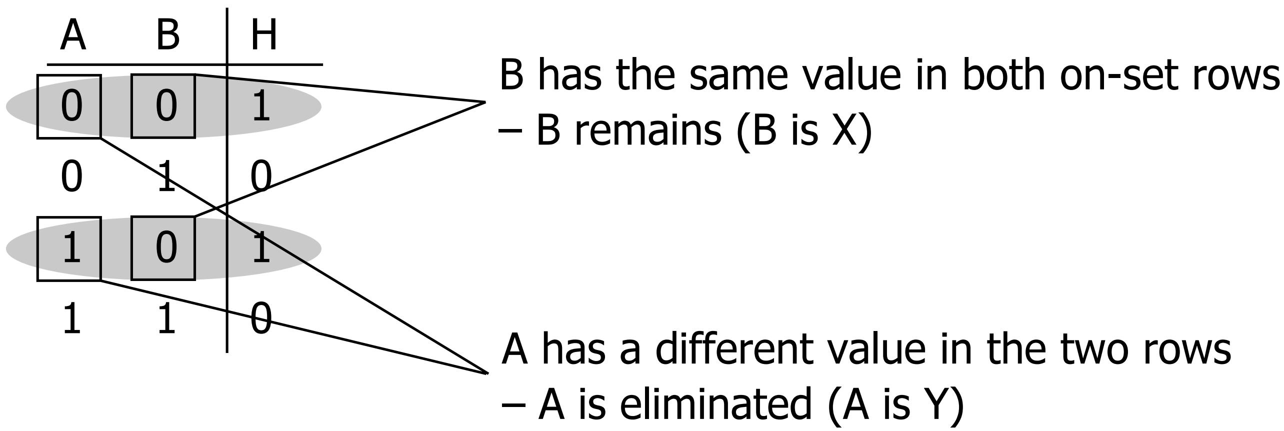
The uniting theorem

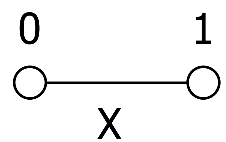
- Key tool to simplification: X Y' + XY = X(Y’+Y) = X(1) = X
- Essence of simplification of two-level logic
  - Find a pair of PoS minterms where only one variable changes its value
  - Merge the two terms into one, and eliminate the changing variable

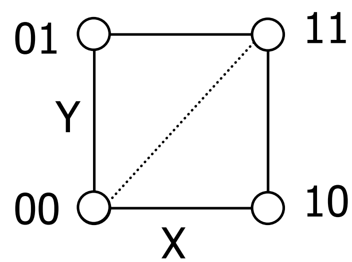


Boolean cubes

- n input variables → one node for each minterm → n^2 nodes
- Edges represent single variable changes between minterms
- A sub-cube is an m-cube, with m<n
- A “face” of a n-cube is an (n-1) sub-cube

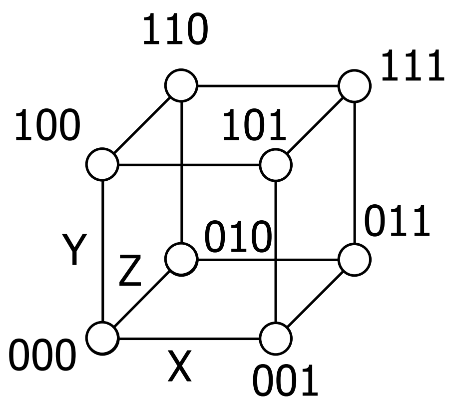
1-cube

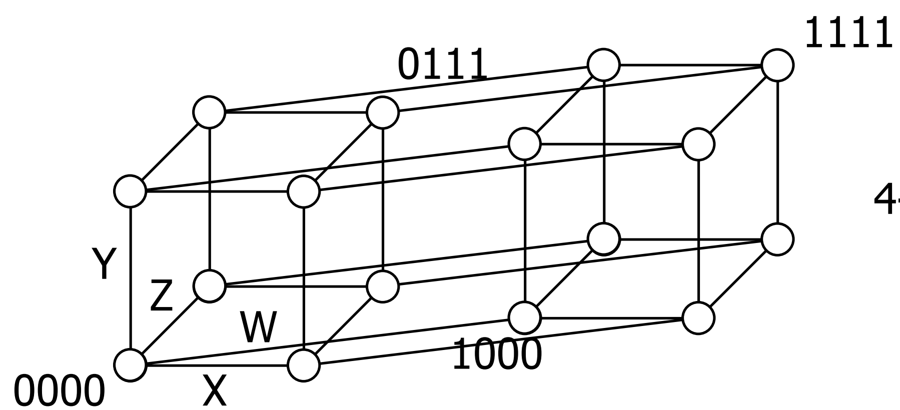




2-cube

3-cube





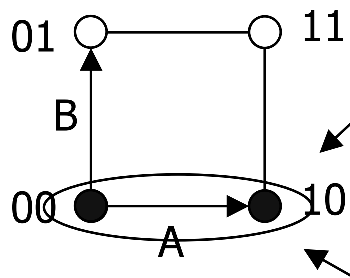
4-cube

Mapping truth tables onto Boolean cubes

- Uniting theorem can be applied to <u>complete</u> faces or sub-cubes of a cube
- <u>Example:</u>

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

F = B′



two faces of size 0 (nodes) combine into a face of size 1(line)

