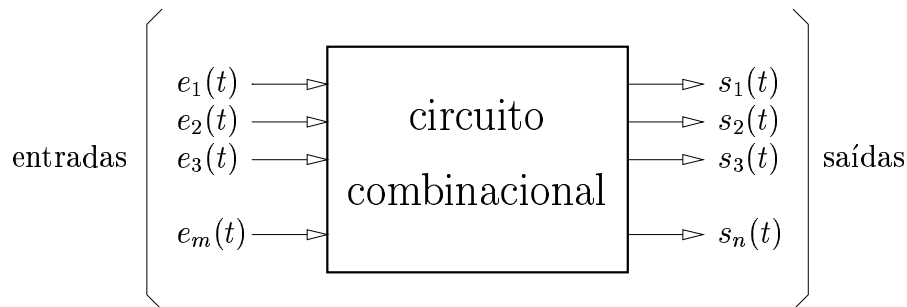


# Capítulo 3

## Circuitos Combinacionais

### 3.1 Introdução

Este capítulo é dedicado ao estudo e projeto de sistemas lógicos combinacionais. Um circuito lógico é dito *combinacional* quando suas saídas em um determinado instante  $t$  são função unicamente de suas entradas naquele instante, ou seja, suas saídas são *combinações lógicas* de suas entradas naquele instante. Isso significa que qualquer modificação nas entradas será imediatamente considerada pelas saídas.



### 3.2 Síntese de circuitos combinacionais

O projeto de um circuito combinacional segue habitualmente os seguintes passos, que devem ser aplicados para cada uma das saídas do circuito:

1. Descrição do comportamento desejado para o sistema, através de uma tabela-verdade enumerando todos os estados possíveis para as entradas e os respectivos valores das saídas.
2. Construção do mapa de Karnaugh da função, a partir da tabela-verdade.
3. Obtenção da forma mínima para a função desejada, a partir dos agrupamentos de termos no mapa de Karnaugh.
4. Construção do circuito indicado pela expressão mínima da função, usando as portas lógicas básicas (*AND*, *OR*, *NOT*). Podem ser efetuadas manipulações sobre a expressão para permitir o uso de portas derivadas (*XOR*) ou universais (*NAND*, *NOR*).

No final deste capítulo veremos casos onde este roteiro não pode ser aplicado diretamente, devido ao grande número de entradas, que torna impraticável a definição de uma tabela-verdade

para o sistema inteiro (por exemplo, 8 entradas geram 256 estados possíveis). Entretanto os circuitos onde isso ocorre podem normalmente ser divididos em sub-sistemas, para os quais o roteiro acima pode ser empregado.

Vejam os o uso do roteiro acima em um exemplo: vamos projetar um detector de números primos de 4 bits, entre  $0000_2$  e  $1111_2$ ). Os números primos de 4 bits são: 1, 2, 3, 5, 7, 11 e 13, e nosso circuito dispõe de quatro entradas (os quatro dígitos do número binário) e uma saída (a indicação de que o número na entrada é primo), o que nos leva a seguinte tabela-verdade:

| $n$ | $A$ | $B$ | $C$ | $D$ | $\mathcal{F}$ |
|-----|-----|-----|-----|-----|---------------|
| 0   | 0   | 0   | 0   | 0   | 0             |
| 1   | 0   | 0   | 0   | 1   | 1             |
| 2   | 0   | 0   | 1   | 0   | 1             |
| 3   | 0   | 0   | 1   | 1   | 1             |
| 4   | 0   | 1   | 0   | 0   | 0             |
| 5   | 0   | 1   | 0   | 1   | 1             |
| 6   | 0   | 1   | 1   | 0   | 0             |
| 7   | 0   | 1   | 1   | 1   | 1             |
| 8   | 1   | 0   | 0   | 0   | 0             |
| 9   | 1   | 0   | 0   | 1   | 0             |
| 10  | 1   | 0   | 1   | 0   | 0             |
| 11  | 1   | 0   | 1   | 1   | 1             |
| 12  | 1   | 1   | 0   | 0   | 0             |
| 13  | 1   | 1   | 0   | 1   | 1             |
| 14  | 1   | 1   | 1   | 0   | 0             |
| 15  | 1   | 1   | 1   | 1   | 0             |

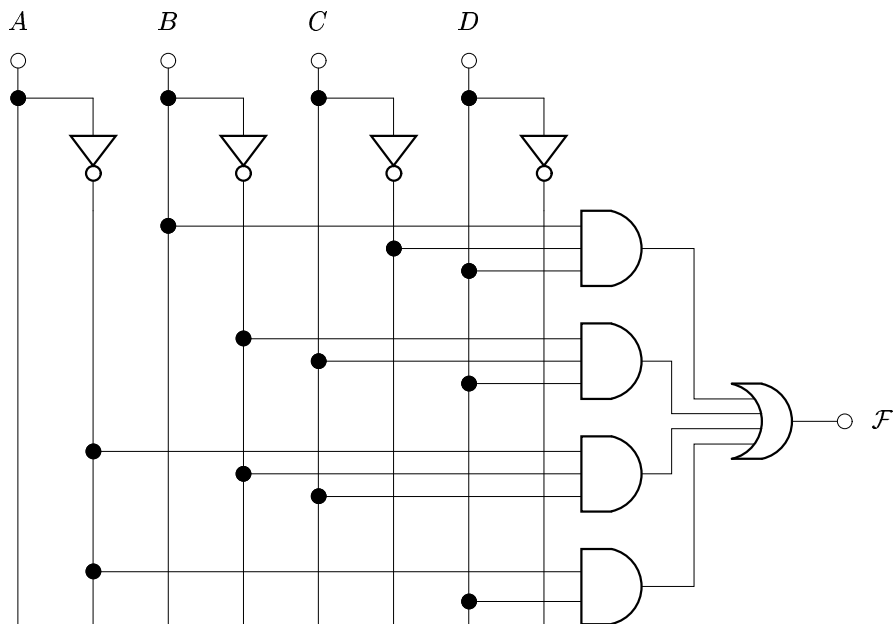
A partir da tabela-verdade acima podemos construir o seguinte mapa de Karnaugh para a função  $\mathcal{F}$ :

| $CD \backslash AB$ | 00 | 01 | 11 | 10 |
|--------------------|----|----|----|----|
| 00                 |    |    |    |    |
| 01                 | 1  | 1  | 1  |    |
| 11                 | 1  | 1  |    | 1  |
| 10                 | 1  |    |    |    |

A expressão que pode ser obtida dos agrupamentos do mapa de Karnaugh tem a forma:

$$\mathcal{F} = \overline{B}\overline{C}D + \overline{B}CD + \overline{A}\overline{B}C + \overline{A}D$$

