

CCF 251 – Introdução aos Sistemas Lógicos

Aula 02 – Lógica Combinacional
Prof. José Augusto Nacif – jnacif@ufv.br



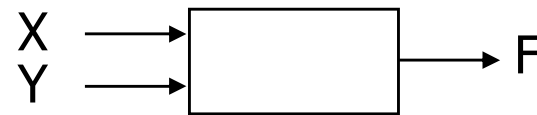
Lógica combinacional

- ▶ **Lógica básica**
 - ▶ Álgebra booleana, prova por reescrita, prova por indução perfeita
 - ▶ Funções lógicas, tabelas verdade, e chaves
 - ▶ NOT, AND, OR, NAND, NOR, XOR, . . . , conjunto mínimo
- ▶ **Realização lógica**
 - ▶ Dois níveis lógicos e formas canônicas
 - ▶ Funções incompletamente especificadas
- ▶ **Simplificação**
 - ▶ Teorema de união
 - ▶ Agrupamento dos termos em funções booleanas
- ▶ **Representações alternativas de funções booleanas**
 - ▶ Cubos
 - ▶ Mapa de Karnaugh



Funções lógicas possíveis de duas variáveis

- ▶ Existem 16 possíveis funções com 2 variáveis de entrada:
 - ▶ Em geral, existem $2^{(2^n)}$ funções de n entradas



X	Y	16 possíveis funções (F0–F15)															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0		X		Y		X xor Y		X or Y		X nor Y not (X or Y)		X = Y		not Y	
		X and Y												not X		X nand Y not (X and Y)	
																1	



Custo de diferentes funções lógicas

- ▶ Diferentes funções são implementadas de maneira mais fácil ou difícil
 - ▶ Cada uma tem um custo associado com o número de chaves necessárias
 - ▶ 0 (F0) and 1 (F15): requerem 0 chaves, estão diretamente conectados as saídas baixo/alto
 - ▶ X (F3) e Y (F5): requerem 0 chaves, saída é uma das entradas
 - ▶ X' (F12) e Y' (F10): requerem 2 chaves para o “inversor” ou a porta NOT
 - ▶ X nor Y (F4) e X nand Y (F14): requerem 4 chaves
 - ▶ X or Y (F7) e X and Y (F1): requerem 6 chaves
 - ▶ $X = Y$ (F9) e $X \oplus Y$ (F6): requerem 16 chaves
- ▶ Sendo assim, por que NOT, NOR, e NAND são mais baratas? Elas são funções que mais são implementamos na prática.



Conjunto mínimo de funções

- ▶ Nós podemos implementar todas as funções lógicas a partir das portas NOT, NOR, e NAND?
 - ▶ Por exemplo, a implementação de X and Y é igual a not (X nand Y)
- ▶ De fato, nós podemos apenas utilizar somente portas NOR ou NAND
 - ▶ NOT é somente uma NAND ou NOR com ambas entradas ligadas juntas

X	Y	X nor Y
0	0	1
1	1	0

X	Y	X nand Y
0	0	1
1	1	0

- ▶ NAND e NOR são "duas". Isto é, fácil de implementar uma utilizando a outra.

$$X \text{ nand } Y \equiv \text{not } ((\text{not } X) \text{ nor } (\text{not } Y))$$

$$X \text{ nor } Y \equiv \text{not } ((\text{not } X) \text{ nand } (\text{not } Y))$$

- ▶ Não vamos avançar muito rápido ...
 - ▶ Vamos olhar o fundamento matemático da lógica



Uma estrutura algébrica

- ▶ Uma estrutura algébrica consiste de
 - ▶ Um conjunto de elementos B
 - ▶ Operações binárias $\{ + , \cdot \}$
 - ▶ E uma operação de união $\{ ' \}$
 - ▶ Tal que, se mantenha os seguintes postulados:

1. o conjunto B contém no mínimo dois elementos: a, b

2. fechamento: $a + b$ é em B

$a \cdot b$ é em B

3. comutativa: $a + b = b + a$

$a \cdot b = b \cdot a$

4. associativa: $a + (b + c) = (a + b) + c$

$a \cdot (b \cdot c) = (a \cdot b) \cdot c$

5. identidade: $a + 0 = a$

$a \cdot 1 = a$

6. distributiva: $a + (b \cdot c) = (a + b) \cdot (a + c)$

$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$

7. complementar: $a + a' = 1$

$a \cdot a' = 0$



Álgebra booleana

- ▶ Álgebra booleana
 - ▶ $B = \{0, 1\}$
 - ▶ Variáveis
 - ▶ $+$ é OR lógico, \cdot é AND lógico
 - ▶ $'$ é NOT lógico
- ▶ Todas as afirmações dos postulados algébricos



Funções lógicas e álgebra booleana

- Qualquer função lógica que possa ser expressa como uma tabela verdade pode ser escrita como uma expressão de álgebra booleana utilizando os operadores: ', +, and •

X	Y	$X \bullet Y$
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X'	$X' \bullet Y$
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

X	Y	X'	Y'	$X \bullet Y$	$X' \bullet Y'$	$(X \bullet Y) + (X' \bullet Y')$
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

$$(X \bullet Y) + (X' \bullet Y') \equiv X = Y$$

X, Y são variáveis da álgebra booleana

Expressão Booleana que é Verdade, quando as variáveis X e Y tem o mesmo valor e falso, caso contrário.



Postulados e teoremas da álgebra booleana

- ▶ Identidade

1. $X + 0 = X$

1D. $X \cdot 1 = X$

- ▶ Nulo

2. $X + 1 = 1$

2D. $X \cdot 0 = 0$

- ▶ Idempotência

3. $X + X = X$

3D. $X \cdot X = X$

- ▶ Involução

4. $(X')' = X$

- ▶ Complementar

5. $X + X' = 1$

5D. $X \cdot X' = 0$

- ▶ Comutativa

6. $X + Y = Y + X$

6D. $X \cdot Y = Y \cdot X$

- ▶ Associativa

7. $(X + Y) + Z = X + (Y + Z)$

7D. $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$



Postulados e teoremas da álgebra booleana

► Distributiva:

$$8. X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z) \quad 8D. X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$$

► União:

$$9. X \cdot Y + X \cdot Y' = X \quad 9D. (X + Y) \cdot (X + Y') = X$$

► Absorção:

$$10. X + X \cdot Y = X$$

$$10D. X \cdot (X + Y) = X$$

$$11. (X + Y') \cdot Y = X \cdot Y$$

$$11D. (X \cdot Y') + Y = X + Y$$

► Fatoração:

$$12. (X + Y) \cdot (X' + Z) = \\ X \cdot Z + X' \cdot Y$$

$$12D. X \cdot Y + X' \cdot Z = \\ (X + Z) \cdot (X' + Y)$$

► Consenso:

$$13. (X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = \\ X \cdot Y + X' \cdot Z$$

$$13D. (X + Y) \cdot (Y + Z) \cdot (X' + Z) = \\ (X + Y) \cdot (X' + Z)$$



Postulados e teoremas da álgebra booleana

- ▶ deMorgan:

$$I4. (X + Y + \dots)' = X' \cdot Y' \cdot \dots \quad I4D. (X \cdot Y \cdot \dots)' = X' + Y' + \dots$$

- ▶ deMorgan genérico:

$$I5. f'(X_1, X_2, \dots, X_n, 0, 1, +, \cdot) = f(X_1', X_2', \dots, X_n', 1, 0, \cdot, +)$$

- ▶ Estabelecer a relação entre \cdot e $+$



Postulados e teoremas da álgebra booleana

► Dualidade

- Um dual de uma expressão booleana é derivado pela substituição de \bullet por $+$, $+$ por \bullet , 0 por 1 , e 1 por 0 . Deixando as variáveis alteradas.
- Qualquer teorema que possa ser provado é, portanto, também comprovado por dual!
- Um meta teorema (um teorema sobre teoremas)

