EE1D1: Digital Systems A

BSc. EE, year 1, 2025-2026, lecture 4

Logic Minimization

Computer Engineering Lab

Faculty of Electrical Engineering, Mathematics & Computer Science

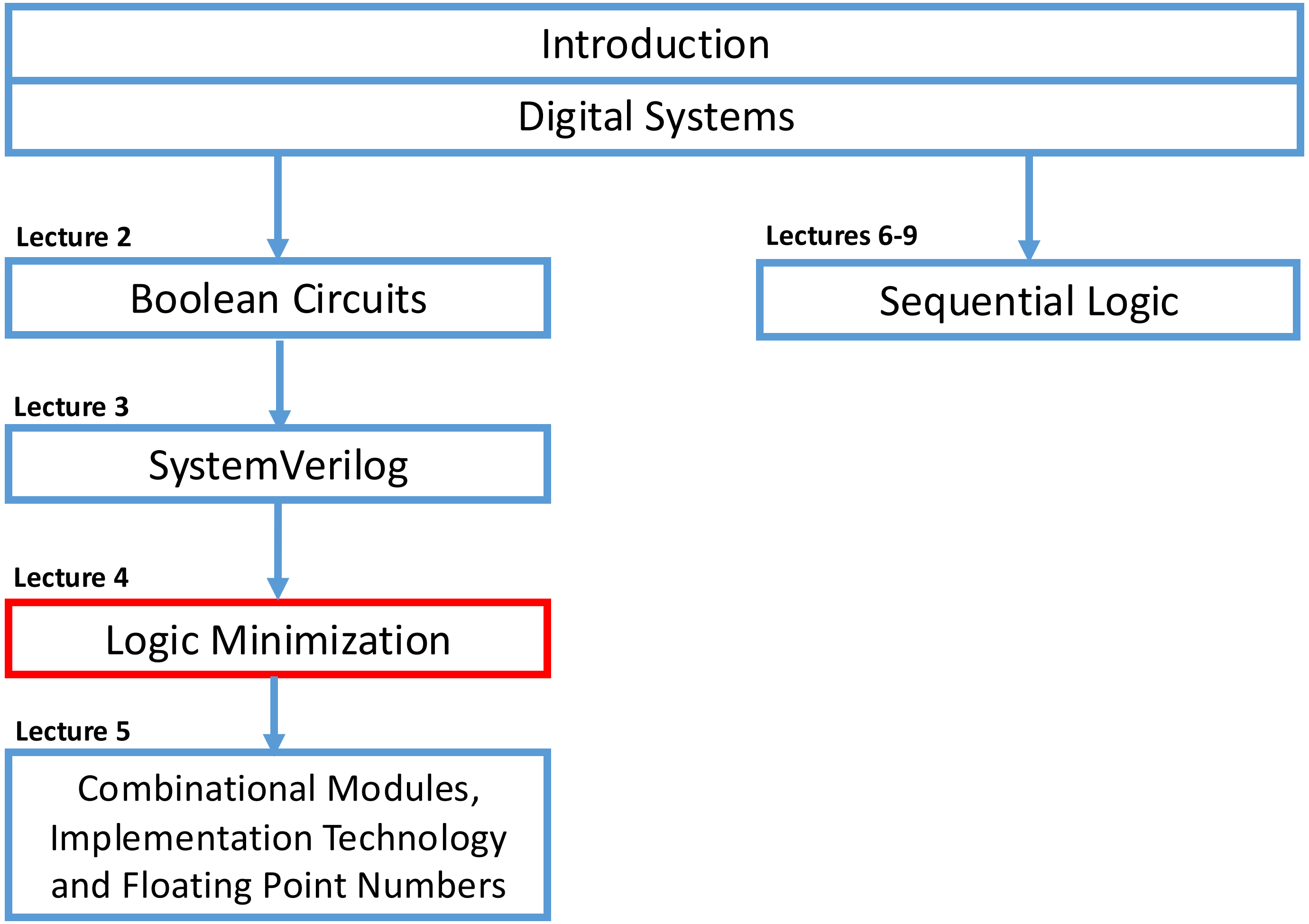
TU Delft

Delft University of Technology

Recap

- System Descriptions
  - Switches, truth table, Boolean algebra, logic gates, timing diagram, HDL
- System Types
  - Combinational circuits
  - Sequential circuits
- Boolean Algebra
  - Logic/Boolean operations
  - Boolean simplification/minimization
  - Prove system equivalence

Recap


Introduction
Digital Systems
Lecture 2
Boolean Circuits
Lectures 6-9
Sequential Logic
Lecture 3
SystemVerilog
Lecture 4
Logic Minimization
Lecture 5
Combinational Modules,
Implementation Technology
and Floating Point Numbers

Recap

Week	Lecture 1 (Mo)	Lecture 2 (Tue)	Assignments	Mock-Up/Exam
1.1	Intro Digital Systems	Boolean Circuits	GP-lec1, GP-lec2	
1.2	SystemVerilog	Logic Minimization	GP-lec3, GP-lec4	
1.3	Combinational Modules, Implementation Technology and Floating Point Numbers		GP-lec5 Course Lab Part 1	
1.4			Course Lab Part 2	Mock Exam (Tuesday) Discuss Mock Exam (Friday)
1.5				Partial Exam 1 (Friday)

Outline

Canonical Expressions Two-Level Networks

- Sum of Products
- Product of Sums

Two-Level Simplification

- Karnaug-maps

Multi-Level networks

- Factorization
- Mapping to NAND-NAND and NOR-NOR networks
- Mapping to AND-OR-inv and OR-AND-inv gates

Timing in Combinatorial Networks

Sections in book: 2.2, 2.3.5, 2.5, 2.7 and 2.9

Learning Objectives

As student you should be able to:

- To use canonical expression (i.e., sum-of products and products-of-sum) to represent logic functions.
- Minimize logic expressions using Karnaugh maps
- Take advantage of don’t care inputs to minimize logic expressions
- Convert two-level and multi-level circuits to NAND or NOR equivalents
- Map expressions to And-Or-Invert circuits and Or-And-Invert
- Analyze the timing behaviour of circuits

Canonical Expressions

Canonical Expressions

- Expressions vs Truth Table
  - Expression ⇒ unique truth table
  - Truth table ⇒ many alternative expressions
- Canonical Form
  - Unique standard form for expressions
  - Truth table ⇒ unique canonical expression
- We look at two canonical forms:
  - Sum-of-Products
  - Product-of-Sums

A	B	C	F	F′
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Sum-of-Products form:

0 1 1   1 0 0   1 0 1   1 1 0   1 1 1

F = A' B C + A B' C′ + A B' C + A B C′ + A B C

F' = A' B' C' + A' B' C + A' B C'

0 0 0   0 0 1   0 1 0

A product term that contains each input signal exactly once is called a minterm

Products-of-Sum form:

0 0 0   0 0 1   0 1 0

F = ((A' B' C')') • ((A' B' C)′) • ((A' B C')')

F = (A + B + C) • (A + B + C′) • (A + B' + C)

A sum term that contains each input signal exactly once is called a maxterm

Canonical expressions – Sum of Products

Truth Table

A	B	C	*Minterms*	F
0	0	0	A'B'C' = m_0	0
0	0	1	A'B'C = m_1	0
0	1	0	A'BC' = m_2	0
0	1	1	A'BC = m_3	1
1	0	0	AB'C' = m_4	1
1	0	1	AB'C = m_5	1
1	1	0	ABC' = m_6	1
1	1	1	ABC = m_7	1

Canonical form

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

= $m_3 + m_4 + m_5 + m_6 + m_7$

= Σ m(3,4,5,6,7)

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

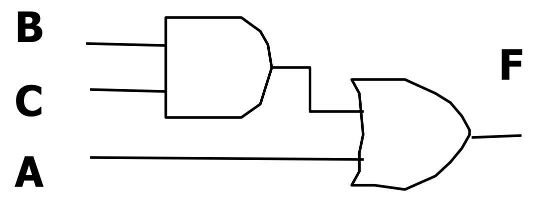
= $m_0 + m_1 + m_2$

= Σ m(0,1,2)

Canonical form to minimal form sum of products

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

Circuit



Canonical Expressions – Product of Sums

Truth Table

A	B	C	*Maxterms*		F	F'
0	0	0	A+B+C	= M_0	0	1
0	0	1	A+B+C'	= M_1	0	1
0	1	0	A+B'+C	= M_2	0	1
0	1	1	A+B'+C'	= M_3	1	0
1	0	0	A'+B+C	= M_4	1	0
1	0	1	A'+B+C'	= M_5	1	0
1	1	0	A'+B'+C	= M_6	1	0
1	1	1	A'+B'+C'	= M_7	1	0

Canonical form

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

$= \prod M(0,1,2) = M_0 M_1 M_2$

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

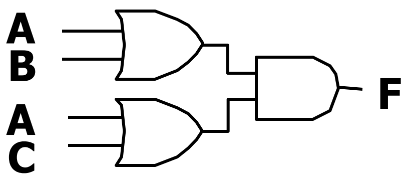
$= \prod M(3,4,5,6,7) = M_3 M_4 M_5 M_6 M_7$

Canonical form to minimal form Product of Sums

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

Note: Y = (A + B)
(Y+C)(Y+C') = YY + YC' + CY + CC'
= Y + YC' + YC = Y

Circuit



Canonical Expressions

- Two-level Canonical Forms
  - (Sum of Products)′ ⇒ Product of Sums:

F = m3 + m4 + m5 + m6 + m7
= A' B C + A B' C' + A B' C + A B C' + A B C

F′ = (A' B C + A B' C' + A B' C + A B C' + A B C)′
= (A' B C)′ (A B' C′)′ (A B' C)′ (A B C′) (A B C)′
= (A + B' + C') (A' + B + C) (A' + B + C') (A' + B' + C) (A' + B' + C')
= $M_3 M_4 M_5 M_6 M_7$

  - (Product of Sums)′ ⇒ Sum of Products:

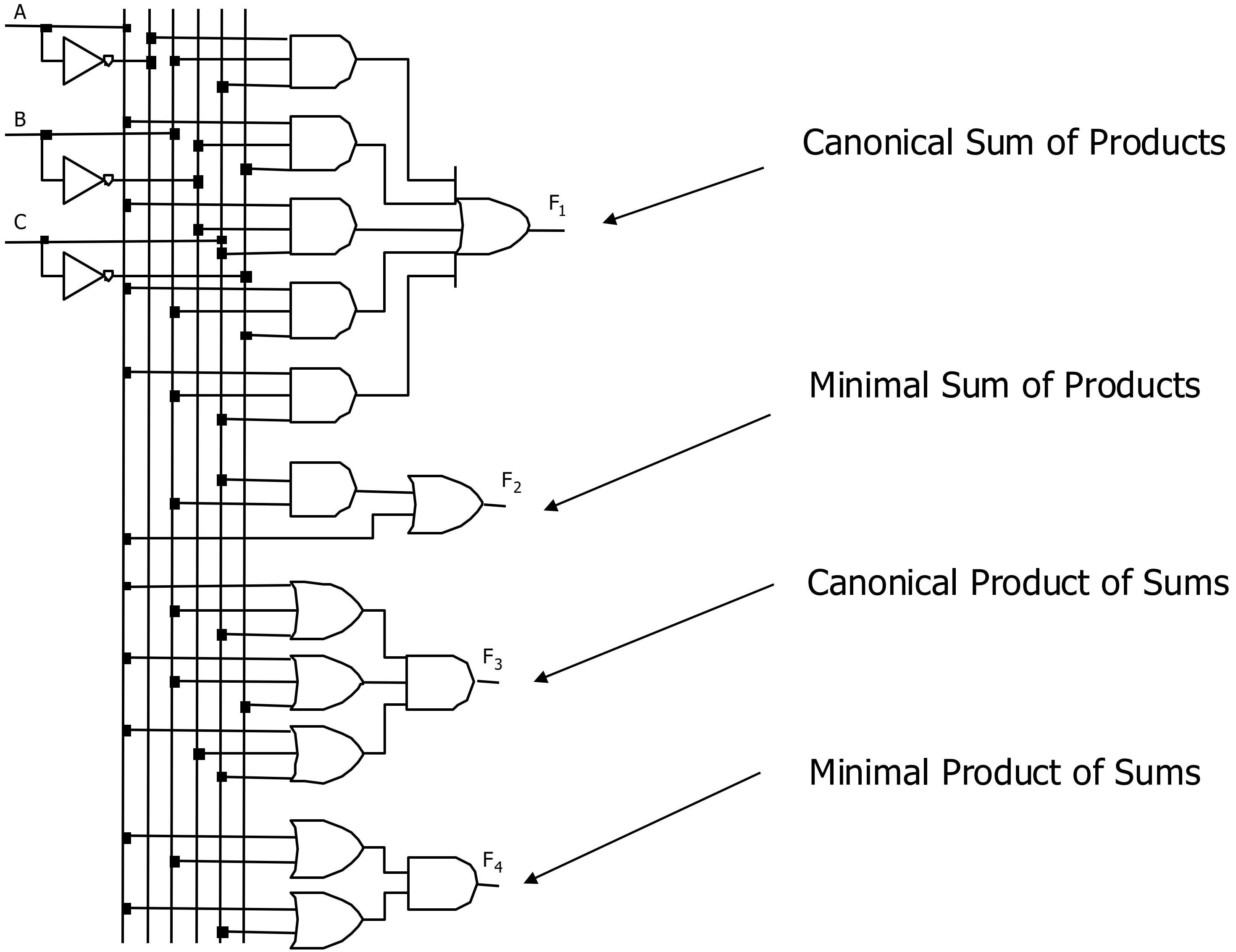
F = $M_0 M_1 M_2$
= (A + B + C) (A + B + C') (A + B' + C)

F′ = {(A + B + C) (A + B + C') (A + B' + C)}′
= (A + B + C)′ + (A + B + C')′ + (A + B' + C)′
= A' B' C' + A' B' C + A' B C′
= m0 + m1 + m2

A	B	C	F	F′
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

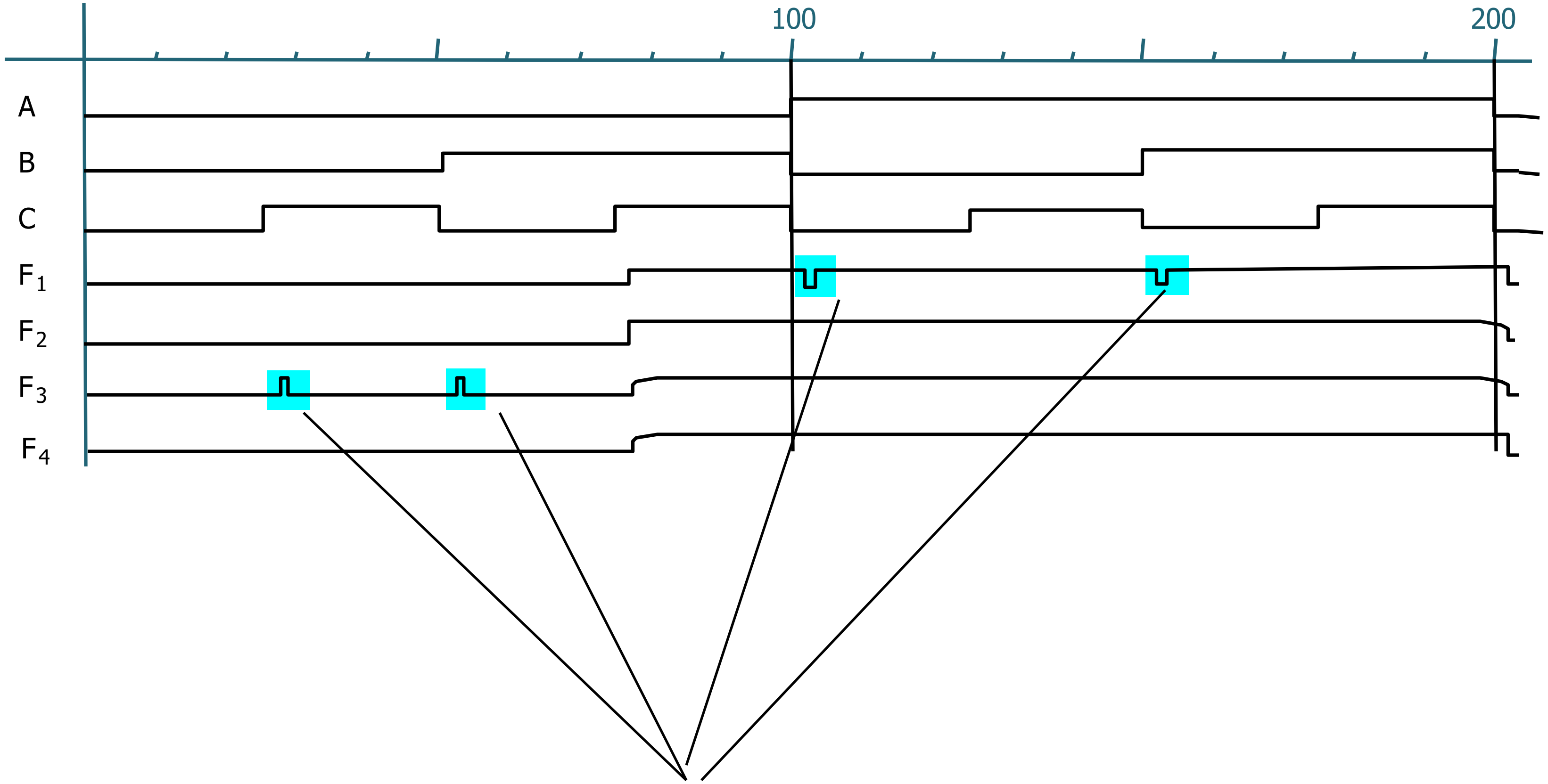
Canonical Expressions

- Alternative implementations of F



Canonical Expressions

- Comparison timing behaviour



Apart from *timing glitches* the behaviour is the same for different implementations

Canonical Expressions

- Incompletely specified functions

n-input function ⇒ 2^n possible input combinations
not all possibilities are always relevant

Example: Binary-Coded-Decimal-Digit-Increment-by-1

	inputs					outputs			
Digit	A	B	C	D	Digit	W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	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

input combinations for Z = 0:
Off-set for Z

input combinations for Z = 1:
On-set for Z

input combinations for Z = X:
Don't care (DC) set for Z

X = don't care (value is not relevant)

$$\mathbf{Z = m_0 + m_2 + \ldots + m_8 + d_{10} + d_{11} + \ldots + d_{15} = M_1 M_3 \ldots M_9 D_{10} D_{11} \ldots D_{15}}$$

Two-Level Simplification

Two-Level Simplification

- Algebraic simplification
  - No fixed procedure
  - How do you know a minimal form has been reached?
- Simplification of two-level networks using Karnaugh-maps (see next slides):
  - Systematic method
  - Always minimal form
  - Limited to max. 4 or 5 inputs
- Computer-Aided Design (CAD) Tools
  - Optimal simplifications require a lot of computation power, especially for functions with many inputs (>10)
  - Therefore sub-optimal solutions are calculated
    - less computation time needed
    - solutions are not optimal but (usually) acceptable
- Manual Simplification Still Useful Though
  - For small circuits: manual simplification gives more insight
  - Insight in CAD tools (espresso, Quartus, ISE)
  - Possibility to check CAD results (for small circuits)
  - No CAD tools available during the exam ...

