Capítulo 3

Análise e Síntese de Circuitos Lógicos Combinacionais

Os circuitos lógicos são classificados em duas categorias:

Circuitos lógicos combinacionais: as saídas dependem somente das entradas **correntes**.

Circuitos lógicos sequenciais: as saídas dependem não só das entradas **correntes** como também da sequência de entradas passadas.

Veremos que um circuito sequencial se caracteriza por conter, no mínimo, um **laço de realimentação**, através do qual é estabelecido um caminho entre a saída e a entrada de uma porta.

Usualmente a modelagem de um problema é a parte mais desafiante, requerendo muitas vezes a "criatividade" do(s) projetistas. Uma vez definidas as variáveis lógicas e as suas relações funcionais, o **diagrama lógico** do circuito pode ser obtido de forma sistemática. Ao processo de obtenção de um ou mais diagramas lógicos a partir de uma descrição não-formal denominamos de **síntese** de circuito. O inverso deste processo é conhecido como **análise** de circuito.

Neste capítulo nós nos ocuparemos com análise e síntese de um circuito combinacional.

3.1 Descrição Formal

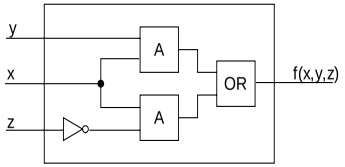
Representações mais utilizadas no projeto de circuitos combinacionais para descrever as relações entre as variáveis lógicas são

1. as **expressões algébricas** (booleanas), obtidas a partir das palavras ligadas pelos conectivos lógicos, e

   Exemplo 3.1 *A afirmação: "A máquina (m) só opera, se o técnico (t) e um dos operadores (a ou b) estiverem presentes." pode ser expressa através da equação: $m = t \,.\, (a+b)$.*

2. pelos axiomas básicos da álgebra booleana, para um ciruito de n variáveis lógicas de entrada existem 2^n possíveis combinações destas variáveis. O resultado destas 2^n combinações pode ser sintetizado numa **tabela-verdade**.

   Exemplo 3.2 *Para o seguinte circuito*

   o resultado das possíveis combinações das entradas x, y e z pode ser sintetizado na seguinte tabela-verdade:

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

3.2 Terminologia e notações

As informações contidas numa tabela-verdade podem ser expressas algebricamente.

Por conveniência, vamos introduzir algumas terminologias:

Literal: é uma variável lógica ou o seu complemento.

Termo de produto: é dois ou mais literais ligados pelo conectivo lógico AND ou um simples literal.

Soma de produtos: dois ou mais termos de produto ligados pelo conectivo OR.

Termo de soma: é dois ou mais literais ligados pelo conectivo lógico OR ou um simples literal.

Produto de somas: dois ou mais termos de somas ligados pelo conectivo AND.

Termo normal: é um termo de produto ou de soma no qual nenhuma variável (complementada ou não) aparece mais de uma vez.

Mintermo (de n variáveis): é um termo de produto normal com n literais. Uma função lógica com n variáveis possui 2^n mintermos.

Maxtermo (de n variáveis): é um termo de soma normal com n literais. Uma função lógica com n variáveis possui 2^n mintermos.

Existe uma relação interessante entre tabelas-verdade, mintermos e maxtermos. Repare nas tabelas-verdade de mintermos envolvendo 2 variáveis

	x	y	xy	xy'	$x'y$	$x'y'$
0	0	0	0	0	0	1
1	0	1	0	0	1	0
2	1	0	0	1	0	0
3	1	1	1	0	0	0

e 3 variáveis

	x	y	z	xyz	xyz'	$xy'z$	$xy'z'$	$x'yz$	$x'yz'$	$x'y'z$	$x'y'z'$
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	1	0	0
3	0	1	1	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0
5	1	0	1	0	0	1	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0	0	0	0

A tabela-verdade de um mintermo só tem exatamente uma linha igual a 1 e a do maxtermo só tem exatamente uma linha igual a 0, como ilustra as duas tabelas seguintes:

	x	y	$x+y$	$x+y'$	$x'+y$	$x'+y'$
0	0	0	0	1	1	1
1	0	1	1	0	1	1
2	1	0	1	1	0	1
3	1	1	1	1	1	0

	x	y	z	$x'y'z'$	$x'y'z$	$x'yz'$	$x'yz$	$xy'z'$	$xy'z$	xyz'	$x+y+z$
0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	1	1	0	1
2	0	1	0	1	1	1	1	1	0	1	1
3	0	1	1	1	1	1	1	0	1	1	1
4	1	0	0	1	1	1	0	1	1	1	1
5	1	0	1	1	1	0	1	1	1	1	1
6	1	1	0	1	0	1	1	1	1	1	1
7	1	1	1	0	1	1	1	1	1	1	1

Portanto, podemos traduzir a função lógica expressa por uma tabela-verdade como uma soma dos mintermos (combinação OR dos mintermos correspondentes às linhas da tabela-verdade nas quais a função vale 1), denominada a **soma canônica** da função, ou como um produto dos maxtermos (combinação AND dos maxtermos correspondentes às linhas da tabela-verdade nas quais a função vale 0), denominado **produto canônico** da função.

