2 Álgebra Booleana e Circuitos Lógicos

Uma álgebra Booleana pode ser definida com um conjunto de operadores e um conjunto de axiomas, que são assumidos verdadeiros sem necessidade de prova.

Em 1854, George Boole introduziu o formalismo que até hoje se usa para o tratamento sistemático da lógica, que é a chamada **Álgebra Booleana**. Em 1938, C. E. Shannon aplicou esta álgebra para mostrar que as propriedades de circuitos elétricos de chaveamento podem ser representadas por uma álgebra Booleana com dois valores.

Diferentemente da álgebra ordinária dos reais, onde as variáveis podem assumir valores no intervalo (-∞;+∞), as variáveis Booleanas só podem assumir um número finito de valores. Em particular, na álgebra Booleana de dois valores, cada variável pode assumir um dentre dois valores possíveis, os quais podem ser denotados por [F,V] (falso ou verdadeiro), [H,L] (high and low) ou ainda [0,1]. Nesta disciplina, adotaremos a notação [0,1], a qual também é utilizada em eletrônica digital. Como o número de valores que cada variável pode assumir é finito (e pequeno), o número de estados que uma função Booleana pode assumir também será finito, o que significa que podemos descrever completamente as funções Booleanas utilizando tabelas. Devido a este fato, uma tabela que descreva uma função Booleana recebe o nome de **tabela verdade,** e nela são listadas todas as combinações de valores que as variáveis de entrada podem assumir e os correspondentes valores da função (saídas).

2.1 Operações Básicas da Álgebra Booleana (ou Álgebra de Chaveamento)

Na álgebra Booleana, existem três operações ou funções básicas. São elas, operação **OU**, operação **E** e **complementação**. Todas as funções Booleanas podem ser representadas em termos destas operações básicas.

2.1.1 Operação OU (Adição Lógica)

Uma definição para a operação **OU**, que também é denominada adição lógica, é:

"A operação **OU** resulta **1** se pelo menos uma das variáveis de entrada vale **1**".

Como uma variável Booleana ou vale **1** ou vale **0**, e como o resultado de uma operação qualquer pode ser encarado como (ou atribuído a) uma variável Booleana, basta que definamos quando a operação vale **1**. Automaticamente, a operação resultará **0** nos demais casos. Assim, pode-se dizer que a operação **OU** resulta **0** somente quando todas as variáveis de entrada valem **0**.

Um símbolo possível para representar a operação **OU** é "+", tal como o símbolo da adição algébrica (dos reais). Porém, como estamos trabalhando com variáveis Booleanas,

sabemos que não se trata da adição algébrica, mas sim da adição lógica. Outro símbolo também encontrado na bibliografia é "$\vee$".

Listando as possibilidades de combinações entre dois valores Booleanos e os respectivos resultados para a operação **OU**, tem-se:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 1 \end{aligned}$$

Note que a operação **OU** só pode ser definida se houver, pelo menos, duas variáveis envolvidas. Ou seja, não é possível realizar a operação sobre somente uma variável. Devido a isso, o operador "+" (**OU**) é dito **binário**.

Nas equações, não costuma-se escrever todas as possibilidades de valores. Apenas adotamos uma letra (ou uma letra com um índice) para designar uma variável Booleana. Com isso, já se sabe que aquela variável pode assumir ou o valor **0** ou o valor **1**. Então, supondo que queiramos demonstar o comportamento da equação A+B (lê-se A ou B), poderíamos fazê-lo utilizando uma tabela verdade, como segue:

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Da mesma forma, podemos mostrar o comportamento da equação A+B+C (lê-se A ou B ou C) por meio de uma tabela verdade. Como na equação há somente o símbolo "+", trata-se da operação **OU** sobre três variáveis. Logo, pode-se aplicar diretamente a definição da operação **OU**: o resultado será **1** se pelo menos uma das variáveis de entrada valer **1.**

A	**B**	**C**	**A+B+C**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

É importante notar que, devido ao fato de haver somente um operador na equação, pode-se também avaliar a equação decompondo-a em pares. Por exemplo, pode-se primeiramente achar o resultado de A+B, para depois operar os valores resultantes com os respectivos valores de C. Esta propriedade é conhecida como **associativa**. Também a ordem em que são avaliadas as variáveis A, B e C é irrelevante (propriedade **comutativa**). Estas propriedades são ilustradas pela tabela verdade a seguir. Nela, os parêntesis indicam subexpressões já avaliadas em coluna imediatamente à esquerda. Note que os valores das

colunas referentes às expressões A+B+C, (A+B)+C e (B+C)+A são os mesmos (na mesma ordem).

A B C	A+B+C	A+B	(A+B)+C	B+C	(B+C)+A
0 0 0	0	0	0	0	0
0 0 1	1	0	1	1	1
0 1 0	1	1	1	1	1
0 1 1	1	1	1	1	1
1 0 0	1	1	1	0	1
1 0 1	1	1	1	1	1
1 1 0	1	1	1	1	1
1 1 1	1	1	1	1	1

2.1.2 Operação E (Multiplicação Lógica)

A operação **E**, ou **multiplicação** lógica, pode ser definida da seguinte forma:

"A operação **E** resulta **0** se pelo menos uma das variáveis de entrada vale **0**".

Pela definição dada, pode-se deduzir que o resultado da operação **E** será **1** se, e somente se, todas as entradas valerem **1**.

O símbolo usualmente utilizado na operação **E** é "·", porém outra notação possível é "∧". Podemos, também, listar as possibilidades de combinações entre dois valores Booleanos e os respectivos resultados, para a operação **E**:

$$\begin{aligned} 0 \cdot 0 &= 0 \\ 0 \cdot 1 &= 0 \\ 1 \cdot 0 &= 0 \\ 1 \cdot 1 &= 1 \end{aligned}$$

Assim como a operação **OU**, a operação **E** só pode ser definida entre, pelo menos duas variáveis. Ou seja, o operador "·" (**E**) também é **binário**.

Para mostrar o comportamento da equação A · B (lê-se A e B), escreve-se uma tabela verdade, como segue:

A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

De forma semelhante, pode-se determinar o resultado da equação A·B·C (lê-se A e B e C) utilizando diretamente a definição da operação **E**: o resultado será **0** se pelo menos uma das variáveis de entrada valer **0**.

A B C	A·B·C
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

Também para a operação **E** valem as propriedades **associativa** e **comutativa**. Então, a equação A·BC pode ainda ser avaliada tomando-se as variáveis aos pares, em qualquer ordem. Veja a tabela verdade a seguir e compare os resultados.

A B C	A·B·C	A·B	(A·B)·C	B·C	A·(B·C)
0 0 0	0	0	0	0	0
0 0 1	0	0	0	0	0
0 1 0	0	0	0	0	0
0 1 1	0	0	0	1	0
1 0 0	0	0	0	0	0
1 0 1	0	0	0	0	1
1 1 0	0	1	0	0	0
1 1 1	1	1	1	1	1

2.1.3 Complementação (ou Negação, ou Inversão)

A operação **complementação** dispensa uma definição. É a operação cujo resultado é simplesmente o valor complementar ao que a variável apresenta. Também devido ao fato de uma variável Booleana poder assumir um entre somente dois valores, o valor complementar será **1** se a variável vale **0** e será **0** se a variável vale **1**.

Os símbolos utilizados para representar a operação complementação sobre uma variável Booleana A são $\overline{A}$, ~A e A' (lê-se A negado). Nesta disciplina, adotaremos o primeiro símbolo. O resultado da operação complementação pode ser listado:

$$\overline{0} = 1$$
$$\overline{1} = 0$$

Diferentemente das operações **OU** e **E,** a **complementação** só é definida sobre uma variável, ou sobre o resultado de uma expressão. Ou seja, o operador **complementação** é dito **unário**.

E a tabela verdade para $\overline{A}$ é:

A	$\overline{A}$
0	1
1	0

2.2 Avaliação de Expressões Booleanas

Dada a equação que descreve uma função Booleana qualquer, deseja-se saber detalhadamente como esta função se comporta para qualquer combinação das variáveis de entrada. O comportamento de uma função é descrito pela sua tabela verdade e este problema é conhecido como **avaliação** da função ou da expressão que descreve a função considerada. Em suma, deseja-se achar a tabela verdade para a função Booleana.

Uma tabela verdade consiste basicamente de um conjunto de colunas, nas quais são listadas todas as combinações possíveis entre as variáveis de entrada (à esquerda) e o resultado da função (à direita). Também, pode-se criar colunas intermediárias, onde são listados os resultados de subexpressões contidas na expressão principal. Isto normalmente facilita a avaliação, principalmente no caso de equações muito complexas e/ou contendo muitas variáveis.

Quando numa mesma equação Booleana aparecem operações **E** e **OU**, é necessário seguir a ordem de precedência. Tal como na álgebra dos reais, a multiplicação (lógica) tem precedência sobre a adição (lógica). Além disso, expressões entre parêntesis têm precedência sobre operadores **E** e **OU** que estejam no mesmo nível. Quanto à complementação, esta deve ser avaliada tão logo seja possível. Caso a complementação seja aplicada sobre uma subexpressão inteira, é necessário que se avalie primeiramente a subexpressão para, só após, inverter o seu resultado.

O número de combinações que as variáveis de entrada podem assumir pode ser calculado por 2^n, onde n é o número de variáveis de entrada.

O procedimento para a criação da tabela verdade a partir de uma equação Booleana é:

1. Criar colunas para as variáveis de entrada e listar todas as combinações possíveis, utilizando a fórmula nº de combinações = 2^n (onde n é o número de variáveis de entrada);
2. Criar uma coluna para cada variável de entrada que apareça complementada na equação e anotar os valores resultantes;
3. Avaliar a equação seguindo a ordem de precedência, a partir do nível de parêntesis mais internos:
   1º multiplicação lógica
   2º adição lógica

Tomemos como exemplo a expressão $W = X + Y \cdot \overline{Z}$. A variável W representa a função Booleana propriamente dita. Esta variável depende das variáveis que estão à direita do sinal =, ou seja, depende de X, Y e Z. Logo, são 3 as variáveis de entrada. O total de combinações entre 3 variáveis será 2^3=8. Então, a tabela verdade para W deverá ter 3 colunas à esquerda e 8 linhas. Seguindo o procedimento dado acima, cria-se uma coluna, na qual listam-se os valores para $\overline{Z}$. Após, inicia-se a avaliação propriamente dita, a partir do nível mais interno de parêntesis. Como não há parêntesis na expressão, resolvem-se as subexpressões que envolvem a operação **E**. No caso em questão, há somente uma tal subexpressão, que é $X \cdot \overline{Y}$. Então, cria-se uma coluna para $X \cdot \overline{Y}$, na qual anotam-se os resultados para este produto. Finalmente, utilizam-se os resultados de $X \cdot \overline{Y}$, listados na coluna anterior, para operar o **OU** com a variável X. Repare os passos descritos na tabela verdade que segue. Nela, os parêntesis em torno do produto $X \cdot \overline{Y}$ indicam somente que este

termo já foi avaliado e que no passo referente a esta coluna, tomaram-se apenas os valores previamente encontrados.

X Y Z	$\overline{Z}$	$X \cdot \overline{Y}$	$W = X + Y \cdot \overline{Z}$
0 0 0	1	0	0
0 0 1	0	0	0
0 1 0	1	1	1
0 1 1	0	0	0
1 0 0	1	0	1
1 0 1	0	0	1
1 1 0	1	1	1
1 1 1	0	0	1

2.3 Portas Lógicas

Já vimos que uma função Booleana pode ser representada por uma equação ou detalhada pela sua tabela verdade. Mas uma função Booleana também pode ser representada de forma gráfica, onde cada operador está associado a um símbolo específico, permitindo o imediato reconhecimento visual. Tais símbolos são conhecidos por **portas lógicas**.

Na realidade, mais do que símbolos de operadores lógicos, as portas lógicas representam recursos físicos, isto é, circuitos eletrônicos, capazes de realizar as operações lógicas. Na eletrônica que trabalha com somente dois estados, a qual é denominada eletrônica digital, o nível lógico 0 normalmente está associado à ausência de tensão (0 volt) enquanto o nível lógico 1, à presença de tensão (a qual geralmente é 5 volts). Nesta disciplina, nos limitaremos ao mundo da álgebra Booleana, admitindo que as portas lógicas representam também circuitos eletrônicos que, de alguma maneira, realizam as funções Booleanas simbolizadas. Então, ao conjunto de portas lógicas e respectivas conexões que simbolizam uma **equação Booleana**, denominaremos **circuito lógico**.

2.3.1 Porta OU

O símbolo da **porta OU** pode ser visto na figura 2.1. Tal como na porta E, as entradas são colocadas à esquerda e a saída, à direita. Deve haver no mínimo duas entradas, mas há somente uma saída. O funcionamento da porta **E** segue a definição da operação **E**, dada na seção 2.1.1.

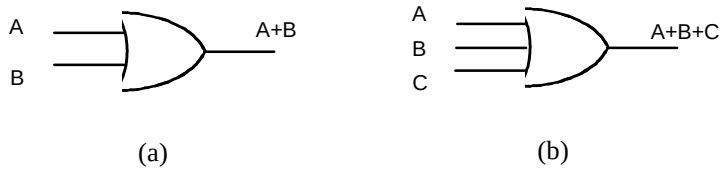


Figura 2.1 - Símbolo da porta lógica **OU** com 2 entradas (a) e com 3 entradas (b).

2.3.2 Porta E

O símbolo da **porta E** é mostrado na figura 2.2. À esquerda estão dispostas as entradas (no mínimo duas, obviamente) e à direita, a saída (única). As linhas que conduzem as variáveis de entrada e saída podem ser interpretadas como fios que transportam os sinais elétricos associados às variáveis. O comportamento da porta E segue extritamente a definição (e tabela verdade) dada na seção 2.1.2.