► dualidade:

$$16. X + Y + \dots \Leftrightarrow X \bullet Y \bullet \dots$$

► dualidade genérica:

$$17. f(X_1, X_2, \dots, X_n, 0, 1, +, \bullet) \Leftrightarrow f(X_1, X_2, \dots, X_n, 1, 0, \bullet, +)$$

► Diferença da lei deMorgan

- É uma afirmação sobre teoremas
- Não é uma maneira de manipular (reescrever) expressões



Comprovando teoremas (reescrita)

► Usando os postulados da álgebra booleana:

► Ex., comprovar o teorema: $X \cdot Y + X \cdot Y' = X$

distributiva (8)	$X \cdot Y + X \cdot Y'$	$=$	$X \cdot (Y + Y')$
complementar (5)	$X \cdot (Y + Y')$	$=$	$X \cdot (1)$
identidade (1D)	$X \cdot (1)$	$=$	$X \checkmark$

► Ex., comprovar o teorema: $X + X \cdot Y = X$

identidade (1D)	$X + X \cdot Y$	$=$	$X \cdot 1 + X \cdot Y$
distributiva (8)	$X \cdot 1 + X \cdot Y$	$=$	$X \cdot (1 + Y)$
identidade (2)	$X \cdot (1 + Y)$	$=$	$X \cdot (1)$
identidade (1D)	$X \cdot (1)$	$=$	$X \checkmark$



Atividade

- Comprovar a expressão abaixo usando as leis da álgebra booleana:

- $(X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = X \cdot Y + X' \cdot Z$

identidade

complementar

distributiva

comutativa

fatoração

nulo

identidade

$$(X \cdot Y) + (Y \cdot Z) + (X' \cdot Z)$$

$$(X \cdot Y) + (1) \cdot (Y \cdot Z) + (X' \cdot Z)$$

$$(X \cdot Y) + (X' + X) \cdot (Y \cdot Z) + (X' \cdot Z)$$

$$(X \cdot Y) + (X' \cdot Y \cdot Z) + (X \cdot Y \cdot Z) + (X' \cdot Z)$$

$$(X \cdot Y) + (X \cdot Y \cdot Z) + (X' \cdot Y \cdot Z) + (X' \cdot Z)$$

$$(X \cdot Y) \cdot (1 + Z) + (X' \cdot Z) \cdot (1 + Y)$$

$$(X \cdot Y) \cdot (1) + (X' \cdot Z) \cdot (1)$$

$$(X \cdot Y) + (X' \cdot Z)$$



Comprovando teoremas (indução perfeita)

- ▶ Utilizando indução perfeita (tabela verdade completa):
 - ▶ Ex., de Morgan's:

$(X + Y)' = X' \cdot Y'$
NOR é equivalente para AND
com entradas complementadas

X	Y	X'	Y'	$(X + Y)'$	$X' \cdot Y'$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

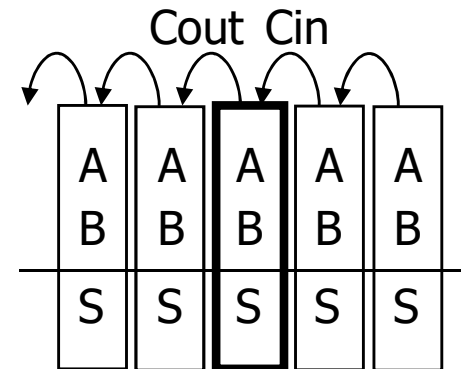
$(X \cdot Y)' = X' + Y'$
NAND é equivalente para OR
com entradas complementadas

X	Y	X'	Y'	$(X \cdot Y)'$	$X' + Y'$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0



Um exemplo simples: somador binário de 1-bit

- ▶ Entradas: A, B, vai 1 (Carry-in)
- ▶ Saídas: Soma, vêm 1 (Carry-out)



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$S = A' B' \text{Cin} + A' B \text{Cin}' + A B' \text{Cin}' + A B \text{Cin}$$

$$\text{Cout} = A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin}$$



Aplicar os teoremas para simplificar expressões

- ▶ Os teoremas de álgebra booleana podem simplificar expressões booleana
- ▶ Ex., função vêm I (carry-out) somador completo (algumas regras são aplicadas para qualquer função)

$$\begin{aligned}\text{Cout} &= A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= A' B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + \boxed{A B \text{Cin} + A B \text{Cin}} \\ &= A' B \text{Cin} + A B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= (A' + A) B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= (1) B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= B \text{Cin} + A B' \text{Cin} + A B \text{Cin}' + \boxed{A B \text{Cin} + A B \text{Cin}} \\ &= B \text{Cin} + A B' \text{Cin} + A B \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= B \text{Cin} + A (B' + B) \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= B \text{Cin} + A (1) \text{Cin} + A B \text{Cin}' + A B \text{Cin} \\ &= B \text{Cin} + A \text{Cin} + A B (\text{Cin}' + \text{Cin}) \\ &= B \text{Cin} + A \text{Cin} + A B (1) \\ &= B \text{Cin} + A \text{Cin} + A B\end{aligned}$$

Adicionar termos
extras criando novas
oportunidades de
fatoração



Atividade

- Preencher a tabela verdade para um circuito que verifique se um número de 4 bits é divisível por 2, 3, or 5

X8	X4	X2	X1	By2	By3	By5
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0

- Escrever abaixo as expressões boolean para By2, By3, e By5



Atividade

X8	X4	X2	X1	By2	By3	By5
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	1	1	0
0	1	1	1	0	0	0
1	0	0	0	1	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	1	1	0
1	1	0	1	0	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	1

$$\begin{aligned} \text{By2} &= X8'X4'X2'X1' + X8'X4'X2X1' \\ &\quad + X8'X4X2'X1' + X8'X4X2X1' \\ &\quad + X8X4'X2'X1' + X8X4'X2X1' \\ &\quad + X8X4X2'X1' + X8X4X2X1' \\ &= X1' \end{aligned}$$

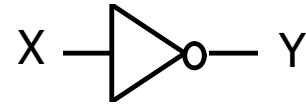
$$\begin{aligned} \text{By3} &= X8'X4'X2'X1' + X8'X4'X2X1 \\ &\quad + X8'X4X2X1' + X8X4'X2'X1 \\ &\quad + X8X4X2'X1' + X8X4X2X1 \end{aligned}$$

$$\begin{aligned} \text{By5} &= X8'X4'X2'X1' + X8'X4X2'X1 \\ &\quad + X8X4'X2X1' + X8X4X2X1 \end{aligned}$$



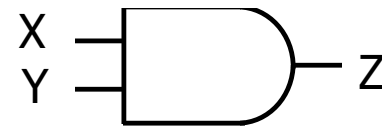
Expressões booleana para portas lógicas

► NOT X' \bar{X} $\sim X$



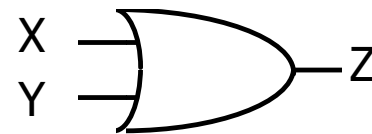
X	Y
0	1
1	0

► AND $X \cdot Y$ XY $X \wedge Y$



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

► OR $X + Y$ $X \vee Y$

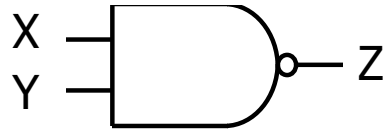


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1



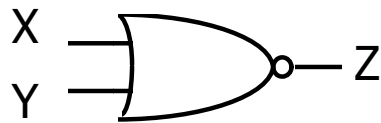
Expressões booleana para portas lógicas

► NAND



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

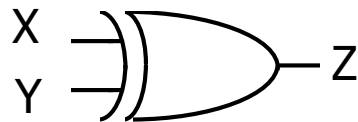
► NOR



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

► XOR

$$X \oplus Y$$

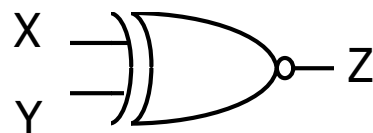


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

$X \text{ xor } Y = X Y' + X' Y$
X ou Y mas não ambas
("desigualdade", "diferença")

