma potência de dois e $N \geqslant 4$, então a implementação em árvore tem um tempo de propagação menor do que o da cadeia de portas:

$$(N-1)T_P = T_{\text{cadeia}} > T_{\text{árvore}} = (\log_2 N)T_P\,.$$

O número de portas de duas entradas é igual nos dois casos. ◁

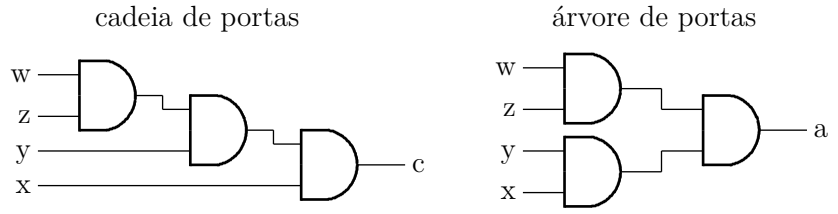


Figura 4.10: Duas implementações para *and-4* com *and-2*.

Exemplo 4.7 Qual o tempo de propagação de um circuito que computa a conjunção de oito sinais, construído com duas árvores como as do Exemplo 4.6?

A implementação do circuito, com duas árvores de altura dois, é

$$a = \bigwedge(x, y, z, w, p, q, r, s) = \bigwedge(x, y, z, w) \wedge \bigwedge(p, q, r, s).$$

Supondo que o tempo de propagação de uma porta *and*-2 seja de 2ns, cada ramo da árvore tem tempo propagação de 4ns, enquanto que o tempo da árvore completa é de 6ns. Como indicado no Exemplo 4.6, $T_{\text{árvore}} = (\log_2 8) \times 2 = 6$ns. Para uma cadeia com sete portas *and*-2, o tempo de propagação é de 14ns. Compare as curvas para $y = x$ e $z = \log_2(x)$. ◁

4.2.1 Conjunto Mínimo de Operadores

Do ponto de vista de tecnologia de fabricação de circuitos integrados, uma questão importante é relacionada ao conjunto mínimo de operadores que deve ser oferecido aos projetistas para permitir a implementação de qualquer função lógica. Pode-se facilmente provar que, com exatamente um dentre os seguintes conjuntos de operadores, é possível implementar qualquer função em $\mathbb{B}$: $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\neg, \Rightarrow\}$. Note que os dois primeiros são as funções *nand* e *nor*, que são as portas lógicas básicas da tecnologia CMOS.

Tabela 4.5: Funções de duas variáveis.

ab	0	$\wedge$	f_2	a	f_4	b	$\neq$	$\vee$	*nor*	$=$	$\neg b$	$\Rightarrow$	$\neg a$	$\Leftarrow$	*nand*	1
00	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
01	0	0	0	0	1	1	1	1	0	0	0	1	1	0	1	1
10	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1	1
11	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

A Tabela 4.5 contém as 16 funções de duas variáveis em $\mathbb{B}$, e indica as funções mais conhecidas. Note que as funções $\neq$ e $=$ também são conhecidas como *xor* ou *ou-exclusivo*, e *xnor*, respectivamente. As funções $\Rightarrow$ e $\Leftarrow$ são implicação lógica: $a \Rightarrow b$, e $b \Rightarrow a$. Quais são as funções identificadas por f_2 e f_4?

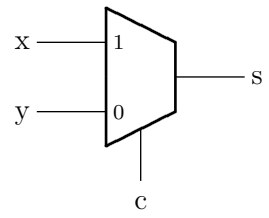
4.3 Circuitos Combinacionais Básicos

Esta seção contém a especificação e implementações de três circuitos combinacionais básicos que permitem escolher um dentre um conjunto de valores. Os circuitos são chamados de multiplexador, demultiplexador e decodificador. Tais circuitos são relevantes porque são empregados de inúmeras maneiras como blocos básicos na construção de circuitos mais complexos.

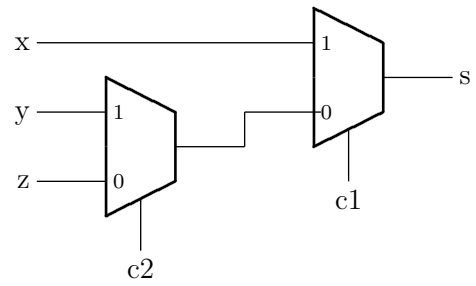
4.3.1 Multiplexador

Como vimos na Seção 3.3, o operador condicional equivale ao comando `if then else`. O circuito que implementa o condicional é chamado de *multiplexador*, e é representado em diagramas por um trapézio, como mostra a Figura 4.11. No lado mais largo ficam as entradas, no lado estreito a saída, e no topo ou na base fica o sinal de controle, ou o sinal de seleção. A Figura 4.11 mostra três `ifs` e os circuitos equivalentes. No terceiro circuito o sinal $c2$ é compartilhado pelos dois multiplexadores da esquerda.

```
if c then x else y
```




```
if c1 then x
      else if c2 then y else z
```




```
if c1 then if c2 then x else y
      else if c2 then z else w
```

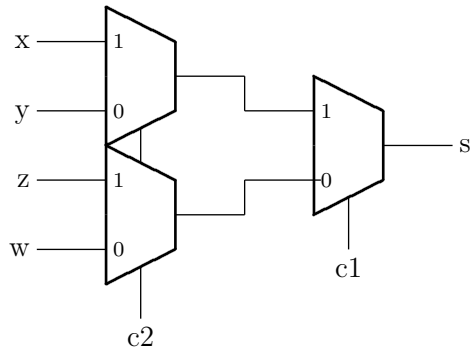


Figura 4.11: Seleção com multiplexadores de duas entradas.

Um *multiplexador de duas entradas* é um circuito com duas entradas a, b, uma entrada de controle s e uma saída z. A entrada de controle s permite escolher qual dentre as entradas a e b será apresentada na saída, conforme especificado pela Equação 4.4. A Figura 4.12 mostra o circuito e o símbolo do multiplexador de duas entradas.

$$
\begin{array}{l}
a, b, s, z : \mathbb{B} \\
\mathit{mux}\text{-}2 : \mathbb{B} \times (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\mathit{mux}\text{-}2(a, b, s, z) \ \equiv \ z = b \lhd s \rhd a
\end{array}
\tag{4.4}
$$

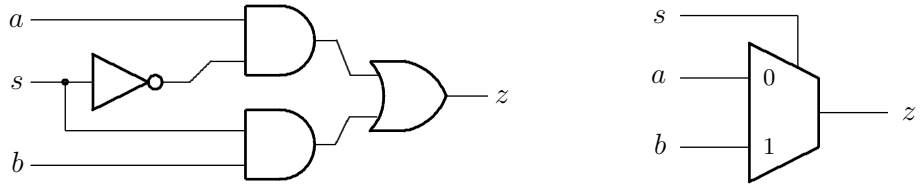


Figura 4.12: Multiplexador de duas entradas.

A expressão "$\mathit{mux}\text{-}2 : \mathbb{B} \times (\mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B}$" define o *tipo* do circuito. O multiplexador possui uma entrada de controle do tipo $\mathbb{B}$, uma entrada que é um par de $\mathbb{B}$, com tipo $(\mathbb{B} \times \mathbb{B})$, e produz uma saída de tipo $\mathbb{B}$.

Multiplexadores com maior número de entradas podem ser construídos pela composição de multiplexadores de duas entradas. A Equação 4.5 define um multiplexador de 4 entradas – preste atenção aos índices das variáveis de controle que estão representados em base-2.

$$
\begin{array}{l}
S : \mathbb{B}^2 \\
A : \mathbb{B}^4 \\
\mathit{mux}\text{-}4 : (\mathbb{B} \times \mathbb{B}) \times (\mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B}) \mapsto \mathbb{B} \\
\mathit{mux}\text{-}4(a_{0..3}, s_{1..0}, z) \ \equiv \ z = (a_{11} \lhd s_0 \rhd a_{10}) \lhd s_1 \rhd (a_{01} \lhd s_0 \rhd a_{00})
\end{array}
\tag{4.5}
$$

Como definido na Equação 4.5, o *mux*-4 é uma árvore de *mux*-2, e esta árvore tem 'altura' dois, contando-se o número de multiplexadores entre a saída e as entradas. Este processo pode ser levado adiante na construção de multiplexadores de qualquer número de entradas, como indica a Equação 4.6.

$$
\begin{array}{l}
S : \mathbb{B}^n \\
A : \mathbb{B}^{2^n} \\
\mathit{mux}\text{-}2^n : (\Pi_n \mathbb{B}) \times (\Pi_{2^n} \mathbb{B}) \mapsto \mathbb{B} \\
\mathit{mux}\text{-}2^n(a_0 \cdots a_{2^n-1}, s_{n-1} \cdots s_0, z) \ \equiv \\
\qquad z = ((a_{n-1} \lhd s_0 \rhd a_{n-2}) \cdots) \lhd s_{n-1} \rhd (\cdots (a_1 \lhd s_0 \rhd a_0))
\end{array}
\tag{4.6}
$$

A Figura 4.13 mostra uma implementação para o circuito do multiplexador de 2^n entradas (a_0 a a_{2^n-1}) por uma árvore de *mux*-2. Esta árvore tem altura de um mais a árvore de multiplexadores de 2^{n-1} entradas.