Two-Level Simplification – Simplification Principle

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

B values are different within on-set rows

A values are not different within on-set rows

B is eliminated, A remains

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

A	B	G
0	0	1
0	1	0
1	0	1
1	1	0

B values are not different within on-set rows

A values are different within on-set rows

A is eliminated, B remains

G = A' B' + A B' = (A' + A) B' = B'

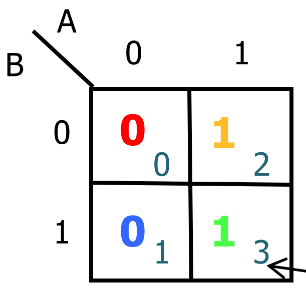
The essence of simplification!

- Find two input combinations of the ON-set where only one variable has a different value.
- This variable apparently doesn't make a difference and can be eliminated.

Two-level simplification – Karnaugh Maps (K-maps)

- K-Maps
  - Alternative method to represent truth tables, such that between 2 “neighbours” exactly one input variable changes its value.
  - If neighbours have the same output value: the input variable that changes can be eliminated

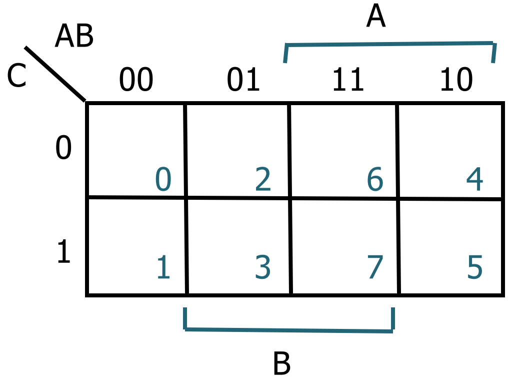
2-variable K-map



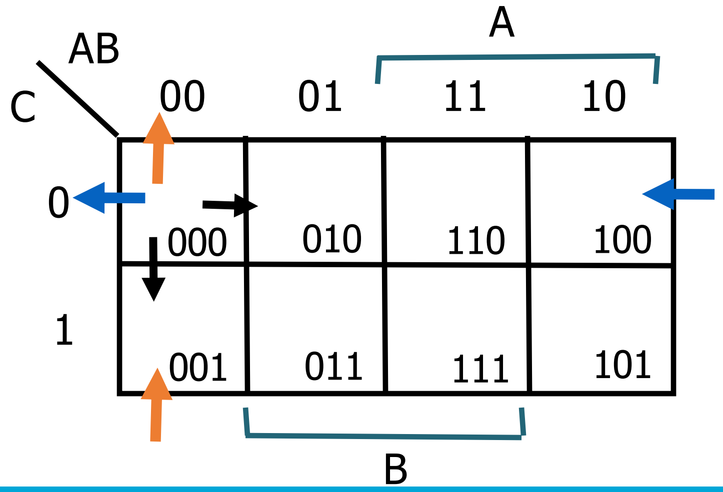
m	A	B	F
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

number in cell denotes index minterm, e.g m_3

3-variable K-map



m	A	B	C	F
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

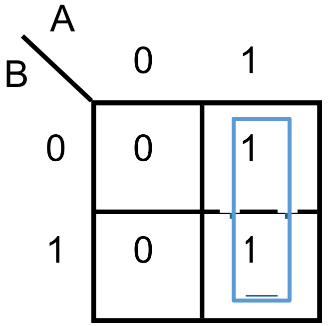


Neighbours in K-maps:

- exactly one variable changes between neighbours
- first-last column are also neighbour
- top-bottom row are also neighbour

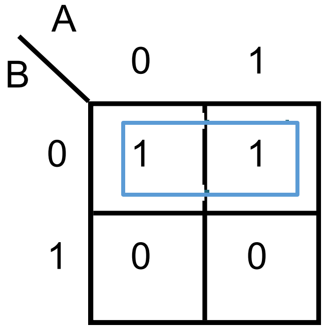
Two-Level Simplification – Karnaugh Maps (K-maps)

- Examples



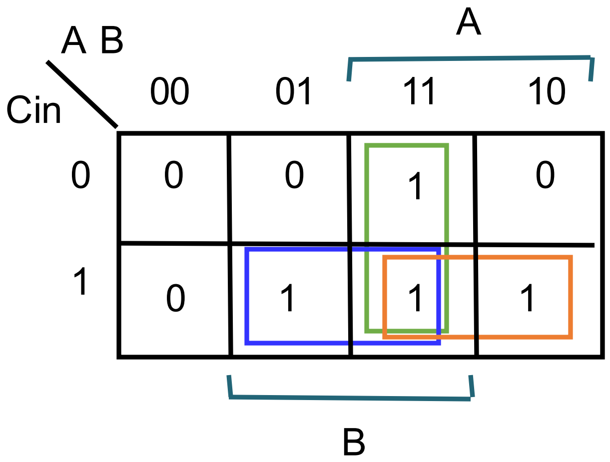
F = A B’ + A B

F = A



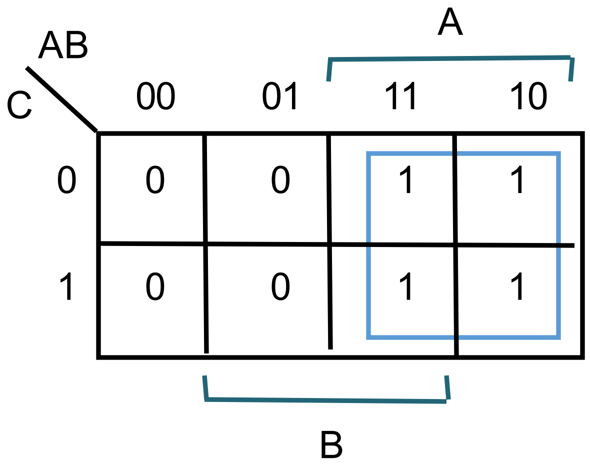
G = A’ B’ + A B’

G = B'



Cout = A’ B Cin + A B’ Cin + A B Cin’ + A B Cin

Cout = A B + B Cin + A Cin



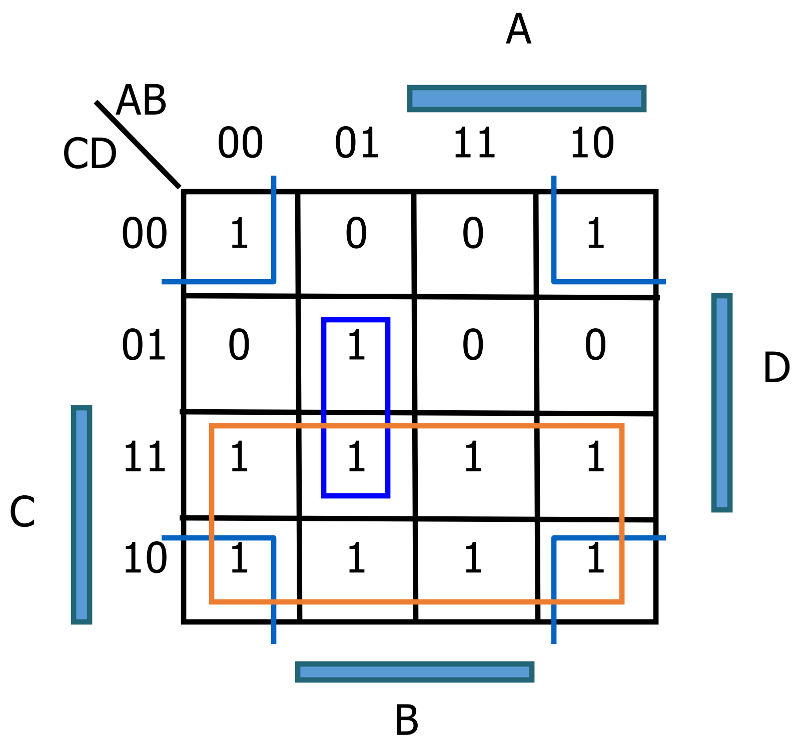
F(A,B,C) = A

- Definitions/Observations
  - Product term (implicant) ⬄ Rectangle with 1’s
  - Rectangle with 1’s larger ⬄ Corresponding product term smaller !
  - Only rectangles corresponding to products terms (with 1, 2, 4, 8 … (2^n) 1’s) can be used

Two-Level Simplification – Karnaugh Maps (K-maps)