► XNOR

$$X = Y$$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

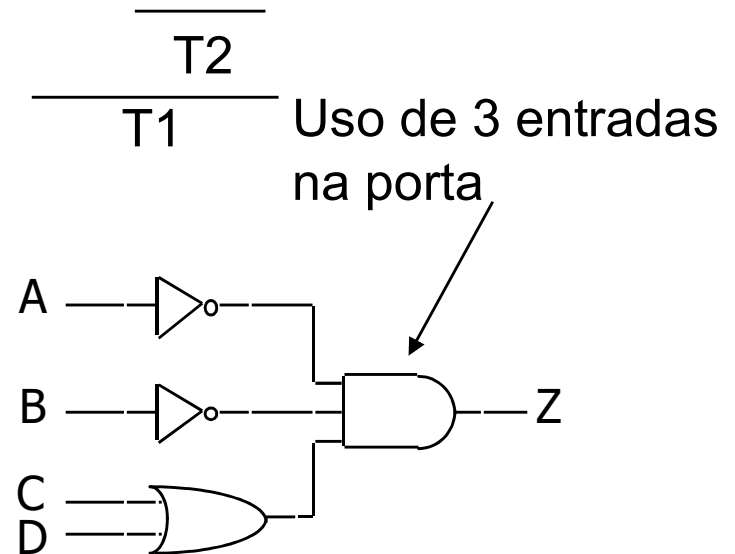
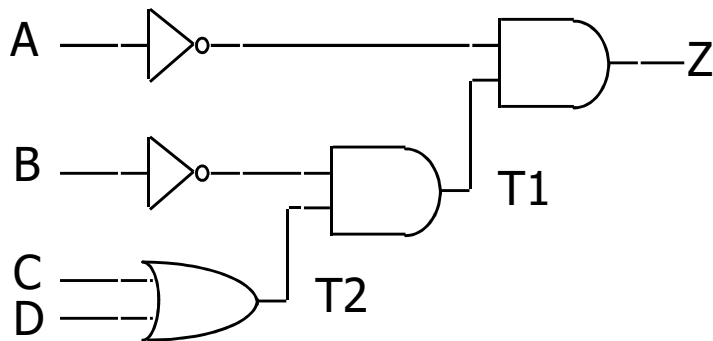
$X \text{ xnor } Y = X Y + X' Y'$
X e Y são as mesmas
("igualdade", "coincidência")



Expressões booleana para portas lógicas

- Mais de uma maneira de mapear expressões para portas

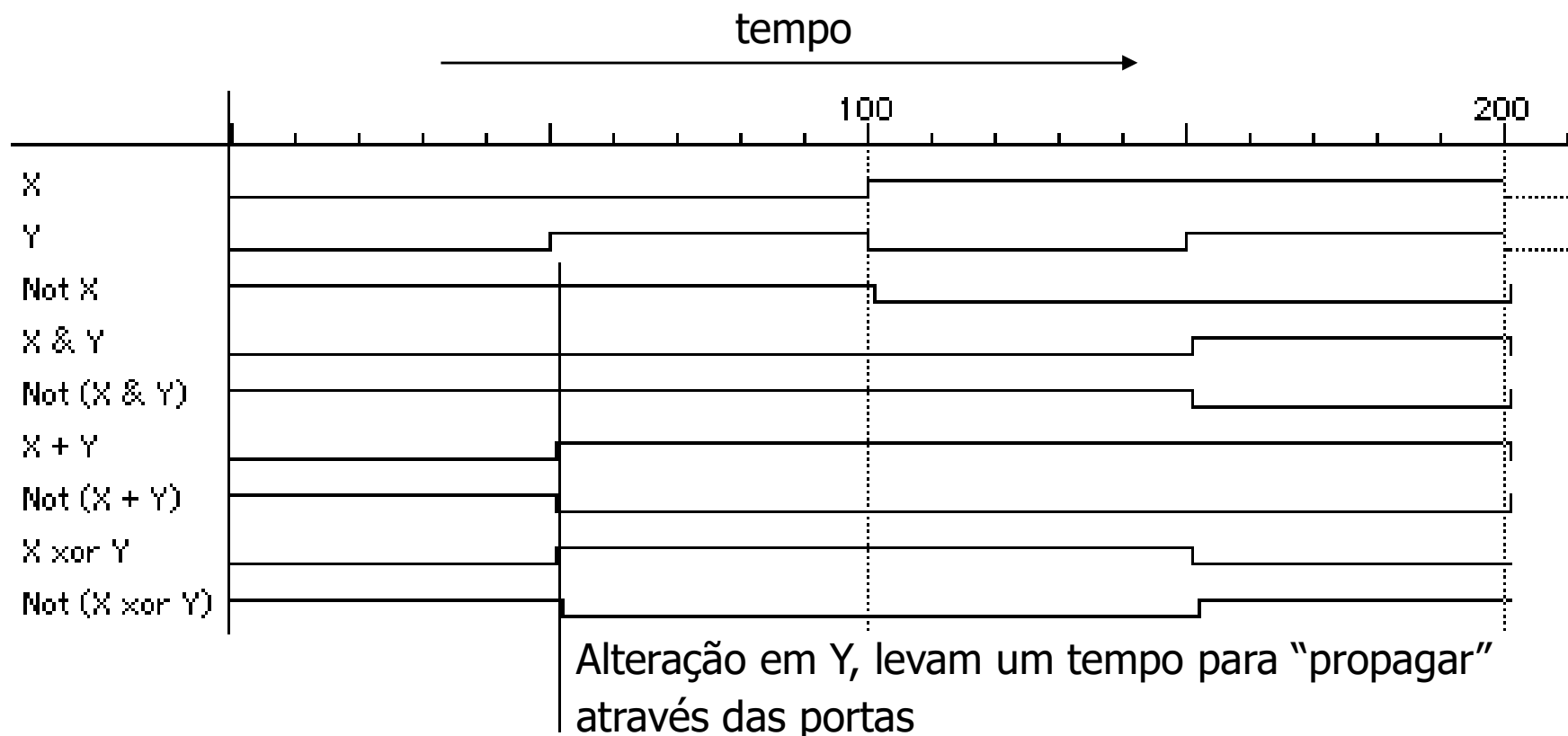
► Ex., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot (C + D)))$