Vale ressaltar ainda que o dual de um mintermo é um maxtermo, e vice-versa!

Uma notação concisa para designar somas canônicas faz uso dos **números de mintermo**. Cada linha da tabela-verdade é associada biunivocamente

a um número de mintermo. E cada dígito da representação binária de um **número de mintermo** é associado a um literal do mintermo. Portanto, a quantidade de dígitos binários que aparecem em cada número é igual à quantidade de variáveis envolvidas em cada função. E este dígito é igual a 0 se a variável correspondente é complementada; senão ele é igual a 1.

Exemplo 3.3 *A notação*

$$\sum_{x,y,z}(4,6,7)$$

também representa a soma canônica da função $f(x,y,z)$ do exemplo 3.2.

Analogamente, uma notação concisa para designar produtos canônicos faz uso dos **números de maxtermo**. Cada linha da tabela-verdade é associada biunivocamente a um número de maxtermo. E cada dígito da representação binária de um **número de maxtermo** é associado a um literal do maxtermo. Exatamente oposto a mintermos, o dígito é igual a 1 se a variável é complementada; senão ele é igual a 0.

Exemplo 3.4 *A notação*

$$\prod_{x,y,z}(0,1,2,3,5)$$

também representa o produto canônico da função $f(x,y,z)$ do exemplo 3.2.

3.3 Análise de um Circuito Combinacional

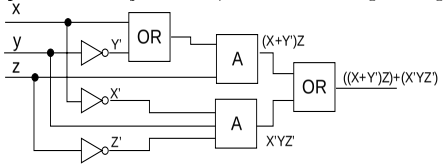
A descrição formal das funções lógicas de um circuito facilita não só prever o comportamento do circuito como obter circuitos alternativos.

Como já comentamos, dado um circuito fechado de n entradas, podemos obter a sua tabela-verdade de forma exaustiva, determinando a saída para as 2^n possíveis combinações dos valores de entrada. Como as combinações de entrada crescem exponencialmente, esta técnica só se aplica para um número pequeno de entradas.

Pelo que vimos na seção anterior, podemos ainda facilmente obter a partir do conteúdo de uma tabela-verdade uma expressão algébrica em forma canônica com uso de mintermos ou maxtermos.

Usualmente, a técnica algébrica é a preferida para obter uma expressão algébrica. Sua complexidade tende a ser linear em relação ao número de entradas. Esta técnica, porém, só pode ser utilizada quando a estrutura

do circuito, por exemplo o seu diagrama lógico, é conhecida. Neste caso, iniciamos a análise a partir das entradas e propagamos a expressão lógica através das portas em direção à saída, como ilustra a seguinte figura.



3.4 Síntese de um Circuito Combinacional

Usualmente um projeto de um circuito combinacional se inicia com uma descrição não-formal de um problema. Se esta descrição for uma lista de possíveis combinações de entradas e suas respectivas saídas esperadas, podemos facilmente transcrevê-la numa tabela-verdade e obter uma expressão algébrica em forma canônica. E, a partir desta expressão, pode-se obter um diagrama lógico com uso de portas ANDs, ORs e/ou NOTs.

No entanto, uma implementação a partir de um diagrama obtido diretamente de uma expressão em forma canônica tem um custo comparativamente elevado. Métodos de minimização são aplicados para reduzir o número de termos de produto/soma nas expressões lógicas. Nas próximas seções veremos algumas técnicas de minimização que são utilizadas na síntese de um circuito combinacional.

3.5 Minimizações

A partir das formas canônicas de uma função lógica, pode-se manipulá-la para otimizar e/ou baratear a implementação do circuito. Os métodos de minimização mais conhecidos fazem uso da propriedade de absorção, para a qual

$$\begin{aligned}(\text{termo de produto})Y + (\text{termo de produto})Y\prime &= \text{termo de produto}\\ (\text{termo de soma} + Y)(\text{termo de soma} + Y\prime) &= \text{termo de soma}\end{aligned}$$

3.5.1 Minimizações Algébricas

Aplicando os teoremas que vimos no capítulo 2, podemos reduzir os literais e/ou os termos envolvidos numa função lógica e, consequentemente, baratear o seu custo de implementação.

Exemplo 3.5 *Após algumas manipulações algébricas, a expressão do exemplo 3.3 se reduz em:*

$$xy'z' + xyz' + xyz = xz'(y + y') + xyz = xz' + xyz = x(z' + yz') = xz' + xy$$

e a expressão do exemplo 3.4:

$$(x + y + z)(x + y + z')(x + y' + z)(x + y' + z')(x' + y + z') = x(y + z')$$

Certos passos de simplificações algébricas não são triviais, técnicas mais intuitivas e diretas, como mapas de Karnaugh, ou mais "computacionais", como Quine-McCluskey, foram desenvolvidas.

3.5.2 Minimizações Gráficas: Mapas de Karnaugh

Um **mapa de Karnaugh** é um diagrama de Venn particular para representação de funções lógicas, no qual mintermos/maxtermos adjacentes (aqueles que se distinguem em uma única variável) são geograficamente "vizinhos".

