P2 Digital Electronics

Lecture 2: Low level logic design

Mark Cannon

mark.cannon@eng.ox.ac.uk

Trinity Term 2026

Overview of lectures

1. Logical functions and logic gates
2. **Low level logic design**
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk

Boolean algebra

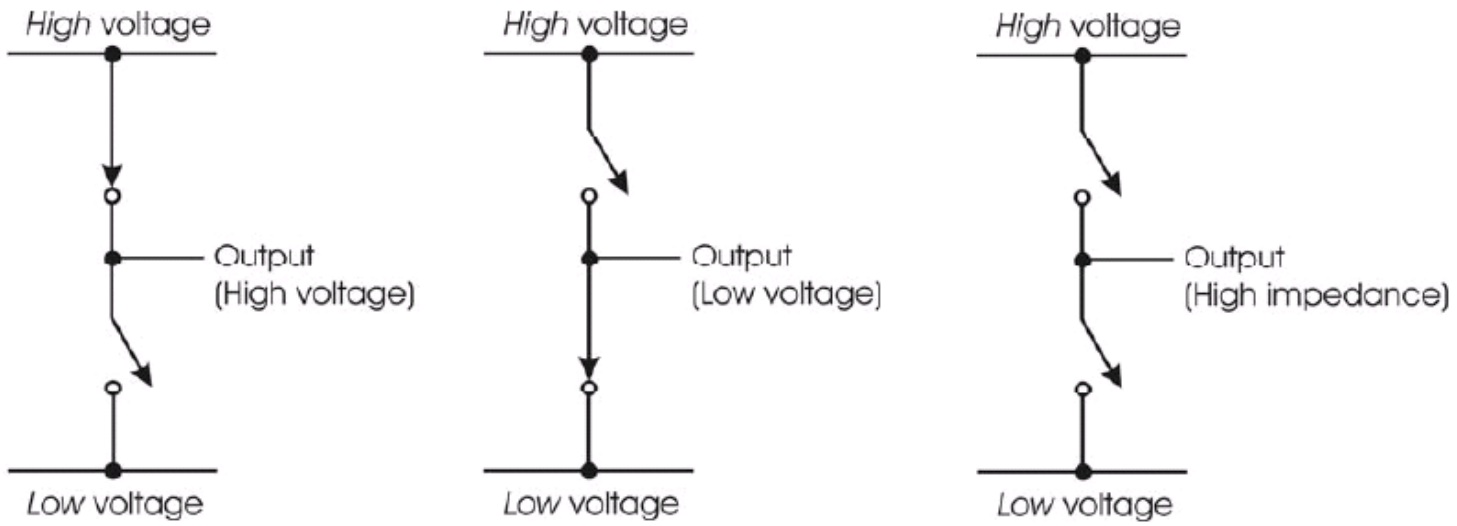
Axioms:

$X + Y = Y + X$		$X.Y = Y.X$		Commutativity (order of args)
$(X + Y) + Z = X + (Y + Z)$		$(X.Y).Z = X.(Y.Z)$		Associativity
$X + 0 = X$	$X + 1 = 1$	$X.0 = 0$	$X.1 = X$	Dominance or identity
$X.(Y + Z) = X.Y + X.Z$				Distributive law 1
$X + Y.Z = (X + Y)(X + Z)$				Distributive law 2

$X.Y = X$ AND Y $\quad$ $X + Y = X$ OR Y $\quad$ $\overline{X} =$ NOT X

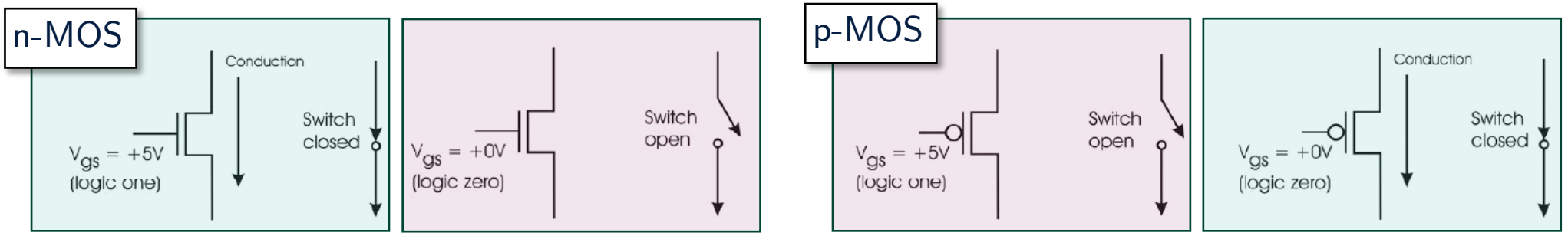
This lecture – how to simplify logical expressions and design simple logic circuits using **Boolean algebra**, **truth tables** and **Karnaugh maps**

