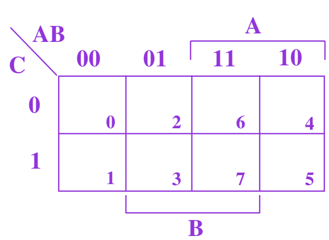
Outline

- Last time:
  - Cube Notation
  - Postulates and Theorems of Boolean Algebra
  - De Morgan's Laws
  - Algebraic Simplification
  - Standard Forms, Min-terms, Short Form
  - Design Example: Simplify algebraically
  - Define implicant, prime implicant, cover, prime cover
- This lecture:
  - Karnaugh Maps
  - Relationship to Cube Notation
  - Design Example: Simplify using Karnaugh Map
  - Multiple-Output Karnaugh Maps
  - "Dont Cares"
  - Completely vs. incompletely specified logic
  - Design Example: Use Karnaugh Map with "don't cares"

Karnaugh Maps

- Hard to draw Boolean N-Cubes beyond four dimensions
- Karnaugh Map (K-map) is useful for visualizing functions up to six dimensions:
  - K-Map consists of 2^*n* squares for an *n-variable function*
  - Each square is labelled or addressed by a *unique* combination of values of the input function (a minterm)
  - Adjacent squares differ by one literal only; these squares are called *adjacencies*.

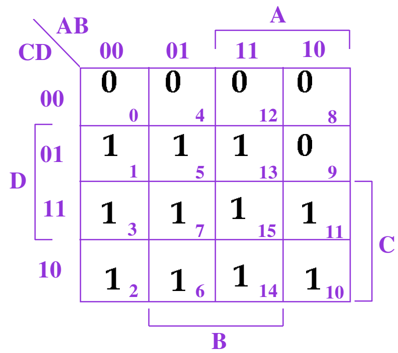


Three-variable map

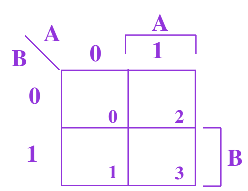
Design Example: Represent Using K-Map

Hot-tub example: T = BAD + C + DA'

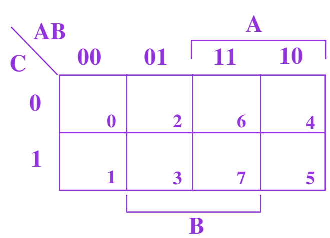
A B C D	T	
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	1	2
0 0 1 1	1	3
0 1 0 0	0	4
0 1 0 1	1	5
0 1 1 0	1	6
0 1 1 1	1	7
1 0 0 0	0	8
1 0 0 1	0	9
1 0 1 0	1	10
1 0 1 1	1	11
1 1 0 0	0	12
1 1 0 1	1	13
1 1 1 0	1	14
1 1 1 1	1	15



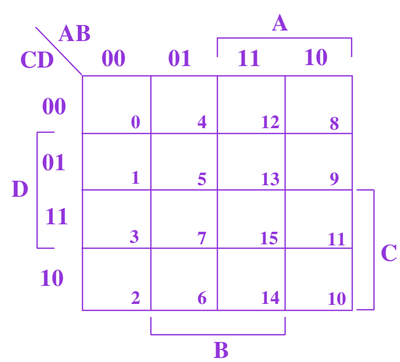
Karnaugh Maps



Two-variable map

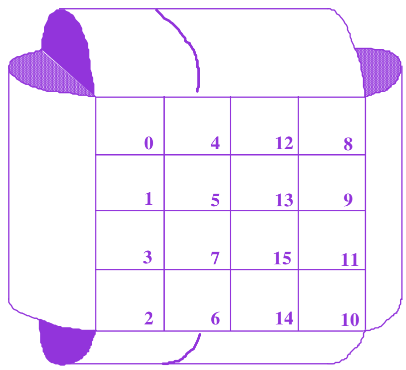


Three-variable map



Four-variable map

The K-Map Really Folds on Itself in X and Y



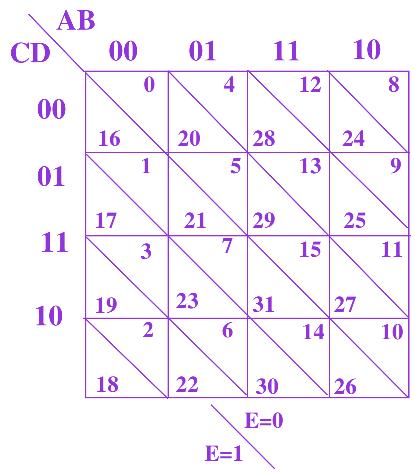
The Gray Code

- ❍ Symmetric way of generating code where adjacent values differ in only one variable.
- ❍ Use the "mirror" approach to generate the Gray code.
- ❍ *Unit-distance* Gray code useful for error detection in a serial process

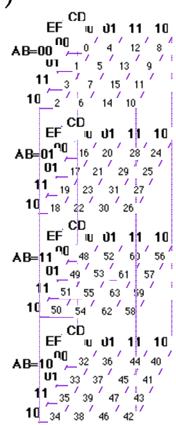
e.g. Water level indicator, wind direction indicator.

