Lógica combinacional modular e multi-níveis

Última revisão em 8 de junho de 2010.

Da mesma forma que funções em programas computacionais são reutilizadas diversas vezes, subcircuitos em circuitos digitais podem ser reutilizadas várias vezes. Do ponto de vista funcional, em ambos os casos, os aspectos dos módulos que realmente interessam são a sua funcionalidade, as suas entradas e as saídas.

No caso de circuitos digitais, alguns subcircuitos, tais como decodificadores, codificadores, multiplexadores e demultiplexadores, são bastante utilizados em sistemas digitais. Em geral, a realização de circuitos que utiliza módulos tende a ser multi-níveis (mais de dois níveis de portas lógicas). Uma conseqüência imediata disso é que esses circuitos serão mais lentos que os circuitos dois-níveis. No entanto, além das vantagens decorrentes da modularidade (por exemplo, possibilidade de realizar testes por partes, uma melhor visão da estrutura global, etc), há casos em que uma realização multi-níveis de uma função utiliza menos portas lógicas do que uma realização dois-níveis.

Referências:

- Hill, F. J. and Peterson, G. R. (1993). *Computer Aided Logical Design with Emphasis on VLSI.* John Wiley & Sons, fourth edition.
- Nelson, V. P., Nagle, H. T., Carroll, B. D., and Irwin, J. D. (1995). *Digital Logic Circuit Analysis and Design.* Prentice-Hall.

1.1 Alguns exemplos iniciais

Somadores

Um exemplo típico da utilização de módulos são os circuitos somadores de números binários vistos anteriormente.

A soma de dois números binários é realizada de forma similar à soma de números na base 10, ou seja, os números são somados coluna a coluna, da direita (dígito menos significativo) para a esquerda (dígito mais significativo). A cada coluna somada, o excedente (vai-um) é enviado para a próxima coluna. A única diferença entre a soma decimal e binária é que os únicos dígitos permitidos no segundo caso são o 0 e o 1.

A soma dos dígitos binários de uma dada coluna pode ser expressa pela seguinte tabela verdade. As colunas a e b representam, respectivamente, os bits a serem somados, c_{in} representa o vai-um vindo

da coluna anterior, a coluna s representa o bit soma e c_{out} o novo bit vai-um.

a	b	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As funções s e c_{out} podem ser expressas por

$$s(a, b, c_{in}) = (a \oplus b) \oplus c_{in}$$

e

$$c_{out}(a, b, c_{in}) = ab + (a \oplus b)c_{in}$$

Essas funções podem ser realizadas pelo circuito mostrado na figura 1.1. Esse circuito é denominado somador completo de bits.

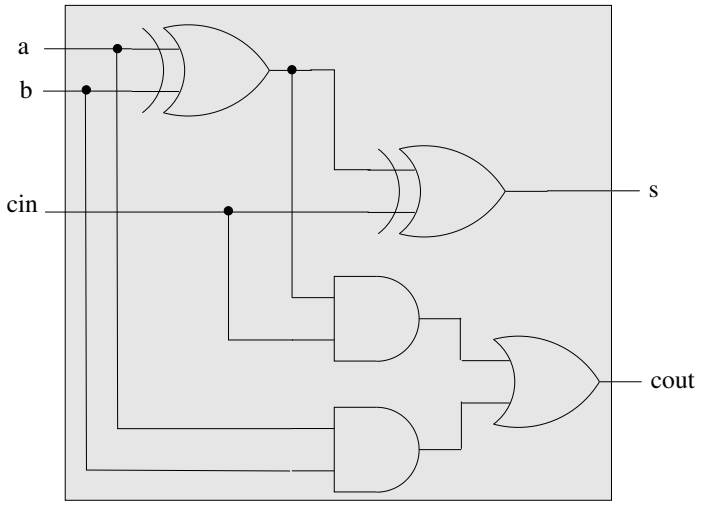

Figura 1.1: Esquema de um somador completo de bits.

Um somador para um número arbitário n de bits pode ser obtido concatenando-se n somadores completos de bits, de forma que a saída c_{out} de um módulo alimente a entrada c_{in} do próximo módulo. A figura 1.2 mostra como seria um somador de 4 bits.

Ao se concatenar dois somadores de 4 bits é possível obter um somador de 8 bits e assim por diante.

Um problema desse tipo de circuito é o tempo de atraso de propagação. Observe-que, o valor do módulo seguinte só estará correto após sua entrada c_{in} ter sido alimentada com o valor da saída c_{out} do módulo anterior. Então, se supormos que o tempo para a propagação do sinal de entrada para a saída em um somador completo de bits é de $8\,ns$ (8 nanosegundos), então o tempo total de atraso de propagação em um somador de n bits será de $8 * n\ ns$.

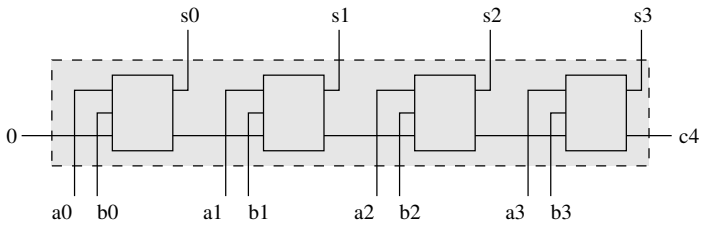

Figura 1.2: Esquema de um somador de 4 bits.

Em circuitos críticos como somadores, pode ser conveniente abrirmos mão da modularidade em troca de redução no tempo de atraso de propagação. Uma forma alternativa para a realização de somadores é apresentada a seguir.

Ao considerarmos um somador de bits, podemos notar que em algumas situações é gerado um vai-um e em outras situações o vai-um é propagado. Mais especificamente,

a	b	c_{in}	Vai-um propagado/gerado
0	1	1	propagado
1	0	1	propagado
1	1	0	gerado
1	1	1	propagado/gerado

Podemos escrever as equações que caracterizam o vai-um gerado (c_g) e o vai-um propagado (c_p, considerando que um vai-um só pode ser propagado quando $c_{in} = 1$), conforme segue:

$$c_g = a\,b$$

e

$$c_p = a + b$$

O novo vai-um c_{out} pode ser expresso em termos de c_g e c_p da seguinte forma:

$$c_{out} = c_g + c_p\,c_{in}$$

Assim, denotando c_{in_i} e c_{out_i} o vai-um entrada e saída, respectivamente, e c_{g_i} e c_{p_i} os vai-uns gerado e propagado do módulo i, temos para $i = 1$

$$c_{out_1} = c_{g_1} + c_{p_1}\,c_{in_1}$$

e para $i = 2$,

$$\begin{aligned} c_{in2} = c_{out_1} & \\ c_{out_2} &= c_{g_2} + c_{p_2}\,c_{in_2} \\ &= c_{g_2} + c_{p_2}\,c_{out_1} \\ &= c_{g_2} + c_{p_2}\,(c_{g_1} + c_{p_1}\,c_{in1}) \\ &= c_{g_2} + c_{p_2}\,c_{g_1} + c_{p_2}\,c_{p_1}\,c_{in_1} \end{aligned}$$

Observe que a última equação acima depende apenas dos bits a serem somados e do vai-um de entrada do primeiro módulo (ou seja, c_{in_1}).

