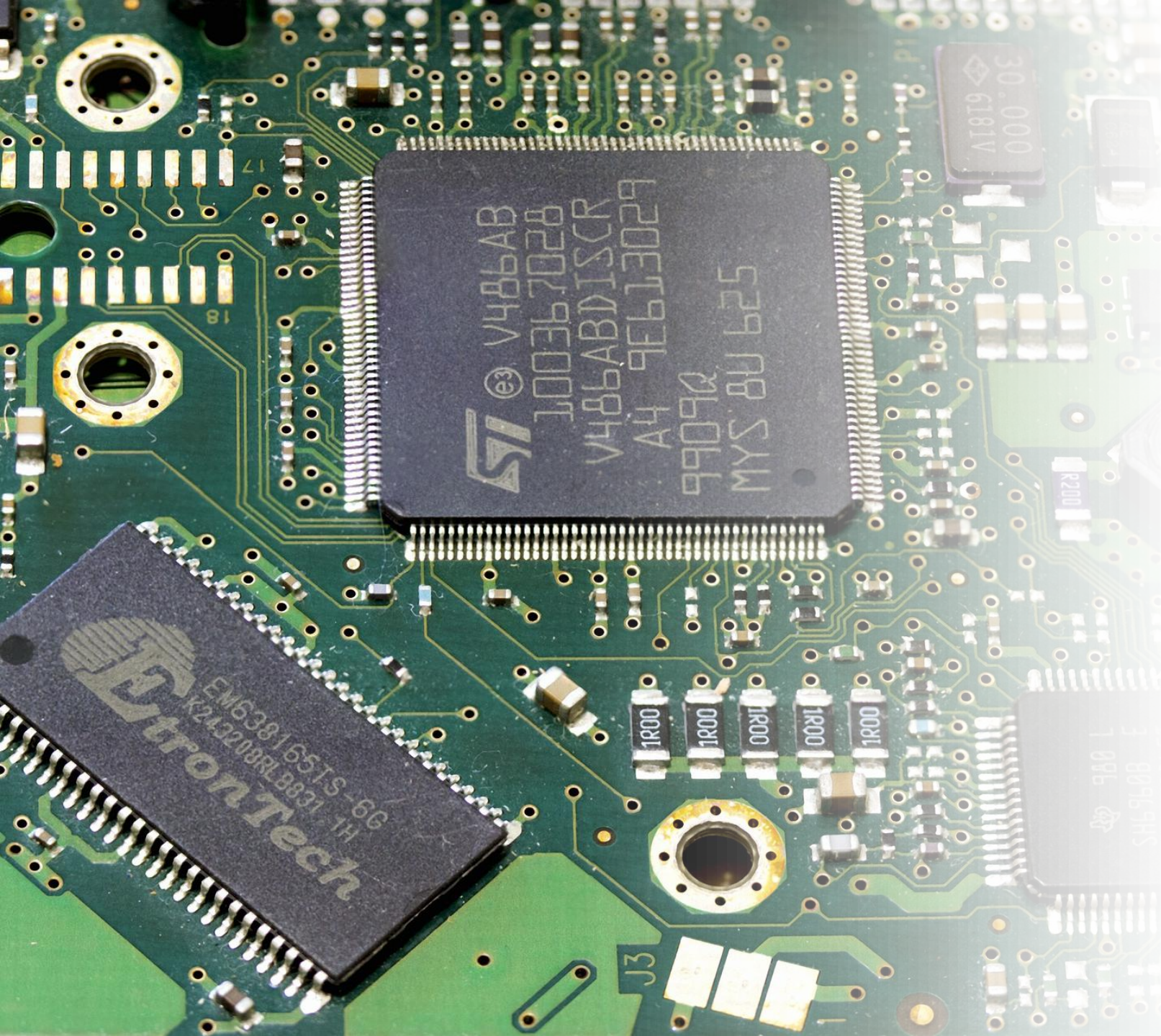




Digital Logic and System Design

4. Gate-level Minimisation

COL215, I Semester 2024-2025

Venue: LH 408

'E' Slot: Tue, Wed, Fri 10:00-11:00

Instructor: Preeti Ranjan Panda

panda@cse.iitd.ac.in

www.cse.iitd.ac.in/~panda/

Dept. of Computer Science & Engg., IIT Delhi

Optimising a Gate-level Implementation

- Improving a gate-level design
- Objective?
 - Area
 - Delay
 - Power/Energy
 - Temperature
 - Testability/Reliability/Security/...
- Smaller is better
 - Always?