Visão forma de ondas das funções lógicas

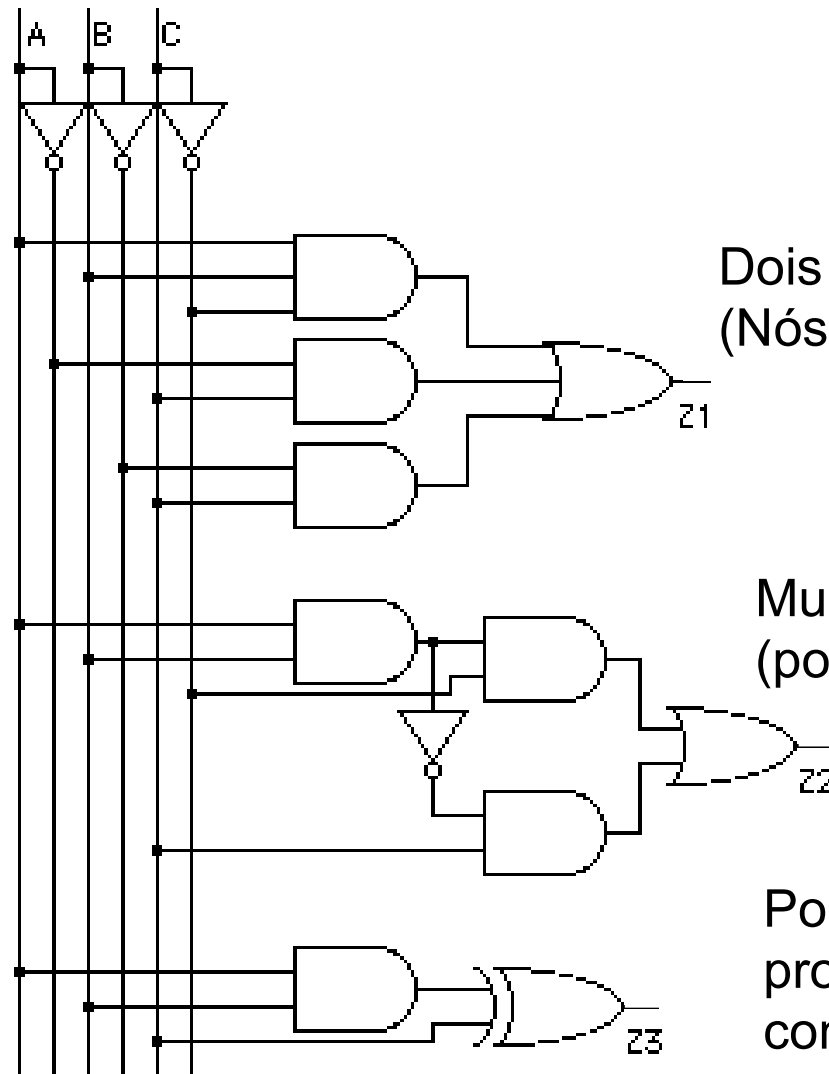
- ▶ Apenas uma tabela verdade para os lados
 - ▶ Mas observe como as bordas não se alinham exatamente
 - ▶ É necessário um tempo para uma porta alterar sua saída!





Escolhendo diferentes formas de implementar uma função

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Dois níveis
(Nós não contamos portas NOT)

Multi níveis
(portas com menos entradas)

Porta XOR (mais fácil para projetar, mas caro para construir)



Qual forma de implementação é melhor?

- ▶ **Reduzir o número de entradas**
 - ▶ literal: variável de entrada (complementada ou não)
 - ▶ Podemos aproximar o custo das portas lógicas com 2 transistores por literal
 - ▶ Por que não contar inversores?
 - ▶ Menos literais significa menos transistores
 - ▶ Circuitos menores
 - ▶ Menos entradas implica em portas rápidas
 - ▶ Portas são menores e portanto mais rápidas
 - ▶ Número de entradas de portas (*fanin*) são limitados em algumas tecnologias
- ▶ **Reduzir o número de portas**
 - ▶ Menos portas significa um circuito menor
 - ▶ Influencia diretamente no custo da fabricação, pois afeta o encapsulamento



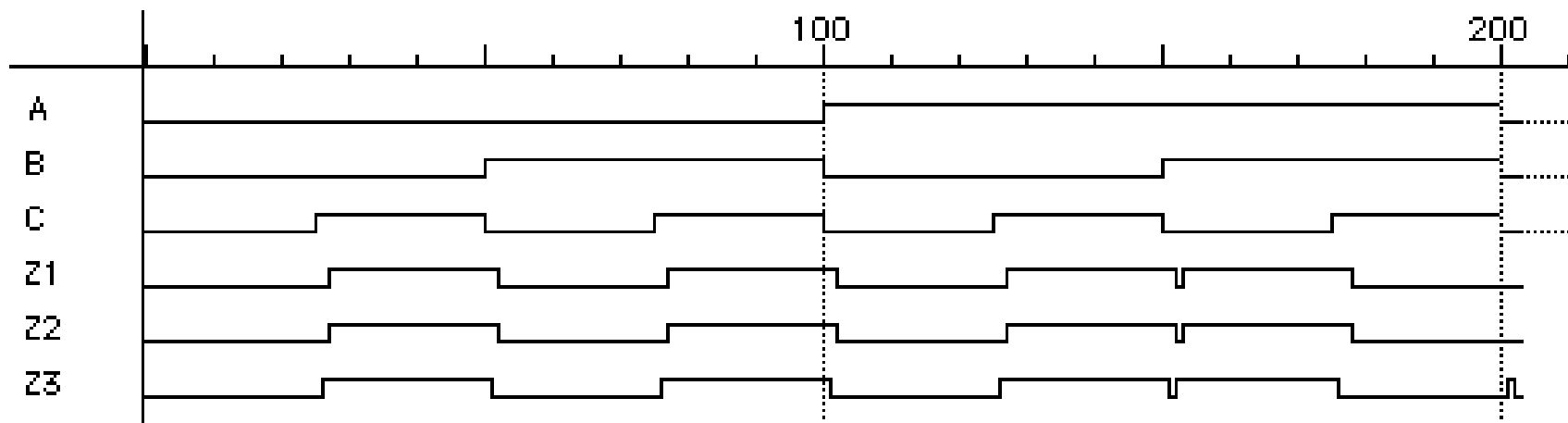
Qual forma de implementação é melhor?

- ▶ **Reduzir o número dos níveis das portas**
 - ▶ Menos níveis das portas implica em reduzir o atraso do sinal de propagação
 - ▶ Configuração do atraso mínimo tipicamente requer mais portas
 - ▶ Mais ampla, circuitos menos profundos
- ▶ **Como nós exploramos compensações entre o aumento no atraso do circuito e o tamanho?**
 - ▶ Ferramentas automatizadas para geração de soluções diferentes
 - ▶ Minimização lógica: reduzir o número de portas e complexidade
 - ▶ Otimização lógica: redução enquanto negociações contra atraso



Todas as formas de implementação são equivalentes?

- ▶ De acordo com os mesmos estímulos de entrada, as três implementações alternativas têm quase o mesmo comportamento de forma de onda
 - ▶ Atrasos são diferentes
 - ▶ Falhas podem surgir – isto pode ser ruim, depende
 - ▶ Variações devido as diferenças no número dos níveis das portas e estrutura
- ▶ As três implementações são funcionalmente equivalentes





Implementação de funções booleanas

- ▶ **Tecnologia independente**
 - ▶ Formas canônicas
 - ▶ Formas de dois níveis
 - ▶ Formas de multi níveis
- ▶ **Escolhas tecnológicas**
 - ▶ Encapsulamento para poucas portas
 - ▶ Lógica regular
 - ▶ Dois níveis de lógica programável
 - ▶ Multi níveis de lógica programável



Formas canônicas

- ▶ Tabela verdade é a assinatura única de uma função booleana
- ▶ A mesma tabela verdade pode ter muitas formas de implementação com portas
- ▶ Formas canônicas
 - ▶ Formas padrão para expressão booleana
 - ▶ Fornece uma assinatura algébrica única



Forma canônica Soma de produtos

- ▶ Também conhecida como forma normal disjunta
- ▶ Também conhecida como expansão mintermo

			$F = 001 \quad 011 \quad 101 \quad 110 \quad 111$				
			$F = A'B'C + A'BC + AB'C + ABC' + ABC$				
A	B	C	F	F'			
0	0	0	0	1			
0	0	1	1	0			
0	1	0	0	1			
0	1	1	1	0			
1	0	0	0	1			
1	0	1	1	0			
1	1	0	1	0			
1	1	1	1	0			

$F' = A'B'C' + A'BC' + AB'C'$




Forma canônica Soma de produtos

- ▶ Termo produto (ou mintermo)
 - ▶ Soma de produtos de literais – combinação da entrada para qual saída é verdadeiro
 - ▶ Cada variável aparece exatamente uma vez, verdadeiro ou invertido (mas não ambos)

A	B	C	mintermos	
0	0	0	$A'B'C'$	m0
0	0	1	$A'B'C$	m1
0	1	0	$A'BC'$	m2
0	1	1	$A'BC$	m3
1	0	0	$AB'C'$	m4
1	0	1	$AB'C$	m5
1	1	0	ABC'	m6
1	1	1	ABC	m7