De forma similar, podemos chegar a mesma conclusão com relação a c_{out_i}. Além disso, os bits soma também podem ser expressos todos em função apenas dos bits a serem somados e do vai-um c_{in_1}. Ou seja, significa que o tempo de atraso devido à propagação do vai-um foi eiminado. Como conseqüência, o cálculo não é mais modular e quanto maior o número de bits maior será o número de níveis de portas no circuito. Mesmo assim, um circuito desses será mais rápido que o modular descrito acima.

Comparadores

Comparadores de igualdade de dois números binários podem ser implementados usando-se portas XOR (ou exclusivo). A saída de uma porta XOR é 1 se e somente se os dois bits de entrada tem valores distintos. Para comparar a igualdade de n bits, poderíamos utilizar n portas XOR que fariam a comparação bit-a-bit, e a saída das portas XOR pode alimentar uma porta OU, cuja saída deve ser negada. Desta forma, a saída será 1 se e somente se todas as portas XOR respondem 0.

Comparadores de bits podem ser projetados de forma mais geral, de forma a produzir três saídas: (1) $a > b$, (2) $a = b$, ou (3) $a < b$. Para que um módulo destes seja utilizado para compor um comparador de n bits, é necessário acrescentar entradas ao módulo, de modo que o "resultado preliminar" dos módulos anteriores possa ser propagado (de forma similar ao vai-um no caso do somador).

Sejam $A = a_n \ldots a_1$ e $B = b_n \ldots b_1$ dois números binários de n bits. Seja um módulo com duas entradas a e b que são alimentados pelos bits a serem comparados a_i e b_i, respectivamente, e três entradas $<_{in}$, $=_{in}$ e $>_{in}$ que serão alimentados pelos sinais propagados pelos módulos anteriores. As saídas $<_{out}$, $=_{out}$ e $>_{out}$ dependem das 5 entradas.

A regra para propagar o sinal do bit mais significativo para o do menos significativo é dada por:

$<_{out} = 1$ se e somente se ($<_{in} = 1$ ou ($=_{in} = 1$ e $a_i < b_i$))

$>_{out} = 1$ se e somente se ($>_{in} = 1$ ou ($=_{in} = 1$ e $a_i > b_i$))

$=_{out} = 1$ se e somente se ($=_{in} = 1$ e $a_i = b_i$)

No primeiro módulo deve-se fazer $=_{in} = 1$.

Observe que da mesma forma do somador, neste caso também ocorre o atraso de propagação.

1.2 Decodificadores

Decodificadores são circuitos combinacionais com n entradas e 2^n saídas. Para cada uma das 2^n possíveis combinações de valores para a entrada, apenas uma saída toma valor 1. A figura 1.3 mostra um esquema genérico de um decodificador $n : 2^n$. As entradas $x_{n-1} \ldots x_1\, x_0$ podem ser interpretadas como um número binário entre 0 e $2^n - 1$ e tem-se $z_i = 1 \iff \sum_{k=0}^{n-1} 2^k\, x_k = i$.

Exemplo: Seja o decodificador $2 : 4$ e suponha que as entradas são $B\,A$ (i.e., $x_0 = A$ e $x_1 = B$). Um circuito correpondente ao decodificador é ilustrado na figura 1.4. No caso temos $z_0 = \overline{B}\,\overline{A}$, $z_1 = \overline{B}\,A$, $z_2 = B\,\overline{A}$ e $z_3 = BA$.

Conceitualmente, o circuito acima poderia ser extendido para realizar decodificadores $n : 2^n$, para um valor de n arbitrariamente grande. No entanto, na prática existem limitações tecnológicas (físicas) conhecidas como *fan-in* (número máximo de entradas possíveis em uma porta lógica) que restringem este valor n. Para valores grandes de n, decodificadores são realizados por circuitos multi-níveis, conforme veremos mais adiante.

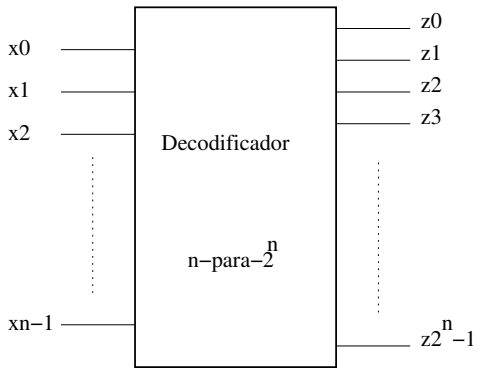

Figura 1.3: Esquema de um decodificador $n : 2^n$.

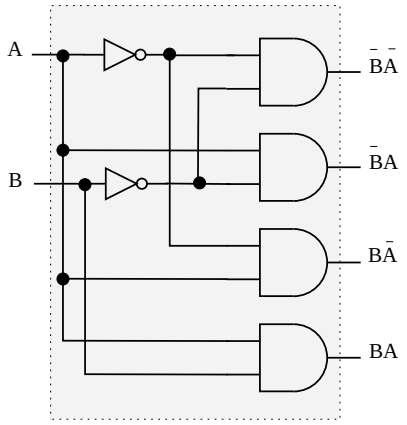

Figura 1.4: Circuito de um decodificador $2 : 4$.

1.2.1 Memórias ROM

Um exemplo de uso de decodificadores são as memórias do tipo ROM (*Read-Only Memory*). Suponha por exemplo uma memória com 4 posições. O endereço destas posições pode ser codificado em dois bits $x_1\, x_0$. Decodificadores podem ser utilizados para se endereçar uma certa posição na memória, gerando-se um sinal na linha de saída que corresponde à posição a ser endereçada.

