

10. Multiplexadores

Alguns sistemas digitais bastante úteis podem ser formados a partir de portas lógicas. Eles possuem aplicações recorrentes e de grande utilidade em sistemas digitais mais complexos e por isso são implementados em circuitos integrados. Neste capítulo um desses sistemas será investigado com maior profundidade, o multiplexador. O multiplexador, ou simplesmente mux, pode se encararado como um bloco de digital capaz de realizar a operação de chaveamento ou comutação.

O mux pode ser implementado por meio de portas lógicas. Essa implementação é discutida no presente capítulo. As famílias de portas lógicas possuem circuitos integrados específicos que implementam multiplexadores.

10.1 Descrição dos Multiplexadores

10.1.1 Multiplexadores 2x1

O multiplexador mais simples é o mux-2x1. O símbolo esquemático usado para representar esse mux está mostrado na Fig. 10.3a. O mux-2x1 é um dispositivo digital que possui 3 entradas e uma saída. Há duas entradas x_0 e x_1 , genericamente denominadas de “entradas de dados”, e uma entrada s , genericamente denominada de “entrada de seleção”. O funcionamento do mux-2x1 pode ser descrito de forma resumida assim: se a entrada de seleção s for igual a 0, a saída z do mux é igual a entrada x_0 , se a entrada de seleção s for igual a 1, a saída z do mux é igual a entrada x_1 . A Tabela 10.1 descreve a funcionalidade do mux-2x1 resumidamente.

Tabela 10.1: Tabela verdade resumida do mux-2x1.

x_1	x_0	s	z
-	-	1	x_1
-	-	0	x_0

Resumidamente, o mux-2x1 constitui um chave eletrônica em que se escolhe, via entrada de seleção s , qual das entradas (x_0 ou x_1) será transferida para a saída do mux. Por isso a denominação

2x1 é utilizada para esse mux.

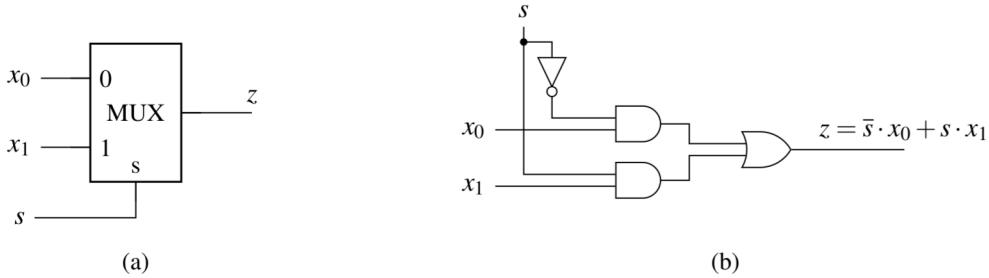


Figura 10.1: Circuito multiplexador 2x1: (a) diagrama de blocos esquemático (b) implementação com portas lógicas.

A tabela verdade completa do mux-2x1 e o mapa-K representativo (com os respectivos agrupamentos de 1s) do sistema estão mostrados na Fig. 10.2.

s	x_1	x_0	z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a)

		$x_1 x_0$			
		00	01	11	10
s	0	0	1	1	0
	1	0	0	1	1

(b)

Figura 10.2: Multiplexador 2x1: (a) descrição pela tabela verdade, (b) descrição/implementação mínima usando mapas-K.

Observando o mapa-K referente ao mux-2x1 (mostrado na Fig. 10.2b) é possível escrever a expressão lógica mínima que o implementa:

$$z = x_1 \cdot s + x_0 \cdot \bar{s}. \quad (10.1)$$

A implementação usando portas lógicas do mux-2x1 está mostrada na Fig. 10.1b.

10.1.2 Multiplexadores 4x1

De forma análoga ao multiplexador mux-2x1 é possível se definir multiplexadores maiores. Nesta seção o mux-4x1 é descrito. O símbolo esquemático usado para representar esse mux está mostrado na Fig. 10.3a.

O mux-4x1 é um dispositivo digital que possui 6 entradas e uma saída. Há quatro entradas x_0 , x_1 , x_2 e x_3 , genericamente denominadas de “entradas de dados”, e duas entradas s_0 e s_1 , genericamente denominada de “entradas de seleção”. O funcionamento do mux-2x1 pode ser descrito de forma resumida assim: há quatro entradas de dados. As entradas de seleção selecionam qual das quatro

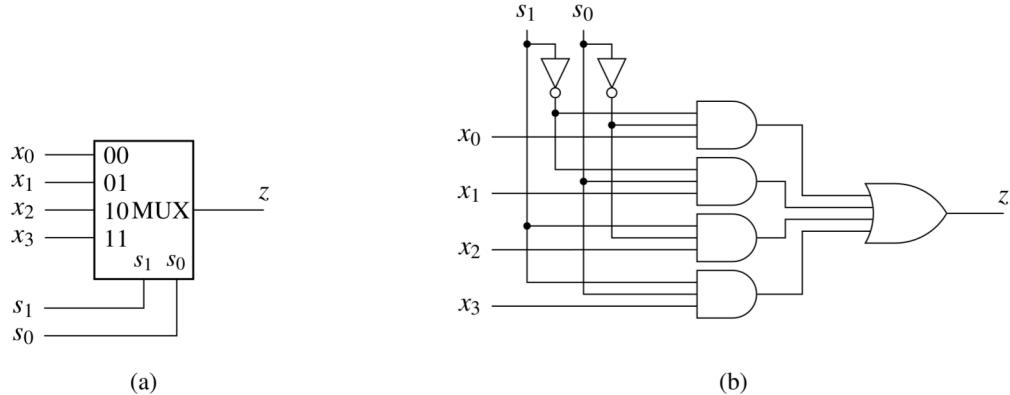


Figura 10.3: Circuito multiplexador 2x1: (a) diagrama de blocos esquemático (b) implementação com portas lógicas.