Notação abreviada para
mintermos de 3 variáveis



F na forma canônica:

$$\begin{aligned}F(A, B, C) &= \Sigma m(1,3,5,6,7) \\&= m1 + m3 + m5 + m6 + m7 \\&= A'B'C + A'BC + AB'C + ABC' + ABC\end{aligned}$$

Forma canônica \neq forma mínima

$$\begin{aligned}F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\&= (A'B' + A'B + AB' + AB)C + ABC' \\&= ((A' + A)(B' + B))C + ABC' \\&= C + ABC' \\&= ABC' + C \\&= AB + C\end{aligned}$$



Forma canônica produto de somas

- ▶ Também conhecida como forma normal conjuntiva
- ▶ Também conhecida como expansão maxtermo

			F =		000		010		100	
			F =		(A + B + C)		(A + B' + C)		(A' + B + C)	
A	B	C	F	F'						
0	0	0	0	1						
0	0	1	1	0						
0	1	0	0	1						
0	1	1	1	0						
1	0	0	0	1						
1	0	1	1	0						
1	1	0	1	0						
1	1	1	1	0						

$$F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')$$



Forma canônica produto de somas

- ▶ Termo soma (ou maxtermos)
 - ▶ Produto de soma de literais – combinação da entrada para qual saída é falsa
 - ▶ Cada variável aparece exatamente uma vez, verdadeiro ou invertido (mas não ambas)

A	B	C	maxtermos	
0	0	0	$A+B+C$	M0
0	0	1	$A+B+C'$	M1
0	1	0	$A+B'+C$	M2
0	1	1	$A+B'+C'$	M3
1	0	0	$A'+B+C$	M4
1	0	1	$A'+B+C'$	M5
1	1	0	$A'+B'+C$	M6
1	1	1	$A'+B'+C'$	M7

F na forma canônica:

$$\begin{aligned}F(A, B, C) &= \Pi M(0,2,4) \\&= M0 \cdot M2 \cdot M4 \\&= (A + B + C) (A + B' + C) (A' + B + C)\end{aligned}$$

Forma canônica \neq forma mínima

$$\begin{aligned}F(A, B, C) &= (A + B + C) (A + B' + C) (A' + B + C) \\&= (A + B + C) (A + B' + C) \\&\quad (A + B + C) (A' + B + C) \\&= (A + C) (B + C)\end{aligned}$$

Notação abreviada para
maxtermos de 3 variáveis

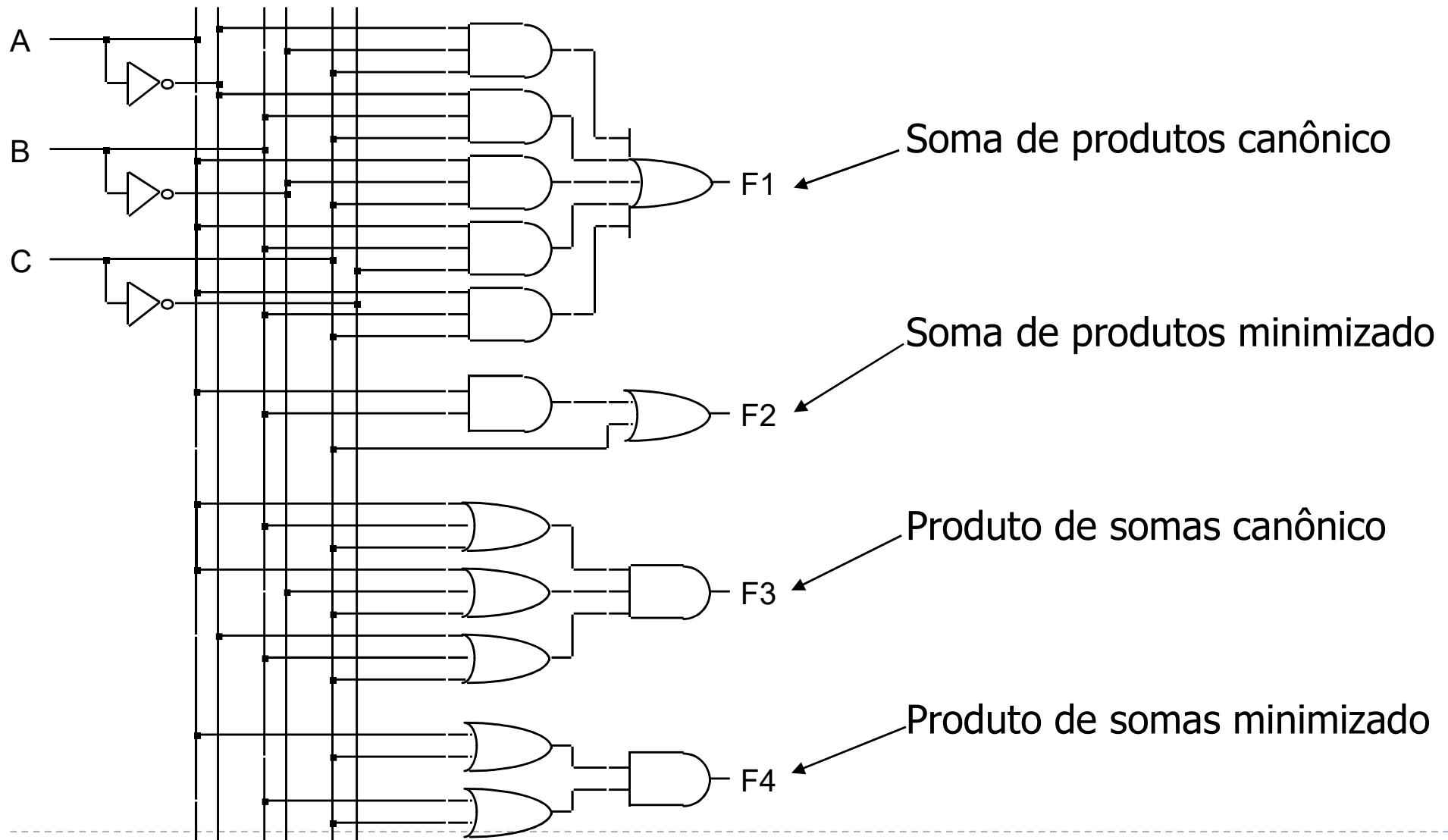


S-o-P, P-o-S, e teorema de deMorgan's

- ▶ Soma de produtos (Sum-of-products)
 - ▶ $F' = A'B'C' + A'BC' + AB'C'$
- ▶ Aplicar deMorgan's
 - ▶ $(F')' = (A'B'C' + A'BC' + AB'C')'$
 - ▶ $F = (A + B + C) (A + B' + C) (A' + B + C)$
- ▶ Produto de somas (Product-of-sums)
 - ▶ $F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')$
- ▶ Aplicar deMorgan's
 - ▶ $(F')' = ((A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C'))'$
 - ▶ $F = A'B'C + A'BC + AB'C + ABC' + ABC$