Literal Count

- Optimal circuit: **minimum number of literals**
- Focus on **Area**
 - **#literals** proportional to **#transistors**
- Simplification in using Literals:
 - Ignoring complement operations
 - Ignoring wires
 - Ignoring blank spaces
 - Ignoring transistor dimensions

Circular Design Flow

- Literal Count useful in spite of limitations
 - Fast decisions
 - Don't need to generate full gate-level circuit, physical layout
- Gate-level decision
 - depends on geometry
 - ...which depends on gate-level decision 😊
- **Fast estimates** are critical!

Logic minimisation

- Optimization by applying Boolean Algebra theorems
 - Problem: in what order?
- **Map** method (**Karnaugh Map** or **K-Map**)
 - Pictorial representation of Truth Table
 - **2ⁿ squares/cells** for n-variable function
 - one square for each input combination (truth table row)
 - value = **1** if corresponding minterm included in function
 - otherwise, value = **0**

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth Table

0	1	1	0
1	0	0	1

Map of function F

2-variable Karnaugh Map

a	b	F
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table



a\b	0	1
0	0	1
1	1	0

Karnaugh Map of F

	b'	b
a'	00 $a'b'$	01 $a'b$
a	10 ab'	11 ab

a\b	0	1
0	00 m_0	01 m_1
1	10 m_2	11 m_3

a\b	0	1
0	0	1
1	1	0

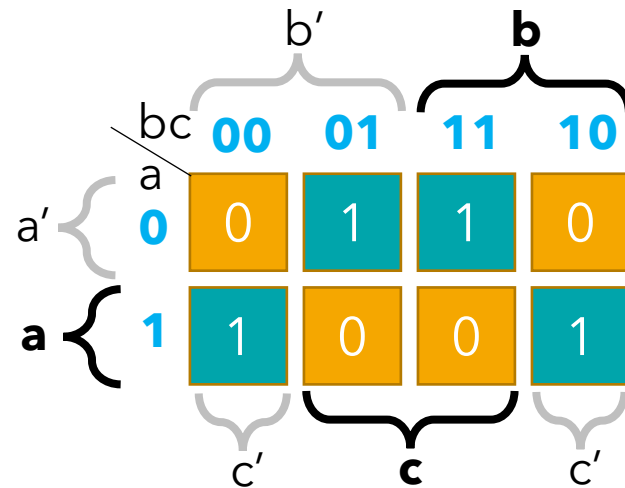
$a'b$
 ab'

$$F = a'b + ab'$$

3-variable Karnaugh Map

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth Table



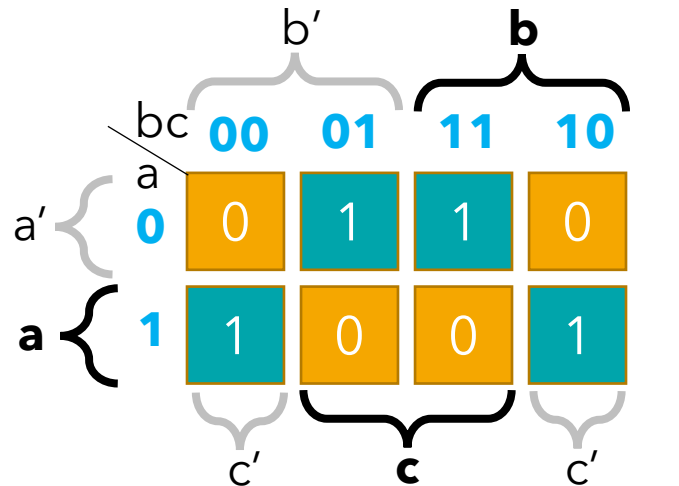
Gray Code sequence
Why?

$$F = a'b'c + a'bc + ab'c' + abc'$$

3-variable Karnaugh Map

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Truth Table



$$F = a'b'c + a'bc + ab'c' + abc'$$

bc	00	01	11	10
a				
0	m ₀	m ₁	m ₃	m ₂
1	m ₄	m ₅	m ₇	m ₆

$$F = m_1 + m_3 + m_4 + m_6 \\ = \sum (1, 3, 4, 6)$$

4-variable Karnaugh Map

		cd 00 01 11 10 ← Gray Code sequence			
ab	00	0	1	1	0
	01	0	1	1	0
	11	1	0	0	0
	10 ↑ Gray Code sequence	1	1	0	0