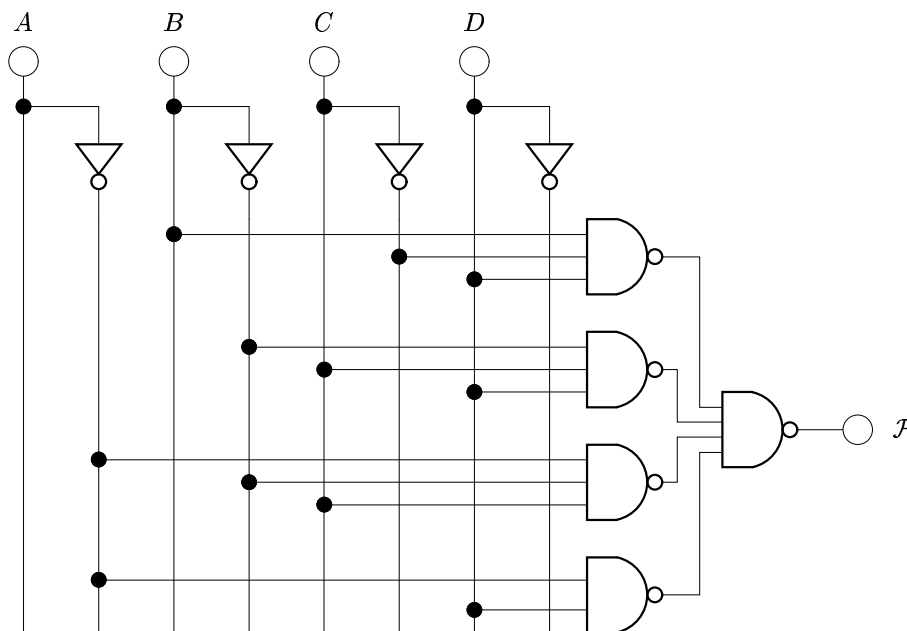
Podemos construir o circuito indicado pela expressão acima usando somente as portas lógicas básicas. Cada termo (produto) da expressão é implementado por uma porta *AND*, e a soma dos termos é implementada por uma porta *OR*, o que nos dá o seguinte circuito:



Podemos empregar os teoremas de Morgan ( $\overline{A + B + C} = \overline{A} \overline{B} \overline{C}$ ) para transformar a função sob a forma de uma soma de produtos em um produto de complementos, eliminando as somas (e a porta *OR*):

$$\begin{aligned}
 \mathcal{F} &= \overline{B}CD + \overline{B}CD + \overline{A}BC + AD \\
 &= \overline{\overline{\overline{B}CD + \overline{B}CD + \overline{A}BC + AD}} \\
 &= \overline{\overline{B}CD} \cdot \overline{\overline{B}CD} \cdot \overline{\overline{A}BC} \cdot \overline{AD}
 \end{aligned}$$

Desta forma a implementação do circuito passa a empregar somente portas *NAND*:



Embora implementado de outra forma, este circuito executa exatamente a mesma função que o anterior. Esta forma de implementação, empregando somente portas *NAND*, é chamada “lógica *NAND-NAND*”.

### 3.3 Conversores de códigos

Os circuitos conversores de códigos se destinam à conversão entre diferentes sistemas de codificação de números binários, como o Gray, o BCD, o 7 segmentos, etc. A maioria destes circuitos se encontra disponível em circuitos integrados comerciais. Vejamos como exemplo a implementação de um conversor de números de 4 bits em código Gray para binário. A tabela-verdade da função de conversão possui quatro entradas (os 4 dígitos do número em código Gray) e quatro saídas (os quatro dígitos do código binário):

| $n$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| 0   | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 1   | 0     | 0     | 0     | 1     | 0     | 0     | 0     | 1     |
| 2   | 0     | 0     | 1     | 1     | 0     | 0     | 1     | 0     |
| 3   | 0     | 0     | 1     | 0     | 0     | 0     | 1     | 1     |
| 4   | 0     | 1     | 1     | 0     | 0     | 1     | 0     | 0     |
| 5   | 0     | 1     | 1     | 1     | 0     | 1     | 0     | 1     |
| 6   | 0     | 1     | 0     | 1     | 0     | 1     | 1     | 0     |
| 7   | 0     | 1     | 0     | 0     | 0     | 1     | 1     | 1     |
| 8   | 1     | 1     | 0     | 0     | 1     | 0     | 0     | 0     |
| 9   | 1     | 1     | 0     | 1     | 1     | 0     | 0     | 1     |
| 10  | 1     | 1     | 1     | 1     | 1     | 0     | 1     | 0     |
| 11  | 1     | 1     | 1     | 0     | 1     | 0     | 1     | 1     |
| 12  | 1     | 0     | 1     | 0     | 1     | 1     | 0     | 0     |
| 13  | 1     | 0     | 1     | 1     | 1     | 1     | 0     | 1     |
| 14  | 1     | 0     | 0     | 1     | 1     | 1     | 1     | 0     |
| 15  | 1     | 0     | 0     | 0     | 1     | 1     | 1     | 1     |

Com base na tabela-verdade podemos construir os mapas de Karnaugh para cada uma das saídas  $B_3 \dots B_0$ . Começando com  $B_3$  temos:

| $G_1G_0$ | $G_3G_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    |    | 1  | 1  |
| 01       |          |    |    | 1  | 1  |
| 11       |          |    |    | 1  | 1  |
| 10       |          |    |    | 1  | 1  |

O que nos leva à expressão  $B_3 = G_3$ . Para  $B_2$  teremos:

| $G_1G_0$ | $G_3G_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    | 1  |    | 1  |
| 01       |          |    | 1  |    | 1  |
| 11       |          |    | 1  |    | 1  |
| 10       |          |    | 1  |    | 1  |

O que nos leva a  $B_2 = \overline{G_3}G_2 + G_3\overline{G_2} = G_3 \oplus G_2$ . Para  $B_1$  teremos:

| $G_1G_0$ | $G_3G_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    | 1  |    | 1  |
| 01       |          |    | 1  |    | 1  |
| 11       |          | 1  |    | 1  |    |
| 10       |          | 1  |    | 1  |    |

$$\begin{aligned}
 B_1 &= \overline{G_3}\overline{G_2}G_1 + \overline{G_3}G_2\overline{G_1} + G_3\overline{G_2}\overline{G_1} + G_3G_2G_1 \\
 &= \overline{G_3}(\overline{G_2}G_1 + G_2\overline{G_1}) + G_3(\overline{G_2}\overline{G_1} + G_2G_1) \\
 &= \overline{G_3}(G_2 \oplus G_1) + G_3(\overline{G_2 \oplus G_1}) \\
 &= G_3 \oplus (G_2 \oplus G_1) \\
 B_1 &= G_3 \oplus G_2 \oplus G_1
 \end{aligned}$$

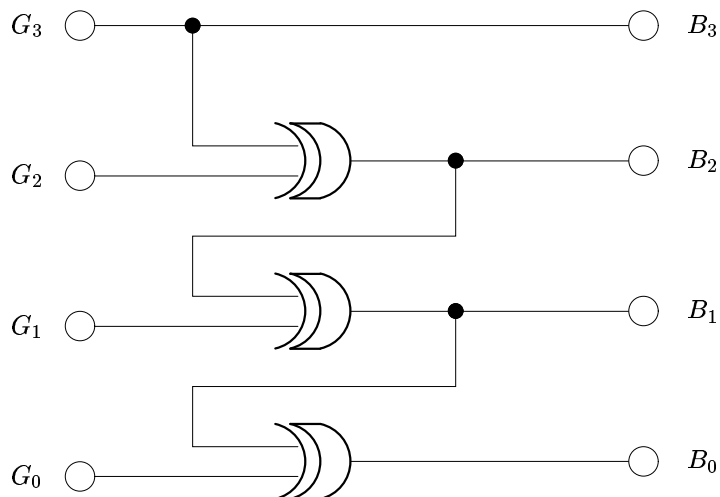
Finalmente para  $B_0$  teremos:

| $G_1G_0$ | $G_3G_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    | 1  |    | 1  |
| 01       |          | 1  |    | 1  |    |
| 11       |          |    | 1  |    | 1  |
| 10       |          | 1  |    | 1  |    |