Exemplo 3.6 *Exemplos de mapas de Karnaugh com 2, 3 e 4 variáveis. Note-se que foram utilizados os números de mintermo para designar o mintermo correspondente a cada célula do mapa.*

y/x	*0*	*1*
0	*0*	*2*
1	*1*	*3*

z/xy	*00*	*01*	*11*	*10*
0	*0*	*2*	*6*	*4*
1	*1*	*3*	*7*	*5*

zw/xy	*00*	*01*	*11*	*10*
00	*0*	*4*	*12*	*8*
01	*1*	*5*	*13*	*9*
11	*3*	*7*	*15*	*11*
10	*2*	*6*	*14*	*10*

Nós veremos que, pelos teoremas $(x \cdot y') + (x \cdot y) = x$ e $(x + y') \cdot (x + y) = x$, podemos utilizar o mapa de Karnaugh para simplificar as formas canônicas.

Definição 3.1 *Uma* **soma minimal** *de uma função é uma soma que contém o menor número de termos de produto e menor número de literais nestes termos.*

Definição 3.2 Teorema de Implicante Primo*: Uma soma minimal é uma soma de implicantes primos.*

Definição 3.3 *Um* **implicante** *de uma função é um termo de produto normal que implica a função, isto é, para toda ocorrência das variáveis nas quais o implicante é igual a 1, a função é 1.*

Definição 3.4 *Um* **implicante de ordem** *n é um implicante que corresponde a um grupo de 2^n células no mapa de Karnaugh.*

Exemplo 3.7 *Um mintermo correspondente à linha da tabela-verdade de uma função que assume o valor 1 é um implicante dessa função. Eles são denominados* **implicantes de ordem 0**.

Definição 3.5 *Um* **implicante primo** *de uma função é um implicante tal que a remoção de qualquer uma variável nele não implicará mais a função. Ou seja, ele não será mais um implicante. Em termos de mapas de Karnaugh, isso significa agrupar maximamente um conjunto de células que contém valor 1.*

Definição 3.6 *Um* **implicante primo essencial** *é um implicante primo que cobre um mintermo não coberto por nenhum outro implicante primo. São essenciais porque eles devem aparecer na soma minimal.*

Exemplo 3.8 *Através do mapa de Karnaugh da função $s(x, y, z) = \sum_{xyz}(3, 4, 6, 7)$, note-se que o implicante primo xy não é essencial*

z/xy	*00*	*01*	*11*	*10*
0			*1*	*1*
1		*1*	*1*	

Exemplo 3.9 *Através do mapa de Karnaugh da função $s(a, b, c, d) = \sum_{abcd}(1, 5, 8, 9, 10, 11, 13)$, note-se que todos os implicantes primos (c'd e ab') são essenciais.*

cd/ab	*00*	*01*	*11*	*10*
00				*1*
01	*1*	*1*	*1*	*1*
11				*1*
10				*1*

Exemplo 3.10 *Através do mapa de Karnaugh da função* $s\prime(a,b,c,d) = \sum_{abcd}(0,2,3,4,6,7,12,14,15)$*, note-se que todos os implicantes primos (*$a\not c\prime$*,* $a\not c$*,* $a\not c$ *e* $b\not c$*) são essenciais.*

cd/ab	*00*	*01*	*11*	*10*
00	*1*	*1*	*1*	
01				
11	*1*	*1*	*1*	
10	*1*	*1*	*1*	

Exemplo 3.11 *Atribuindo os valores 1 ao mapa de Karnaugh seguindo a função* $s = f(a,b,c,d) = \sum_{abcd}(0,2,8,10)$ *obtemos*

cd/ab	*00*	*01*	*11*	*10*
00	*1*			*1*
01				
11				
10	*1*			*1*

Neste caso, só temos um implicante primo essencial $b\not d\prime$*.*

Exemplo 3.12 *Através do mapa de Karnaugh da função* $F(a,b,c,d) = \sum_{abcd}(0,2,4,5,10,11,13,15)$ *(mapa cíclico), note-se que não há implicantes essenciais.*

cd/ab	*00*	*01*	*11*	*10*
00	*1*	*1*		
01		*1*	*1*	
11			*1*	*1*
10	*1*			*1*

Vimos que os implicantes primos essenciais devem aparecer na soma minimal. Quando nós não temos nenhum implicante primo essencial, como no exemplo 3.12, quais implicantes primos devem aparecer na soma minimal? Neste caso, a solução é por tentativas e erros. Usualmente, escolhemos aleatoriamente um implicante primo com o menor número de literais e o "batizamos" como o essencial e o incluimos na soma minimal. Este implicante primo escolhido é denominado o **implicante primo essencial secundário**. Descartamos os mintermos que ele cobre no mapa. A partir dos mintermos que sobraram, escolhemos o próximo implicante primo essencial secundário, e assim sucessivamente até processarmos todos os implicantes primos. Este procedimento é conhecido como **método de ramificação**.