A cada endereço $(x_1\, x_0)$ fonecido como entrada do decodificador, apenas uma saída do decodificador ficará ativa. A palavra na posição de memória endereçada (isto é, o dado armazenado na linha de saída ativada) será transmitida para a saída $(z_3\, z_2\, z_1\, z_0)$ (note que o esquema da figura está simplificado; a rigor, cada porta OU possui exatamente 4 entradas que poderão ou não estar conectadas a cada uma das linhas de saída do decodificador).

Generalizando o esquema acima, para 2^n posições de memória com palavras de m bits, precisaríamos de um decodificador $n : 2^n$ e um plano OU com m portas (um para cada bit da palavra).

Memórias ROM possuem uma estrutura semelhante aos PLAs (i.e., um plano de portas E e um plano de portas OU). As diferenças em relação a um PLA são o fato de que ROMs possuem necessariamente 2^n portas E (todos os produtos são canônicos) e apenas o plano OU é programável. Se o plano OU tem conexões fixas, trata-se de uma ROM. Se o plano OU pode ser programado, então trata-se de uma PROM (*Programmable ROM*), e se o plano OU pode ser reprogramado trata-se de uma EPROM

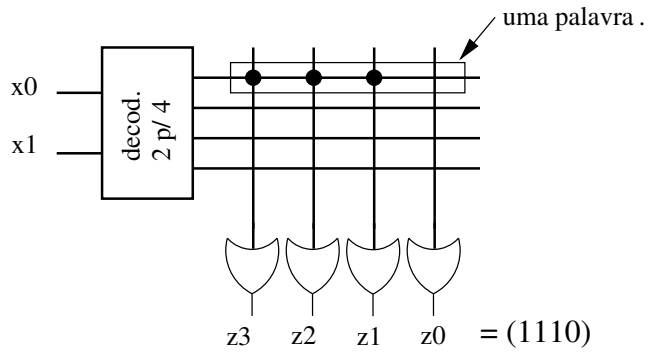

Figura 1.5: Esquema de uma memória ROM com 4 posições e palavras de 4 bits.

(*Erasable Programmable ROM*).

1.2.2 Realização de funções com decodificadores

Uma vez que um decodificador $n : 2^n$ realiza todos os produtos canônicos de n variáveis, qualquer função com n variáveis pode ser realizada com um decodificador $n : 2^n$ e uma porta OU (com um número de entradas maior ou igual ao número de 1s da função) ou uma porta NÃO-OU (com um número de entradas maior ou igual ao número de 0s da função).

O custo da realização de uma função com decodifcadores é, em termos de portas lógicas, (muito provavelmente) maior que o da realização SOP minimal. No entanto, a simplicidade de projeto torna-o atraente. Além disso, quando múltiplas funções precisam ser realizadas, menor tende a ser a diferença dos custos entre a realização SOP minimal e a realização com decodificadores.

Exemplo: A função $f(a,b,c) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$ pode ser realizada usando decodificadores conforme ilustrado na figura 1.6. No caso da realização com porta NÃO-OU, observe que $f(a,b,c) = \prod M(2,3,5) = \overline{\overline{M_2 \cdot M_3 \cdot M_5}} = \overline{\overline{M_2} + \overline{M_3} + \overline{M_5}} = \overline{m_2 + m_3 + m_5}$.

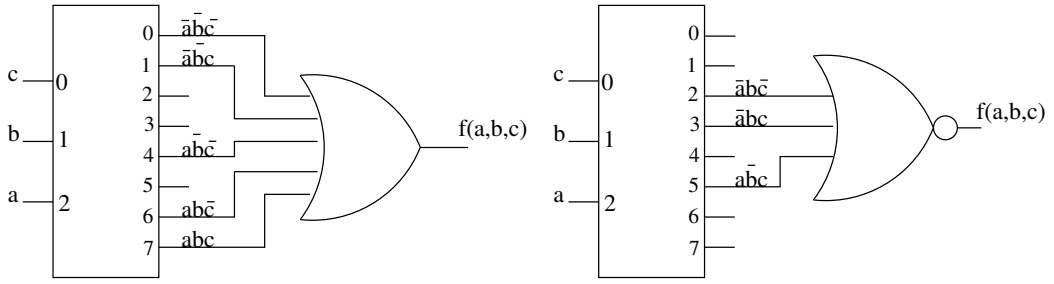

Figura 1.6: Realização de $f(a,b,c) = \sum m(0,1,4,6,7)$ com decodificador $3 : 8$ e uma porta OU (esquerda) ou uma porta NÃO-OU (direita).

1.2.3 Realização multi-níveis de decodificadores*

Vimos que decodificadores possuem n entradas e 2^n saídas e que sua realização trivial utiliza 2^n portas E, com n entradas cada. Para contornar o problema de *fan-in* (número máximo de entradas

possíveis em uma porta lógica), decodificadores com grande número de entradas podem ser realizados por circuitos com múltiplos níveis. A figura 1.7 mostra como pode ser realizado um decodificador $12 : 2^{12}$ em três níveis.

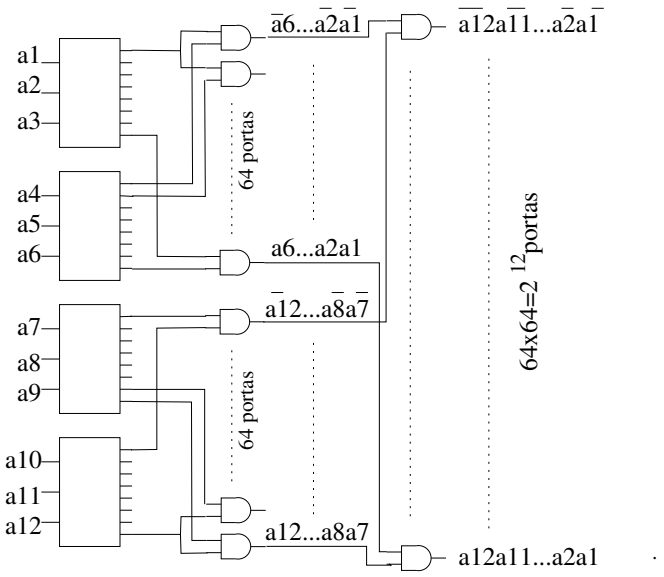

Figura 1.7: Realização três-níveis de um decodificador $12 : 2^{12}$.

