CIRCUITOS LÓGICOS
MAPA DE KARNAUGH

Marco A. Zanata Alves
Luis Allan Künzle

Slides baseados nos slides de Rodrigo Hausen - CMCC – UFABC (2013)
http://compscinet.org/hausen/courses/circuitos/

SIMPLIFICAÇÃO DE LÓGICAS

Teorema: Toda função lógica pode ser escrita como disjunção de **mintermos** (também chamada "soma de produtos" – SOP).

Portanto, toda função lógica possui uma expressão que a define.

A forma de soma de produtos é uma forma padrão de representação de expressões booleanas. Outra forma padrão é o **produto de somas (maxtermos).**

REGRAS BÁSICAS DA ÁLGEBRA BOOLEANA

	Propriedade	OU	E
P1	Identidade	$X + 1 = 1$	$X \cdot 0 = 0$
P2	Elemento Neutro	$X + 0 = X$	$X \cdot 1 = X$
P3	Idempotência	$X + X = X$	$X \cdot X = X$
P4	Involução	$\overline{\overline{X}} = X$	$\overline{\overline{X}} = X$
P5	Complemento	$X + \overline{X} = 1$	$X \cdot \overline{X} = 0$
P6	Comutatividade	$X + Y = Y + X$	$X \cdot Y = Y \cdot X$
P7	Associatividade	$(X + Y) + Z = X + (Y + Z)$	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
P8	Distributividade	$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$	$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
P9	Cobertura	$X \cdot (X + Z) = X$	$X + (X \cdot Y) = X$
P10	Combinação	$(X \cdot Y) + (X \cdot \overline{Y}) = X$	$(X + Y) \cdot (X + \overline{Y}) = X$
P11	Consenso	$(X \cdot Y) + (\overline{X} \cdot Z) + (Y \cdot Z) = (X \cdot Y) + (\overline{X} \cdot Z)$	$(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \cdot (\overline{X} + Z)$
P12	De Morgan	$\overline{(X + Y)} = \overline{X} \cdot \overline{Y}$	$\overline{(X \cdot Y)} = \overline{X} + \overline{Y}$

SIMPLIFICAÇÃO DE LÓGICAS

Como podemos simplificar tal expressão?

$$\overline{V}FU\overline{N} + \overline{V}FUN + VFU\overline{N} + VFUN$$

SIMPLIFICAÇÃO DE LÓGICAS

Como podemos simplificar tal expressão?

$$\overline{V}FU\overline{N} + \overline{V}FUN + VFU\overline{N} + VFUN$$

$$FU(\overline{VN} + \overline{V}N + V\overline{N} + VN)$$

$$FU\left(\overline{V}\left(\overline{N} + N\right) + V\left(\overline{N} + N\right)\right)$$

$$FU\left(\overline{V} + V\right)$$

$$FU$$

SIMPLIFICAÇÃO DE LÓGICAS

Observe que, quando temos algo do tipo:

$$\ldots + A\overline{B} + AB + \cdots$$

em uma expressão na forma soma-de-produtos podemos colocar A em evidência:

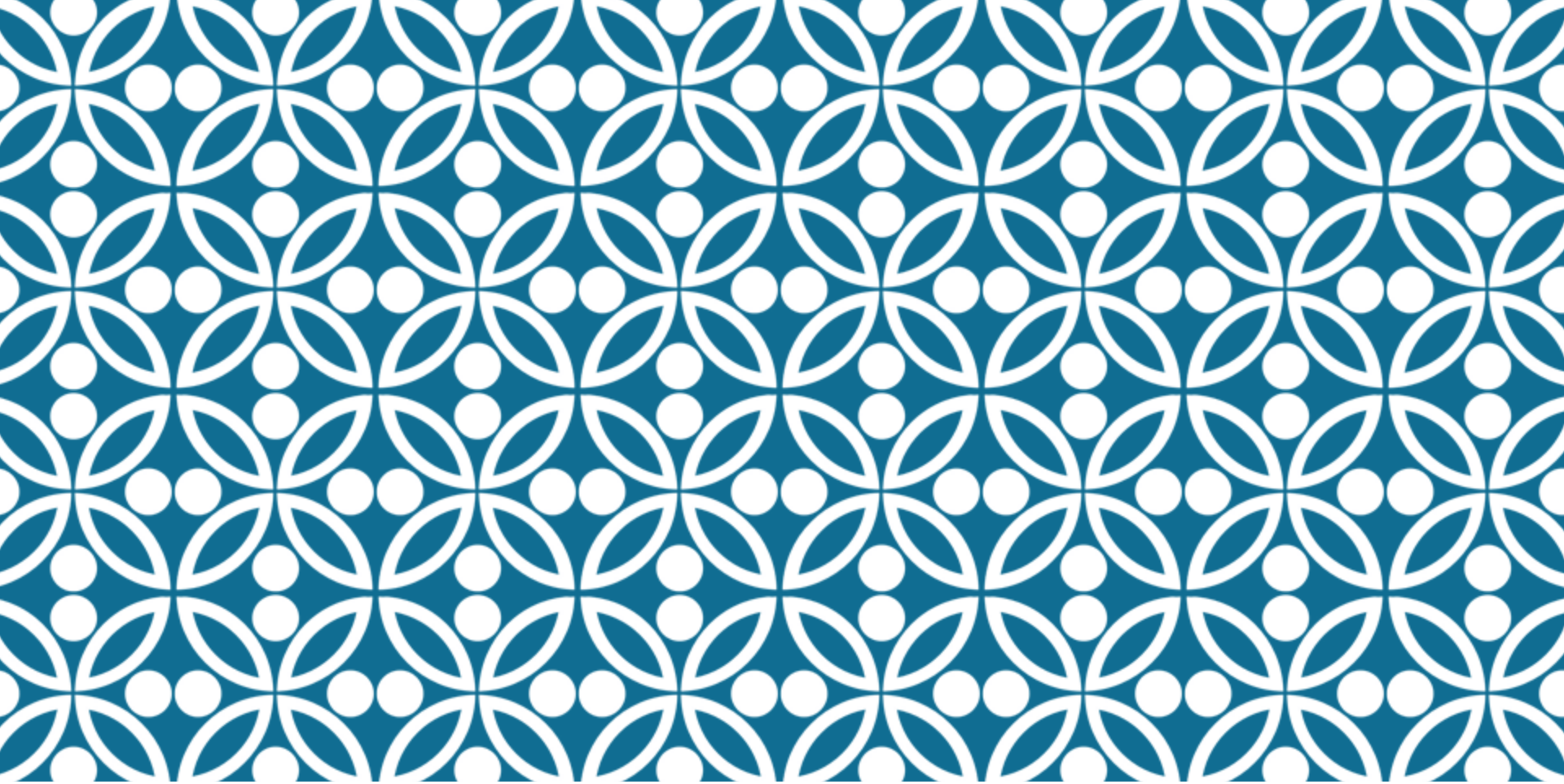
$$\ldots + A(\overline{B} + B) + \cdots$$

e simplificar por:

$$\ldots + A + \cdots$$

O problema sempre foi: Como encontrar dois mintermos idênticos a menos de uma mesma variável B, que aparece como B e $\overline{B}$?