Um multiplexador de 2^n entradas é um circuito que apresenta em sua saída o valor da entrada a_k, quando o número representado pelos bits da entrada de seleção S é $k = num(S)$. Esta formulação útil é dada na Equação 4.7.

$$
\begin{array}{l}
S : \mathbb{B}^n \\
A : \mathbb{B}^{2^n} \\
\mathit{mux}\text{-}2^n : (\mathbb{B}^n \times \mathbb{B}^{2^n}) \mapsto \mathbb{B} \\
\mathit{mux}\text{-}2^n(a_0 \cdots a_{2^n-1}, s_{n-1} \cdots s_0, z) \ \equiv \ z = a_{num(S)}
\end{array}
\tag{4.7}
$$

Se, ao invés de usarmos *mux*-2 como elemento base na construção de multiplexadores, usarmos um *mux*-4, então um *mux*-16 pode ser construído com 4 *mux*-4 no primeiro nível da árvore, e um *mux*-4 no segundo nível. Esta árvore tem altura 2 porque $\log_4 16 = 2$.

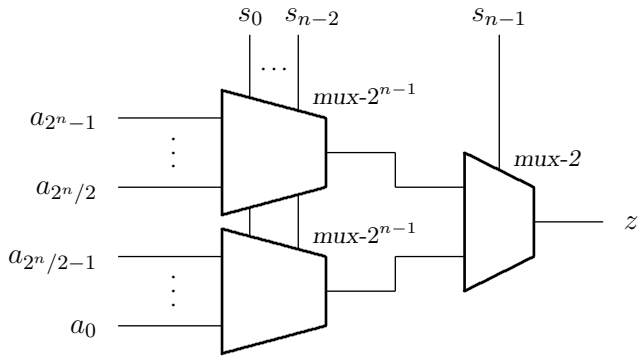


Figura 4.13: Multiplexador de 2^n entradas: um *mux-2* e dois *mux*-2^{n-1}.

Exemplo 4.8 Considere a função de três variáveis lógicas $t(p, q, r)$ definida na Figura 4.14. A função t pode ser implementada trivialmente com um multiplexador 8 entradas, bastando ligar à entrada i do multiplexador o valor de t na linha da tabela verdade com índice i. A Figura 4.14 mostra a implementação de t com um mux-8, que é a mera reprodução da tabela verdade. ◁

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

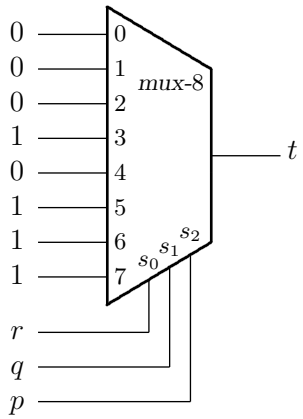


Figura 4.14: Implementação de função de três variáveis com um *mux-8*.

Exemplo 4.9 A função t do Exemplo 4.8 pode ser implementada com um multiplexador de 4 entradas. A tabela verdade de t é repetida na Figura 4.15, com as linhas agrupadas de duas em duas, para permitir a elisão da entrada r. A figura mostra a implementação com o mux-8 na qual r substitui alguns dos valores fixos em 0 ou 1 nas entradas, e a implementação com um mux-4. A aplicação desta técnica, em geral, permite usar um multiplexador com "metade da altura", mais um inversor. ◁

A otimização da função t equivale a uma redução no tamanho da tabela que define a função, de uma tabela com oito linhas (t) para uma de quatro linhas (t'), sendo que em algumas linhas da imagem da função, esta contém a variável r ao invés de 1 ou 0. A função t' equivale a uma tabela de 4 linhas, com uma das variáveis da função "incluída na linha". As linhas de t têm as seguintes correspondências em t':

$$(p \wedge q) \Rightarrow (t' = 1), \quad (p \wedge \overline{q}) \Rightarrow (t' = r) \quad \text{e} \quad (\overline{p} \wedge q) \Rightarrow (t' = r).$$

índice	t	p	q	r	t'
0	0	0	0	0	0
1	0	0	0	1	
2	0	0	1	0	r
3	1	0	1	1	
4	0	1	0	0	r
5	1	1	0	1	
6	1	1	1	0	1
7	1	1	1	1	

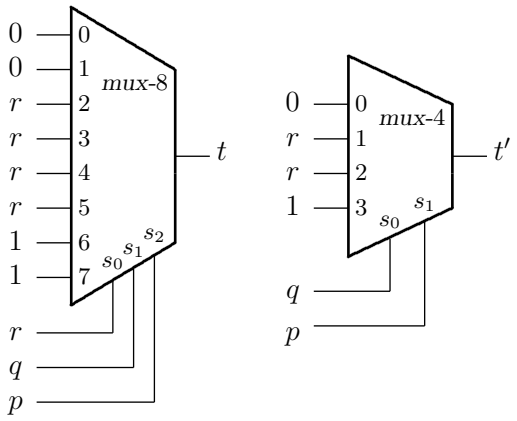


Figura 4.15: Implementação de função de três variáveis com um *mux-4*.

Exemplo 4.10 Vejamos como implementar a função $c(p, q, r, s)$ do Exercício 3.9:

$$c = (\neg p \wedge \neg q \wedge r) \vee \neg(s \wedge (p \vee r \vee \neg q))$$

A tabela verdade de c é mostrada na Figura 4.16, e a função c' resulta do agrupamento das linhas de duas em duas, para a elidir a entrada s. A função c pode ser implementada diretamente um multiplexador de 16 entradas com uma entrada associada a cada linha da tabela verdade. A figura mostra também a implementação com um mux-8, com $\overline{s}$ ligado nas entradas que lhe correspondem em c'. Um bom exercício para a aluna aplicada é a tentativa de implementar c'' com um mux-4. ◁

índice	p	q	r	s	c	c'
0	0	0	0	0	1	$\overline{s}$
1	0	0	0	1	0	
2	0	0	1	0	1	1
3	0	0	1	1	1	
4	0	1	0	0	1	1
5	0	1	0	1	1	
6	0	1	1	0	1	$\overline{s}$
7	0	1	1	1	0	
8	1	0	0	0	1	$\overline{s}$
9	1	0	0	1	0	
10	1	0	1	0	1	$\overline{s}$
11	1	0	1	1	0	
12	1	1	0	0	1	$\overline{s}$
13	1	1	0	1	0	
14	1	1	1	0	1	$\overline{s}$
15	1	1	1	1	0	

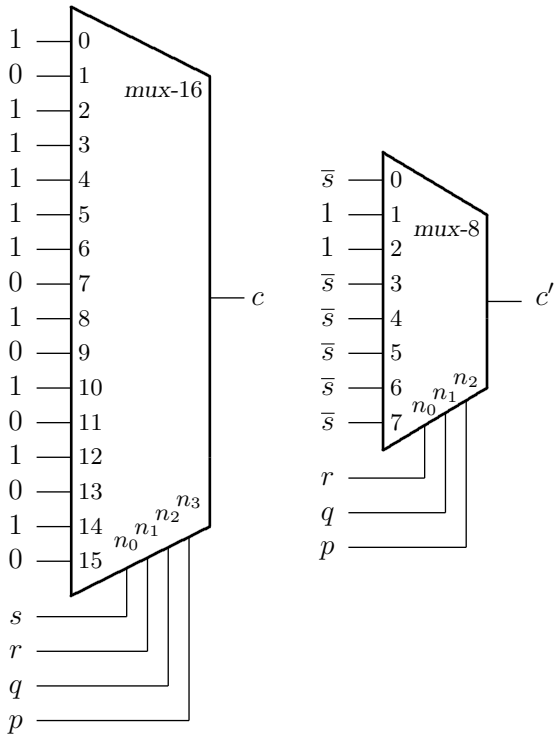


Figura 4.16: Implementação de função de quatro variáveis com um *mux-8*.