Tabela 10.2: Tabela verdade resumida do multiplexador 4x1

x_3	x_2	x_1	x_0	s_1	s_0	z
-	-	-	-	0	0	x_0
-	-	-	-	0	1	x_1
-	-	-	-	1	0	x_2
-	-	-	-	1	1	x_3

entradas de dados é reproduzida na saída do mux. Por exemplo, se as entradas de seleção s_1s_0 forem iguais a 00, a saída z do mux é igual a entrada x_0 . Na saída do mux será reproduzida a entrada cujo sub-índice decimal corresponde ao binário colocado nas entradas de seleção s_1 e s_0 (nessa ordem). A Tabela 10.2 descreve a funcionalidade do mux-4x1 resumidamente.

Como o mux-4x1 é um sistema de 6 entradas, a sua tabela verdade contém 64 linhas e por isso ela não será mostrada de forma completa, mas apenas de forma resumida como na Tabela 10.2. No entanto, na Fig. 10.4 está mostrado o mapa-K completo do sistema mux-4x1 com os agrupamentos de 1s convenientes para obter a expressão mínima para esse sistema.

Escrevendo a expressão mínima que pode ser obtida dos agrupamentos mostrados no mapa-K chega-se a:

$$z = x_0 \cdot \overline{s_1} \cdot \overline{s_0} + x_1 \cdot \overline{s_1} \cdot s_0 + x_2 \cdot s_1 \cdot \overline{s_0} + x_3 \cdot s_1 \cdot s_0 \quad (10.2)$$

A implementação usando portas lógicas do mux-4x1 está mostrada na Fig. 10.3b.

10.2 Universalidade dos multiplexadores

Multiplexadores juntamente com o uso das contantes 0 e 1 formam conjuntos universais. Como visto na Seção ?? a universalidade de um conjunto de portas pode ser mostrado por meio da implementação de todas as portas de um conjunto universal conhecido. Extrapolando o conceito de conjunto de portas para conjunto de dispositivos (como os multiplexadores) é possível se fazer essa demonstração usando a mesma ideia.

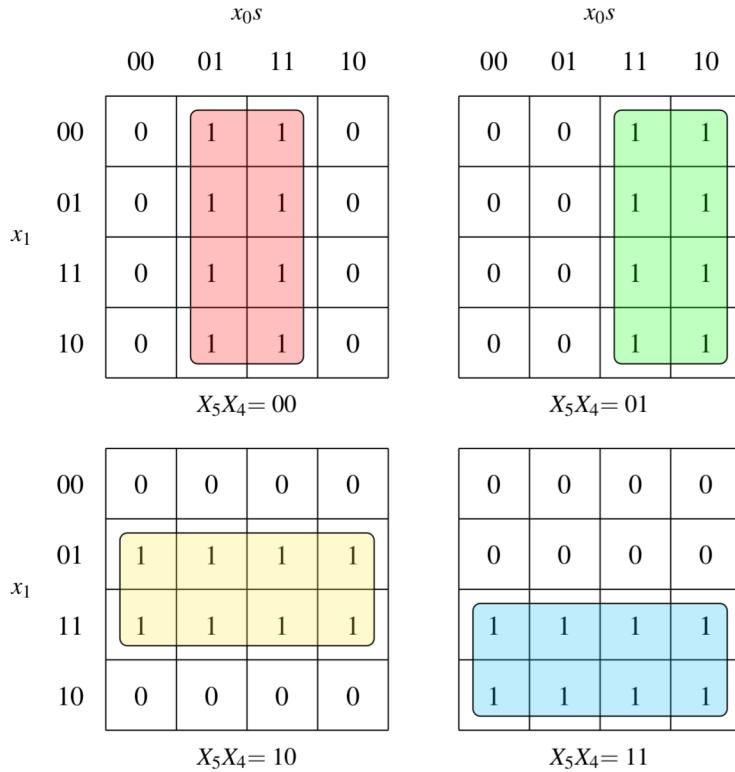


Figura 10.4: Mapa-K e agrupamentos de 1s para a implementação mínima do mux-4x1.

Nas subseções 10.2.1 e 10.2.2 são mostradas as implementações referentes aos conjuntos universais que podem ser formados com mux-2x1 e mux-4x1, demonstrando como implementar as portas lógicas do conjunto {AND, OR, NOT} usando ou apenas mux-2x1 (e constantes) ou apenas mux-4x1 (e constantes).