Combinações de Entradas Irrelevantes

Há especificações de circuitos combinacionais para as quais certas combinações de entrada nunca ocorrem em operação normal ou podem ser ignoradas. Estas combinações de entrada são conhecidas como **combinações de entrada irrelevantes** e definem um conjunto *d-set* (*don't care-set*).

Exemplo 3.13 *Supondo o seguinte problema: "Uma comissão, composta por dois membros* x*,* y *e um presidente* p *vota decisões* $d(x,y,p)$*. O presidente sempre vota, da seguinte forma: reproduz o voto da maioria, ou vota livremente em caso de empate." Neste caso, podemos traduzí-lo na seguinte expressão:*

$$\sum_{x,y,p}(3,5,7) + d(1,6)$$

uma vez que as entradas 001 e 110 nunca devem ocorrer se o presidente for honesto, conforme a seguinte tabela

	x	y	p	d
0	*0*	*0*	*0*	*0*
1	*0*	*0*	*1*	*X*
2	*0*	*1*	*0*	*0*
3	*0*	*1*	*1*	*1* .
4	*1*	*0*	*0*	*0*
5	*1*	*0*	*1*	*1*
6	*1*	*1*	*0*	*X*
7	*1*	*1*	*1*	*1*

Certamente, se o presidente for desonesto, as duas situações correspondentes às linhas 1 e 6 poderão ocorrer e elas não poderão mais ser consideradas irrelevantes.

Exemplo 3.14 *Para um conversor de códigos BCD normal para BCD 3 em excesso, as entradas 1010, 1011, 1100, 1101, 1110 e 1111 nunca ocorrerão, como mostra a seguinte tabela-verdade.*

	b3	*b2*	*b1*	*b0*	*s3*	*s2*	*s1*	*s0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
1	*0*	*0*	*0*	*1*	*0*	*1*	*0*	*0*
2	*0*	*0*	*1*	*0*	*0*	*1*	*0*	*1*
3	*0*	*0*	*1*	*1*	*0*	*1*	*1*	*0*
4	*0*	*1*	*0*	*0*	*0*	*1*	*1*	*1*
5	*0*	*1*	*0*	*1*	*1*	*0*	*0*	*0*
6	*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*
7	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*
8	*1*	*0*	*0*	*0*	*1*	*0*	*1*	*1*
9	*1*	*0*	*0*	*1*	*1*	*1*	*0*	*0*
10	*1*	*0*	*1*	*0*	*X*	*X*	*X*	*X*
11	*1*	*0*	*1*	*1*	*X*	*X*	*X*	*X*
12	*1*	*1*	*0*	*0*	*X*	*X*	*X*	*X*
13	*1*	*1*	*0*	*1*	*X*	*X*	*X*	*X*
14	*1*	*1*	*1*	*0*	*X*	*X*	*X*	*X*
15	*1*	*1*	*1*	*1*	*X*	*X*	*X*	*X*

.

Neste caso, as fuções lógicas para cada dígito de saída são

- $s3(b3,b2,b1,b0) = \sum_{b3b2b1b0}(5,6,7,8,9) + d(10,11,12,13,14,15)$,
- $s2(b3,b2,b1,b0) = \sum_{b3b2b1b0}(1,2,3,4,9) + d(10,11,12,13,14,15)$,
- $s1(b3,b2,b1,b0) = \sum_{b3b2b1b0}(0,3,4,7,8) + d(10,11,12,13,14,15)$ *e*
- $s0(b3,b2,b1,b0) = \sum_{b3b2b1b0}(0,2,4,6,8) + d(10,11,12,13,14,15)$.

Em termos de síntese de circuitos combinacionais, as combinações de entrada irrelevantes são marcadas no mapa de Karnaugh por X ou d e elas **devem ser utilizadas** na definição de implicantes primos de forma a obter uma soma ainda menor para a função.

Um Procedimento de Síntese

Um procedimento de síntese de uma soma de produtos a partir de uma tabela-verdade segue os seguintes passos:

1. obter uma forma canônica (soma de mintermos) da função a partir da tabela-verdade,

2. preencher com 1 as células do mapa de Karnaugh correspondentes aos **mintermos** que aparecem na forma canônica,
3. agrupar (maximamente) as células com valor igual a 1 ou X em grupo de 2^n com $n \in N$ para obter os **implicantes primos**,
4. adicionar à expressão final da função a soma dos **implicantes primos essenciais** e descartá-los do mapa,
5. por tentativas e erros, selecionar os **implicantes primos essenciais secundários** e adicioná-los à expressão final da função.