O que nos levará a  $B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$  (este resultado pode ser encontrado de maneira análoga ao obtido para  $B_1$ ). Desta forma temos o seguinte conjunto de funções que devem ser implementadas:

$$\begin{aligned}
 B_3 &= G_3 \\
 B_2 &= G_3 \oplus G_2 \\
 B_1 &= G_3 \oplus G_2 \oplus G_1 = B_2 \oplus G_1 \\
 B_0 &= G_3 \oplus G_2 \oplus G_1 \oplus G_0 = B_1 \oplus G_0
 \end{aligned}$$

A implementação mais simples e direta destas expressões é dada pelo circuito abaixo:



Veremos agora em outro exemplo a conversão de binário para complemento 2, com palavras binárias de 4 bits ( $B_3B_2B_1B_0$ ) e seus respectivos complementos ( $C_3C_2C_1C_0$ ). A tabela-verdade assume a seguinte forma:

| $n$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ | $C_3$ | $C_2$ | $C_1$ | $C_0$ |
|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| 0   | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |
| 1   | 0     | 0     | 0     | 1     | 1     | 1     | 1     | 1     |
| 2   | 0     | 0     | 1     | 0     | 1     | 1     | 1     | 0     |
| 3   | 0     | 0     | 1     | 1     | 1     | 1     | 0     | 1     |
| 4   | 0     | 1     | 0     | 0     | 1     | 1     | 0     | 0     |
| 5   | 0     | 1     | 0     | 1     | 1     | 0     | 1     | 1     |
| 6   | 0     | 1     | 1     | 0     | 1     | 0     | 1     | 0     |
| 7   | 0     | 1     | 1     | 1     | 1     | 0     | 0     | 1     |
| 8   | 1     | 0     | 0     | 0     | 1     | 0     | 0     | 0     |
| 9   | 1     | 0     | 0     | 1     | 0     | 1     | 1     | 1     |
| 10  | 1     | 0     | 1     | 0     | 0     | 1     | 1     | 0     |
| 11  | 1     | 0     | 1     | 1     | 0     | 1     | 0     | 1     |
| 12  | 1     | 1     | 0     | 0     | 0     | 1     | 0     | 0     |
| 13  | 1     | 1     | 0     | 1     | 0     | 0     | 1     | 1     |
| 14  | 1     | 1     | 1     | 0     | 0     | 0     | 1     | 0     |
| 15  | 1     | 1     | 1     | 1     | 0     | 0     | 0     | 1     |

Com base na tabela-verdade podemos construir os mapas de Karnaugh e deduzir as expressões lógicas para cada uma das saídas  $C_3 \dots C_0$ :

| $B_1B_0$ | $B_3B_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    | 1  |    | 1  |
| 01       |          | 1  | 1  |    |    |
| 11       |          | 1  | 1  |    |    |
| 10       |          | 1  | 1  |    |    |

| $B_1B_0$ | $B_3B_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    | 1  | 1  |    |
| 01       |          | 1  |    |    | 1  |
| 11       |          | 1  |    |    | 1  |
| 10       |          | 1  |    |    | 1  |

$$C_3 = \overline{B_3}B_1 + \overline{B_3}B_0 + \overline{B_3}B_2 + B_3\overline{B_2}\overline{B_1}\overline{B_0}$$

$$C_2 = \overline{B_2}B_1 + \overline{B_2}B_0 + B_2\overline{B_1}\overline{B_0}$$

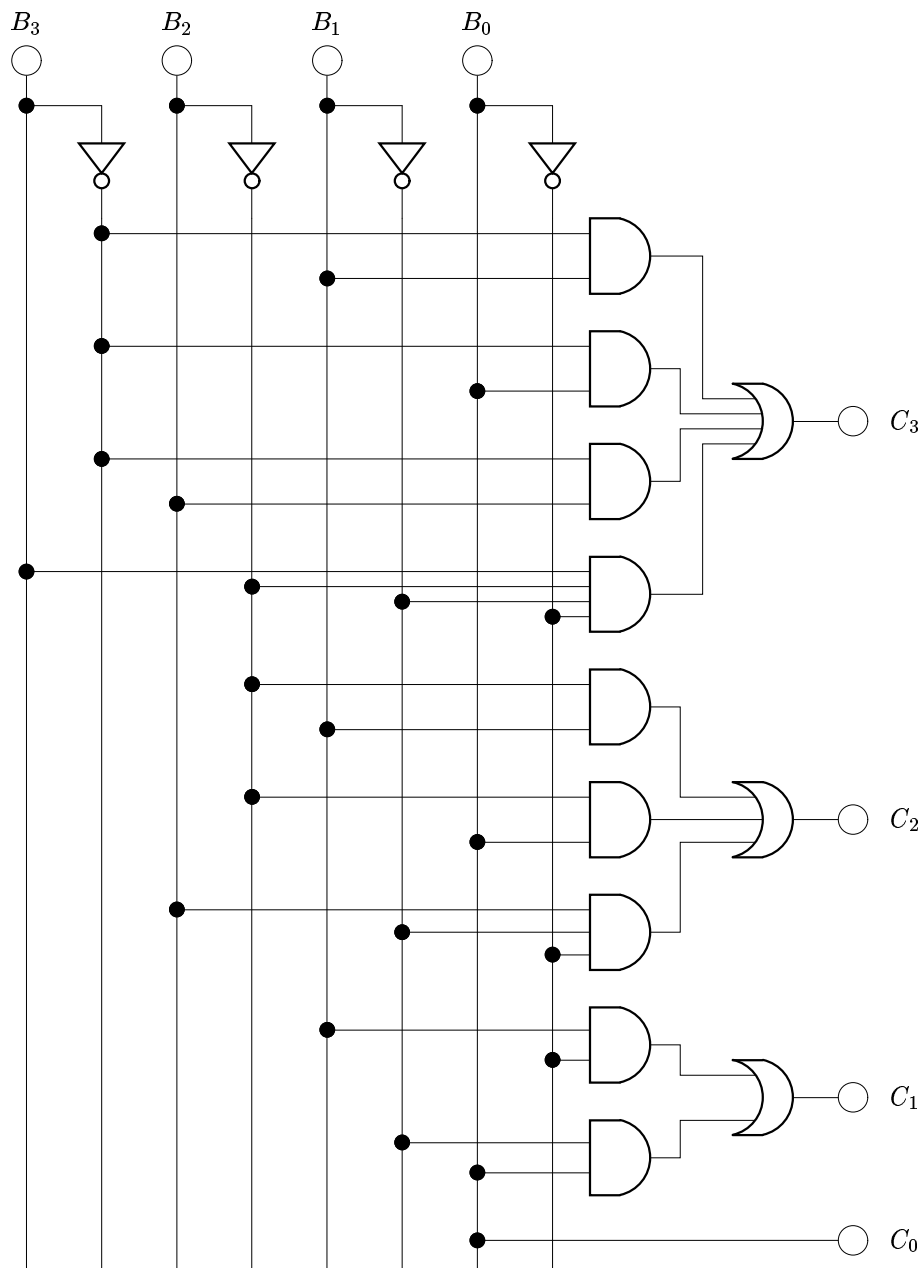
| $B_1B_0$ | $B_3B_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    |    |    |    |
| 01       |          | 1  | 1  | 1  | 1  |
| 11       |          |    |    |    |    |
| 10       |          | 1  | 1  | 1  | 1  |