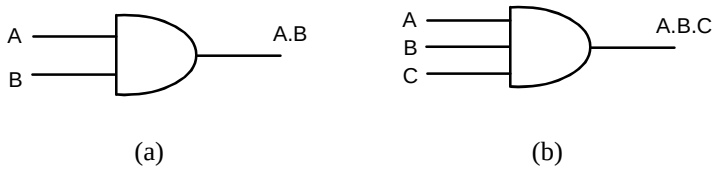


Figura 2.2 - Símbolo da porta lógica **E** com 2 entradas (a) e com 3 entradas (b).

2.3.3 Inversor (ou Porta Inversora, ou Negador)

A porta que simboliza a operação **complementação** é conhecida como **inversor** (ou porta inversora, ou negador). Como a operação complementação só pode ser realizada sobre uma variável por vez (ou sobre o resultado de uma subexpressão), o inversor só possui uma entrada e, obviamente, uma saída. Caso se queira complementar uma expressão, é necessário obter-se primeiramente o seu resultado, para só então aplicar a complementação. O símbolo do inversor é mostrado na figura 2.3.

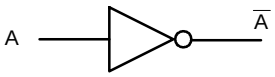


Figura 2.3 - Símbolo do **inversor** (também conhecido como negador ou porta inversora).

2.3.4 Exemplo de Circuito Lógico

Dada uma equação Booleana qualquer, é possível desenhar-se o circuito lógico que a implementa. O circuito lógico é composto das portas lógicas relacionadas às operações que são realizadas sobre as variáveis de entrada. Os resultados das operações são conduzidos por fios, os quais, no desenho, são representados por linhas simples.

Os passos a serem seguidos para se realizar o desenho do circuito lógico a partir de uma equação são praticamente os mesmos usados na avaliação da expressão. Tomemos como exemplo a equação, avaliada na seção 2.2. Inicialmente, identificamos as variáveis independentes, que no caso são X, Y e Z. Para cada uma destas, traçamos uma linha (da esquerda para a direita), representando os fios que conduzem os valores. Feito isto, deve-se seguir desenhando as portas necessárias para representar cada uma das subexpressões, na mesma ordem tomada para a avaliação, ou seja:

1º parêntesis (dos mais internos para os mais externos);

2º operações E;

3º operações OU.

A figura 2.4 mostra o circuito lógico para a equação $W = X + Y \cdot \overline{Z}$.

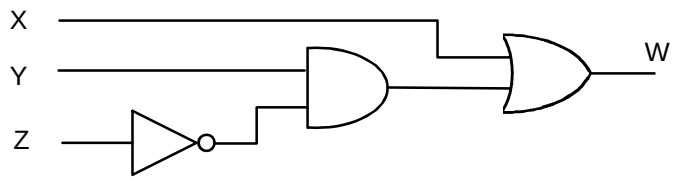


Figura 2.4 - Um circuito lógico.

2.4 Leis Fundamentais e Propriedades da Álgebra Booleana

As leis da álgebra Booleana dizem respeito ao espaço Booleano (isto é., valores que uma variável pode assumir) e operações elementares deste espaço. Já as propriedades podem ser deduzidas a partir das definições das operações.

Sejam A e B duas variáveis Booleanas. Então, o espaço Booleano é definido:

se A≠0, então A=1;

se A≠1, então A=0.

As operações elementares deste espaço são operação **OU**, operação **E** e **complementação**, cujas definições foram dadas nas seções 2.1.1, 2.1.2 e 2.1.3, respectivamente.

As propriedades da álgebra Booleana são as seguintes.

Da adição lógica:

(1) $A + 0 = A$
(2) $A + 1 = 1$
(3) $A + A = A$
(4) $A + \overline{A} = 1$

Da multiplicação lógica:

(5) $A \cdot 0 = 0$
(6) $A \cdot 1 = A$
(7) $A \cdot A = A$
(8) $A \cdot \overline{A} = 0$

Da complementação:

(9) $\overline{\overline{A}} = A$

Comutatividade:

(10) $A + B = B + A$
(11) $A \cdot B = B \cdot A$

Associatividade:

$$A + (B + C) = (A + B) + C = (A + C) + B \quad (12)$$

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C = (A \cdot C) \cdot B \quad (13)$$

Distributiva (da multiplicação em relação à adição):

$$A \cdot (B + C) = A \cdot B + A \cdot C \quad (14)$$

2.4.1 Teoremas de De Morgan

O primeiro teorema de De Morgan diz que a complementação de um produto (lógico) equivale à soma (lógica) das negações de cada variável do referido produto. Sob a forma de equação, teríamos:

$$\overline{A \cdot B \cdot C \cdot ...} = \overline{A} + \overline{B} + \overline{C} + ... \quad (2.1)$$

O segundo teorema é o dual (i.e., o espelho) do primeiro, ou seja, a complementação de uma soma (lógica) equivale ao produto das negações individuais das variáveis:

$$\overline{A + B + C + ...} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot ... \quad (2.2)$$

Particularizando os teoremas de De Morgan para duas variáveis, temos:

$$\overline{A \cdot B} = \overline{A} + \overline{B} \quad (2.3)$$

$$\overline{A + B} = \overline{A} \cdot \overline{B} \quad (2.4)$$

2.5 Derivação de Expressões Booleanas

Dada uma função Booleana, descrita por sua tabela verdade, derivar uma expressão Booleana para esta função é encontrar uma equação que a descreva. Logo, a derivação de expressões Booleanas é o problema inverso da avaliação de uma expressão Booleana, descrito na seção 2.2

Há basicamente duas maneiras de se definir (ou descrever) uma função Booleana: descrevendo-se todas as situações das variáveis de entrada para as quais a função vale **1** ou, alternativamente, todas as situações em que a função vale **0**. O primeiro método é conhecido por **soma de produtos** (SdP), enquanto que o segundo é chamado **produto de somas** (PdS). Qualquer função Booleana pode ser descrita por meio de soma de produtos ou por meio de produto de somas. Como as funções Booleanas só podem assumir um dentre dois valores (0 ou 1), basta usar-se um dos dois métodos para se encontrar uma equação para uma função.

A seguir, são detalhados os métodos de derivação de expressões Booleanas.

2.5.1 Derivação de Expressões usando Soma de Produtos (SdP)

Dada uma função Booleana de n variáveis (ou seja, n entradas), haverá 2^n combinações possíveis de valores. Dizemos que esse conjunto de valores que as variáveis podem assumir, juntamente com os respectivos valores da função, constituem o espaço da função. A cada combinação de entradas podemos associar um **termo produto,** no qual todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável correspondente vale **0**, ela deve aparecer negada; se a variável vale **1**, ela deve aparecer não negada. A tabela a seguir lista os termos produto associados a cada combinação de entradas para uma função Booleana de três variáveis (A, B e C, por exemplo).

A B C	mintermo
0 0 0	$\overline{A}\cdot\overline{B}\cdot\overline{C}$
0 0 1	$\overline{A}\cdot\overline{B}\cdot C$
0 1 0	$\overline{A}\cdot B\cdot\overline{C}$
0 1 1	$\overline{A}\cdot B\cdot C$
1 0 0	$A\cdot\overline{B}\cdot\overline{C}$
1 0 1	$A\cdot\overline{B}\cdot C$
1 1 0	$A\cdot B\cdot\overline{C}$
1 1 1	$A\cdot B\cdot C$

Cada termo produto construído conforme a regra anteriormente descrita é denominado **mintermo** (ou minitermo). Note que, para um dado mintermo, se substituirmos os valores das variáveis associadas, obteremos 1. Porém, se substituirmos nesse mesmo mintermo quaisquer outras combinações de valores, obteremos 0. Dessa forma, se quisermos encontrar a equação para uma função a partir de sua tabela verdade, basta montarmos um **OU** entre os mintermos associados aos **1**s da função (também chamados **mintermos 1**).

Exemplo 2.1: encontrar a equação em soma de produtos (SdP) para a função F, descrita pela seguinte tabela verdade:

A B C	F
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	0

F é função das variáveis A, B e C. Os valores de (A,B,C) para os quais F=1 são (0,1,0), (0,1,1), (1,0,1) e (1,1,0). Os mintermos associados a essas condições (ou seja, os mintermos 1), são $\overline{A}\cdot B\cdot\overline{C}$, $\overline{A}\cdot B\cdot C$, $A\cdot\overline{B}\cdot C$ e $A\cdot B\cdot\overline{C}$, respectivamente. Logo, a equação em soma de produtos para F será o **OU** entre estes produtos, conforme segue:

$$F = \overline{A}\cdot B\cdot\overline{C} + \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} \qquad (2.5)$$

A fim de simplificar a notação, o símbolo da operação **E** pode ser omitido. Desta forma, a equação anterior pode ser reescrita de maneira mais concisa:

$$F = \overline{A}\,B\overline{C} + \overline{A}\,BC + A\,\overline{B}C + A\,B\overline{C} \qquad (2.6)$$

2.5.2 Derivação de Expressões usando Produto de Somas (PdS)

O método de derivação usando produto de somas é o **dual** (isto é, o oposto) do método de derivação em soma de produtos. A cada combinação das variáveis de entrada de uma função podemos associar um **termo soma,** no qual todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável correspondente vale **1**, ela deve aparecer negada; se a variável vale **0**, ela deve aparecer não negada. A tabela a seguir lista os termos soma associados a cada combinação de entradas para uma função Booleana de três variáveis (A, B e C, por exemplo).

A B C	maxtermos
0 0 0	$A+B+C$
0 0 1	$A+B+\overline{C}$
0 1 0	$A+\overline{B}+C$
0 1 1	$A+\overline{B}+\overline{C}$
1 0 0	$\overline{A}+B+C$
1 0 1	$\overline{A}+B+\overline{C}$
1 1 0	$\overline{A}+\overline{B}+C$
1 1 1	$\overline{A}+\overline{B}+\overline{C}$

Cada termo soma construído conforme a regra anteriormente descrita é denominado **maxtermo** (ou maxitermo). Note que, para um dado maxtermo, se substituirmos os valores das variáveis associadas, obteremos 0. Porém, se substituirmos nesse mesmo maxtermo quaisquer outras combinações de valores, obteremos 1. Dessa forma, se quisermos encontrar a equação para uma função a partir de sua tabela verdade, basta montarmos um **E** entre os maxtermos associados aos **0**s da função (também chamados **maxtermos 0**).

Exemplo 2.2: encontrar a equação em produto de somas (PdS) para a função F, descrita pela seguinte tabela verdade:

A B C	F
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	0

Foi escolhida a mesma função do exemplo anterior, para que se possa estabelecer comparações entre os dois métodos de derivação. Os valores das variáveis de entrada (A,B,C) para os quais F=0 são (0,0,0), (0,0,1), (1,0,0) e (1,1,1). Os maxtermos associados a essas condições (ou seja, os maxtermos 0), são $A+B+C$, $A+B+\overline{C}$, $\overline{A}+B+C$ e $\overline{A}+\overline{B}+\overline{C}$, respectivamente. Logo, a equação em produto de somas para F será o **E** entre estas somas:

$$F=(A+B+C)(A+B+\overline{C})(\overline{A}+B+C)(\overline{A}+\overline{B}+\overline{C}) \quad (2.7)$$

Note que a ordem de precedência de uma expressão em produto de somas é "primeiro cada soma deve ser avaliada, para só então avaliar-se o produto". Isto significa que os parêntesis em torno de cada termo soma **são obrigatórios**! Repare também que os símbolos referentes à operação **E** (entre os termos soma) podem ser omitidos.

2.6 Formas Canônicas, Padrão e Não-Padrão

As representações em soma de produtos e em produto de somas são denominadas formas **padrão**. A soma de produtos e o produto de somas descritos nas duas seções anteriores apresentam ainda uma característica bastante particular: em cada termo soma e em cada termo produto todas as variáveis da função estão presentes. Devido a essa característica, essas formas são chamadas **canônicas**.

Além das representações descritas nas seções anteriores, há representações alternativas (e mais concisas) para as expressões canônicas. Se associarmos cada combinação das variáveis de entrada ao seu equivalente em decimal, cada mintermo pode ser representado por m_i, onde i é o decimal associado. De forma similar, cada maxtermo pode ser representado por M_i, onde i é o decimal associado. A tabela a seguir lista todos os mintermos e maxtermos de uma função de três variáveis (A, B e C).

A B C	mintermo	maxtermo
0 0 0	m_0	M_0
0 0 1	m_1	M_1
0 1 0	m_2	M_2
0 1 1	m_3	M_3
1 0 0	m_4	M_4
1 0 1	m_5	M_5
1 1 0	m_6	M_6
1 1 1	m_7	M_7

Voltando à função F das seções anteriores, podemos reescrever a expressão em soma de produtos, na forma canônica, como segue:

$$F = m_2 + m_3 + m_5 + m_6 \quad (2.8)$$

Ou ainda, de maneira mais concisa:

$$F = \sum(2,3,5,6) \quad (2.9)$$

E sua expressão em produto de somas, na forma canônica, pode ser reescrita como:

$$F = M_0 \cdot M_1 \cdot M_4 \cdot M_7 \qquad (2.10)$$

Ou simplesmente, como:

$$F = \prod(0,1,4,7) \qquad (2.11)$$

Apesar da praticidade das representações canônicas, elas são pouco úteis para a implementação de circuitos digitais. O número de elementos (portas lógicas e conexões) de um circuito lógico depende diretamente do número de operações Booleanas (inversão, E e OU) contidas na expressão associada. Desta forma, é normal que se deseje reduzir o número de operações contidas numa função, de modo a poder-se implementá-la com circuitos lógicos mais simples, e portanto, de menor custo. A redução do número de operações é obtida mediante a eliminação de literais da expressão, aplicando-se as propriedades da álgebra Booleana descritas na seção 2.4. Um literal é uma variável negada ou uma variável não negada. O processo de redução de literais (ou de redução de operações, equivalentemente) é denominado **simplificação**.