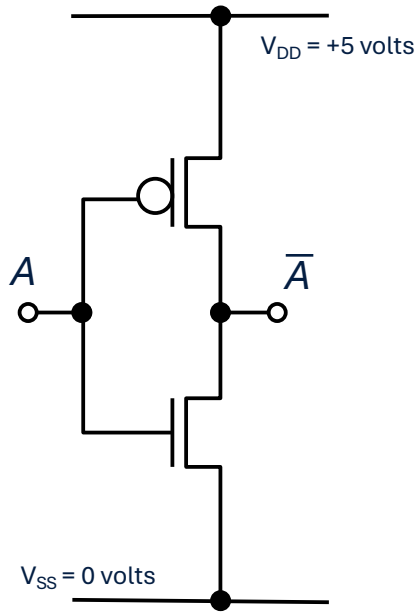
Review: Logic as switches

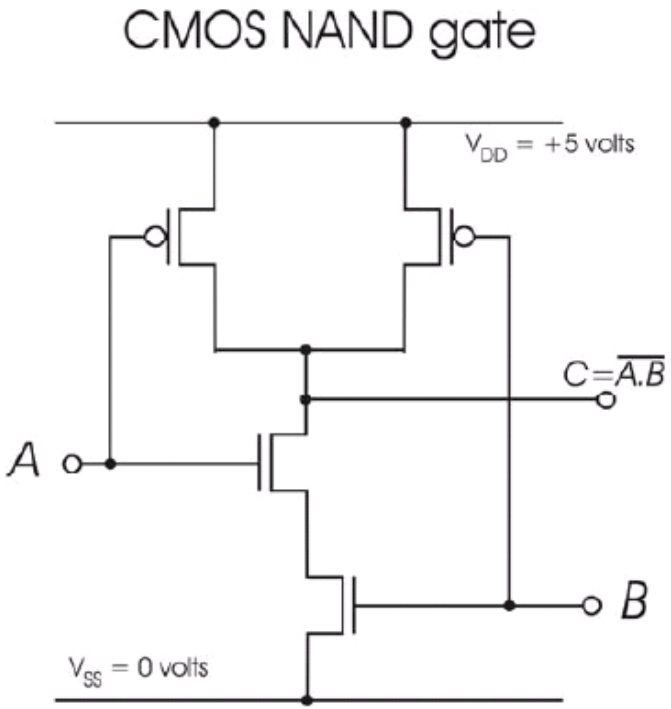


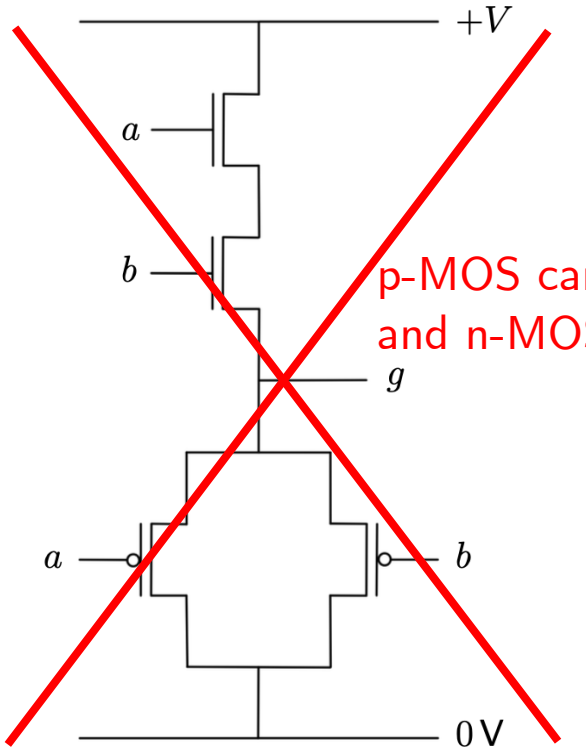
- To implement logic circuits, we use CMOS FET (Complementary Metal-Oxide-Semiconductor, Field Effect Transistors) which are thermally stable and power efficient
- A logic level is represented by two transistors used as switches. The output is controlled by deciding which switch is open or closed

Review: CMOS digital logic implementation









p-MOS can't pull down and n-MOS can't pull up

CMOS AND gate?

Boolean algebra

Truth tables and Karnaugh maps are limited to small numbers of variables (e.g. < 6).
Boolean algebra can handle more variables. We only need 3 theorems:

1. The **Redundancy** theorem: $A + A.B = A$

   [Proof: $A + A.B = A(1 + B) = A.1 = A$]

   $\Rightarrow$ in sum-of-products form, a product that contains all the factors of another product is redundant

   Example: $A.B + A.B.C + A.B.C.D.E.F.G.H.I.J.K = A.B$

   zap

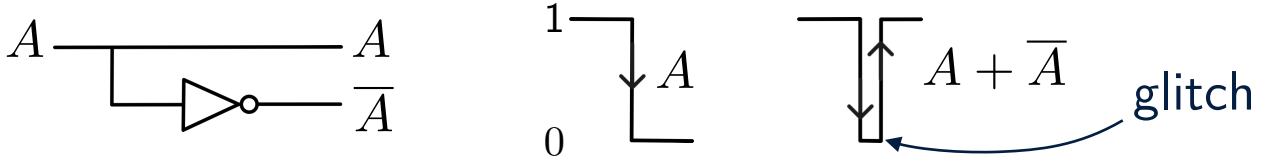
Boolean algebra

2. The **Race-Hazard** theorem: $A.B + \overline{A}.C = A.B + \overline{A}.C + B.C$

Proof: $A.B + \overline{A}.C + (A + \overline{A})B.C = A.B + A.B.C + \overline{A}.C + \overline{A}.C.B$

$= A.B(1 + C) + \overline{A}.C(1 + B) = A.B + \overline{A}.C$

Gets its name because an inverter introduces a delay in practice:



Examples:

(i) $A + \overline{A}.B = A + \overline{A}.B + B = A + (\overline{A} + 1)B = A + B$

(ii) $A.B + \overline{A}.C + \overline{B}.D.E + \overline{C}.D.E = A.B + \overline{A}.C + D.E$

Boolean algebra

3. **De Morgan's** theorem: $\overline{A+B} = \overline{A}.\overline{B}$ NOR $\equiv$ INVERT-AND

$\overline{A.B} = \overline{A} + \overline{B}$ NAND $\equiv$ INVERT-OR

[Proof: compare truth tables for $\overline{A+B}$ and $\overline{A}.\overline{B}$ $\overline{A.B}$ and $\overline{A} + \overline{B}$]

Examples:

(i) $\overline{\overline{A.B}.C} = A.B + \overline{C}$

(ii) $A.B + B.C + A.C = \overline{(\overline{A} + \overline{B}).(\overline{B} + \overline{C}).(\overline{A} + \overline{C})}$

Useful for switching between sum-of-products and product-of-sums forms

Boolean algebra

$X + Y = Y + X$		$X.Y = Y.X$		Commutativity (order of args)
$(X + Y) + Z = X + (Y + Z)$		$(X.Y).Z = X.(Y.Z)$		Associativity
$X + 0 = X$	$X + 1 = 1$	$X.0 = 0$	$X.1 = X$	Dominance or identity
$X.(Y + Z) = X.Y + X.Z$				Distributive law 1
$X + Y.Z = (X + Y)(X + Z)$				Distributive law 2
$\overline{X + Y} = \overline{X}.\overline{Y}$				De Morgan's law 1
$\overline{X.Y} = \overline{X} + \overline{Y}$				De Morgan's law 2
$X + X.Y = X$				Redundancy
$X.Y + \overline{X}.Y = X.Y + \overline{X}.Y + Y.Z$				Race-Hazard

Some Boolean algebra examples

1. Simplify $f = C + \overline{B.C}$

$$f = C + (\overline{B} + \overline{C}) \quad \text{De Morgan}$$
$$= (C + \overline{C}) + \overline{B} \quad \text{Associativity}$$
$$= 1 + \overline{B} = 1 \quad \text{Dominance}$$

2. Simplify $f = \overline{A.B}(\overline{A} + B)(\overline{B} + B)$

$$f = \overline{A.B}(\overline{A} + B) \quad \text{Redundancy}$$
$$= (\overline{A.B}).(\overline{A.\overline{B}}) \quad \text{De Morgan 1}$$
$$= \overline{A.B + A.\overline{B}} \quad \text{De Morgan 2}$$
$$= \overline{A.(B + \overline{B})} = \overline{A} \quad \text{Redundancy}$$

