Digital Logic Design

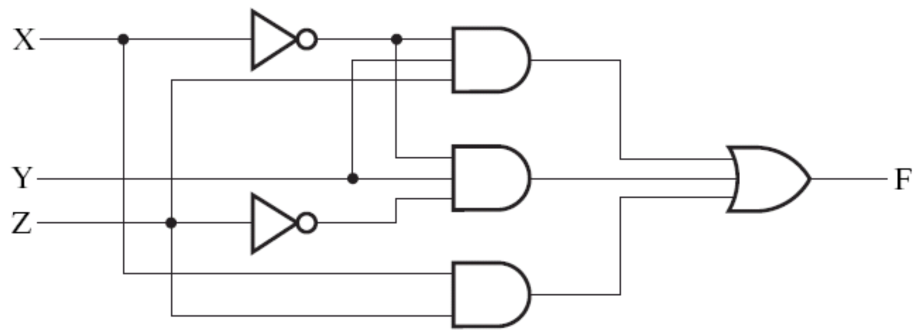
Combinational Logic (Karnuph Maps)

Simplification

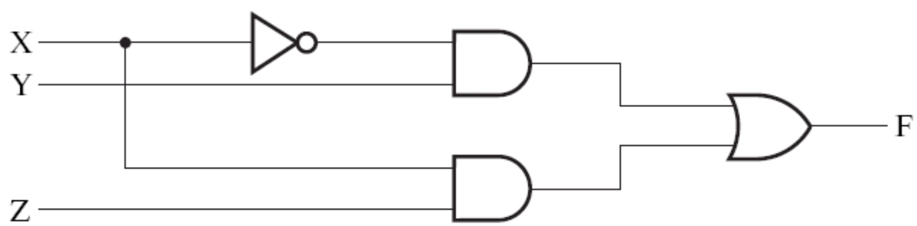
- Simplification using Algebra
- Simplification using Karnaugh Maps (K-Maps)

Simplification using Algebra

$$\begin{aligned} F &= X'YZ + X'YZ' + XZ \\ &= X'Y(Z+Z') + XZ \\ &= X'Y.1 + XZ \\ &= X'Y + XZ \end{aligned}$$



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(b) $F = \overline{X}Y + XZ$

- Simplification may mean different things
- here it means less number of literals

Simplification Revisited

- Algebraic methods for minimization is limited:
  - No formal steps, need experience.
  - No guarantee that a minimum is reached
  - Easy to make mistakes
- Karnaugh maps (k-maps) is an alternative convenient way for minimization:
  - A graphical technique
  - Introduced by Maurice Karnaugh in 1953
- K-maps for up to 4 variables are straightforward to build
- Building higher order K-maps (5 or 6 variable) are a bit more cumbersome
- Simplified expression produced by K-maps are in SOP or POS forms

Gray Code &
Truth Table Adjacencies

•Remember that Only one bit changes with each number increment in gray codes

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

These minterms are adjacent in a gray code sense – they differ by only one bit.

We can apply :

F = A'B' + A'B = A'(B'+B) = A' (1)
= A'

- When we group adjacent minterms (gray codes), we Keep common literal only!

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

Same idea:

F = A'B + AB = B

Keep common literal only!

K-Map

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

A different way to draw a truth table !

Take advantage of adjacency and gray codes

A \ B	0	1
0	0 A' B'	1 A' B
1	0 A B'	1 A B

F = A'B + AB = B

Keep common literal only!

Minimization (Simplification) with K-maps

1. Draw a K-map
2. Combine maximum number of 1's following rules:
    1. Only adjacent squares can be combined
    2. All 1's must be covered
    3. Covering rectangles must be of size 1,2,4,8, … 2^n
3. Check if all covering are really needed
4. Read off the SOP expression

2-variable K-map

- Given a function with 2 variables: F(X,Y), the total number of minterms are equal to 4:

  m_0, m_1, m_2, m_3

- The size of the k-map is always equal to the total number of minterms.
  - Each entry of the k-map corresponds to one minterm for the function:
  - Row 0 represents: X'Y', X'Y
  - Row 1 represents: XY', XY'

X \ Y	0	1
0	m_0	m_1
1	m_2	m_3

X \ Y	0	1
0	*0*	*1*
1	*2*	*3*

Example 1

For a given function F(X,Y) with the following truth table, minimize it using k-maps

X	Y	F
0	0	0
0	1	0
1	0	1
1	1	1

X \ Y	0	1
0	0	0
1	1	1

Combining all the 1's in only the adjacent squares

The final reduced expression is given by the common literals from the combination:

- Therefore, since for the combination, Y has different values (0, 1), and X has a fixed value of 1,

The reduced function is: F(X,Y) = X

Example 2

Q. Simplify the function $F(X,Y) = \sum m(1,2,3)$

Sol. This function has 2 variables, and three 1-squares (three minterms where function is 1)

$F = m_1 + m_2 + m_3$

Note: The 1-squares can be combined more than once

X \ Y	0	1
0	0	1
1	1	1

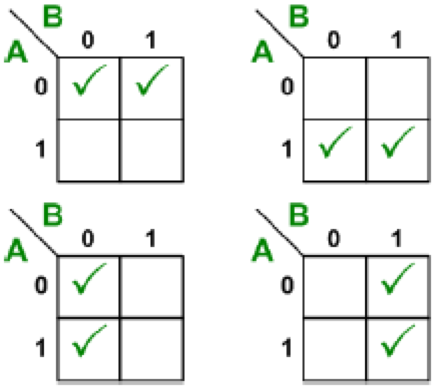
Y is the common literal in the adjacent 1-squares

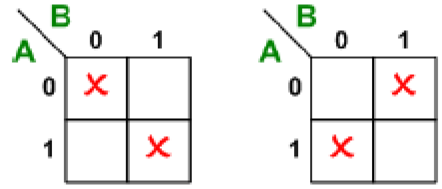
X is the common literal

Minimized expression: F = X + Y

2 variable K-Maps (Adjacency)

In an n-variable k-map, each square is adjacent to exactly *n* other squares





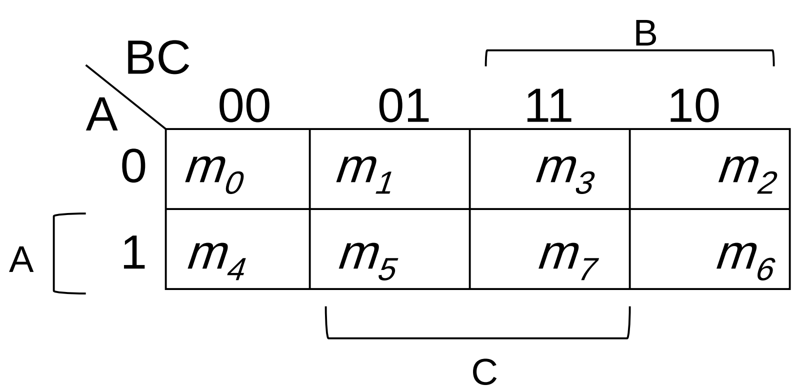
Q: What if you have 1 in all squares?