Para exemplificar os passos básicos para a simplificação algébrica (literal) de expressões Booleanas, tomemos a expressão canônica, em soma de produtos, para a função F:

$$F = \overline{A}\,B\overline{C} + \overline{A}\,BC + A\,\overline{B}C + A\,B\overline{C} \qquad (2.12)$$

O primeiro passo é identificar pares de mintermos que se diferenciam por apenas um literal, a fim de aplicar a propriedade (14). Os mintermos $\overline{A}B\overline{C}$ e $\overline{A}BC$, por exemplo, possuem os mesmos literais, exceto pela variável C: no primeiro, o literal é $\overline{C}$, enquanto no segundo, o literal é C. Então, com o uso da propriedade (14), pode-se fatorar esses dois mintermos, obtendo-se:

$$F = \overline{A}\,B(\overline{C} + C) + A\,\overline{B}C + A\,B\overline{C} \qquad (2.13)$$

Pela propriedade (4), tem-se que $\overline{C} + C = 1$. Então, substituindo em 2.13, segue:

$$F = \overline{A}B \cdot 1 + A\overline{B}C + AB\overline{C} \qquad (2.14)$$

E pela propriedade (6), $\overline{A}B \cdot 1 = \overline{A}B$. Substituindo em 2.14, obtém-se:

$$F = \overline{A}\,B + A\,\overline{B}C + A\,B\overline{C} \qquad (2.15)$$

Assim, pela manipulação algébrica, obtivemos uma expressão em soma de produtos que é simplificada em relação a sua expressão em soma de produtos na forma canônica, pois o número de operações e também de literais foram reduzidos (compare 2.15 com 2.12).

Entretanto, na equação 2.12, o mintermo $\overline{A}B\overline{C}$ também poderia ter sido agrupado com o mintermo $AB\overline{C}$, pois ambos possuem os mesmos literais, exceto pela variável A ($\overline{A}$ no primeiro e A no segundo). Naturalmente, os passos a serem seguidos seriam os mesmos descritos anteriormente. E a equação resultante seria um pouco diferente, mas com o mesmo número de operações, sendo portanto, de mesma complexidade. Na verdade, o melhor seria se pudéssemos agrupar o mintermo $\overline{A}B\overline{C}$ com o mintermo $AB\overline{C}$ e ao mesmo tempo com o mintermo $\overline{A}BC$. Felizmente, a propriedade (3) da álgebra Booleana diz que o **OU** entre duas ou mais variáveis Booleanas iguais é igual a própria variável Booleana em questão. Estendendo esta propriedade, pode-se dizer que o **OU** entre duas ou mais funções (inclusive

produtos) Booleanas iguais equivale à própria função Booleana em questão. Desta forma, pode-se expandir o mintermo $\overline{A}B\overline{C}$ para

$$\overline{A}B\overline{C} = \overline{A}B\overline{C} + \overline{A}B\overline{C} \qquad (2.16)$$

que é uma manipulação algébrica decorrente da propriedade (3).

Retomando a equação 2.12 e utilizando 2.16, segue que:

$$F = \overline{A}\,B\overline{C} + \overline{A}\,BC + A\,\overline{B}C + A\,B\overline{C} + \overline{A}\,B\overline{C} \quad (2.17)$$

Então, a propriedade (3) garante que as expressões 2.12 e 2.17 são equivalentes, embora o mintermo $\overline{A}B\overline{C}$ apareça duplicado. E pelo fato de aparecer duas vezes, pode-se usar uma cópia de $\overline{A}B\overline{C}$ para simplificar com $AB\overline{C}$ e outra para simplificar com $\overline{A}BC$. Os passos da simplificação são os mesmos já descritos: pela propriedade (14), segue:

$$F = \overline{A}\,B(\overline{C} + C) + A\,\overline{B}C + (A + \overline{A})B\overline{C} \qquad (2.18)$$

E pela propriedade (6), vem:

$$F = \overline{A}B \cdot 1 + A\overline{B}C + 1 \cdot B\overline{C} \qquad (2.19)$$

Finalmente, pela propriedade (4), tem-se:

$$F = \overline{A}\,B + A\,\overline{B}C + B\overline{C} \qquad (2.20)$$

Repare que o mintermo $A\overline{B}C$ não pôde ser agrupado com nenhum outro mintermo. Note também que foram feitas todas as simplificações possíveis, uma vez que foram agrupados e simplificados todos os pares de mintermos que se diferenciam de somente uma variável. Logo, a expressão 2.20 representa a máxima simplificação possível sob a forma de soma de produtos. E por esse motivo, ela é dita equação **mínima** em **soma de produtos** da função F. Quanto a expressão 2.15, diz-se ser uma equação em **soma de produtos simplificada** (porém, não-mínima). Logo, toda equação mínima é simplificada, porém, nem toda equação que foi simplificada é necessariamente mínima.

Embora a equação mínima em soma de produtos apresente menor número de operações Booleanas que a representação na forma canônica, as vezes pode ser possível reduzir-se ainda mais o número de operações, fatorando-se literais. Por exemplo, na expressão 2.20 pode-se fatorar o primeiro e o terceiro mintermos como segue:

$$F = B(\overline{A} + \overline{C}) + A\,\overline{B}C \qquad (2.21)$$

A expressão 2.21, obtida pela fatoração de 2.20, não é nem do tipo soma de produtos, nem produto de somas, pois há um termo que não é nem produto, nem soma. Diz-se que a expressão está na **forma fatorada**. No caso de 2.21, a fatoração não resultou em redução do número de operações.

No que se refere a terminologia, as formas soma de produtos e produto de somas são ditas **formas padrão** (formas *standard*). A forma fatorada é dita **não-padrão**. As formas **canônicas** são, pois, casos especiais de formas padrão, nas quais os termos são mintermos ou maxtermos. A fim de diferenciar somas de produtos canônicas de somas de produtos simplificadas, usaremos a expressão "**soma de mintermos"**. De maneira similar, usaremos a expressão "**produto de maxtermos"** para diferenciar produtos de somas canônicos de produtos de somas simplificados.

2.7 Circuitos Lógicos para Formas Padrão e Não-Padrão

As regras gerais para se realizar o desenho de circuitos lógicos já foram apresentadas na seção 2.3 (item 2.3.4). As regras seguintes devem ser observadas, a fim de facilitar a compreensão do desenho:

- as variáveis de entrada devem ser identificadas preferencialmente à esquerda, junto aos respectivos fios;
- inversores devem ser providos para as variáveis que aparecem negadas na equação;
- as portas que implementam as operações Booleanas que aparecem na equação normalmente são posicionadas da esquerda para a direita, seguindo a ordem de avaliação dos operadores (descrita em 2.3.4).

No caso de equações na forma soma de produtos (canônica ou simplificada), há um primeiro nível (desconsiderando-se possíveis inversores), constituído somente por portas **E**, onde cada porta **E** implementa um dos produtos da equação. Há ainda um segundo nível, constituído por uma porta **OU**, responsável pela "soma" lógica dos produtos. A figura 2.5 mostra um possível circuito lógico para a equação 2.12.

Repare que em todas as interseções de fios em que há conexão física, **deve haver um ponto** (suficientemente grande), como se fora uma "solda". Logo, quando não há o referido ponto na interseção de fios, significa que tais fios estão "eletricamente isolados".

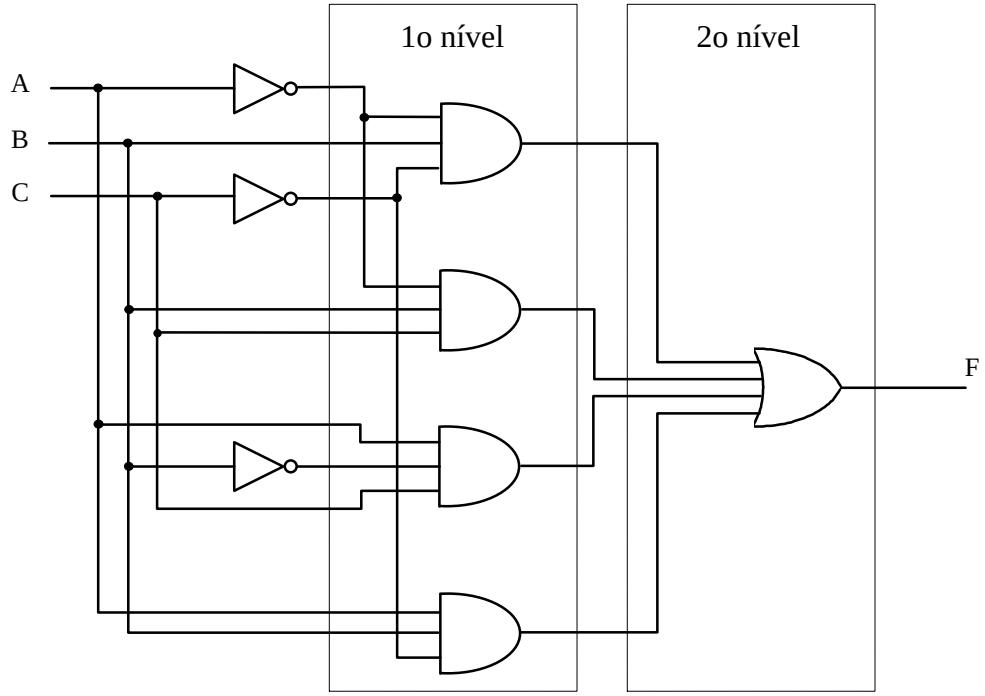


Figura 2.5 - Um circuito lógico para soma de produtos.

O circuito da figura 2.5 pode ainda ser desenhado utilizando-se uma notação simplificada para os inversores das entradas. Ao invés de se desenhar um inversor para cada variável que aparece negada na equação, coloca-se um círculo junto a cada entrada de cada

porta na qual há uma variável negada. A aplicação desse procedimento para o circuito da figura 2.5 resulta no seguinte desenho:

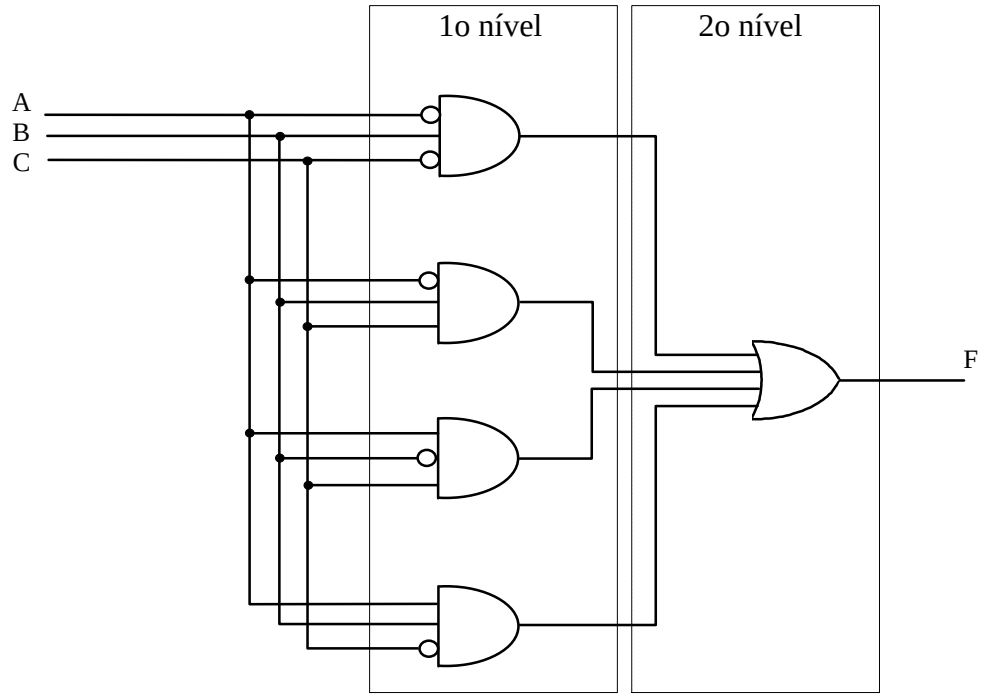


Figura 2.6 - Um circuito lógico para soma de produtos - outra possível representação.

No caso de equações na forma produto de somas (canônica ou simplificada), o primeiro nível é constituído por portas **OU**, sendo cada uma responsável por uma das "somas" lógicas da equação. O segundo nível, por sua vez, é constituído por uma porta **E**, que realiza o produto lógico das parcelas. A figura 2.7 mostra um possível circuito lógico para a equação 2.7.