| $B_1B_0$ | $B_3B_2$ | 00 | 01 | 11 | 10 |
|----------|----------|----|----|----|----|
| 00       |          |    |    |    |    |
| 01       |          | 1  | 1  | 1  | 1  |
| 11       |          | 1  | 1  | 1  | 1  |
| 10       |          |    |    |    |    |

$$C_1 = \overline{B_1}B_0 + B_1\overline{B_0}$$

$$C_0 = B_0$$

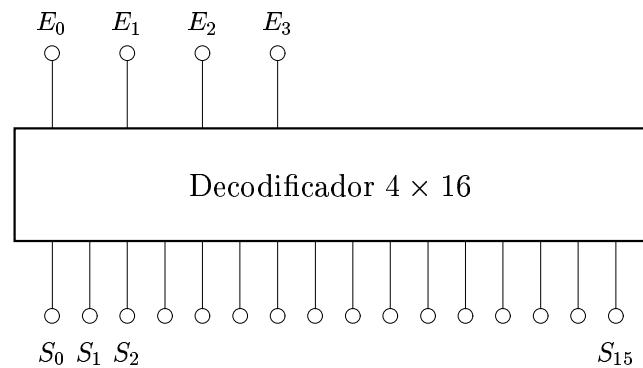
As expressões acima nos permitem implementar o circuito como apresentado na figura a seguir, usando as portas básicas. Obviamente outras implementações são possíveis, manipulando adequadamente as expressões acima.



Além dos circuitos apresentados acima, podemos ter codificadores de binário para complemento 1 e vice-versa, de BCD para 7 segmentos, etc. que podem ser facilmente projetados pelo leitor usando a técnica apresentada acima.

### 3.4 Codificadores e decodificadores

Um decodificador é um dispositivo com  $n$  entradas e  $2^n$  saídas. Para cada valor da entrada uma única saída é verdadeira e as demais são falsas. Normalmente representamos um decodificador através de um bloco, como por exemplo para o decodificador com 4 entradas e 16 saídas:



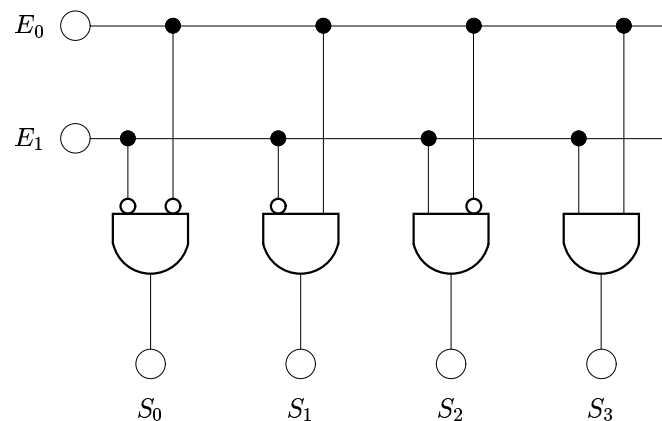
A tabela-verdade do decodificador  $4 \times 16$  acima assume esta forma:

| $E_3$    | $E_2$    | $E_1$    | $E_0$    | $S_0$    | $S_1$    | $S_2$    | $S_3$    | $\dots$  | $S_{15}$ |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 0        | 0        | 0        | 0        | 1        | 0        | 0        | 0        | $\dots$  | 0        |
| 0        | 0        | 0        | 1        | 0        | 1        | 0        | 0        | $\dots$  | 0        |
| 0        | 0        | 1        | 0        | 0        | 0        | 1        | 0        | $\dots$  | 0        |
| 0        | 0        | 1        | 1        | 0        | 0        | 0        | 1        | $\dots$  | 0        |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| 1        | 1        | 1        | 1        | 0        | 0        | 0        | 0        | $\dots$  | 1        |

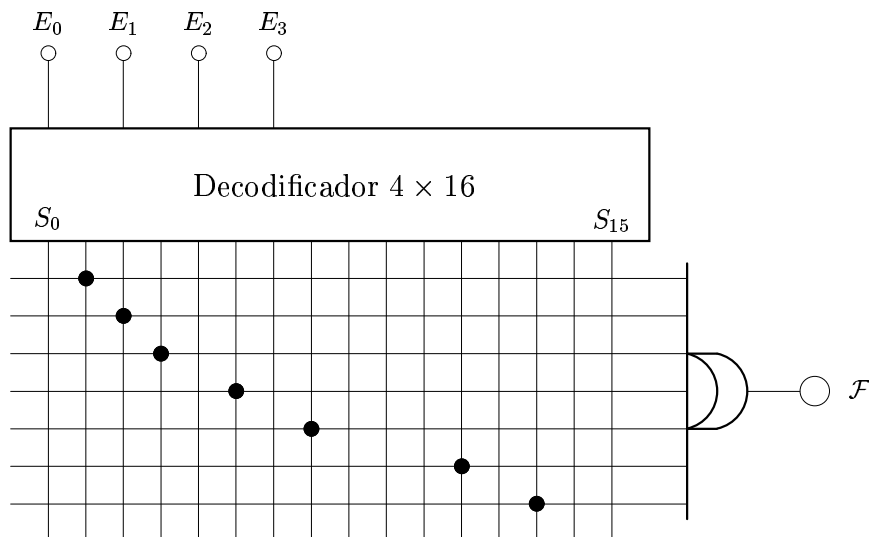
Por exemplo, para o decodificador de  $2 \times 4$  linhas teremos a seguinte tabela-verdade:

| $E_1$ | $E_0$ | $S_0$ | $S_1$ | $S_2$ | $S_3$ |
|-------|-------|-------|-------|-------|-------|
| 0     | 0     | 1     | 0     | 0     | 0     |
| 0     | 1     | 0     | 1     | 0     | 0     |
| 1     | 0     | 0     | 0     | 1     | 0     |
| 1     | 1     | 0     | 0     | 0     | 1     |

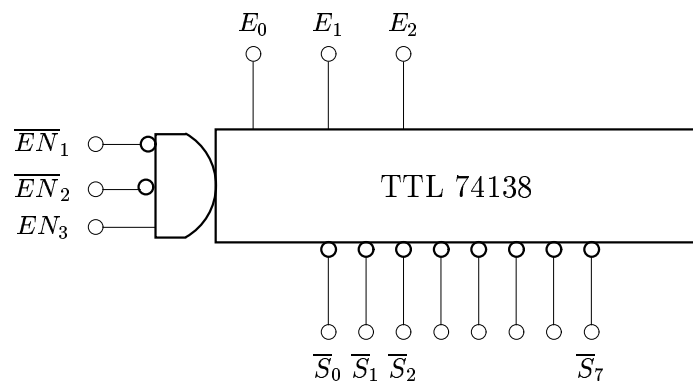
Que pode ser facilmente implementada através do circuito abaixo (note que usamos uma nova notação para as entradas complementadas):



Um decodificador pode ser útil na implementação de funções lógicas simples, como alguns dos circuitos vistos anteriormente. Por exemplo, eis uma implementação do detector de números primos de 4 bits usando um decodificador  $4 \times 16$ :