10.2.1 Conjunto universal com multiplexadores 2x1

Tomando como partida a expressão (10.1), a mínima que define o mux-2x1, e fazendo substituições nos termos x_0 , x_1 , e s é possível implementar as portas {NOT, AND, OR}. A implementação de NOT é feita se aplicando $x_0 = 1$, $x_1 = 0$, e $s = x$. Assim, substituindo em (10.1) vem:

$$\begin{aligned}
 z &= x_1 \cdot s + x_0 \cdot \bar{s} \\
 &= 0 \cdot s + 1 \cdot \bar{x} \\
 &= \bar{x},
 \end{aligned} \tag{10.3}$$

que demonstra que NOT pode ser implementada com um mux-2x1. A segunda coluna da primeira linha da Tabela 10.3 mostra essa implementação.

A implementação de AND é feita se aplicando $x_0 = 0$, $x_1 = x$ e $s = y$. Assim, substituindo em

(10.1) vem:

$$\begin{aligned}
 z &= x_1 \cdot s + x_0 \cdot \bar{s} \\
 &= x \cdot y + 0 \cdot \bar{y} \\
 &= x \cdot y,
 \end{aligned} \tag{10.4}$$

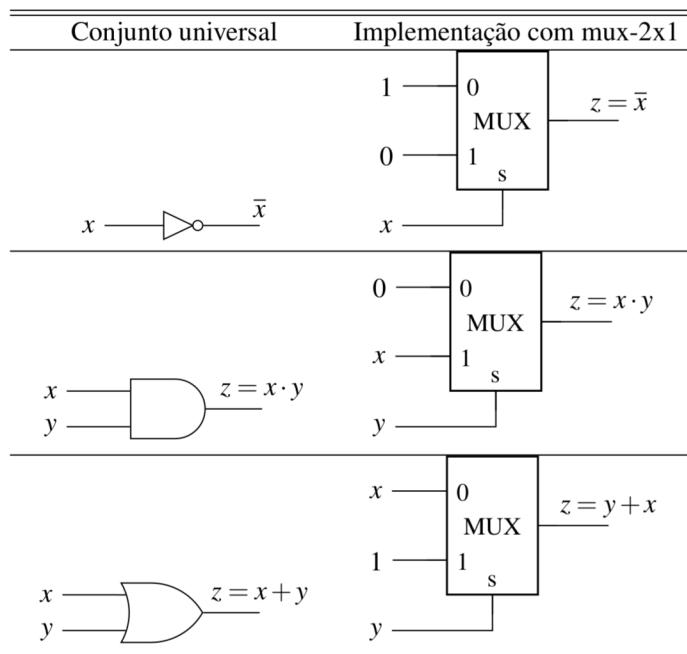
que demonstra que AND pode ser implementada com um mux-2x1. A segunda coluna da segunda linha da Tabela 10.3 mostra essa implementação.

A implementação de OR é feita se aplicando $x_0 = x$, $x_1 = 1$ e $s = y$. Assim, substituindo em (10.1) vem:

$$\begin{aligned}
 z &= x_1 \cdot s + x_0 \cdot \bar{s} \\
 &= 1 \cdot y + x \cdot \bar{y} \\
 &= y + x \cdot \bar{y} \\
 &= y + x,
 \end{aligned} \tag{10.5}$$

que demonstra que OR pode ser implementada com um mux-2x1. A segunda coluna da terceira linha da Tabela 10.3 mostra essa implementação.

Tabela 10.3: Implementação de um conjunto universal conhecido {NOT, AND, OR} usando mux-2x1 e as constantes 0 e 1.



10.2.2 Conjunto universal com multiplexadores 4x1

De forma análoga à feita para os mux-2x1 também é possível se demonstrar a implementação das portas NOT, AND e OR usando apenas um mux-4x1 e as constantes 0 e 1.

Tomando como partida a expressão (10.2), a mínima que define o mux-4x1, e fazendo substituições nos termos $x_0, x_1, e s$ é possível de implementar as portas {NOT, AND, OR}. A implementação de NOT é feita se aplicando $x_0 = 1, x_1 = 0, x_2 = -, x_3 = -, s_1 = 0, s_0 = x$. As entradas indicadas com “-” não são utilizadas para a implementação da função NOT com o mux-4x1. Assim, substituindo em (10.2) vem:

$$\begin{aligned} z &= x_0 \cdot \bar{s}_1 \cdot \bar{s}_0 + x_1 \cdot \bar{s}_1 \cdot s_0 + x_2 \cdot s_1 \cdot \bar{s}_0 + x_3 \cdot s_1 \cdot s_0 \\ &= 1 \cdot 1 \cdot \bar{x} + 0 \cdot 1 \cdot x + x_2 \cdot 0 \cdot \bar{x} + x_3 \cdot 0 \cdot x \\ &= \bar{x}, \end{aligned} \quad (10.6)$$

que demonstra que NOT pode ser implementada com mux-4x1. A segunda coluna da primeira linha da Tabela 10.4 mostra essa implementação.

A implementação de AND é feita se aplicando $x_0 = 0, x_1 = 0, x_2 = 0, x_3 = 1, s_1 = x, s_0 = y$. Assim, substituindo em (10.2) vem:

$$\begin{aligned} z &= x_0 \cdot \bar{s}_1 \cdot \bar{s}_0 + x_1 \cdot \bar{s}_1 \cdot s_0 + x_2 \cdot s_1 \cdot \bar{s}_0 + x_3 \cdot s_1 \cdot s_0 \\ &= 0 \cdot \bar{x} \cdot \bar{y} + 0 \cdot \bar{x} \cdot y + 0 \cdot x \cdot \bar{y} + 1 \cdot x \cdot y \\ &= x \cdot y \end{aligned} \quad (10.7)$$

que demonstra que AND pode ser implementada com mux-4x1. A segunda coluna da segunda linha da Tabela 10.4 mostra essa implementação.