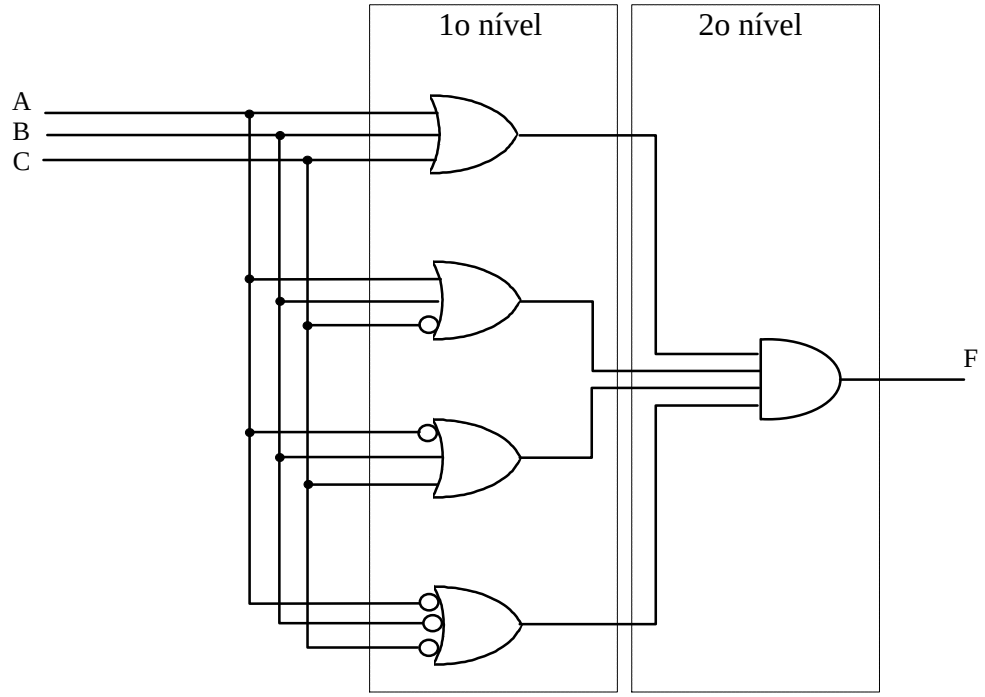


Figura 2.7 - Um circuito lógico para produto de somas.

Pelo fato de apresentarem apenas dois níveis de portas (dois níveis lógicos), circuitos para equações representadas nas formas padrão, canônicas ou simplificadas, são ditos **circuitos em dois níveis** (ou **lógica a dois níveis**).

Dada a equação canônica de uma função qualquer, o circuito para uma equação simplificada a partir da canônica possui menos portas e/ou portas de menor complexidade. (A complexidade relativa de uma porta pode ser medida pelo número de entradas que ela apresenta). A figura 2.8 mostra o circuito lógico para a equação 2.20, que é a forma mínima para a função da equação 2.12. Note que este circuito é de menor complexidade que o circuito da figura 2.6. A complexidade relativa de um circuito lógico pode ser calculada somando-se o número de entradas das portas. Nos circuitos das figuras 2.6 e 2.7 há 4 portas de 3 entradas e 1 porta de 4 entradas. Então, a complexidade relativa será 4x3+1x4=16. No circuito da figura 2.8 há 2 portas de 2 entradas e 2 portas de 3 entradas. Sua complexidade relativa será 2x2+2x3=10. Claramente, o circuito da figura 2.8 é de menor complexidade que os circuitos das figuras 2.6 e 2.7. Estes dois são de mesma complexidade relativa. No cálculo da complexidade relativa, as inversões normalmente não são levadas em conta.

Circuitos para formas fatoradas podem ser vistos como o caso mais genérico. Em geral, as formas fatoradas conduzem a circuitos cujo número de níveis lógicos é maior do que dois. Por isso, circuitos lógicos para formas fatoradas são denominados **circuitos multinível** (**lógica multinível**). Como dito na seção 2.6, as vezes uma forma fatorada pode apresentar menor número de operações do que a respectiva forma padrão. Quando isso ocorre, o circuito associado à forma fatorada também será de menor complexidade relativa. Entretanto, se não ocorrer redução no número de operações, mesmo assim é possível que o circuito para a forma fatorada seja de menor complexidade relativa, pois o conceito de complexidade relativa também inclui o número de entradas de cada porta. Então, a maneira mais segura de saber se o circuito associado à forma fatorada é de menor complexidade ou não é desenhá-lo e somar o número de entradas. A figura 2.9 mostra o circuito para a equação 2.21, obtida a partir da equação 2.20 fatorando-se o literal B. Note que o número de operações Booleanas destas equações é o mesmo: 4. No entanto, a complexidade do circuito da forma fatorada é 3x2+1x3=9, portanto menor do que a complexidade do circuito da figura 2.8.

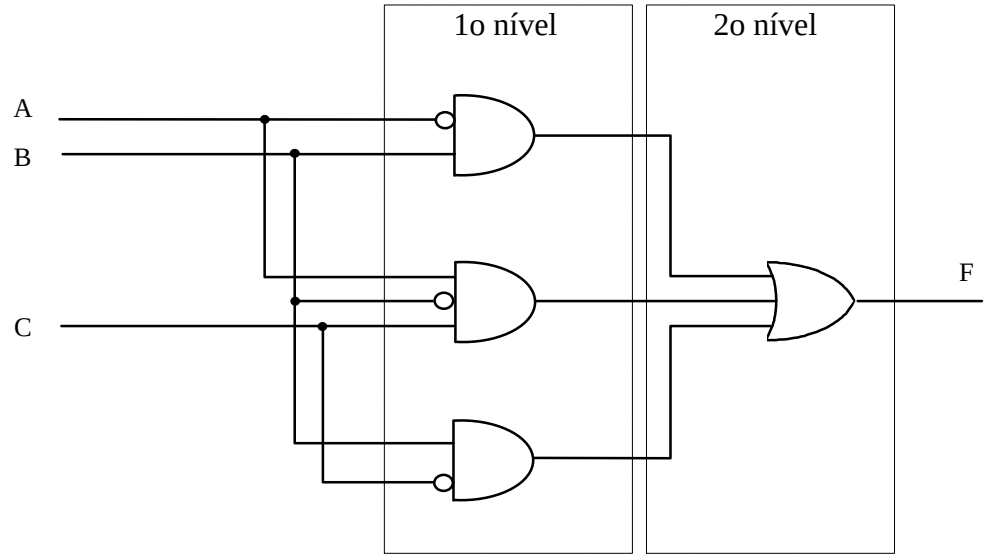


Figura 2.8 - Circuito lógico para a equação 2.20.

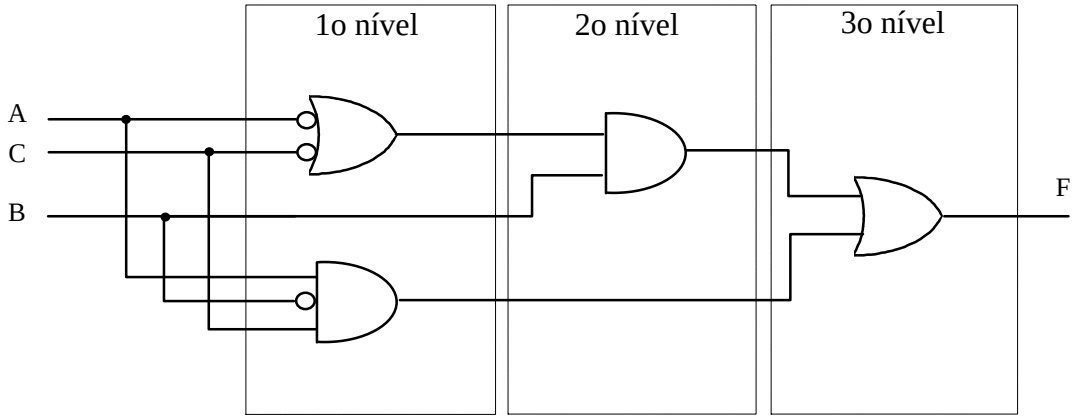


Figura 2.9 - Circuito lógico multinível, associado à equação 2.21, a qual está na forma fatorada.

2.8 Simplificação de Funções Booleanas usando Mapas de Karnaugh

O método de simplificação apresentado na seção 2.6 é de aplicabilidade limitada, uma vez que é bastante difícil certificar-se que todos os pares de mintermos que podiam ser simplificados foram determinados. Alternativamente àquele método, há outro método de simplificação baseado na identificação visual de grupos de mintermos passíveis de serem simplificados. No entanto, para que se possa identificar tais grupos, é necessário que os mintermos sejam dispostos de maneira mais conveniente, o que será explicado a seguir.

Na descrição do método de simplificação literal, foi mencionado que os pares de mintermos que se diferenciam de somente uma variável são passíveis de simplificação. Observando a ordem com que os mintermos de uma função Booleana qualquer (com, por exemplo, 3 variáveis) aparecem na tabela verdade, vemos que, se trocarmos o 3° mintermo com o 4° e trocarmos também o 7° mintermo com o 8°, obteremos uma nova ordem, na qual quaisquer dois mintermos adjacentes são passíveis de simplificação. É interessante notar também que o 1° mintermo pode ser simplificado com o 5°, o 2° mintermo pode ser simplificado com o 6° e assim por diante.

A B C	mintermo
0 0 0	$\overline{A}\cdot\overline{B}\cdot\overline{C}$
0 0 1	$\overline{A}\cdot\overline{B}\cdot C$
0 1 0	$\overline{A}\cdot B\cdot\overline{C}$
0 1 1	$\overline{A}\cdot B\cdot C$
1 0 0	$A\cdot\overline{B}\cdot\overline{C}$
1 0 1	$A\cdot\overline{B}\cdot C$
1 1 0	$A\cdot B\cdot\overline{C}$
1 1 1	$A\cdot B\cdot C$

Figura 2.10 - Mintermos para uma função de 3 variáveis.

Então, usando o novo ordenamento e re-arranjando os mintermos em duas linhas, temos a seguinte tabela:

$\overline{A}\cdot\overline{B}\cdot\overline{C}$	$\overline{A}\cdot\overline{B}\cdot C$	$\overline{A}\cdot B\cdot C$	$\overline{A}\cdot B\cdot\overline{C}$
$A\cdot\overline{B}\cdot\overline{C}$	$A\cdot\overline{B}\cdot C$	$A\cdot B\cdot C$	$A\cdot B\cdot\overline{C}$

Figura 2.11 - Tabela de adjacências para uma função de 3 variáveis.

Repare que nessa nova tabela, quaisquer dois mintermos adjacentes (na horizontal ou na vertical) são passíveis de serem simplificados, pois só se diferenciam de uma variável. É importante ressaltar que esse conceito de adjacência **não está restrito aos limites da tabela**, uma vez que os elementos extremos de uma mesma linha (ou de uma mesma coluna) também são simplificáveis. Isto implica que a tabela de adjacências de mintermos da figura 2.11 pode e deve ser encarada como uma figura geométrica tridimensional do tipo "toróide" (ou uma "rosquinha"). A figura a seguir explicita as relações de adjacência dos mintermos para uma função de três variáveis.

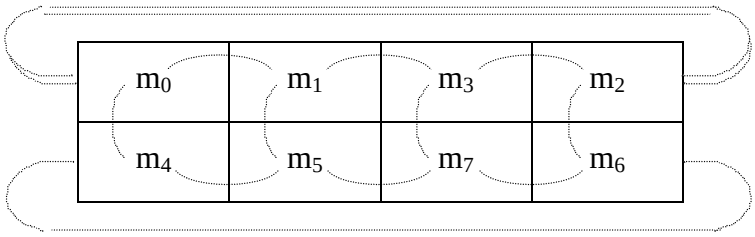


Figura 2.12 - Simplificações possíveis entre os mintermos de uma função de 3 variáveis.

É importante ressaltar que o conceito de adjacência é aplicável na horizontal e na vertical, **mas nunca na diagonal**. Por exemplo, observe que m_0 e m_5 não são adjacentes, pois não estão nem na mesma linha, nem na mesma coluna.

O processo de simplificação usando a nova disposição inicia pela construção da tabela em si: os valores da função que se quer simplificar devem ser preenchidos conforme a nova ordem. Após, identificam-se todos os grupos de mintermos-1 adjacentes entre si. Cada grupo origina um termo produto, no qual somente as variáveis comuns a todos os mintermos-1 permanecem. Desde que os grupos de mintermos-1 adjacentes tenham sido corretamente identificados, a simplificação se torna trivial.

Exemplo 2.3: simplificar a função F, cuja tabela verdade encontra-se no item 2.5.2.

O primeiro passo é construir uma tabela para F, usando a nova disposição dos mintermos.

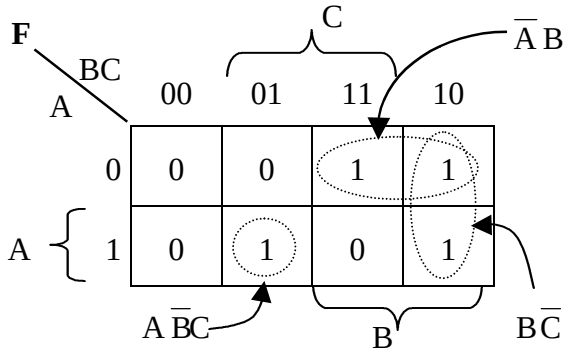


Figura 2.13 - Grupos de mintermos-1 adjacentes e termos produto para uma função de 3 variáveis.

Após, deve-se identificar todos os grupos de mintermos-1 adjacentes entre si. Cada grupo de mintermos-1 originará um produto, conforme indicado na figura 213. A equação em soma de produtos simplificada será o **OU** entre os produtos encontrados:

$$F = \overline{A}\,B + A\,\overline{B}\,C + B\,\overline{C} \qquad (2.22)$$

2.8.1 Mapas de Karnaugh e Subcubos

A tabela modificada usada para a identificação dos mintermos-1 adjacentes no exemplo anterior é denominada **mapa de Karnaugh** (no caso, para 3 variáveis). Na figura 2.14 são mostrados os mapas de Karnaugh para funções de 2, 3 e 4 variáveis.

