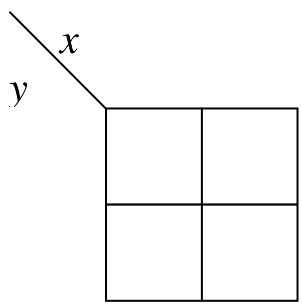
Chapter 3: Karnaugh Maps [Pronounced "Kar-nof"]

Karnaugh map (K-maps) is a graphical method for "plotting" the minterms or maxterms of a Boolean function.

3.1: Introduction to K-Map

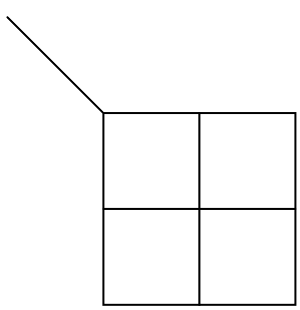
- K-maps consist of 2, 4, 8, … 2^n squares, one for each minterm of a function
- Location of each minterm is pre-defined with a given arrangement
- Two adjacent squares (horizontal or vertical, not diagonal) are neighbors

Two-Variable K-Map: $F(x,y)$ – x is first, y is last. Order matters!!!

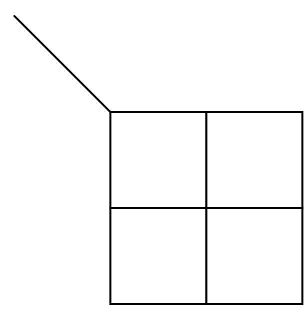


Plotting functions in 2-variable K-map:

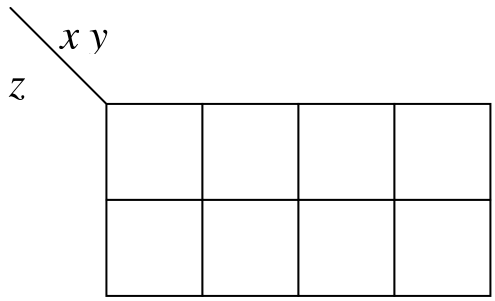
Ex1: $F(a,b) = a'b' + a\,b$

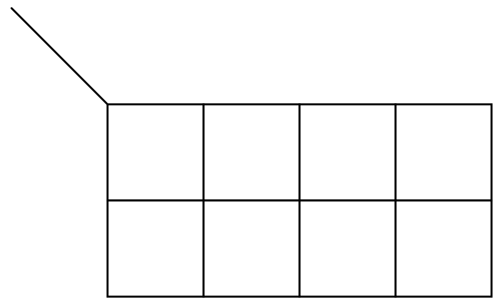


Ex2: $G(A,B) = \Sigma m(1,3)$

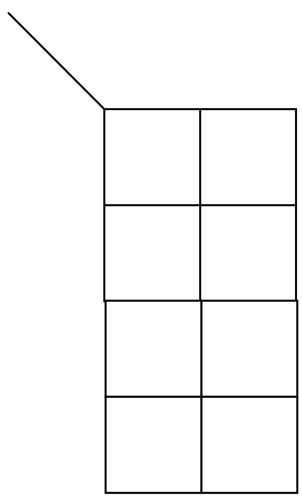


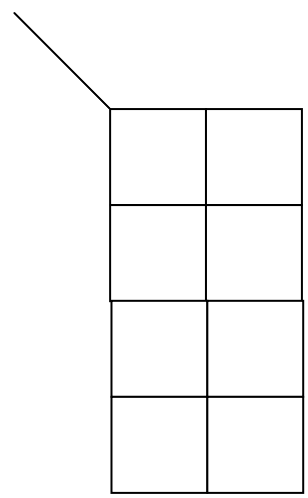
Three-Variable K-Map: $F(x,y,z)$ – x is first, z is last. Order matters!!!



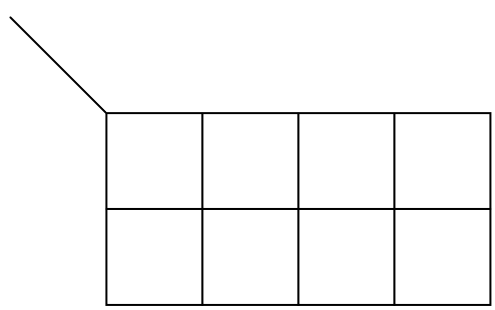


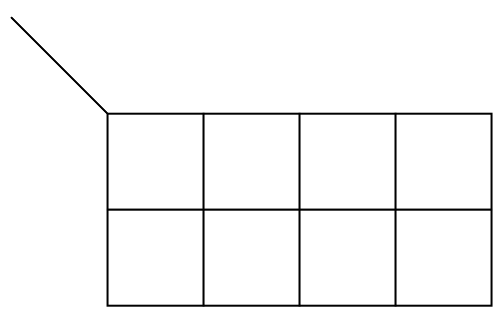
A 3-variable K-map can also be arranged vertically (see p. 116)