- Example derivation minimal sum for 4 variables

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

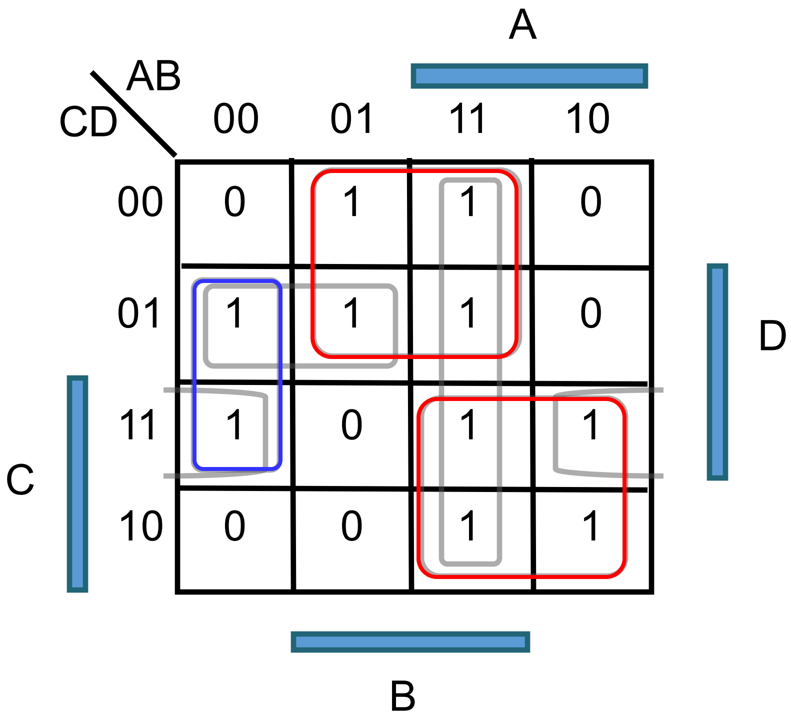


F = C + A' B D + B' D'

- Minimal sum of products:
  - Find the lowest number of rectangles as large as possible, that covers the ON-set
  - Because: less rectangles = less product terms
  - larger rectangle = less variables in product term

Two-Level Simplification – Recipe Minimal Sum of Products

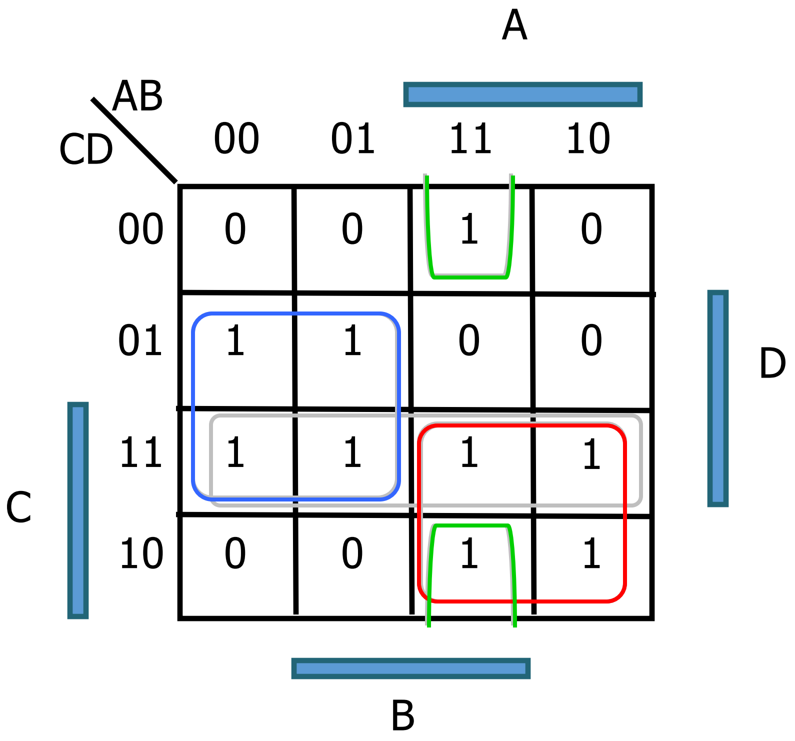
- Recipe:
  1. Find all maximally large rectangles covering 1's: **prime implicants**.
  2. Find all prime implicants that are the only ones that cover a certain 1: essential prime implicants. The essential prime implicants are for sure a product term part of the minimal expression.
  3. Cover the remaining 1's using as less as possible non-essential prime implicants. These prime implicants represent the other product terms part of the minimal expression.



F = BC' + AC + A'B'D

Two-Level Simplification – Question 1

What is a minimal sum of products for the K-map below?



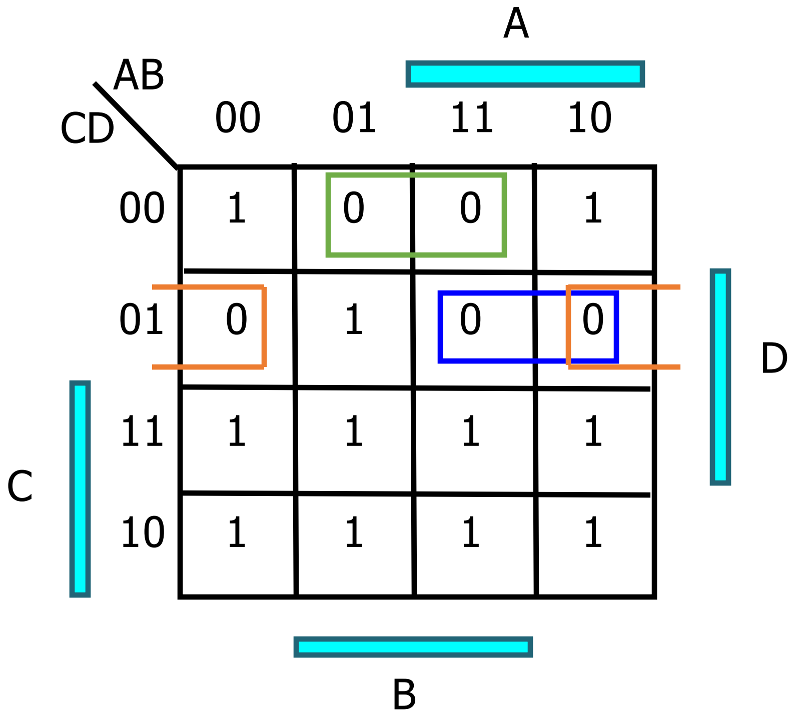
a. A′ D + A C + C D + A B D′
b. A C + C D + A B C′ D′
c. A C + A′ D + A B C′ D′
d. There is no correct answer listed above.

A C + A′ D + A B D′

Hence answer d

Two-Level Simplification

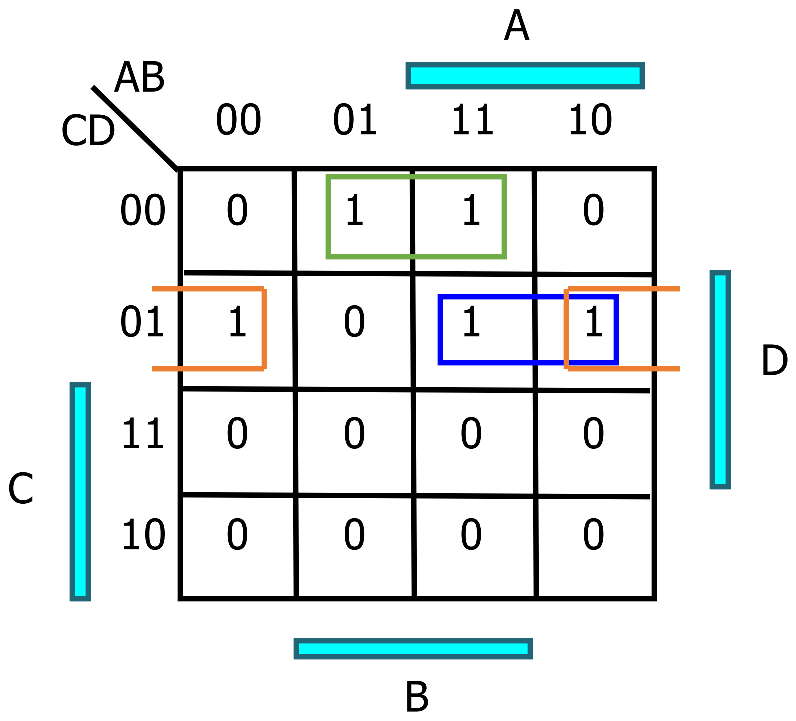
- Dual method: minimal *product of sums* vs *sum of products*



B=1, C=0, D=0

A=1, C=0, D=1

B=0, C=0, D=1



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

$F' = B\,C'\,D' + A\,C'\,D + B'\,C'\,D$

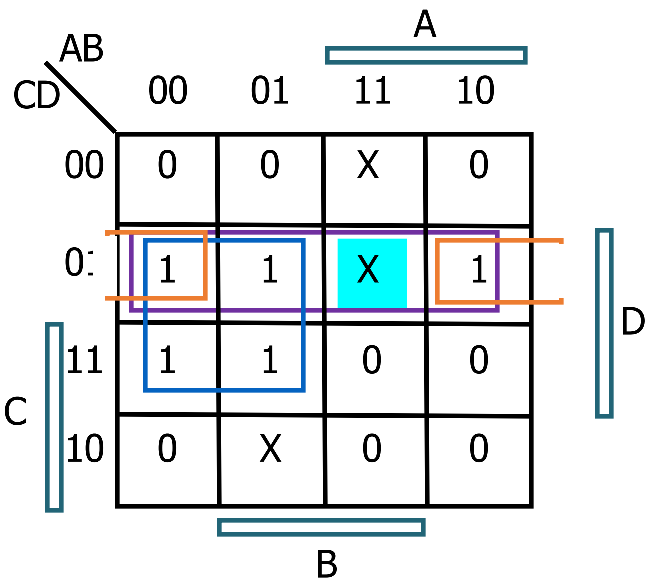
This method can be explained as follows:

$(F')' = (B\,C'\,D' + A\,C'\,D + B'\,C'\,D)'$

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

Two-Level Simplification

- Don’t Cares
  - Don’t cares should be used as 1 or 0 to obtain better results



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

via K-map:

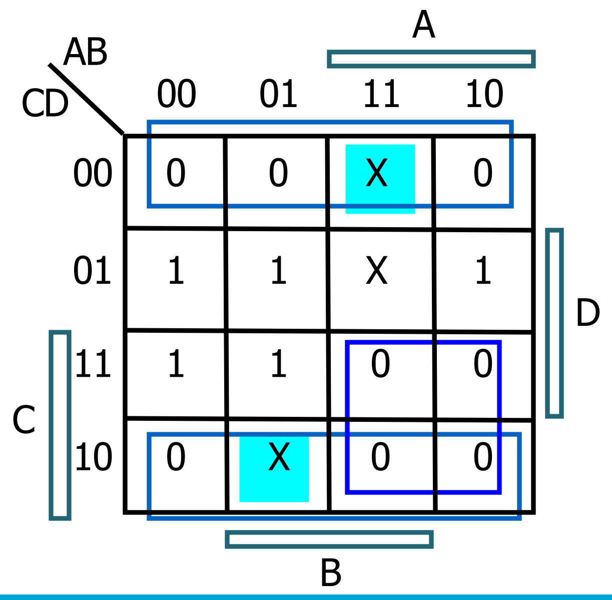
F = A′ D + B' C' D without don't cares

F = A' D + C′ D using don't cares

Via dual method:

F = D (A' + C′)

Hence even less terms



Two-Level Simplification – Summary Two-Level Networks

- Primitives:
  - INVERTER, AND, OR
- Canonical forms:
  - Sum of Products (minterms), Product of Sums (maxterms) , incl. don't cares
- Logical simplification:
  - 2-level realisation with minimum number of gates and/or gate inputs
  - K-map method (incl. dual method)

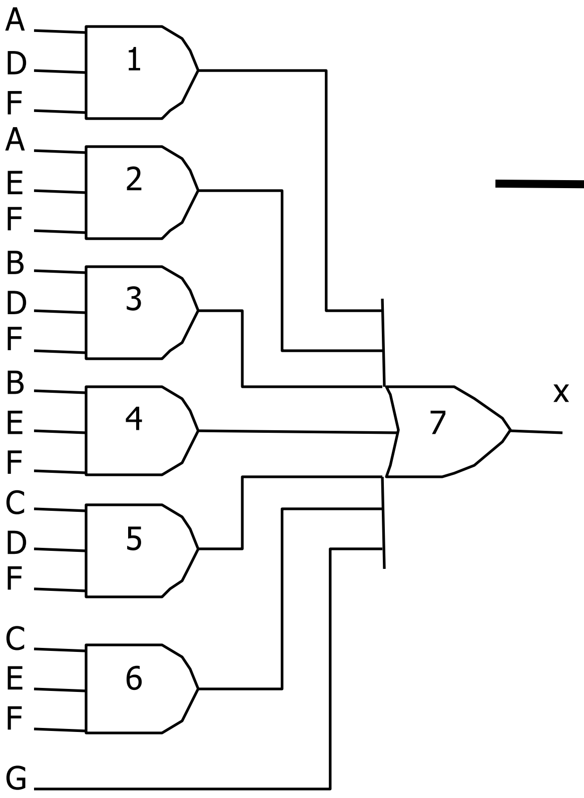
Multi-Level Networks

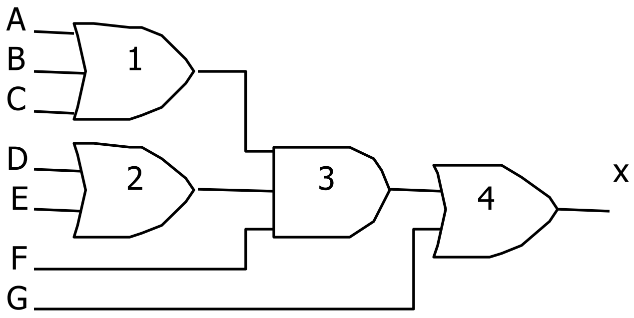
Multi-Level Networks – Advantages

Consider following sum of products form (already 2-level reduced!):

X = A D F + A E F + B D F + B E F + C D F + C E F + G

- 6 x 3-input AND gates + 1 x 7-input OR gate (often not available)
- 25 connections (19 literals plus 6 internal connections)





Conversion to <u>factorised</u> form gives more levels, but also a smaller circuit:
X = (A + B + C) (D + E) F + G

- 1 x 3-input OR, 2 x 2-input OR, 1 x 3-input AND
- 10 connections (7 literals plus 3 intern)

So factorization may provide better result.
However, no systematic way to do it

Multi-Level Networks - Conversion to NAND/NAND and NOR/NOR

- Conversion
  - Initial network often expressed in ANDs and ORs (canonical form)
  - Preferred gates for implementation are however NANDs en NORs (better technological properties)
  - Hence we will rewrite AND/OR expressions to expressions that can be used for implementation with NANDs and/or NORs

Multi-Level Networks - Conversion to NAND/NAND and NOR/NOR

De Morgan's law:

$(A + B)' = A' \bullet B'$ $\Rightarrow$ $A + B = (A' \bullet B')'$

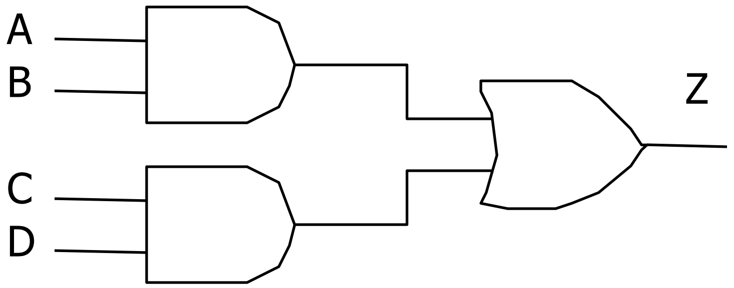
$(A \bullet B)' = A' + B'$ $\Rightarrow$ $A \bullet B = (A' + B')'$

Hence:

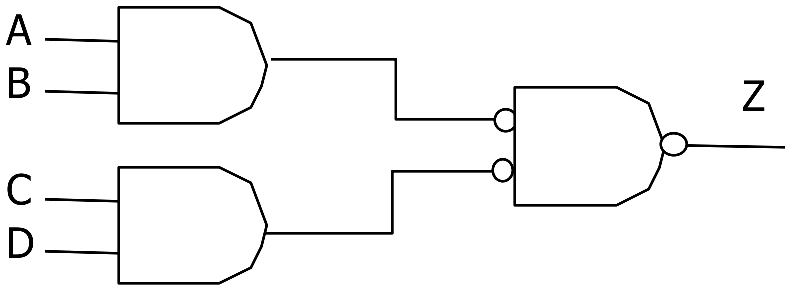
- NOR is equivalent to an AND with inverted inputs
- OR is equivalent to a NAND with inverted inputs
- NAND is equivalent to an OR with inverted inputs
- AND is equivalent to a NOR with inverted inputs

Multi-Level Networks

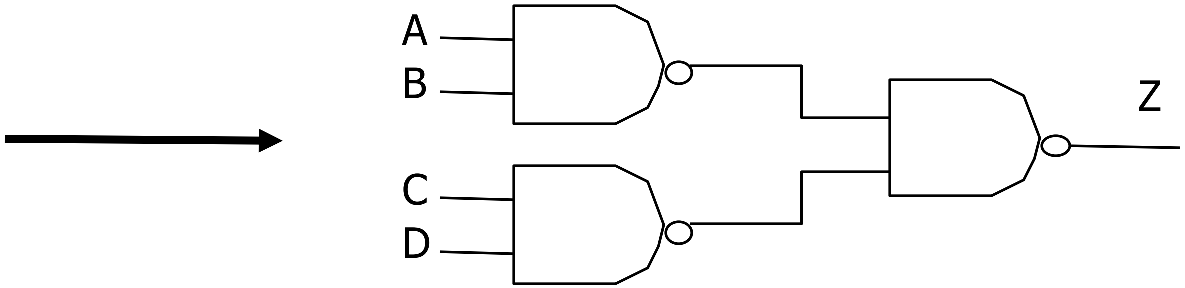
- Application: Conversion AND/OR network to NAND/NAND network