No primeiro nível são usados 4 decodificadores $3 : 8$. No segundo nível, 64 portas E de duas entradas são usadas para combinar cada uma das 8 saídas do primeiro decodificador com cada uma das 8 saídas do segundo decodificador. A mesma coisa para as saídas do terceiro com as do quarto decodificador. Cada uma das saídas das primeiras 64 portas E do segundo nível são combinadas com cada uma das saídas das últimas 64 portas E no mesmo nível, resultando em um total de $64 \times 64 = 2^{12}$ portas E no terceiro nível. As saídas dessas 2^{12} portas E correspondem aos produtos canônicos de 12 variáveis.

A solução acima utiliza portas E com três entradas no primeiro nível e portas E com duas entradas nos demais níveis. Se o circuito fosse realizado em apenas um nível, as portas E teriam 12 entradas.

Em uma outra possível realização, poderíamos substituir as 128 portas E de duas entradas no segundo nível acima por 2^{12} portas E de quatro entradas e eliminar as portas do terceiro nível. Isto aparentemente reduziria o número total de portas, mas uma vez que 2^{12} domina de longe 128 e uma vez que as portas agora teriam quatro entradas em vez de duas, não se pode dizer que há economia no custo total.

Um outro problema devido às limitações tecnológicas é o conhecido por *fan-out* (número máximo de portas que podem ser alimentadas por uma saída de uma porta lógica). No caso da realização três-níveis do decodificador $12 : 2^{12}$ visto acima, as saídas das portas no segundo nível alimentam 64 portas no terceiro nível.

Para contornar o *fan-out*, uma possível solução são as realizações em estruturas de árvore. A figura 1.8 mostra a realização de um decodificador $3 : 8$ em uma estrutura de árvore. Em vez de termos todas as variáveis alimentando portas no primeiro nível, temos variáveis que alimentam portas nos outros

níveis. Usando este esquema, pode-se reduzir o número de portas no próximo nível que precisam ser alimentadas pela saída de uma porta no nível anterior. Mesmo assim, o problema de *fan-out* não é totalmente eliminado pois as variáveis introduzidas nos níveis posteriores do circuito precisam alimentar muitas portas. No entanto, é mais fácil controlar o sinal de algumas poucas entradas (variáveis) para que eles sejam capazes de alimentar um maior número de portas do que fazer o mesmo com as saídas das portas lógicas. Esta solução possui um maior número de níveis e um maior número de portas lógicas do que o esquema mostrado na figura 1.7, mas para decodificadores de muitas entradas pode ser a única solução.

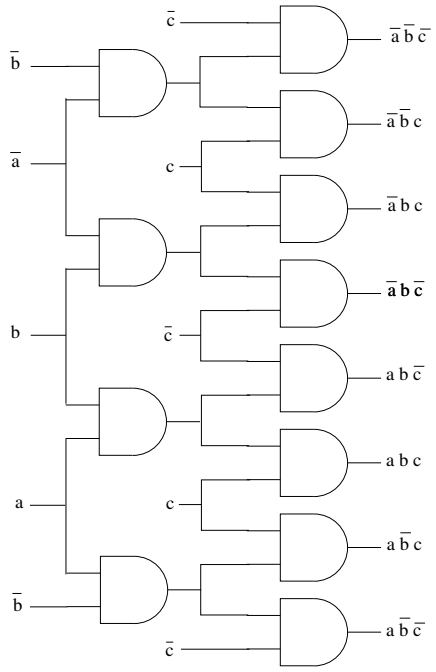

Figura 1.8: Decodificador em estrutura de árvore.

Na prática, as realizações de decodificadores para um número grande de entradas é baseada em uma combinação das estruturas da figura 1.7 e da figura 1.8.

1.3 Codificadores

Codificadores são circuitos combinacionais que são o inverso de decodificadores. Um codificador de n entradas deve possuir s saídas satisfazendo

$$2^s \geq n \quad \text{ou } s \geq \log_2 n$$

Usualmente deve-se ter apenas uma entrada ativa e a saída será o código binário correspondente à entrada. Isto é, se a i-ésima entrada estiver ativa, a saída será o código binário de i. A figura 1.9 mostra o esquema de um codificador de n entradas.

Embora usualmente os codificadores sejam definidos como um circuito no qual apenas uma entrada encontra-se ativa, é possível termos codificadores com propósitos específicos que, por exemplo, para certos tipos de combinação de entradas gera um dado código de saída e para outras combinações de entradas gera outro código de saída.

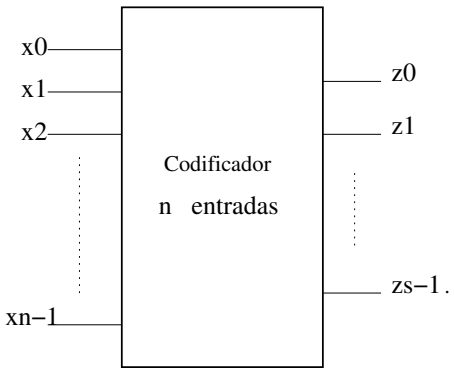

Figura 1.9: Esquema de um codificador.

1.3.1 Teclado

Decodificadores e codificadores podem ser usados, por exemplo, em teclados. Suponha por exemplo que um teclado simplificado possui 70 teclas. Em vez de se ter 70 fios conectando cada uma das teclas a um gerador de código ASCII, podemos ter um esquema como o ilustrado na figura 1.10.

O cruzamento das saídas do decodificador 3-8 com as entradas do codificador 16×4 corresponde a uma tecla. Quando houver sinal na linha de saída correspondente à tecla pressionada, o sinal entrará no codificador. A saída do codificador indica em qual das 14 colunas está a tecla pressionada, enquanto as linhas que ligam a saída do decodificador ao gerador de código ASCII indicam em qual das 5 linhas a tecla está. O contador à esquerda da figura alimenta as entradas do decodificador (varia ciclicamente de 0 a 7), tendo o efeito de gerar saída em uma das 5 linhas, ciclicamente. Obviamente há várias questões que precisam ser consideradas tais como garantir que o contador realize um ciclo completo durante o período de tempo em que uma tecla está pressionada (para "não perder" a tecla pressionada), mas também não mais que um ciclo (para não produzir o efeito de duas pressões), ou então tratar as combinações de teclas que usualmente são pressionadas simultaneamente (como SHIFT+outra, CTRL+outra). Essas questões não são consideradas no modelo simplificado do teclado.