A implementação de OR é feita se aplicando $x_0 = 0, x_1 = 1, x_2 = 1, x_3 = 1, s_1 = x, s_0 = y$. Assim, substituindo em (10.2) vem:

$$\begin{aligned} z &= x_0 \cdot \bar{s}_1 \cdot \bar{s}_0 + x_1 \cdot \bar{s}_1 \cdot s_0 + x_2 \cdot s_1 \cdot \bar{s}_0 + x_3 \cdot s_1 \cdot s_0 \\ &= 0 \cdot \bar{x} \cdot \bar{y} + 1 \cdot \bar{x} \cdot y + 1 \cdot x \cdot \bar{y} + 1 \cdot x \cdot y \\ &= \bar{x} \cdot y + x \cdot \bar{y} + x \cdot y \\ &= \bar{x} \cdot y + x \\ &= x + y, \end{aligned} \quad (10.8)$$

que demonstra que OR pode ser implementada com mux-4x1. A segunda coluna da terceira linha da Tabela 10.4 mostra essa implementação.

Há também a possibilidade de se implementar ANDs e ORs de três entradas com os mux-4x1.

A implementação de AND de três entradas (rotuladas como a_0, a_1 e a_2) com mux-4x1 é feita se aplicando $x_0 = 0, x_1 = 0, x_2 = 0, x_3 = a_0, s_1 = a_1, s_0 = a_2$. Assim, substituindo em (10.2) vem:

$$\begin{aligned} z &= x_0 \cdot \bar{s}_1 \cdot \bar{s}_0 + x_1 \cdot \bar{s}_1 \cdot s_0 + x_2 \cdot s_1 \cdot \bar{s}_0 + x_3 \cdot s_1 \cdot s_0 \\ &= 0 \cdot \bar{a}_1 \cdot \bar{a}_2 + 0 \cdot \bar{a}_1 \cdot a_2 + 0 \cdot a_1 \cdot \bar{a}_2 + a_0 \cdot a_1 \cdot a_2 \\ &= a_0 \cdot a_1 \cdot a_2, \end{aligned} \quad (10.9)$$

que demonstra que AND de três entradas pode ser implementada com mux-4x1. A Fig. 10.5a mostra a implementação do AND de três entradas usando mux-4x1.

A implementação de OR de três entradas (rotuladas como a_0, a_1 e a_2) com mux-4x1 é feita se

Tabela 10.4: Implementação de um conjunto universal conhecido {NOT, AND, OR} usando mux-4x1 e as constantes 0 e 1.

Conjunto universal	Implementação com mux-4x1
$x \rightarrow \bar{x}$	
$x \cdot y = z$	
$x + y = z$	

aplicando $x_0 = a_0$, $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $s_1 = a_1$, $s_0 = a_2$. Assim, substituindo em (10.2) vem:

$$\begin{aligned}
 z &= x_0 \cdot \bar{s}_1 \cdot \bar{s}_0 + x_1 \cdot \bar{s}_1 \cdot s_0 + x_2 \cdot s_1 \cdot \bar{s}_0 + x_3 \cdot s_1 \cdot s_0 \\
 &= a_0 \cdot \bar{a}_1 \cdot \bar{a}_2 + 1 \cdot \bar{a}_1 \cdot a_2 + 1 \cdot a_1 \cdot \bar{a}_2 + 1 \cdot a_1 \cdot a_2 \\
 &= a_0 \cdot \bar{a}_1 \cdot \bar{a}_2 + \bar{a}_1 \cdot a_2 + a_1 \cdot \bar{a}_2 + a_1 \cdot a_2 \\
 &= a_0 \cdot \bar{a}_1 \cdot \bar{a}_2 + \bar{a}_1 \cdot a_2 + a_1 \\
 &= a_0 \cdot \bar{a}_1 \cdot \bar{a}_2 + (a_1 + a_2) \\
 &= a_0 \cdot (\overline{a_1 + a_2}) + (a_1 + a_2) \\
 &= a_0 + a_1 + a_2
 \end{aligned} \tag{10.10}$$

que demonstra que OR de três entradas pode ser implementada com mux-4x1. A Fig. 10.5b mostra a implementação do OR de três entradas usando mux-4x1.

10.3 Expansão de Shannon

Há uma forma de se fazer a descomposição de uma função de chaveamento binária a partir de suas variáveis. Isso pode ser feito por meio da expansão de Shannon. A expansão de Shannon apresenta versões nas forma de soma de produtos e de produto de somas. Ambas são discutidas nessa seção. É interessante notar que apesar da expansão de Shannon ser uma construção teórica ela permite sua implementação direta em *hardware* via multiplexadores. Pode-se dizer que os multiplexadores são

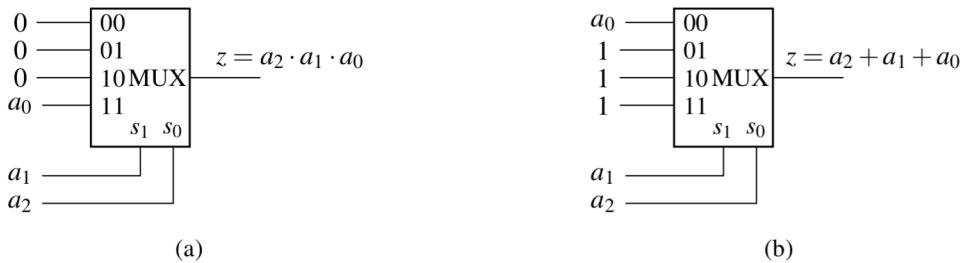


Figura 10.5: Implementação usando mux-4x1 das função de três entradas: (a) AND e (b) OR.

dispositivos físicos reais que implementam diretamente a identidade abstrata/teórica formalizada pela expansão de Shannon.