A loop enclosing a sub-cube indicates that a single term is needed to "cover" those minterms

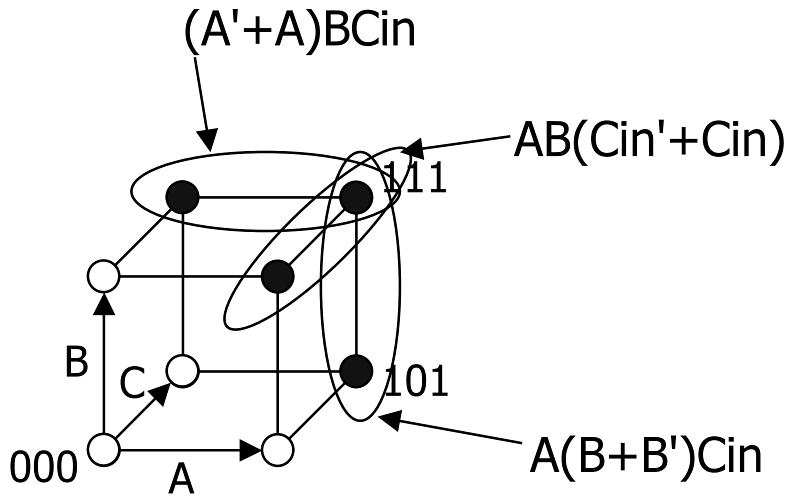
The larger the loop, the fewer literals in the term

ON-set = solid nodes
OFF-set = empty nodes
DC-set = ×'d nodes

Three variable example

- Binary full-adder carry-out logic

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



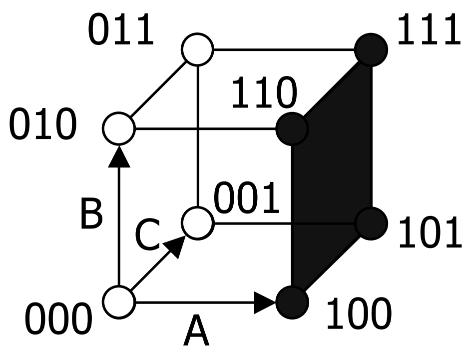
the on-set is completely covered by the combination (OR) of the sub-cubes of lower dimensionality - note that “111” is covered three times

Cout = BCin+AB+ACin

Interpretation of Subcubes (Faces)

- For a 3-cube
  - A node (0-cube) is a 3-literal term (like AB′C′)
  - An edge (1-cube) is a 2-literal term (Like AB′)
  - A face (2-cube) is a 1-literal term (Like A)

F(A,B,C) = Σm(4,5,6,7)



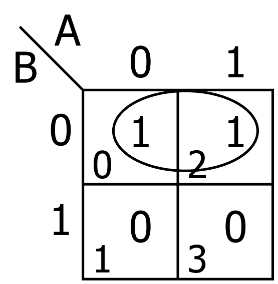
A is asserted (true) and unchanged
B and C vary

- In general
  - an m-subcube within an n-cube (m < n) yields a term with n – m literals

Karnaugh maps

- Flat map of Boolean cube
  - Cells in the table are adjacent if they vary by 1 bit
  - wrap–around at edges
  - hard to draw and visualize for more than 4 dimensions
  - virtually impossible for more than 6 dimensions
- Alternative to truth-tables to help visualize adjacencies
  - guide to applying the uniting theorem
  - on-set elements with only one variable changing value are adjacent unlike the situation in a linear truth-table

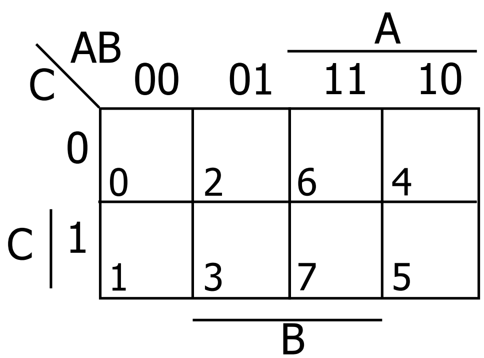
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

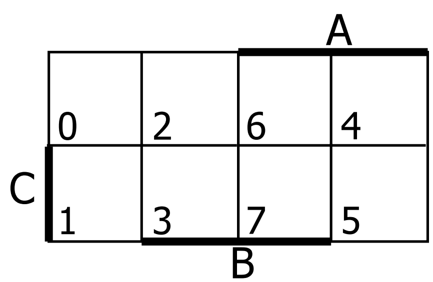


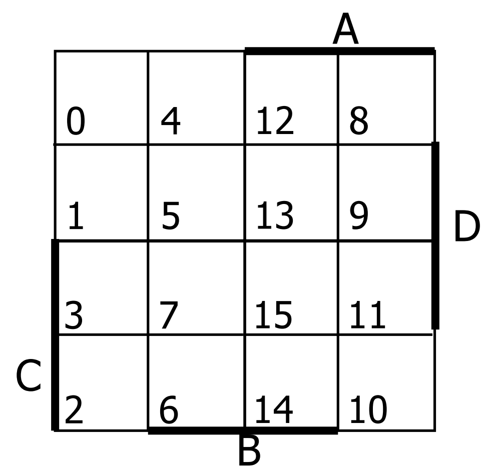
$$F = B'$$

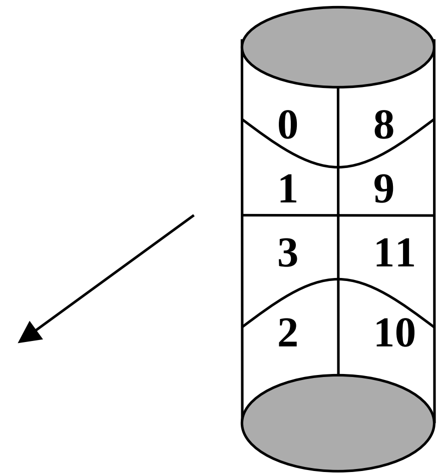
Karnaugh maps (cont'd)

- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10
  - only a single bit changes in code for adjacent map cells including wraparound





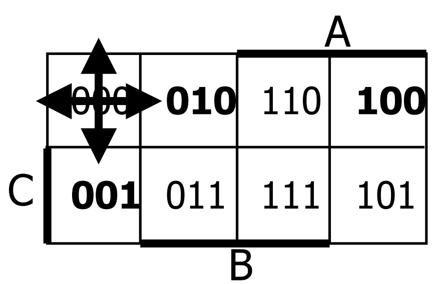


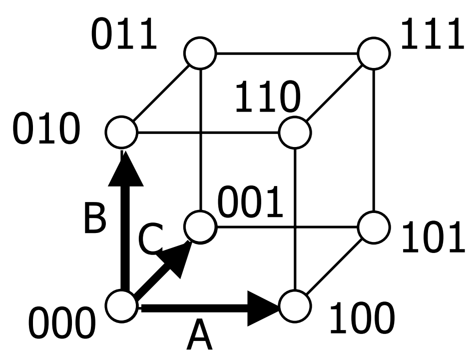


13 = 1101= ABC'D

Adjacencies in Karnaugh maps

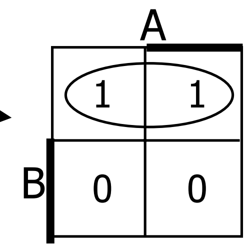
- Wrap from first to last column
- Wrap top row to bottom row



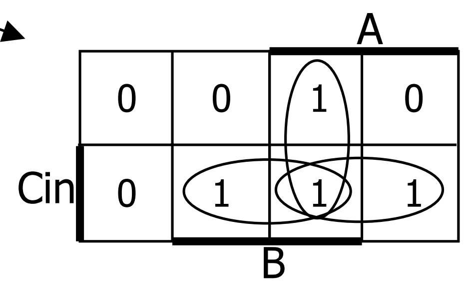


Karnaugh map examples

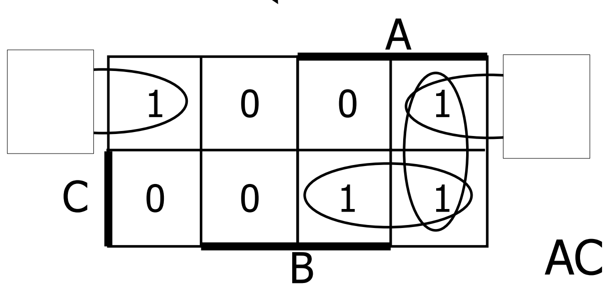
- F =
- Cout =
- f(A,B,C) = Σm(0,4,6,7)



B′



AB+ ACin + BCin

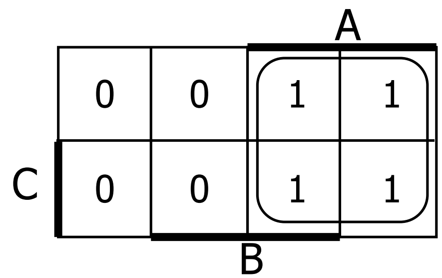


AC + B′C′ + ~~AB′~~

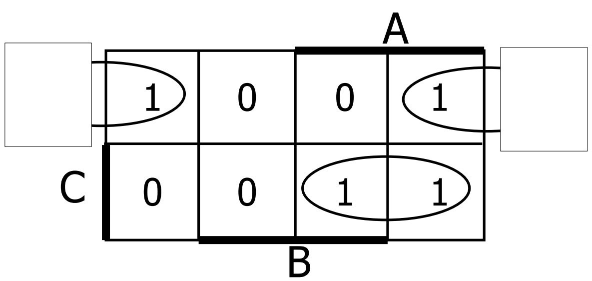
The consensus theorem
AB′ is "covered" by other terms

obtain the complement of the function by covering 0s with sub-cubes

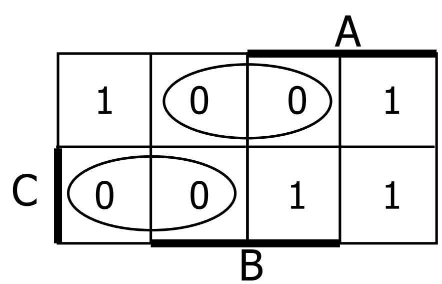
More Karnaugh map examples



G(A,B,C) = A



F(A,B,C) = Σm(0,4,5,7) = AC + B’C’