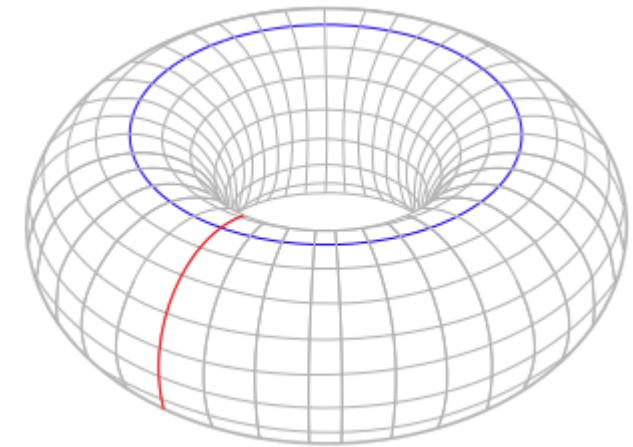
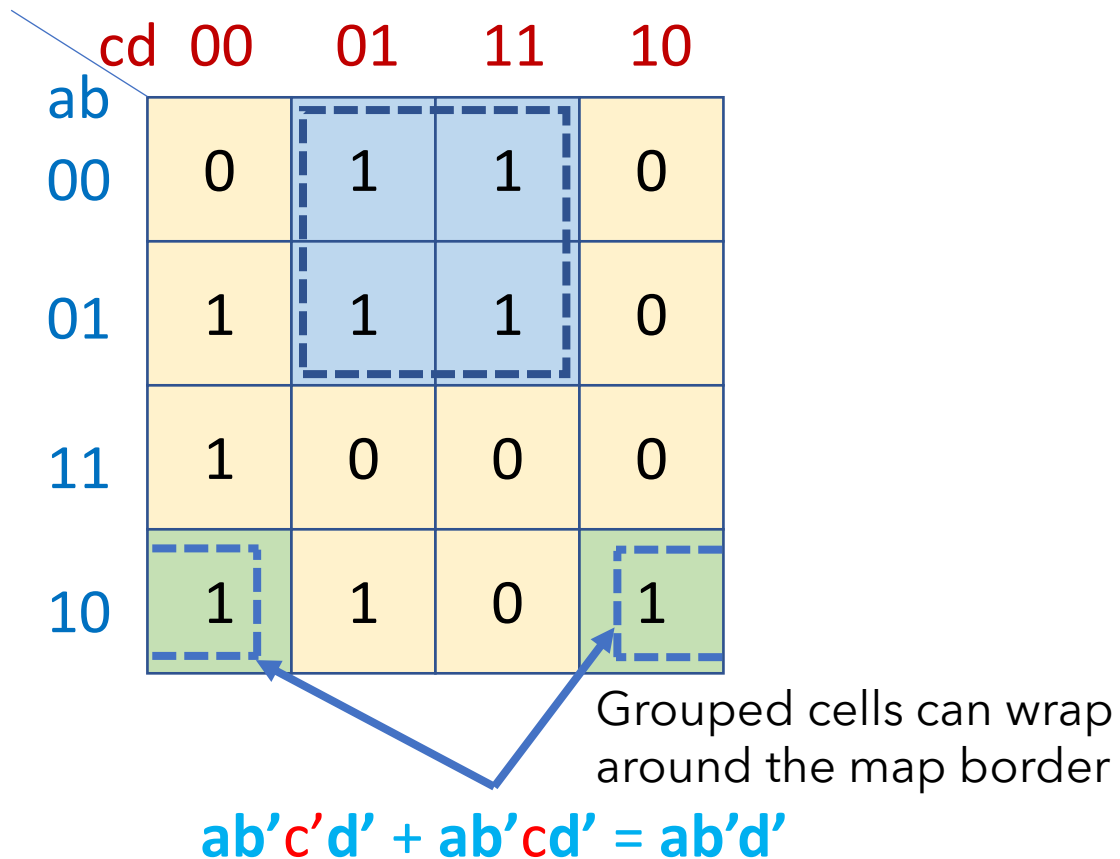
Grouping of cells in K-Map

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

$a'b'c'd + a'b'cd + a'bc'd + a'bcd = a'd$

Group represents **term**
Larger group means **smaller term**

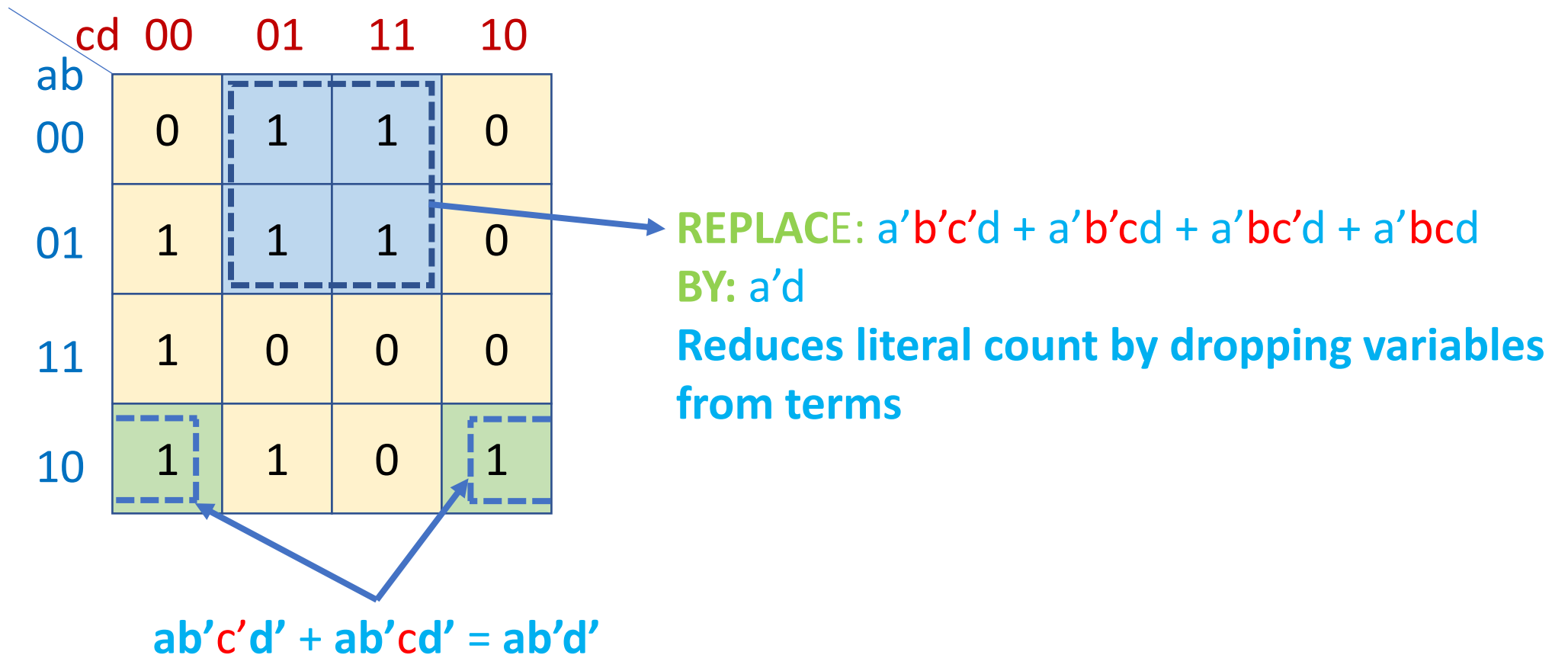
Wrapping around



Groups/Regions can wrap around, as in a **Torus**

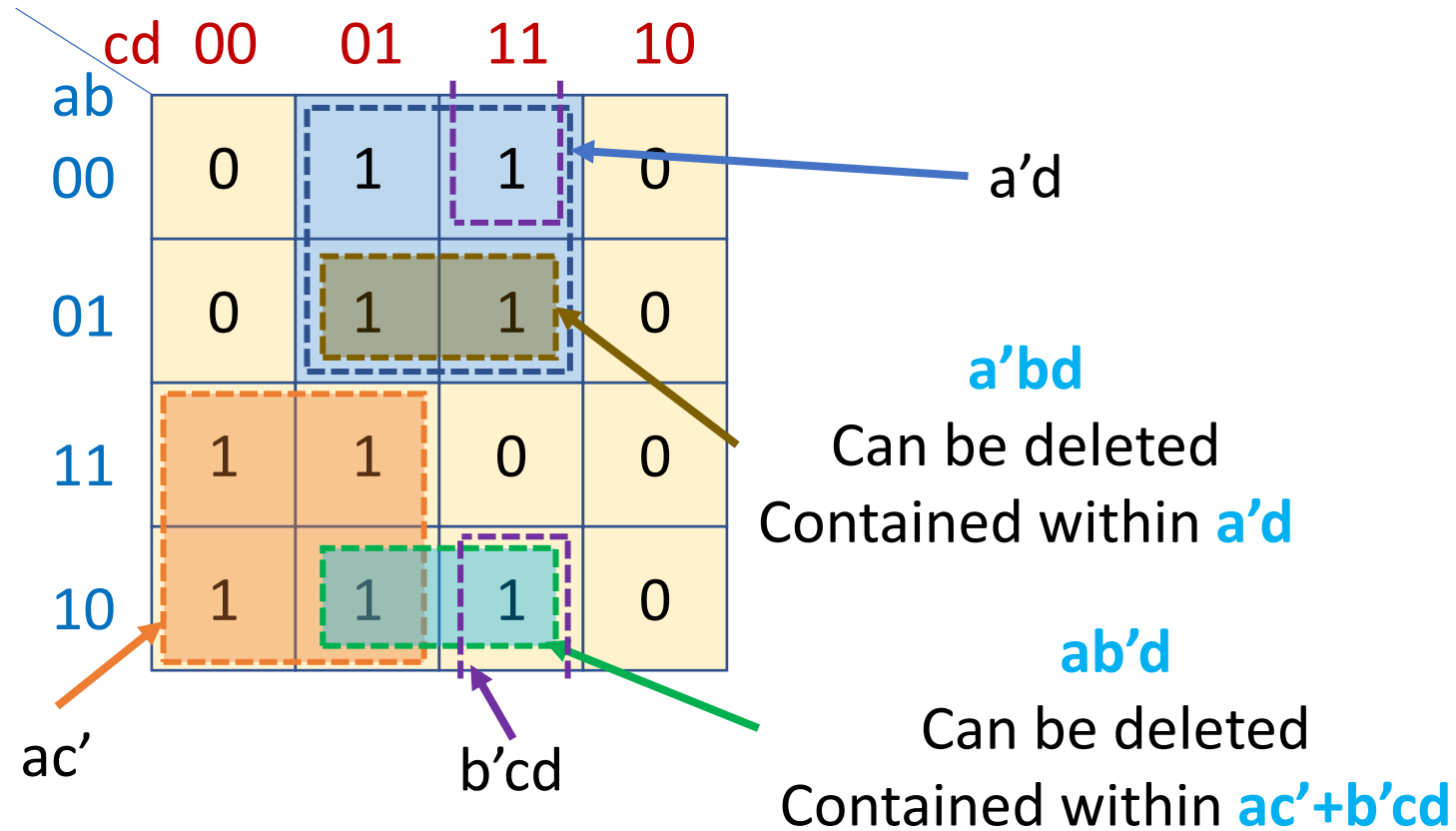
[Figure Source: Wikimedia
https://commons.wikimedia.org/wiki/File:Simple_torus_with_cycles.svg]

Cell grouping: minimizing Boolean expressions

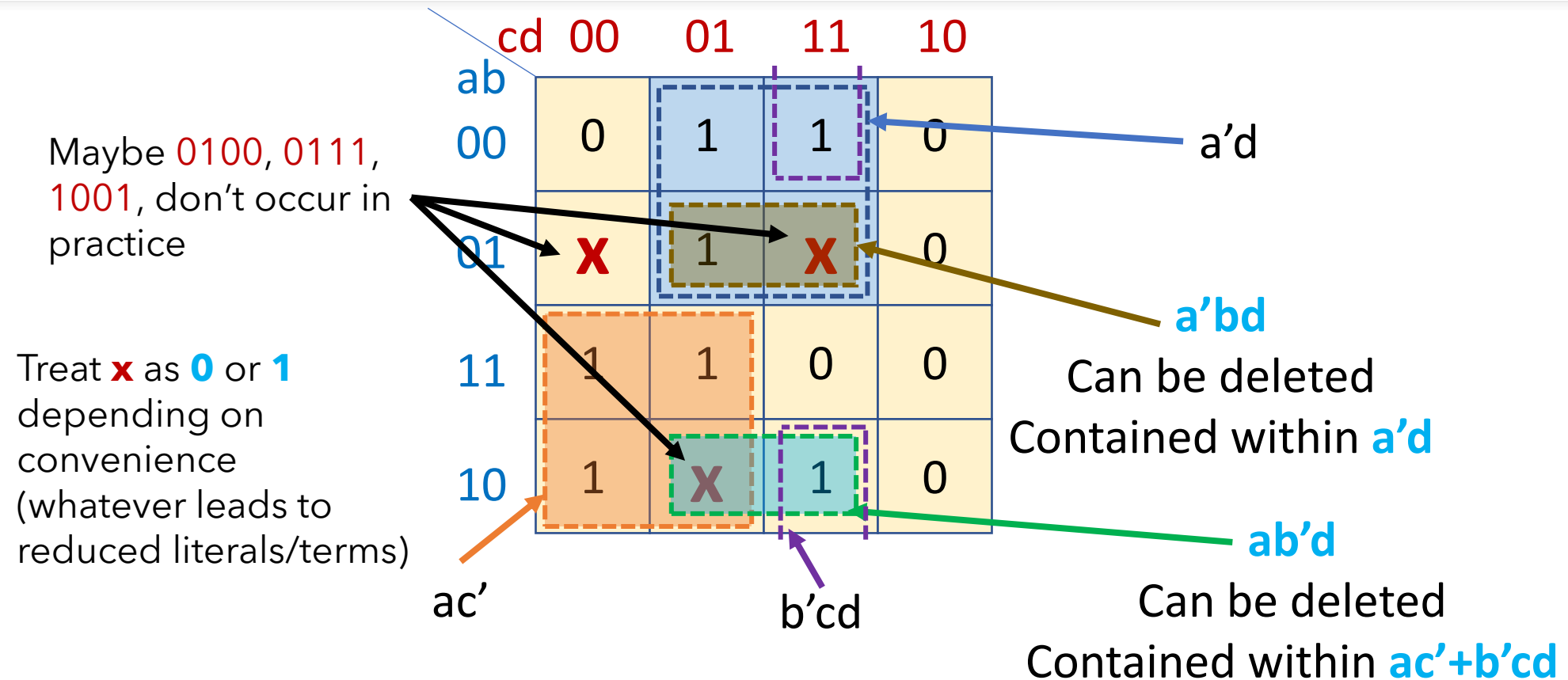


Deleting redundant groups

Second optimization:
dropping terms



Don't Care: Input combination ignored



Terminology: Implicants

Essential Prime Implicant

Must be included in any optimal covering of function

Prime Implicant
Not contained in any other implicant

Implicant

If term is 1, then function is 1

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

Product of Sums Simplification

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

Diagram illustrating a 4x4 Karnaugh map for Product of Sums Simplification. The map shows the function F' (complement of F) with 1s in the first three columns and 0s in the last column. The map is partitioned into two groups: ac (orange dashed box) and cd' (green dashed box).

$$F' = ac + cd'$$

$$F = (c' + a')(c' + d)$$

$$= c' + a'd$$

Should be able to read this directly from the K-map

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

Diagram illustrating a 4x4 Karnaugh map for Product of Sums Simplification. The map shows the function F with 1s in the first two columns and 0s in the last two columns. The map is partitioned into two groups: $a'd$ (orange dashed box) and c' (green dashed box).

Confirm:

$$F = c' + a'd$$

The Exclusive-OR (XOR) Function

- XOR: $x \oplus y = xy' + x'y$
- $x \oplus 0 = ?$
- $x \oplus 1 = ?$
- $x \oplus x = ?$
- $x \oplus x' = ?$
- Commutative: $x \oplus y = y \oplus x$
- Associative: $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

XOR as ODD Function

- XOR is 1 if there are an **ODD number of 1s** in input variables
- Could be used for **Parity Function**
 - **generation** (add Parity bit)
 - **checking** (even #1s including Parity bit)

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

$a \oplus b \oplus c \oplus d$

XOR with NAND gates