10.3.1 Forma soma de produtos

A forma mais conhecida da expansão de Shannon é a enunciada pelo teorema:

Teorema 10.3.1 (Teorema da expansão/decomposição de Shannon) Toda função de chaveamento de n variáveis pode ser decomposta usando a expressão:

$$f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = \overline{x_0} \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 0) + x_0 \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 1). \quad (10.11)$$

Além disso, qualquer uma das variáveis x_k , $k \in \{0, 1, \dots, n-1\}$ pode ser escolhida para realizar a expansão mostrada na expressão.

A expressão (10.11) é uma forma elegante de se aplicar um teste lógico na variável x_0 e obter resultados diferentes de acordo com o resultado do teste. Se a variável x_0 é igual a 0, a primeira parcela do lado direito de (10.11) permanece e a segunda parcela é zerada. No caso da variável x_0 ser igual a 1 acontece o oposto, a primeira parcela do lado direito de (10.11) é zerada e a segunda parcela permanece na expressão. Sendo assim, a expressão (10.11) é uma outra forma de se escrever:

$$f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = \begin{cases} f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 0), & \text{se } x_0 = 0 \\ f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 1), & \text{se } x_0 = 1. \end{cases} \quad (10.12)$$

Compare (10.11) e (10.12), substituindo $x_0 = 0$ ou $x_0 = 1$ convenientemente, e perceba que elas são equivalentes.

Em (10.11) a variável expandida é a variável x_0 . No entanto, qualquer uma das variáveis da função pode ser escolhida para realizar a expansão. Alguns exemplos são a expansão da variável x_2 :

$$f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = \overline{x_2} \cdot f(x_{n-1}, x_{n-2}, \dots, 0, x_1, x_0) + x_2 \cdot f(x_{n-1}, x_{n-2}, \dots, 1, x_1, x_0)$$

ou da variável x_{n-1} :

$$f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = \overline{x_{n-1}} \cdot f(0, x_{n-2}, \dots, x_2, x_1, x_0) + x_{n-1} \cdot f(1, x_{n-2}, \dots, x_2, x_1, x_0).$$

Ao se comparar (10.1) com (10.11) percebe-se facilmente que a expansão de Shannon mostrada em (10.11) pode ser fisicamente implementada por um mux-2x1. Essa implementação para uma função genérica de n variáveis está mostrada na Fig. 10.6.

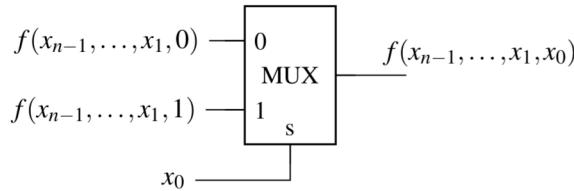


Figura 10.6: Implementação em *hardware* da expansão de Shannon usando um mux-2x1.

A expansão mostrada em (10.11) é aplicada a apenas uma variável (x_0) gerando dois termos produtos. Ela pode ser aplicada novamente às funções $f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 0)$ e $f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 1)$ expandindo-se uma segunda variável, por exemplo x_1 . Ao fazer isso facilmente obtém-se:

$$\begin{aligned} f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = & \bar{x}_1 \cdot \bar{x}_0 \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, 0, 0) + \\ & + \bar{x}_1 \cdot x_0 \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, 0, 1) + \\ & + \bar{x}_1 \cdot x_0 \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, 1, 0) + \\ & + x_1 \cdot x_0 \cdot f(x_{n-1}, x_{n-2}, \dots, x_2, 1, 1). \end{aligned} \quad (10.13)$$

A implementação de (10.13) pode ser feita diretamente em *hardware* com um mux-4x1 como mostrado na Fig. 10.7. Esse é o formato da expansão de Shannon aplicada diretamente a duas variáveis de uma única vez. Perceba que quaisquer duas variáveis da função podem ser escolhidas para realizar essa expansão.

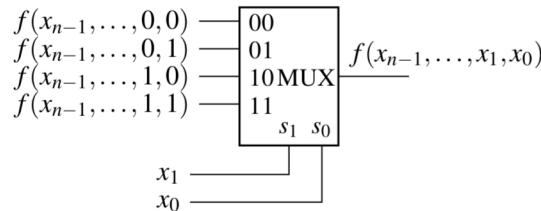


Figura 10.7: Implementação em *hardware* da expansão de Shannon usando um mux-4x1.

A expansão aplicada a apenas uma variável gera a soma de dois termos produtos (10.11), aplicada a duas variáveis gera a soma de quatro termos produtos (10.13). É possível se realizar a expansão, de uma única vez, de um número arbitrário k de variáveis, gerando no processo, a soma de 2^k termos produto.

