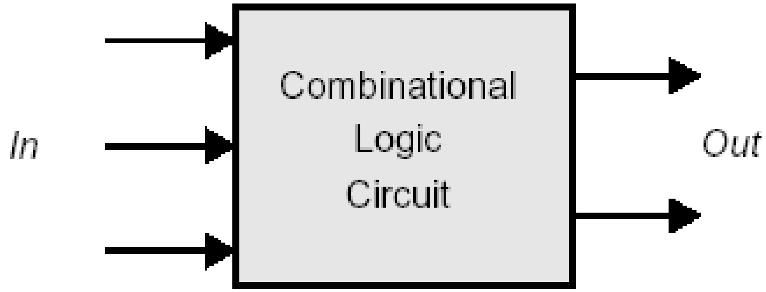
Lecture 7 Overview

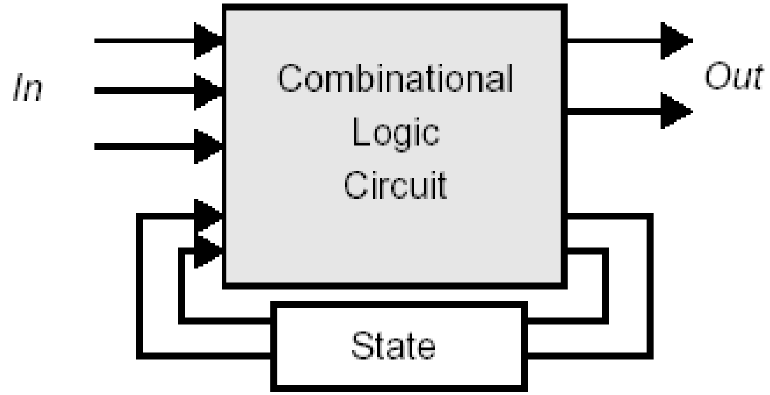
- Boolean Algebra
- Standard or Canonical forms
- Minterms/Maxterms
- Karnaugh maps

Combinational Logic Circuits

- Logic gates combine several logic-variable inputs to produce a logic-variable output.
- ***Combinational logic circuits*** are "memoryless" because their output value at a given instant depends only on the input values at that instant.



(a) Combinational



(b) Sequential

- ***Sequential logic circuits*** possess memory because their present output value depends on previous as well as present input values.

Terminology

- A **literal** is a variable or its complement, e.g. X, X' or $\overline{X}$
- An **expression** consists of literals combined with AND, OR parentheses, complementation.
  - **X+Y**
  - **P Q R**
  - **A + B C**
- If in doubt about the meaning of an expression, make liberal use of parentheses.
- An **equation** consists of the form: Variable = Expression.

$$P = (\overline{(X + Y)}) + \overline{A}\ B$$

Simplifying Logic Functions (Logic Synthesis)

Logic Synthesis: ***reduce complexity of the gate level implementation***

- reduce number of literals (gate inputs)
- reduce number of levels of gates
- fewer inputs implies faster gates in some technologies
- fan-ins (number of gate inputs) are limited in some technologies
- fewer levels of gates implies reduced signal propagation delays

Boolean Algebra

- Switching algebra - deals with Boolean values - 0,1
- Positive logic convention - LOW, HIGH - 0,1
- Negative logic seldom used.
- Signal values denoted by X,Y, A, B, C...

Boolean Operators (Truth table)

AND	X . Y

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

OR	X + Y

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Complement (opposite)	X'

X	NOT X
0	1
1	0

Method I: Boolean Algebra

	AND:	***OR:***	***NOT:***	
Identity Law:	$A \cdot 1 = A$	$A + 0 = A$	$\overline{\overline{A}} = A$	***Involution Law:***
	$A \cdot 0 = 0$	$A + 1 = 1$		
	$A \cdot A = A$	$A + A = A$		***Idempotent Law:***
	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$		***Laws of Complementarity:***

Associative Law:

$$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$$

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

Distributive Law:

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

DeMorgan's Theorem:

$$\overline{(A \cdot B)} = \overline{A} + \overline{B} \quad \text{(NAND)}$$

$$\overline{(A + B)} = \overline{A} \cdot \overline{B} \quad \text{(NOR)}$$

Commutative Law:

$$A \cdot B = B \cdot A$$

$$A + B = B + A$$

Precedence:

$$AB = A \cdot B$$

$$A \cdot B + C = (A \cdot B) + C$$

$$A + B \cdot C = A + (B \cdot C)$$

Absorption Law:

$$A + A.B = A$$

$$A.(A + B) = A$$

Boolean Algebra

- **Duality: a dual of a Boolean expression is derived by replacing AND operations by ORs, OR operations by ANDs, constant 0s by 1s, and 1s by 0s (literals are left unchanged).**

Any statement that is true for an expression is also true for its dual!

$$A.(B+C)=(A.B)+(A.C)$$

$$A+(B.C)=(A+B).(A+C)$$

Proving Theorems via Boolean Algebra

Proving theorems via axioms of Boolean Algebra:

prove the theorem:	$X \cdot Y + X \cdot Y' = X$	
distributive law	$X \cdot Y + X \cdot Y' = X \cdot (Y + Y')$	
complementary law	$X \cdot (Y + Y')$	$= X \cdot (1)$
identity	$X \cdot (1)$	$= X$

prove the theorem:	$X + X \cdot Y$	$= X$
Identity	$X + X \cdot Y$	$= X \cdot 1 + X \cdot Y$
distributive law	$X \cdot 1 + X \cdot Y = X \cdot (1 + Y)$	
identity	$X \cdot (1 + Y)$	$= X \cdot (1)$
identity	$X \cdot (1)$	$= X$

Using the Rules of Boolean Algebra

Example: Simplify the following function:

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

Use $\mathrm{X} + \overline{\mathrm{X}} = 1$

$$= \overline{A} \bullet \cancel{\left(\overline{B} + B\right)} \bullet D + B \bullet C \bullet D + A \bullet C \bullet D$$

$$= \overline{A} \bullet D + B \bullet C \bullet D + A \bullet C \bullet D$$

$$= B \bullet C \bullet D + \left(\overline{A} + A \bullet C\right) \bullet D$$

Use $\mathrm{X} + \mathrm{YZ} = (\mathrm{X} + \mathrm{Y}) \cdot (\mathrm{X} + \mathrm{Z})$

and $\mathrm{X} + \overline{\mathrm{X}} = 1$

expand this

$$= B \bullet C \bullet D + \left(\overline{A} + C\right) \bullet D$$

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

=1

$$f(A,B,C,D) = \left(\overline{A} + C\right) \bullet D$$

SOP and POS

• DeMorgan's Theorem shows that any logic function can be implemented by using just OR and NOT gates , or by just AND and NOT gates

• A consequence of this is that any logical expression can be reduced to either a "Sum-of-Products (SOP)" form or a "Product-of-Sums (POS)" form

DeMorgan's Theorem:

$$\overline{(A \cdot B)} = \overline{A} + \overline{B} \quad \text{(NAND)}$$

$$\overline{(A + B)} = \overline{A} \cdot \overline{B} \quad \text{(NOR)}$$

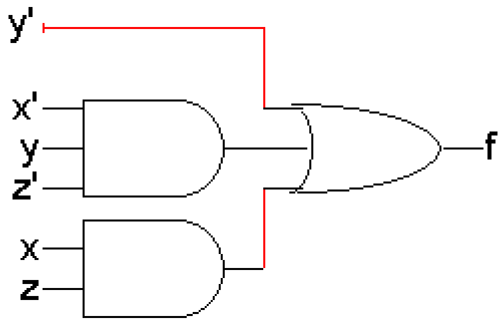
THE DUAL IDEA

SUM OF PRODUCTS (SOP)

- We can write expressions in many ways, but some ways are more useful than others
- A sum of products (SOP) expression contains:
  - Only OR (sum) operations at the "outermost" level
  - Each term that is summed must be a product of literals

f(x,y,z) = y' + x'yz' + xz

- The advantage is that any sum of products expression can be implemented using a two-level circuit
  - literals and their complements at the '
  - AND gates at the first level
  - a single OR gate at the second level