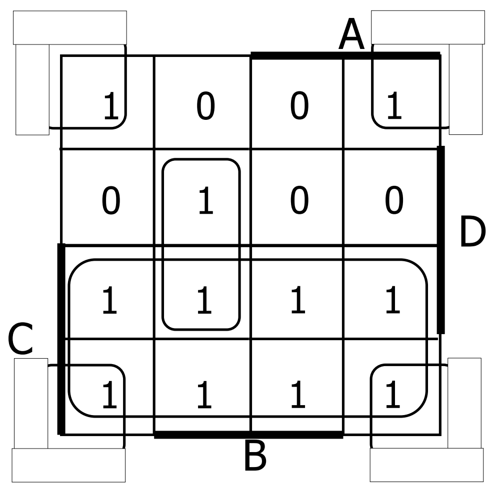
F' simply replace cover the 0’s instead of the 1’s

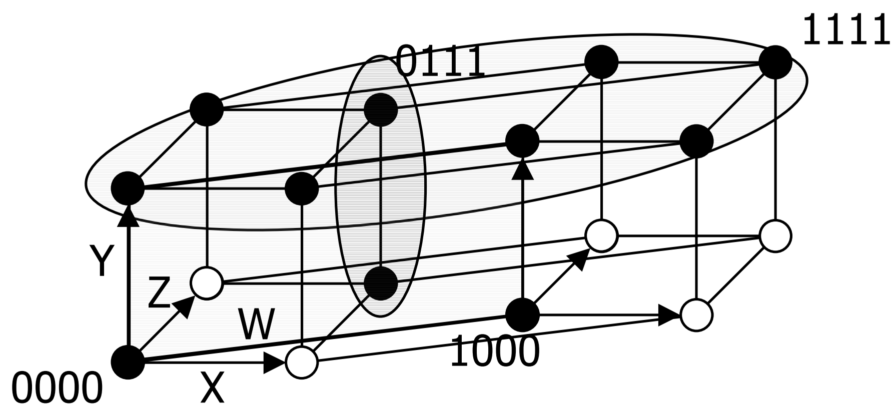
F’(A,B,C) = Σ m(1,2,3,6) = BC’ + A’C

Karnaugh map: 4-variable example

- F(A,B,C,D) = Σm(0,2,3,5,6,7,8,10,11,14,15)

  F = C + A′BD + B′D′

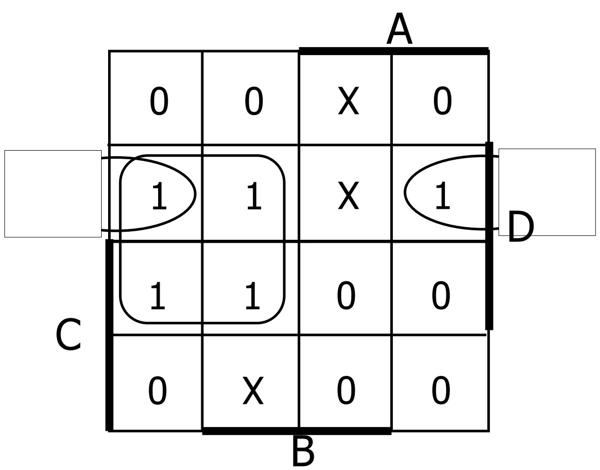




find the smallest number of the largest possible
sub-cubes to cover the ON-set
(fewer terms with fewer inputs per term)

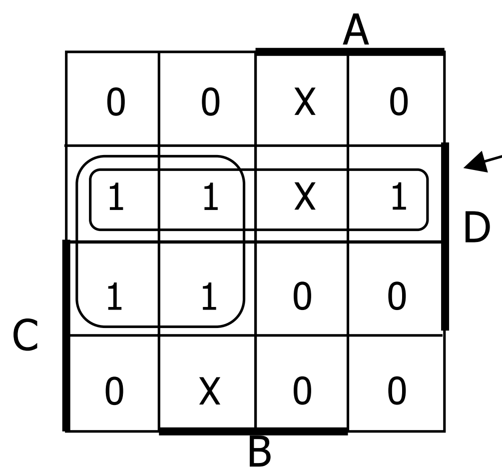
Karnaugh maps: don’t cares

- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - without don't cares (don’t cares become 0)
    - f = A′D + B′C′D



Karnaugh maps: don't cares (cont'd)

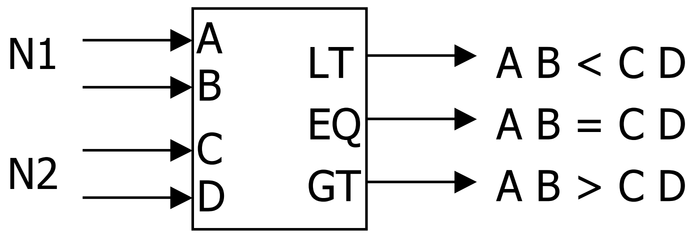
- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - f = A'D + B'C'D — without don't cares
  - f = A'D + C'D — with don't cares