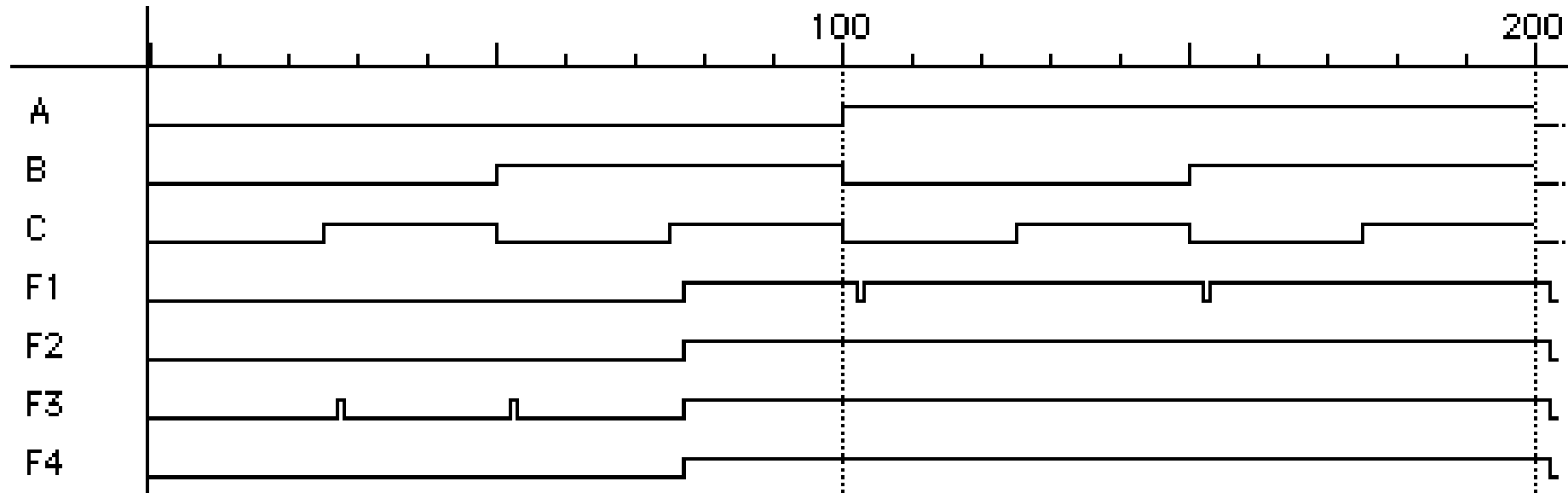
Quatro alternativas de implementações dois níveis de $F = AB + C$





Formas de onda para as quarto alternativas

- ▶ Formas de onda são essencialmente idênticas
 - ▶ Exceto para temporização de dependências (falhas)
 - ▶ Atrasos quase idênticos (modelado como um atraso por nível, não o tipo de porta ou o número das entradas para porta)





Mapeamento entre formas canônicas

- ▶ Conversão mintermo para maxtermo
 - ▶ Utiliza maxtermos, cujo os índices não aparecem na expansão mintermo
 - ▶ Ex., $F(A,B,C) = \sum m(1,3,5,6,7) = \prod M(0,2,4)$
- ▶ Conversão maxtermo para mintermo
 - ▶ Utiliza mintermos, cujo índices não aparecem na expansão maxtermo
 - ▶ Ex., $F(A,B,C) = \prod M(0,2,4) = \sum m(1,3,5,6,7)$
- ▶ Expansão mintermo de F para expansão mintermo de F'
 - ▶ Utiliza mintermos, cujo índices não aparecem
 - ▶ Ex., $F(A,B,C) = \sum m(1,3,5,6,7) \quad F'(A,B,C) = \sum m(0,2,4)$
- ▶ Expansão maxtermo de F para expansão maxtermo de F'
 - ▶ Utiliza maxtermos, cujo índices não aparecem
 - ▶ Ex., $F(A,B,C) = \prod M(0,2,4) \quad F'(A,B,C) = \prod M(1,3,5,6,7)$



Funções incompletamente especificadas

- ▶ Exemplo: código binário decimal incrementado por 1
 - ▶ Dígitos BCD codificados em dígitos decimais de 0 – 9 nos padrões de bits 0000 – 1001

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

off-set de W

on-set de W

don't care (DC) set de W

Estes padrões de entradas nunca devem se encontrados na prática – **"don't care"** sobre os valores das saídas associados, podem ser explorados na minimização



Notação para Funções incompletamente especificadas

- ▶ Don't cares e formas canônicas
 - ▶ Até agora, somente representamos on-set e off-set
 - ▶ Também temos que representar os don't-care-set
- ▶ Representações canônicas da função BCD incrementado por 1:
 - ▶ $Z = m_0 + m_2 + m_4 + m_6 + m_8 + d_{10} + d_{11} + d_{12} + d_{13} + d_{14} + d_{15}$
 - ▶ $Z = \Sigma [m(0,2,4,6,8) + d(10,11,12,13,14,15)]$
 - ▶ $Z = M_1 \cdot M_3 \cdot M_5 \cdot M_7 \cdot M_9 \cdot D_{10} \cdot D_{11} \cdot D_{12} \cdot D_{13} \cdot D_{14} \cdot D_{15}$
 - ▶ $Z = \Pi [M(1,3,5,7,9) \cdot D(10,11,12,13,14,15)]$



Simplificação de dois níveis lógica combinacional

- ▶ Encontrar uma realização mínima de soma de produtos ou produto de somas
 - ▶ Explorar informações don't care no processo
- ▶ Simplificação algébrica
 - ▶ Nenhum procedimento algorítmico/sistemático
 - ▶ Como você sabe quando a realização mínima foi encontrada?
- ▶ Ferramentas projeto Computer-aided
 - ▶ Soluções precisas requerem longo tempo computacional, especialmente para funções com muitas entradas (> 10)
 - ▶ Métodos heurísticos empregados – “palpites” para reduzir a quantidade de processamento e não são as melhores soluções
- ▶ Métodos manuais ainda relevantes
 - ▶ Para entender ferramentas automatizadas e seus pontos positivos e negativos
 - ▶ Capacidade para verificar resultados (em exemplos pequenos)



O teorema de união

- ▶ Ferramenta chave para a simplificação: $A(B' + B) = A$
- ▶ Essência da simplificação de dois níveis lógicos
 - ▶ Encontrar dois subconjuntos de elementos em ON-set, tal que, o valor de uma variável é alterado. Esta simples variação da variável pode ser eliminado e um simples termo de produto utilizado para representar ambos elementos

$$F = A'B' + AB' = (A' + A)B' = B'$$

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

B tem o mesmo valor em ambas linhas on-set
– B permanece

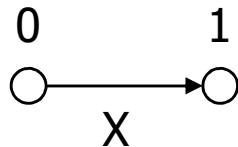
A tem valor diferente nas duas linhas
– A é eliminado



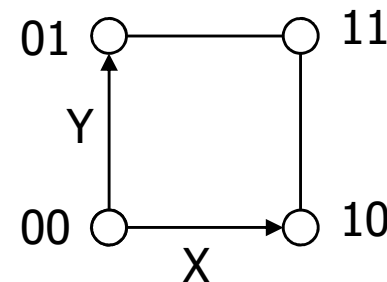
Cubos booleanos

- ▶ Técnica visual para identificar quando o teorema de união pode ser aplicado
- ▶ n variáveis de entrada = n -“cubos” dimensionais