Exemplo 4.11 Para somar dois números representados em *notação científica*, ou em *ponto flutuante*, deve-se deslocar o expoente do menor número para 'emparelhar' as vírgulas e então efetuar a soma. Vejamos o projeto do circuito que descobre de quantas posições a parte fracionária deve ser deslocada.

Considere um circuito que tem como entrada um vetor de quatro bits $B = \langle b_3, b_2, b_1, b_0 \rangle$, e duas saídas: a saída V é 1 sempre que os quatro bits de B são zero, e a saída $N = \langle n_1, n_0 \rangle$ é o número (índice) da posição do bit mais à esquerda (mais significativo) que *não é zero* – o número N indica qual é a posição do 'primeiro' bit em 1.

A Tabela 4.6 mostra as combinações de entradas e saídas para a detecção do bit mais significativo de um quarteto de bits. Quando $B = \langle 0,0,0,0 \rangle$, então $V = 1$ e N pode ser ignorado porque não há nenhum bit em 1. Quando $V = 0$, N deve indicar qual é o bit mais significavo que não é zero. Se $b_k = 1$, então o valor dos $b_i, i < k$ é irrelevante, e isso é denotado com 'X' na tabela verdade.

Da tabela obtemos $n_1 = b_3 \vee b_2$, $n_0 = b_3 \vee (\overline{b_3} \wedge \overline{b_2} \wedge b_1)$ e $V = \overline{b_3} \wedge \overline{b_2} \wedge \overline{b_1} \wedge \overline{b_0}$. ◁

Tabela 4.6: Tabela verdade do bit mais significativo de um quarteto.

b_3	b_2	b_1	b_0	n_1	n_0	V
0	0	0	0	0	0	1
0	0	0	1	0	0	0
0	0	1	X	0	1	0
0	1	X	X	1	0	0
1	X	X	X	1	1	0

Exemplo 4.12 Vejamos como usar duas cópias do circuito do Exemplo 4.11 para projetar um circuito com uma entrada de 8 bits $C = \langle c_7, c_6, \ldots, c_2, c_1, c_0 \rangle$, e duas saídas: a saída V é 1 sempre que os oito bits de C são zero, e a saída $M = \langle m_2, m_1, m_0 \rangle$ é o número (índice) da posição do bit mais à esquerda (mais significativo) que não é zero.

A Tabela 4.7 mostra as combinações de entradas e saídas para a detecção do bit mais significativo de um quarteto de bits. Da tabela observamos que se $V = v_1 \wedge v_0 = 1$ então M pode ser ignorado porque não há nenhum bit em 1.

Na metade superior da tabela, quando $v_1 = 1$, $m_2 = 0$ e $\langle m_1, m_0 \rangle$ podem ser obtidos com o circuito do Exemplo 4.11 aplicado aos bits $\langle c_3, c_2, c_1, c_0 \rangle$.

Na metade inferior da tabela (em itálico), quando $v_1 = 0$, $m_2 = 1$ e $\langle m_1, m_0 \rangle$ podem ser obtidos com o circuito do Ex. 4.11 aplicado aos bits $\langle c_7, c_6, c_5, c_4 \rangle$.

O bit v_1 pode ser usado para escolher um dentre dois pares de sinais. O circuito completo é mostrado na Figura 4.17. As funções são $m_2 = \overline{v_1}$, $m_1 = s_1 \lhd v_1 \rhd r_1$, $m_0 = s_0 \lhd v_1 \rhd r_0$ e $V = v_1 \wedge v_0$. ◁

Espaço em branco proposital.

Tabela 4.7: Tabela verdade do bit mais significativo de um octeto.

c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	m_2	m_1	m_0	v_1	v_0
0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	1	X	0	0	1	1	0
0	0	0	0	0	1	X	X	0	1	0	1	0
0	0	0	0	1	X	X	X	0	1	1	1	0
0	0	0	1	X	X	X	X	*1*	*0*	*0*	*0*	X
0	0	1	X	X	X	X	X	*1*	*0*	*1*	*0*	X
0	1	X	X	X	X	X	X	*1*	*1*	*0*	*0*	X
1	X	X	X	X	X	X	X	*1*	*1*	*1*	*0*	X

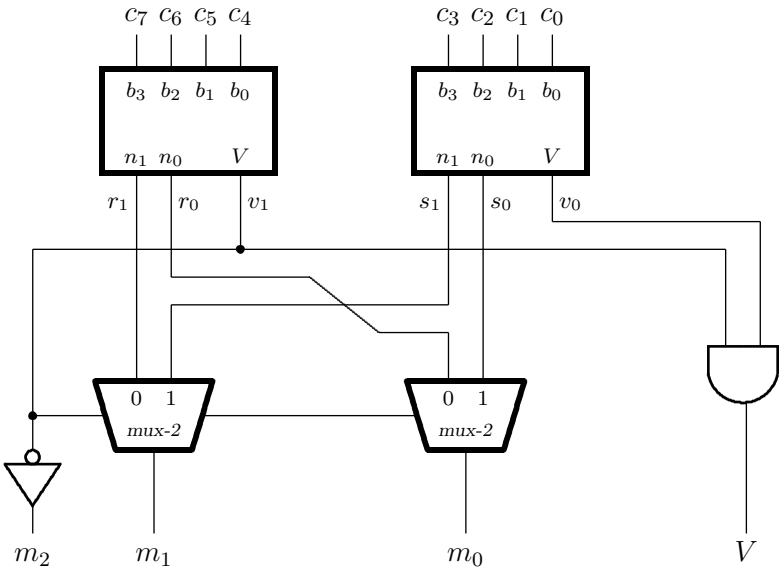


Figura 4.17: Circuito que identifica o bit mais significativo de um octeto.

4.3.2 Decodificador

Um *decodificador de duas saídas* é um circuito com uma entrada de controle s e duas saídas y_0, y_1. A entrada de controle s permite escolher qual das duas saídas será ativada, conforme definido pela Equação 4.8 e na tabela verdade na Figura 4.18. A figura mostra também o circuito que implementa o decodificador de duas saídas e o seu símbolo.

$$\begin{array}{l} s : \mathbb{B} \\ Y : \mathbb{B}^2 \\ \mathit{decod}\text{-}2 : \mathbb{B} \mapsto (\mathbb{B} \times \mathbb{B}) \\ \mathit{decod}\text{-}2(s, y_0, y_1) \;\equiv\; \left\{ \begin{array}{lcl} y_0 & = & 0 \lhd s \rhd 1 \\ y_1 & = & 1 \lhd s \rhd 0 \end{array} \right. \end{array} \tag{4.8}$$

s	y_1	y_0
0	0	1
1	1	0

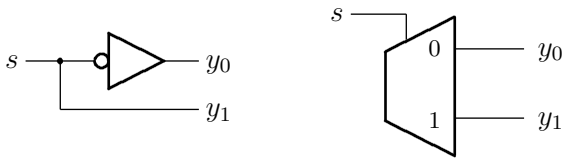


Figura 4.18: Decodificador de duas saídas.

No caso geral, um decodificador com n bits de entrada seleciona e ativa exatamente uma dentre as 2^n saídas. Posto de outra forma, este circuito *decodifica* o número codificado na entrada de controle: se $S = K$, a saída $\mathit{num}(K)$ é ativada, ficando todas as demais inativas. Um decodificador de 2^n saídas é especificado pela Equação 4.9.

$$\begin{array}{l} S : \mathbb{B}^n \\ Y : \mathbb{B}^{2^n} \\ \mathit{decod}\text{-}2^n : \mathbb{B} \mapsto (\Pi_n \mathbb{B}) \mapsto (\Pi_{2^n} \mathbb{B}) \\ \mathit{decod}\text{-}2^n(s_{n-1} \cdots s_0, y_0 \cdots y_{2^n-1}) \;\equiv\; y_i = 1 \lhd (\mathit{num}(S) = i) \rhd 0 \end{array} \tag{4.9}$$

A Figura 4.19 mostra a tabela verdade e o circuito de um decodificador de 4 saídas. A saída y_0 é ativada quando $S = \langle 0, 0 \rangle = 0$; a saída y_2 é ativada quando $S = \langle 1, 0 \rangle = 2$.

s_1	s_0	y_3	y_2	y_1	y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

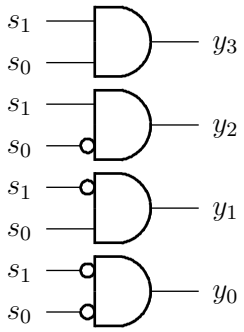


Figura 4.19: Decodificador de quatro saídas.