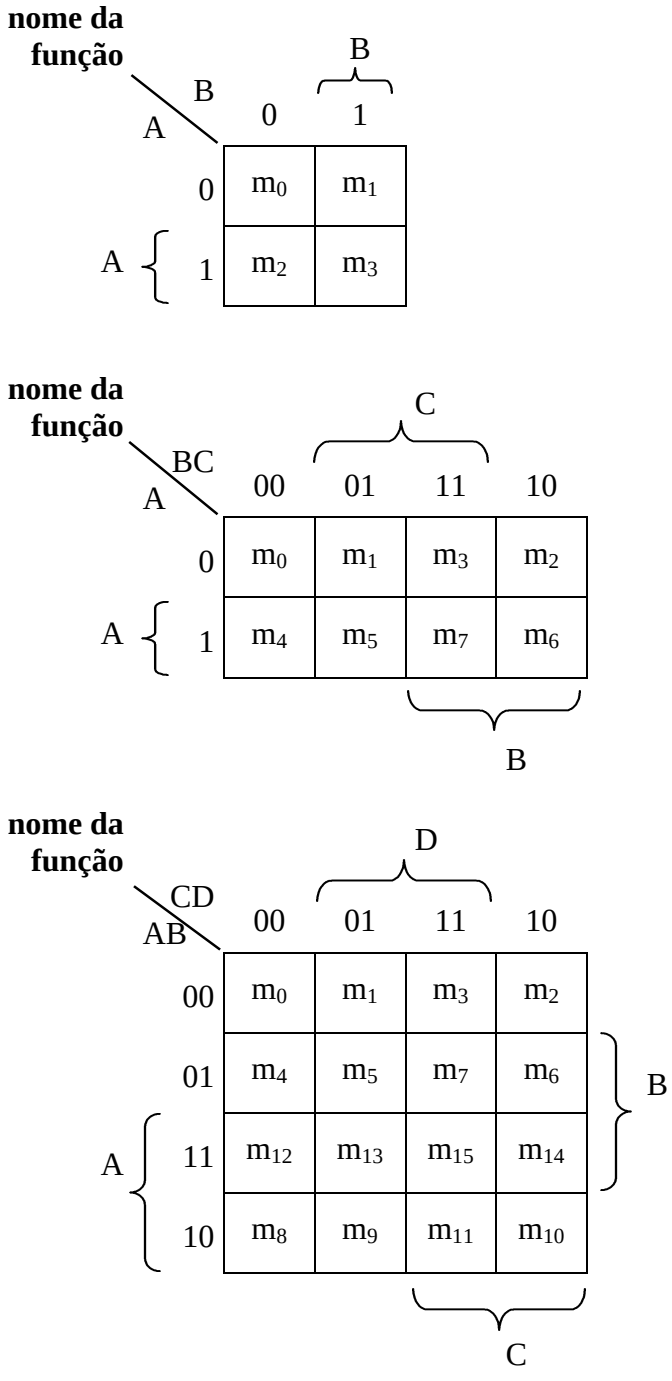
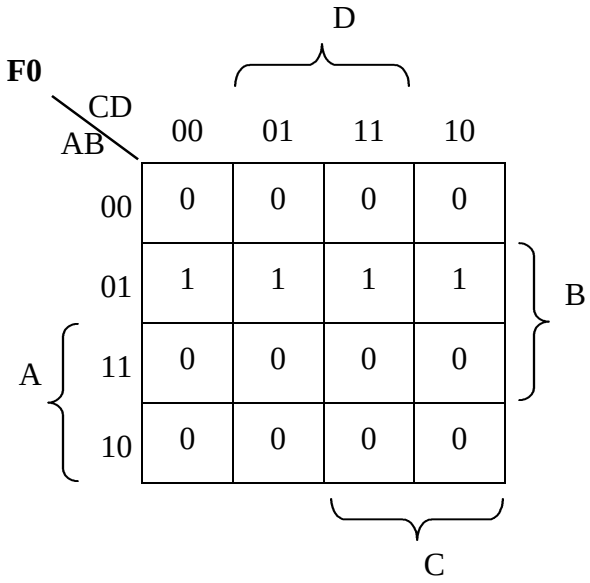


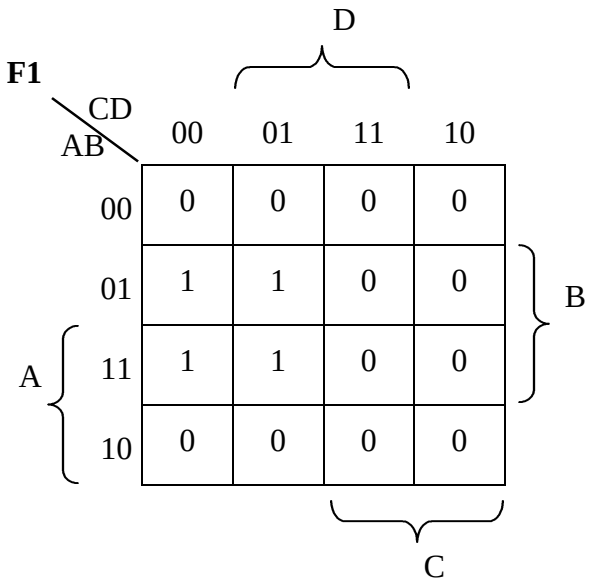
Figura 2.14 - Mapas de Karnaugh para funções de 2, 3 e 4 variáveis.

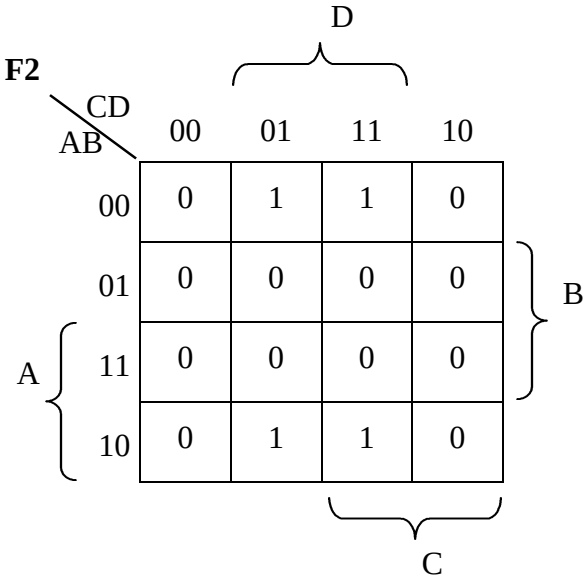
O primeiro passo para simplificar-se uma função usando mapa de Karnaugh é escolher o mapa conforme o número de variáveis da função e preencher os valores dos mintermos conforme a tabela verdade fornecida, ou conforme a equação fornecida. O segundo passo é identificar grupos de mintermos adjacentes que formem grupos de 2^m elementos adjacentes entre si, com $0 \leq m \leq n$, onde n é o número de variáveis da função. Estes grupos são denominados **subcubos**.

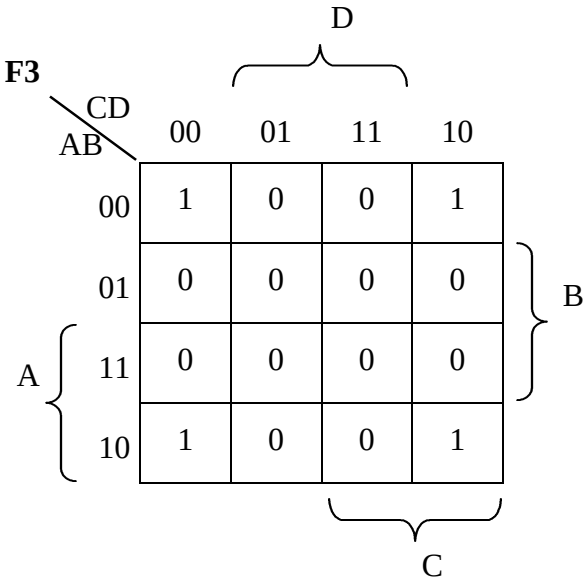
No caso de se querer encontrar uma expressão em soma de produtos, estaremos interessados nos subcubos de mintermos-1. Então, cada subcubo contendo mintermos-1 irá originar um produto, no qual uma ou mais variáveis poderão estar ausentes devido à simplificação que é obtida. Os produtos associados aos subcubos de mintermos-1, simplificados ou não, são denominados **implicantes**. É importante ressaltar que quanto maior o número de elementos do subcubo, maior será a simplificação obtida.

Exemplo 2.4: determinar os implicantes das funções dadas a seguir.









No caso de se querer encontrar uma expressão em produtos de somas, estaremos interessados nos subcubos de mintermos-0. Então, cada subcubo contendo mintermos-0 irá originar uma soma, no qual uma ou mais variáveis poderão estar ausentes devido à simplificação que é obtida. As somas associados aos subcubos de mintermos-0, simplificadas ou não, são denominadas **implicados**. Também neste caso, quanto maior o número de elementos do subcubo, maior será a simplificação obtida.

Exemplo 2.5: determinar os implicados das funções dadas a seguir.

F4

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	0	1	1
10	1	1	1	1

D, B, A, C

F5

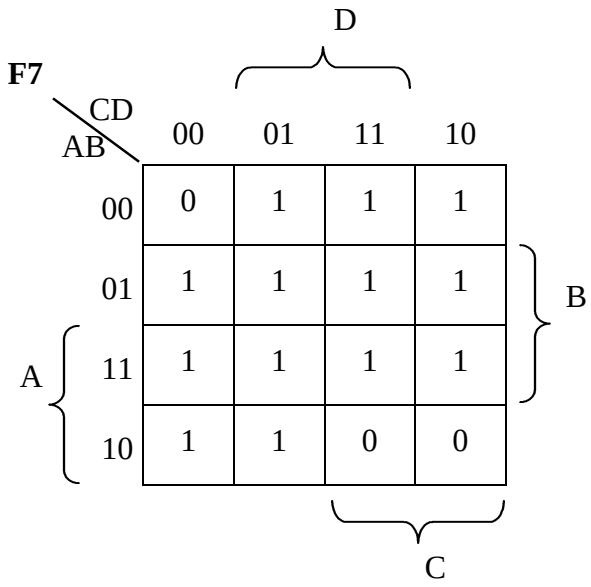
AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

D, B, A, C

F6

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

D, B, A, C



2.8.2 Cobertura dos Mapas de Karnaugh

Normalmente, é possível identificar-se numa mesma função Booleana mais de um implicante (ou mais de um implicado). Neste caso, é necessário determinar o conjunto de implicantes (ou implicados) que melhor "cobre" a função, onde a melhor cobertura significa necessariamente a expressão mais simplificada possível, a qual é denominada expressão mínima.

O procedimento básico para se determinar a melhor cobertura (também chamada cobertura mínima) para uma expressão em soma de produtos é o seguinte:

- Identificar os subcubos de mintermos-1 com maior número de elementos possível, iniciando do tamanho 2^n, onde n é o número de variáveis da função. Caso algum mintermo-1 fique isolado (isto é, não há nenhum outro mintermo-1 adjacente a ele), então ele constituirá um subcubo de um elemento;

- Identificar o menor conjunto de subcubos de modo que cada mintermo-1 pertença a pelo menos um subcubo (= seja coberto pelo menos uma vez).

Observações:

1. Cada mintermo-1 pode ser coberto por mais de um subcubo, caso isso resulte numa simplificação maior;

2. Um último teste para verificar se a expressão obtida é realmente a mínima consiste em verificar se algum subcubo pode ser removido, sem deixar algum mintermo-1 descoberto. Um subcubo que poder ser removido sem descobrir mintermos é dito subcubo **não-essencial**. Logo, todo o subcubo que não pode ser removido é dito **essencial**.;

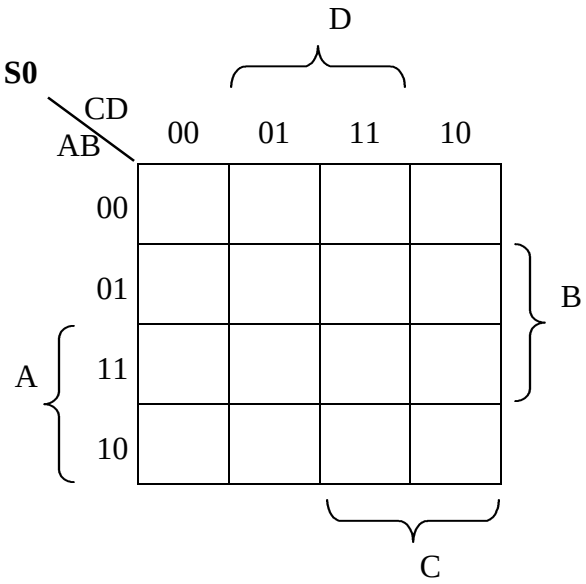
3. Pode haver mais de uma expressão mínima para uma mesma função Booleana;

4. A expressão mínima é aquela de menor complexidade. E a complexidade será medida pelo número de literais de uma função.