The boolean function does not depend on the variable , so it is a fixed logic 1

3-variable K-maps

- For 3-variable functions, the k-maps are larger and look different.
- Total number of minterms that need to be accommodated in the k-map = 8

To maintain adjacency neighbors don't have more than 1 different bit



3-variable K-maps

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

A \ BC	00	01	11	10
0	$\mathbf{m_0}$	m_1	m_3	$\mathbf{m_2}$
1	$\mathbf{m_4}$	m_5	m_7	$\mathbf{m_6}$

Note: You can only combine a power of 2 adjacent 1-squares. For e.g. 2, 4, 8, 16 squares. You cannot combine 3, 7 or 5 squares

- Minterms m_o, m_2, m_4, m_6 can be combined as m_0 and m_2 are adjacent to each other, m_4 and m_6 are adjacent to each other
- m_o and m_4 are also adjacent to each other, m_2 and m_6 are also adjacent to each other

Example 1

Simplify F = $\sum m(1, 3, 4, 6)$ using K-map

B

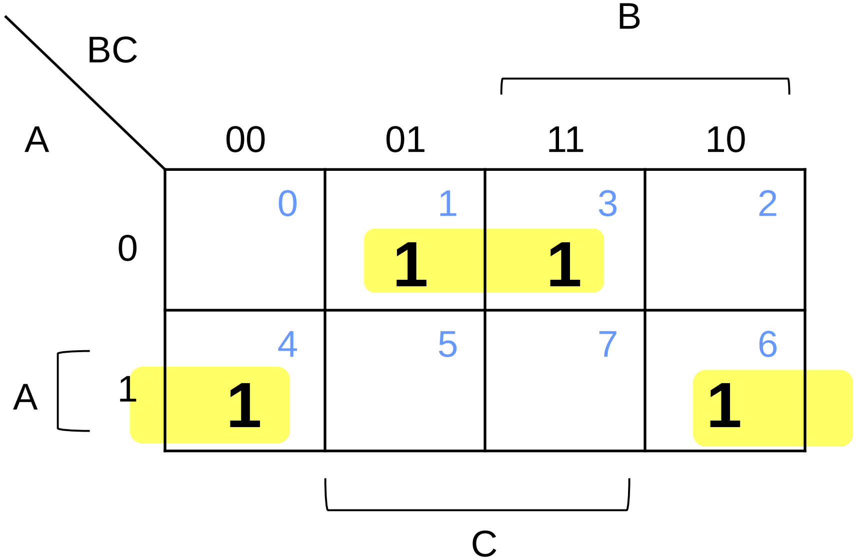
A \ BC	00	01	11	10
0	0	1 **1**	3 **1**	2
A 1	4 **1**	5	7	6 **1**

C

Example 1

Simplify F = $\sum m(1, 3, 4, 6)$ using K-map

F = A'C + AC'



Example 2

Simplify F = $\sum m(0,1, 2, 4, 6)$ using K-map

B

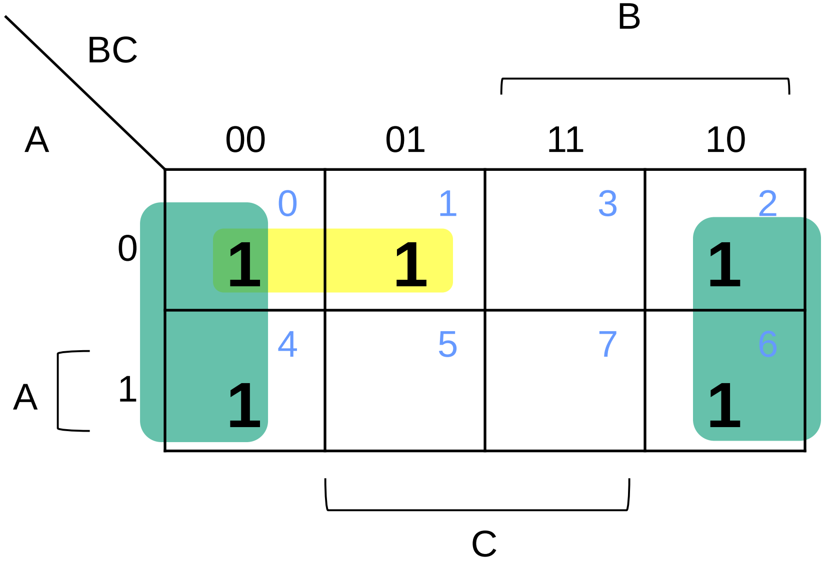
A \ BC	00	01	11	10
0	0 **1**	1 **1**	3	2 **1**
A 1	4 **1**	5	7	6 **1**

C

Example 2

Simplify $F = \sum m(0, 1, 2, 4, 6)$ using K-map

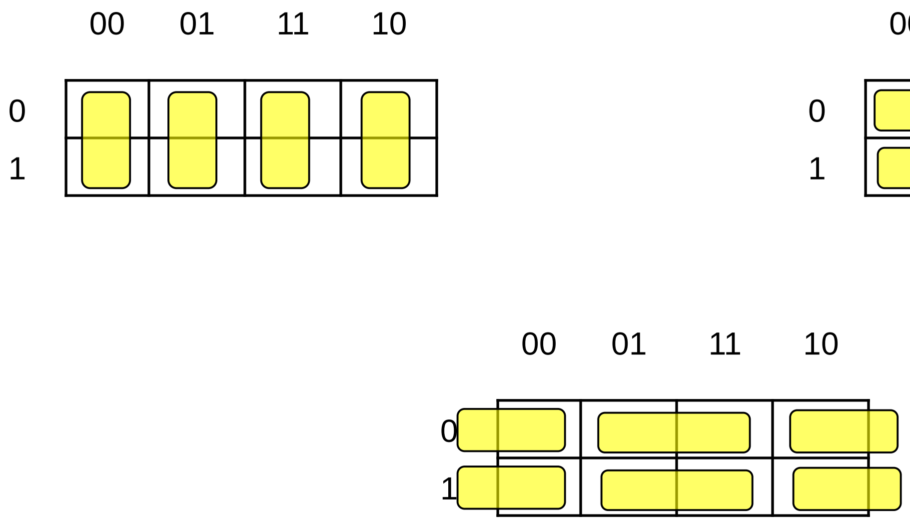
$F = A'B' + C'$



3 variable K-Maps (Adjacency)