Exemplo 3.15 *Vamos ilustrar este procedimento com a simplificação da função do exemplo 3.8.*

z/xy	*00*	*01*	*11*	*10*
0			*1*	*1*
1		*1*	*1*	

z/xy	*00*	*01*	*11*	*10*
0			*1*	*1*
1		*1*	*1*	

$s(x,y,z) = \sum_{xyz}(3,4,6,7)$ $\qquad$ $s(x,y,z) = yz + xz'$

z/xy	*00*	*01*	*11*	*10*
0				
1				

Exemplo 3.16 *Este exemplo ilustra uma simplificação de uma função (exemplo 3.12) que não tem implicantes primos essenciais.*

cd/ab	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10	1			1

$F(a,b,c,d) = \sum_{abcd}(0,2,4,5,10,11,13,15)$

cd/ab	00	01	11	10
00				
01		1	1	
11			1	1
10	1			1

$F(a,b,c,d) = a\prime c\prime d\prime + \cdots$

cd/ab	00	01	11	10
00				
01				
11			1	1
10	1			1

$F(a,b,c,d) = a\prime c\prime d\prime + b c\prime d + \cdots$

cd/ab	00	01	11	10
00				
01				
11				
10	1			1

$F(a,b,c,d) = a\prime c\prime d\prime + b c\prime d + acd + \cdots$

cd/ab	00	01	11	10
00				
01				
11				
10				

$F(a,b,c,d) = a\prime c\prime d\prime + b c\prime d + acd + b\prime c d\prime$

Antes de prosseguirmos, vamos definir o conceito de custo de um circuito que adotaremos nesta displina.

Definição 3.7 Custo *é a soma aritmética do número de terminais de entrada das portas lógicas AND e OR usadas na representação de uma função.*

Conhecendo a soma minimal, falta ainda verificar se o produto minimal contém mais ou menos termos para decidirmos pela de menor custo. Não há como se saber a priori se é ou não vantajosa a utilização de um ou de outro

para obter um circuito de menor custo. Assim, deve-se verificar as duas formas canônicas. Para obter a forma mínima de um produto de somas, podemos

1. minimizar a soma de mintermos da função negada e aplicar o teorema de Morgan para obter o produto mínimo, ou
2. minimizar o produto de maxtermos da função e utilizar o mesmo procedimento de redução aplicado para os mintermos, com ressalva de marcar com 1 as células do mapa que correspondem a maxtermos que aparecem na forma canônica da função e lembrar ainda que o dígito 1 corresponde à variável complementada e 0 a própria variável

Assim, um procedimento de síntese de um circuito de menor custo compreende três passos:

1. obter a soma mínima,
2. obter o produto mínimo, e
3. desenhar o diagrama lógico do circuito a partir da expressão de menor custo.

Exemplo 3.17 *Vamos ilustrar o procedimento através da simplificação do produto canônico da função do exemplo 3.8.*

+z/x+y	0+0	0+1	1+1	1+0
+0	1	1		
+1	1			1

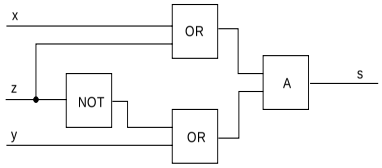
z/xy	0+0	0+1	1+1	1+0
+0	1	1		
+1	1			1

$s(x,y,z) = \prod_{xyz}(0,1,2,5)$

$s(x,y,z) = (x+z)\cdot(y+z\prime)$

z/xy	0+0	0+1	1+1	1+0
+0				
+1				

No exemplo 3.15 foi mostrado que o custo da implementação da soma de produtos ($s(x,y,z) = yz + xz\prime$) é igual a 6, que é o mesmo do produto de somas. Portanto, o diagrama lógico de um circuito "minimal" é:



Exemplo 3.18 *Vamos obter agora a expressão simplificada para a função dada no exemplo 3.11.*

- *Soma de produtos*

cd/ab	*00*	*01*	*11*	*10*
00	*1*			*1*
01				
11				
10	*1*			*1*

cd/ab	*00*	*01*	*11*	*10*
00				
01				
11				
10				

$s = f(a,b,c,d) = \sum_{abcd}(0,2,8,10)$ $\qquad f(a,b,c,d) = b\prime d\prime$

- *Produto de somas*

cd/ab	*0+0*	*0+1*	*1+1*	*1+0*
+0+0		*1*	*1*	
+0+1	*1*	*1*	*1*	*1*
+1+1	*1*	*1*	*1*	*1*
+1+0		*1*	*1*	

cd/ab	*00*	*01*	*11*	*10*
00				
01				
11				
10				

$s = f(a,b,c,d) =$
$\prod_{abcd}(1,3,4,5,6,7,9,11,12,13,14,15)$ $\qquad f(a,b,c,d) = b\prime \cdot d\prime$

Conclui-se, então, que que o custo da implementação da soma de produtos (2) é o mesmo da implementação do produto de somas (2). Portanto, neste caso, a função minimal é $f(a,b,c,d) = b\prime \cdot d\prime$.

Exemplo 3.19 *Vamos obter agora a simplificação da função do exemplo 3.13 que tem duas combinações de entradas irrelevantes.*

- *Soma de produtos*

p/xy	*00*	*01*	*11*	*10*
0			*X*	
1	*X*	*1*	*1*	*1*

p/xy	*00*	*01*	*11*	*10*
0			*X*	
1				

$d(x,y,p) = \sum_{xyp}(3,5,7) + d(1,6)$ $\qquad d(x,y,p) = p$

- *Produto de somas*

+p/x+y	*0+0*	*0+1*	*1+1*	*1+0*
+0	*1*	*1*	*X*	*1*
+1	*X*			

+p/x+y	*0+0*	*0+1*	*1+1*	*1+0*
+0				
+1				

$d(x,y,p) = \prod_{xyp}(0,2,4) + d(1,6)$ $\qquad d(x,y,p) = p$

Neste caso, o custo é igual para as duas formas, envolvendo somente uma variável. Conclui-se que a função minimal é $d(x,y,p) = p$.