This don't care completes a 2 cube

Don't cares can be treated as 1s or 0s depending on which is more advantageous

Design example: two-bit comparator

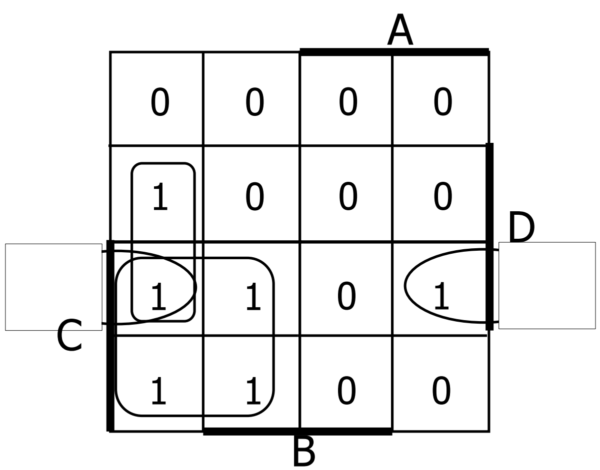


block diagram
and
truth table

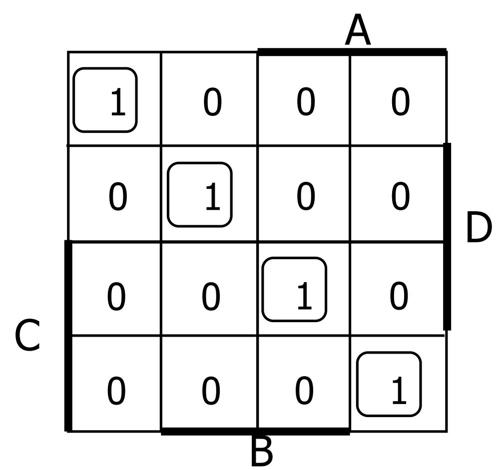
A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

we'll need a 4-variable Karnaugh map
for each of the 3 output functions

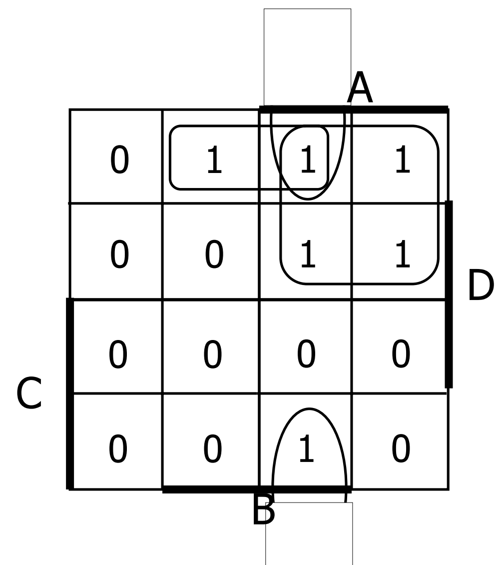
Design example: two-bit comparator (cont'd)



K-map for LT



K-map for EQ



K-map for GT

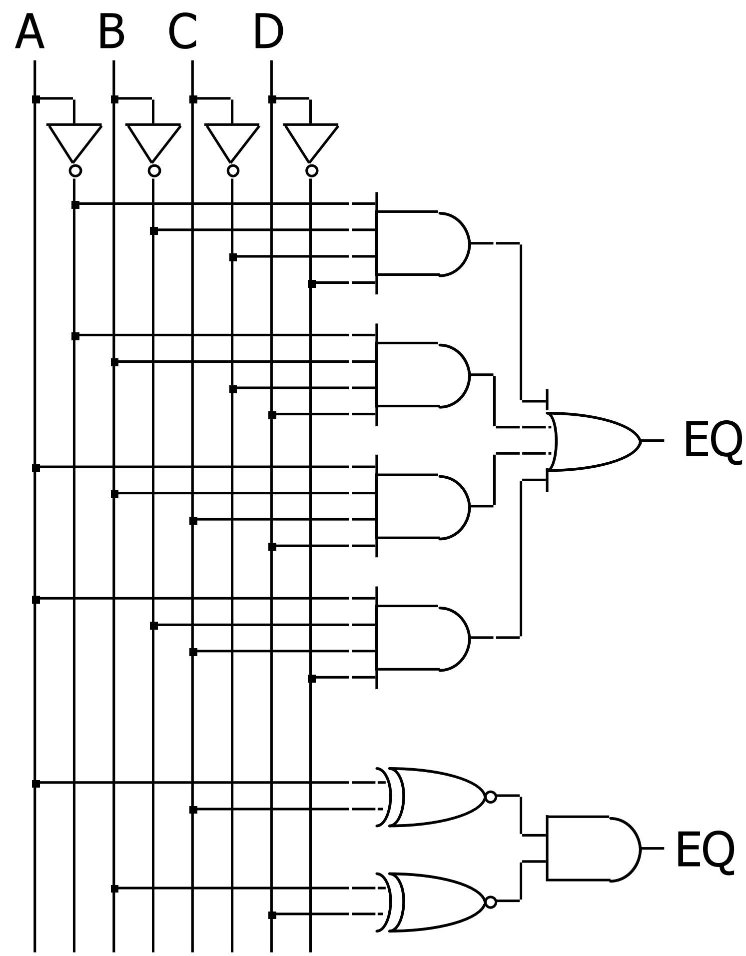
LT = A' B' D + A' C + B' C D

EQ = A' B' C' D' + A' B C' D + A B C D + A B' C D'= (A xnor C) • (B xnor D)

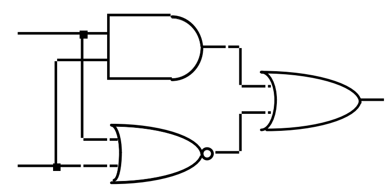
GT = B C' D' + A C' + A B D'

LT and GT are similar (flip A/C and B/D)

Design example: two-bit comparator (cont'd)