Solução: expresse a tabela verdade de forma que isso seja fácil de encontrar!



GRAY CODING "CODIFICAÇÃO CINZA"

GRAY CODING

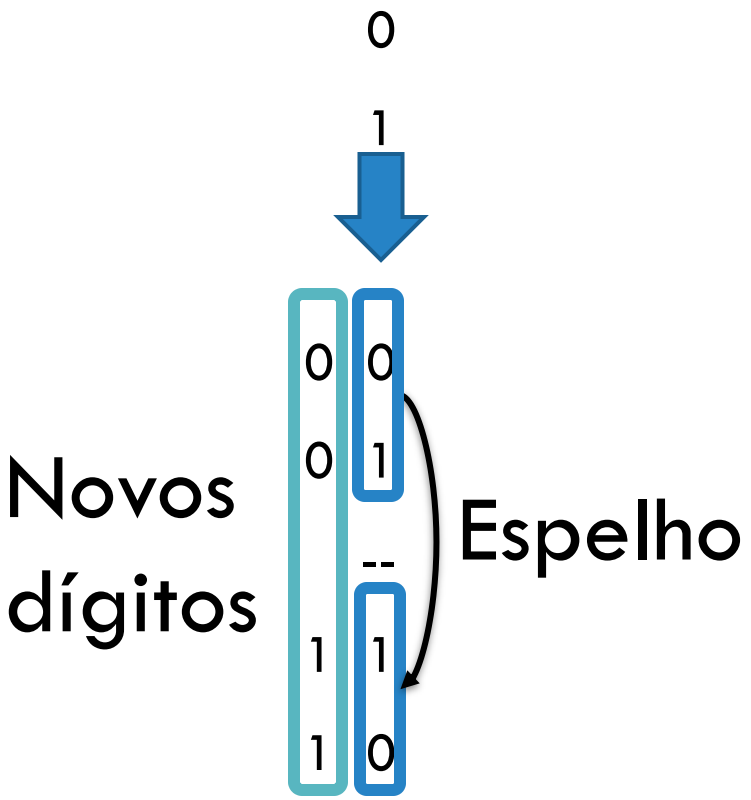
Trata-se de um algoritmo para geração de uma sequência de números onde apenas um bit muda por vez

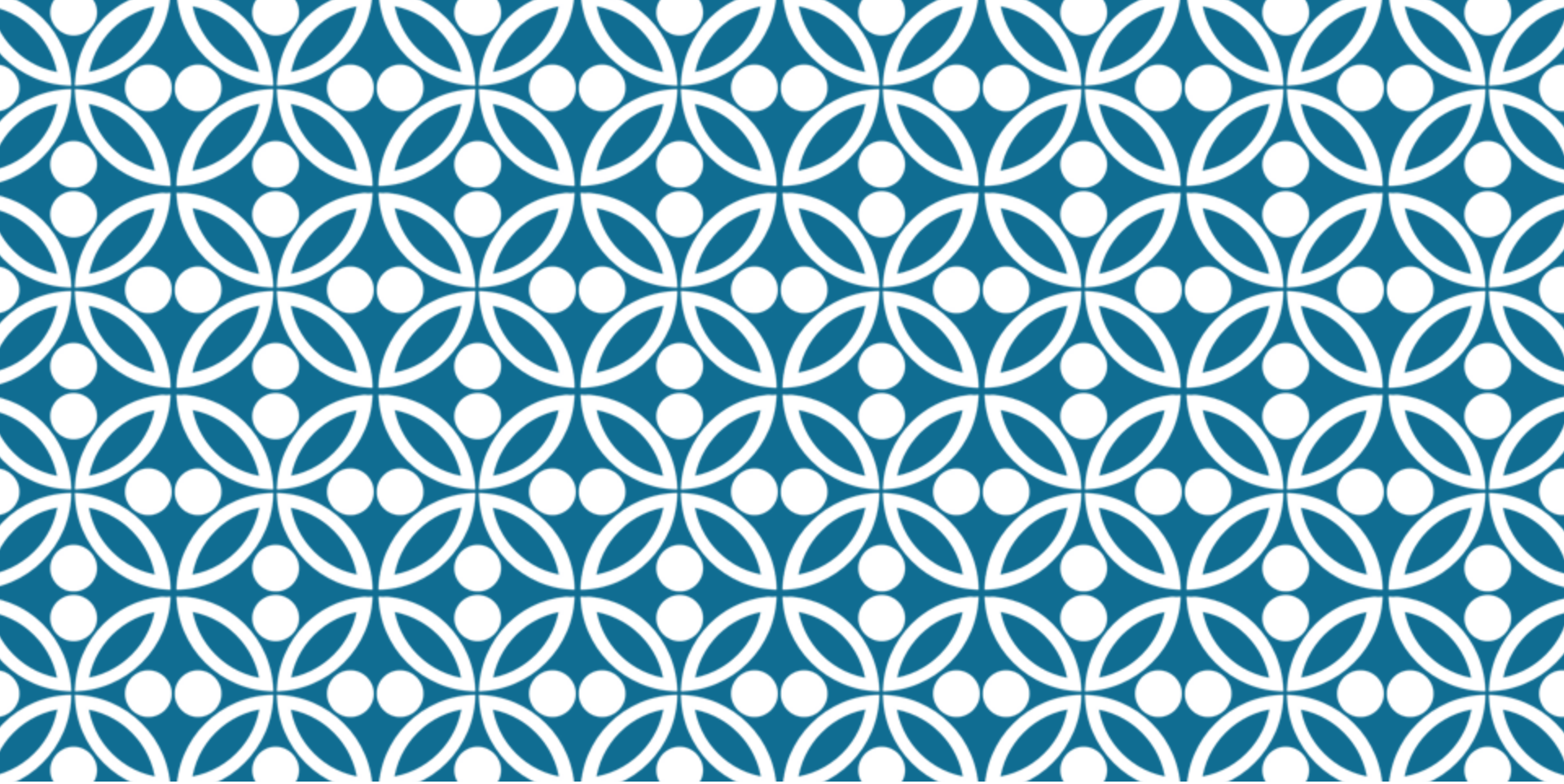
GRAY CODING

Para criar uma codificação gray, começamos com 1 dígito (0 ou 1)