AB+CD



(AB+CD)″
((AB)′ • (CD)′)′



In a similar way an OR/AND network can be converted to a NOR/NOR network

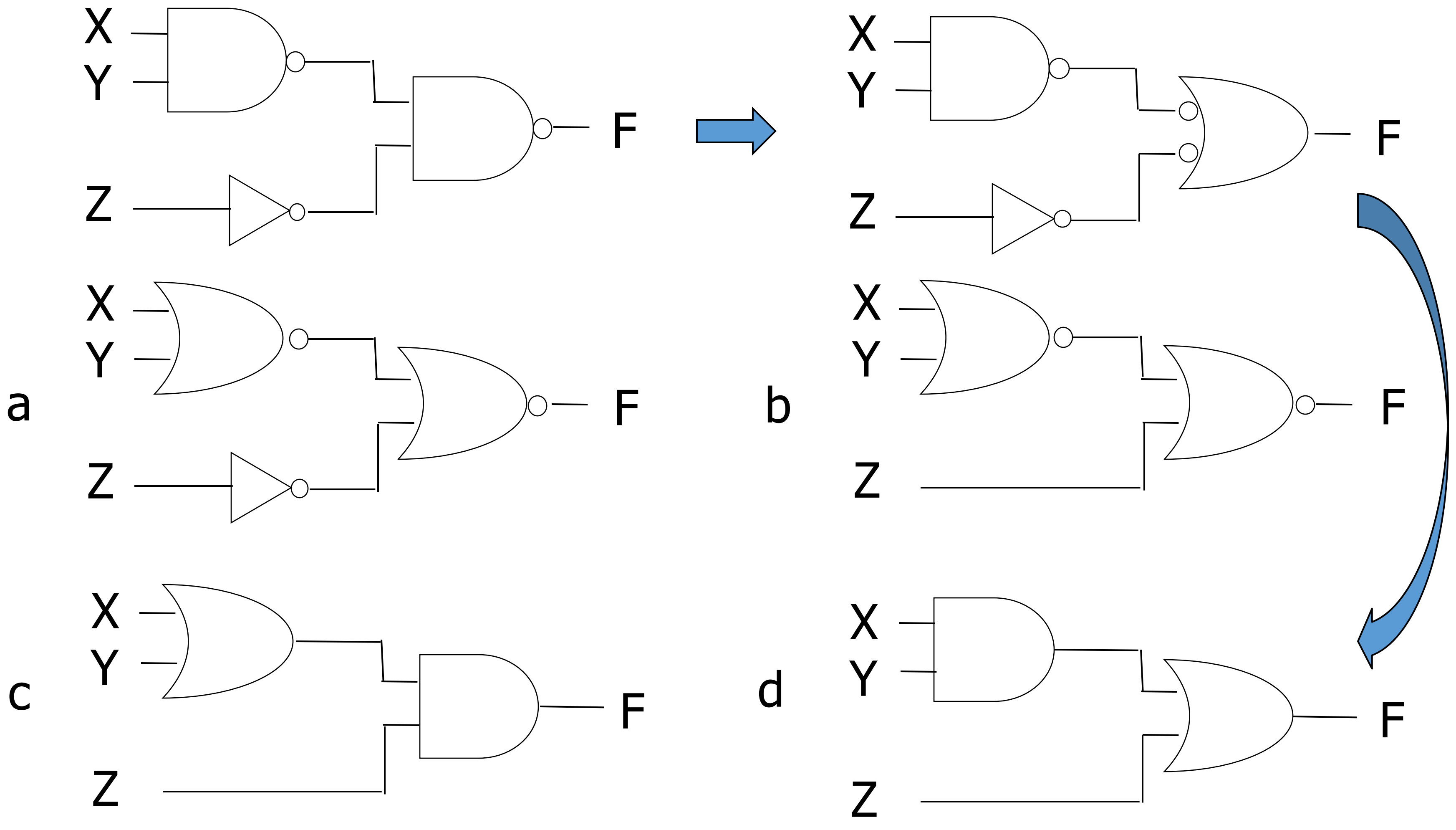
Multi-Level Networks

- Recipe implementation in NAND/NAND (NOR/NOR) circuit
  - Create K-map
  - Derive minimal sum of products (product of sums)
  - Possibly apply factorisations
  - Convert to NAND/NAND (NOR/NOR)

(see previous slides)

Multi-Level Networks – Question 2

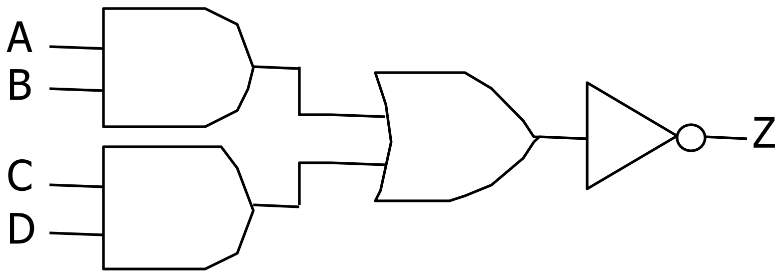
What is an alternative for the circuit below?



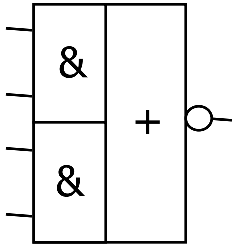
Multi-Level Networks – AND-OR-Invert & OR-AND-Invert gates

Besides NAND and NOR gates, also AOI and OAI gates are preferred over AND or OR gates.

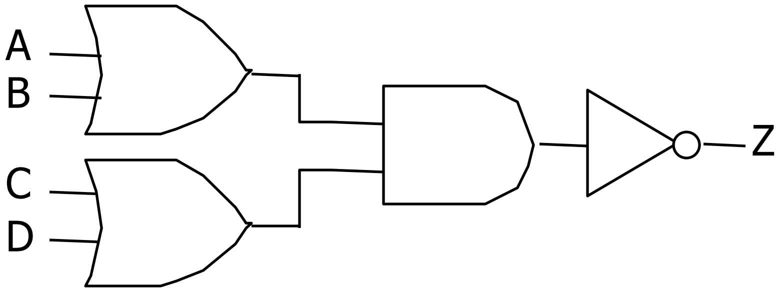
AOI



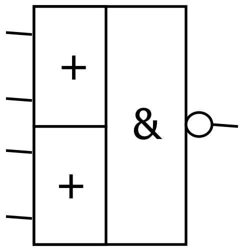
2x2 AOI Schematic Symbol



OAI



2x2 OAI Schematic Symbol

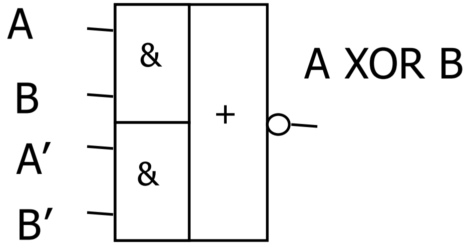


Multi-Level Networks – Implementation Examples

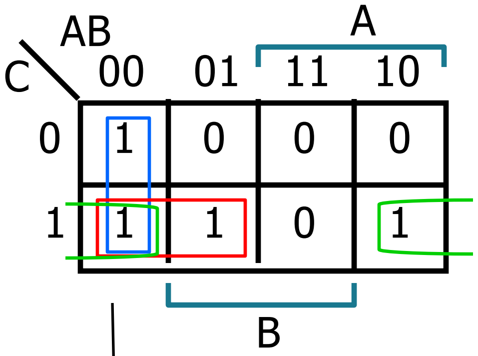
F = A XOR B

$F' = (A \text{ XOR } B)' = (A'B + AB')'$

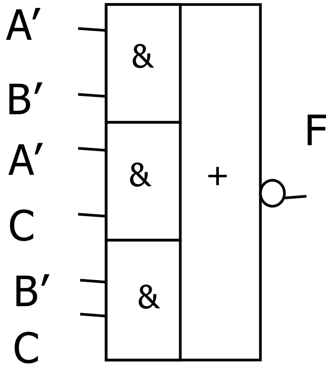
$= (A + B')(A' + B) = AB + A'B'$



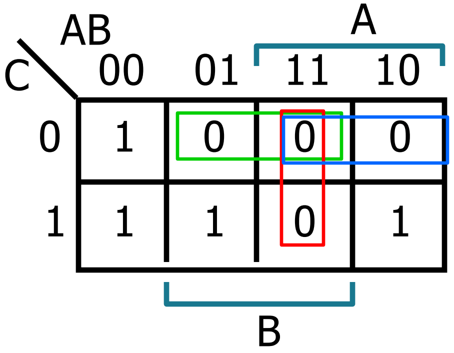
$F = \sum m(2,4,6,7) \Rightarrow F' = \sum m(0,1,3,5)$