MINTERMS

- A minterm is a special product of literals, in which each input variable appears exactly once.
- A function with n variables has 2^n minterms (since each variable can appear complemented or not)
- A three-variable function, such as f(x,y,z), has $2^3 = 8$ minterms:

x'y'z'	x'y'z	x'yz'	x'yz
xy'z'	xy'z	xyz'	xyz

- Each minterm is true for exactly one combination of inputs:

Minterm	Is true when…	Shorthand
x'y'z'	x=0, y=0, z=0	m_0
x'y'z	x=0, y=0, z=1	m_1
x'yz'	x=0, y=1, z=0	m_2
x'yz	x=0, y=1, z=1	m_3
xy'z'	x=1, y=0, z=0	m_4
xy'z	x=1, y=0, z=1	m_5
xyz'	x=1, y=1, z=0	m_6
xyz	x=1, y=1, z=1	m_7

x	y	z	minterms
0	0	0	m_0
0	0	1	m_2
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

SUM OF MINTERMS FORM

- Every function can be written as a sum of minterms, which is a special kind of sum of products form
- The sum of minterms form for any function is *unique*
- If you have a truth table for a function, you can write a sum of minterms expression just by picking out the rows of the table where the function output is 1.

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$f = x'y'z' + x'y'z + x'yz' + x'yz + xyz'$$
$$= m_0 + m_1 + m_2 + m_3 + m_6$$
$$= \Sigma m(0,1,2,3,6)$$

$$f' = xy'z' + xy'z + xyz$$
$$= m_4 + m_5 + m_7$$
$$= \Sigma m(4,5,7)$$

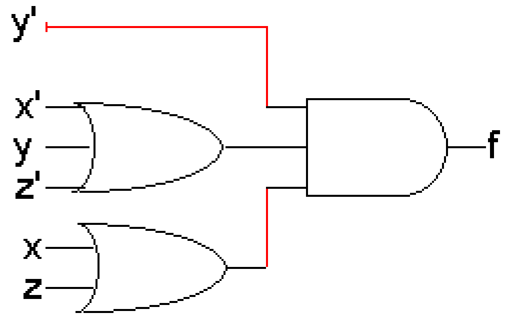
f' contains all the minterms not in f

PRODUCTS OF SUMS (POS)

- A product of sums (POS) expression contains:
  - Only AND (product) operations at the "outermost" level
  - Each term must be a sum of literals

$$f(x,y,z) = y' (x' + y + z') (x + z)$$

- Product of sums expressions can be implemented with two-level circuits
  - literals and their complements at the "0th" level
  - *OR gates* at the first level
  - a single *AND gate* at the second level
- Compare this with sums of products



MAXTERMS

- A maxterm is a *sum* of literals, in which each input variable appears exactly once.
- A function with n variables has 2^n maxterms
- The maxterms for a three-variable function f(x,y,z):

$x' + y' + z'$	$x' + y' + z$	$x' + y + z'$	$x' + y + z$
$x + y' + z'$	$x + y' + z$	$x + y + z'$	$x + y + z$

- Each maxterm is *false* for exactly one combination of inputs:

Maxterm	Is *false* when…	Shorthand
$x + y + z$	x=0, y=0, z=0	M_0
$x + y + z'$	x=0, y=0, z=1	M_1
$x + y' + z$	x=0, y=1, z=0	M_2
$x + y' + z'$	x=0, y=1, z=1	M_3
$x' + y + z$	x=1, y=0, z=0	M_4
$x' + y + z'$	x=1, y=0, z=1	M_5
$x' + y' + z$	x=1, y=1, z=0	M_6
$x' + y' + z'$	x=1, y=1, z=1	M_7

PRODUCT OF MAXTERMS FORM

- Every function can be written as a *unique* product of maxterms
- If you have a truth table for a function, you can write a product of maxterms expression by picking out the rows of the table where the function output is 0. (Be careful if you're writing the actual literals!)