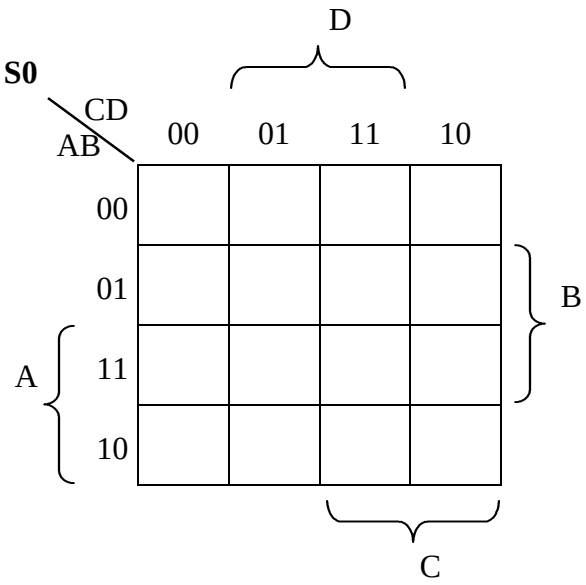
Exemplo 2.6:. ache a equação mínima em soma de produtos para as funções Booleanas que seguem:

$$S0(A,B,C,D) = \sum(0,1,2,5,6,7,13,15)$$

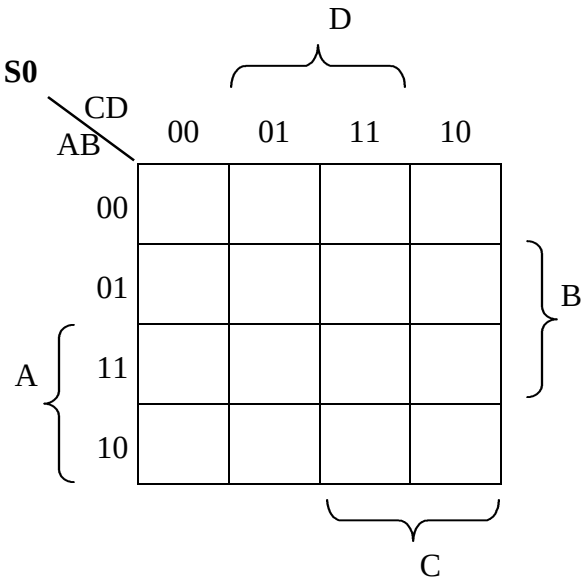
Cobertura 1:



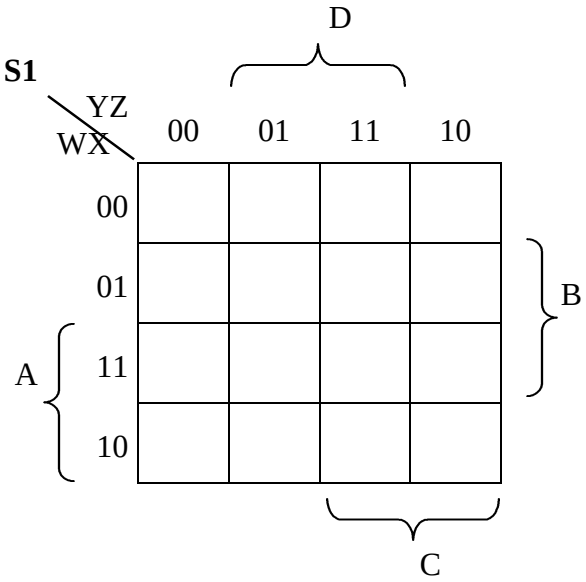
Cobertura 2:



Cobertura 3:



$$S1(W,X,Y,Z) = \sum(0,1,2,5,8.9.10)$$



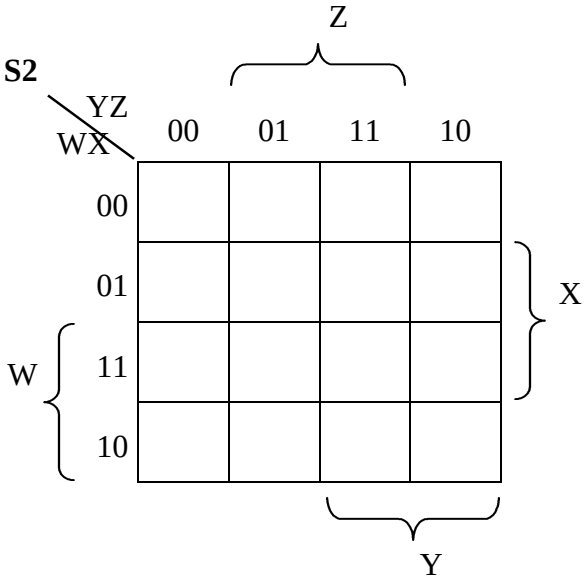
Conforme já mencionado anteriormente, também é possível obter-se uma expressão mínima em **produto de somas** a partir do mapa de Karnaugh da função Booleana. Para tanto, deve-se identificar os subcubos de mintermos-0, ao invés de subcubos de mintermos-1. Cada subcubo de mintermo-0 irá originar um termo soma, possivelmente já simplificado, o qual recebe o nome de **implicado**.

Os passos para a obtenção de uma cobertura mínima são os mesmos já descritos para a obtenção da expressão em soma de produtos.

Exemplo 2.7: determine a equação mínima em **produto de somas** para a função Booleana.

$$S2(W,X,Y,Z) = \sum(0,1,2,5,8,9,10)$$

Observação importante: repare que a função foi especificada pela descrição de seus mintermos-1. Mas como foi solicitada a expressão em produto de somas, uma vez montado o mapa de Karnaugh usando a informação fornecida, passaremos a identificar os subcubos de mintermos-0.

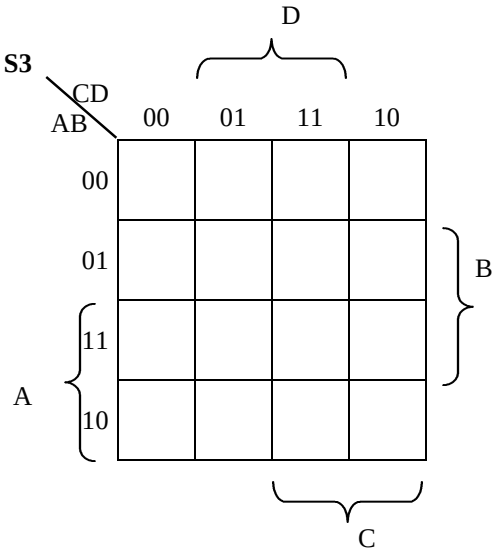


Exemplo 2.8: determinar a expressão mínima em **soma de produtos** e a expressão mínima em produto de somas para a função Booleana dada a seguir. Desenhar o circuito lógico para cada expressão obtida.

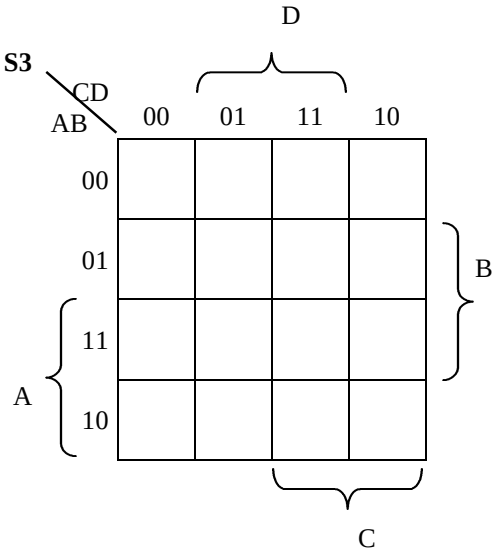
$$S3(A,B,C,D) = \prod(1,2,3,6,7,8,9,12,14)$$

Obs: existe mais de uma cobertura mínima possível para essa função.

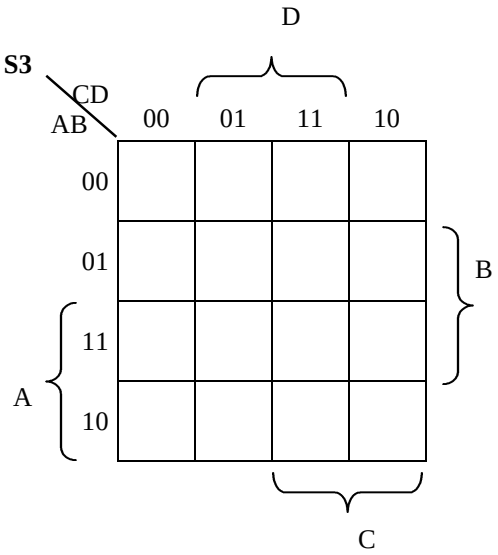
Cobertura 1 para soma de produtos:



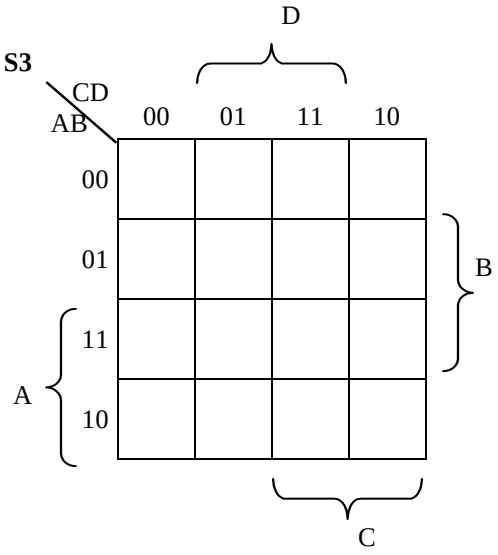
Cobertura 2 para soma de produtos:



Cobertura 1 para produto de somas:



Cobertura 2 para produto de somas:



2.9 Funções Incompletamente Especificadas

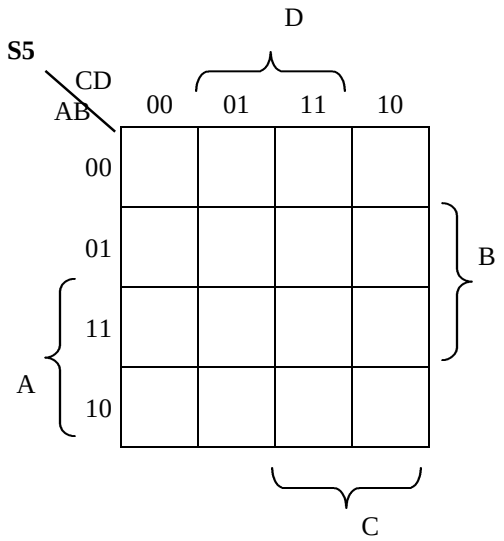
Durante o projeto lógico, pode ocorrer que algumas condições (combinações) de entradas de uma função Booleana não sejam especificadas, por não afetarem a função no contexto do projeto. Cada condição de entrada cujo valor não é especificado é sinalizada com "DC" ou com "X" na tabela verdade (ou no mapa de Karnaugh) e são denominadas condições de ***don't care*** (ou simplesmente, ***don't care***).

Quando da simplificação de uma função Booleana que contenha condições de ***don't care***, essas condições podem e devem ser exploradas no sentido de se obter a máxima simplificação possível. Para tanto, pode-se assumir cada condição de ***don't care*** como valendo 0 ou 1, independente das demais condições de ***don't care***. A escolha do valor (0 ou 1) irá depender do contexto, mas sempre deverá ocorrer com o objetivo de levar a uma simplificação máxima da função Booleana.

Exemplo 2.9: determinar a expressão mínima em soma de produtos para a função.

$$S5(A,B,C,D) = \sum(0,1,2,12,13) + DC(3,7,10,11,14,15)$$

onde o conjunto de condições de *don't care* está especificado por "DC" Repare que no caso de uma função que contenha condições de *don't care* (ou seja, uma função incompletamente especificada), não basta dizer somente onde estão os uns (ou somente os zeros) da função: é necessário informar também quais condições de entrada não são especificadas, o que é feito pelo conjunto "DC".



Observações:

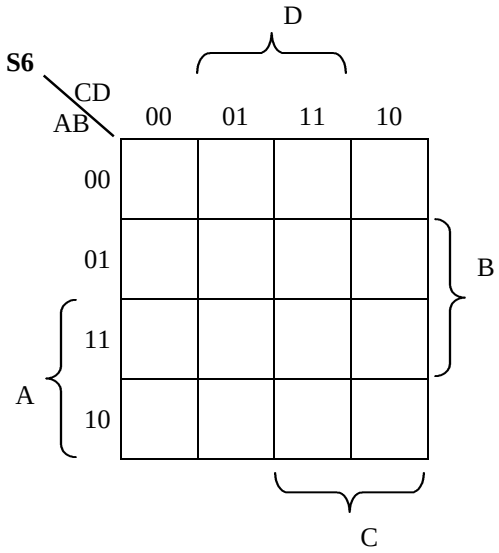
- Iremos identificar cada condição de entrada não especificadas (ou *don't care*) por meio de um "X" (xis maiúsculo). O valor que será assumido para um *don't care* **é totalmente independente dos demais** *don't cares;*

- Quando da identificação dos subcubos, **não é aconselhável** escrever o valor que foi assumido para um determinado *don't care*, pois isso pode gerar confusões. O simples fato de um *don't care* pertencer (ou não) a algum subcubo já identifica o valor que foi assumido para ele;
- Para uma mesma função Booleana, deve-se identificar a solução em soma de produtos num mapa de Karnaugh e a solução em produto de somas em outro mapa, a fim de evitar confusões

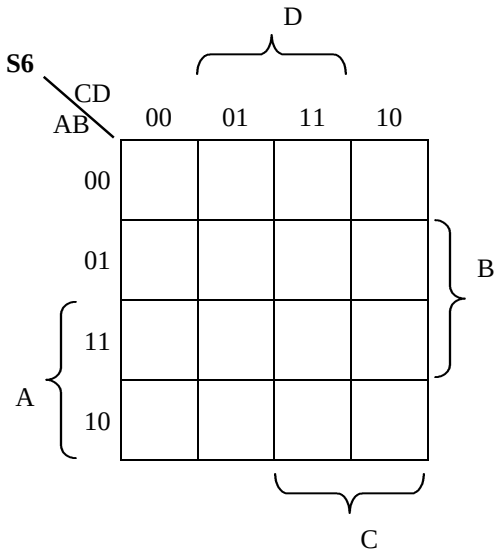
Exemplo 2.10:. dada a função que segue, determinar as equações mínimas em SdP (soma de produtos) e em PdS (produto de somas) para a função dada. Desenhar os respectivos circuitos lógicos e identificar, o circuito de menor complexidade (entre SdP e PdS).

$$S6(A,B,C,D) = \sum(0,3,5,6,7) + DC(10,11,12,13,14,15)$$

Cobertura para soma de produtos:



Cobertura para produto de somas:



Observação: um mesmo *don't care* pode assumir um valor lógico para a cobertura em SdP e outro valor lógico para a cobertura em PdS, pois as coberturas são totalmente independentes.