Exemplo 3.20 *Vamos obter agora a simplificação da função do dígito s3 do exemplo 3.14 que tem duas combinações de 6 entradas irrelevantes.*

- *Soma de produtos*

b_1b_0/b_3b_2	*00*	*01*	*11*	*10*
00			*X*	*1*
01		*1*	*X*	*1*
11		*1*	*X*	*X*
10		*1*	*X*	*X*

b_1b_0/b_3b_2	*00*	*01*	*11*	*10*
00			*X*	*1*
01		*1*	*X*	*1*
11		*1*	*X*	*X*
10		*1*	*X*	*X*

$s3(b3,b2,b1,b0) = \sum_{b3b2b1b0}(5,6,7,8,9) +$
$+d(10,11,12,13,14,15)$ $\qquad s3(b3,b2,b1,b0) = b3 + b2b0 + b2b1$

- *Produto de somas*

$+b_1+b_0/b_3+b_2$	0+0	0+1	1+1	1+0
+0+0	1	1	X	
+0+1	1		X	
+1+1	1		X	X
+1+0	1		X	X

$s3(b3,b2,b1,b0) = \prod_{b3b2b1b0}(0,1,2,3,4)+$
$+d(10,11,12,13,14,15)$

$+b_1+b_0/b_3+b_2$	0+0	0+1	1+1	1+0
+0+0	1	1	X	
+0+1	1		X	
+1+1	1		X	X
+1+0	1		X	X

$s3(b3,b2,b1,b0) = (b3+b2)\cdot(b2\prime+b1+b0)$

Neste caso, o custo da soma de produtos (7) é também igual ao do produto de somas (7). Conclui-se, então, que a função minimal pode ser $s3(b3,b2,b1,b0) = (b3+b2)\cdot(b2\prime+b1+b0)$ *ou* $s3(b3,b2,b1,b0) = b3+b2b0+b2b1$.

3.5.3 Algoritmo de Quine-McCluskey

O mapa de Karnaugh é utilizado de forma bastante eficiente para minimizar funções de até cinco ou seis variáveis. A principal vantagem desta técnica é a sua representação gráfica (2D) que facilita a visualização dos passos de minimização. Quando se trata de funções com uma grande quantidade de variáveis, é impraticável representar de forma apropriada a relação das variáveis com uso de mapas.

Para contornar este problema, outras técnicas foram propostas. Entre elas, o algoritmo de Quine-McCluskey que trabalha diretamente com a representação binária dos mintermos da função, podendo ser programado numa linguagem de programação.

O algoritmo de Quine-McCluskey consiste de duas fases:

1. geração dos implicantes primos a partir de uma lista $\mathcal{L}$ de mintermos da função, e

2. cobertura máxima dos mintermos.

Um mintermo de uma função de n variáveis é representado na lista por n dígitos, onde cada dígito indica se a variável correspondente é complementada ou não. Estas duas possibilidades são representadas por

1. 1, se a variável estiver na forma não-complementada e

2. 0, se a variável estiver na forma complementada.

Exemplo 3.21 *Os mintermos* $vwxyz\prime$ *e* $vwx\prime yz$ *são representados, respectivamente, por 00110 e 10011.*

Com esta representação binária, a primeira fase do algoritmo Quine-McCluskey consiste essencialmente em reduzir as entradas na lista $\mathcal{L}$, procurando agrupar os pares com base na propriedade $xy + xy\prime = x$. Isto é, para cada duas entradas que diferem somente de 1 dígito na posição i, podemos reduzi-las numa entrada com os mesmos dígitos exceto na posição i que será atribuído "–" para indicar que a variável correspondente não aparecerá na expressão. O processo se repetirá para a nova lista na qual duas entradas que diferem de um dígito diferente de "–" são combinadas numa única, gerando uma nova lista. E assim sucessivamente, até esgotar todas as possíveis combinações. As entradas não combinadas formam o conjunto de implicantes primos da função. Esta fase é equivalente ao agrupamento de células por linhas e por colunas no mapa de Karnaugh.

Observe que

1. para reduzir o custo da comparação entre as entradas na lista, é interessante agrupar os mintermos em quantidade de 1's. Assim, pode-se limitar a comparação entre os mintermos/implicantes entre dois blocos adjacentes; e

2. uma entrada com mais de um "–" pode ser obtida de várias maneiras e para evitar processamentos desnecessários que levam a um mesmo resultado, apenas são comparadas as entradas cujos rótulos formam uma sequência crescente e marcamos as outras entradas que contém os mesmos rótulos como "utilizada" para construir um implicante de ordem superior. Este controle pode ser feito através de um flag (1=utilizada e 0=não utilizada).

