

Lógica combinacional modular e multi-níveis

Uma possível estruturação de um programa de computador é a sua decomposição em módulos/funções. Do ponto de vista global, apenas interessam as entradas e saídas dos módulos para que os mesmos possam ser compostos de forma adequada no programa.

Este mesmo princípio pode ser observado também no projeto de circuitos lógicos. Alguns circuitos comumente utilizados podem ser modularizados e utilizados na realização de circuitos complexos. Exemplos de módulos bastante usados são os decodificadores, codificadores, multiplexadores e demultiplexadores.

Quando módulos são utilizados, em geral a realização de circuitos tende a ser multi-níveis (mais de dois níveis de portas lógicas). Uma consequência imediata disso é que esses circuitos serão mais lentos que os circuitos dois-níveis. No entanto, há casos em que uma realização multi-níveis utiliza menos portas lógicas do que uma realização dois níveis. Além disso, a modularidade é um aspecto altamente atraente no projeto de circuitos complexos (por possibilitar, por exemplo, testes por partes e uma melhor visão de sua estrutura global).

Esta seção apresenta circuitos conhecidos por decodificadores, codificadores, multiplexadores e demultiplexadores. Apresenta também alguns exemplos de sua utilização em circuitos específicos e na realização de funções booleanas quaisquer. Inclui também uma breve introdução à lógica multi-níveis. Referências para este capítulo: [Nelson et al., 1995, Hill and Peterson, 1993].

1.1 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 1.1 mostra um esquema genérico de um decodificador $n : 2^n$. As entradas $x_{n-1} \dots x_1 x_0$ 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 BA (i.e., $x_0 = A$ e $x_1 = B$). Um circuito correspondente ao decodificador é ilustrado na figura 1.2. 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 estendido 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

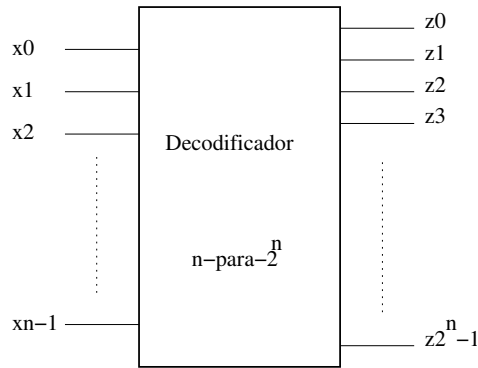


Figura 1.1: Esquema de um decodificador $n : 2^n$.

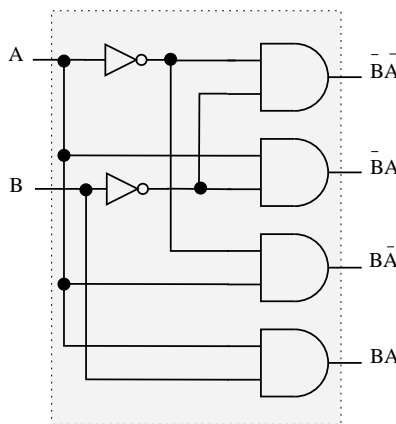


Figura 1.2: Circuito de um decodificador $2 : 4$.

este valor n . Para valores grandes de n , decodificadores são realizados por circuitos multi-níveis, conforme veremos mais adiante.

1.1.1 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 ($x_1 x_0$) fornecido 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

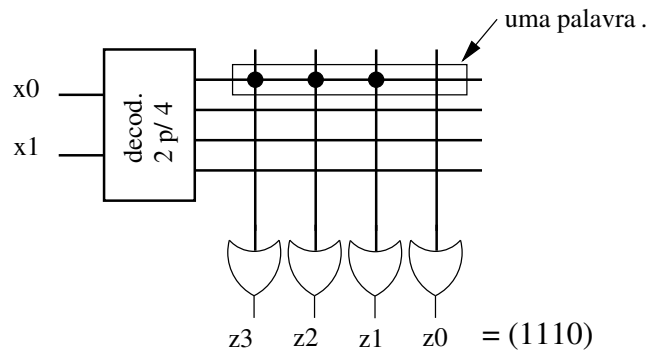


Figura 1.3: Esquema de uma memória ROM com 4 posições e palavras de 4 bits.

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 (*Erasable Programmable ROM*).

1.1.2 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 decodificadores 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 1.4. No caso da realização com porta NÃO-OU, observe que $f(a, b, c) = \prod M(2, 3, 5) = \overline{M_2} \cdot \overline{M_3} \cdot \overline{M_5} = \overline{M_2 + M_3 + M_5} = \overline{m_2 + m_3 + m_5}$.