Some Boolean algebra examples

3. Simplify $f = (A + C)(A.D + A.\overline{D}) + A.C + C$

$$
\begin{aligned}
f &= (A + C)(A.D + A.\overline{D}) + C && \text{Redundancy} \\
&= A(A.D + A.\overline{D}) + C(A.D + A.\overline{D}) + C && \text{Distributivity} \\
&= A(A.D + A.\overline{D}) + C && \text{Redundancy} \\
&= A + C && \text{Redundancy}
\end{aligned}
$$

Race-Hazard

Race-Hazard theorem (or Consensus theorem)

$$f = X.Y + \overline{X}.Z = X.Y + \overline{X}.Z + Y.Z$$

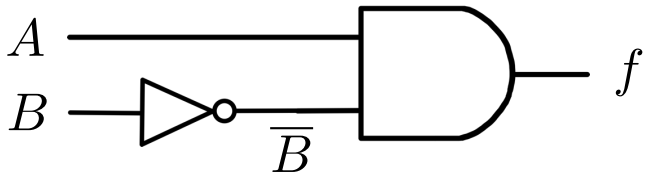
Prove it using a truth table:

X	Y	Z	$X.Y$	$\overline{X}.Z$	$Y.Z$	f
0	0	0				
0	0	1		1		1
0	1	0				
0	1	1		1	1	1
1	0	0				
1	0	1				
1	1	0	1			1
1	1	1	1		1	1

or alternatively using algebra:

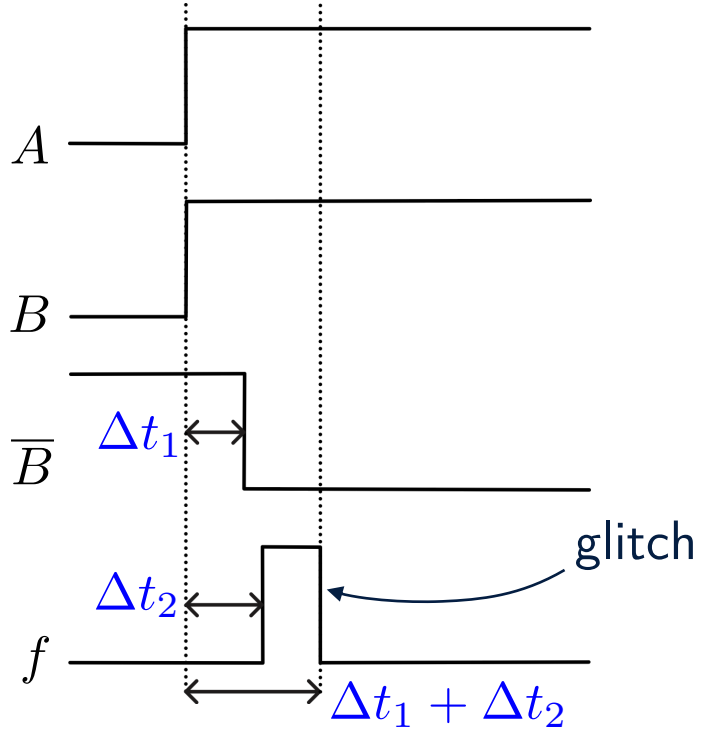
$$\begin{aligned} f &= X.Y + \overline{X}.Z + (X + \overline{X})Y.Z \\ &= X.Y + X.Y.Z + \overline{X}.Z + \overline{X}.Y.Z \\ &= X.Y(1 + Z) + \overline{X}.Z(1 + Y) = X.Y + \overline{X}.Z \end{aligned}$$

Race-Hazard illustration

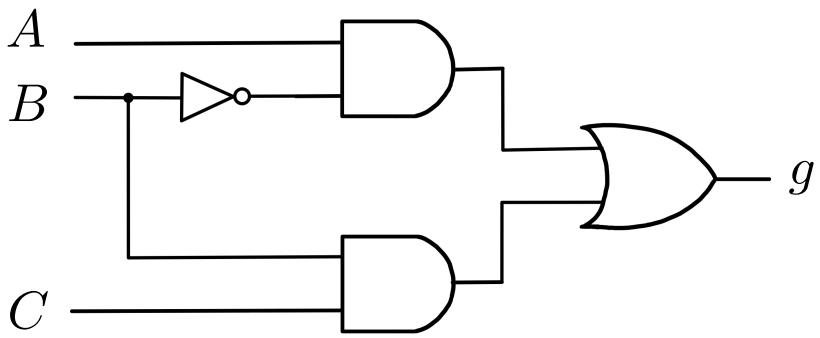


Gate time delay: Δt_1 Δt_2

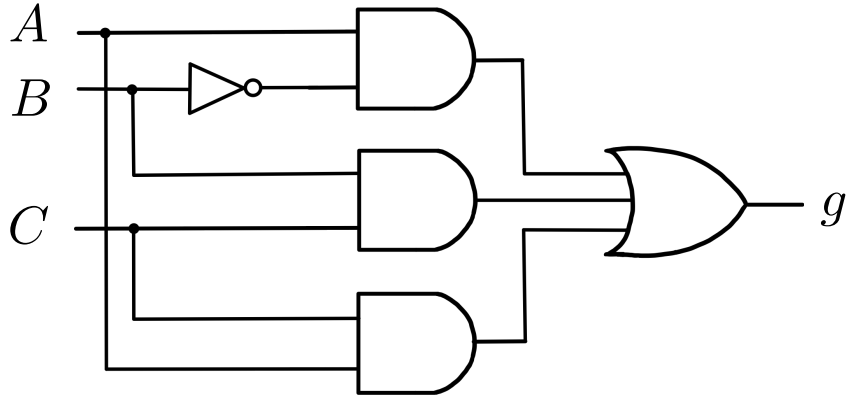
Timing diagram:



Eliminate glitches using the Race-Hazard theorem:



$$g = A.\overline{B} + C.B$$



$$g = A.\overline{B} + C.B + A.C$$

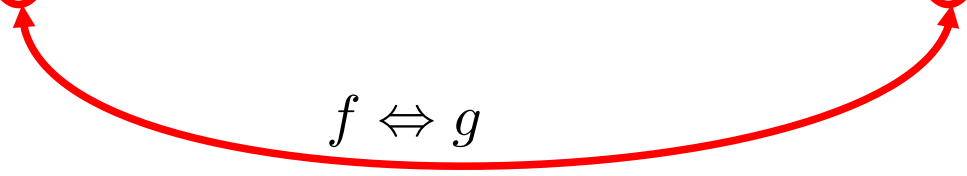
Equivalence of logic functions

If the truth tables of two functions are identical then they are said to be equivalent

e.g. $f = \overline{a} + b$ and $g = \overline{a}.b + a.b + \overline{a}.\overline{b}$