Decodificadores são também chamados de *seletores* porque, ao decodificar um dentre os n possiveis valores na entrada, este circuito seleciona uma das 2^n saídas.

Exemplo 4.13 Um decodificador pode ser usado para computar a *função inversa* de um multiplexador: a partir de uma entrada e, um sinal de controle com n bits direciona e para uma das 2^n saídas. A Figura 4.20 mostra um circuito que de-multiplexa, implementado com um seletor de duas saídas. ◁

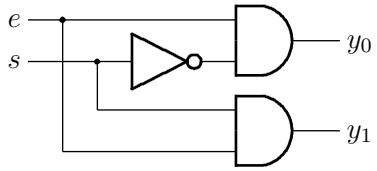


Figura 4.20: Circuito que computa a função inversa de um *mux-2*.

Exemplo 4.14 Considerando que os tempos de propagação das portas lógicas são aqueles indicados na Tabela 4.8, quais são os tempos de propagação do decodificador da Figura 4.18, e os decodificadores da Figura 4.21?

O tempo de propagação do *decod-2* é o de um inversor: $T_{\text{dec}-2} = 2$ns. O tempo de propagação do decodificador de 4 saídas é a soma dos tempos das portas no caminho mais longo, que é um inversor mais uma porta *and-2*: $T_{\text{dec}-4} = 2 + 4 = 6$ns. Com oito saídas, $T_{\text{dec}-8} = 2 + 6 = 8$ns. ◁

Tabela 4.8: Tempo de propagação de portas lógicas.

porta	T_p	porta	T_p
not	2	*xor-2*	6
and-2	4	*or-2*	4
and-3	6	*or-3*	6
and-4	8	*or-4*	8
and-5	12	*or-5*	12
and-6	16	*or-6*	16

Tempos em nanosegundos.

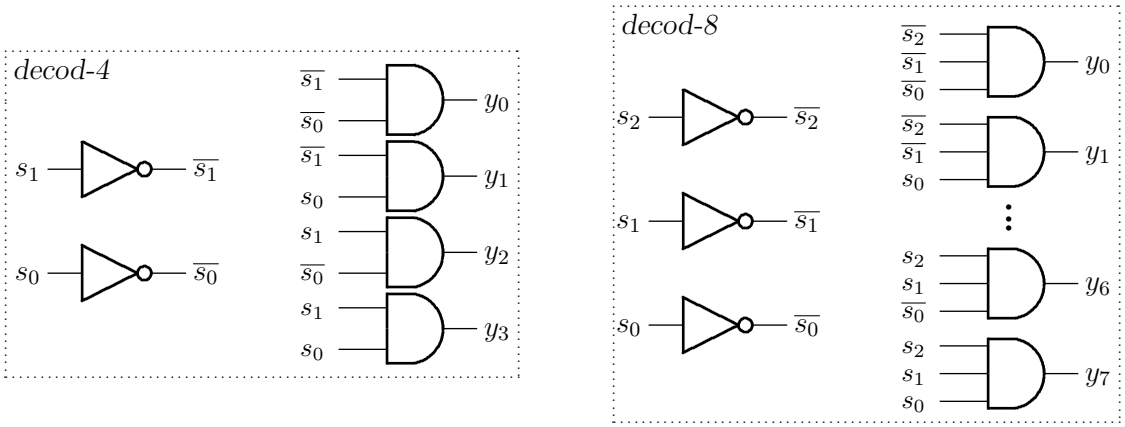


Figura 4.21: Decodificadores de quatro e de oito saídas.

Exemplo 4.15 Um decodificador, assim como um multiplexador, pode ser usado para implementar uma função definida por uma tabela verdade de maneira simples e eficiente e que é, em espírito,

similar àquela discutida na Seção 4.3.1. As saídas do decodificador geram os termos com os produtos da função, e uma porta *or* efetua a soma dos produtos. A Figura 4.22 mostra a implementação da função t com um decodificador e uma porta *or* de quatro entradas – a porta está desenhada com "asas de morcego" para acomodar todas as entradas. ◁

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

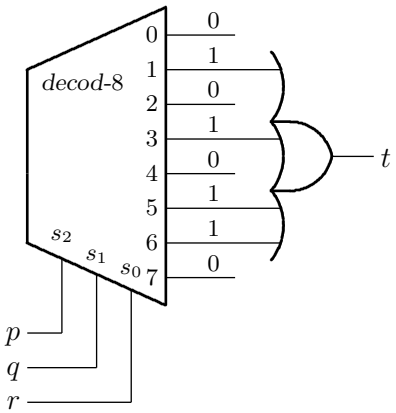


Figura 4.22: Implementação de função de três variáveis com um *decod-8*.

Exemplo 4.16 Considere um computador em que oito dispositivos disputam pela atenção do processador. Um dispositivo solicita a atenção do processador ao ativar um sinal de requisição de serviço. Quando o processador atende ao pedido do dispositivo, aquele remove o pedido de atenção.

O projetista do computador, conhecendo as especificidades de cada dispositivo escolhe e atribui distintas prioridades a cada dispositivo. O dispositivo mais importante – talvez porque produza/consuma o maior volume de dados – recebe a prioridade mais alta, que é 7 no nosso exemplo, e o dispositivo menos importante – talvez porque seja lento com relação aos demais – recebe a prioridade mais baixa, que é 0.

Para evitar que um dispositivo de baixa prioridade impeça o processador de atender a um dispositivo de prioridade mais alta, emprega-se um *decodificador de prioridades* que informa ao processador qual é o dispositivo de maior prioridade que está solicitando atenção num determinado instante. Se os dispositivos 5, 4 e 1 estão solicitando atenção, o decodificador indica que o pedido mais prioritário é o do dispositivo 5.

O decodificador tem oito entradas e quatro saídas. Às entradas $I = \langle i_7 \ldots i_0 \rangle$ são ligados os sinais de requisição, e as três saídas $A = \langle a_2, a_1, a_0 \rangle$ são ligadas ao processador e indicam o nível de prioridade da requisição pendente. A quarta saída, p, indica se há alguma requisição ativa: $p \wedge (A = 0)$ indica que somente a requisição de nível 0 está ativa. A Figura 4.23 mostra o circuito que implementa as funções a_2, a_1 e a_0. A função p não é mostrada.

O bit a_2 fica em 1 sempre que uma dentre i_7, i_6, i_5, i_4 estiver ativa.

O bit a_1 fica em 1 sempre que i_7 ou i_6 estejam ativas, ou se ambas i_5 e i_4 estejam inativas e i_3 ou i_2 estejam ativas.

O bit a_0 fica em 1 sempre que i_7 ou i_5 estejam ativas e i_6 inativa, ou i_3 ativa com i_6 e i_4 inativas, ou i_1 ativa com i_6, i_4 e i_2 inativas.

A saída p é a disjunção das oito entradas.

A implementação mostrada neste exemplo é similar ao circuito integrado[3] 74148. ◁

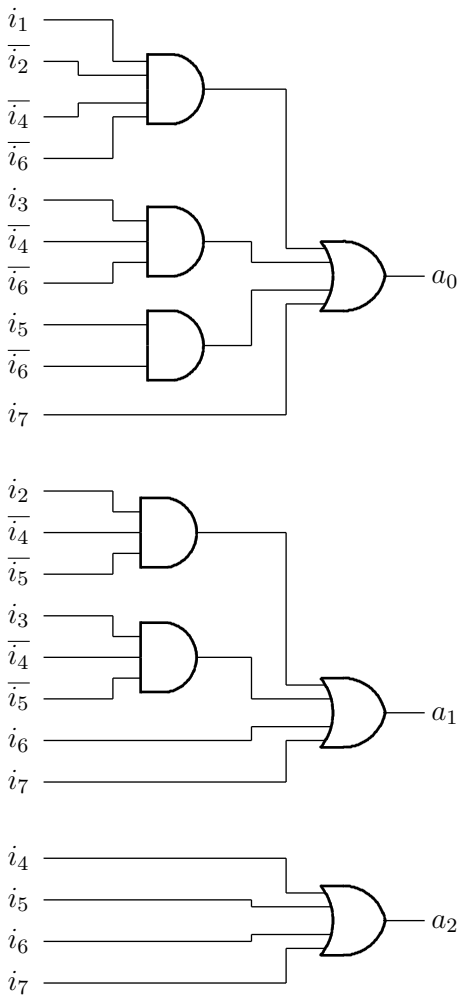


Figura 4.23: Decodificador de prioridades $8 \mapsto 3$.