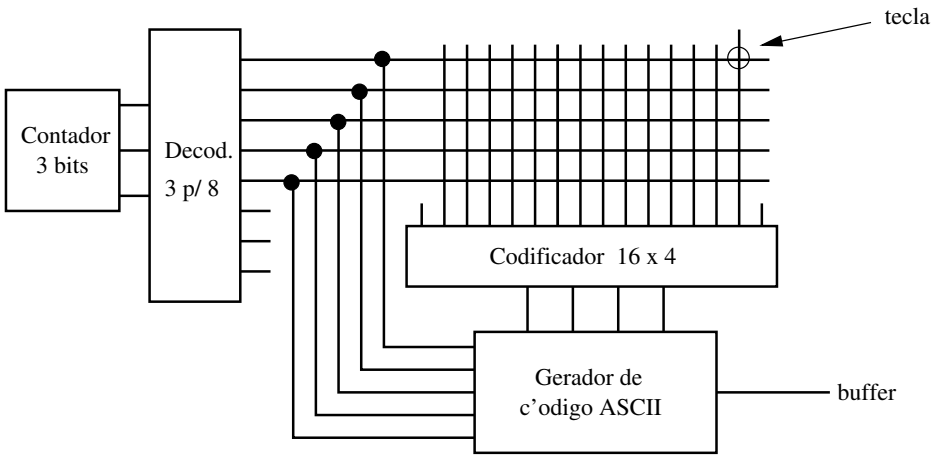

Figura 1.10: Esquema de um teclado. O decodificador identifica a linha e o codificador a linha da tecla pressionada.

1.4 Multiplexadores (Seletores de dados)

Multiplexadores são circuitos combinacionais com n entradas e uma saída. Apenas uma entrada é selecionada para ser enviada à saída. A entrada selecionada é justamente aquela que é especificada pelos seletores, que consistem de k sinais, satisfazendo $k \geq \log_2 n$. A figura 1.11 mostra um multiplexador 4×1, isto é, um multiplexador de 4 entradas.

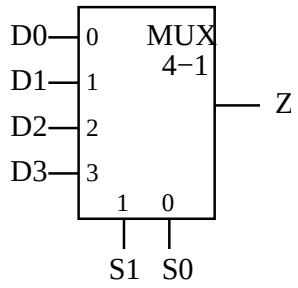

Figura 1.11: Multiplexador de 4 entradas.

Se os seletores forem $S_1S_0 = 00$, então teremos $Z = D_0$; se forem $S_1S_0 = 01$, então teremos $Z = D_1$; $S_1S_0 = 10$, então teremos $Z = D_2$; $S_1S_0 = 11$, então teremos $Z = D_3$.

Podemos observar que Z é uma função das variáveis S_0, S_1 e D_0, D_1, D_2, D_4. Assim, podemos escrever Z como

$$Z(D_0, D_1, D_2, D_4, S_1, S_0) = D_0\,\overline{S}_0\,\overline{S}_1 + D_1\,\overline{S}_0\,S_1 + D_2\,S_0\,\overline{S}_1 + D_3\,S_0\,S_1$$

e portanto multiplexadores podem ser realizados com circuitos dois-níveis, consistindo de n portas E no primeiro nível e uma porta OU no segundo nível, conforme mostrado na figura 1.12. Note que para cada possível combinação de valores de S_1S_0, apenas um dos produtos toma valor 1 na equação acima.

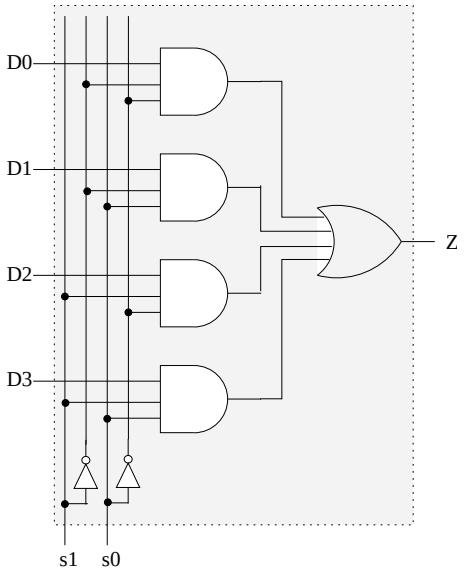

Figura 1.12: Circuito de um multiplexador de 4 entradas.

1.4.1 Realização de funções com MUX

Funções podem ser realizadas utilizando-se MUX. Para tanto, deve-se escolher as variáveis que funcionarão como seletores e, em seguida, as n entradas que poderão ou não depender das demais variáveis.

Exemplo: Realizar a função $f(a,b,c) = \overline{a}\,\overline{b} + a\,c$ usando um MUX 4×1, com as variáveis a e b como seletores.

Uma possível forma para fazer isso é expandir a expressão da função de forma que os literais correspondentes às variáveis a e b apareçam em todos os produtos da expressão resultante. No caso da função dada, fazemos

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

Assim, no MUX 4×1 basta fazermos $s_1 = a$, $s_0 = b$, $D_0 = 1$, $D_1 = 0$, $D_2 = c$ e $D_3 = c$.

Exercício 1: Escreva a realização da função $f(a,b,c) = \overline{a}\,\overline{b} + a\,c$ usando um MUX 8×1, com as variáveis a, b e c como seletores.

Exercício 2: Escreva a realização da função $f(a,b,c,d) = \sum m(0,1,3,6,7,8,11,12,14)$ usando um MUX 8×1, com as variáveis a, b e c como seletores.

Exercício 3: Escreva a realização da função $f(a,b,c,d) = \sum m(0,1,3,6,7,8,11,12,14)$ usando um MUX 4×1, com as variáveis a e b (neste caso, as entradas possivelmente dependerão das variáveis c e d e serão necessárias portas adicionais para a realização de f).

1.4.2 Realização de MUX como composição de MUX

Um MUX pode ser realizado como composição de MUXes com um menor número de entradas. Por exemplo, um MUX 4×1 pode ser realizado através de 3 MUX 2×1, conforme ilustrado na figura 1.13.

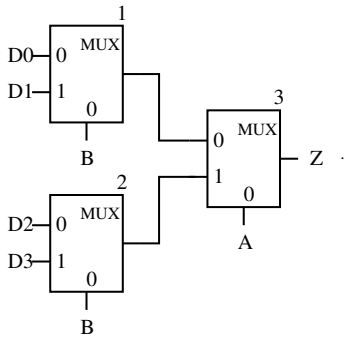

Figura 1.13: Realização de um MUX 4×1 como composição de três MUX 2×1.