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$f = (x' + y + z)(x' + y + z')(x' + y' + z')$$
$$= M_4\ M_5\ M_7$$
$$= \Pi M(4,5,7)$$

$$f' = (x + y + z)(x + y + z')(x + y' + z)$$
$$(x + y' + z')(x' + y' + z)$$
$$= M_0\ M_1\ M_2\ M_3\ M_6$$
$$= \Pi M(0,1,2,3,6)$$

0 in input column implies true literal
1 in input column implies complemented literal

f' contains all the maxterms not in f

MINTERMS AND MAXTERMS ARE RELATED

- Any minterm m_i is the *complement* of the corresponding maxterm M_i

Minterm	Shorthand	Maxterm	Shorthand
x'y'z'	m_0	x + y + z	M_0
x'y'z	m_1	x + y + z'	M_1
x'yz'	m_2	x + y' + z	M_2
x'yz	m_3	x + y' + z'	M_3
xy'z'	m_4	x' + y + z	M_4
xy'z	m_5	x' + y + z'	M_5
xyz'	m_6	x' + y' + z	M_6
xyz	m_7	x' + y' + z'	M_7

- For example, $m_4' = M_4$ because $(xy'z')' = x' + y + z$

CONVERTING BETWEEN STANDARD FORMS

- We can convert a sum of minterms to a product of maxterms

From before	f	$= \Sigma m(0,1,2,3,6)$	
and	f'	$= \Sigma m(4,5,7)$	
		$= m_4 + m_5 + m_7$	
complementing	$(f')'$	$= (m_4 + m_5 + m_7)'$	
so	f	$= m_4' \, m_5' \, m_7'$	[DeMorgan's law]
		$= M_4 \, M_5 \, M_7$	[By the previous page]
		$= \Pi M(4,5,7)$	

- In general, just replace the minterms with maxterms, using maxterm numbers that don't appear in the sum of minterms:

$$f = \Sigma m(0,1,2,3,6)$$
$$= \Pi M(4,5,7)$$

- The same thing works for converting from a product of maxterms to a sum of minterms

Method II: Karnaugh Maps

A simpler way to handle most (but not all) jobs of <u>manipulating</u> logic functions.

Karnaugh Map Advantages

- Minimization can be done more systematically
- Much simpler to find minimum solutions
- Easier to see what is happening (graphical)

Almost always used instead

of boolean minimization.

Gray Codes

- Gray code is a binary value encoding in which adjacent values only differ by one bit

2-bit Gray Code
00
01
11
10

- Gray Codes
  - To generate a gray code for n+1 bits, write down the gray code sequence for n bits
  - Form one sequence with a prepended '0' to all the code words
  - Form another sequence with a prepended '1' to all the code words
  - Write the latter in reverse order.
  - Concatenate the sequences.

- For example, to generate a 3 bit gray code:
  - Write 00, 01, 11, 10
  - Prepend 0 => 000, 001, 011, 010
  - Prepend 1 => 100, 101, 111, 110
  - Write latter in reverse order => 110, 111, 101, 100
  - Concatenate => 000, 001, 011, 010, 110, 111, 101, 100

Truth Table Adjacencies

F = A'

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

These are *adjacent in a gray code sense* - they differ by 1 bit

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

F = B

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

Same idea:

A'B + AB = B

Key idea:

Gray code adjacency allows use of simplification theorems

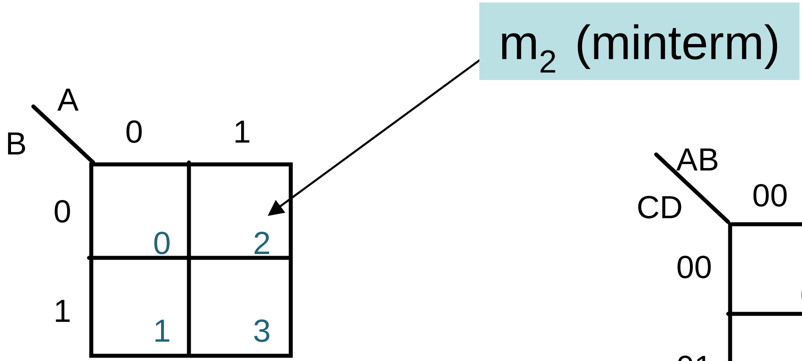
Problem:

Physical adjacency in truth table does not indicate gray code adjacency

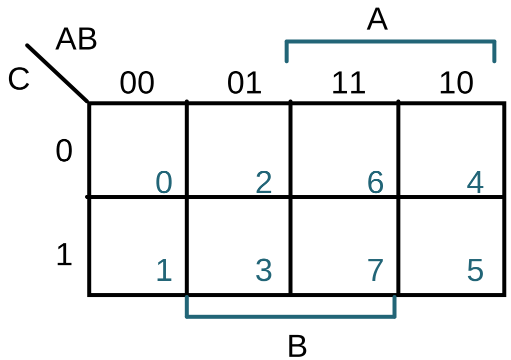
Karnaugh Map Method

K-map is an alternative method of representing the truth table that helps visualize adjacencies in up to 6 dimensions

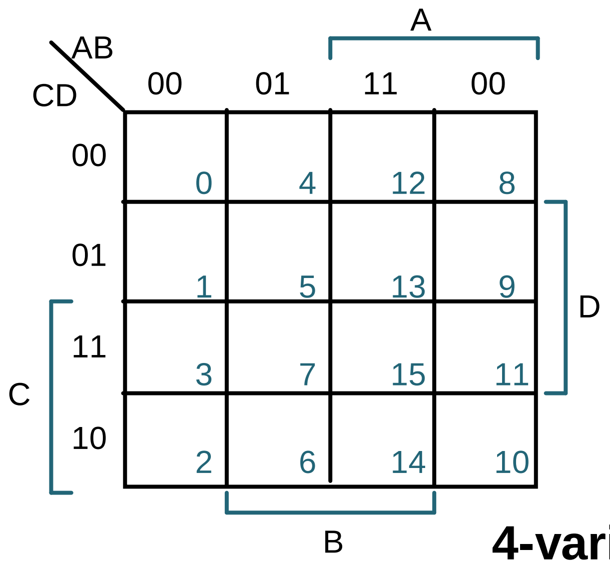
Beyond that, computer-based methods are needed