Além disso, se a expansão for aplicada sucessivamente à função, variável por variável se chega à forma canônica SDP (soma de mintermos) da função.

10.3.2 Forma produto de somas

A forma produto de somas da expansão de Shannon não é tão utilizada como a forma soma de produtos. Ela é definida pela expressão (10.14) e enunciada no teorema:

Teorema 10.3.2 (Teorema da expansão/decomposição de função na forma produto de somas)

$$f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) = (x_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 0)) \cdot (\bar{x}_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, 1)). \quad (10.14)$$

Além disso, qualquer uma das variáveis x_k , $k \in \{0, 1, \dots, n-1\}$ pode ser escolhida para realizar a expansão mostrada na expressão.

Aplicando o teorema novamente em (10.14), expandindo-se a variável x_1 , chega-se a expressão para duas variáveis:

$$\begin{aligned} f(x_{n-1}, x_{n-2}, \dots, x_2, x_1, x_0) &= (x_1 + x_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, 0, 0)) \cdot \\ &\quad \cdot (x_1 + \bar{x}_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, 0, 1)) \cdot \\ &\quad \cdot (\bar{x}_1 + x_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, 1, 0)) \cdot \\ &\quad \cdot (\bar{x}_1 + \bar{x}_0 + f(x_{n-1}, x_{n-2}, \dots, x_2, 1, 1)). \end{aligned} \quad (10.15)$$

■ **Exemplo 10.1** (Expansão de Shannon)

O teorema da expansão de Shannon pode ser usado conjuntamente com multiplexadores para se encontrar uma implementação em *hardware* de uma função de chaveamento combinacional. Essa implementação pode ser obtida aplicando a expansão se Shannon sucessivamente às variáveis da função, uma após a outra até restarem apenas constantes ou as próprias variáveis da função. A questão aqui é que a sequência de variáveis escolhida para a aplicação da expansão influencia no número de multiplexadores necessários para a implementação da função.

Neste exemplo, é aplicada a técnica da expansão de Shannon à função de chaveamento

$$f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0) \quad (10.16)$$

em duas sequências diferentes:

1. Sequência x_0, x_1, x_2, x_3
2. Sequência x_1, x_0, x_2, x_3 .

Ao final é feita uma comparação das implementações finais obtidas usando cada uma das sequências.

Sequência de aplicação x_0, x_1, x_2, x_3

Aplicando a expansão a função e expandindo-se x_0 :

$$\begin{aligned} f(x_3, x_2, x_1, x_0) &= x_3 \cdot (x_1 + x_2 \cdot x_0) \\ &= \bar{x}_0 \cdot [x_3 \cdot x_1] + x_0 \cdot [x_3 \cdot (x_1 + x_2)]. \end{aligned} \quad (10.17)$$

Esse primeira aplicação da expansão pode ser implementada em *hardware* usando um mux-2x1 como mostrado na Fig. 10.8.

Se faz necessário agora continuar o processo aplicando a expansão de Shannon às funções $f(x_3, x_2, x_1, 0) = x_3 \cdot x_1$ e $f(x_3, x_2, x_1, 1) = x_3 \cdot (x_1 + x_2)$ expandir a próxima variável da sequência, x_1 . Assim, obtem-se:

$$\begin{aligned} f(x_3, x_2, x_1, 0) &= x_3 \cdot x_1 \\ &= \bar{x}_1 \cdot [0] + x_1 \cdot [x_3] \end{aligned} \quad (10.18)$$

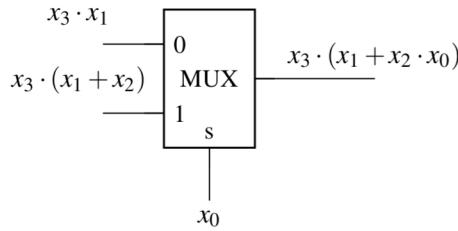


Figura 10.8: Implementação parcial da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão da variável x_0 .

e

$$\begin{aligned} f(x_3, x_2, x_1, 1) &= x_3 \cdot (x_1 + x_2) \\ &= \bar{x}_1 \cdot [x_3 \cdot x_2] + x_1 \cdot [x_3] \end{aligned} \quad (10.19)$$

Implementando-se $f(x_3, x_2, x_1, 0)$ e $f(x_3, x_2, x_1, 1)$ com dois novos mux-2x1 e os conectando, respectivamente, às entradas rotuladas com 0 e 1 do mux-2x1 usado para implementar a expansão de em x_0 obtém-se a implementação mostrada na Fig. 10.9.