Note que se $AB = 00$, então como $B = 0$ a saída do MUX 1 é D_0 e a do MUX 2 é D_2 e, como $A = 0$, a saída do MUX 3 é D_0. Se $AB = 01$ então as saídas são, respectivamente, D_1, D_3 e D_1. Se $AB = 10$, então as saídas são, respectivamente, D_0, D_2 e D_2. Finalmente, se $AB = 11$, então as saídas são,

respectivamente, D_1, D_2 e D_3. Em todos os casos, a terceira saída é a mesma de um MUX 4×1 com AB como entrada para seletores.

Pergunta: E se trocarmos as entradas para os seletores na figura acima? Se colocarmos A no seletor dos MUX 1 e 2 e B na do MUX 3, ainda é possível realizar um MUX 4×1 com AB como entrada para seletores ?

1.4.3 Realização multi-níveis de funções com MUX

Uma função pode ser realizada com múltiplos níveis de multiplexadores. Para cada nível deve-se definir as variáveis que alimentarão os seletores. Em função disso fica definida as variáveis que alimentarão as entradas dos multiplexadores no primeiro nível.

Considere a função $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$. Se for utilizado um MUX 8×1, então serão necessários três variáveis para alimentar os seletores. A quarta variável pode ser diretamente alimentada nas entradas, conforme mostrado na figura 1.14.

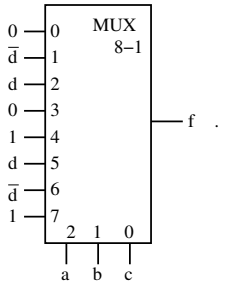

Figura 1.14: Realização de $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$ com um MUX 8×1.

Se pensarmos em utilizar MUX 4×1 e MUX 2×1 na realização de f, duas possíveis soluções, mostradas na figura 1.15, são:

1. dois MUX 4×1 no primeiro nível, alimentando um MUX 2×1 no segundo nível
2. 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

Estas estruturas podem ser obtidas a partir da análise dos mintermos arranjados em forma tabular, conforme mostrado a seguir. A tabela da esquerda considera o uso das entradas b e c como seletores dos MUXes 4×1 no primeiro nível e o uso da varíável a como seletor do MUX 2×1 do segundo nível. A tabela da direita faz o análogo para a implementação com 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

$abcd$	a	bc	d	input
0010	0	01	0	$\overline{d}$
0101	0	10	1	d
1000	1	00	0	$\overline{d}+d=1$
1001	1	00	1	
1100	1	10	0	$\overline{d}$
1110	1	11	0	$\overline{d}+d=1$
1111	1	11	1	

$abcd$	ab	c	d	input
0010	00	1	0	$\overline{d}$
0101	01	0	1	d
1000	10	0	0	$\overline{d}+d=1$
1001	10	0	1	
1100	11	0	0	$\overline{d}$
1110	11	1	0	$\overline{d}+d=1$
1111	11	1	1	

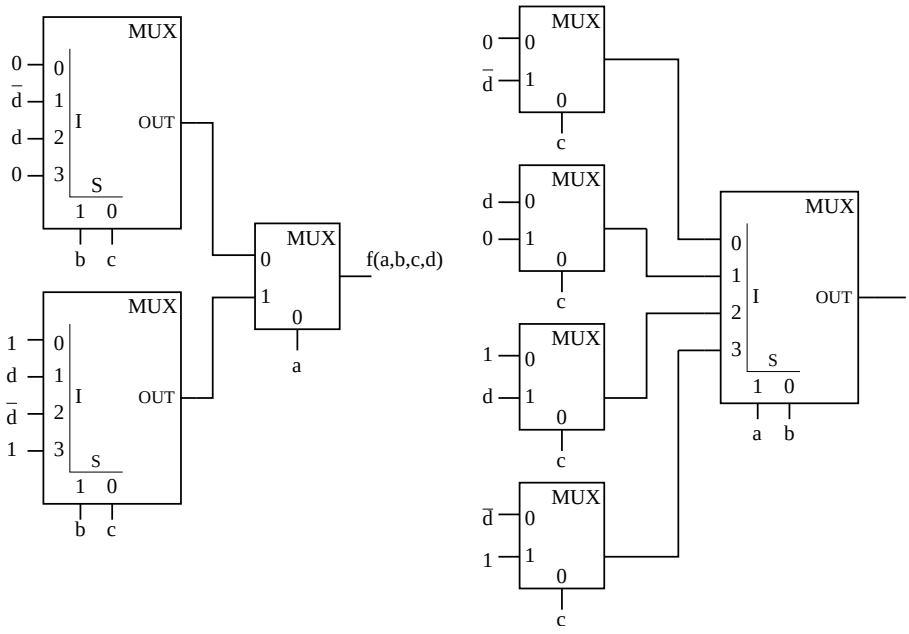

Figura 1.15: Realização de $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$ com (a) dois MUX 4×1 no primeiro nível e um MUX 2×1 no segundo nível e (b) 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

Outra abordagem para determinar a estrutura hierárquica dos MUXes na realizações de funções com múltiplos níveis de MUXes é baseada na aplicação sucessiva da expansão de Shannon. Dependendo da seqüência de variáveis em torno das quais a expansão é aplicada, pode-se chegar a diferentes estruturas.

O teorema de **Expansão de Shannon** afirma que qualquer função f de n variáveis pode ser escrita em termos de funções de $n-1$ variáveis da seguinte forma:

$$f(x_1,\ldots,x_k,\ldots,x_n) = \overline{x}_i f(x_1,\ldots,0,\ldots,x_n) + x_i f(x_1,\ldots,1,\ldots,x_n)$$

As funções $f(x_1,\ldots,0,\ldots,x_n)$ e $f(x_1,\ldots,1,\ldots,x_n)$ são funções de $n-1$ variáveis. O teorema pode ser aplicado recursivamente nessas duas funções.