Para cada novo dígito a ser adicionado uma função de espelho é aplicada

E em cada parte do espelho adiciona-se 0s ou 1s





MAPAS DE KARNAUGH

MAPAS DE KARNAUGH

Considerando a seguinte tabela verdade

Como gerar a lógica que implementa a função f(A,B,C) ?

Como implementar essa lógica de maneira otimizada?

R1. Regras de álgebra booleana

R2. **Mapas de Karnaugh**

A	B	C	f	
0	0	0	0	
0	0	1	0	
0	1	0	1	<= !A · B · !C
0	1	1	1	<= !A · B · C
1	0	0	1	<= A · !B · !C
1	0	1	0	
1	1	0	0	
1	1	1	1	<= A · B · C

MAPAS DE KARNAUGH...

Os mapas de Karnaugh permite simplificar funções utilizando **técnicas de mapeamento visual**

Para iniciar, precisamos criar uma grade para mapear as entradas (ex. A,B,C)

A	B	C	f	
0	0	0	0	
0	0	1	0	
0	1	0	1	<= !A · B · !C
0	1	1	1	<= !A · B · C
1	0	0	1	<= A · !B · !C
1	0	1	0	
1	1	0	0	
1	1	1	1	<= A · B · C

MAPAS DE KARNAUGH...

Representação em matriz para a tabela verdade, onde **em linhas ou colunas adjacentes apenas uma variável muda de 1 para 0 ou vice-versa.**

A\BC	00	01	11	10
0	0	0	1	1
1	1	0	1	0

A	B	C	f	
0	0	0	0	
0	0	1	0	
0	1	0	1	<= !A · B · !C
0	1	1	1	<= !A · B · C
1	0	0	1	<= A · !B · !C
1	0	1	0	
1	1	0	0	
1	1	1	1	<= A · B · C

A · !B · !C

MAPAS DE KARNAUGH...

A\BC	00	01	11	10
0	0	0	1	1
1	1	0	1	0

$!A \cdot B\ (C + !C) = \mathbf{!A \cdot B}$

MAPAS DE KARNAUGH

A\BC	00	01	11	10
0	0	0	1	1
1	1	0	1	0

A · !B · !C

A · !A (B · C) = **B · C**

!A · B (C + !C) = **!A · B**

A!B!C+B(!A+C)

MAPAS DE KARNAUGH

Simplifique:

$$C = \overline{V}\ F\ \overline{U}\ \underline{N} + V\ \overline{F}\ \overline{U}\ N + \overline{V}\ \overline{F}\ UN + \overline{V}FUN + VFUN + \overline{V}\ FU\ \overline{N} + VFU\overline{N} + V\ \overline{F}\ U\ \overline{N}$$

MAPAS DE KARNAUGH

\UN VF\	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	0	1	1
10	0	1	0	1

$$C = FU + \overline{V}FN + \overline{V}UN + VUN + V\overline{F}\ \overline{U}N$$

MAPAS DE KARNAUGH

Simplifique:

$$C = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + AB\overline{C}$$

MAPAS DE KARNAUGH - EXEMPLOS

A\BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1

$\Rightarrow$ **(!B · !C) + (B · !C)**

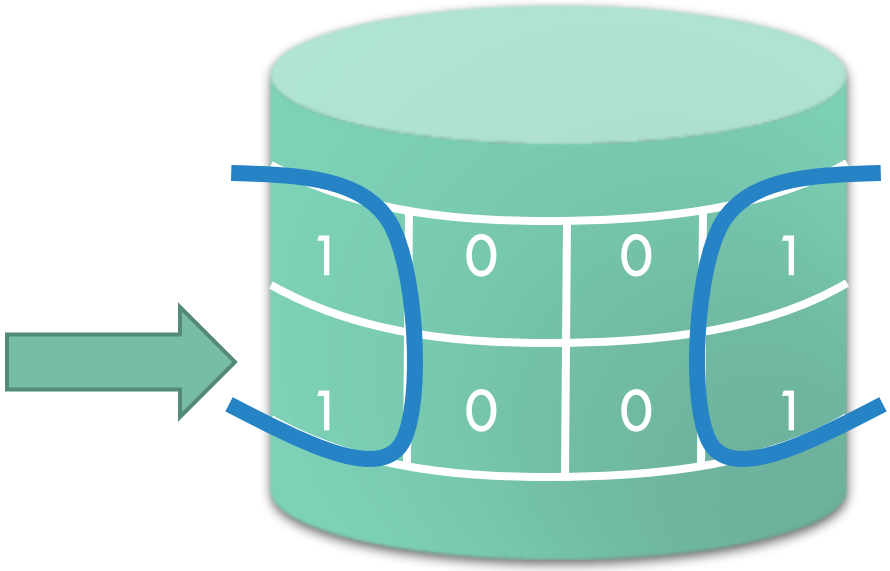
$\Rightarrow$ **!C(!B+B)**

$\Rightarrow$ **!C**

Será que poderíamos inferir isso pelo mapa?