A 3-variable map has 12 possible groups of 2 minterms

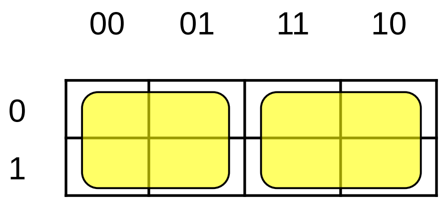
They become product terms with 2 literals

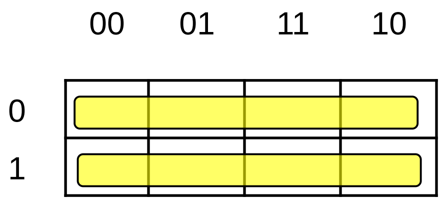


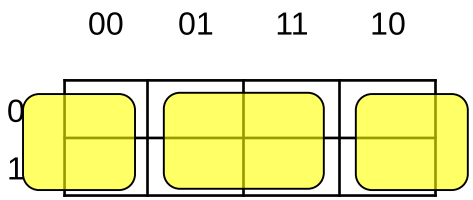
3 variable K-Maps (Adjacency)

A 3-variable map has 6 possible groups of 4 minterms

They become product terms with 1 literals







4-variable K-maps

A 4-variable function will consist of 16 minterms and therefore a size 16 k-map is needed

Each square is adjacent to 4 other squares

A square by itself will represent a minterm with 4 literals

Combining 2 squares will generate a 3-literal output

Combining 4 squares will generate a 2-literal output

Combining 8 squares will generate a 1-literal output

C

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

B

A

D

4-variable K-maps (Adjacency)

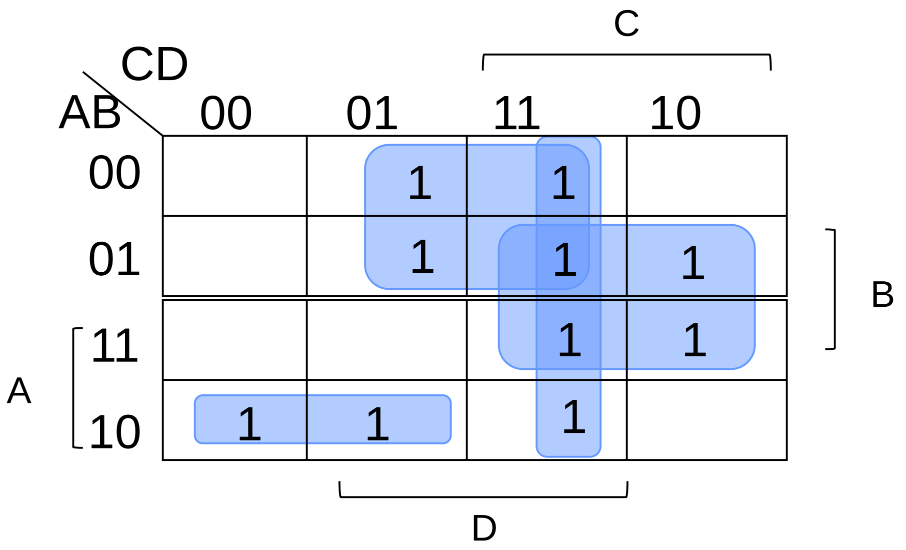
AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Note: You can only combine a power of 2 adjacent 1-squares. For e.g. 2, 4, 8, 16 squares. You cannot combine 3, 7 or 5 squares

- Right column and left column are adjacent; can be combined
- Top row and bottom column are adjacent; can be combined
- Many possible 2, 4, 8 groupings

Example

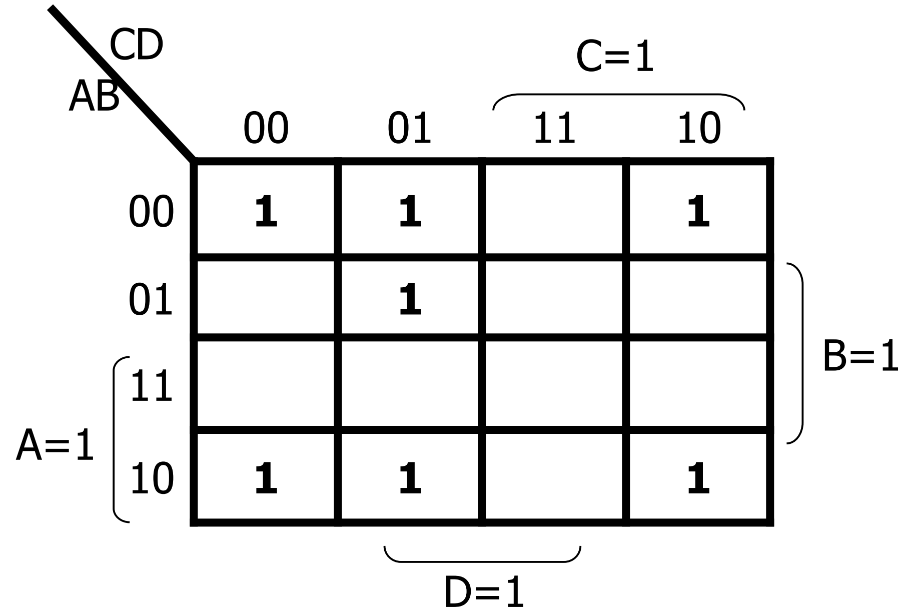
Minimize the function F(A,B,C,D)=∑m(1,3,5,6,7,8,9,11,14,15)



F = CD + A'D + BC + AB'C'

Example

F(A,B,C,D) = Σm(0,1,2,5,8,9,10)