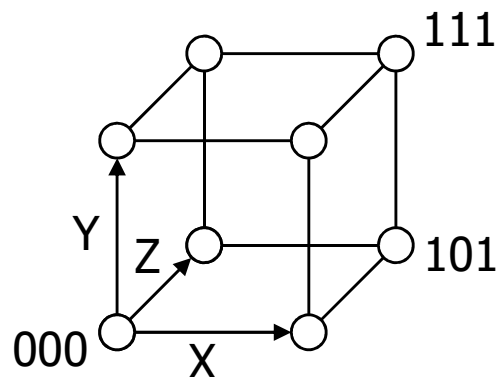
1-cubo



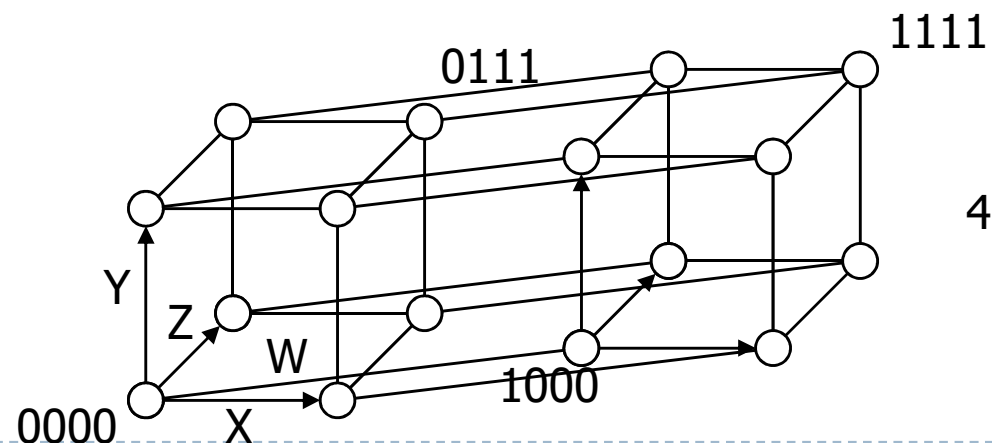
2-cubos



3-cubos



4-cubos

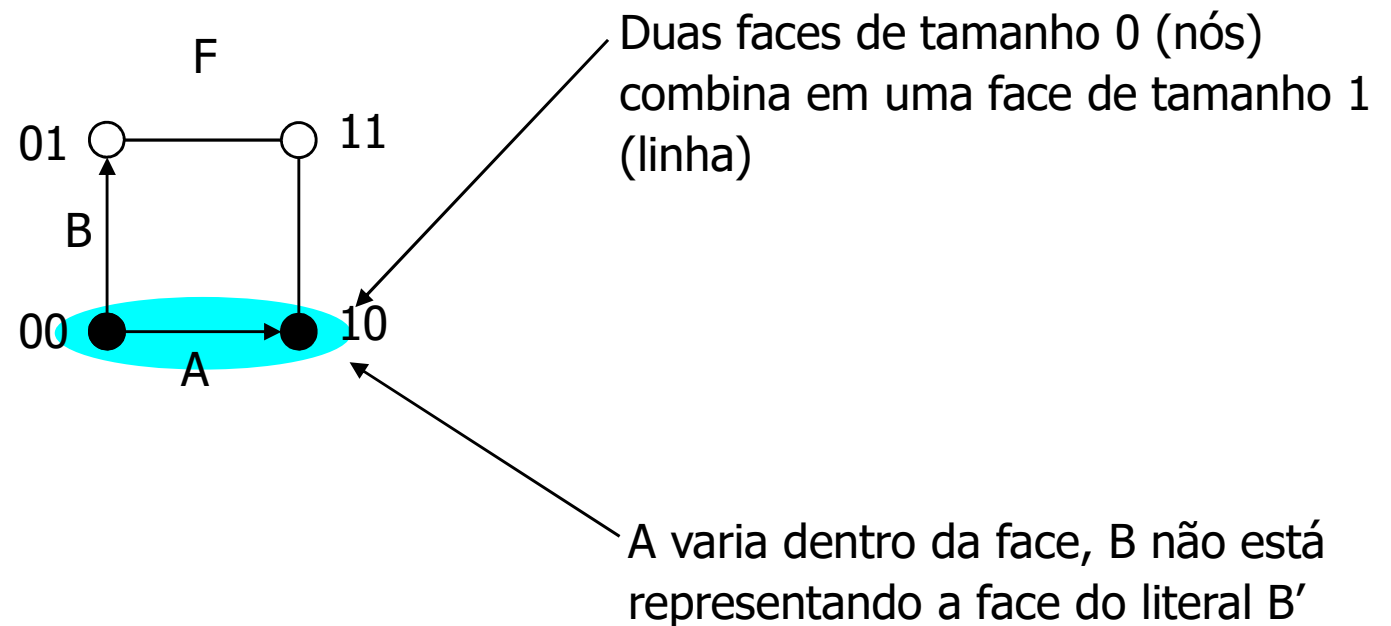




Mapeando tabelas verdade em cubos booleana

- ▶ Teorema de união combina em duas “faces” de um cubo em uma grande “face”
- ▶ Exemplo:

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



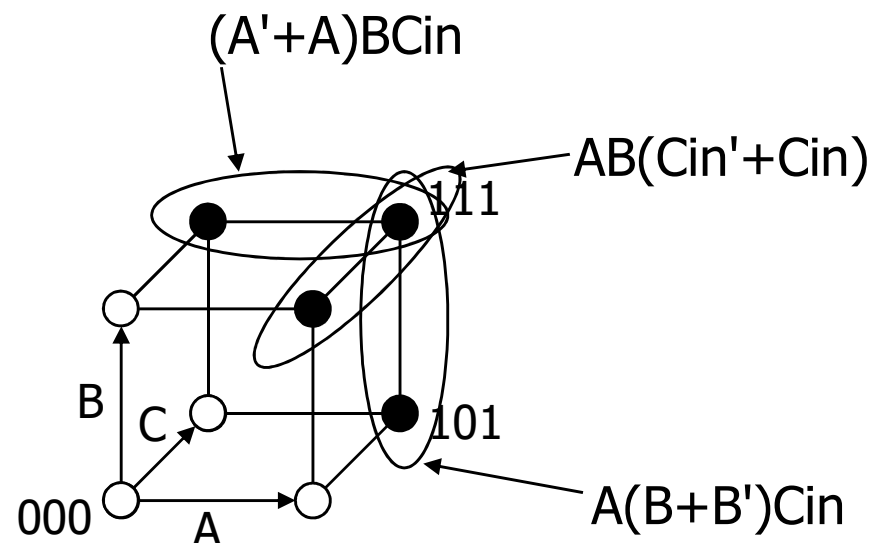
ON-set = nós sólidos
OFF-set = nós vazios
DC-set = nós x'd



Exemplo três variáveis

- Somador completo binário lógica vêm 1 (carry-out)

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



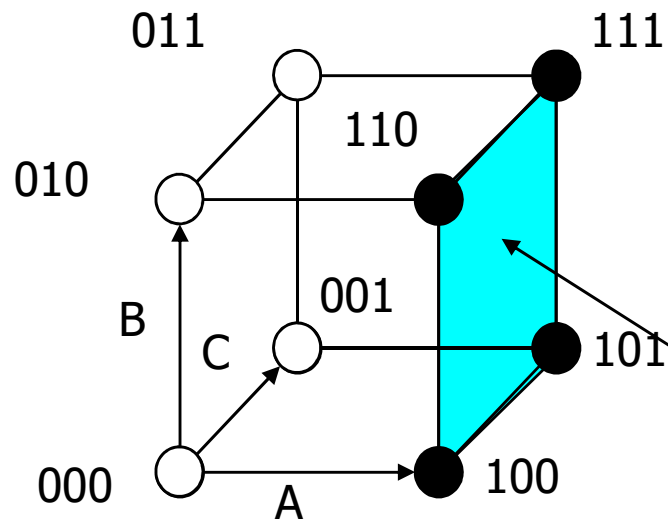
On-set é completamente coberta pela combinação (OR) dos sub cubos de baixa dimensionalidade – observe que “111” é coberto por três vezes

$$Cout = BCin + AB + ACin$$



Cubos dimensionais superiores

► Sub cubos de dimensão superior a 2



$$F(A,B,C) = \Sigma m(4,5,6,7)$$

Formas on-set de um quadrado
Ex., um cubo de dimensão 2

*Representa uma expressão em uma variável
Ex., 3 dimensões – 2 dimensões*

A é uma afirmação (verdade), B inalterado
e C varia

Este sub cubo representa o
literal A



m-cubos dimensionais em um espaço booleano n-dimensional

- ▶ Em 3-cubos (três variáveis):
 - ▶ 0-cubo, Ex., um nó simples, produz um termo em 3 literais
 - ▶ 1-cubo, Ex., uma linha de dois nós, produz um termo em 2 literais
 - ▶ 2-cubos, Ex., um plano de 4 nós, produz um termo em 1 literal
 - ▶ 3-cubos, Ex., um cubo de 8 nós, produz um termo constante “1”
- ▶ Em geral,
 - ▶ Um m-sub cubo dentro de um n-cubo ($m < n$) produz um termo com $n - m$ literais