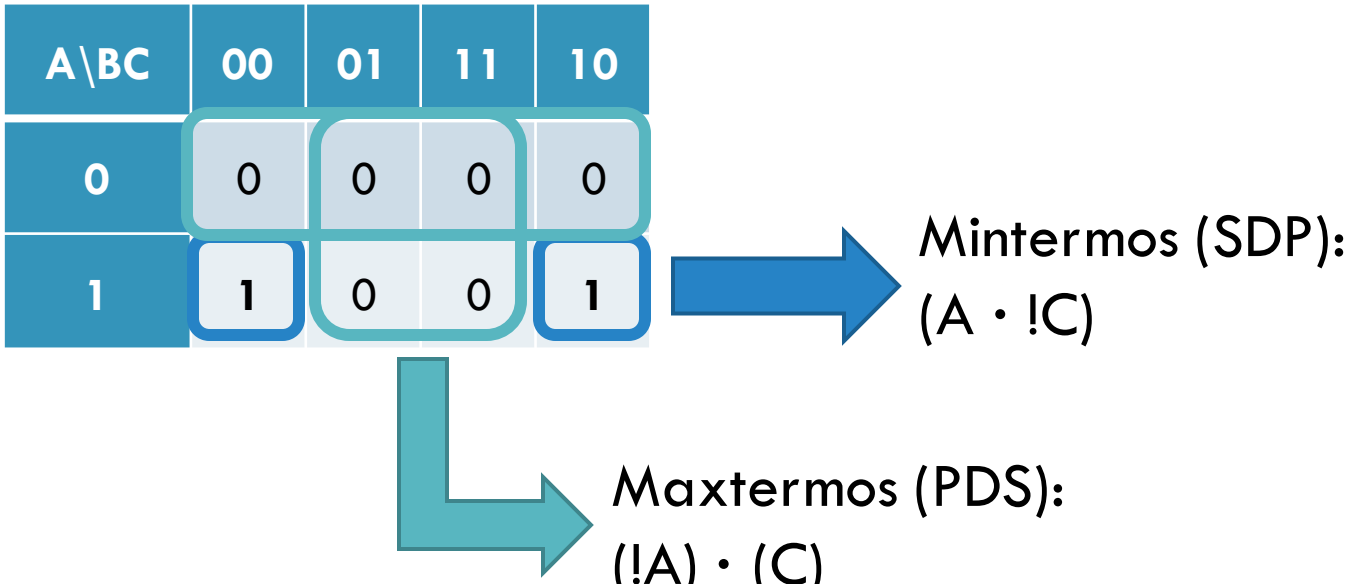
MAPAS DE KARNAUGH - EXEMPLOS

A\BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1



=> !C

MAPAS DE KARNAUGH - EXEMPLOS



MAPAS DE KARNAUGH

Note que nosso mapa de Karnaugh não seguiu a ordem crescente para os termos

Isso se deve a um requisito dos mapas de Karnaugh, entre células adjacentes, deve ter apenas 1 bit mudando por vez

Podemos gerar uma lista dessas utilizando **codificação gray**

A\BC	00	01	11	10
0	0	0	1	1
1	1	0	1	0

ALGORITMO DE MAPAS DE KARNAUGH PARA ATÉ 4 VARIÁVEIS

1) Expresse a tabela verdade como uma matriz, com no máximo duas variáveis para as linhas/colunas. Em linhas ou colunas adjacentes, apenas uma das variáveis muda (use gray coding).

2) Enquanto houver uma célula contendo 1 que não tiver sido agrupada, agrupe nesta ordem:

- Retângulos com 16 uns (Obs.: se houver, então F = 1)
- Retângulos com 8 uns (2x4 ou 4x2)
- Retângulos com 4 uns (1x4, 4x1 ou 2x2)
- Retângulos com 2 uns (1x2 ou 2x1)

Importante: a última linha/coluna é adjacente à primeira linha/coluna.

3) Elimine grupos redundantes (se puder)

4) Para cada grupo, escreva uma soma de produtos onde apenas as variáveis que não mudaram são representadas.
Importante: Se, no grupo, uma variável W é mantida em 0, então escreva $\overline{W}$.

EXEMPLO

Simplifique $F(A, B, C, D)$, cuja tabela verdade é dada pelo mapa de Karnaugh ao lado.

ab \ cd	00	01	11	10
00	1	1	0	0
01	1	1	1	1
11	0	0	0	0
10	0	1	1	0

EXEMPLO

Simplifique $F(A, B, C, D)$, cuja tabela verdade é dada pelo mapa de Karnaugh ao lado.

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

ab \ cd	00	01	11	10
00	1	1	0	0
01	1	1	1	1
11	0	0	0	0
10	0	1	1	0

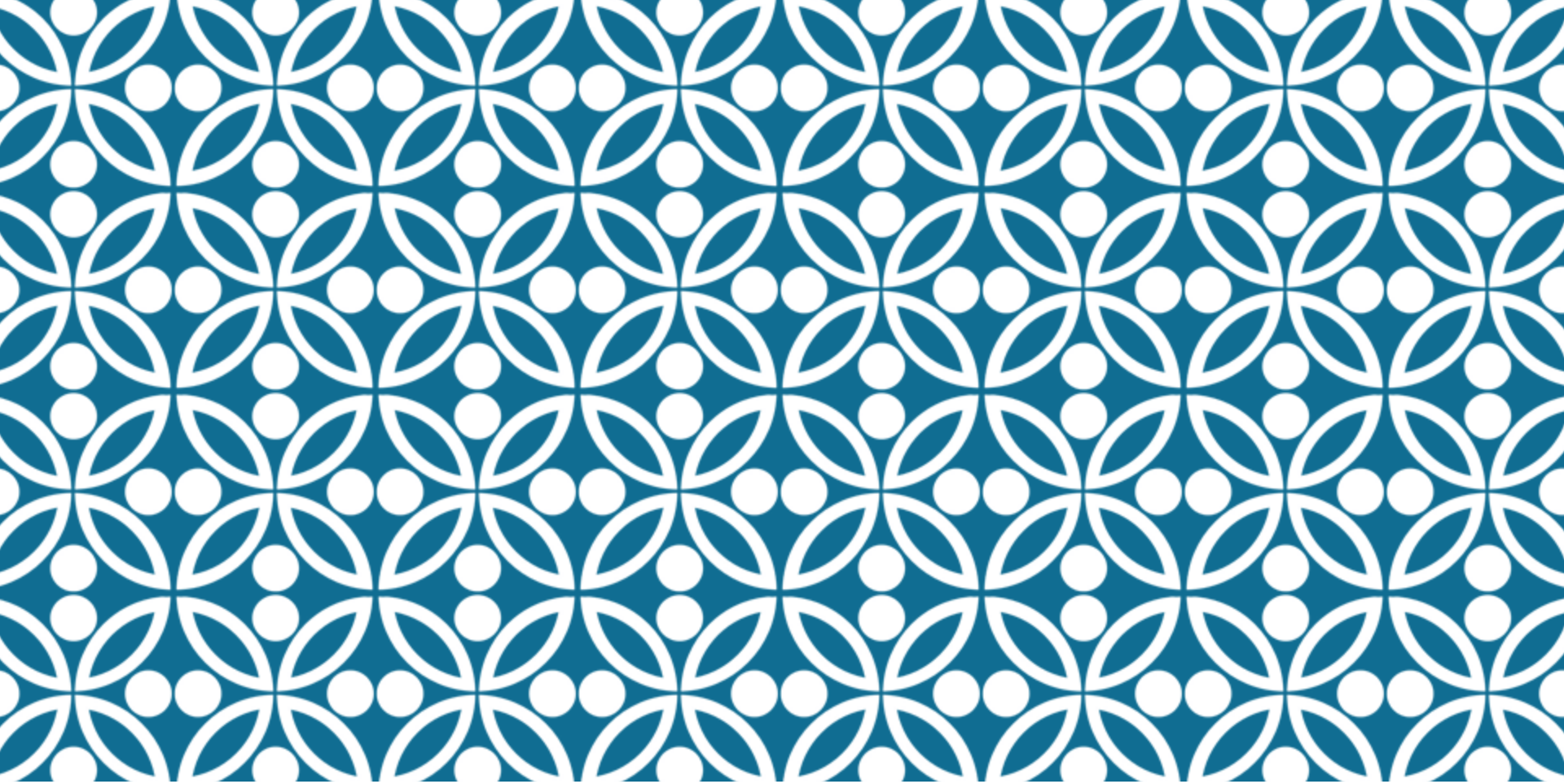
EXERCÍCIOS:

Simplifique:

1) A’B’C’+A’B’C+ABC’+AB’C’

2) A’B’C+A’BC’+ABC’+ABC

3) ABCD+ABC’D+ABC’D’+AB’CD+A’BCD+A’BCD’+A’BC’D+A’B’C’D



BITS DE DON'T-CARE

BITS DON'T-CARE (SEM IMPORTÂNCIA)

Bit de Don't-Care (X) é uma sequência de entrada para qual a saída não importa.

Uma entrada a qual é conhecida por nunca ocorrer é chamada de Can't-Happen.

Os dois casos são tratado de mesma forma no projeto de lógica, e são referidos de modo genérico por don't-care.

O projetista não precisa se importar com a saída gerada para esses termos, logo essa saída pode ser tratada da forma que for mais conveniente (gerar menor circuito).

Ex. valores binários 1010~1111 (em um circuito BCD)

BIT DON'T CARE (X) - EXEMPLOS

Don´t cares (X) podem ser utilizados se necessário para aumentar a cobertura e assim simplificar a lógica

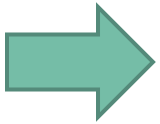
A\BC	00	01	11	10
0	0	X	0	0
1	X	X	1	1

=> A · B

BIT DON'T CARE (X) - EXEMPLOS

Don´t cares (X) podem ser utilizados se necessário para aumentar a cobertura e assim simplificar a lógica