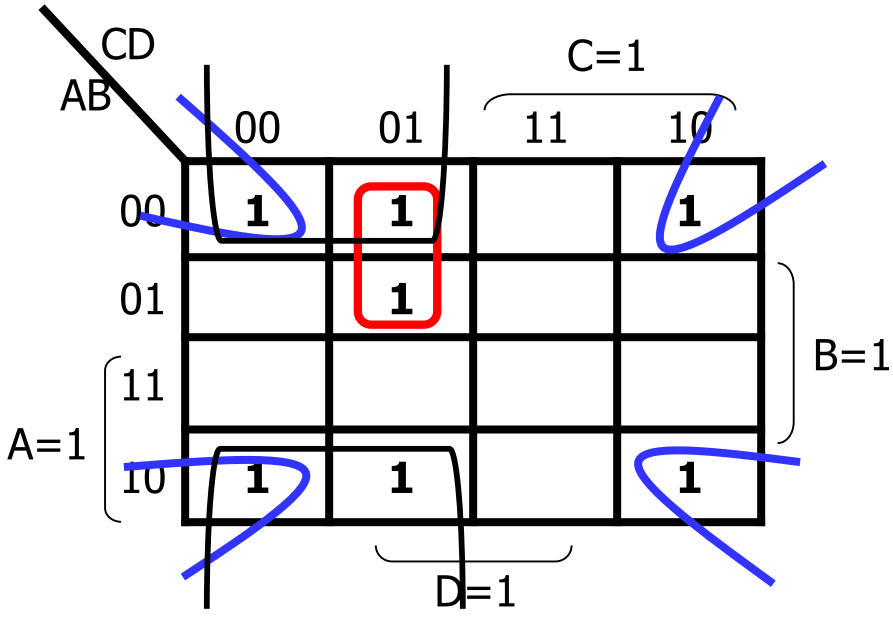


Example

F(A,B,C,D) = Σm(0,1,2,5,8,9,10)

Solution:

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



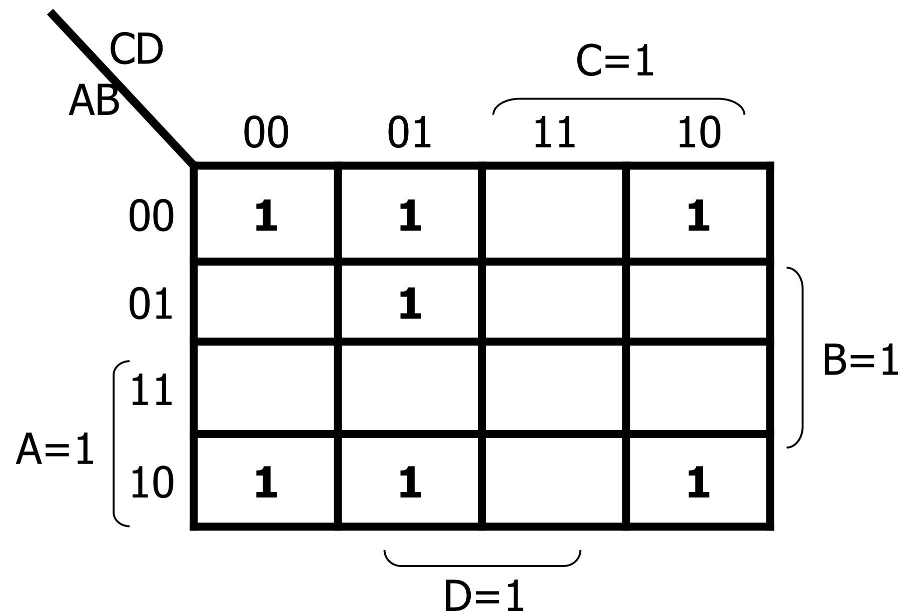
Using (POS)

F(A,B,C,D) = Σm(0,1,2,5,8,9,10)

Write F in the simplified product of sums (POS) not (SOP)

Two methods?

You already know one!



Using (POS)

F(A,B,C,D) = Σm(0,1,2,5,8,9,10)

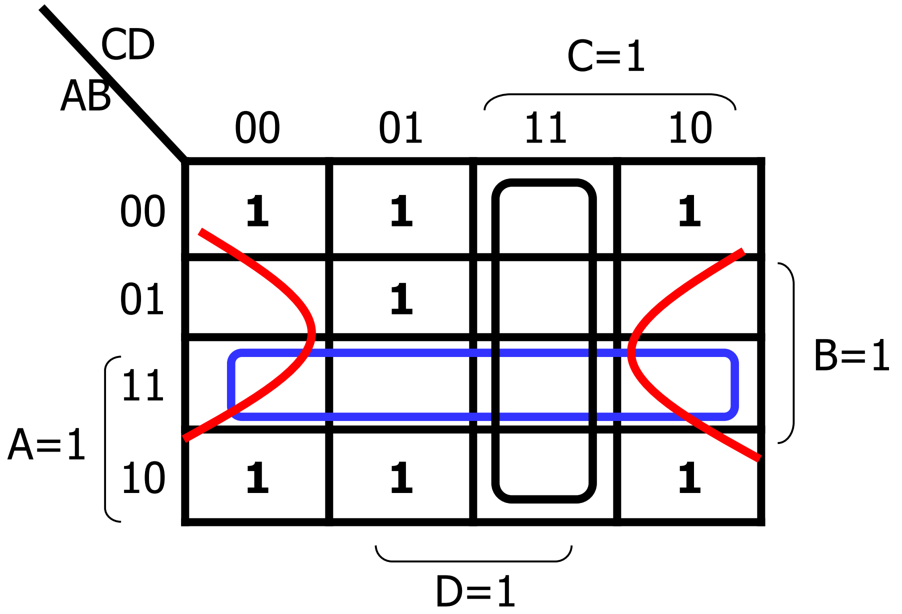
Write F in the simplified product of sums (POS) not (SOP)

- Follow same rule as before but for the ZEROs

F' = AB + CD + BD'

- Therefore,

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



Don’t Cares

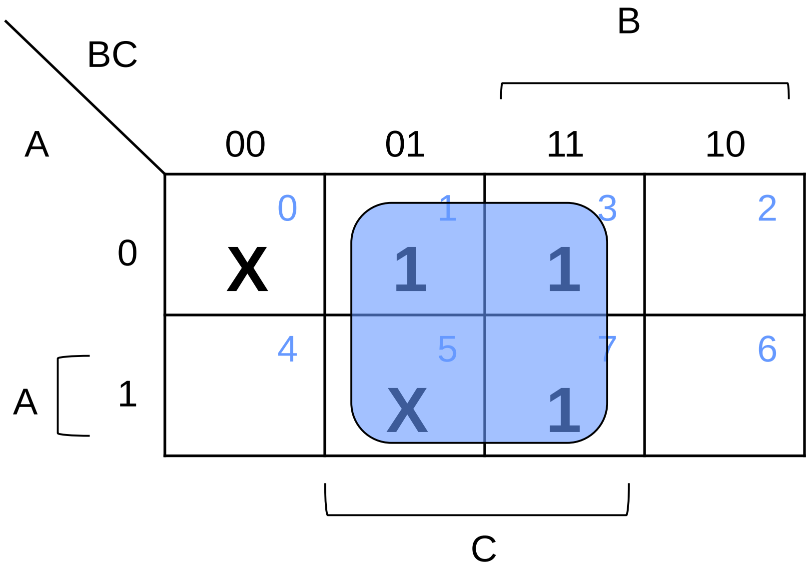
- In some cases, the output of the function (1 or 0) is not specified for certain input combinations either because
  - The input combination never occurs (Example BCD codes), or
  - We don’t care about the output of this particular combination
- Such functions are called incompletely specified functions
- Unspecified minterms for these functions are called don’t cares
- While minimizing a k-map with don’t care minterms, their values can be selected to be either (1 or 0) depending on what is needed for achieving a minimized output.