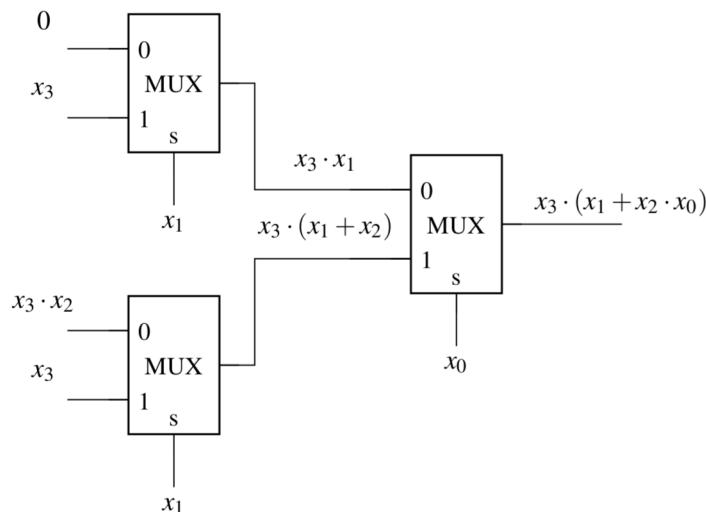


Figura 10.9: Implementação parcial da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão das variáveis x_0 e x_1 .

Percebe-se na Fig. 10.9 que em todas as entradas dos multiplexadores estão as variáveis das funções, exceto em uma delas, a entrada 0 de um dos multiplexadores que está com a função $f(x_3, x_2, 0, 1) = x_3 \cdot x_2$. Aplicando a expansão de Shannon a essa função, expandindo a variável x_2 :

$$\begin{aligned} f(x_3, x_2, 0, 1) &= x_3 \cdot x_2 \\ &= \bar{x}_2 \cdot [0] + x_2 \cdot [x_3] \end{aligned} \quad (10.20)$$

Implementando-se $f(x_3, x_2, 0, 1)$ com um novo mux-2x1 e o conectando à entrada rotulada com 0 do mux-2x1 usado para implementar a expansão de $f(x_3, x_2, x_1, 1)$ em x_1 obtém-se a implementação

mostrada na Fig. 10.10. Na Fig. 10.10 todas as entradas dos multiplexadores recebem apenas variá-

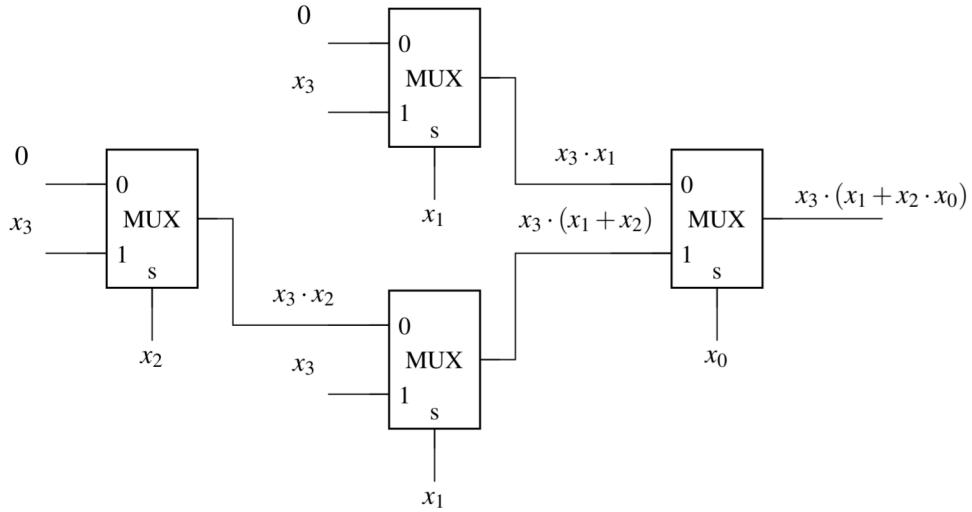


Figura 10.10: Implementação final da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão das variáveis x_0 , x_1 e x_2 .

veis da função ou constantes. Sendo assim a rede de mux-2x1 mostrada na Fig. 10.10 implementa a função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$

Sequência de aplicação x_1, x_0, x_2, x_3 Aplicando a mesma metodologia usada para a sequência x_0, x_1, x_2, x_3 é possível montar o passo a passo da expansão da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ na sequencia x_0, x_1, x_2, x_3 . Os cálculos das expansões ficam a cargo do leitor. Os resultados parciais das expansões estão mostradas na Figs. 10.11, 10.12 e 10.13: A Fig. 10.11 é o resultado da expansão da variável x_1 , a Fig. 10.12 é o resultado da expansão da variável x_0 e a Fig. 10.13 é o resultado da expansão da variável x_2 .

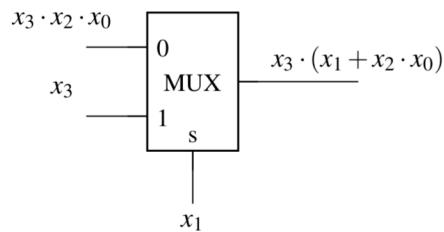


Figura 10.11: Implementação parcial da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão da variável x_1 .

A conclusão é que ao se implementar a $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ usando a expansão de Shannon e mux-4x1 são encontradas implementação diferentes de acordo com a ordem com que as variáveis são expandidas. Ao se considerar a ordem de expansão de x_0, x_1, x_2, x_3 obtém-se a implementação mostrada na Fig. 10.10, na qual são necessários 4 mux-2x1. Por outro lado, ao se considerar a ordem de expansão de x_1, x_0, x_2, x_3 , obtém-se a implementação mostrada na Fig. 10.13, na qual são necessários 3 mux-2x1.