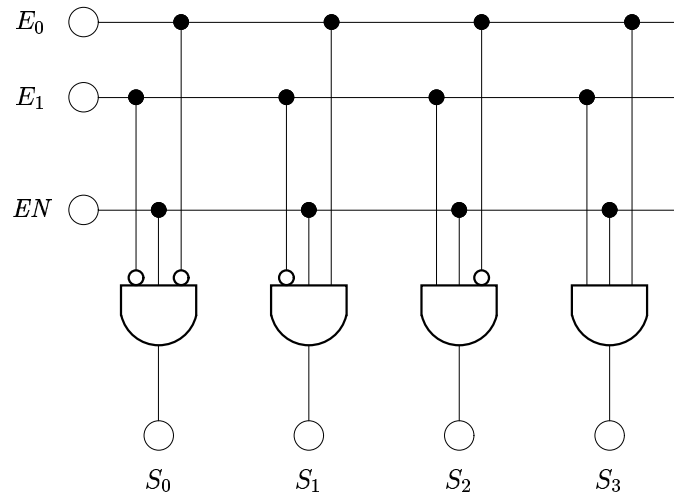
Devido às suas características construtivas, nas principais famílias lógicas uma saída no estado baixo (0 ou L(ow)) drena uma corrente significativamente maior que uma saída em estado alto (1 ou H(igh)). Para uma porta TTL convencional, a corrente em uma saída em estado baixo é da ordem de 1.6 mA, enquanto uma saída em estado alto drena apenas cerca de 200  $\mu$ A, ou seja, uma corrente 8 vezes menor. O comportamento esperado para um decodificador com  $2^n$  saídas é ter a saída ativa em estado alto e as demais  $2^n - 1$  saídas em estado baixo, sendo por isso chamado de *ativo alto*. Entretanto, para diminuir o consumo de energia e a conseqüente dissipação de calor nos circuitos integrados, a maioria dos decodificadores comerciais emprega uma lógica de *ativo baixo*, na qual a saída ativa se encontra em estado baixo (0) e as demais em estado alto (1). O decodificador comercial integrado TTL 74138, de  $3 \times 8$  vias, é então representado por:



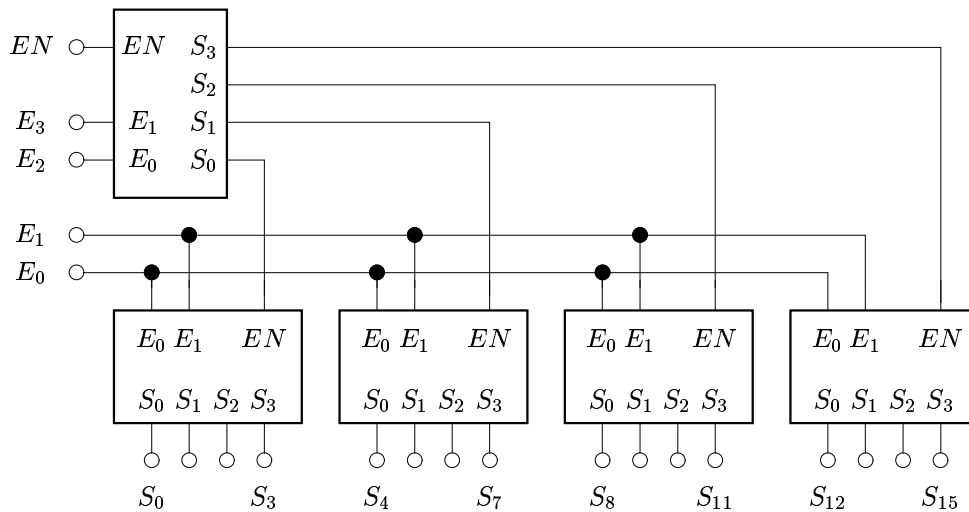
Além das entradas que definem a saída ativa, um decodificador pode ter uma ou mais entradas do tipo *strobe* ou *enable/disable*, como as entradas  $\overline{EN}_1$ ,  $\overline{EN}_2$  e  $EN_3$  do decodificador  $3 \times 8$  acima. A função dessas entradas é inibir a saída ativa do decodificador: na situação acima, enquanto  $\overline{EN}_1 \overline{EN}_2 EN_3$  for falso, nenhuma saída será ativada, não importando o estado das entradas  $E_i$ . Considerando um decodificador  $2 \times 4$  com uma entrada *enable* teremos a seguinte tabela-verdade:

| $EN$ | $E_1$ | $E_0$ | $S_0$ | $S_1$ | $S_2$ | $S_3$ |
|------|-------|-------|-------|-------|-------|-------|
| 0    | X     | X     | 0     | 0     | 0     | 0     |
| 1    | 0     | 0     | 1     | 0     | 0     | 0     |
| 1    | 0     | 1     | 0     | 1     | 0     | 0     |
| 1    | 1     | 0     | 0     | 0     | 1     | 0     |
| 1    | 1     | 1     | 0     | 0     | 0     | 1     |

Este comportamento pode ser facilmente implementado através do seguinte circuito:



As entradas do tipo *enable* podem ser usadas para conectar decodificadores em cascata, permitindo assim a construção de decodificadores maiores. O exemplo abaixo mostra a implementação de um decodificador  $4 \times 16$  usando 5 decodificadores  $2 \times 4$ :



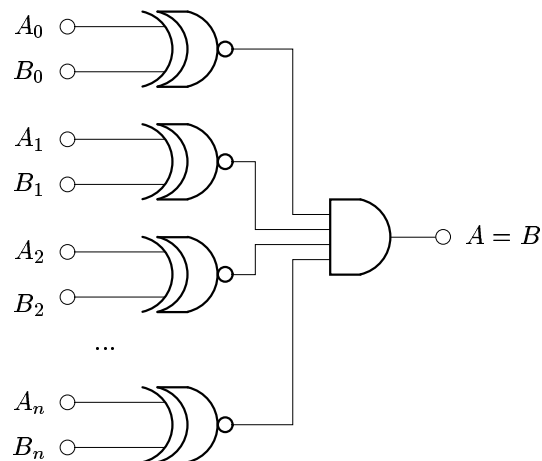
Ao contrário dos decodificadores, os codificadores convertem entre  $2^n$  entradas e uma saída com  $n$  bits. O circuito integrado 74148 é um codificador de  $3 \times 8$  vias, funcionando em ativo baixo (sua entrada ativa está em estado 0). Sua tabela-verdade é dada por:

| $\overline{EN}$ | $\overline{E}_0$ | $\overline{E}_1$ | $\overline{E}_2$ | $\overline{E}_3$ | $\overline{E}_4$ | $\overline{E}_5$ | $\overline{E}_6$ | $\overline{E}_7$ | $\overline{S}_0$ | $\overline{S}_1$ | $\overline{S}_2$ | $\overline{EO}$ | $\overline{GS}$ |
|-----------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|-----------------|-----------------|
| 1               | x                | x                | x                | x                | x                | x                | x                | x                | 1                | 1                | 1                | 0               | x               |
| 0               | x                | x                | x                | x                | x                | x                | x                | 0                | 0                | 0                | 0                | 1               | 0               |
| 0               | x                | x                | x                | x                | x                | x                | 0                | 1                | 0                | 0                | 1                | 1               | 0               |
| 0               | x                | x                | x                | x                | x                | 0                | 1                | 1                | 0                | 1                | 0                | 1               | 0               |
| 0               | x                | x                | x                | x                | 0                | 1                | 1                | 1                | 0                | 1                | 1                | 1               | 0               |
| 0               | x                | x                | x                | 0                | 1                | 1                | 1                | 1                | 1                | 0                | 0                | 1               | 0               |
| 0               | x                | x                | 0                | 1                | 1                | 1                | 1                | 1                | 1                | 0                | 1                | 1               | 0               |
| 0               | x                | 0                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 0                | 1               | 0               |
| 0               | 0                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1               | 0               |
| 0               | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 1                | 0               | 1               |