2-variable K-map



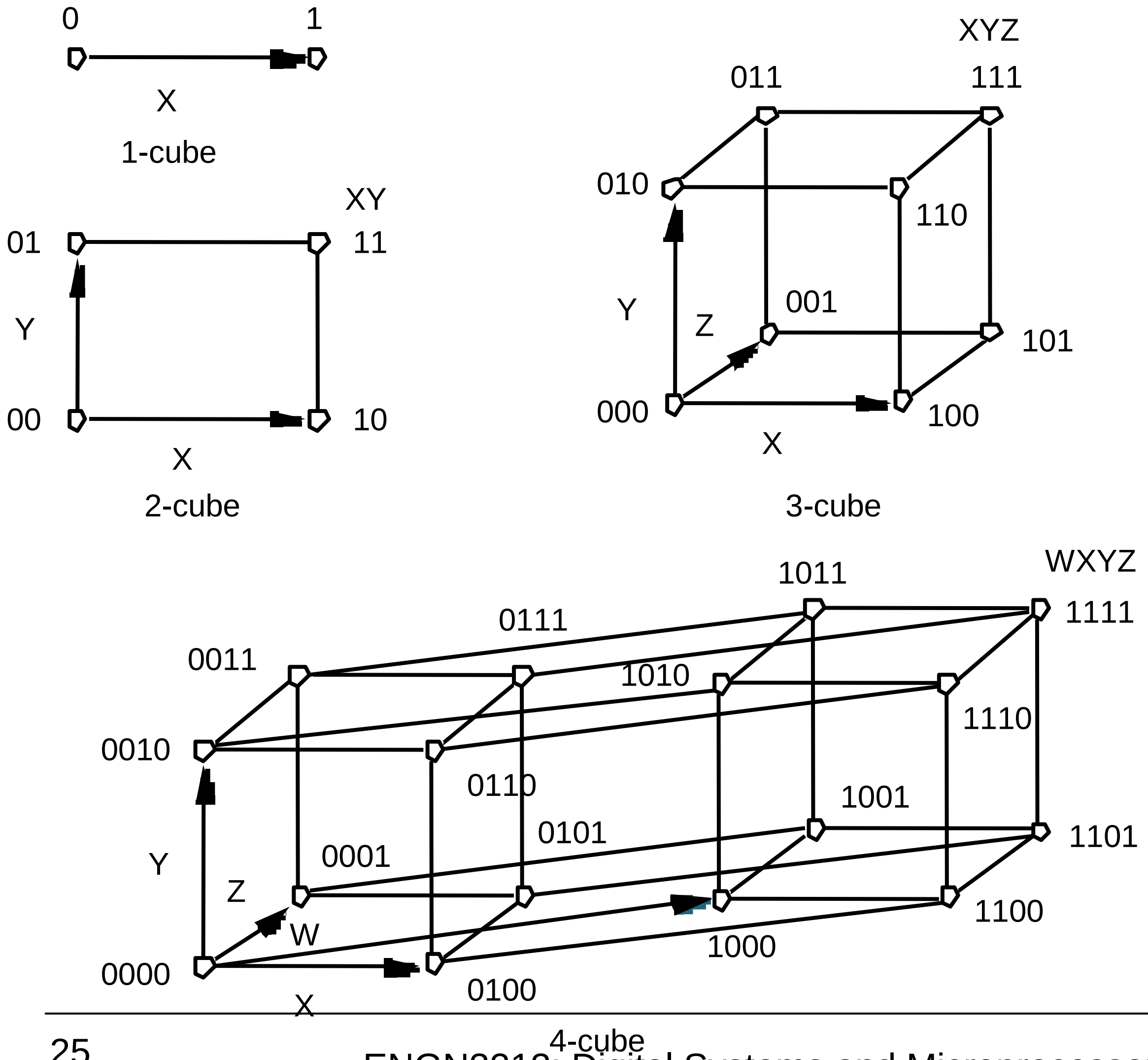
3-variable K-map



4-variable K-map

Numbering Scheme: **00, 01, 11, 10**
Gray Code — only a single bit changes from code word to next code word

Visualizing Boolean Cubes



Just another way to represent the truth table

n input variables = n dimensional "cube"

Adjacencies

- Adjacent squares differ by exactly one variable

BC \ A	0	1
00		AB'C'
01	A'B'C	AB'C
11		ABC
10		ABC'