4.3.3 Demultiplexador

Um *demultiplexador de duas saídas* é um circuito com uma entrada a, um sinal de controle s e duas saídas y_0, y_1. A entrada de controle s determina em qual das duas saídas é apresentada a entrada, conforme define a Equação 4.10. A Figura 4.24 mostra o circuito e o símbolo do demultiplexador de duas saídas, que é aquele do Exemplo 4.13. Numa analogia aquática, o multiplexador se assemelha a um funil, enquanto que o demultiplexador lembra um regador.

$$\begin{array}{l} a, s : \mathbb{B} \\ Y : \mathbb{B}^2 \\ \mathit{demux}\text{-}2 : \mathbb{B} \mapsto \mathbb{B} \mapsto (\mathbb{B} \times \mathbb{B}) \\ \mathit{demux}\text{-}2(a, s, y_0, y_1) \equiv \left\{ \begin{array}{lcl} y_0 & = & 0 \lhd s \rhd a \\ y_1 & = & a \lhd s \rhd 0 \end{array} \right. \end{array} \tag{4.10}$$

[3] A especificação do 74148 pode ser obtida de `http://www.ti.com/product/SN74LS148`.

O *tipo* do demultiplexador é definido por suas duas entradas do tipo $\mathbb{B}$ ($\mathbb{B} \mapsto \mathbb{B}$) e pela saída que tem tipo par de $\mathbb{B}$ ($\mathbb{B} \times \mathbb{B}$).

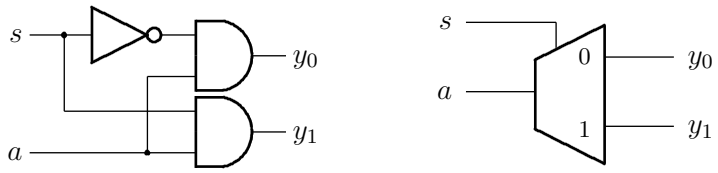


Figura 4.24: Demultiplexador de duas saídas.

Demultiplexadores com maior número de entradas podem ser construídos pela composição de demultiplexadores de duas saídas.

$$
\begin{array}{l}
a : \mathbb{B} \\
S : \mathbb{B}^2 \\
Y : \mathbb{B}^4 \\
\textit{demux}\text{-}4 : \mathbb{B} \mapsto (\mathbb{B} \times \mathbb{B}) \mapsto (\mathbb{B} \times \mathbb{B} \times \mathbb{B} \times \mathbb{B}) \\
\textit{demux}\text{-}4(a, s_1, s_0, y_0, y_1, y_2, y_3) \equiv \left\{ \begin{array}{lcl} y_0 & = & 0 \lhd s_1 \rhd (0 \lhd s_0 \rhd a) \\ y_1 & = & 0 \lhd s_1 \rhd (a \lhd s_0 \rhd 0) \\ y_2 & = & (0 \lhd s_0 \rhd a) \lhd s_1 \rhd 0 \\ y_3 & = & (a \lhd s_0 \rhd 0) \lhd s_1 \rhd 0 \end{array} \right.
\end{array}
\tag{4.11}
$$

Como no caso dos multiplexadores, este processo pode ser levado adiante na construção de demultiplexadores de qualquer número de saídas. A Figura 4.25 mostra um *demux*-8 construído com sete *demux*-2, pela extensão óbvia da Equação 4.11.

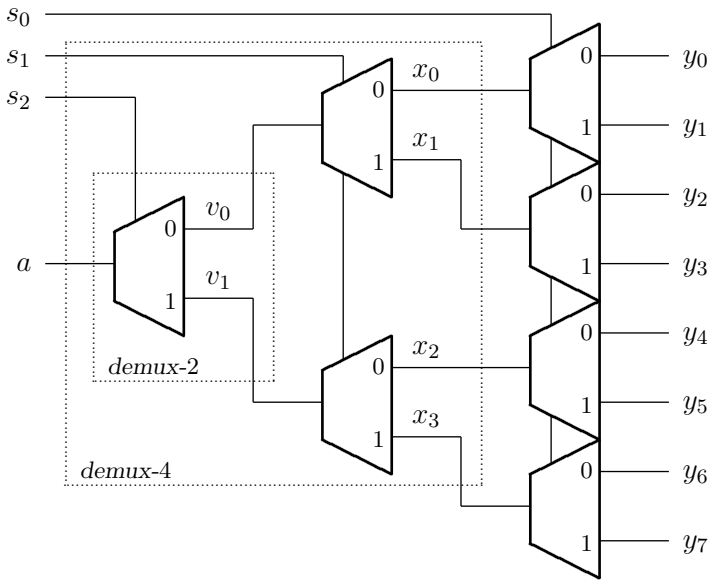


Figura 4.25: Demultiplexador de oito saídas.

As saídas do demultiplexador são determinadas pelo número binário representado pelas entradas de seleção. No comportamento definido na Equação 4.11, a saída y_2 apresenta o valor da entrada a quando $s_1 s_0 = 2$. Assim, o comportamento do demultiplexador de 2^n saídas é aquele especificado pela Equação 4.12.

$$
\begin{array}{l}
a : \mathbb{B} \\
S : \mathbb{B}^n \\
Y : \mathbb{B}^{2^n} \\
\textit{demux-}2^n : \mathbb{B} \mapsto (\Pi_n \mathbb{B}) \mapsto (\Pi_{2^n} \mathbb{B}) \\
\textit{demux-}2^n(a, s_{n-1} \cdots s_0, y_0 \cdots y_{2^n-1}) \;\equiv\; y_i = a \triangleleft (num(S) = i) \triangleright 0
\end{array}
\tag{4.12}
$$

Exemplo 4.17 Considerando que os tempos de propagação das portas lógicas são aqueles indicados na Tabela 4.8, quais são os tempos de propagação do demultiplexador da Figura 4.24, e o da Figura 4.25?

O tempo de propagação do *demux-2* é o de um inversor mais uma porta *and-2*: $T_{\mathrm{dmx}-2} = 2+4 = 6$ns. Para o *demux-8*, o caminho mais longo atravessa um inversor no primeiro nível, mais uma porta *and-2* em cada um dos três níveis da árvore: $T_{\mathrm{dmx}-8} = 2 + 4 + 4 + 4 = 18$ns. Os sinais s_1 e s_0 *não* estão no caminho crítico – se todas as entradas se alteram ao mesmo tempo, então as perturbações causadas pelos sinais s_1 e s_0 estabilizam *antes* que a influência de s_2 se propague até as portas *and-2* do segundo e terceiro níveis. ◁

4.3.4 Aplicação: Circuitos de Memória

Vejamos duas aplicações importantes dos circuitos combinacionais básicos, que são as memórias ROM (*Read-Only Memory*), somente de leitura, e as memórias RAM (*Random Access Memory*), que permitem a leitura e a atualização de conteúdos. Aqui, veremos uma forma simples de organizar estas memórias; estes tópicos são aprofundados nas Seções 5.5 e 5.6.

Como vimos no Capítulo 1, uma memória somente de leitura é uma função cujos domínio e imagem são subconjuntos dos Naturais. Considere uma memória somente de leitura com capacidade para 256 bits; seu domínio é o intervalo $[0, 256)$, e sua imagem são 256 valores escolhidos do conjunto $\mathbb{B}$. Os valores desta função ROM são fixados na construção da memória e não podem ser alterados. Para obter o valor de um dos 256 bits, basta apresentar seu *endereço* à ROM e o valor endereçado é apresentado na saída de dados.

Uma memória RAM, por outro lado, nos permite alterar o conteúdo da função. Considere uma RAM com capacidade para armazenar 256 letras gregas. Se desejamos alterar o conteúdo da posição 120, que é π, para λ, basta apresentarmos o novo par $(120, \lambda)$ à RAM e ativar o sinal de atualização, porque então o novo valor sobre-escreverá o valor anterior. Seja R a função que descreve o conteúdo da RAM num dado instante

$$R = \{(0, \kappa), (1, \omega), (2, \alpha), (3, \sigma) \ldots (120, \pi), \ldots (254, \tau), (255, \theta)\}\,.$$

Para atualizar a posição 120 da RAM, é necessário subtrair ($\setminus$) o par $(120, \pi)$ e adicionar o par $(120, \lambda)$, o que resulta na função R'

$$R' = [R \setminus \{(120, \pi)\}] \cup \{(120, \lambda)\}\,.$$

Os circuitos que estudamos neste texto são fabricados na superfície de um circuito integrado, em dispostos em estruturas retangulares ou quadradas, para melhor aproveitamento da superfície. Uma memória de 256 bits, como a descrita acima, se implementada de forma simplória, resulta numa estrutura estreita e longa – um vetor – e geralmente este formato usa o espaço

bidimensional de maneira ineficiente. Se esta memória for implementada como uma matriz retangular, então o espaço pode ser melhor aproveitado. Para simplificar a utilização das memórias, a interface com o mundo externo deve ser a de um vetor, mesmo que sua implementação seja uma matriz quadrada ou retangular.