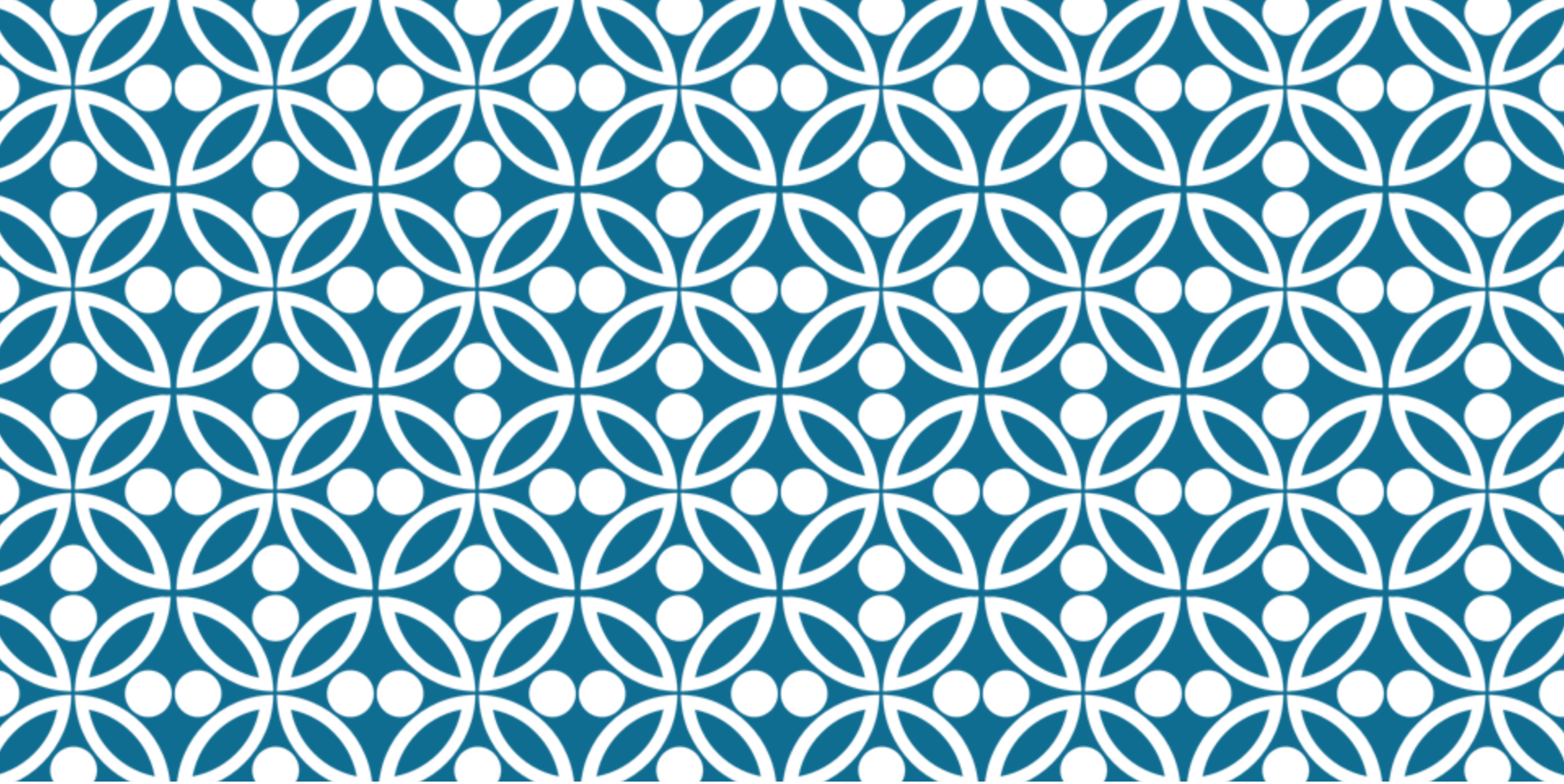
A\BC	00	01	11	10
0	0	X	0	0
1	X	X	1	1



A\BC	00	01	11	10
0	0	X	0	0
1	X	X	1	1

=> A · B

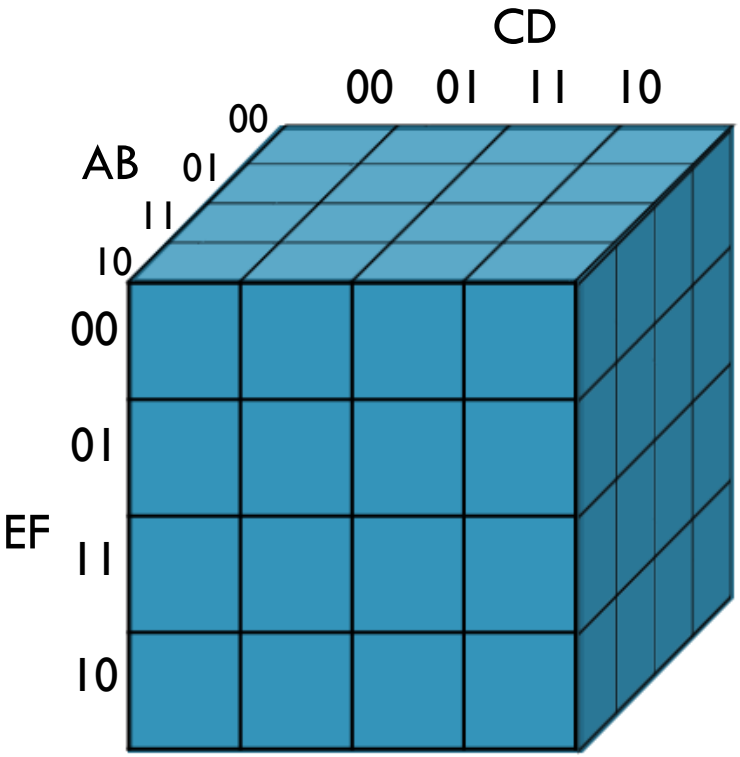
=> A



MAPAS DE KARNAUGH MAIS DE 4 VARIÁVEIS

MAPAS DE KARNAUGH PARA MAIS DE 4 VARIÁVEIS

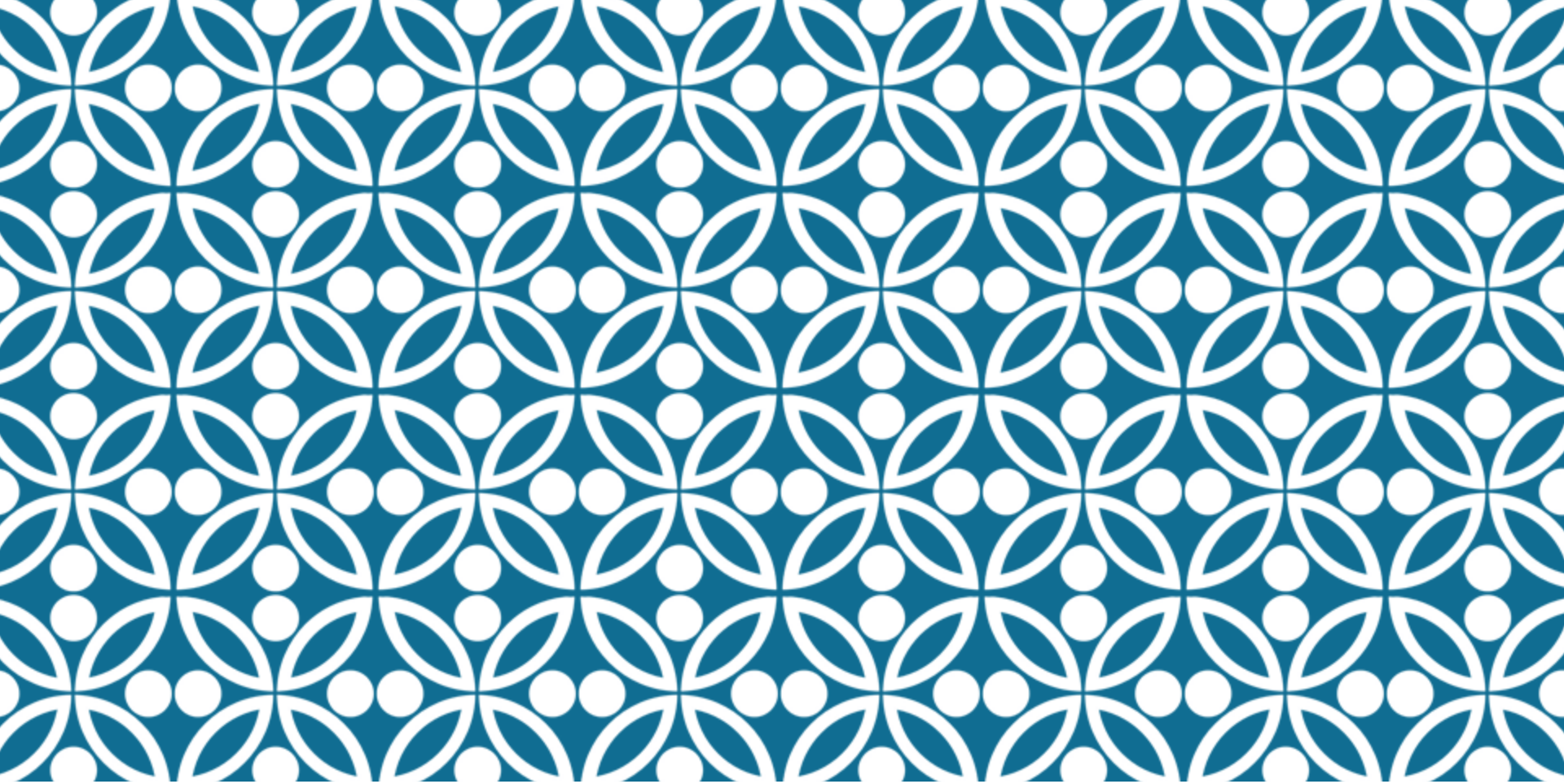
É possível construir mapas de Karnaugh para mais de 4 variáveis, mas eles se tornam difíceis de representar pois o mapa torna-se um cubo:



MAPAS DE KARNAUGH PARA MAIS DE 4 VARIÁVEIS

Entre 4 e 30 (aprox.) variáveis, é possível executar o método de Quine-McCluskey, que é exato mas possui complexidade exponencial.