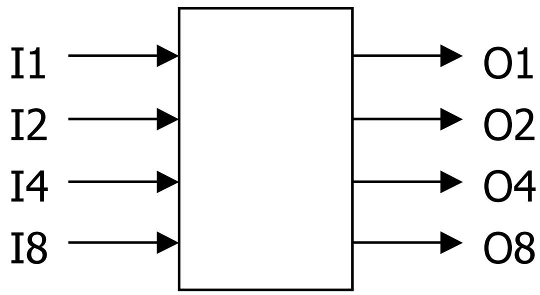


two alternative
implementations of EQ
with and without XNOR



XNOR is implemented with
at least 3 simple gates

Design example: BCD increment by 1

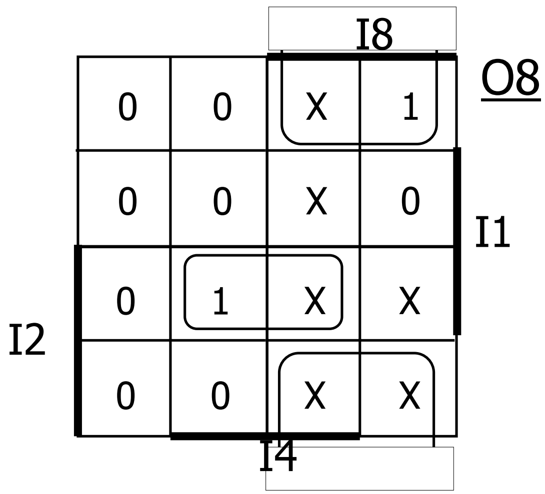


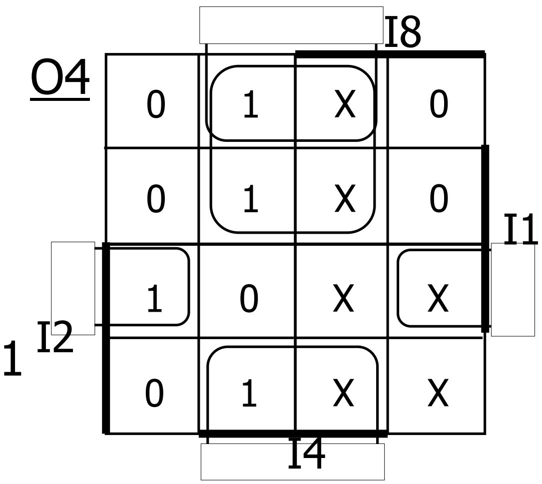
block diagram
and
truth table

I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	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

4-variable K-map for each of
the 4 output functions

Design example: BCD increment by 1 (cont'd)



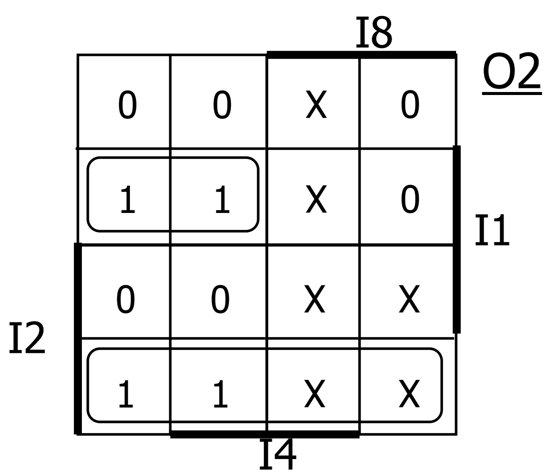


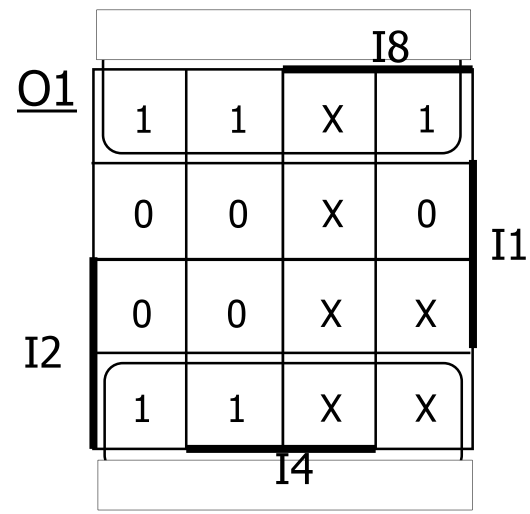
O8 = I4 I2 I1 + I8 I1'

O4 = I4 I2' + I4 I1' + I4' I2 I1

O2 = I8' I2' I1 + I2 I1'

O1 = I1'

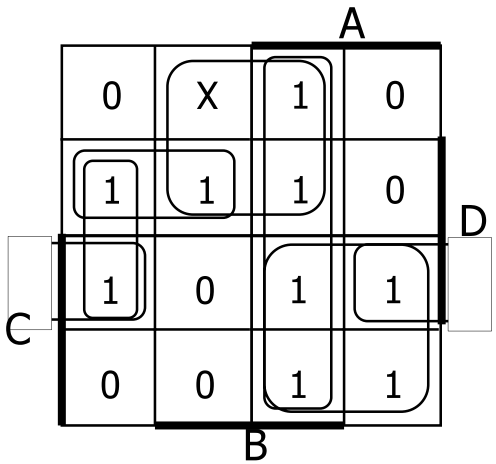




Definition of terms for two-level simplification

- Implicant
  - A subset of the on-set that combine to form a cube
- Prime implicant
  - implicant that can't be combined with another to form a larger subcube
- Essential prime implicant
  - prime implicant is essential if it alone covers an element of the ON-set
  - will participate in ALL possible covers of the ON-set
  - DC-set used to form prime implicants but not to make implicant essential
- Objective:
  - grow implicant into prime implicants
    (minimize literals per term)
  - cover the ON-set with as few prime implicants as possible
    (minimize number of product terms)