Considere um dispositivo de três terminais que é uma chave controlada, similar ao interruptor de luz da sala em que você se encontra. O terminal de controle determina a abertura (apaga-se a luz) ou o fechamento da chave (a luz acende). O símbolo para este dispositivo é mostrado na Figura 4.26. Quando o terminal de controle c está em 0, os terminais a e b estão desconectados (chave aberta), e quando c está em 1, então a chave fecha e os terminais a e b estão conectados.

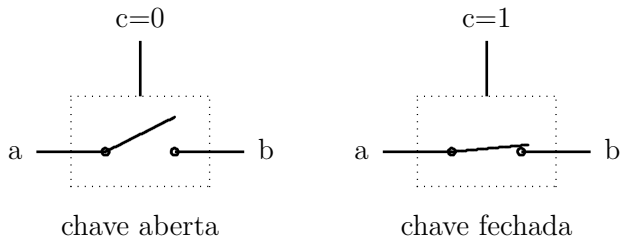


Figura 4.26: Símbolo para a chave digital.

Memória ROM

O conteúdo de uma memória ROM é definido durante a fabricação, e não pode ser alterado posteriormente. Vejamos como organizar uma ROM com 4 bits de altura e um de largura, identificada como uma ROM 4×1. Empregamos um decodificador de quatro saídas para selecionar um dentre os quatro bits, e o bit selecionado é apresentado na saída da memória. O lado esquerdo da Figura 4.27 mostra o símbolo de uma memória ROM 4×1, e o lado direito a sua implementação.

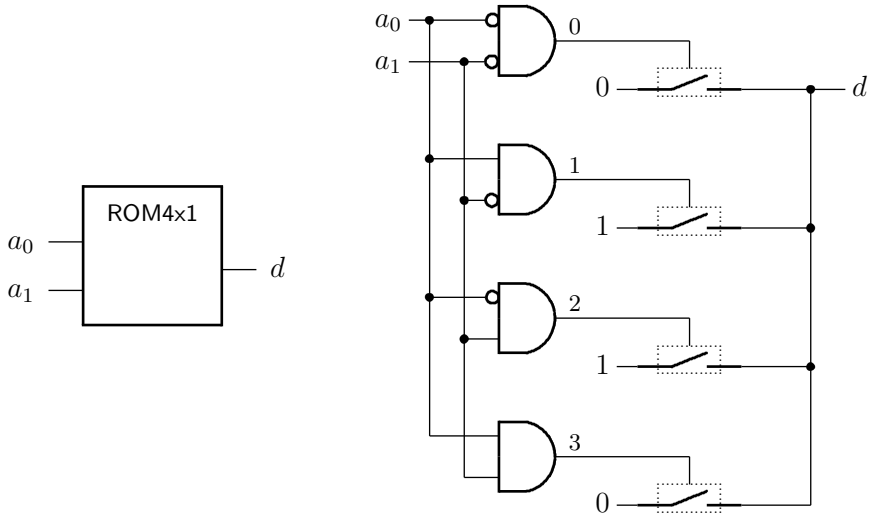


Figura 4.27: Símbolo e implementação de uma ROM 4×1.

As duas *linhas de endereçamento* $\langle a_1, a_0 \rangle$ selecionam um dentre os quatro bits, que é exibido no sinal d. Esta memória ROM tem um comportamento puramente combinacional: decorrido

o tempo de propagação do circuito interno, o bit na saída d é aquele selecionado pelos sinais nas entradas de endereçamento.

O decodificador de duas entradas para quatro saídas escolhe uma das chaves, e o bit correspondente, 0 ou 1, é ligado à saída d. Esta ROM implementa a mesma função que uma porta *xor* de duas entradas. Os bits à esquerda das chaves são programados durante a fabricação e não podem ser alterados.

Para construir uma ROM 8×1, basta acrescentar mais uma linha de endereçamento e duplicar as portas *and* no decodificador. Para construir uma ROM de oito bits, mas com três bits de largura (ROM 4×3) deve-se acrescentar duas colunas adicionais com chaves, como mostra a Figura 4.28. Quando uma das quatro linhas é endereçada por $\langle a_1, a_0 \rangle$, os três bits da linha são apresentados simultaneamente nas saídas $\langle d_2, d_1, d_0 \rangle$.

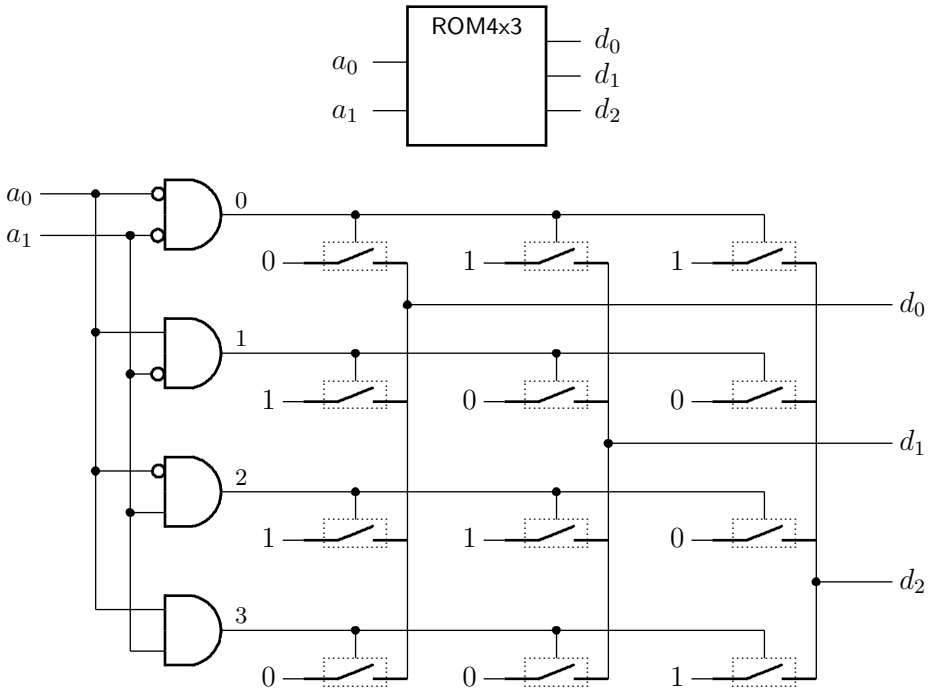


Figura 4.28: Circuito de uma ROM 4×3.

Exemplo 4.18 Quais são as três funções de duas entradas (a_1, a_0) implementadas na ROM da Figura 4.28? As três funções (d_2, d_1, d_0) são indicadas na Tabela 4.9. ◁

Tabela 4.9: Três funções de duas variáveis implementadas com uma ROM 4×3.

índice	a_1	a_0	d_2	d_1	d_0
0	0	0	1	1	0
1	0	1	0	0	1
2	1	0	0	1	1
3	1	1	1	0	0

Normalmente, memórias têm de milhares a milhões de bits, e a organização em uma única linha vertical não é viável, por causa da geometria do componente, que fica longa e estreita. Por razões de construção é necessário construir circuitos mais parecidos com um quadrado, e por isso empregam-se várias linhas verticais. A Figura 4.29 mostra uma memória ROM com 16 bits, organizada como uma matriz de quatro linhas, com quatro bits em cada linha.