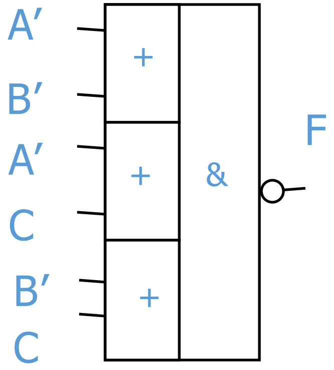


$F' = A'B' + A'C + B'C$



$F = \prod M(0,1,3,5) \Rightarrow F' = \prod M(2,4,6,7)$



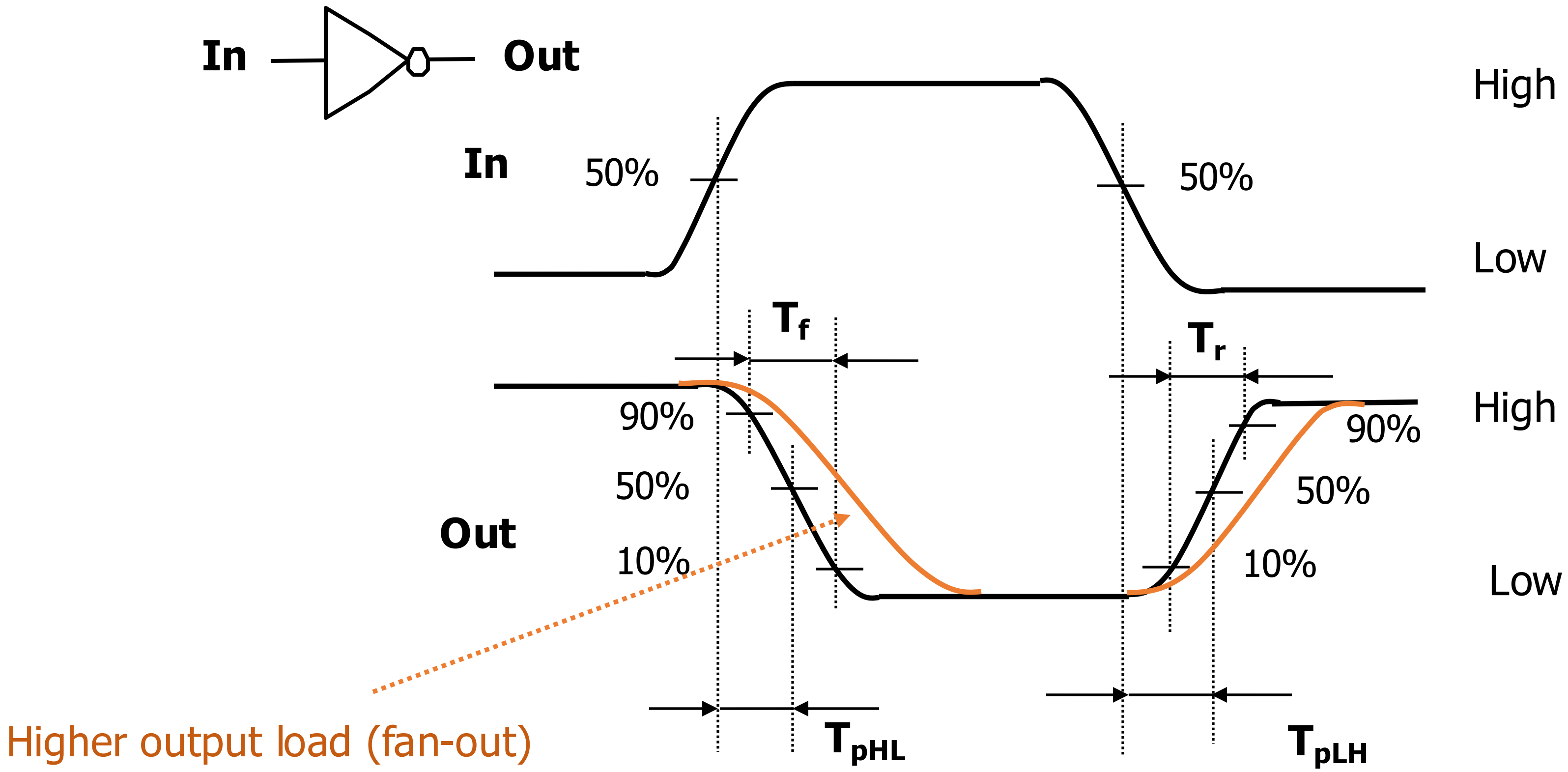


$F' = (A' + B')(A' + C)(B' + C)$

Timing in Combinatorial Circuits

Timing in Combinatorial Circuits

- Time response of gates



T_f : fall time H-L transition
T_{pHL} : propagation time H-L transition

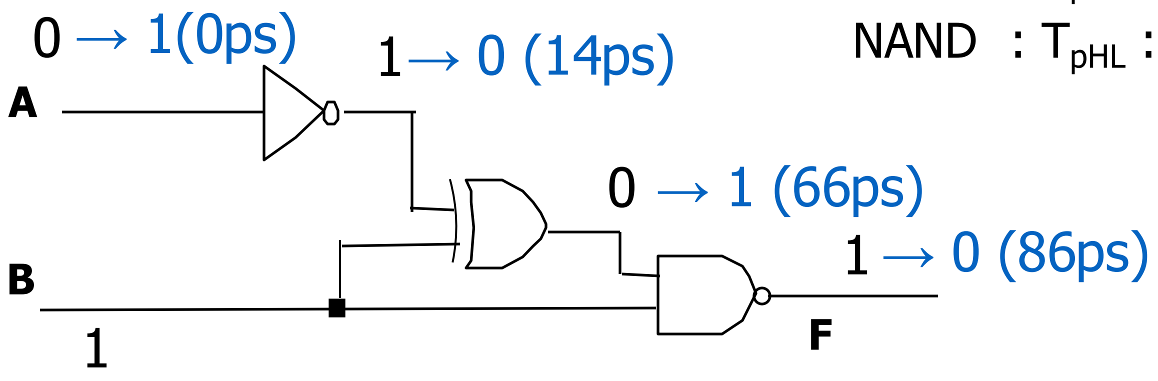
T_r : rise time L-H transition
T_{pLH} : propagation time L-H transition

Often nominal, minimum and maximum values per gate type
Propagation time e.g. TpHL = 2.0 + 1.2 x L ps, with L load

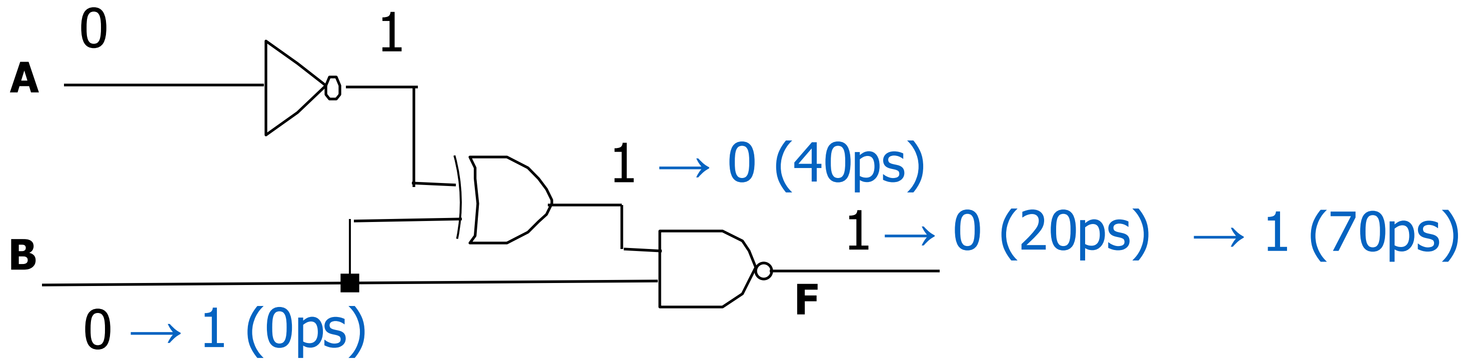
Timing in combinatorial circuits

- Time response of combinatorial circuits

INV : T_{pHL} : 14ps T_{pLH} : 18ps
EXOR : T_{pHL} : 40ps T_{pLH} : 52ps
NAND : T_{pHL} : 20ps T_{pLH} : 30ps



Propagation times add to each other

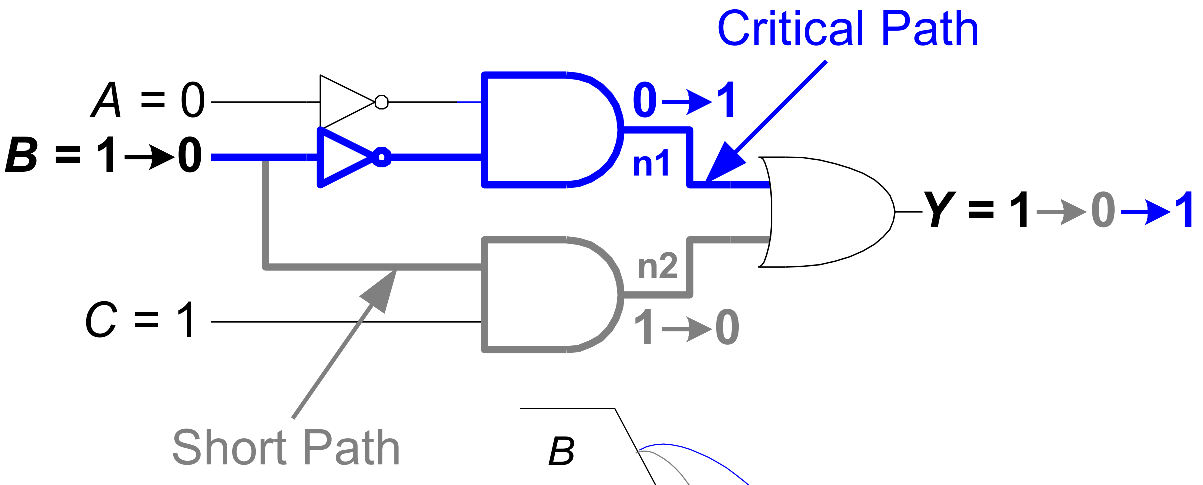


spike/glitch (1-0-1) at the output!