Example

$F = \sum m(1, 3, 7) + \sum d(0, 5)$

Circle the x's that help get bigger groups of 1's (or 0's if POS).

Don't circle the x's that don't help.



F = C

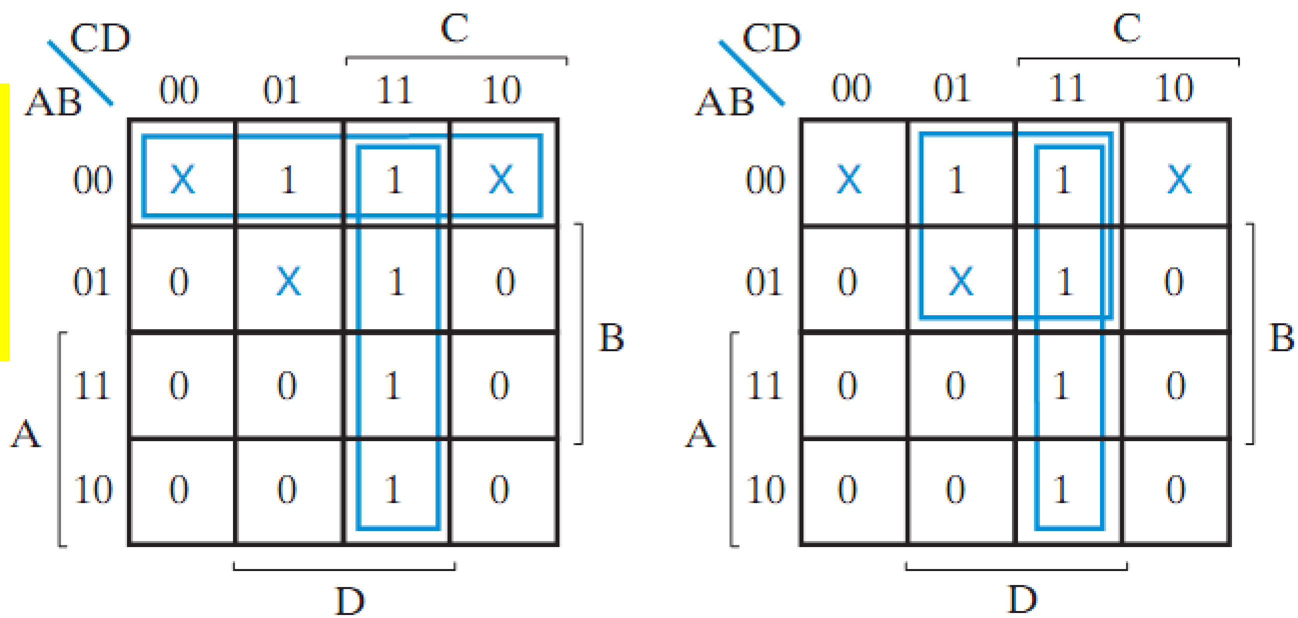
Example 2

$F(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 5)$

-Two possible solutions!

-Both acceptable.

-All 1's covered



(a) $F = CD + \overline{A}\,\overline{B}$ (b) $F = CD + \overline{A}D$

Src: Mano's Textbook

5-variable K-maps

A=0

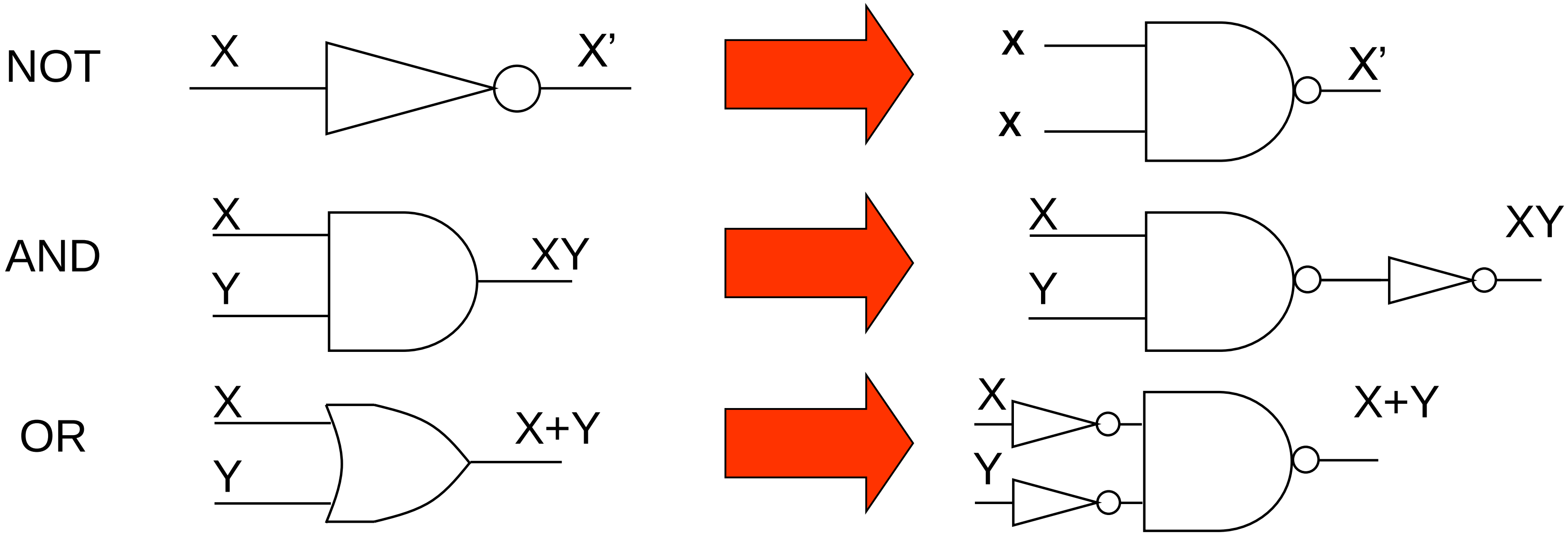
BC \ DE	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

A=1

	00	01	11	10
	m_{16}	m_{17}	m_{19}	m_{18}
	m_{20}	m_{21}	m_{23}	m_{22}
	m_{28}	m_{29}	m_{31}	m_{30}
	m_{24}	m_{25}	m_{27}	m_{26}