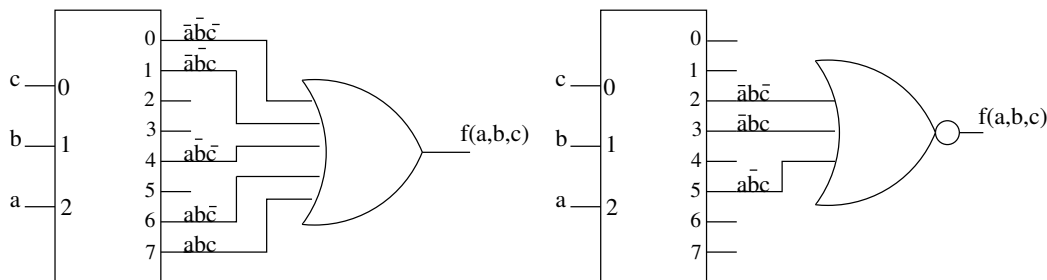


Figura 1.4: Realização de $f(a, b, c) = \sum m(0, 1, 4, 6, 7)$ com decodificador 3 : 8 e uma porta OU (esquerda) ou uma porta NÃO-OU (direita).

1.1.3 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 1.5 mostra como pode ser realizado um decodificador $12 : 2^{12}$ em três níveis.

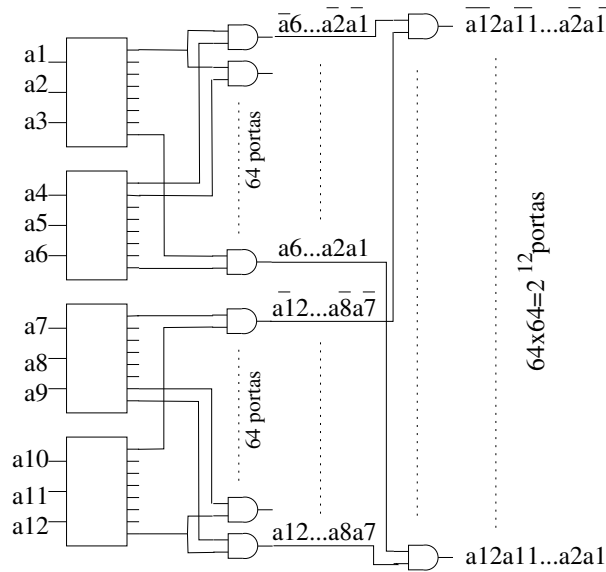


Figura 1.5: Realização três-níveis de um decodificador $12 : 2^{12}$.

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 1.6 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, como o exemplo mostrado na figura 1.2. 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 1.5, mas para decodificadores de muitas entradas pode ser a única solução.

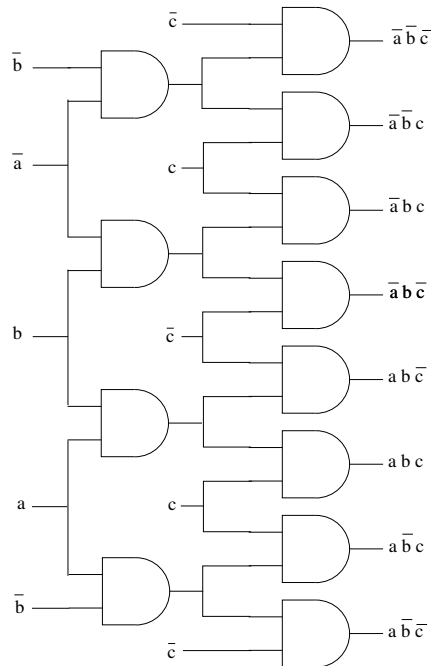


Figura 1.6: Decodificador em estrutura de árvore.

Na prática, as realizações de decodificadores para um número grande de entradas é baseada em uma combinação das estruturas da figura 1.5 e da figura 1.6.

1.2 Codificadores

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

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

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 1.7 mostra o esquema de um codificador de n entradas.

Embora usualmente os codificadores sejam definidos como um circuito no qual apenas uma entrada

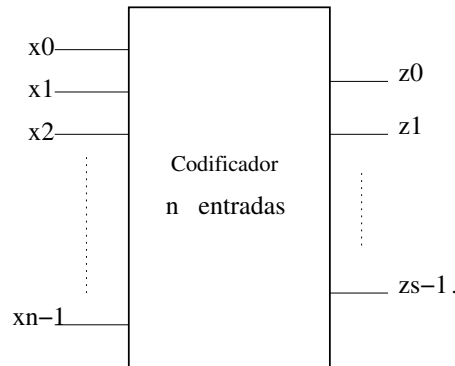


Figura 1.7: Esquema de um codificador.