Exemplo 3.22 *Determine os implicantes primos de*
$f = \sum(0,2,4,6,7,8,10,11,12,13,14,16,18,19,29,30)$

Iteração 1

rótulo	*v*	*w*	*x*	*y*	*z*	*flag*
(0)	*0*	*0*	*0*	*0*	*0*	*1*
(2)	*0*	*0*	*0*	*1*	*0*	*1*
(4)	*0*	*0*	*1*	*0*	*0*	*1*
(8)	*0*	*1*	*0*	*0*	*0*	*1*
(16)	*1*	*0*	*0*	*0*	*0*	*1*
(6)	*0*	*0*	*1*	*1*	*0*	*1*
(10)	*0*	*1*	*0*	*1*	*0*	*1*
(12)	*0*	*1*	*1*	*0*	*0*	*1*
(18)	*1*	*0*	*0*	*1*	*0*	*1*
(7)	*0*	*0*	*1*	*1*	*1*	*1*
(11)	*0*	*1*	*0*	*1*	*1*	*1*
(13)	*0*	*1*	*1*	*0*	*1*	*1*
(14)	*0*	*1*	*1*	*1*	*0*	*1*
(19)	*1*	*0*	*0*	*1*	*1*	*1*
(29)	*1*	*1*	*1*	*0*	*1*	*1*
(11)	*1*	*1*	*1*	*1*	*0*	*1*

Iteração 2

rótulo	*v*	*w*	*x*	*y*	*z*	*flag*
(0,2)	*0*	*0*	*0*	-	*0*	*1*
(0,4)	*0*	*0*	-	*0*	*0*	*1*
(0,8)	*0*	-	*0*	*0*	*0*	*1*
(0,16)	-	*0*	*0*	*0*	*0*	*1*
(2,6)	*0*	*0*	-	*1*	*0*	*1*
(2,10)	*0*	-	*0*	*1*	*0*	*1*
(2,18)	-	*0*	*0*	*1*	*0*	*1*
(4,6)	*0*	*0*	*1*	-	*0*	*1*
(4,12)	*0*	-	*1*	*0*	*0*	*1*
(8,10)	*0*	*1*	*0*	-	*0*	*1*
(8,12)	*0*	*1*	-	*0*	*0*	*1*
(16,18)	*1*	*0*	*0*	-	*0*	*1*
(6,7)	*0*	*0*	*1*	*1*	-	*0*
(6,14)	*0*	-	*1*	*1*	*0*	*1*
(10,11)	*0*	*1*	*0*	*1*	-	*0*
(10,14)	*0*	*1*	*1*	*0*	-	*1*
(12,13)	*0*	*1*	*1*	*0*	-	*0*
(12,14)	*0*	*1*	*1*	-	*0*	*1*
(18,19)	*1*	*0*	*0*	*1*	-	*0*
(13,29)	-	*1*	*1*	*0*	*1*	*0*
(14,30)	-	*1*	*1*	*1*	*0*	*0*

Iteração 3

rótulo	*v*	*w*	*x*	*y*	*z*	*flag*
(0,2,4,6)	*0*	*0*	-	-	*0*	*1*
(0,2,8,10)	*0*	-	*0*	-	*0*	*1*
(0,2,16,18)	-	*0*	*0*	-	*0*	*0*
(0,4,8,12)	*0*	-	-	*0*	*0*	*1*
(2,6,10,14)	*0*	-	-	*1*	*0*	*1*
(4,6,12,14)	*0*	-	*1*	-	*0*	*1*
(8,10,12,14)	*0*	*1*	-	-	*0*	*1*

Iteração 4

rótulo	*v*	*w*	*x*	*y*	*z*	*flag*
(0,2,4,6,8 10,12,14)	*0*	*0*	*0*	-	*0*	*1*

Uma possível expressão para a função f, envolvendo somente os seus implicantes primos é:

$f = (0,2,4,6,8,10,12,14) + (0,2,16,18) + (14,30) + (13,29) + (18,19) + (10,11) + (12,13) + (6,7)$

Tendo os implicantes primos, é construída uma **tabela de implicantes primos** cujas colunas são os mintermos da função e as linhas, os implicantes primos. Nas interseções entre os implicantes primos e os mintermos insere-se uma marca "X".

Exemplo 3.23 *A tabela de implicantes primos do exemplo 3.22 é*

	0	*2*	*4*	*8*	*16*	*6*	*10*	*12*	*18*	*7*	*11*	*13*	*14*	*19*	*29*	*30*
1	*X*	*X*	*X*	*X*		*X*	*X*	*X*					*X*			
2	*X*	*X*			*X*				*X*							
3													*X*			*X*
4												*X*			*X*	
5									*X*					*X*		
7							*X*				*X*					
5									*X*					*X*		
6							*X*				*X*					
7								*X*				*X*				
8						*X*				*X*						

A partir desta tabela procura-se selecionar um subconjunto mínimo de linhas da tabela (implicantes primos) que cubram todos os mintermos da função através dos seguintes passos:

1. identificar as **linhas essenciais** que correspondem aos **implicantes primos essenciais**. Em termos de algoritmo, estas linhas são as quais que contém um mintermo que não pertencem a nenhuma outra linha.