- 32 minterms require 32 squares in the k-map
- Minterms 0-15 belong to the squares with variable A=0, and minterms 16-32 belong to the squares with variable A=1
- Each square in A’ is also adjacent to a square in A (one is above the other)
- Minterm 4 is adjacent to 20, and minterm 15 is to 31

NAND Gate is Universal



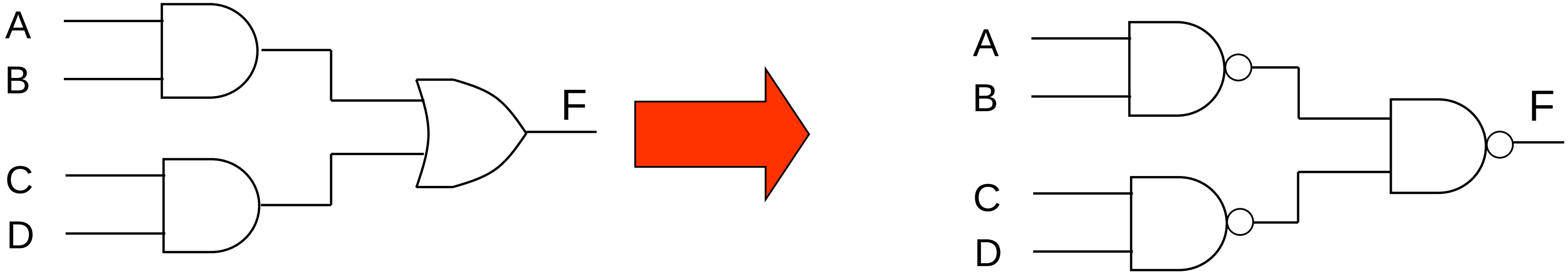
- Therefore, we can build all functions we learned so far using NAND gates ONLY *(Exercise: Prove that NOT can be built with NAND)*
- NAND is a UNIVERSAL gate

Rules for 2-Level NAND Implementations

1. Simplify the function and express it in sum-of-products form
2. Draw a NAND gate for each product term (with 2 literals or more)
3. Draw a single NAND gate at the 2nd level (in place of the OR gate)
4. A term with single literal requires a NOT

Implementation using NANDs

Example: Consider F = AB + CD



Proof:

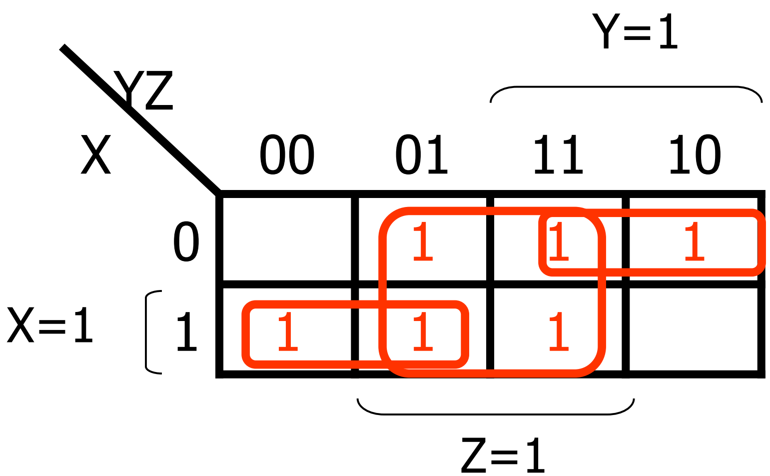
F = F′′= ((AB)′.(CD)′)′

= ((AB)′)′ + ((CD)′)′

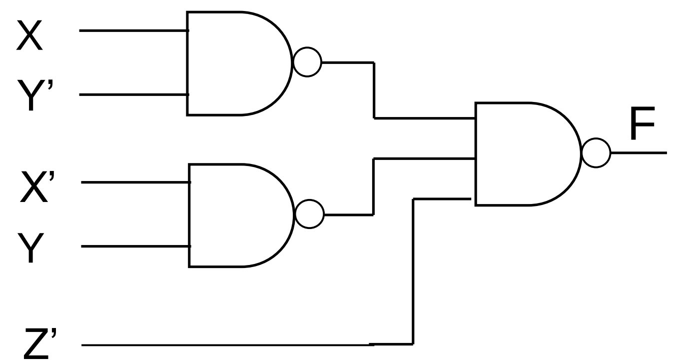
= AB + CD

Implementation using NANDs

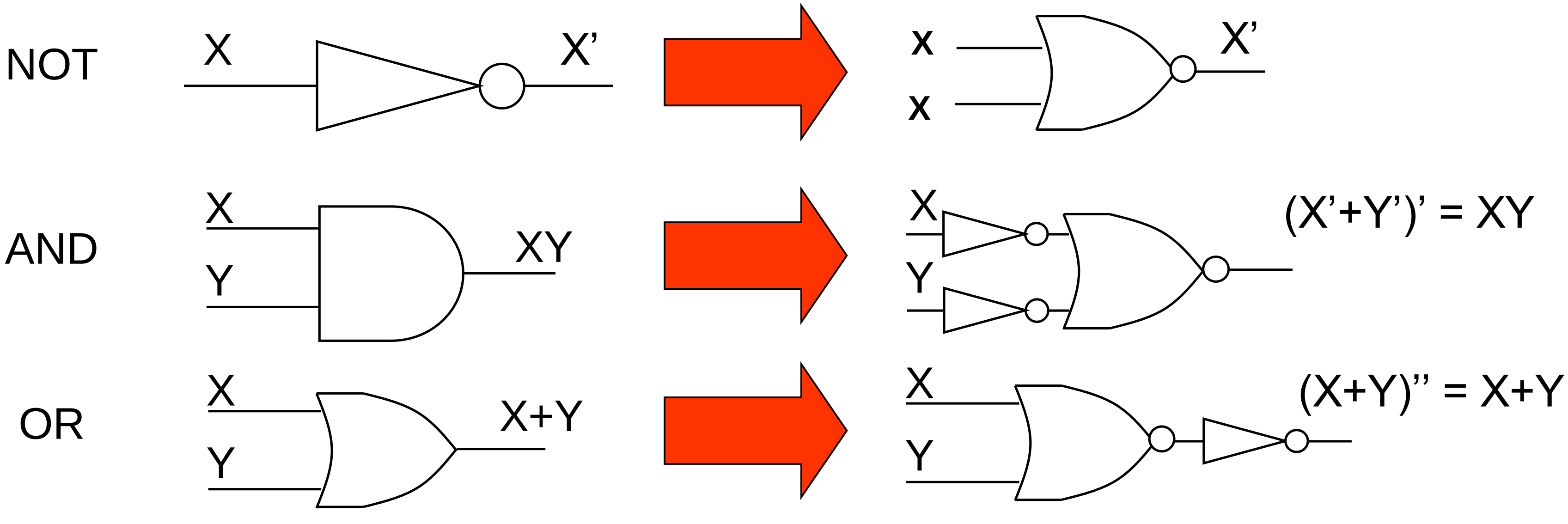
Consider F =Σm(1,2,3,4,5,7) – Implement using NAND gates