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.

1.2.1 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 1.8.

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 realiza 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 serão consideradas neste curso.

1.3 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 1.9 mostra um multiplexador 4×1 , isto é, um multiplexador de 4 entradas.

Se os seletores forem $S_1 S_0 = 00$, então teremos $Z = D_0$; se forem $S_1 S_0 = 01$, então teremos $Z = D_1$; $S_1 S_0 = 10$, então teremos $Z = D_2$; $S_1 S_0 = 11$, 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_3 . Assim, podemos escrever Z como

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

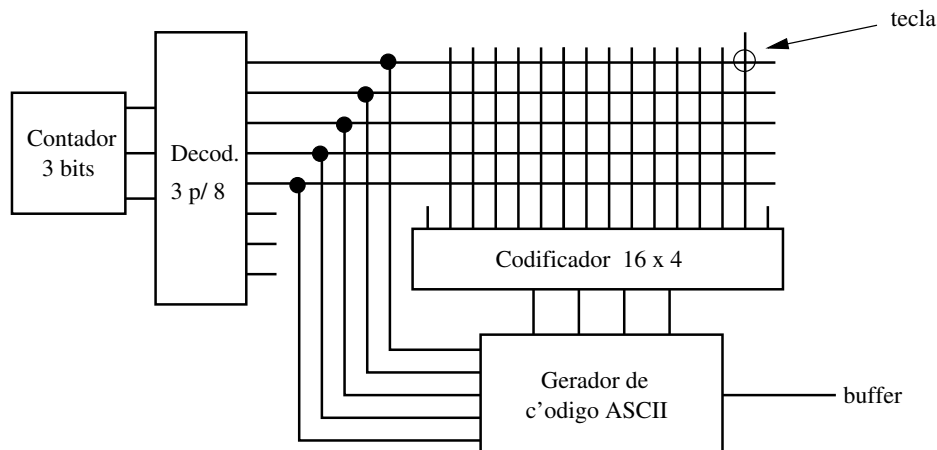


Figura 1.8: Esquema de um teclado.

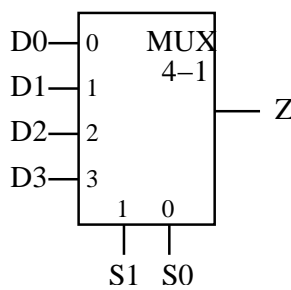


Figura 1.9: Multiplexador de 4 entradas.

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 1.10. Note que para cada possível combinação de valores de S_1S_0 , apenas um dos produtos toma valor 1 na equação acima.

1.3.1 Realização de funções com MUX

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

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

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

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

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

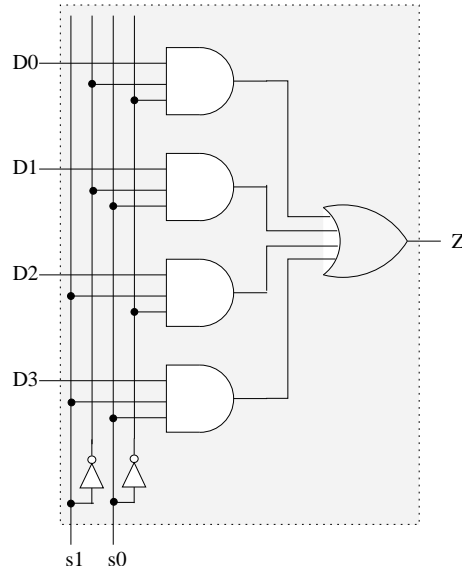


Figura 1.10: Circuito de um multiplexador de 4 entradas.

Exercício 1: Escreva a realização da função $f(a, b, c) = \bar{a}\bar{b} + ac$ 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).

1.3.2 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 1.11.

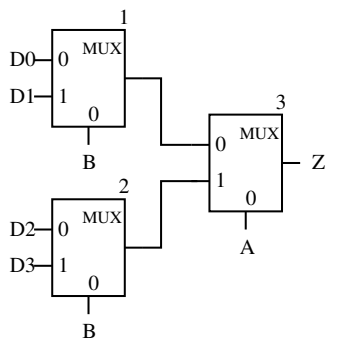


Figura 1.11: Realização de um MUX 4×1 como composição de três MUX 2×1 .

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 ?

1.3.3 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 1.12.