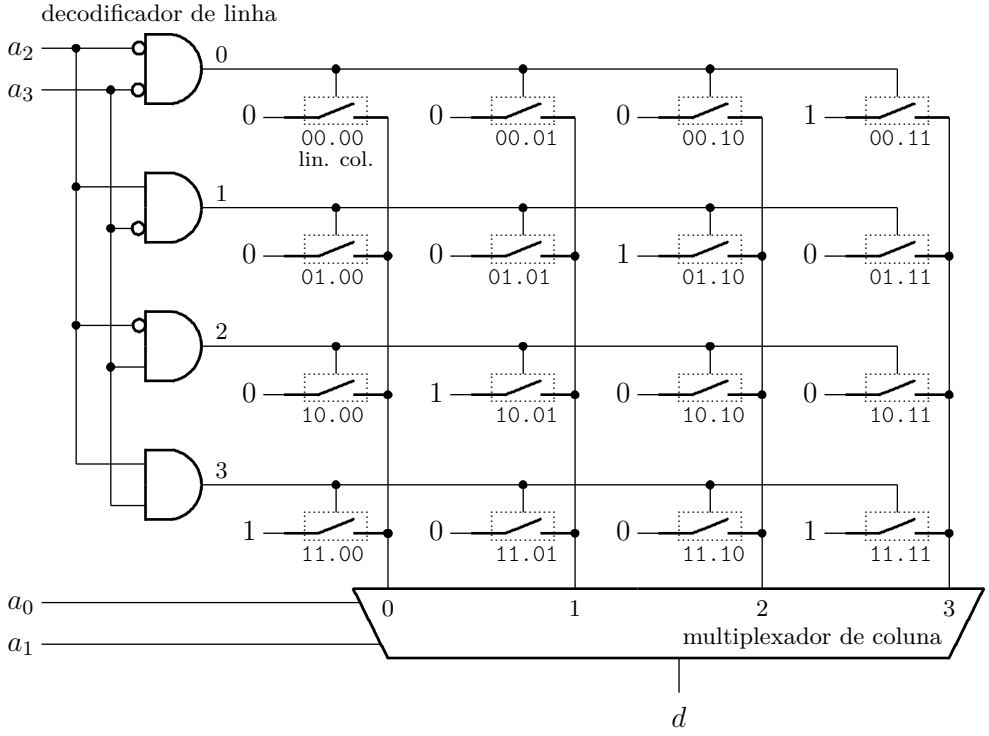


Figura 4.29: Circuito de uma ROM 16×1.

A Figura 4.29 mostra uma memória ROM com um bit de largura e capacidade de 16 bits. Internamente, ela é organizada como uma matriz quadrada, com 4 palavras, de 4 bits cada palavra. O *decodificador de linha* seleciona uma dentre as quatro linhas, ou palavras, enquanto que o *multiplexador de coluna* seleciona um dos quatro bits da palavra. Ao custo adicional de um multiplexador, é possível construir memórias com matrizes de 1024×1024 bits obtendo-se uma geometria razoável.

Há uma diferença importante entre a organização interna da memória – matriz de N linhas por M colunas – e sua interface externa – memória com P palavras de B bits de largura. A organização interna é otimizada para obter-se a maior capacidade possível, e ao mesmo tempo reduzir o tempo de acesso, enquanto que a interface externa, que é essencialmente a largura da porta de dados, depende de restrições comerciais – as larguras usuais são 8, 16 ou 32 bits.

Memória RAM

Uma memória RAM permite que seu conteúdo seja alterado pelo circuito do qual ela faz parte. Para tanto, circuitos de RAM possuem sinais adicionais para controlar a atualização dos conteúdos. Como um mínimo, é necessário um sinal de escrita, tipicamente chamado de *write* (*wr*), que quando ativado, atualiza os conteúdos da memória. A Figura 4.30 mostra o diagrama de blocos de uma memória RAM simples, com quatro linhas de um bit em cada linha.

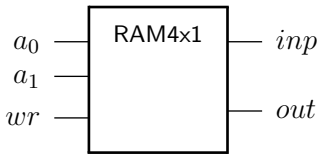


Figura 4.30: Símbolo de uma RAM 4×1.

Ao contrário da ROM, na qual o valor de cada bit é determinado na fabricação, um bit de RAM deve ser um circuito que mantém seu valor inalterado até que ocorra a próxima atualização. Este circuito é mostrado na Figura 4.31. O terminal *wr*, quando em 1, permite que o sinal de entrada (*inp*) seja aplicado à entrada do laço. O sinal *out* permite observar o valor armazenado no bit. Este circuito é chamado de *célula de RAM*, ou célula de memória.

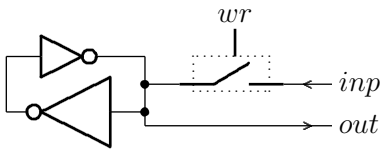


Figura 4.31: Circuito que armazena um bit de memória RAM.

Este circuito *viola* a definição de circuito combinacional por causa do laço de realimentação – há um ciclo que liga a saída do circuito a sua entrada. Suponha que $out = 0$ e $wr = 0$; então saída do inversor maior é 1, a saída do inversor menor é 0, o que mantém a saída do inversor maior em 1.

É justamente o laço de realimentação – a saída de um inversor ligado na entrada do outro – que faz com que este circuito armazene um bit. Enquanto a alimentação dos inversores for mantida, este laço mantém o último valor gravado na célula.

Quando $wr = 1$, o sinal apresentado em *inp* se sobrepõe à saída do inversor pequeno, e altera a entrada do inversor maior, o que altera a saída do inversor menor, assim fechando o laço de realimentação. Os dois inversores são construídos de tal forma que o sinal produzido pelo inversor pequeno é "mais fraco" do que os sinais apresentados na saída do inversor grande e em *inp*. Voltaremos ao tema de "sinais fortes" e "sinais fracos" no Capítulo 5.

Quando uma linha é selecionada por $\langle a_1, a_0 \rangle$, se $wr = 0$ então o valor do bit selecionado é exibido em *out*. Se $wr = 1$ então a célula selecionada por $\langle a_1, a_0 \rangle$ é atualizada com o (novo) valor apresentado em *inp*, e assim permanece até a próxima escrita. O amplificador ligado à *out* isola, eletricamente, o circuito interno da RAM dos circuitos externos.

A Figura 4.32 mostra a interface e o circuito de uma (RAM 4×1). Uma RAM com 4 bytes (RAM 4×8) pode ser facilmente construída ao replicar mais sete vezes a coluna das células e a linha *inp*-*out*. Uma memória com capacidade para 64 bytes pode ser construída com um seletor de 6 para 64 linhas, e esta RAM 64×8 é acessada com seis linhas de endereço.

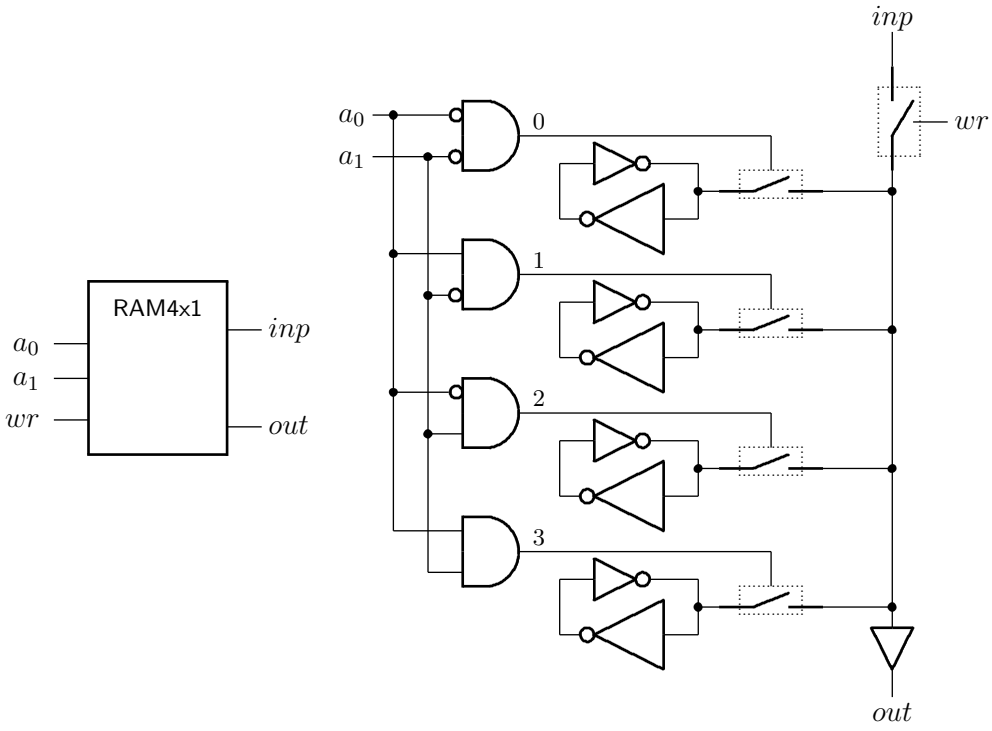


Figura 4.32: Circuito de uma RAM 4×1.

Exemplo 4.19 Vejamos como projetar um decodificador de 6 endereços para 64 linhas. Uma possibilidade é estender os circuitos do Exemplo 4.14 e empregar 64 portas *and* de 6 entradas e seis inversores. Este circuito pode ser considerado uma árvore larga e baixa.