2. exclui-se as colunas dos mintermos que são cobertos pelos implicantes essenciais. Se não sobrar nenhuma coluna, fim.

3. Dentre as linhas não marcadas, escolhe-se aquela que cobre o maior número de mintermos e exclui-se as colunas dos mintermos que são cobertos por ela. Repete-se o procedimento até eliminar todas as colunas.

As linhas marcadas correspondem aos implicantes primos que devem fazer parte da soma de produtos com um número mínimo de termos.

Exemplo 3.24 *Aplicando os passos mencionados na tabela de implicantes primos do exemplo 3.23 as linhas 1,2,3,4,5,7 e 8 são identificadas como linhas essenciais (marcadas com *)*

	0	2	4	8	16	6	10	12	18	7	11	13	14	19	29	30	
1	X	X	X	X		X	X	X					X				*
2	X	X			X				X								*
3													X			X	*
4												X			X		*
5									X					X			*
7							X				X						*
5									X					X			*
6							X				X						
7								X				X					*
8						X				X							*

Sobraram somente as colunas 4, 8, 16, 7, 11, 19, 29 e 30. Como a linha 6 não contém os mintermos correspondentes, a expressão procurada é

$$f = (0,2,4,6,8,10,12,14)+(0,2,16,18)+(14,30)+(13,29)+(18,19)+(10,11)+(6,7).$$

Vale observar aqui que o algoritmo Quin-McCluskey se adapta perfeitamente às funções com estados "don't care", desde que

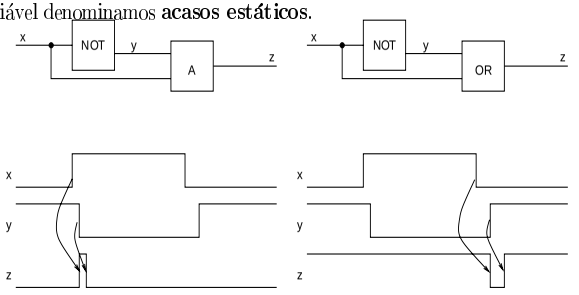
- na primeira fase trata-se os "don't care" da mesma forma que se trata dos mintermos, e
- na segunda fase, ignora-se os "don't care", pois eles não precisam ser cobertos.

3.6 Acasos

Os métodos de análise e síntes que apresentamos até agora ignoram os atrasos nos circuitos, isto é, eles consideram que as variações entre os níveis 0 e 1 nas entradas refletem instantaneamente nas saídas. Nos circuitos reais o **tempo de propagação** das variações nas entradas para as saídas não é zero. Ele pode depende de vários fatores, como tecnologia e temperatura.

Por causa dos atrasos na resposta dos circuitos, o **comportamento transiente** de um circuito pode ser diferente daquele esperado pelos métodos de análise que já vimos. Pulsos espúrios (*glitch*) podem aparecer nos sinais de saída na transição de uma combinação de entradas à outra, que teoricamente não deve levar à variação no valor da saída. Dizemos, então, que existem **acasos** (*hazards*) no circuito, como ilustram os dois circuitos seguintes. Note

que apareceu um pulso (espúrio) na saída dos dois circuitos. A situações em que a saída apresenta um pulso durante a transição do estado de uma variável denominamos **acasos estáticos**.



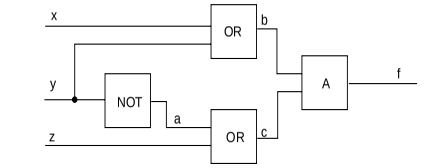
Acaso-0 Acaso-1

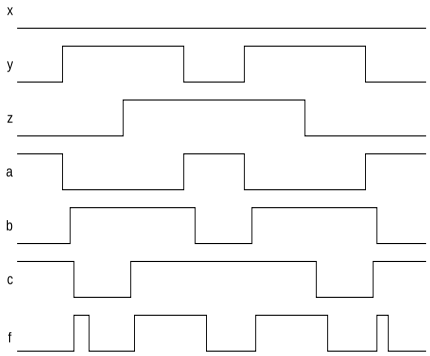
Geralmente, um **acaso estático** é contornável quando apenas uma variável muda de estado a cada instante. Nestas situações, a adição de um termo de **consenso** pode reduzir os acasos, como ilustra o seguinte exemplo, evitando que os mintermos adjacentes que produzem o mesmo valor de saída não sejam cobertos simultaneamente por UM mesmo implicante primo que aparece na expressão final da função.

Exemplo 3.25 *Seja uma função $f = (x+y)\cdot(y'+z)$ que é um produto de somas. Marcando 1 nos maxtermos do mapa de Karnaugh que implicam a função, temos*

$+z/x+y$	0+0	0+1	1+1	1+0
+0	1	1	1	
+1	1			

A "descontinuidade" entre as duas células pode introduzir "pulsos espúrios" nas transições $000 \to 010$ e $010 \to 000$, conforme a seguinte figura, quando a saída "escorrega" momentaneamente para o nível 1.





Se, porém, introduzirmos o termo $x'z'$ na expressão f para "cobrir" esta "lacuna", os pulsos serão eliminados, como mostra no diagrama abaixo.