Mapa de Karnaugh

- ▶ Mapa de plano de cubos booleanos
 - ▶ Agrupamento – próximo das bordas
 - ▶ Difícil para desenhar e visualizar para mais de 4 dimensões
 - ▶ Virtualmente impossível para mais de 6 dimensões
- ▶ Alternativa para tabelas verdade, para ajudar visualizar adjacências
 - ▶ Guia para a aplicação do teorema de união
 - ▶ Elementos on-set com apenas uma variável de valor alterado são adjacentes, ao contrário da situação de uma tabela verdade linear

B \ A	0	1
0	0 1	2 1
1	1 0	3 0

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



Mapa de Karnaugh

- ▶ Esquema de numeração baseado na codificação Gray
 - ▶ Ex., 00, 01, 11, 10
 - ▶ Apenas um simples bit alterado no código para células do mapa adjacente

C \ AB	A			
	00	01	11	10
0	0	2	6	4
1	1	3	7	5

C	A			
	0	2	6	4
1	1	3	7	5

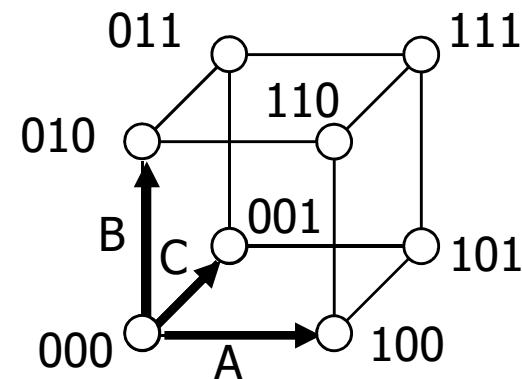
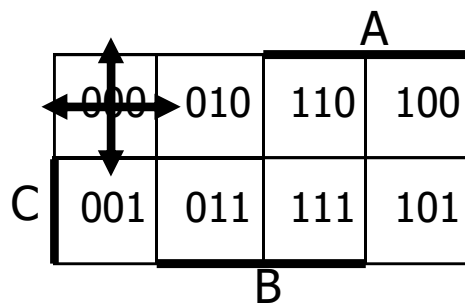
C	A			
	0	4	12	8
	1	5	13	9
	3	7	15	11
2	6	14	10	

$$13 = 1101 = ABC'D$$



Adjacência no Mapa de Karnaugh

- ▶ Agrupamento a partir da primeira para última coluna
- ▶ Agrupamento da linha do topo para a linha abaixo





Exemplos Mapa de Karnaugh

► $F =$

► $Cout =$

► $f(A,B,C) = \Sigma m(0,4,5,7)$

	A	
	1	0
B	0	0

B'

	A			
	0	0	1	0
Cin	0	1	1	1
	B			

$AB + ACin + BCin$

	A			
	1	0	0	1
C	0	0	1	1
	B			

$AC + B'C' + \cancel{AB'}$

Obtêm a complementação da função por cobertura de 0s com sub-cubos



Mais exemplos Mapa de Karnaugh

		A	
C	0	0	1
	0	0	1
		B	

$$G(A,B,C) = A$$

		A	
C	1	0	0
	0	0	1
		B	

$$F(A,B,C) = \sum m(0,4,5,7) = AC + B'C'$$

		A	
C	0	1	1
	1	1	0
		B	

F' substituição simples de 1's com 0's e vice versa

$$F'(A,B,C) = \sum m(1,2,3,6) = BC' + A'C$$

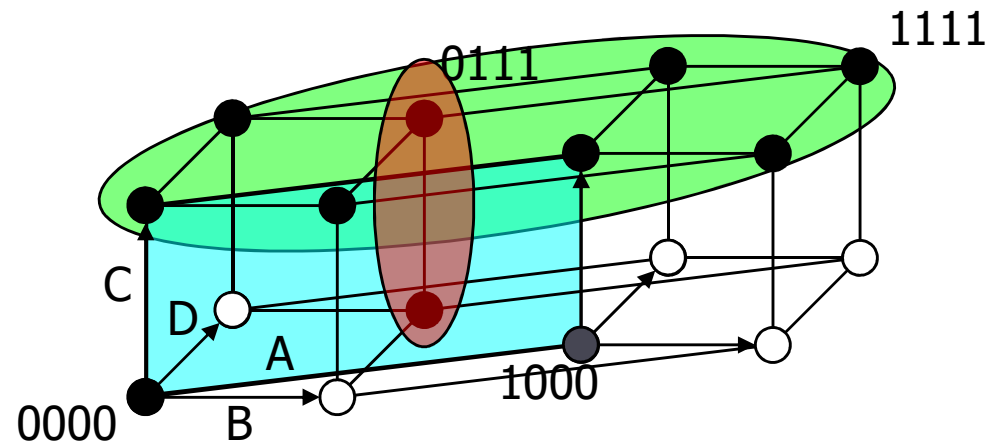


Mapa de Karnaugh: Exemplo 4-variáveis

► $F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15)$

$$F = C + A'BD + B'D'$$

				A	
		1	0	0	1
		0	1	0	0
C	1	1	1	1	
	1	1	1	1	
				D	
				B	



Encontrar o menor número dos maiores sub cubos possíveis para cobrir o ON-set
(menos termos com menos entradas por termo)



Mapa de Karnaugh: don't cares

- ▶ $f(A,B,C,D) = \Sigma m(1,3,5,7,9) + d(6,12,13)$
 - ▶ Sem don't cares
 - ▶ $f = A'D + B'C'D$

		A			
		0	0	X	0
C		1	1	X	1
		1	1	0	0
		0	X	0	0
		B		D	



Mapa de Karnaugh: don't cares (cont'd)

► $f(A,B,C,D) = \Sigma m(1,3,5,7,9) + d(6,12,13)$

► $f = A'D + B'C'D$

sem don't cares

► $f = A'D + C'D$

com don't cares

A			
0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0
B			

C

D

Usando don't care como "1"
2-cubos podem ser formados
ao invés de 1-cubo por cobertura

don't cares podem ser tratados como
1s ou 0s
dependendo de qual é mais vantajoso



Atividade

- Minimizar a função $F = \sum m(0, 2, 7, 8, 14, 15) + d(3, 6, 9, 12, 13)$

	A			
	1	0	X	1
	0	0	X	X
C	X	1	1	0
	1	X	1	0
	B			

$$F = \cancel{AC'} + \cancel{A'C} + \cancel{BC} + \cancel{AB} + \cancel{A'B'D'} + \cancel{B'C'D'}$$

$$F = BC + A'B'D' + B'C'D'$$

$$F = A'C + AB + B'C'D'$$

	A			
	1	0	X	1
	0	0	X	X
C	X	1	1	0
	1	X	1	0
	B			

	A			
	1	0	X	1
	0	0	X	X
C	X	1	1	0
	1	X	1	0
	B			



Resumo lógica combinacional

- ▶ Funções lógicas, tabelas verdade, e chaves
 - ▶ NOT, AND, OR, NAND, NOR, XOR, ..., conjunto mínimo
- ▶ Postulados e teoremas de álgebra booleana
 - ▶ Comprovar por reescrita e indução perfeita
- ▶ Portas lógicas
 - ▶ Redes de funções booleanas e seus comportamentos de tempo
- ▶ Forma canônica
 - ▶ Dois níveis e funções incompletamente especificadas
- ▶ Simplificação
 - ▶ Começar a entender simplificação dois níveis
- ▶ Mais tarde
 - ▶ Automatização de simplificação
 - ▶ Lógica multi níveis
 - ▶ Comportamento de tempo
 - ▶ Linguagem descrição de hardware
 - ▶ Projeto estudo de casos