Outra possibilidade é construir uma árvore de altura 6, similar ao demultiplexador da Figura 4.25, com um *decod*-2 ao invés de um *demux*-2 no primeiro nível.

Ainda outra implementação é possível com compomnentes de 4 saídas, como o circuito da Figura 4.33. Esta é uma árvore de altura 3 porque $\log_4 64 = 3$.

Qual destas implementações é *a melhor*? A resposta mais precisa é *depende*. O Exercício 4.6 pede uma comparação dos tempos de propagação das três versões propostas aqui. Outra questão relacionada ao tempo de propagação é a influência do número de entradas conectadas à saída de uma porta lógica, ou o *fan-out* de cada porta lógica no seletor. Este problema é investigado na Seção 5.4. ◁

Exercícios

Ex. 4.1 Prove que $(a \vee b) \wedge \overline{(a \wedge b)} = (a \wedge \overline{b}) \vee (\overline{a} \wedge b)$.

Ex. 4.2 Enuncie a lei de formação de um *mux*-N a partir de *mux*-(N-1).

Ex. 4.3 Enuncie a lei de formação de um *demux*-N a partir de *demux*-(N-1).

Ex. 4.4 Enuncie a lei de formação de um *decod*-N a partir de *decod*-(N-1).

Ex. 4.5 Mostre como implementar um circuito *demux*-2 com 4 portas *nor*.

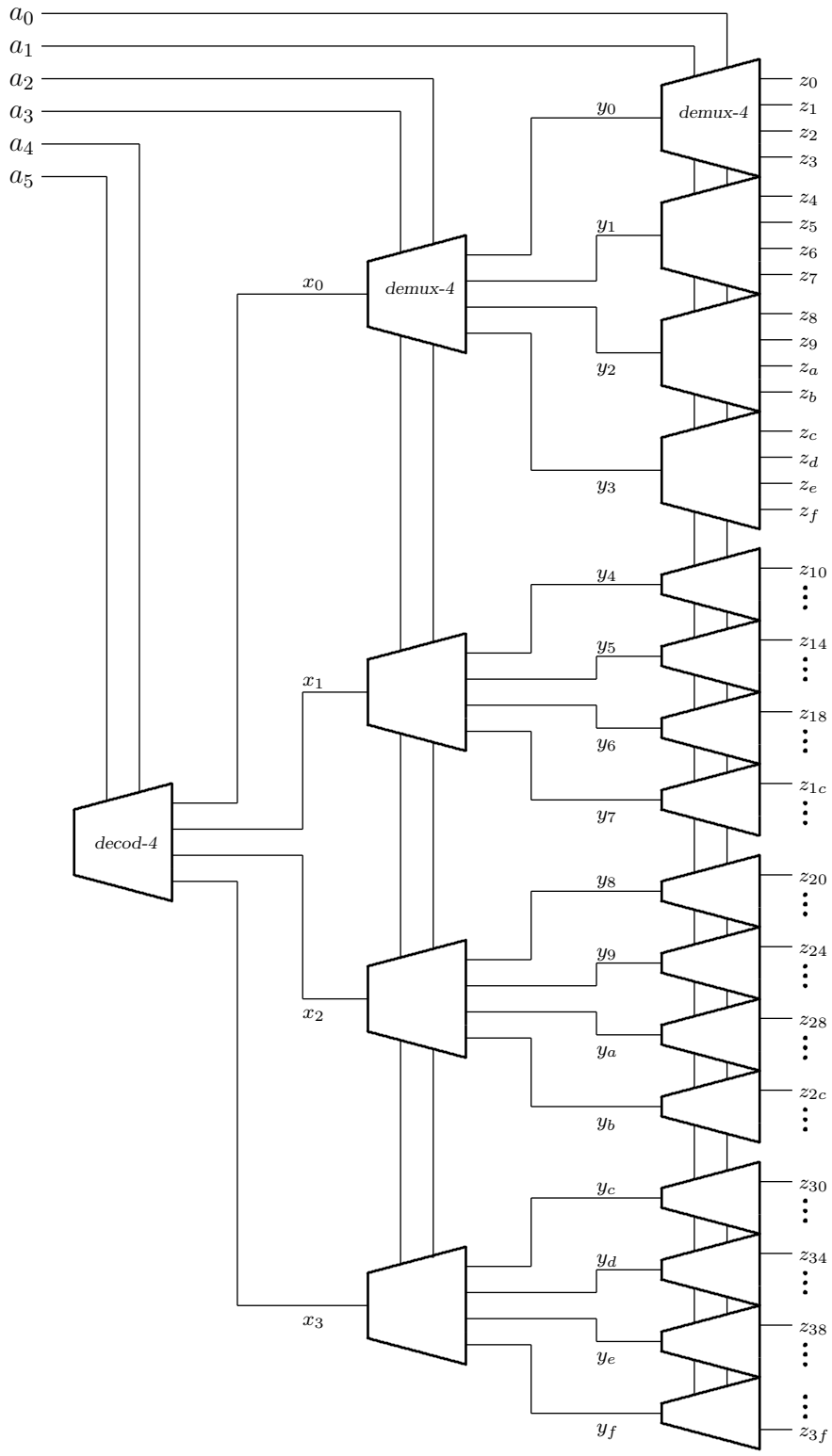


Figura 4.33: Decodificador 6 entradas para 64 saídas.

Ex. 4.6 Calcule o tempo de propagação dos três seletores de 64 saídas propostos no Exemplo 4.19. Empregue os tempos de propagação da Tabela 4.8.

Ex. 4.7 Repita o Ex. 4.6 levando em conta a *carga* ligada em cada porta. A carga é o número de entradas alimentadas pela saída da porta lógica. Empregue os tempos de propagação da Tabela 4.8 e considere que o tempo de propagação de uma porta piora em 10% para cada entrada que ela alimenta além da primeira. Para cada ligação adicional, além da primeira, o tempo de propagação aumenta em 10%. Considere uma taxa composta: se uma porta κ alimenta 4 saídas, então seu tempo de propagação é $T_\kappa \times (1,1)^3$, que cada linha de endereço a_i é gerada por uma porta lógica, e que cada saída z_i alimenta um inversor.

Ex. 4.8 Mostre como um decodificador de 2^n saídas pode ser construído com um decodificador de duas saídas e dois demultiplexadores de 2^{n-1} saídas.

Ex. 4.9 Uma possível implementação para multiplexadores de 2^n entradas consiste em usar um decodificador de n para 2^n, 2^n portas *and* de 2 entradas, e uma porta *or* de 2^n entradas. O sinal de entrada é ligado a todas as portas *and*, cada saída do decodificador é ligada a outra entrada da porta *and*. As saídas de todas as *and* são ligadas às respectivas 2^n entradas da porta *or*. As n entradas de seleção são ligadas às n entradas do decodificador. Desenhe o circuito de multiplexador de oito entradas implementado como descrito aqui.

Ex. 4.10 Prove que um *mux*-2 implementa as funções lógicas $\wedge$, $\vee$, $\neg$ e $\oplus$. Por que isso poderia ser relevante?

Ex. 4.11 Com base na definição do multiplexador, mostre que um *mux*-N pode-se implementar qualquer função lógica de $\log_2 N$ variáveis.

Ex. 4.12 Com base na definição do multiplexador, mostre que com um *mux*-N e inversores, pode-se implementar qualquer função de $\log_2(N+1)$ variáveis.

Ex. 4.13 Projete e implemente um comparador de magnitude ($m = A > B$) para números de dois bits. Escreva a tabela verdade e simplifique a função m com um Mapa de Karnaugh.

Ex. 4.14 Estenda a solução do Ex. 4.13 para números de três bits. *Pista*: aproveite a solução do Ex. 4.13; se os bits a_2 e b_2 são iguais, o que determina se $A > B$? Se os bits a_2 e b_2 são diferentes, o que determina se $A > B$?

Ex. 4.15 Estenda a solução do Ex. 4.14 para números de quatro bits.

Ex. 4.16 Mostre como implementar uma memória ROM com capacidade para 64 palavras de 1 bit.

Ex. 4.17 Mostre como implementar uma memória RAM com capacidade para 64 palavras de 1 bit. Lembre da atualização.

Ex. 4.18 Mostre como implementar uma memória ROM com capacidade para 64 palavras de 4 bits.

Ex. 4.19 Mostre como implementar uma memória RAM com capacidade para 64 palavras de 4 bits. Lembre da atualização.

Índice Remissivo