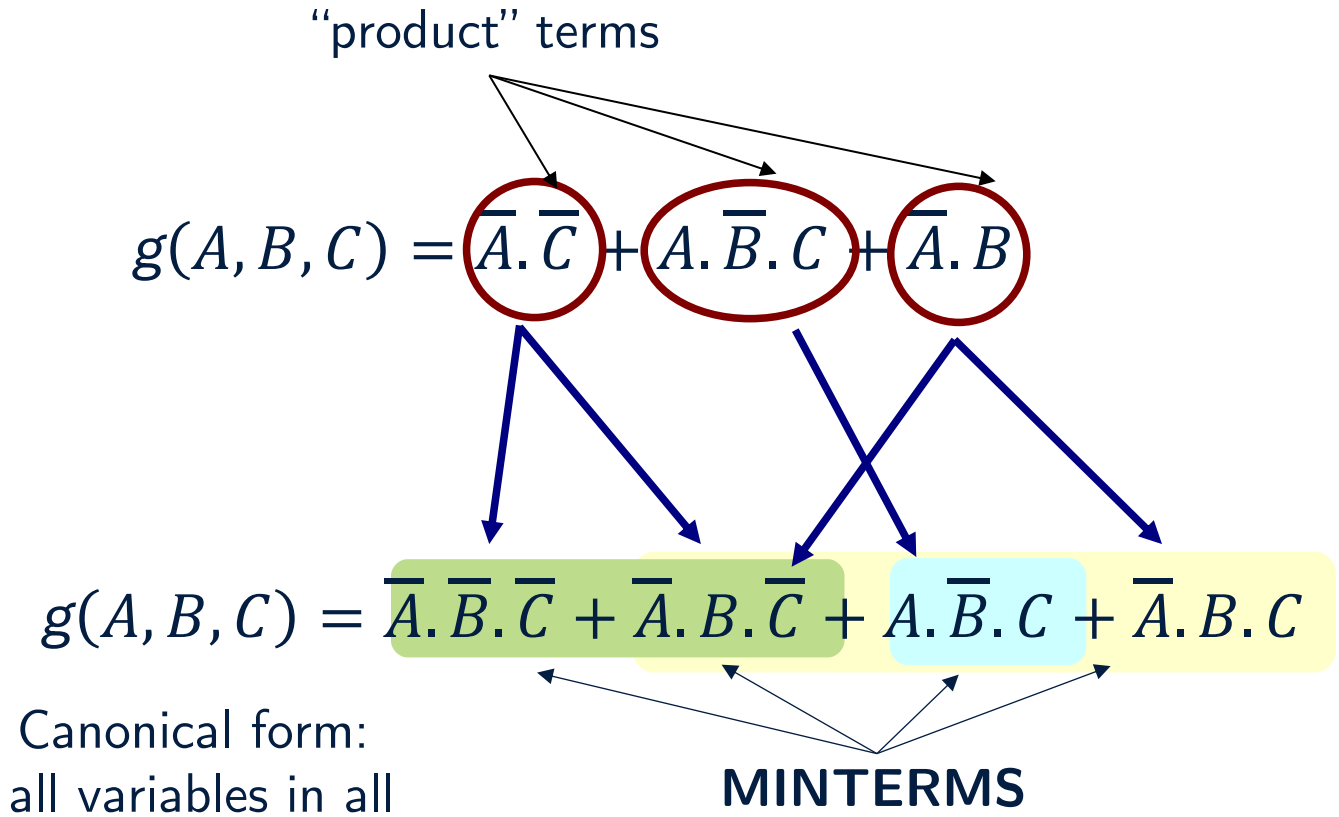
a	b	f
0	0	1
0	1	1
1	0	0
1	1	1

$\overline{a}.b$	$a.b$	$\overline{a}.\overline{b}$	g
0	0	1	1
1	0	0	1
0	0	0	0
0	1	0	1



Standard forms 1: Sum-of-Products (SOP)

OR of ANDs



Canonical form: all variables in all terms

A	B	C	g	
0	0	0	1	$\overline{A}.\overline{B}.\overline{C}$
0	0	1	0	
0	1	0	1	$\overline{A}.B.\overline{C}$
0	1	1	1	$\overline{A}.B.C$
1	0	0	0	
1	0	1	1	$A.\overline{B}.C$
1	1	0	0	
1	1	1	0	

One term for each true (1) output generated by the function

Sum-of-Products example

The lift door closes ($D = 1$) if the timer has timed-out ($T = 0$) and the safety light is not obstructed ($S = 1$) and a call-button on another floor has been pressed ($C = 1$) or a floor-button inside the lift has been pressed ($F = 1$).

Truth table for $D(T, S, F, C)$:

decimal value	T	S	F	C	D
0	0	0	0	0	0
1	0	0	0	1	0
⋮					⋮
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
⋮					⋮
15	1	1	1	1	0

$$\Rightarrow \quad D = \sum m(5, 6, 7) \qquad \leftarrow \text{ MINTERM list}$$

$$\Rightarrow \quad D = \overline{T}S\overline{F}C + \overline{T}SF\overline{C} + \overline{T}SFC$$

Standard forms 2: Product-of-Sums (POS)

AND of ORs

A	B	C	g	
0	0	0	1	
0	0	1	0	← $(A + B + \overline{C})$
0	1	0	1	
0	1	1	1	
1	0	0	0	← $(\overline{A} + B + C)$
1	0	1	1	
1	1	0	0	← $(\overline{A} + \overline{B} + C)$
1	1	1	0	← $(\overline{A} + \overline{B} + \overline{C})$

one product term for each
0 in truth table
(by De Morgan)

"Sums"

$$g(A,B,C) = (A + B + \overline{C}).(\overline{A} + C).(\overline{A} + \overline{B})$$

$$g(A,B,C) = (A + B + \overline{C}).(\overline{A} + B + C).(\overline{A} + \overline{B} + C).(\overline{A} + \overline{B} + \overline{C})$$

Canonical POS form

MAXTERMS

Standard forms 2: Product-of-Sums (POS)

$$g(A,B,C) = (A+B+\overline{C}).(\overline{A}+C).(\overline{A}+\overline{B})$$

Truth table:

A	B	C	$A+B+\overline{C}$	$\overline{A}+C$	$\overline{A}+\overline{B}$	g	$\overline{g}$	
0	0	0	1	1	1	1	0	$\overline{A}.\overline{B}.\overline{C}$
0	0	1	0	1	1	0	1	$\overline{A}.\overline{B}.C$
0	1	0	1	1	1	1	0	$\overline{A}.B.\overline{C}$
0	1	1	1	1	1	1	0	$\overline{A}.B.C$
1	0	0	1	1	1	1	0	$A.\overline{B}.\overline{C}$
1	0	1	1	0	1	0	1	$A.\overline{B}.C$
1	1	0	1	1	0	0	1	$A.B.\overline{C}$
1	1	1	1	0	0	0	1	$A.B.C$

$$\overline{g} = \overline{A}.\overline{B}.C + A.\overline{B}.C + A.B.\overline{C} + A.B.C$$ SOP for $\overline{g}$

$$g = \overline{(\overline{A}.\overline{B}.C)}.\overline{(A.\overline{B}.C)}.\overline{(A.B.\overline{C})}.\overline{(A.B.C)}$$ (De Morgan)

$$= (A+B+\overline{C}).(\overline{A}+B+\overline{C}).(\overline{A}+\overline{B}+C).(\overline{A}+\overline{B}+\overline{C})$$ POS for g

Importance of logic design

- Reducing complexity and minimizing gate count
  (important for design of complex integrated circuits)
- Reducing cost
  (manufacturing cost and power consumption)
- Increasing operating speed
- Avoiding safety problems due to poor design
  (hardware glitches or software bugs)

Poor logic design can be catastrophic

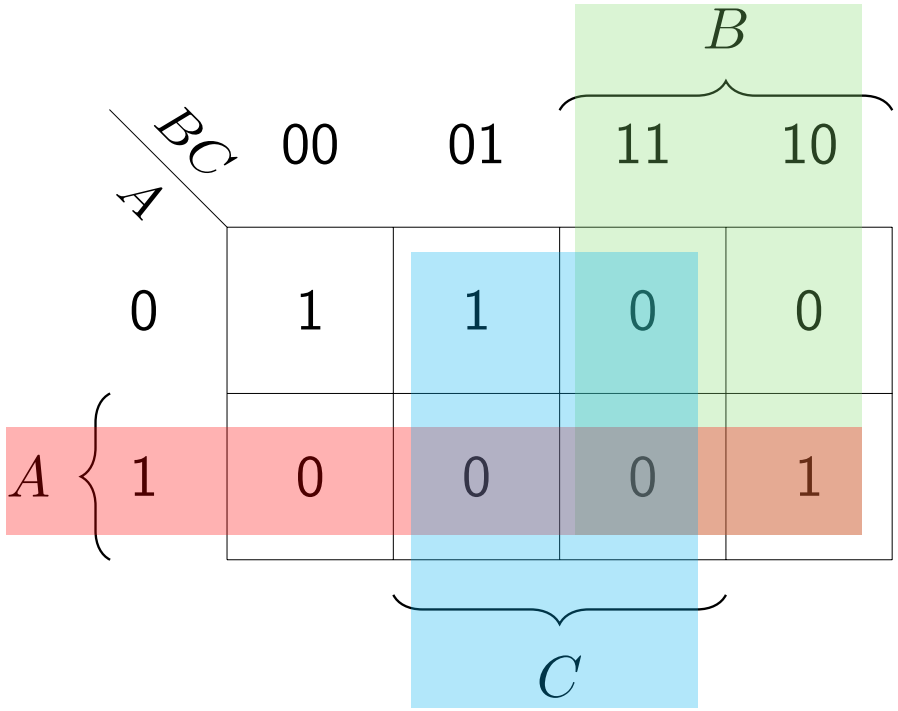
- Therac 25 radiation therapy machine (1985-7)
  Software race hazards played role in accidents
- Northeast blackout: US/Canada (2003). 55 million people without power
  Race hazard crashed a control system for power grid

Karnaugh maps

Maurice Karnaugh

- An alternative truth table layout allowing simplification of Boolean expressions for up to 4 variables in practice
- Terms are grouped in a particular way (using Gray code)