F(X,Y) = Z+XY′+X′Y



NOR Gate is Universal



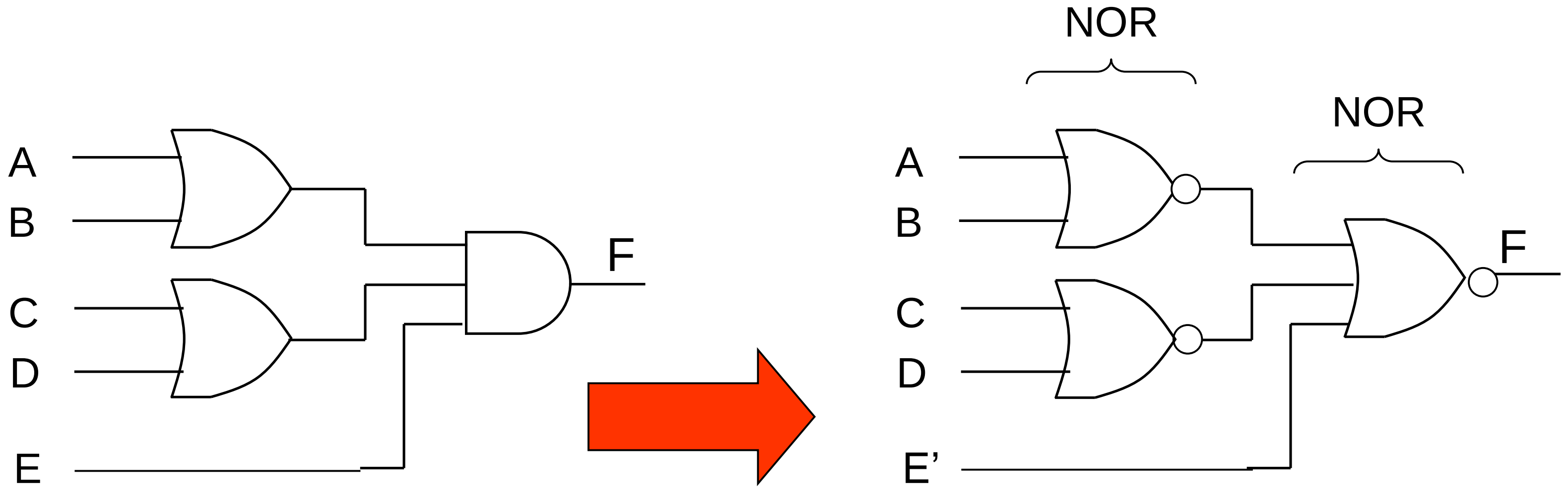
- Therefore, we can build all functions we learned so far using NOR gates ONLY *(Exercise: Prove that NOT can be built with NOR)*
- NOR is a UNIVERSAL gate

Rules for 2-Level NOR Implementations

1. Simplify the function and express it <u>in product of sums form</u>
2. Draw a NOR gate (using OR-NOT symbol) for each sum term (with 2 literals or more)
3. Draw a single NOR gate (using NOT-AND symbol) the 2nd level (in place of the AND gate)
4. A term with single literal requires a NOT