Acima de 30 variáveis, há o minimizador Espresso, baseado em métodos heurísticos (não exato).



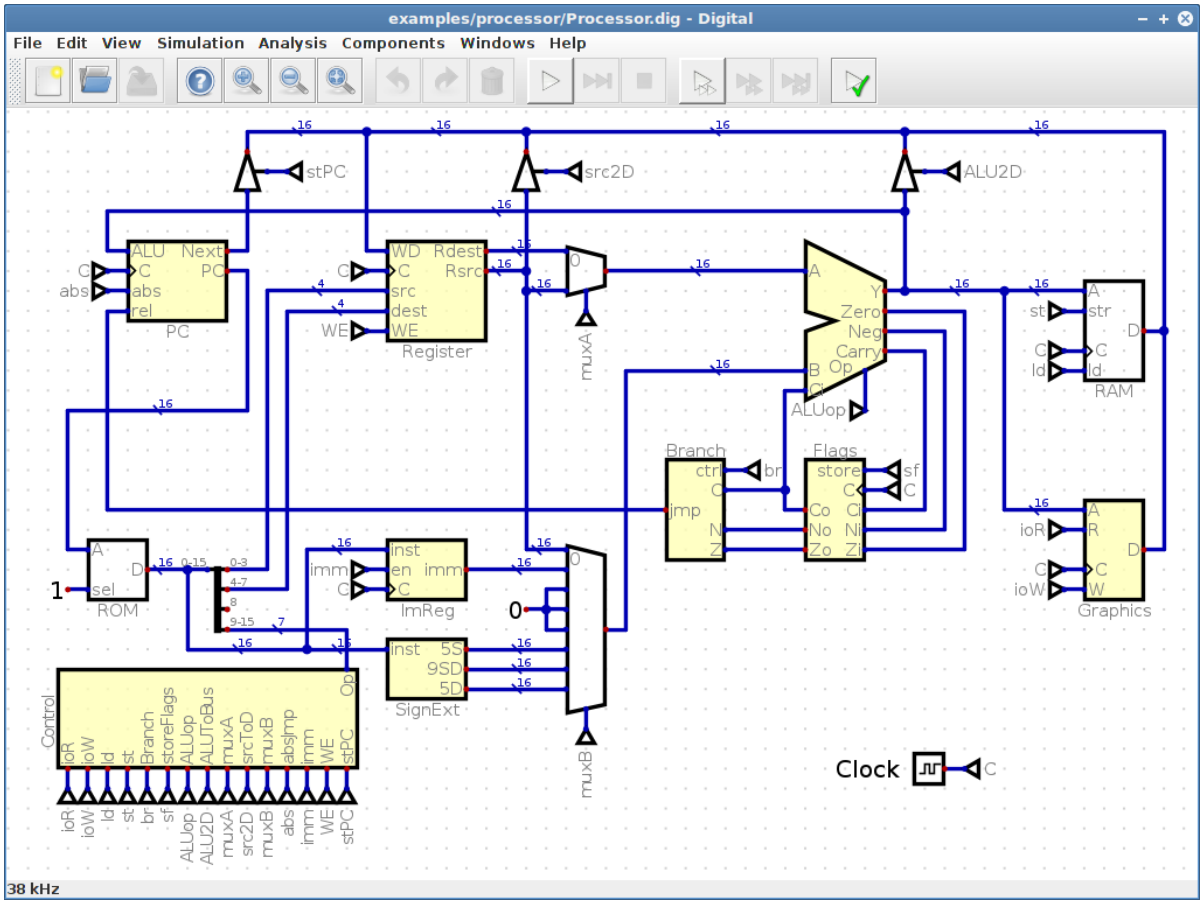
LOGISIM

SIMULADOR "DIGITAL"

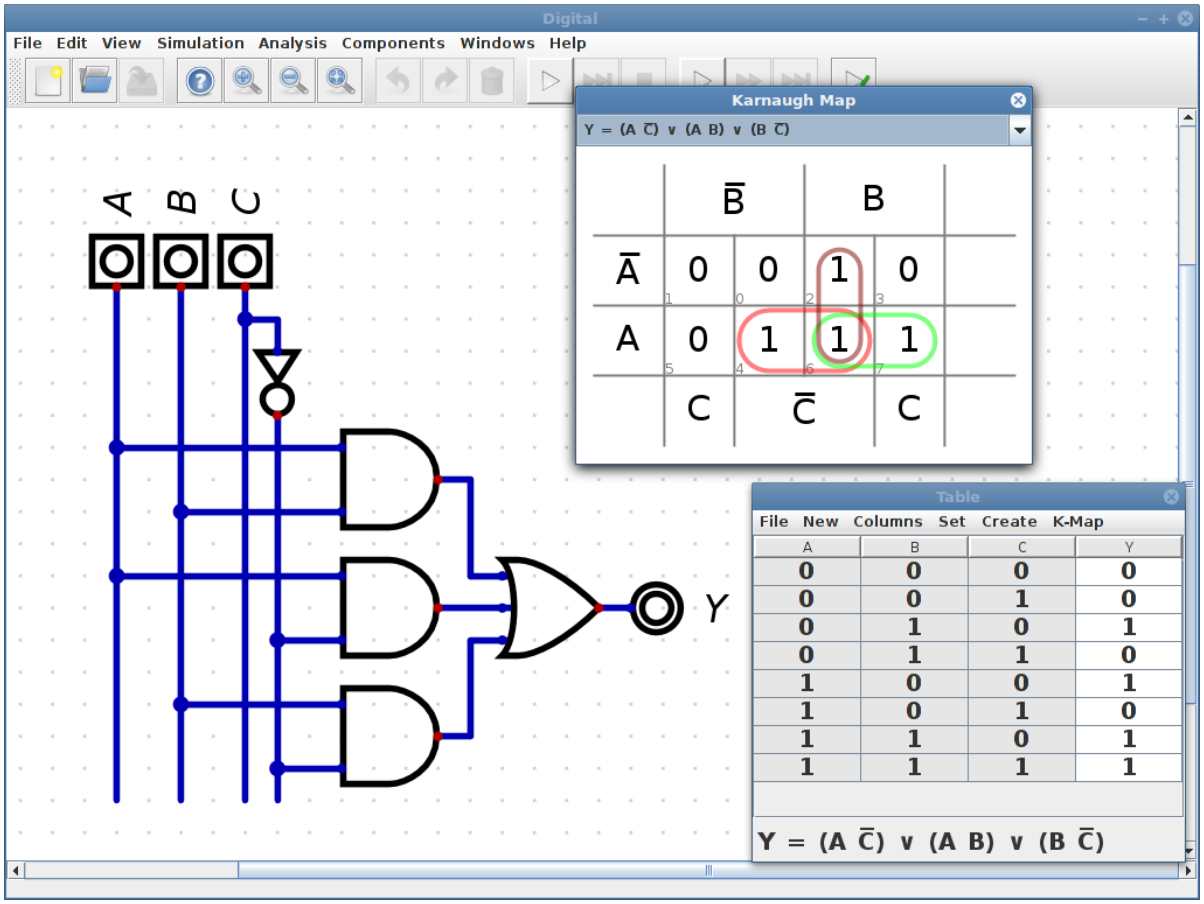
A partir de agora, usaremos o simulador de circuitos Digital:

https://github.com/hneemann/Digital

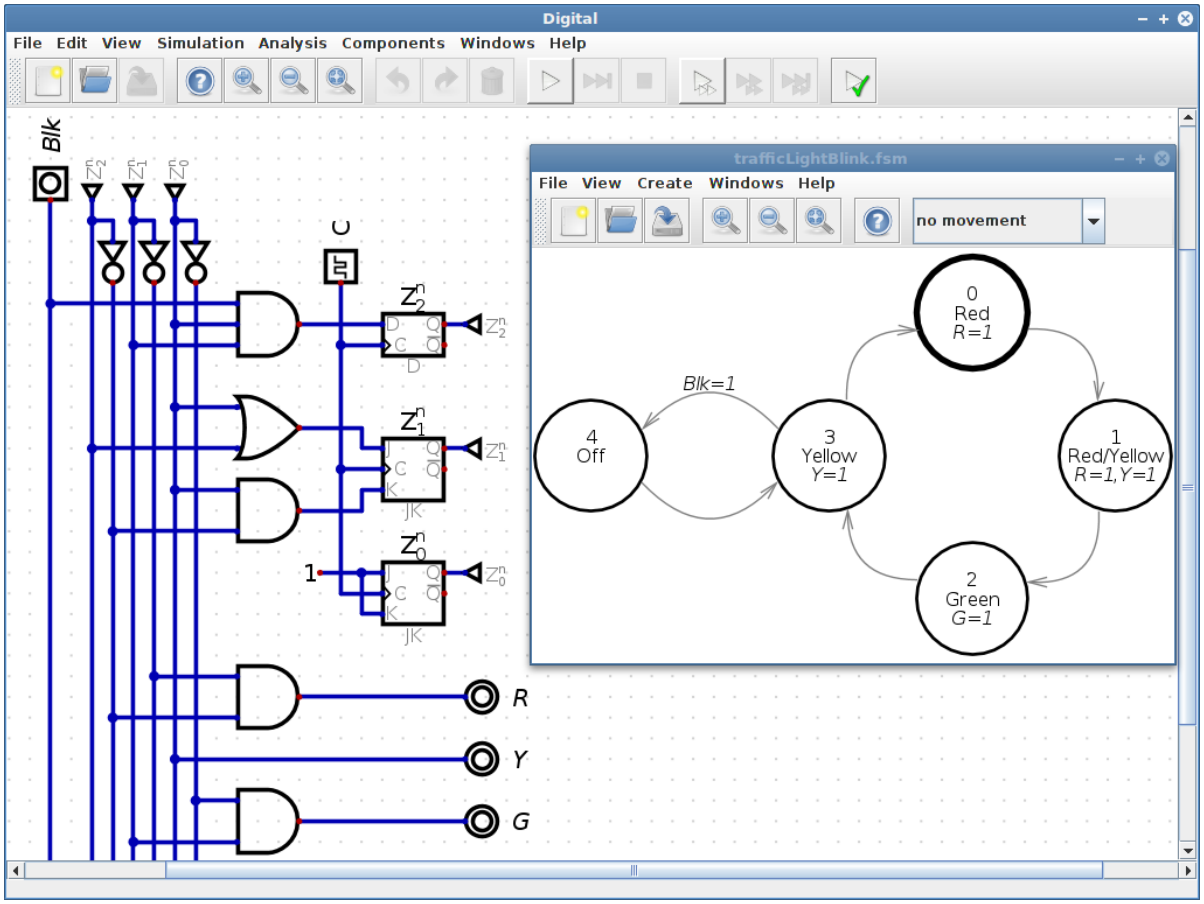
SIMULADOR "DIGITAL"



SIMULADOR "DIGITAL"



SIMULADOR "DIGITAL"



OS QUATRO PASSOS PARA UM CIRCUITO DIGITAL FELIZ.

1. Obtenha e simplifique a expressão lógica para cada saída.

2. Desenhe o diagrama do circuito. Cuidado com os erros comuns: esquecer de ligar entradas de portas lógicas, juntar saídas sem usar porta lógica (cheiro de circuito torrado no ar), esquecer de fazer a soma dos produtos (ou seja, omitir a porta or), etc.

OS QUATRO PASSOS PARA UM CIRCUITO DIGITAL FELIZ.

3. Analise o circuito e verifique as saídas. Vá "caminhando" das entradas em direção às saídas, escrevendo na saída de cada porta lógica a expressão equivalente.

4. Monte o circuito, faça a sua tabela verdade e verifique se ela coincide com as tabelas verdade das expressões obtidas no primeiro passo. Primeiro simule, depois monte o circuito em uma placa padrão