There is wrap-around:
top and bottom rows are adjacent

2-Variable Karnaugh Map

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

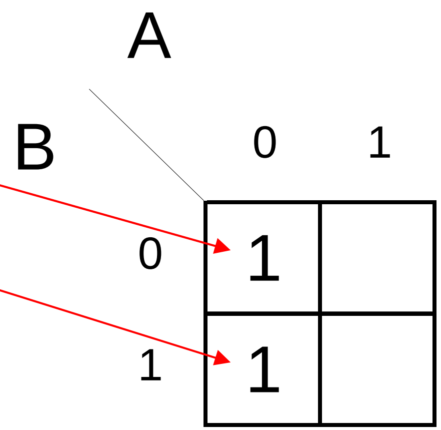
2-Variable Karnaugh Map

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

B \ A	0	1
0		
1		

2-Variable Karnaugh Map

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



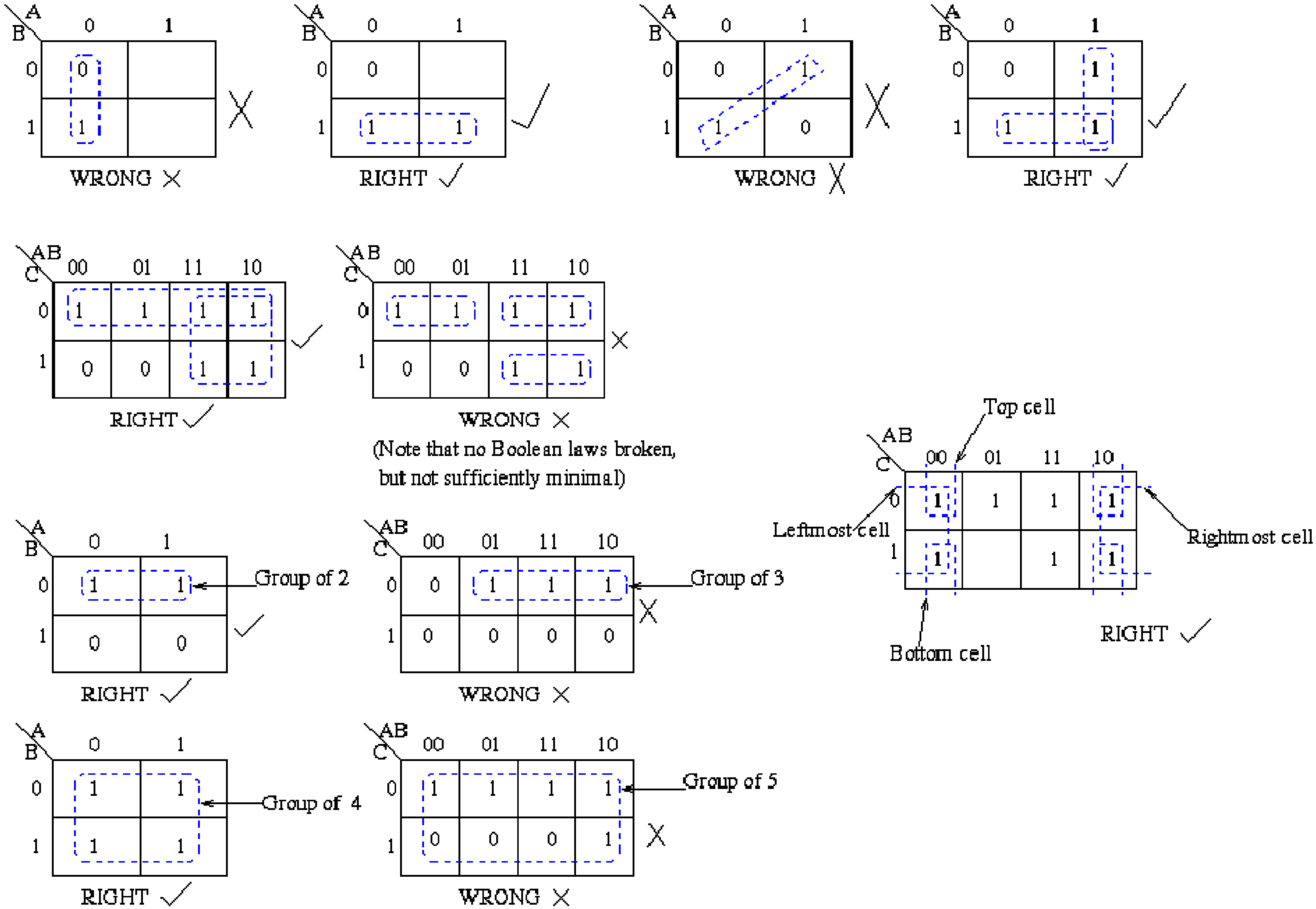
2-Variable Karnaugh Map

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

B \ A	0	1
0	1	0
1	1	0

Rules

- Row and column assignments arranged such that adjacent terms change by only one bit (gray code): use 00,01,11,10 instead of 00,01,10,11
- Each map consists of 2^n cells, where n is the number of logic variables
- 2^n 1's can be circled (group) at a time 1, 2, 4, 8, … OK, 3 not OK
- No zeros allowed.
- No diagonals.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping and "wraps around itself" - i.e. the top and bottom, right and left edges are touching.
- Fewest number of groups possible.