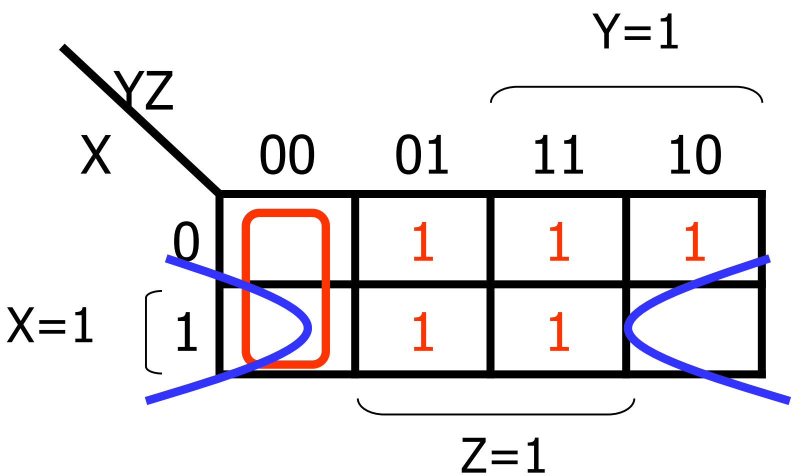
Implementation using NOR gates

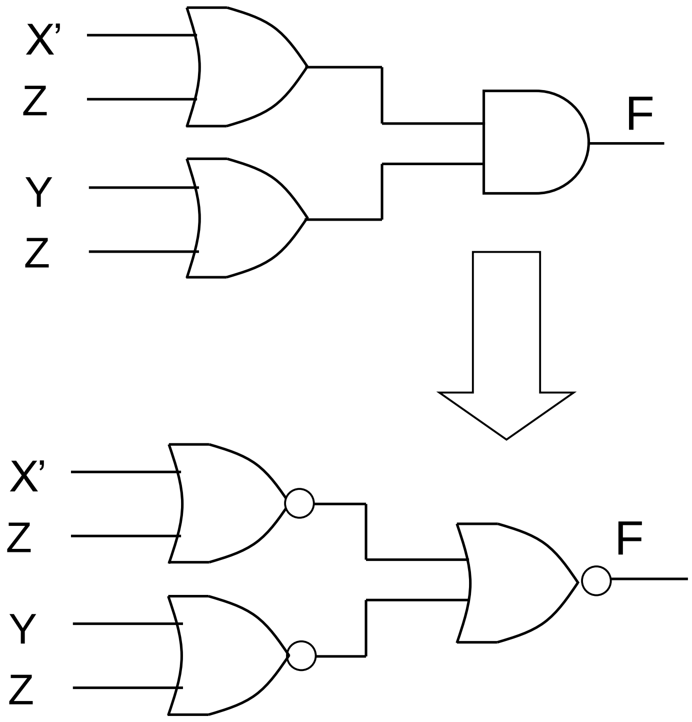
Consider F = (A+B)(C+D)E



Implementation using NOR gates

Consider F =Σm(1,2,3,5,7) – Implement using NOR gates



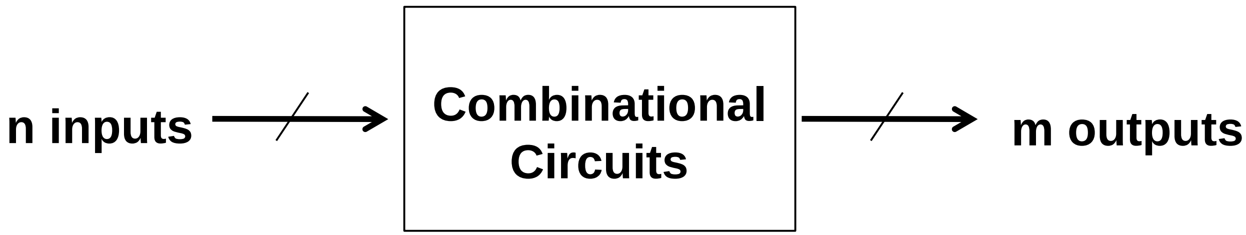


F′(X,Y) = Y′Z′+XZ′, or
F(X,Y) = (Y+Z)(X′+Z)

Combinational Circuits

- Two classes of logic circuits:
  - Combinational Circuits
  - Sequential Circuits
- A **Combinational circuit** consists of logic gates
  - Output depends only on input
- A **Sequential circuit** consists of logic gates and memory
  - Output depends on current inputs and previous ones (stored in memory)
  - Memory defines the state of the circuit.

Combinational Circuits



- A combinational circuit has:
  - n Boolean inputs (1 or more),
  - m Boolean outputs (1 or more)
  - logic gates mapping the inputs to the outputs

- How to design a combinational circuit?
  - Use all the information and tools you learned
    - Binary system, Boolean Algebra, K-Maps, etc.
  - Follow the step-by-step procedure given next

Design Procedure

1. Specification
   - Write a specification for the circuit if one is not already available
   - Specify/Label input and output
2. Formulation
   - Derive a truth table or initial Boolean equations that define the required relationships between the inputs and outputs, if not in the specification
3. Optimization
   - Apply 2-level and multiple-level optimization (Boolean Algebra, K-Map, software)
   - Draw a logic diagram for the resulting circuit using ANDs, ORs, and inverters

Design Procedure (Cont.)

4. Technology Mapping
   - Map the logic diagram to the implementation technology selected (e.g. map into NANDs or NORs)
5. Verification
   - Verify the correctness of the final design manually or using simulation

<u>Practical Considerations:</u>
- Cost of gates (Number)
- Maximum allowed delay
- Fanin/Fanout

Example 1 (cont.)

Question: Design a circuit that has a 3-bit input and a single output (F) specified as follows:

- F = 0, when the input is less than $(5)_{10}$
- F = 1, otherwise

Solution:

Step 1 (Specification):

- Label the inputs (3 bits) as X, Y, Z
  - X is the most significant bit, Z is the least significant bit
- The output (1 bit) is F:
  - F = 1 → $(101)_2$, $(110)_2$, $(111)_2$
  - F = 0 → other inputs

Example 1 (cont.)

Step 2 (Formulation)

Obtain Truth table

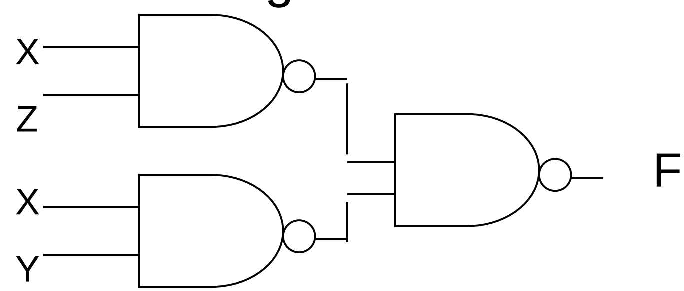
X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 3 (Optimization)

X \ YZ	00	01	11	10
0	0	0	0	0
1	0	1	1	1

F = XZ + XY

Circuit Diagram



SOP can be implemented using all NAND circuit

Example 2

Question (BCD-to-Seven-Segment Decoder)

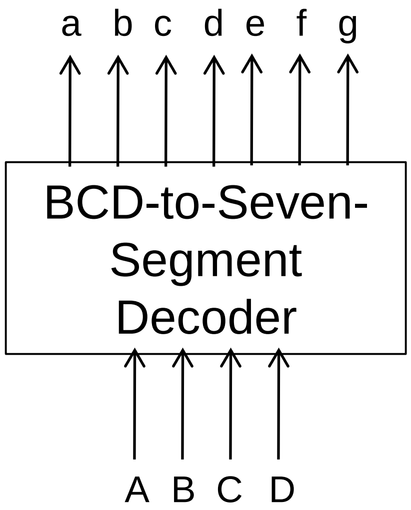
src: Mano's book

- A seven-segment display is digital readout found in electronic devices like clocks, TVs, etc.
  - Made of seven light-emitting diodes (LED) segments; each segment is controlled separately.
- **A BCD-to-Seven-Segment decoder** is a combinational circuit
  - Accepts a decimal digit in BCD (input)
  - Generates appropriate outputs for the segments to display the input decimal digit (output)

Example 2 (cont.)

Step 1 (Specification):

- 4 inputs (A, B, C, D)
- 7 outputs (a, b, c, d, e, f, g)



Example 2 (cont.)

Step 2 (Formulation)

	BCD Input				7 Segment Decoder						
Decimal	**A**	**B**	**C**	**D**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10-15	All Other Inputs				0	0	0	0	0	0	0

Invalid
BCD
codes
=
No Light

Example 2 (cont.)

Step 3 (Optimization)

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	0	0
10	1	1	0	0

a

AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	0	0	0
10	1	1	0	0

b

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	0	0	0
10	1	1	0	0

c

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	0	0	0	0
10	1	1	0	0

d

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	0	0	0	0
10	1	0	0	0

e

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

f

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	0	0	0
10	1	1	0	0

g

Example 2 (cont.)

Step 3 (Optimization) (cont.)

a = A’C + A’BD + AB’C’ + B’C’D’

b = A’B’ + A’C’D’ + A’CD + B’C’

c = A’B + B’C’ + A’C’ + A’D

d = A’CD’ + A’B’C + B’C’D’+AB’C’+A’BC’D

e = A’CD’ + B’C’D’

f = A’BC’ + A’C’D’ + A’BD’ + AB’C’

g = A’CD’ + A’B’C + A’BC’ + AB’C’

Exercise: Draw the circuit