Timing in combinatorial circuits

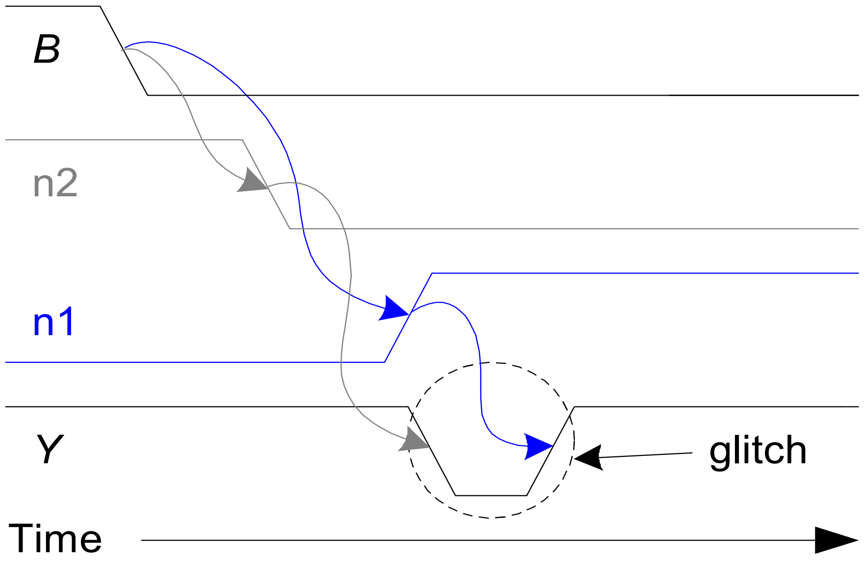
- How to fix Glitches?

Glitches usually cause no problems, but we will show how to fix them



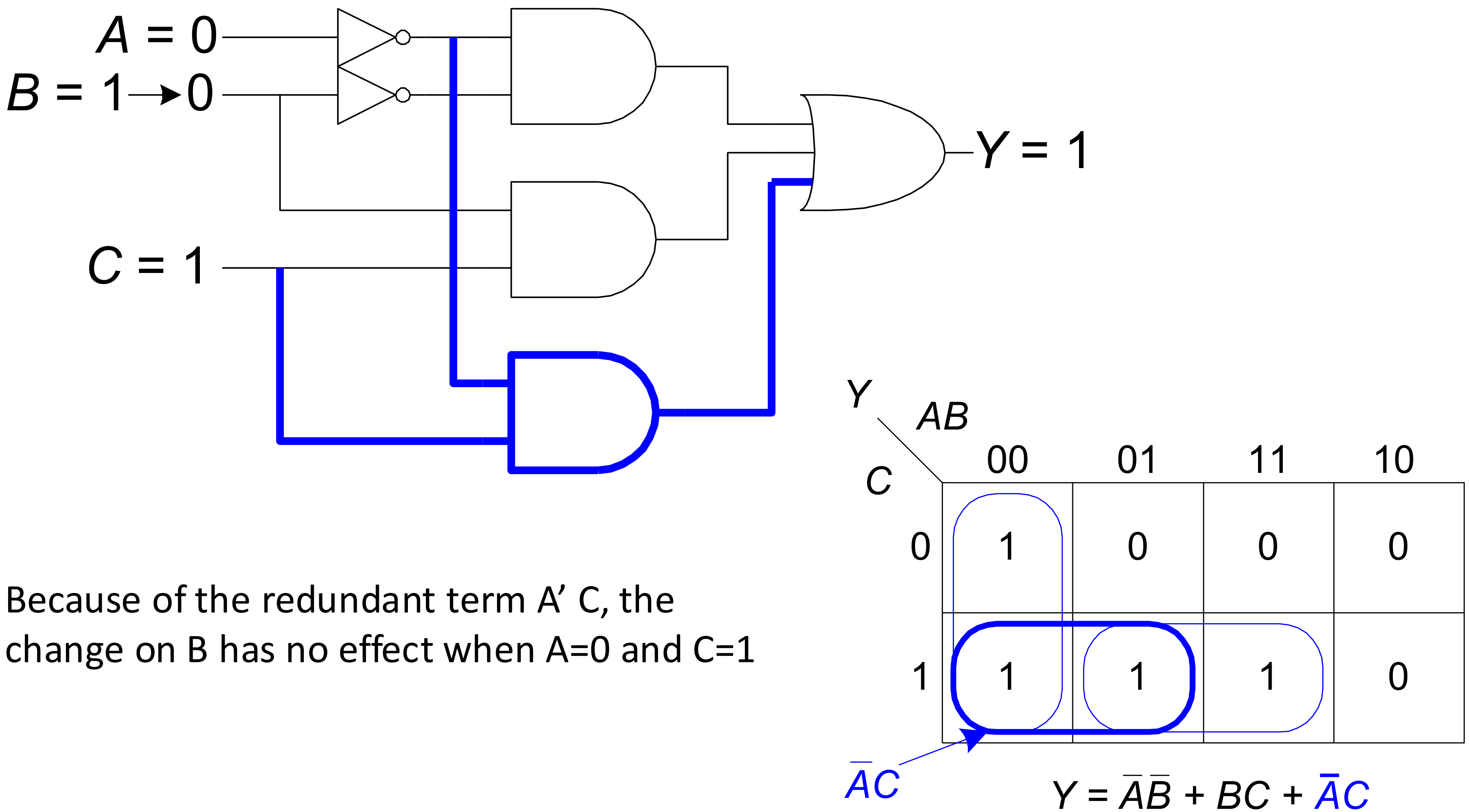
Y: C \ AB	00	01	11	10
0	1	0	0	0
1	1	1	1	0

$Y = \overline{A}\overline{B} + BC$



Timing in combinatorial circuits

- Fixing the Glitch

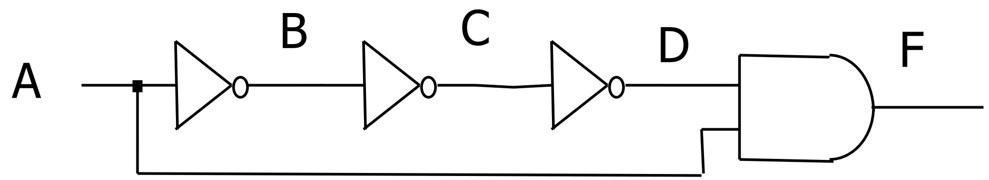


$Y = \bar{A}\bar{B} + BC + \bar{A}C$

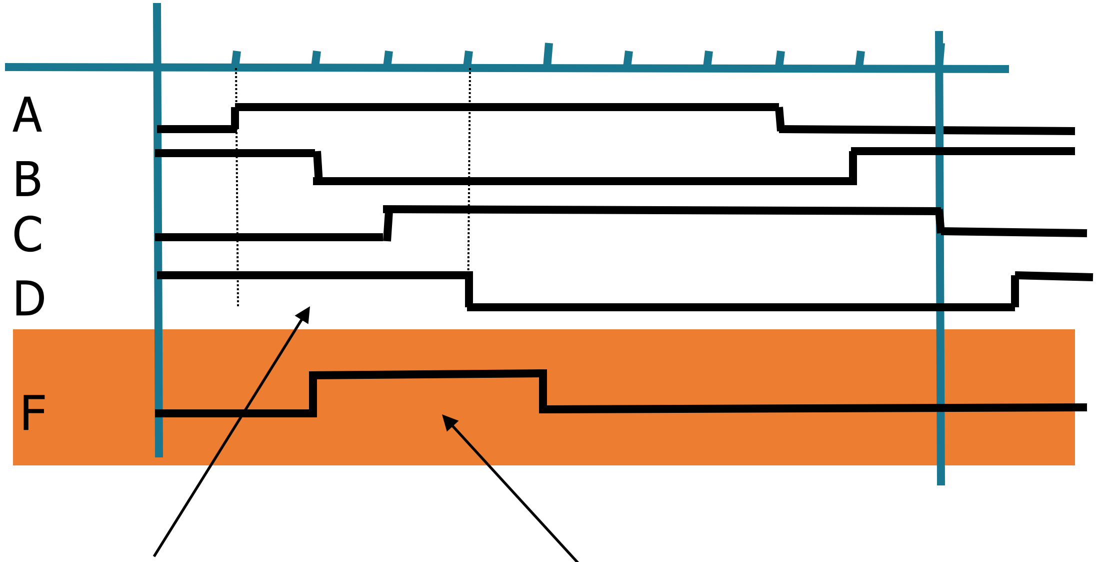
Because of the redundant term A’ C, the change on B has no effect when A=0 and C=1

Timing in Combinatorial Circuits

- Application of spikes: pulse generator



static theory: $F = A' \bullet A = 0$



D stays 1 during 3 gate delays after A changes from 0 to 1

Hence F is not always 0 ==> pulse

Summary

- Canonical expressions two-level networks
  - Sum of Products
  - Product of Sums
- Two-level simplification
- Multi-level networks
  - Factorization
  - Mapping to NAND-NAND and NOR-NOR networks
  - Mapping to AND-OR-INV and OR-AND-INV gates
- Timing in combinatorial networks
  - Time response of gates
  - Time response of combinatorial circuits
  - Fixing the Glitch

To do list

- Reading Material book "Digital Design":
  - Sections 2.2, 2.3.5, 2.5, 2.7 and 2.9

- Assignments for this lecture:
  - Gated Practise Lecture 3

Thank you