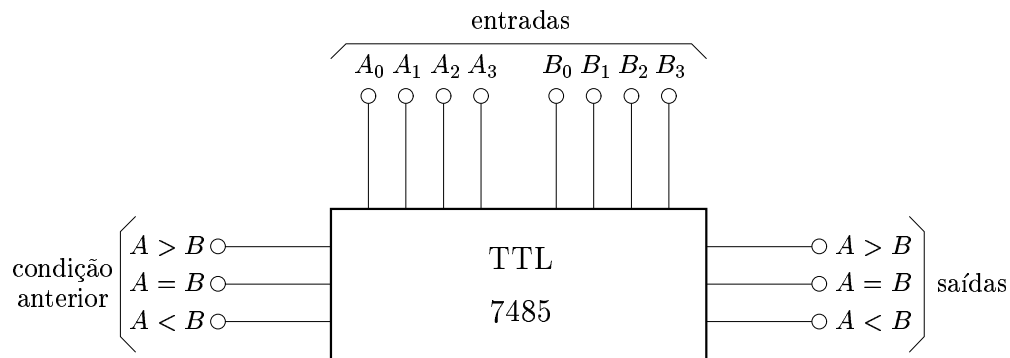
A entrada  $\overline{EN}$  habilita a saída. As saídas complementadas  $S_0 \dots S_2$  indicam o dígito binário correspondente à entrada ativa de número mais elevado (trata-se então de um codificador com prioridade, onde as entradas de número mais elevado tem prioridade). A saída  $\overline{EO}$  é ativada (0) quando não existir entrada ativa, ou caso as saídas estejam desabilitadas ( $\overline{EN} = 1$ ). A saída  $\overline{GS}$  (chamada *group service*) é ativada caso haja alguma entrada ativa, independente da habilitação das saídas.

### 3.5 Comparadores de palavras

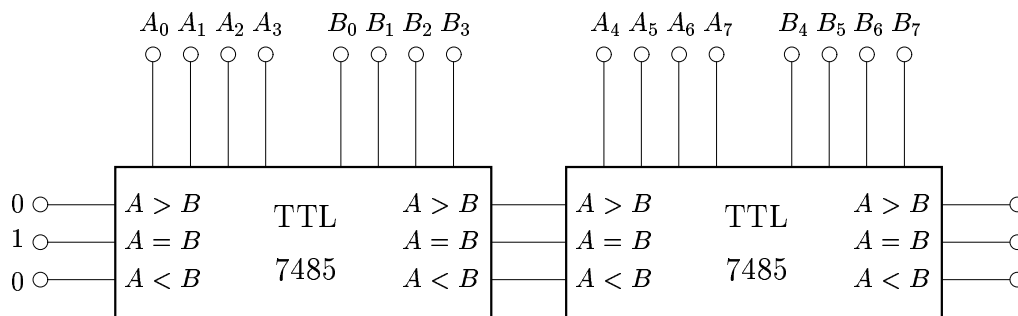
Um circuito comparador simples permite comparar duas palavras de  $n$  bits, indicando quando estas são iguais. Esse tipo de comparador pode ser facilmente implementado através de portas *não-ou-exclusivo* entre os respectivos bits das palavras a comparar ( $A_i \oplus B_i = A_i B_i + \overline{A_i} \overline{B_i}$ ):



Circuitos comparadores mais complexos podem ser construídos para indicar se  $A = B$ ,  $A > B$  ou  $A < B$ . O circuito integrado TTL 7485 é um comparador de palavras de 4 bits com essa possibilidade:

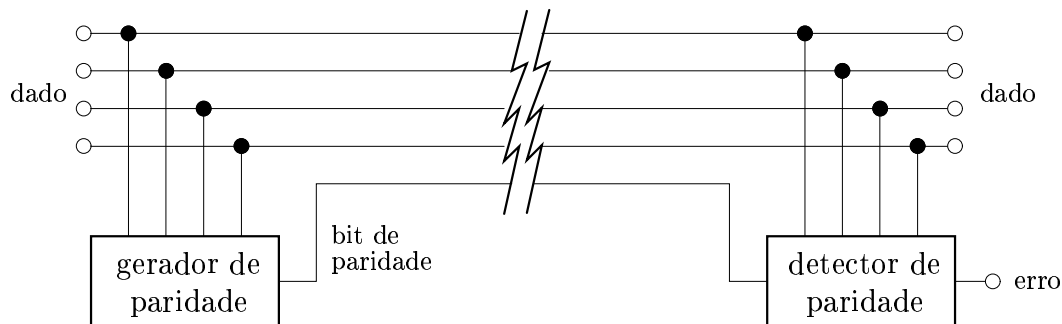


As entradas indicadas como *condição anterior* permitem a conexão de comparadores em cascata para efetuar a comparação de palavras maiores. No circuito do exemplo a seguir podem ser comparadas palavras de 8 bits:

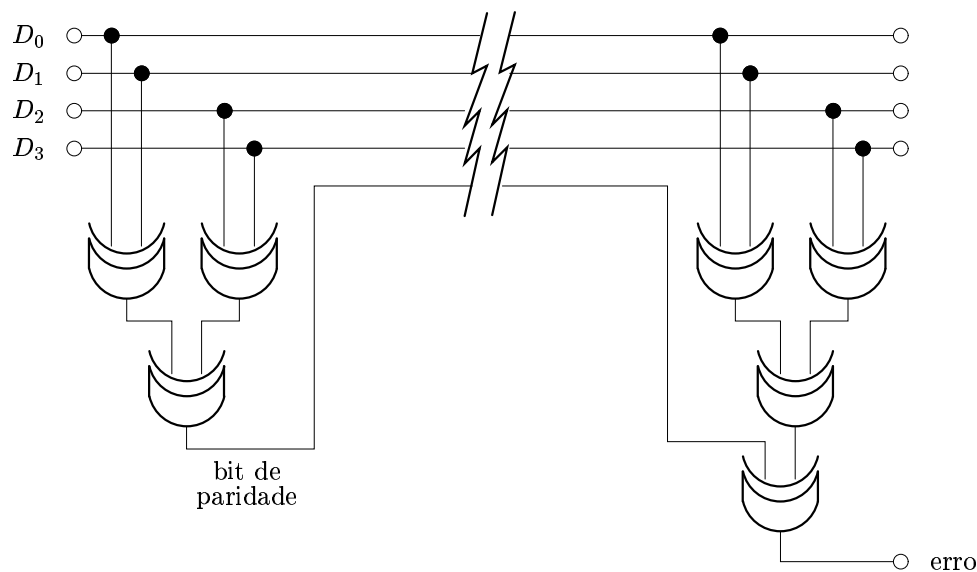


### 3.6 Geradores e detectores de paridade

Os geradores e detectores de paridade são muito úteis em comunicação de dados, onde permitem detectar a presença de erros de transmissão. A técnica básica consiste em associar um bit de paridade ao dado a ser transmitido, indicando se este tem um número par ou ímpar de bits ativos. Ao ser recebido, o dado é testado em relação ao bit de paridade, e eventuais erros podem ser detectados.