Examples to illustrate terms



6 prime implicants:

A'B'D, BC', AC, A'C'D, AB, B'CD

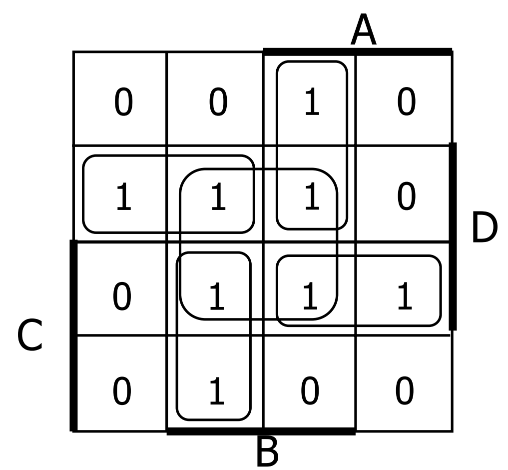
essential

minimum cover: AC + BC' + A'B'D

5 prime implicants:

BD, ABC', ACD, A'BC, A'C'D

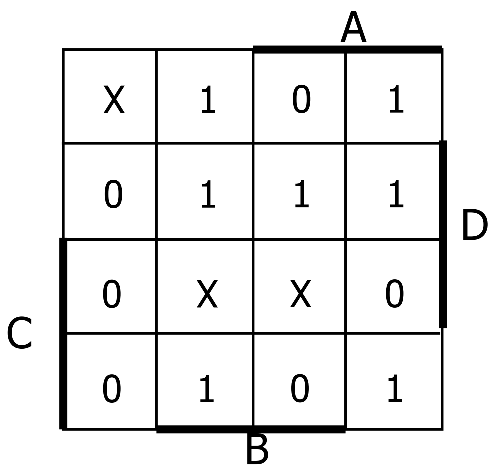
essential

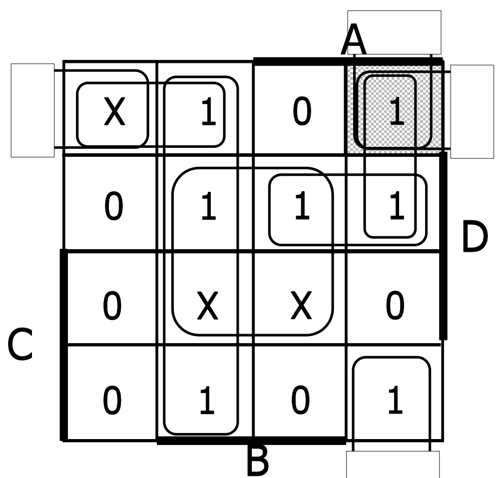


minimum cover: 4 essential implicants

Algorithm for two-level simplification (example)

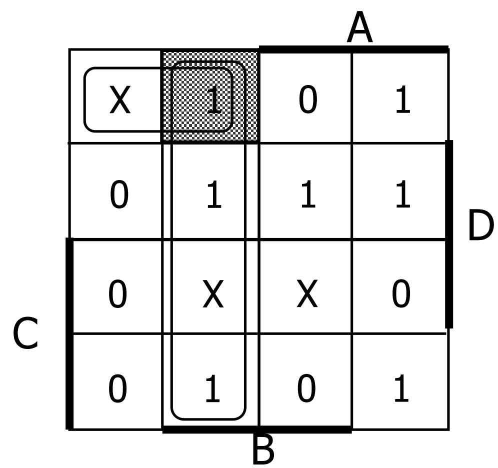
1. Find all Primes



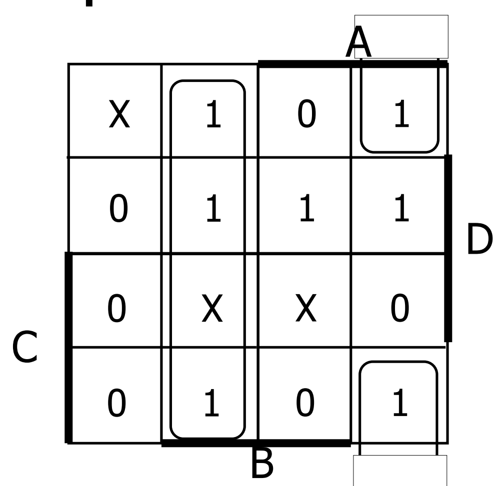


3 primes around AB'C'D'