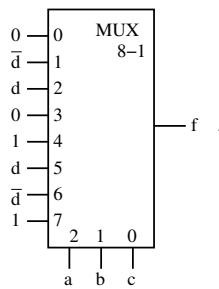


Figura 1.12: Realização de $f(a, b, c, d) = \sum m(2, 5, 8, 9, 11, 12, 14, 15)$ com um MUX 8×1 .

Se pensarmos em utilizar MUX 4×1 e MUX 2×1 na realização de f , duas possíveis soluções, mostradas na figura 1.13, 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 variável 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.

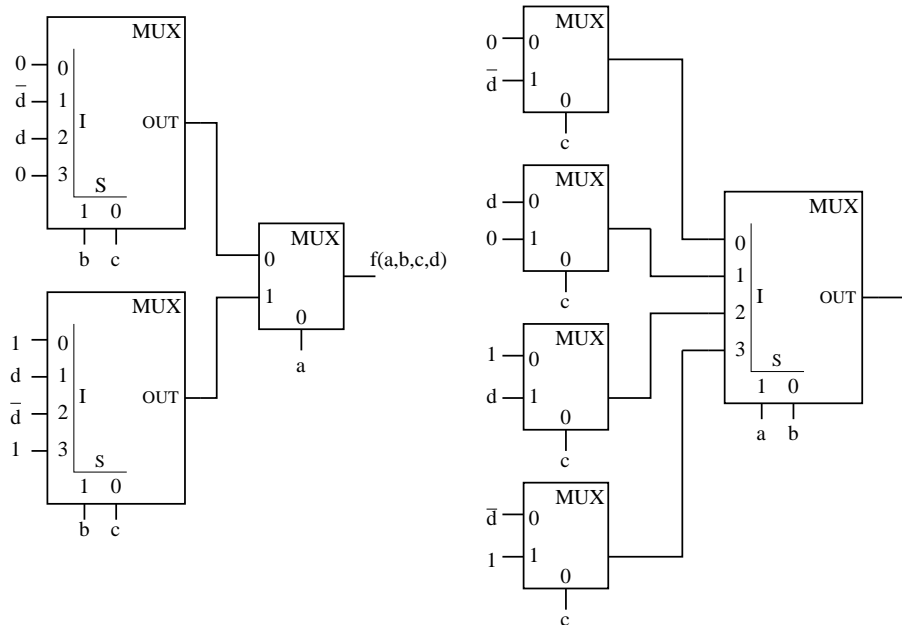


Figura 1.13: Realização de $f(a, b, c, d) = \sum m(2, 5, 8, 9, 11, 12, 14, 15)$ com (a) dois MUX 4×1 no primeiro nível e um MUX 2×1 no segundo nível e (b) 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

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

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

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:

$$f(x_1, \dots, x_k, \dots, x_n) = \bar{x}_k f(x_1, \dots, 0, \dots, x_n) + x_k f(x_1, \dots, 1, \dots, x_n)$$

As funções $f(x_1, \dots, 0, \dots, x_n)$ e $f(x_1, \dots, 1, \dots, x_n)$ 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 1.12 acima. Vamos mostrar agora que aplicando-se sucessivamente a expansão de Shannon é possível obter diferentes estruturas de realizações de f

usando MUX.

$$\begin{aligned}
 f &= \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + a\bar{b}\bar{c}\bar{d} + a\bar{b}c\bar{d} + a\bar{b}cd + ab\bar{c}\bar{d} + abc\bar{d} + abcd \\
 &= \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + a\bar{b}\bar{c}\bar{d} + ab\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}cd + abc\bar{d} + abcd \quad (\text{rearranjo}) \\
 &= \bar{a}\bar{b}c\bar{d} + \bar{a}b\bar{c}d + a\bar{c}\bar{d} + a\bar{b}d + abc \quad (\text{algumas simplificações}) \\
 &= \bar{a}(\bar{b}c\bar{d} + b\bar{c}d) + a(\bar{c}\bar{d} + \bar{b}d + bc) \quad (\text{expansão em torno de } a) \\
 &= \bar{a}(\bar{b}(c\bar{d}) + b(\bar{c}d)) + a(\bar{b}(\bar{c}\bar{d} + d) + b(\bar{c}\bar{d} + c)) \quad (\text{expansão em torno de } b) \\
 &= \bar{a}\bar{b}(c\bar{d}) + \bar{a}b(\bar{c}d) + a\bar{b}(\bar{c}\bar{d} + d) + ab(\bar{c}\bar{d} + c) \quad (\text{distribuição com respeito a } a) \\
 &= \bar{a}\bar{b}(\bar{c}(0) + c(\bar{d})) + \bar{a}b(\bar{c}(d) + c(0)) + a\bar{b}(\bar{c}(1) + c(d)) + ab(\bar{c}(\bar{d}) + c(1)) \quad (\text{expansão em torno de } c)
 \end{aligned}$$

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:

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

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.