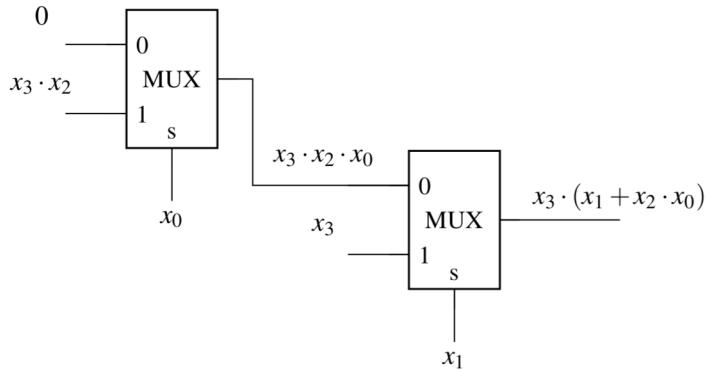


Figura 10.12: Implementação parcial da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão das variáveis x_1 e x_0 .

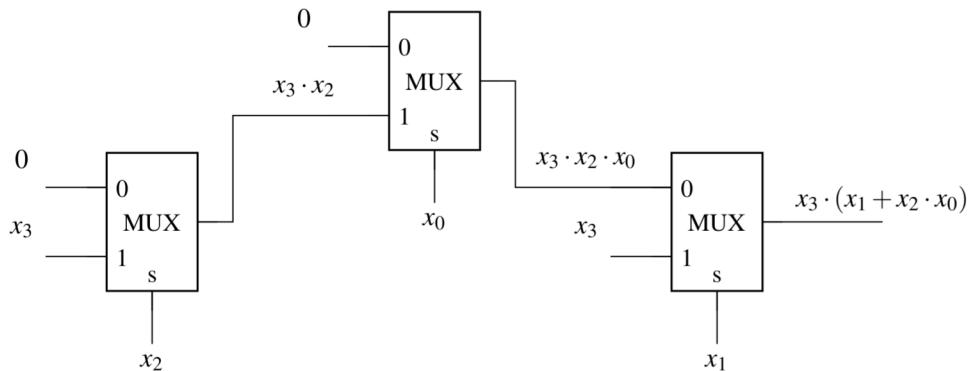


Figura 10.13: Implementação final da função $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_1 + x_2 \cdot x_0)$ do Exemplo 10.1 após a expansão das variáveis x_1 , x_0 e x_2 .

10.4 Problemas propostos

Problema 10.1 Aplique a expansão de Shannon e encontre as implementações das funções listadas usando apenas mux-2x1.

- a) $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_0 + x_2 \cdot x_1)$
- b) $f(x_3, x_2, x_1, x_0) = (x_0 + x_2 \cdot x_1) \cdot (x_1 + x_0 \cdot x_2)$
- c) $f(x_2, x_1, x_0) = \text{Conjunto-UM}(0, 1, 2, 6)$
- d) $f(x_2, x_1, x_0) = \text{Conjunto-UM}(0, 1, 3, 5, 6)$
- e) $f(x_3, x_2, x_1, x_0) = \text{Conjunto-UM}(1, 2, 5, 9, 13, 15)$
- f) $f(x_3, x_2, x_1, x_0) = \text{Conjunto-UM}(4, 5, 7, 9, 10, 12)$

Problema 10.2 Aplique a expansão de Shannon e encontre as implementações das funções listadas usando apenas mux-4x1.

- a) $f(x_3, x_2, x_1, x_0) = x_3 \cdot (x_0 + x_2 \cdot x_1)$

- b) $f(x_3, x_2, x_1, x_0) = (x_0 + x_2 \cdot x_1) \cdot (x_1 + x_0 \cdot x_2)$
- c) $f(x_2, x_1, x_0) = \text{Conjunto-UM}(0, 1, 2, 6)$
- d) $f(x_2, x_1, x_0) = \text{Conjunto-UM}(0, 1, 3, 5, 6)$
- e) $f(x_3, x_2, x_1, x_0) = \text{Conjunto-UM}(1, 2, 5, 9, 13, 15)$
- f) $f(x_3, x_2, x_1, x_0) = \text{Conjunto-UM}(4, 5, 7, 9, 10, 12)$

Problema 10.3 No capítulo foram mostradas as fórmulas para a expansão de Shannon na forma SDP que consideram a expansão de um única variável e a expansão de duas variáveis. Encontre a expressão dessa expansão para três variáveis. Como seria o mux capaz de implementar em *hardware* essa expansão?

Problema 10.4 Um engenheiro identificou a necessidade de implementar um multiplexador 8x1 em seu projeto. Ele percebeu que no seu estoque só há mux-2x1. Sugira para ele uma forma de implementar o multiplexador 8x1 apenas usando mux-2x1.

Problema 10.5 Considere um comutador de sinais S que possui duas entradas (x_0 e x_1), duas saídas (z_0 e z_1) e dois *bits* de controle (c_0 e c_1). As entradas de controle c_0 e c_1 controlam como as entradas x_0 e x_1 são comutadas para as saídas z_0 e z_1 . A Tabela 10.5 mostra como esse controle é feito. Implemente o comutador S usando apenas mux-2x1.

Tabela 10.5: Tabela verdade que descreve o sistema considerado no exercício 10.5.

c_1	c_0	z_1	z_0
0	0	x_1	x_0
0	1	x_0	x_0
1	0	x_1	x_1
1	1	x_0	x_1