A	B	C	D	
0	0	0	0	0
0	0	0	1	1
0	0	1	1	3
0	0	1	0	2
0	1	1	0	6
0	1	1	1	7
0	1	0	1	5
0	1	0	0	4
1	1	0	0	12
1	1	0	1	13
1	1	1	1	15
1	1	1	0	14
1	0	1	1	11
1	0	1	0	10
1	0	0	1	9
1	0	0	0	8

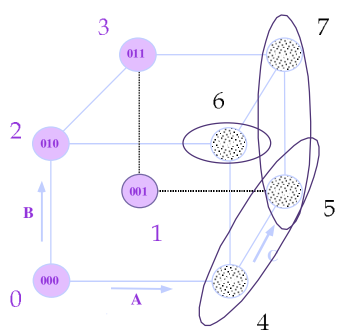
Five & Six-Variable K-Map

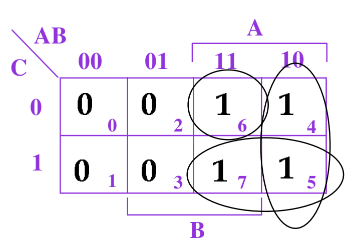


- Can get to six variables by using four enties/square or by going to stacked maps (3-D)



Simplification Using K-Maps





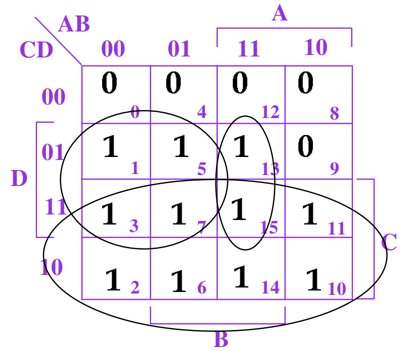
Alternate format:

A	B	C	f
1	1	0	1
1	0	*	1
1	*	1	1

Design Example: Simplification Using K-Maps

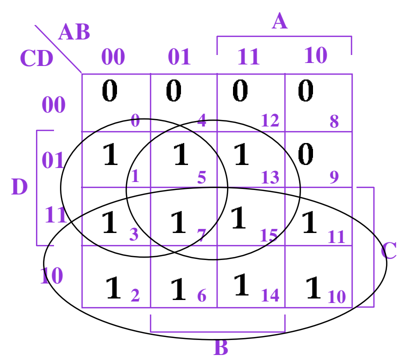
Hot-tub example: T = BAD + C + DA'

A B C D	T	
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	1	2
0 0 1 1	1	3
0 1 0 0	0	4
0 1 0 1	1	5
0 1 1 0	1	6
0 1 1 1	1	7
1 0 0 0	0	8
1 0 0 1	0	9
1 0 1 0	1	10
1 0 1 1	1	11
1 1 0 0	0	12
1 1 0 1	1	13
1 1 1 0	1	14
1 1 1 1	1	15



Design Example: Simplification Using K-Maps

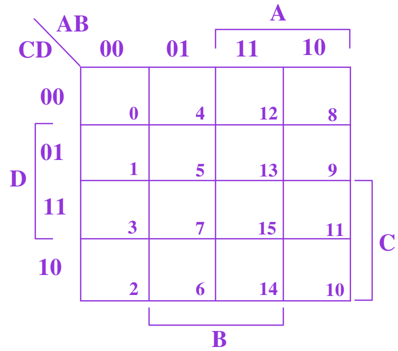
- **Hot-tub example: T =**



Design Example: Multiple Output & Don't Cares

A	B	C	D	Z	
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0		10
1	0	1	1		11
1	1	0	0		12
1	1	0	1		13
1	1	1	0		14
1	1	1	1		15

- Design a minimum two-level network (minimum number of gates and gate inputs) that accepts a binary number and delivers 10 unique outputs (Binary-to-Decimal Converter)
- 10 outputs => 4 binary variables



Forming Minimum Sum-of-Products from K-Map

Step 1: Choose an element of ON-set not already covered by an implicant

Step 2: Find "maximal" groupings of 1's and X's adjacent to that element. Remember to consider top/bottom row, left/right column, adjacencies. (always a power of 2 number of elements).

Repeat Steps 1 and 2 to find all prime implicants

Step 3: Revist the 1's elements in the K-map. If covered by single prime implicant, it is *essential*, and participates in final cover. The 1's it covers do not need to be revisited

Step 4: If there remain 1's not covered by essential prime implicants, then select the smallest number of prime implicants that cover the remaining 1's