2. Find Essentials

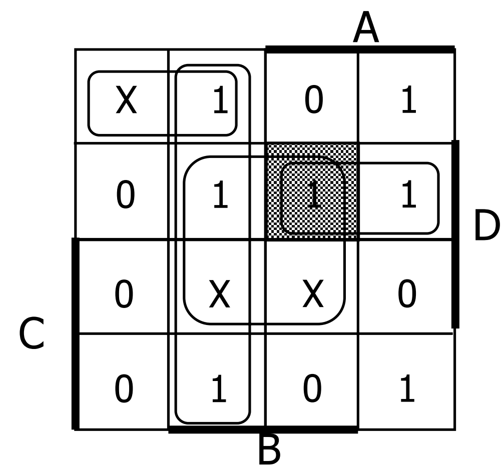


2 primes around A'BC'D'

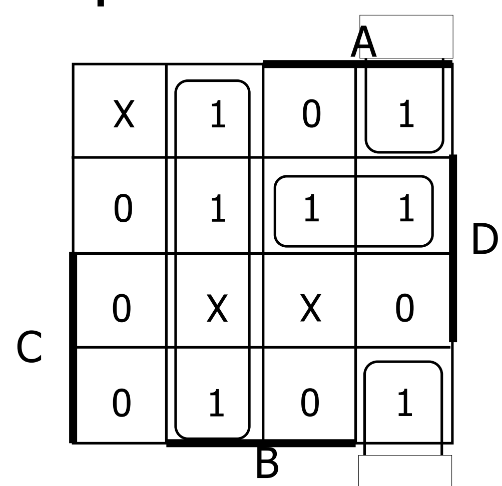


2 essential primes

3. Find a Cover



2 primes around ABC'D

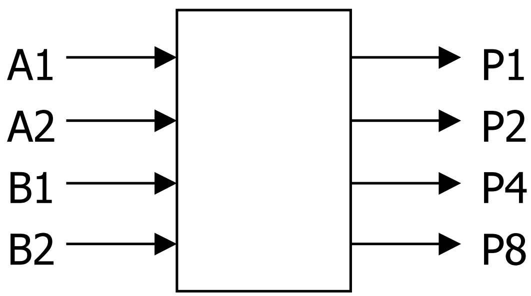


minimum cover (3 primes)

Combinational logic summary

- Logic functions, truth tables, and switches
  - NOT, AND, OR, NAND, NOR, XOR, . . ., minimal set
- Axioms and theorems of Boolean algebra
  - proofs and simplification
- Gate logic
  - networks of Boolean functions and their time behavior
- Canonical forms
  - two-level and incompletely specified functions
- Simplification
  - two-level simplification
- Later
  - multi-level logic
  - design case studies
  - time behavior

Design example: 2x2-bit multiplier

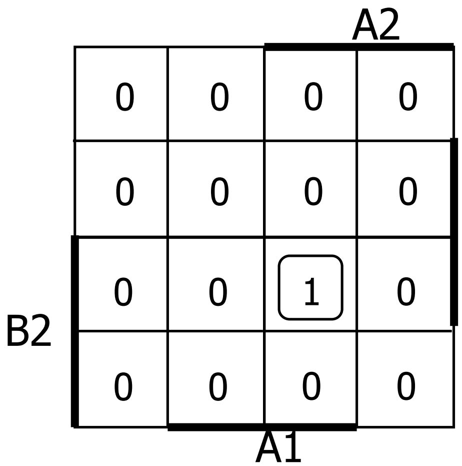


block diagram
and
truth table

A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

4-variable K-map
for each of the 4
output functions

Design example: 2x2-bit multiplier (cont'd)

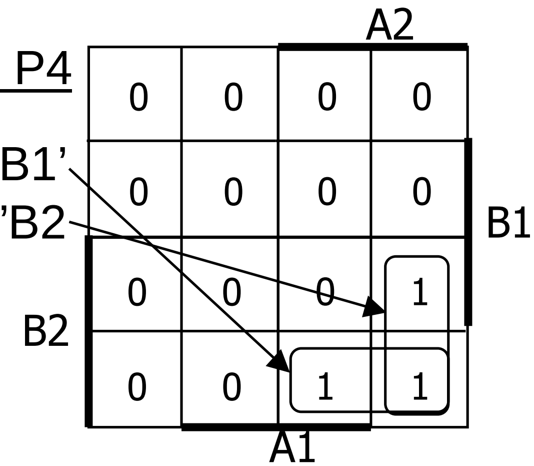


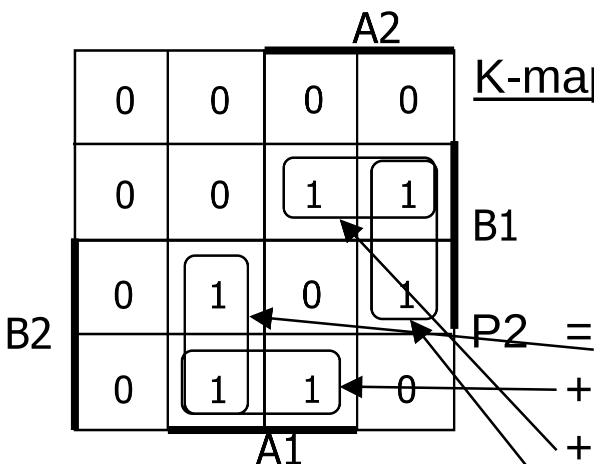
K-map for P8

P8 = A2A1B2B1

K-map for P4

P4 = A2B2B1'
+ A2A1'B2





K-map for P2

P2 = A2'A1B2
+ A1B2B1'
+ A2B2'B1
+ A2A1'B1

K-map for P1

P1 = A1B1