Exemplo: Consideremos novamente a função $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$. Sua realização com um MUX 8×1 está mostrada na figura 1.14 acima. Vamos mostrar agora que aplicando-se sucessivamente a expansão de Shannon é possível obter diferentes estruturas de realizações de f usando MUX.

$$\begin{aligned}
f &= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{b}\,\overline{c}\,\overline{d} + a\,\overline{b}\,\overline{c}\,d + a\,\overline{b}\,c\,d + a\,b\,\overline{c}\,\overline{d} + a\,b\,c\,\overline{d} + a\,b\,c\,d \\
&= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{b}\,\overline{c}\,\overline{d} + a\,b\,\overline{c}\,\overline{d} + a\,\overline{b}\,\overline{c}\,d + a\,\overline{b}\,c\,d + a\,b\,c\,\overline{d} + a\,b\,c\,d \quad \text{(rearranjo)} \\
&= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{c}\,\overline{d} + a\,\overline{b}\,d + a\,b\,c \quad \text{(algumas simplificações)} \\
&= \overline{a}\,(\overline{b}\,c\,\overline{d} + b\,\overline{c}\,d) + a(\overline{c}\,\overline{d} + \overline{b}\,d + b\,c) \quad \text{(expansão em torno de } a) \\
&= \overline{a}\,(\overline{b}(c\,\overline{d}) + b(\overline{c}\,d)) + a\,(\overline{b}(\overline{c}\,\overline{d} + d) + b\,(\overline{c}\,\overline{d} + c)) \quad \text{(expansão em torno de } b) \\
&= \overline{a}\,\overline{b}(c\,\overline{d}) + \overline{a}\,b(\overline{c}\,d)) + a\,\overline{b}(\overline{c}\,\overline{d} + d) + a\,b\,(\overline{c}\,\overline{d} + c) \quad \text{(distribuição com respeito a } a) \\
&= \overline{a}\,\overline{b}(\overline{c}(0) + c\,(\overline{d})) + \overline{a}\,b(\overline{c}\,(d) + c(0)) + a\,\overline{b}(\overline{c}\,(1) + c(d)) + a\,b\,(\overline{c}\,(\overline{d}) + c(1)) \quad \text{(expansão em torno de } c)
\end{aligned}$$

A última expressão acima pode ser realizada com 4 MUX 2×1 no primeiro nível, com c como entrada para os seletores, mais um MUX 4×1 no segundo nível, com a e b como entrada para os seletores.

Nas equações acima, logo após a expansão em torno de b, poderíamos ter prosseguido da seguinte forma:

$$\begin{aligned} f &= \overline{a}\,(\overline{b}(c\,\overline{d}) + b(\overline{c}\,d)) + a\,(\overline{b}(\overline{c}\,\overline{d} + d) + b\,(\overline{c}\,\overline{d} + c)) \quad \text{(expansão em torno de } b\text{)} \\ &= \overline{a}\,(\overline{b}\,c\,(\overline{d}) + \overline{b}\,\overline{c}(0) + b\,\overline{c}\,(d) + b\,c\,(0)) + a\,(\overline{b}\overline{c}\,(1) + \overline{b}\,c(d) + b\,\overline{c}\,(\overline{d}) + b\,c\,(1))) \end{aligned}$$

Esta última expressão pode ser realizada com 2 MUX 4×1 no primeiro nível, com b e c como entrada para os seletores, mais um MUX 2×1 no segundo nível, com a como entrada para os seletores.

1.5 Demultiplexadores (Distribuidores de dados)

Demultiplexadores são circuitos combinacionais inversos aos multiplexadores, isto é, possuem apenas uma entrada que é transmitida a apenas uma das n saídas. A saída a ser escolhida é selecionada pelos valores dos seletores. Se o demultiplexador possui n saídas, então são necessários k seletores, com $2^k \geq n$.

Pergunta: Para que serve um demultiplexador? Suponha que há dois sistemas A e B e que A possui n saídas que geram ciclicamente sinais que devem ser transmitidos para os respectivos n receptores de B. A transmissão seria direta se houvesse um canal de comunicação entre cada saída de A para a respectiva entrada de B. Se houver apenas um canal de transmissão entre A e B, pode-se utilizar um multiplexador na saída de A e um demultiplexador na entrada de B. Os seletores devem ser ajustados para que, no multiplexador, seja selecionado o sinal que se deseja transmitir a B e, no demultiplexador, seja selecionada a saída conectada à entrada de B para o qual se deseja direcionar o sinal transmitido.

1.6 Lógica combinacional multi-níveis*

Existem funções cuja realização por circuitos com mais de dois níveis é natural e simples enquanto que sua realização por circuitos de dois níveis é proibitivamente ineficiente ou caro.

Um desses exemplos é o circuito verificador de paridade. Suponha que em um sistema de transmissão os dados são codificados em 7 bits. O oitavo bit é utilizado para guardar a informação sobre a paridade desses 7 bits. Se o número de bits 1 no dado é impar, o oitavo bit é 1; se é par, o oitavo bit é 0. Assim, a cada grupo de oito bits transmitidos deve-se ter necessariamente um número par de bits 1. Após a transmissão dos dados, verificar se a paridade a cada 8 bits é par é um teste simples que pode detectar erros (não todos) na transmissão. De fato, pode-se mostrar através de uns cálculos que, por exemplo, quando a probabilidade de ocorrer erros na linha de transmissão é de 1 em cada 10^4 bits transmitidos, as chances de erro diminuem de 7×10^{-4} para 28×10^{-8} caso seja feita a verificação de paridade.

Como poderia ser realizado o circuito para verificar a paridade? Suponha inicialmente uma situação com 4 bits (3 de dados e um de paridade). Então, o seguinte circuito produz saída 1 se, e somente se, se a paridade é impar.

Isto pode ser verificado analisando-se a seguinte tabela.

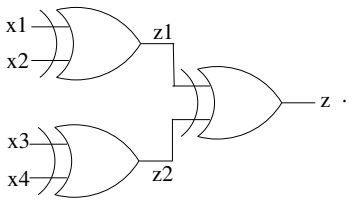

Figura 1.16: Circuito para verificação de paridade para uma entrada de 4 bits.

# de x_is iguais a 1	# de z_is iguais a 1	z
0	0	0
1	1	1
2	0 ou 2	0
3	1	1
4	0	0

Note que o número de saídas $z_i = 1$ é impar se, e somente se, o número de entradas $x_i = 1$ é também ímpar. Se o número de $z_i = 1$ é impar, então $z = 1$. Assim, se a paridade é par, temos $z = 0$ e se é ímpar temos $z = 1$.

O circuito acima pode ser extendido para verificar a paridade de 8 bits, simplesmente conectando-se a saída de dois circuitos daqueles a uma porta XOR, conforme mostrado na figura 1.17. Para realizar tal circuito são necessários 7 portas XOR ou então 21 portas NÃO-E e 14 inversores.