A	B	C	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

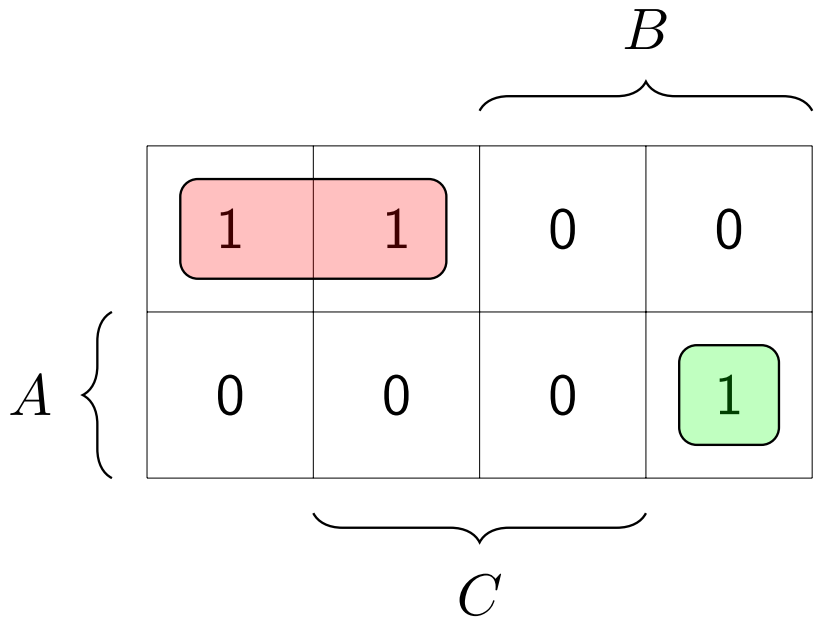


Karnaugh maps: simplifying logic expresssions

$$f = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + A.B.\overline{C}$$

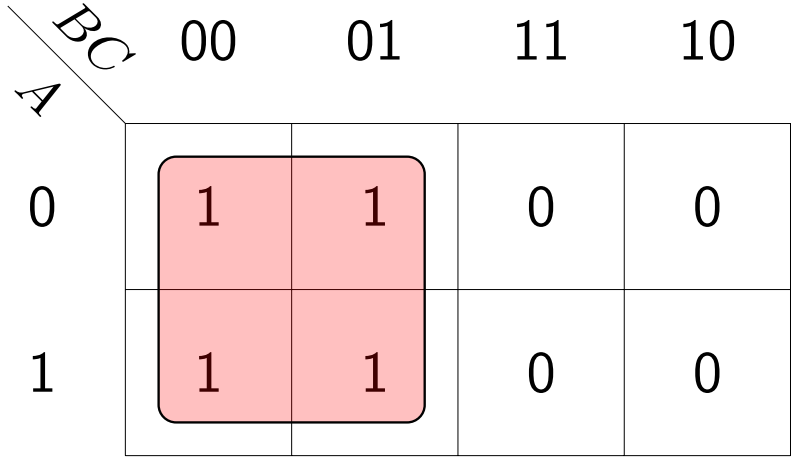
A \ BC	00	01	11	10
0	1	1	0	0
1	0	0	0	1

OR

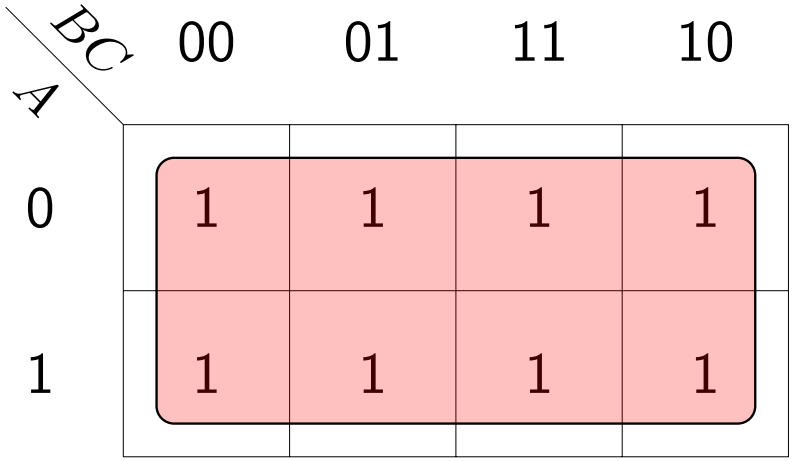


$$\Rightarrow \quad f = \overline{A}.\overline{B} + A.B.\overline{C}$$

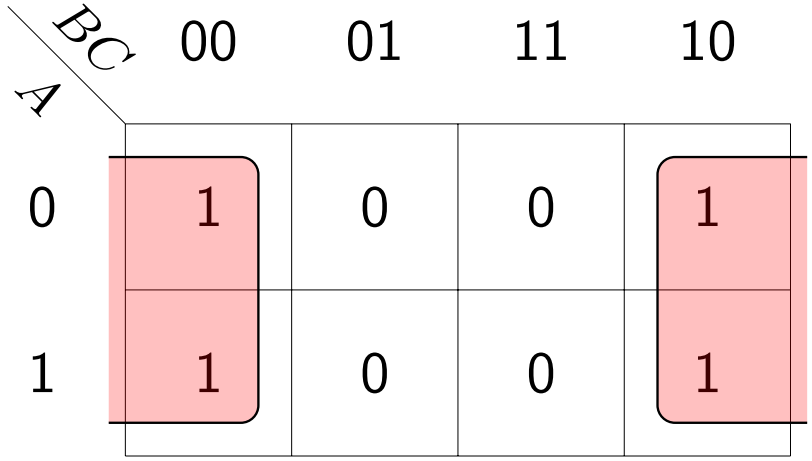
Karnaugh maps: grouping rules