Um gerador simples de bit de paridade para palavras de 4 bits e seu respectivo detector são apresentados no diagrama a seguir.

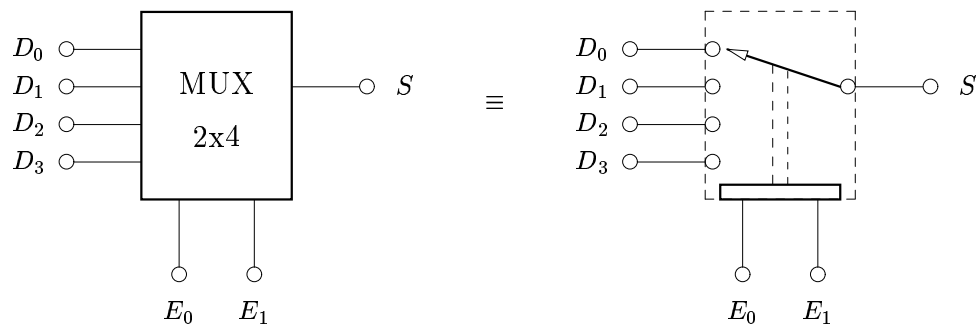


Um circuito é dito de *paridade par*, quando o número total de bits “1”, incluindo o de paridade, é par. Caso contrário, o circuito é dito de *paridade ímpar*. O circuito acima apresentado é de paridade par, pois o bit de paridade vale 0 se o dado possuir um número par de bits ativos, e 1 senão; em conseqüência, o número total de bits ativos é sempre par.

No caso da detecção de erro em um dado, o sistema de recepção de dados pode solicitar que este seja retransmitido, ou tomar outra atitude. Os circuitos de paridade são capazes de detectar erros quando um número ímpar de bits é alterado durante a transmissão. Existem circuitos mais complexos de comunicação de dados capazes de detectar erros de modo mais abrangente e mesmo corrigir erros pequenos, sem necessidade de retransmissão. No entanto esses circuitos necessitam de mais bits de controle associados a cada dado.

### 3.7 Multiplexadores e demultiplexadores

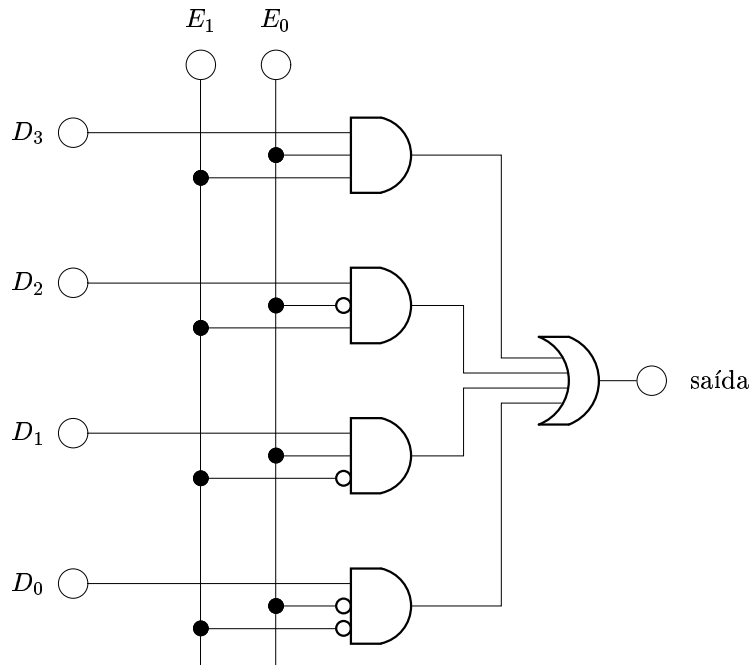
Um multiplexador é um dispositivo com  $2^n$  entradas de dados, uma saída e  $n$  entradas de seleção que definem qual entrada de dados transferir para a saída. Um típico multiplexador de 4 vias é mostrado abaixo:



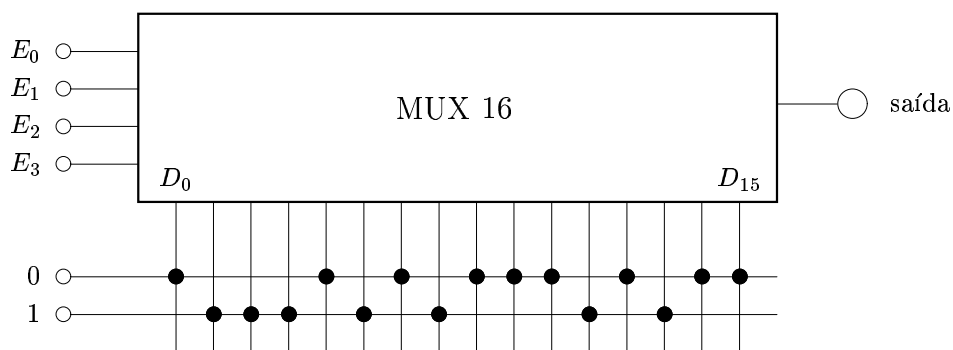
A tabela-verdade do multiplexador acima é dada por:

| $E_1$ | $E_0$ | $S$   |
|-------|-------|-------|
| 0     | 0     | $D_0$ |
| 0     | 1     | $D_1$ |
| 1     | 0     | $D_2$ |
| 1     | 1     | $D_3$ |

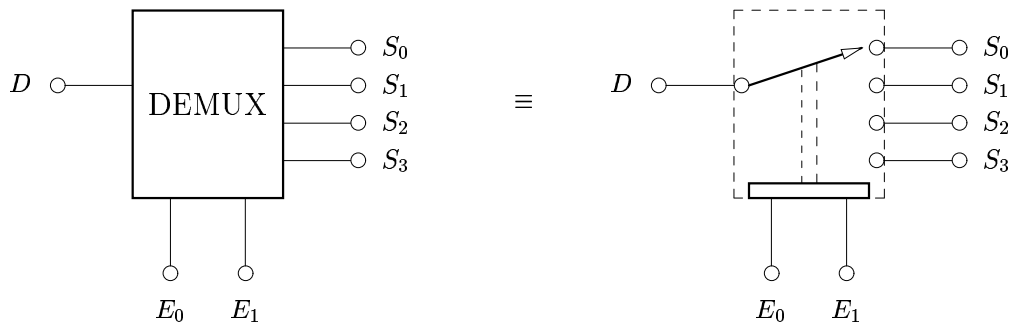
O que nos leva a  $S = \overline{E_1}\overline{E_0}D_0 + \overline{E_1}E_0D_1 + E_1\overline{E_0}D_2 + E_1E_0D_3$ . Esta função pode ser facilmente implementada pelo circuito abaixo:



Assim como os decodificadores, os multiplexadores podem ser usados na implementação de funções lógicas, como por exemplo a detecção de números primos:



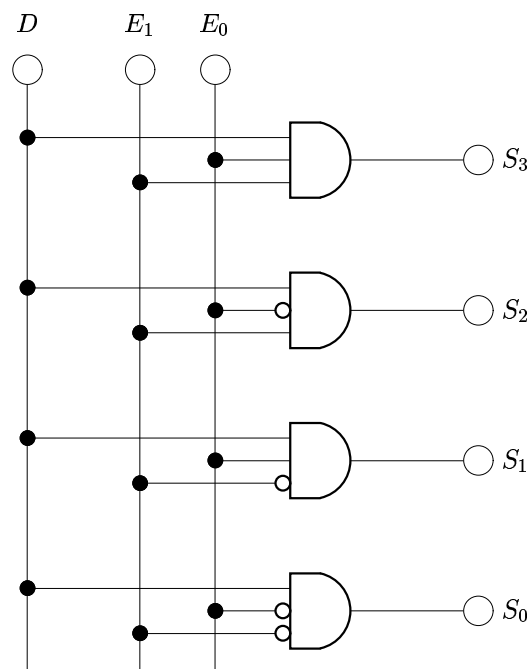
Os demultiplexadores permitem distribuir um dado em sua entrada para uma entre  $2^n$  saídas, escolhida de acordo com as entradas de seleção:



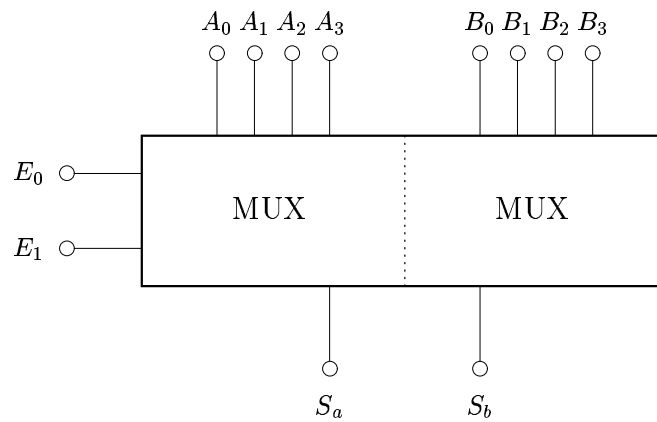
Para um demultiplexador de 4 vias teríamos a seguinte tabela-verdade:

| $E_1$ | $E_0$ | $S_0$ | $S_1$ | $S_2$ | $S_3$ |
|-------|-------|-------|-------|-------|-------|
| 0     | 0     | $D$   | 0     | 0     | 0     |
| 0     | 1     | 0     | $D$   | 0     | 0     |
| 1     | 0     | 0     | 0     | $D$   | 0     |
| 1     | 1     | 0     | 0     | 0     | $D$   |

Esta tabela pode ser implementada pelo seguinte circuito:



Os dispositivos mux/demux comerciais geralmente possuem uma ou mais entradas do tipo *enable*, complementadas ou não, que permitem conectar circuitos em cascata. As principais aplicações desses dispositivos são a seleção e encaminhamento de dados, a conversão série-paralelo de dados, a geração de formas de onda e a síntese de funções lógicas. Um exemplo de circuito integrado comercial é o TTL 74153, que possui dois multiplexadores de 4 vias controlados simultaneamente (as entradas de seleção são comuns a ambos):



### 3.8 Somadores

Os circuitos somadores permitem efetuar operações de soma entre duas palavras de  $n$  bits, levando em conta o “vai-um” (excesso ou *carry*). Para a soma de duas palavras  $A$  e  $B$  de 1 bit teremos a seguinte tabela-verdade, para a soma  $S$  e o excesso  $C$ :

| $A$ | $B$ | $S$ | $C$ |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 1   | 1   | 0   |
| 1   | 0   | 1   | 0   |
| 1   | 1   | 0   | 1   |

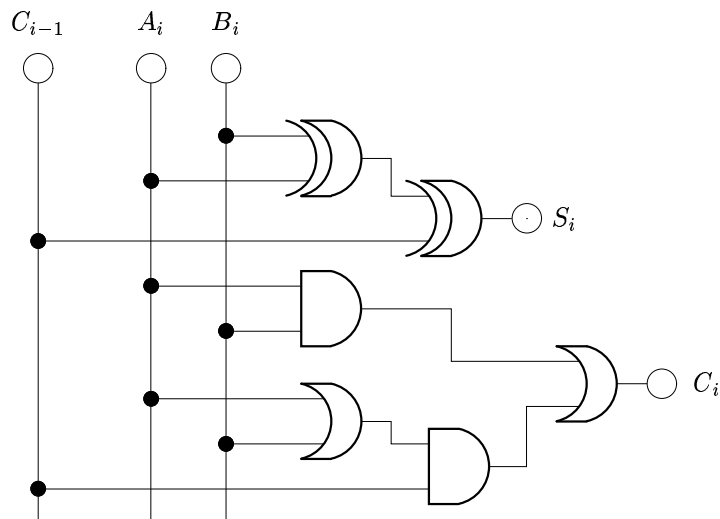
Pode-se então concluir que  $S = A \oplus B$  e  $C = A \cdot B$ . O circuito definido por essas funções é chamado “meio-somador” e sua implementação é trivial, usando apenas duas portas lógicas. Um circuito “somador completo” deve levar em conta o excesso (“vai-um”) de um eventual antecessor, para poder ser associado em cascata a outros somadores e assim implementar somadores para palavras mais longas que 1 bit. A tabela-verdade para um somador completo de 1 bit é dada por:

| $C_{i-1}$ | $A_i$ | $B_i$ | $S_i$ | $C_i$ |
|-----------|-------|-------|-------|-------|
| 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     |

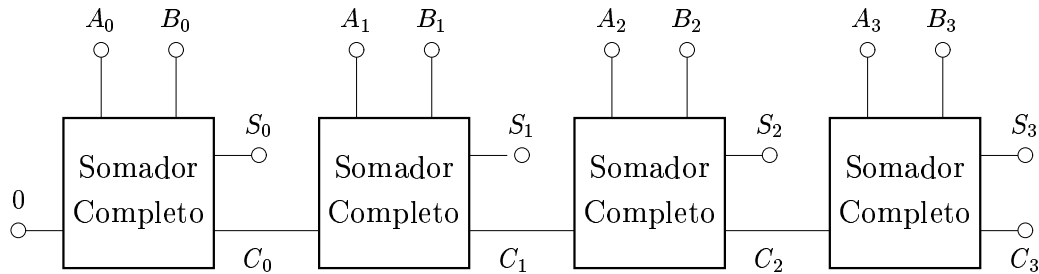
Através dos mapas de Karnaugh obtemos as funções lógicas e sua implementação:

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + C_{i-1} (A_i + B_i)$$



A partir desse módulo somador completo de 1 bit podemos construir somadores mais complexos, para palavras com  $n$  bits. Por exemplo, eis a implementação de um somador para palavras de 4 bits:



O circuito acima é encontrado comercialmente no integrado TTL 7483. Dois somadores deste tipo podem ser associados para constituir um somador de 8 bits.

A construção de subtratores pode ser feita associando somadores a conversores de complemento 1 ou 2. Por exemplo, o circuito TTL 7483 pode ser empregado para implementar um subtrator de palavras positivas de 4 bits, usando o complemento de 1. Para isso é necessário complementar o número  $B$  ( $A - B = A + (-B)$ ) e atribuir 1 ao excesso inicial ( $C_0 = 1$ ). O bit  $S_3$  indicará o sinal da saída, e os números negativos serão apresentados em complemento 1. O circuito abaixo indica a forma de implementação:

