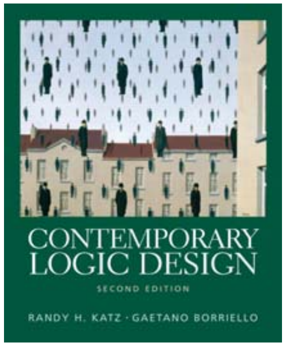
CSE 370 Spring 2006
Introduction to Digital Design

Lecture 7: Karnaugh and Beyond



Last Lecture
- Quiz
- Karnaugh Maps
- K-maps & Minimization

Today
- Design Examples & K-maps
- Minimization Algorithm

Administrivia

- Pick up Quiz 1
  Average: 9.2/10, Median 10/10
- Lab 3 this week (Verilog!)
- Homework 3 on the web

Quiz Review

Problem 1: -5_{10} as a four bit expression using

$5 = 4 + 1 = 101_2 \rightarrow 010_2$

a) sign and magnitude

$-5 = 1\,101_2$ (↑ neg)

b) ones-complement

$-5 = 1010_2$

c) twos-complement   ones comp + 1

$$\begin{array}{r} 1010_2 \\ +\,0001 \\ \hline 1011_2 \end{array}$$

Quiz Review

f=AB+B’C+AC’

a) Truth table

	A B C	AB	B̄C	AC̄	F
0	0 0 0	0	0	0	0 ← (A+B+C)
1	0 0 1	0	1	0	1 ← $\bar{A}\bar{B}C$
2	0 1 0	0	0	0	0 ← $(A+\bar{B}+C)$
3	0 1 1	0	0	0	0 ← $(A+\bar{B}+\bar{C})$
4	1 0 0	0	0	1	1
5	1 0 1	0	1	0	1
6	1 1 0	1	0	1	1
7	1 1 1	1	0	0	1

b) Sum of products

$f = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$

$= \Sigma m(1,4,5,6,7)$

↑ minterms

Quiz Review

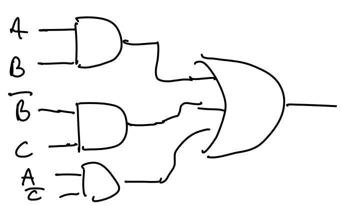
$f = AB + B'C + AC'$

b) Product of Sums

$f = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C})$

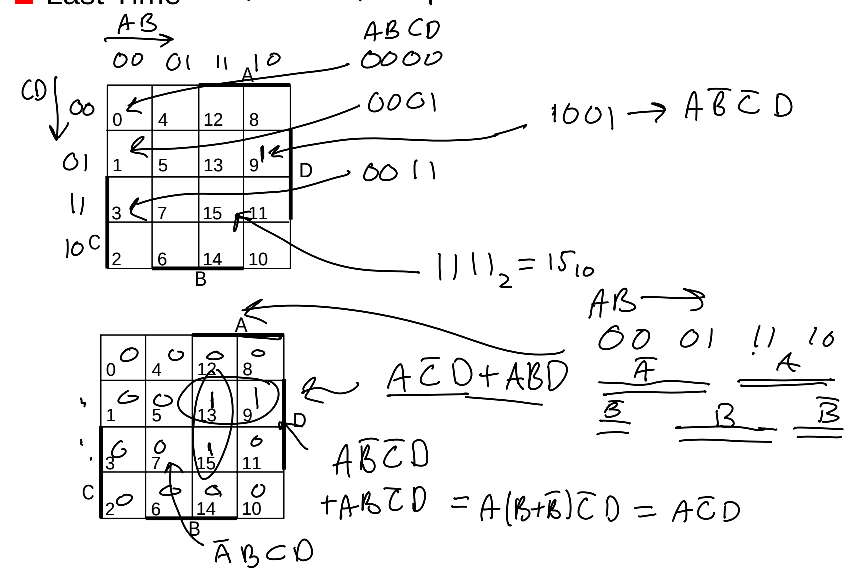
$f = \prod M(0,2,3)$

c) Circuit using AND, OR, NOT



Karnaugh Maps

- Last Time 4 literal K-map



Karnaugh Map Don't Cares

x = do not care

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

0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0

(A, B, C, D labels on map)

Karnaugh Map Don't Cares

- 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

0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0

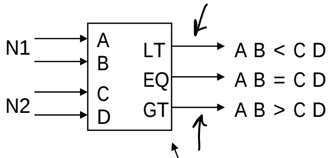
(A, B, C, D labels on map)

C'D

by using don't care as a "1" a 2-cube can be formed rather than a 1-cube to cover this node

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

Design example: two-bit comparator

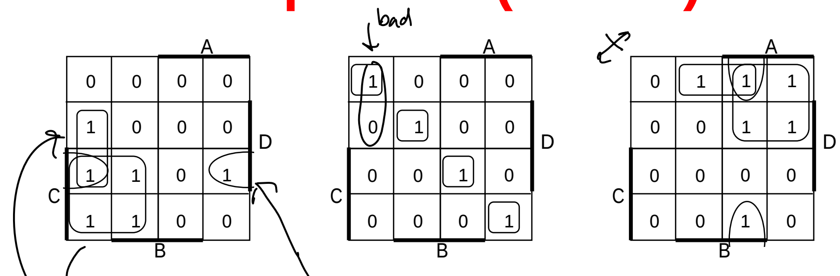


A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

block diagram and truth table

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

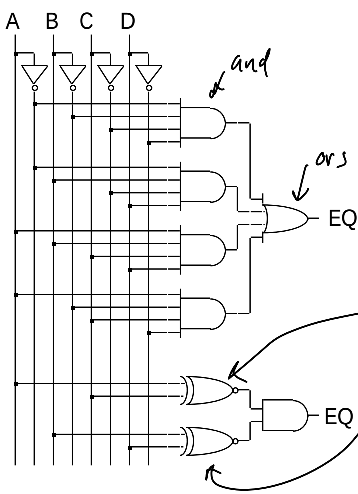
LT = A' B' D + A' C + B' C D

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

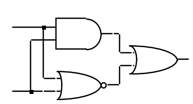
GT = B C' D' + A C' + A B D'

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

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

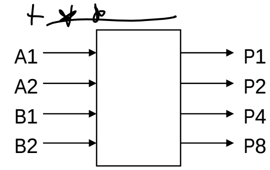


two alternative implementations of EQ with and without XOR



XNOR is implemented with at least 3 simple gates

Design example: 2x2-bit multiplier

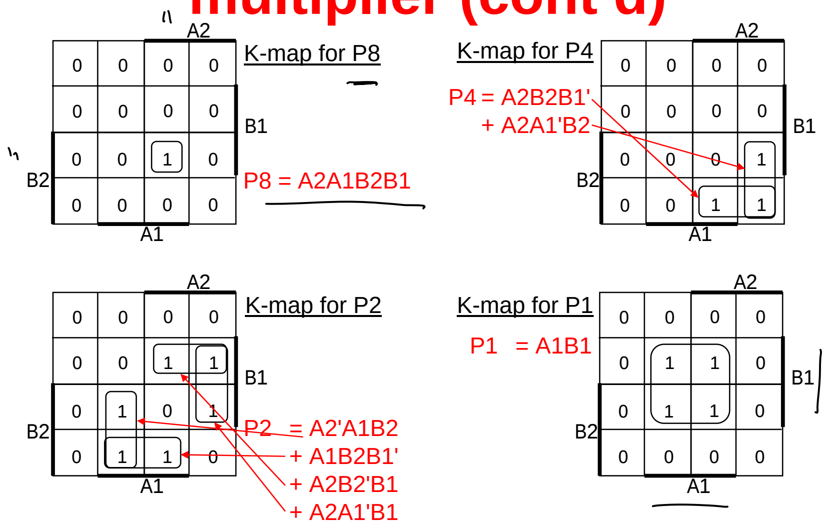


block diagram and truth table

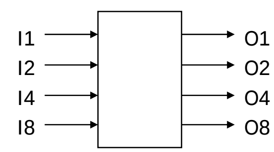
A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

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

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



Design example: BCD increment by 1

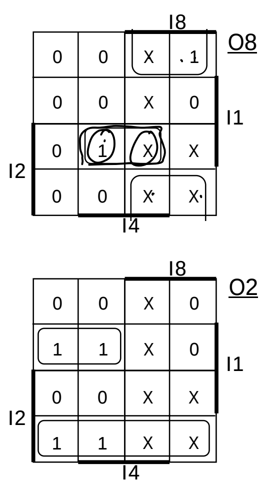


block diagram and truth table

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)

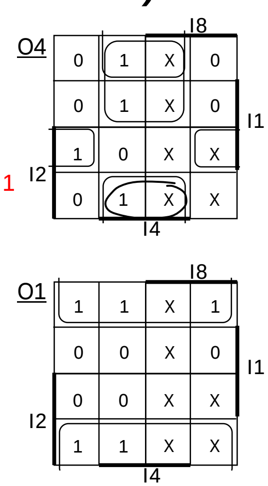


O8 = I4 I2 I1 + I8 I1'

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

O2 = I8' I2' I1 + I2 I1'

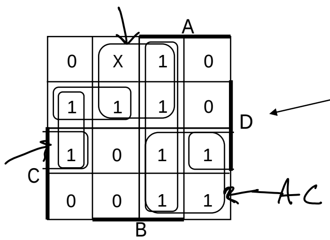
O1 = I1'



Definition of terms for two-level simplification

- Implicant
  - single element of ON-set or DC-set or any group of these elements that can be combined to form a subcube
- 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 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

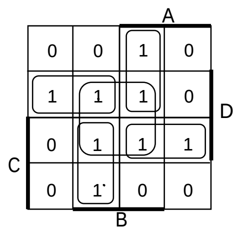
essential

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

5 prime implicants:
BD, ABC', ACD, A'BC, A'C'D

essential

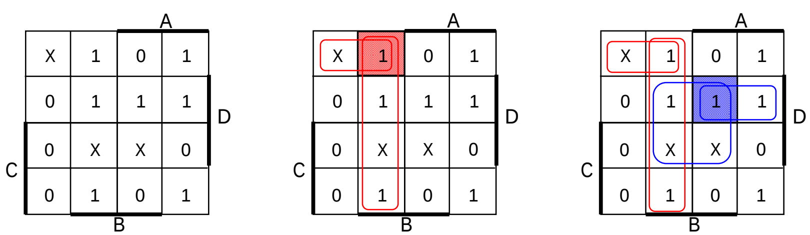
minimum cover: 4 essential implicants



Algorithm for two-level simplification

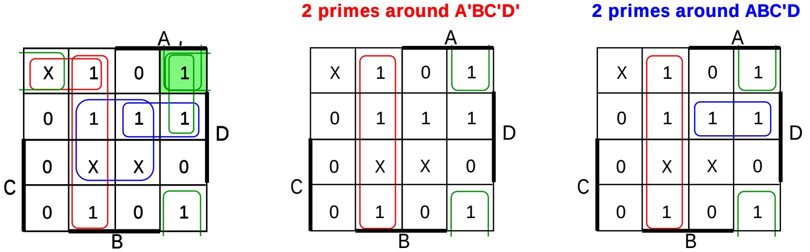
- Algorithm: minimum sum-of-products expression from a Karnaugh map
  - Step 1: choose an element of the ON-set
  - Step 2: find "maximal" groupings of 1s and Xs adjacent to that element
    - consider top/bottom row, left/right column, and corner adjacencies
    - this forms prime implicants (number of elements always a power of 2)
  - Repeat Steps 1 and 2 to find all prime implicants
  - Step 3: revisit the 1s in the K-map
    - if covered by single prime implicant, it is essential, and participates in final cover
    - 1s covered by essential prime implicant do not need to be revisited
  - Step 4: if there remain 1s not covered by essential prime implicants
    - select the smallest number of prime implicants that cover the remaining 1s

Algorithm for two-level simplification (example)



2 primes around A'BC'D'

2 primes around ABC'D



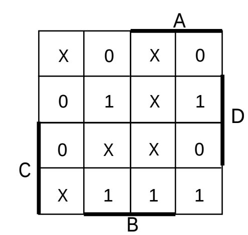
3 primes around AB'C'D'

2 essential primes

minimum cover (3 primes)

Activity

- List all prime implicants for the following K-map:



- Which are essential prime implicants?
- What is the minimum cover?

Loose end: POS minimization using k-maps

- Using k-maps for POS minimization
  - Encircle the zeros in the map
  - Interpret indices complementary to SOP form

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

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

Check using de Morgan's on SOP

F' = BC'D'+B'C'D+ABC'

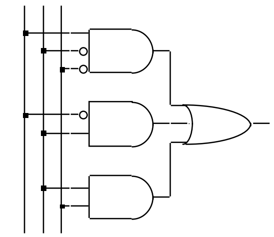
(F')' = (BC'D'+B'C'D+ABC')'

(F')' = (BC'D')'+(B'C'D)'+(ABC')'

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

Implementations of two-level logic

- Sum-of-products
  - AND gates to form product terms (minterms)
  - OR gate to form sum



- Product-of-sums
  - OR gates to form sum terms (maxterms)
  - AND gates to form product