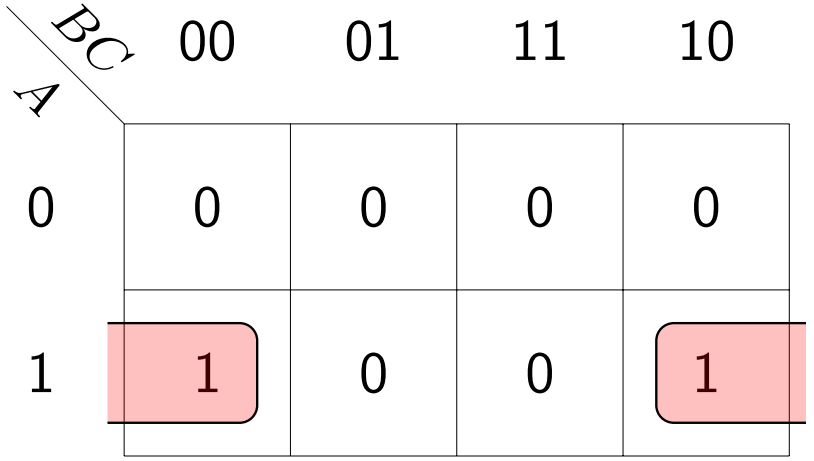
$f = \overline{B}$



$f = 1$



$f = \overline{C}$

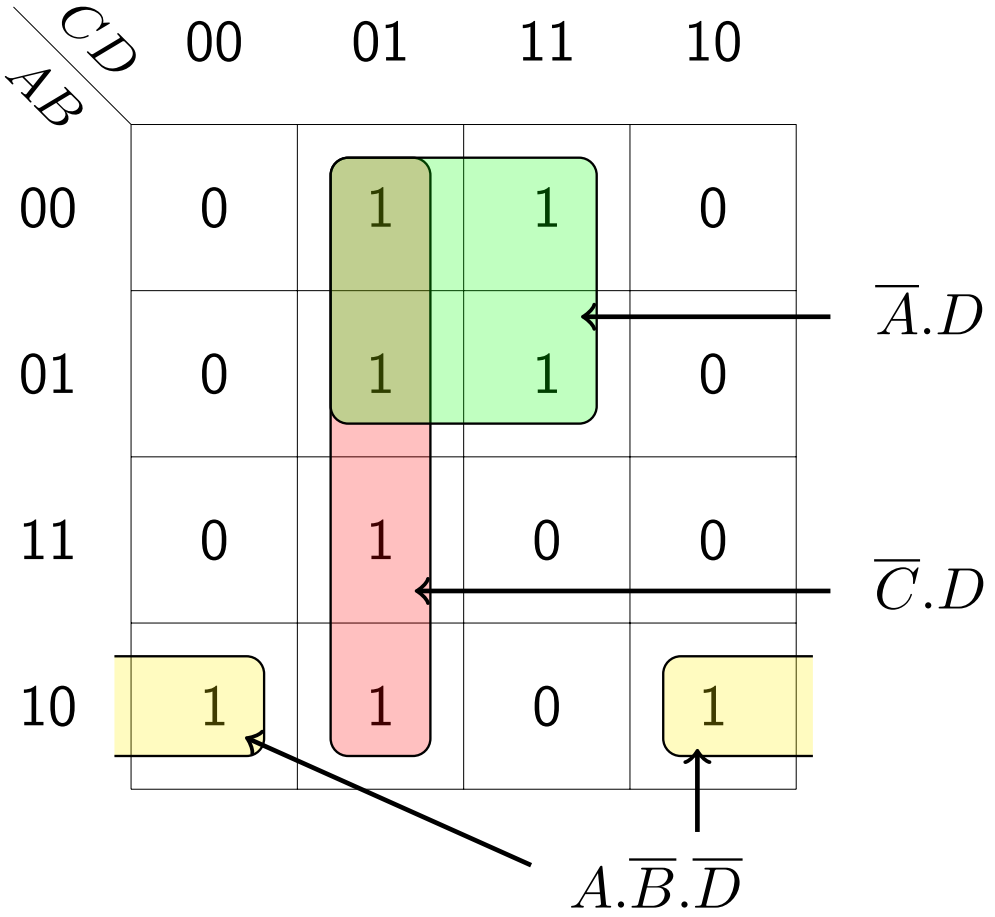


$f = A.\overline{C}$

Karnaugh maps with four variables

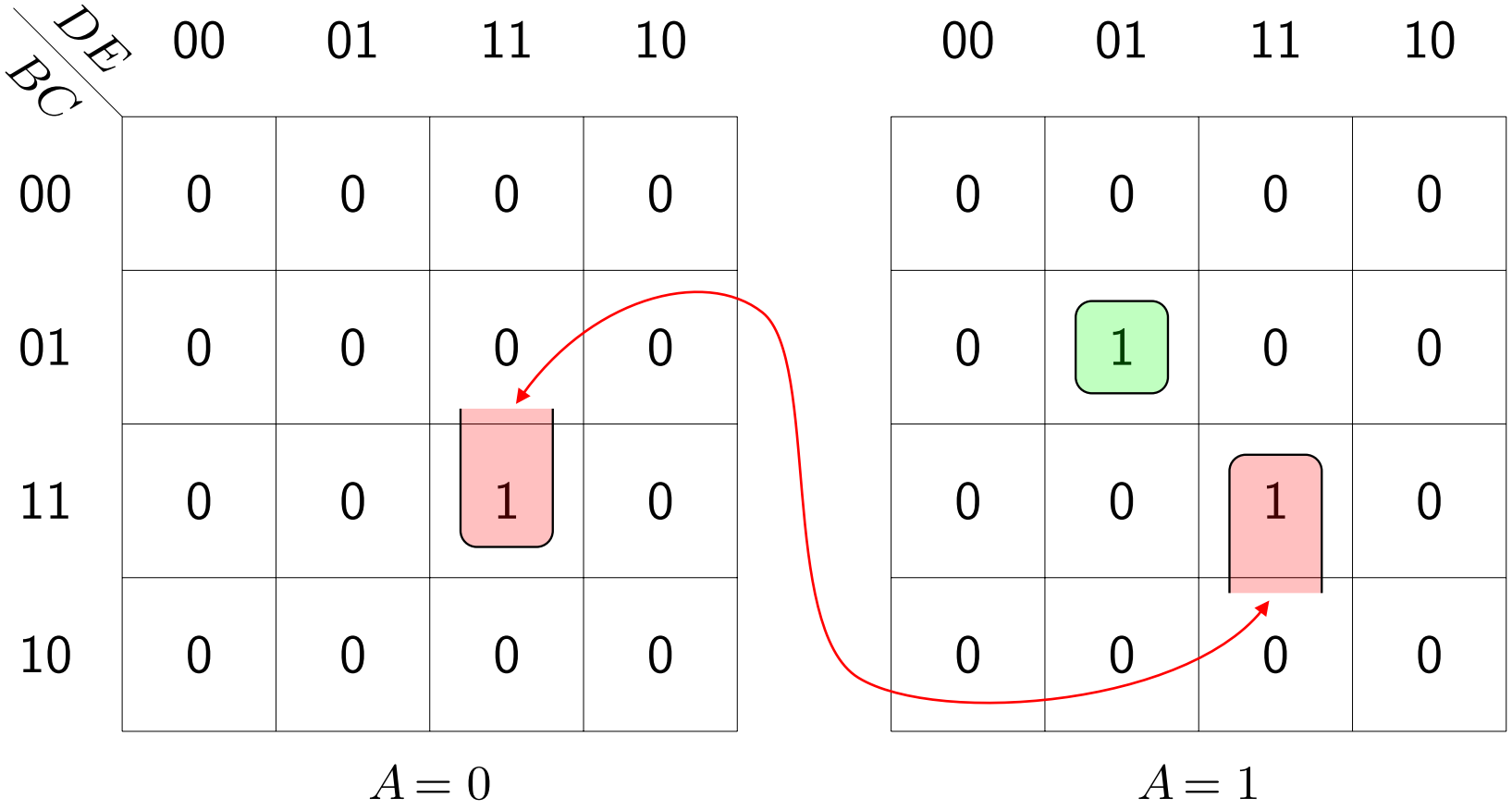
Example:

$$f(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + A.B.\overline{C}.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.\overline{D}$$



$$\Rightarrow \quad f = \overline{A}.D + \overline{C}.D + A.\overline{B}.\overline{D}$$