1.4 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.

1.5 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 ímpar, 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 ímpar.

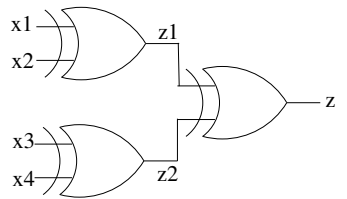


Figura 1.14: Circuito para verificação de paridade para uma entrada de 4 bits.

Isto pode ser verificado analisando-se a seguinte tabela.

# de x_i s iguais a 1	# de z_i s 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 ímpar se, e somente se, o número de entradas $x_i = 1$ é também ímpar. Se o número de $z_i = 1$ é ímpar, então $z = 1$. Assim, se a paridade é par, temos $z = 0$ e se é ímpar temos $z = 1$.

O circuito acima pode ser estendido para verificar a paridade de 8 bits, simplesmente conectando-se a saída de dois circuitos daqueles a uma porta XOR, conforme mostrado na figura 1.15. Para realizar tal circuito são necessários 7 portas XOR ou então 21 portas NÃO-E e 14 inversores.

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

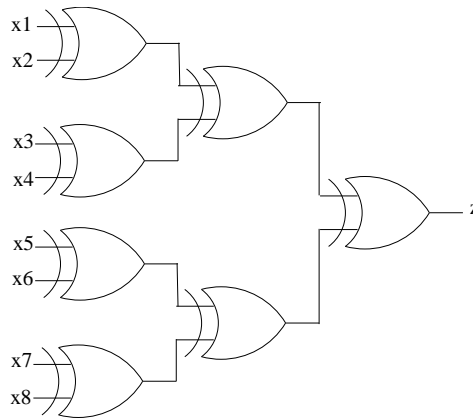


Figura 1.15: Circuito para verificação de paridade para entradas de 8 bits.

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.

1.5.1 Decomposição funcional e estrutural de funções

Não é objetivo deste curso aprofundar-se nestes tópicos. Aqui apresentaremos apenas uma breve introdução.

Na **decomposição funcional** [Perkowski et al., 1995], 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, \dots, x_n . Uma possível decomposição de f pode ser dada por

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

Tanto g como h são funções que dependem de menos variáveis. Conseqüentemente, 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 1.16. Esta estrutura de decomposição é apenas um exemplo de uma possível forma de decomposição. Dada uma função qualquer, verificar se ela pode ser expressa em uma certa estrutura de decomposição não é tarefa fácil.

Na **decomposição estrutural** [Jacobi, 1996], 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

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

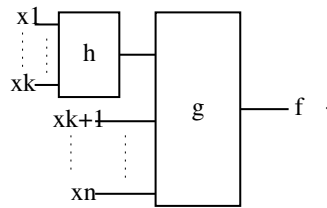


Figura 1.16: Exemplo de decomposição funcional.

Rearranjando os produtos e efetuando algumas manipulações,

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

O compartilhamento de subfunções resulta em uma estrutura de rede de funções, conforme mostrada na figura 1.17. Este tipo de decomposição pode ser realizado levando-se em conta a implementação

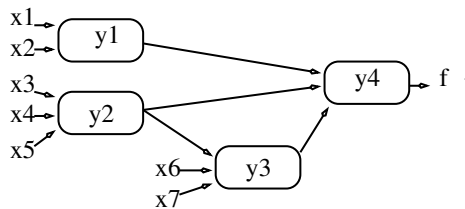


Figura 1.17: Exemplo de decomposição estrutural.

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. O projeto multi-níveis pode também ser aplicado no contexto de aprendizado computacional (indução de classificadores) [Zupan, 1997] e de operadores morfológicos para processamento de imagens.

Referências Bibliográficas

- [Hill and Peterson, 1993] Hill, F. J. and Peterson, G. R. (1993). *Computer Aided Logical Design with Emphasis on VLSI*. John Wiley & Sons, fourth edition.
- [Jacobi, 1996] Jacobi, R. (1996). Síntese de Circuitos Lógicos Combinacionais. Décima Escola de Computação, Campinas.
- [Nelson et al., 1995] Nelson, V. P., Nagle, H. T., Carroll, B. D., and Irwin, J. D. (1995). *Digital Logic Circuit Analysis and Design*. Prentice-Hall.
- [Perkowski et al., 1995] 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.
- [Zupan, 1997] Zupan, B. (1997). *Machine Learning Based on Function Decomposition*. PhD thesis, University of Ljubljana.