2.10 Outras Operações Lógicas

Até agora, tem-se visto apenas duas funções booleanas de duas variáveis, OU e E. Mas, existem 2^{2^n} funções Booleanas com n variáveis binárias. Assim, existem 16 funções Booleanas de duas variáveis e as funções E e OU são apenas duas dessas 16 funções. A tabela a seguir lista todas as 16 funções Booleanas de duas variáveis, x e y.

Nome	Símbolo	Valores para x,y= 00 01 10 11	Expressão algébrica	Comentário
Zero	**0**	0 0 0 0	$F_0 = 0$	Constante 0
E	x.y	0 0 0 1	$F_1 = xy$	x e y
Inibição	x/y	0 0 1 0	$F_2 = x\bar{y}$	x mas não y
Transferência		0 0 1 1	$F_3 = x$	x
Inibição	y/x	0 1 0 0	$F_4 = \bar{x}y$	y mas não x
Transferência		0 1 0 1	$F_5 = x$	y
XOR	x⊕y	0 1 1 0	$F_6 = x\bar{y} + \bar{x}y$	x ou y, mas não ambos
OU	x+y	0 1 1 1	$F_7 = x + y$	x ou y
NOR	x↓y	1 0 0 0	$F_8 = (x + y)'$	Não-OU
Equivalência		1 0 0 1	$F_9 = xy + x'\,y'$	x igual a y
Complemento	y'	1 0 1 0	$F_{10} = y'$	Não y
Implicação	x⊂y	1 0 1 1	$F_{11} = x + \bar{y}$	Se y, então x
Complemento	x'	1 1 0 0	$F_{12} = \bar{x}$	Não x
Implicação	x⊃y	1 1 0 1	$F_{13} = \bar{x} + y$	Se x, então y
NAND	x↑y	1 1 1 0	$F_{14} = (xy)'$	Não E
Um	**1**	1 1 1 1	$F_{15} = 1$	Constante 1

Cada linha na tabela acima representa uma função de duas variáveis e os nomes destas funções são dados na primeira coluna. As tabelas verdades de cada uma das funções são dadas na terceira coluna. Note que cada função foi designada por F_i onde i é o decimal equivalente ao número binário que é obtido interpretando-se os valores das funções dados na terceira coluna como números binários.

Cada uma destas funções pode ser dada em termos de operações E, OU e complemento. Como pode ser visto na tabela, existem duas funções constantes, **0** e **1**, que retornam sempre 0 ou 1, respectivamente, independente de que valores se tem na entrada. Existem quatro funções de uma variável, que representam o complemento e a transferência. E existem 10 funções que definem oito operações binárias: E, inibição, XOR, OU, NOR,

equivalência, implicação e NAND. Sendo que inibição e implicação nunca são usados no projeto de circuitos, principalmente, por poderem ser facilmente implementados usando-se operações E, OU e complemento. A função NOR é o complemento da operação OU e a função NAND é o complemento da operação E. Pode-se notar também que XOR e equivalência são o complemento uma da outra e, por esta razão, a equivalência é normalmente chamada XNOR.

Para implementar as funções mostradas na tabela acima, são usadas geralmente oito tipos de portas lógicas: portas E, OU, complemento ou inversora, portas de transferência ou driver, portas NAND, XOR e XNOR. A figura 2.15 mostra os símbolos de cada uma destas portas com respectivos atrasos (em nanossegundos) e custo (em número de transistores).

Figura 2.15 - Símbolos para portas lógicas com respectivos atrasos e custos

Na figura acima, os números dentro do símbolo de cada porta lógica indica o atraso através da porta, ou seja, o tempo que leva para que o resultado da operação lógica correspondente apareça na saída da porta, quando se aplica valores na entradas. Assim, por exemplo, a porta NOR possui um atraso de propagação de 1,4 ns. O número ao lado do símbolo de uma porta lógica indica a quantidade de transistores necessários para implementá-la. Assim, pode-se notar que a implementação de uma porta XOR (14 transistores) é mais "cara" que a implementação de uma porta XNOR (12 transistores). Evidentemente, para portas lógicas com múltplas entradas (mais de duas entradas) estes valores são diferentes. A figura 2.16 mostra portas lógicas com múltiplas entradas.

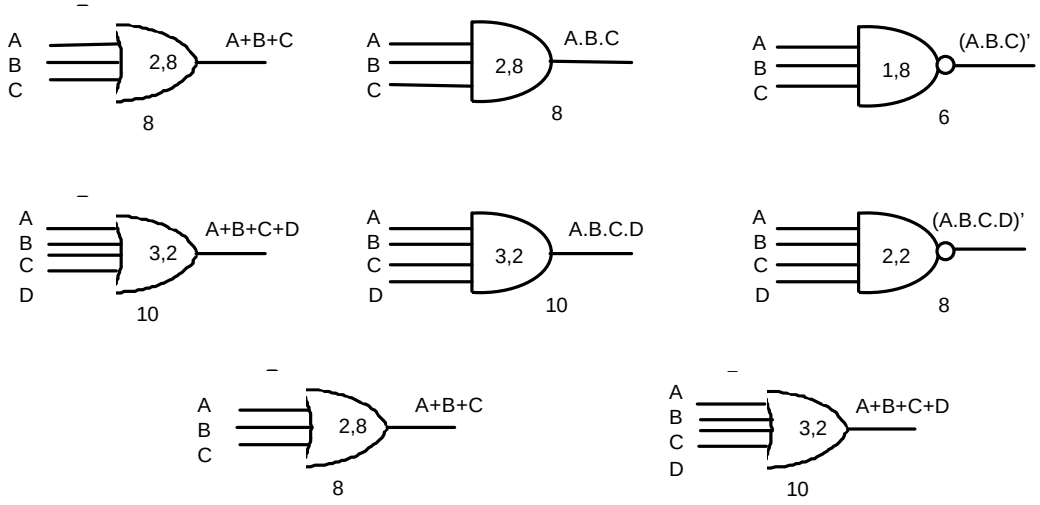


Figura 2.16 - Portas lógicas com múltiplas entradas

Obviamente, portas lógicas com múltiplas entradas são mais difíceis de contruir e como não são muito utilizadas, o custo da manutenção de bibliotecas contendo tais portas e das ferramentas de projeto tende a ser maior que a utilidade destas portas. Por isso, em geral, se tem somente portas de três e quatro entradas em bibliotecas de portas lógicas.

Quando se está implementando funções booleanas usando estas portas lógicas, usualmente tenta-se encontrar uma expressão booleana que melhor satisfaça um conjunto de requisitos de projeto. Geralmente, estes requisitos envolvem restrições de custo, que pode ser dado em termos de número de transistores e/ou atraso dos sinais ao longo do circuito.

Muitas vezes se está interessado em obter o circuito mais rápido, ou seja, o que tenha o menor atraso a partir das entradas para a saída, mas também existem casos em que se deseja o circuito mais barato, ou seja, com o menor número de transistores. Por sua natureza, estes dois objetivos são conflitantes, pois um circuito rápido avalia subexpressões em paralelo, o que vai exigir mais portas do que um circuito de baixo, em que subexpressões são fatoradas e executadas serialmente.

2.11 Mapeamento

No projeto de circuitos, tem-se frequentemente o seguinte problema: dada uma função booleana (mínima ou não), deve-se projetar um circuito lógico usando somente alguns poucos tipos de portas que estão disponíveis. Esta ação de adaptar-se a lógica de uma função ao conjunto de portas que está disponível é denominada **mapeamento tecnológico** (ou simplesmente **mapeamento**) da função. O conjunto de portas disponível é denominado **biblioteca de portas**, e depende de vários fatores, dentre os quais, da tecnologia em que o circuito será implementado.

Assim, o problema de mapeamento pode ser formulado da seguinte maneira: dada uma função booleana simplificada, em SdP ou em PdS, ou dado um circuito para essa função, e dada uma biblioteca de portas, mapear a função usando somente as portas (tipos de portas) que existem na biblioteca.

Este processo envolve várias tarefas. Primeiro será mostrado como converter uma função ou circuito lógico envolvendo portas E, OU e complemento em uma função ou circuito lógico contendo apenas portas NAND (ou somente portas NOR). Esta tarefa consiste de duas partes: conversão e otimização. Durante a conversão, cada porta E e porta OU é substituída por uma porta NAND ou NOR equivalente, enquanto que durante a otimização, são eliminados quaisquer dois inversores, um seguido do outro, que possam ter surgido durante a conversão.

As regras para a tarefa de conversão são baseadas nos Teoremas de De Morgan, como mostrado a seguir:

Regra 1: $xy = ((xy)')'$

Regra 2: $x + y = ((x + y)')' = (x'\, y')'$

Regra 3: $xy = ((xy)')' = (x' + y')'$

Regra 4: $x + y = ((x + y)')'$

Regra 5: $(x')' = x$

As regras 1 e 2 são usadas para a conversão para portas NAND, enquanto as regras 3 e 4 são usadas para a conversão para portas NOR.. A regra 5 é usada para a otimização do

circuito e permite eliminar quaisquer dois inversores que aparecem um seguido do outro no circuito. A figura 2.17 ilustra graficamente as regras dadas acima.

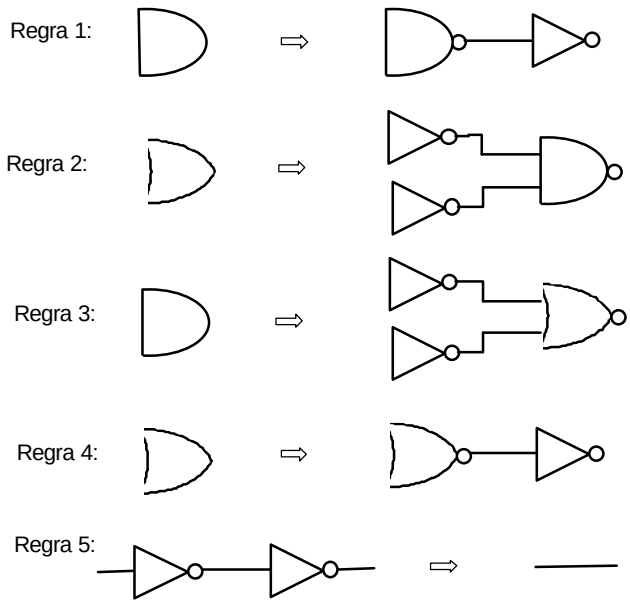


Figura 2.17 - Regras de conversão

Exemplo 2.11: encontrar uma implementação usando apenas portas NAND e outra implementação usando apenas portas NOR para a função de carry $c_{i+1} = x_i y_i + x_i c_i + y_i c_i$ ou $c_{i+1} = (x_i + y_i)(x_i + c_i)(y_i + c_i)$.

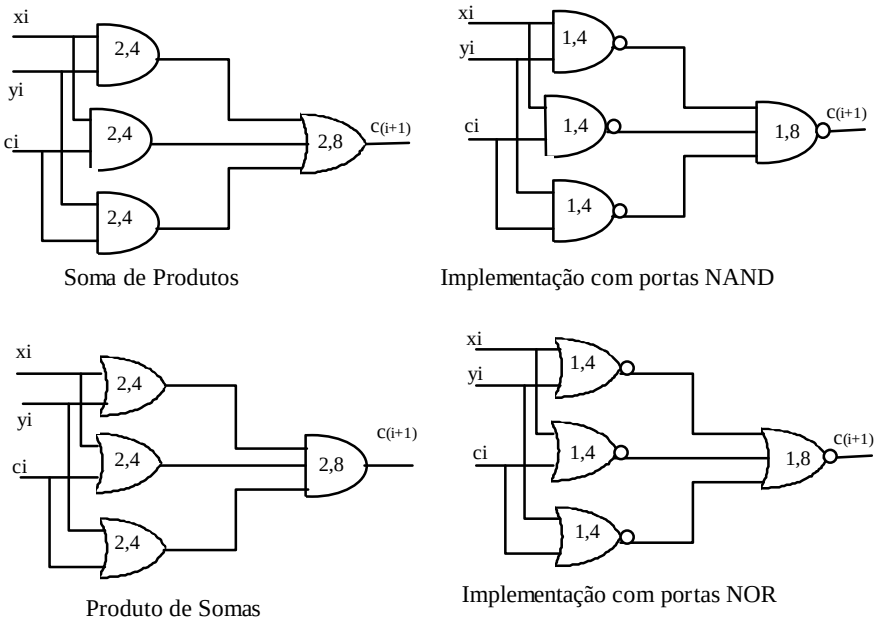


Figura 2.18 – soma de produtos e produto de somas com portas NAND e portas NOR, respectivamente.

2.12 Comportamento Dinâmico de Circuitos Combinacionais

Até o presente, analisamos somente o comportamento estático dos circuitos combinacionais. Ou seja, dada uma combinação de valores aplicados às entradas do circuito há um tempo suficientemente longo, determinamos os valores de suas saídas. Entretanto, durante o funcionamento normal e um circuito, as entradas podem estar mudando com uma determinada freqüência, como conseqüência da aplicação sucessiva de diferentes conjuntos de dados. Então, desde que o circuito tenha sido projetado para funcionar nesta freqüência, os valores das saídas também mudarão (com a mesma freqüência, na pior das hipóteses), como conseqüência das mudanças das entradas.

A partir de agora, estaremos assumindo que uma variável Booleana pode se alterar ao longo do tempo, porém sempre assumindo um valor no intervalo [0;1]. Assim, a representação gráfica de uma variável Booleana ao longo de um intervalo de tempo é denominada **forma de onda** da variável. A figura 3.13 mostra um exemplo de forma de onda qualquer. Repare que uma forma de onda pode ser imaginada sobre o plano cartesiano, onde o eixo dos **x** representa o tempo (crescente da esquerda para a direita) e o eixo dos **y** representa o valor lógico da variável (sempre dentro do intervalo [0;1]).