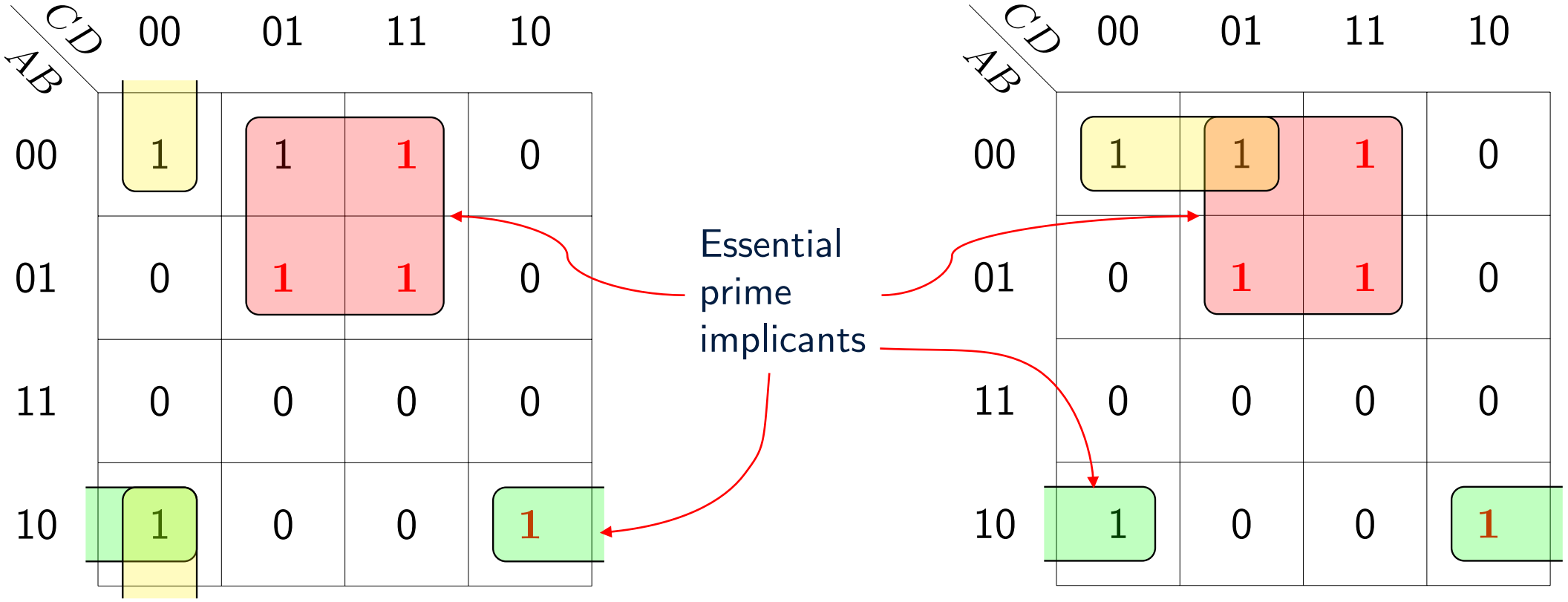
Karnaugh maps with five variables



$$f = A.\overline{B}.C.\overline{D}.E + A.B.C.D.E + \overline{A}.B.C.D.E$$
$$= A.\overline{B}.C.\overline{D}.E + B.C.D.E$$

Prime implicants

Prime implicants are the simplest terms that describe the function uniquely



$f = \overline{A}.D + A.\overline{B}.\overline{D} + \overline{B}.\overline{C}.\overline{D}$ or $f = \overline{A}.D + A.\overline{B}.\overline{D} + \overline{A}.\overline{B}.\overline{C}$

Don't care states

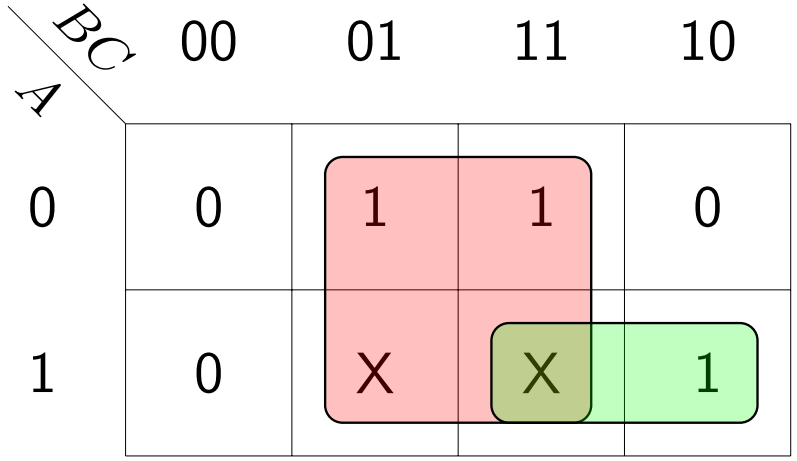
Don't care states: states that the system can never achieve

e.g. in a control system with START and END sensors that can't both be TRUE at the same time

A	B	C	$f(A,B,C)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	X
1	1	0	1
1	1	1	X

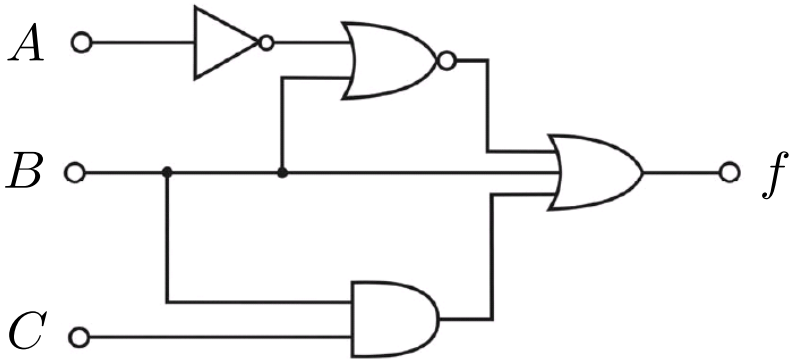
States involving $A = C = 1$ are don't care states

$\Rightarrow$

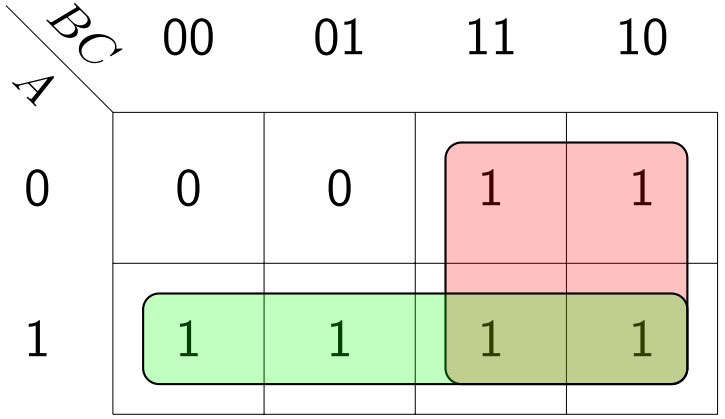


$f = C + A.B$

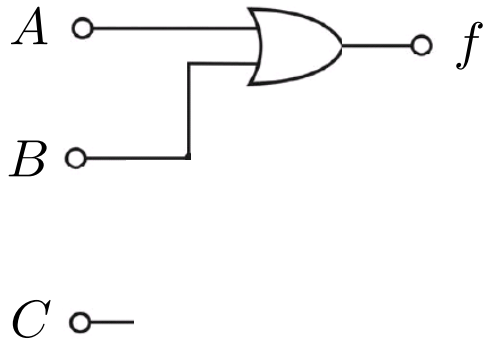
Example: circuit design



A	B	C	$f(A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



$$f = A + B$$



Example: exclusive-OR (XOR)

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

$f = A \oplus B$

A \ B	0	1
0	0	1
1	1	0

The XOR k-map has no group greater than 1 element

XOR therefore has the largest possible number of prime implicants

$f = A \oplus B \oplus C$

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

Overview of lectures

1. Logical functions and logic gates
2. **Low level logic design**
3. Binary number representation
4. Binary arithmetic
5. Integration of digital logic components
6. Memory and sequential circuits
7. Design of sequential logic
8. Data converters: analogue to digital / digital to analogue

Please send feedback, comments and corrections to mark.cannon@eng.ox.ac.uk