WRONG
RIGHT
WRONG
RIGHT
RIGHT
WRONG
(Note that no Boolean laws broken,
but not sufficiently minimal)
Top cell
Leftmost cell
Rightmost cell
Bottom cell
RIGHT
Group of 2
RIGHT
Group of 3
WRONG
Group of 4
RIGHT
Group of 5
WRONG

2-Variable Karnaugh Map

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

B \ A	0	1
0	1	0
1	1	0

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

Two Level Simplification

Key Tool: The Uniting Theorem — A’ (B' + B) = A’

B \ A	0	1
0	1	0
1	1	0

A asserted, unchanged
B varies

B \ A	0	1
0	1	1
1	0	0

B complemented, unchanged
A varies

F = A’ B' + A’ B = A’ (B' + B) = A’

F = A’

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

G = B'

Essence of Simplification:

find two element subsets of the ON-set where only one variable changes its value.
This single varying variable can be eliminated!

A (columns 11, 10); B (columns 01, 11)

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

F(A,B,C) = A

Another Example

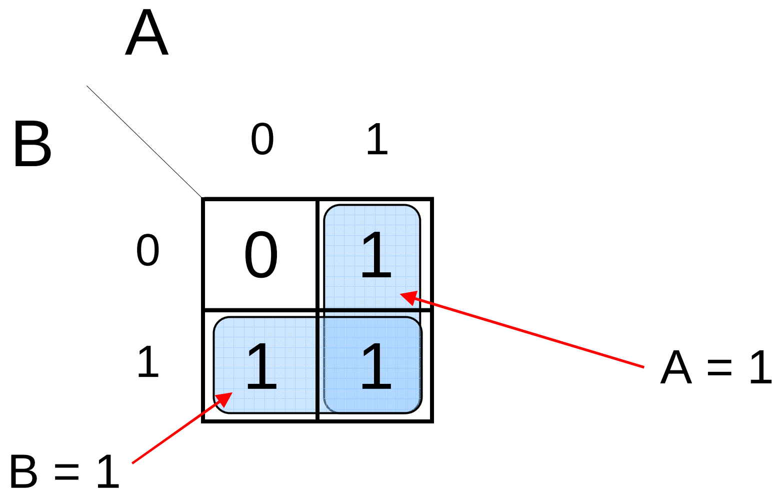
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

B \ A	0	1
0	0	1
1	1	1

$$
\begin{aligned}
\mathbf{F} &= \mathbf{A'B + AB' + AB} \\
&= \mathbf{(A'B + AB) + (AB' + AB)} \\
&= \mathbf{A + B}
\end{aligned}
$$

Another Example

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



F = A + B

Truth Table to Karnaugh Map

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

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

3-Variable Karnaugh Map

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

AB'C+ABC = AC

A'BC+A'BC' = A'B

F = A'B + AC

Another Example

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

A'B'C+A'BC = A'C

AB'C'+ABC' = AC'

$$F = A'C + AC' = A \oplus C$$

Minterm Expansion to K-Map

$$F = \Sigma m(1, 3, 4, 6)$$

BC \ A	0	1
00	m0	m4
01	m1	m5
11	m3	m7
10	m2	m6

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

Minterms are the 1's, everything else is 0

Maxterm Expansion to KMap

$$F = \prod M(0, 2, 5, 7)$$

BC \ A	0	1
00	M0	M4
01	M1	M5
11	M3	M7
10	M2	M6

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

Maxterms are the 0's, everything else is 1

Yet Another Example

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

A'B'C'+AB'C'+A'B'C+AB'C = B'

AB'C+ABC = AC

F = B' + AC

The larger the group of 1's
the simpler the resulting product term

4-Variable Karnaugh Map

CD \ AB	00	01	11	10
00	**m0**	**m4**	**m12**	**m8**
01	**m1**	**m5**	**m13**	**m9**
11	**m3**	**m7**	**m15**	**m11**
10	**m2**	**m6**	**m14**	**m10**

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

AB’C’

D

A’BC

F = A’BC + AB’C’ + D

Note the row
and column orderings.

Required for adjacency

Find a POS Solution

BC’

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

C’D

AB’CD’

F’ = C’D + BC’ + AB’CD’

F = (C+D’)(B’+C)(A’+B+C’+D)

Find solutions to groups of 0’s to find F’

Invert to get F then use DeMorgan’s

Dealing With Don't Cares

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

BC \ A	0	1
00		
01		
11		
10		

Dealing With Don't Cares

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

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

A'B'C+AB'C+A'BC+ABC = C

F = C

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