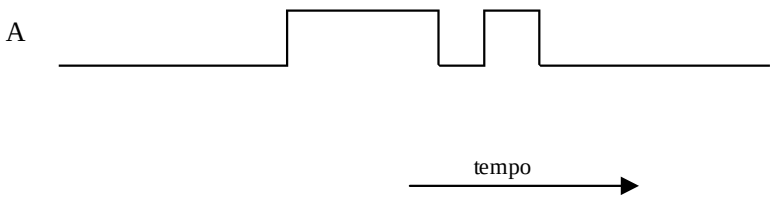


Figura 2.19 - exemplo de forma de onda.

Quando uma variável Booleana se modifica ao longo do tempo, ela costuma ser chamada de **sinal**. Portanto, na análise temporal (também conhecida como análise de *timing*) de um circuito, pode-se associar um sinal a cada entrada do circuito e a cada saída de cada porta do circuito (o conjunto de entradas e de saídas de portas de um circuito é chamado **nós** ou **nodos** do circuito.) Normalmente, os sinais referentes às entradas de um circuito são dados.

As portas lógicas são fabricadas com material semicondutor (silício). Apesar das reduzidas dimensões que a tecnologia atual permite que sejam alcançadas, as portas lógicas não conseguem responder de maneira instantânea às variações em suas entradas. Ao tempo que decorre entre alguma das entradas de uma porta se modificar e essa modificação se propagar até a saída, dá-se o nome de **atraso** (da porta lógica). O atraso de uma porta P normalmente é representado por td(P) ou td_P ou tp(P) ou ainda tp_P. É importante ressaltar que cada porta lógica pode apresentar um atraso diferente, mesmo em se tratando de portas de mesma função e mesmo número de entradas. O valor do atraso de uma porta depende de vários fatores, dentre eles a tecnologia de fabricação, as dimensões dos transistores que a compõem, a temperatura do local de operação etc.

Exemplo 2.11: determinar o sinal de saída S da porta que segue, a partir dos sinais A e B aplicados às suas entradas.

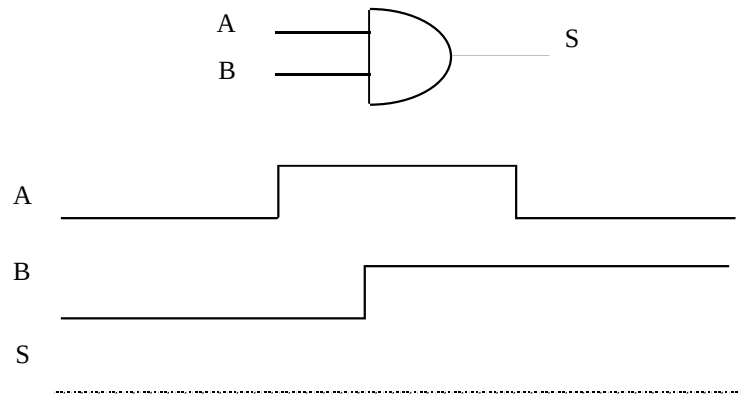
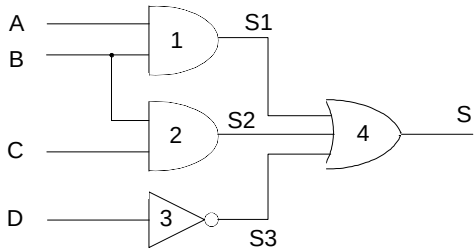


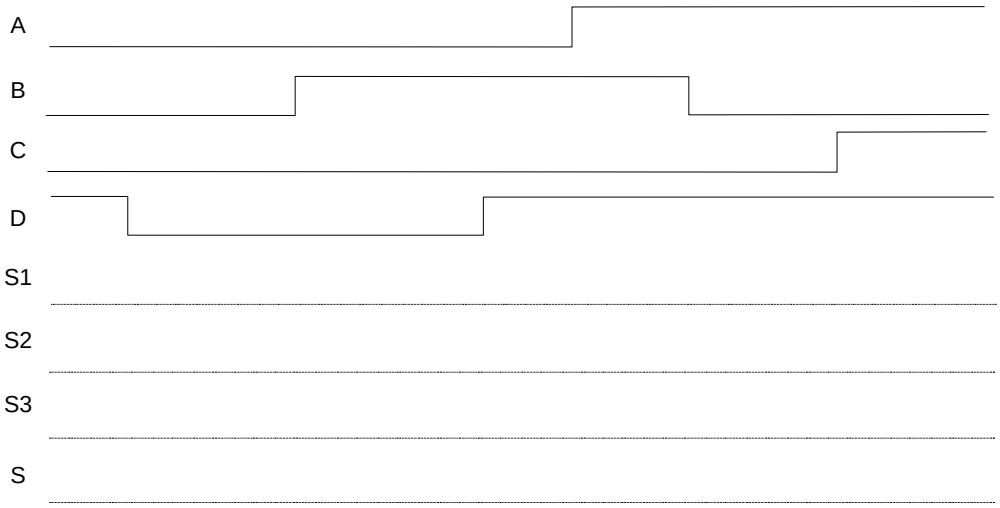
Figura 2.20 - exemplo de determinação da forma de onda da saída de uma porta.

Na análise temporal de um circuito combinacional, é prudente seguirem-se os seguintes passos:

- identificar com uma linha vertical cada mudança no valor (=transição) dos sinais das entradas. Note que as formas de onda das entradas sempre são fornecidas, fazendo parte dos dados do problema.
- Para cada intervalo de tempo delimitado por duas linhas verticais adjacentes, identificar o sinal resultante usando a tabela verdade da porta em questão;
- Considerar um pequeno intervalo de tempo após cada transição de entrada, referente ao atraso de propagação da porta.

Exemplo 2.11: determine as formas de onda e os atrasos envolvidos para S1, S2, S3 e S no circuito que segue, a partir das entradas fornecidas. Considere que as portas 1 e 2 têm o mesmo valor para o atraso individual. Identifique o atraso total do circuito, indicando as parcelas que o compõem.


A
B
C
D
1
2
3
4
S1
S2
S3
S


A
B
C
D
S1
S2
S3
S

Exercícios

Exercício 2.1 - Dada a equação abaixo (que está na forma fatorada), use as propriedades da álgebra Booleana para encontrar a equação mínima em soma de produtos.

$$S = Z \cdot (X + \overline{X} \cdot \overline{Y})$$

Exercício 2.2 - Ache a equação em **soma de mintermos** e a equação em **produto de maxtermos** para as funções **F1** e **F2** descritas pela tabela verdade abaixo e desenhe os respectivos circuitos lógicos

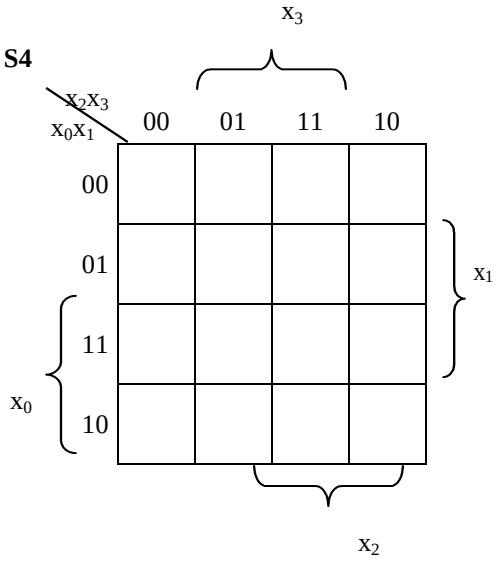
A B C	F1	F2
0 0 0	1	0
0 0 1	0	1
0 1 0	0	0
0 1 1	1	1
1 0 0	1	0
1 0 1	1	0
1 1 0	0	0
1 1 1	0	1

Exercício 2.3 - Reescreva as equações do exercício 2, usando a notação compacta.

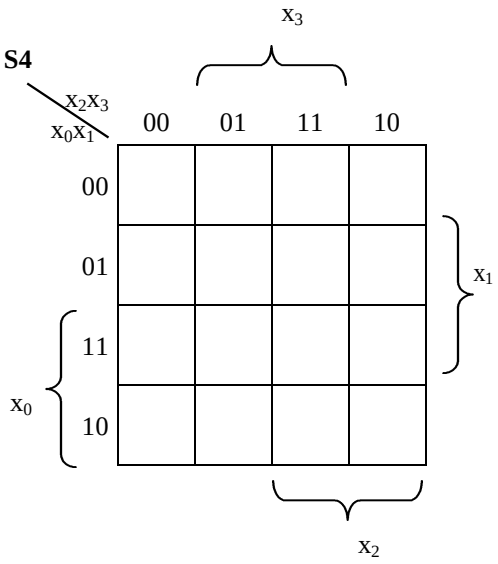
Exercício 2.4 - determinar a expressão mínima em **soma de produtos** e a expressão mínima em **produto de somas** para a função Booleana dada a seguir. Desenhar o circuito lógico para cada expressão obtida.

$$S4(x_0, x_1, x_2, x_3) = \sum(1,3,5,7,8,9,11,13)$$

Cobertura para soma de produtos:



Cobertura para produto de Somas:

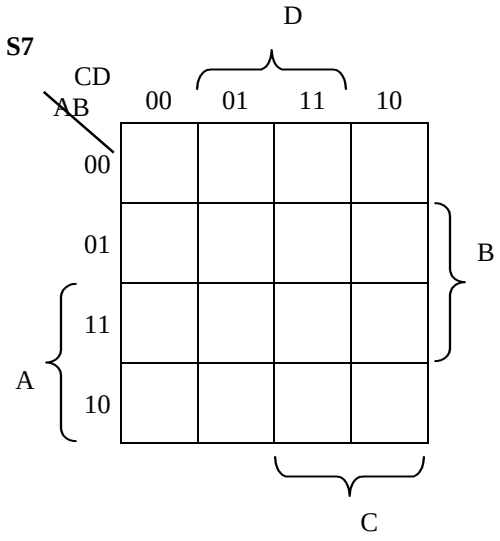


Exemplo 2.11:. dada a função que segue, determinar as equações mínimas em SdP (soma de produtos) e em PdS (produto de somas) para a função dada. Desenhar os respectivos circuitos lógicos e identificar, o circuito de menor complexidade (entre SdP e PdS).

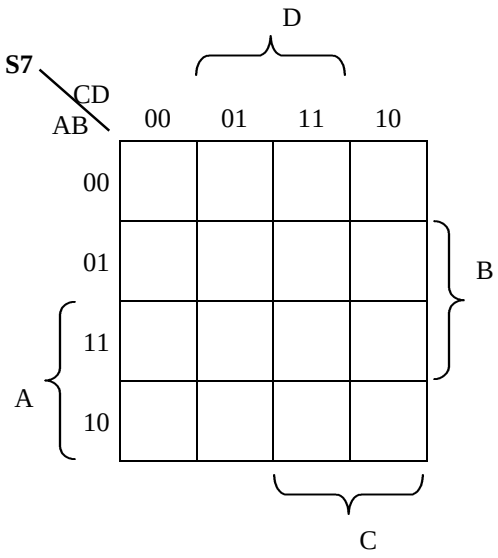
$$S7(A,B,C,D) = \prod(4,6,8,9,12,13,14) + DC(0,2,5,10)$$

Observação: neste caso, há duas coberturas mínimas para SdP. Determine cada uma nos dois primeiros mapas.

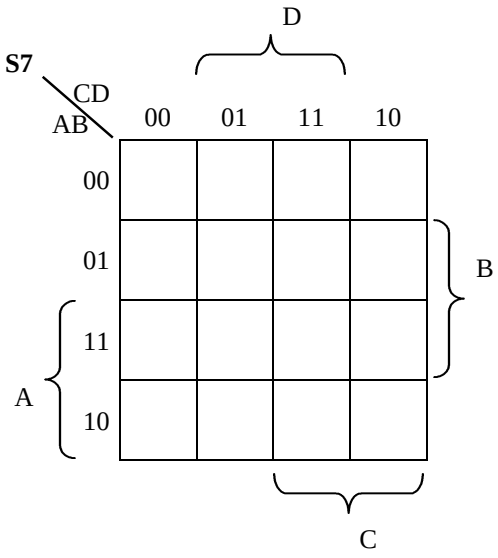
Cobertura 1 para soma de produtos:



Cobertura 2 para soma de produtos:



Cobertura 1 para produto de somas:



Bibliografia Suplementar

[1] GAJSKI, Daniel D. **Principles of Digital Design**, New Jersey: Prentice Hall, 1997 (ISBN 0-13-301144-5)

[2] MANO, M. Morris; **Computer Engineering: Hardware Design.** New Jersey: Prentice Hall, 1988 (ISBN 0-13-162926-3)

[3] TAUB, H. **Circuitos Digitais e Microprocessadores.** McGraw-Hill, 1982.

[4] BROWN, Stephen; VRANESIC, Zvonko. **Fundamentals of Digital Logic with VHDL Design**, McGraw-Hill Higher Education (a McGraw-Hill Company), 2000 (ISBN texto: 0-07-012591-0 CD parte da coleção: 0-07-235596-4)

[5] ERCEGOVAC, Milos; LANG, Tomás; MORENO, Jaime H. **Introdução aos Sistemas Digitais.** Porto Alegre: Bookman, 2000 (ISBN: 85-7307-698-4)

[6] KATZ, Randy H. **Contemporary Logic Design.** The Benjamin/Cummings Publishing Company, Inc. , 1994 (ISBN: 0-8053-2703-7)