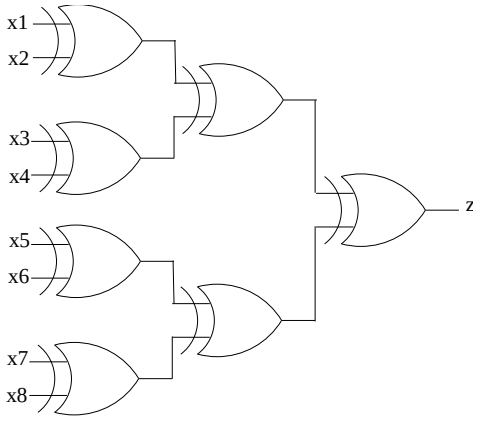

Figura 1.17: Circuito para verificação de paridade para entradas de 8 bits.

Se considerarmos a forma SOP, é fácil constatarmos que não é possível fazer nenhuma minimização a partir da forma SOP canônica. Portanto, no caso de 4 variáveis, como são 7 mintermos correspondentes a entradas de paridade ímpar, seriam necessárias 7 portas E com 4 entradas e uma porta OU com 7 entradas (se pensarmos somente em portas com duas entradas, claramente a forma SOP é muito pior que a apresentada acima).

De uma forma geral, podemos dizer que a lógica multi-níveis é mais flexível que a lógica 2-níveis, isto é, circuitos multi-níveis oferecem maiores possibilidades para reduzir a quantidade de portas lógicas

necessárias. No entanto, o problema de otimização associado à lógica multi-níveis pode ser muito mais complexo. Não existem técnicas para, dada uma função f qualquer, encontrar a realização ótima multi-níveis de f, para um número de níveis fixo qualquer (exceto para 2 níveis).

Algumas das abordagens existentes para minimização lógica multi-níveis resumem-se à composição seqüencial de circuitos de subfunções (módulos) mais simples. Este assunto não será tratado em detalhes neste curso. A seguir serão apresentadas duas abordagens utilizadas para projeto de circuitos multi-níveis.

1.6.1 Decomposição funcional e estrutural de funções

Não é objetivo deste curso aprofundar-se nestes tópicos. Aqui apresentaremos apenas uma breve introdução. Detalhes adicionais podem ser encontrados, por exemplo, em

- Jacobi, R. (1996). Síntese de Circuitos Lógicos Combinacionais. Décima Escola de Computação, Campinas.
- Perkowski, M. A., Grygiel, S., and the Functional Decomposition Group (1995). A Survey of Literature on Function Decomposition - Version IV. Technical report, PSU Electrical Engineering Department.

Na **decomposição funcional**, o objetivo é, dada uma função f de n variáveis, escrevê-la como composição de duas ou mais funções com menor número de variáveis.

Exemplo: seja f uma função com n variáveis $x_1, x_2, \ldots, x_n$. Uma possível decomposição de f pode ser dada por

$$f(x_1, x_2, \ldots, x_n) = g(h(x_1, x_2, \ldots, x_k), x_{k+1}, \ldots, x_n)$$

Tanto g como h são funções que dependem de menos variáveis. Consequentemente, pode ser mais fácil encontrar uma realização eficiente e barata de g e h do que uma realização eficiente e barata de f.

Os circuitos resultantes da decomposição funcional possuem uma estrutura hierárquica, ou seja, de múltiplos níveis, onde cada uma das subfunções pode ser vista como um módulo. Veja a figura 1.18. Esta estrutura de decomposição é apenas um exemplo de uma possível forma de decomposição. Dada

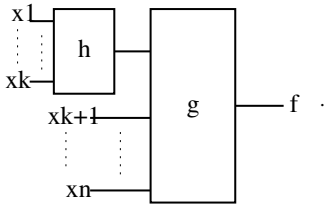

Figura 1.18: Exemplo de decomposição funcional.

uma função qualquer, verificar se ela pode ser expressa em uma certa estrutura de decomposição não é tarefa fácil.

Na **decomposição estrutural**, o objetivo é, dada uma função f de n variáveis, encontrar subfunções comuns que possam ser compartilhadas. Essa idéia é mostrada através de um exemplo. Consideremos

a função f dada por

$$f = x_1\, x_3\, \overline{x}_4 + x_1\, x_5 + x_2\, x_3\, \overline{x}_4 + x_2\, x_5 + x_3\, \overline{x}_4\, x_7 + x_5\, x_7 + x_6$$

Rearranjando os produtos e efetuando algumas manipulações,

$$\begin{aligned} f &= \underbrace{x_1\, x_3\, \overline{x}_4 + x_2\, x_3\, \overline{x}_4}_{\underbrace{(x_1 + x_2)}_{y_1}\, x_3\, \overline{x}_4} + \underbrace{x_1\, x_5 + x_2\, x_5}_{\underbrace{(x_1 + x_2)}_{y_1}\, x_5} + \underbrace{x_3\, \overline{x}_4\, x_7 + x_5\, x_7}_{x_7\, \underbrace{(x_3\, \overline{x}_4 + x_5)}_{y_2}} + x_6 \\ &= \underbrace{y_1\, x_3\, \overline{x}_4 + y_1\, x_5}_{y_1\, \underbrace{(x_3\, \overline{x}_4 + x_5)}_{y_2}} + \underbrace{y_2\, x_7 + x_6}_{y_3} \\ &= y_1\, y_2 + y_3 \end{aligned}$$

O compartilhamento de subfunções resulta em uma estrutura de rede de funções, conforme mostrada na figura 1.19. Este tipo de decomposição pode ser realizado levando-se em conta a implementação

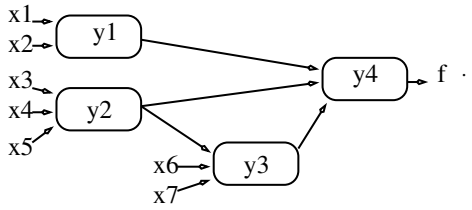

Figura 1.19: Exemplo de decomposição estrutural.

de mais de uma função booleana.

O projeto multi-níveis é um problema computacionalmente difícil. Para abordá-lo, é necessário estabelecer as condições de contorno que caracterizam o tipo de decomposição desejada. Entre estas condições, podemos citar o número máximo de níveis, o número máximo de variáveis de entrada em cada subfunção, a utilização de determinados componentes, etc. A solução do problema, mesmo em condições de contorno bem limitadas, não é trivial ou então nem existe ainda.