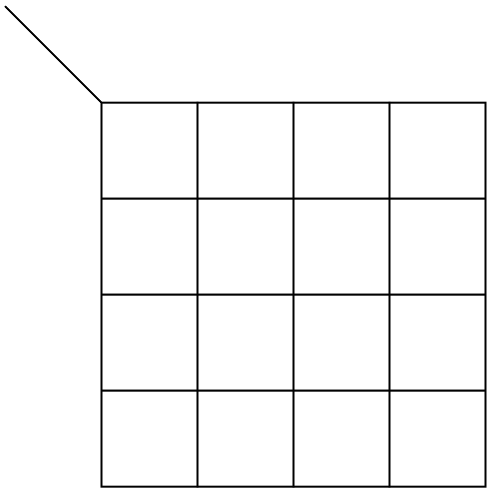


Some example product terms:

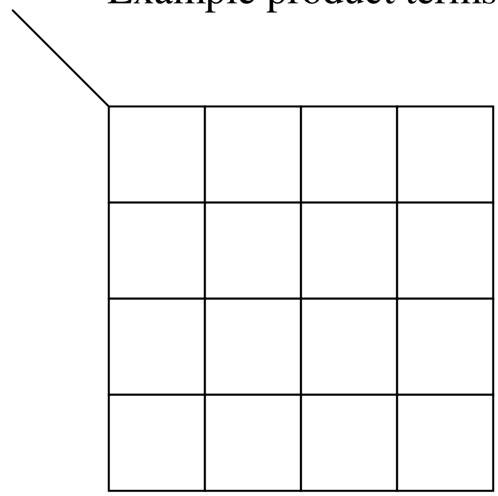




Four-Variable K-Map



Example product terms

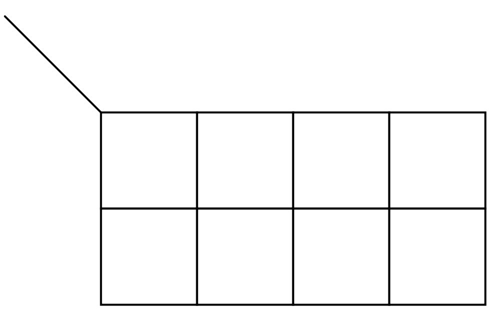


3.2: Function Simplification using K-Map Method

Goal: Given function (algebraic or minterm list) → Simplify it in SOP form

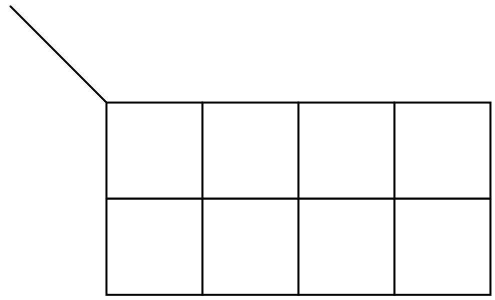
1. Plot its K-map → Place all minterms and don't cares in K-map.
2. Find all ***essential prime implicants*** → Make groups of 1, 2, 4, 8, 16, etc, to "cover" the minterms. Groupings must be as large as possible.
3. Write the minimum (simplified) SOP form → Use the **fewest** number of product terms to define the function in SOP form [it may be possible to have more than minimum solution].

Ex1: Simplify $F(A,B,C) = A\,B' + A\,B + B'C'$ in SOP form

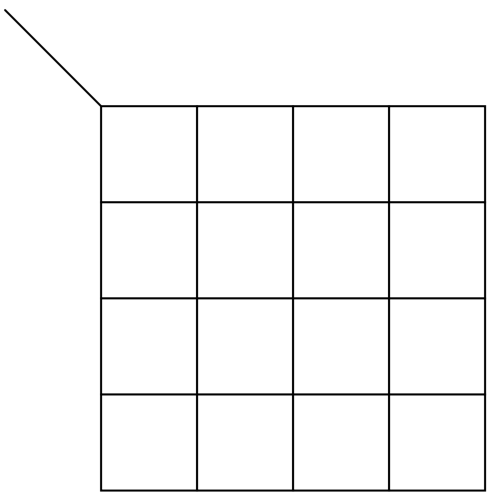


Shortcut: You can skip the minterm list, and go directly from algebraic function to K-map.

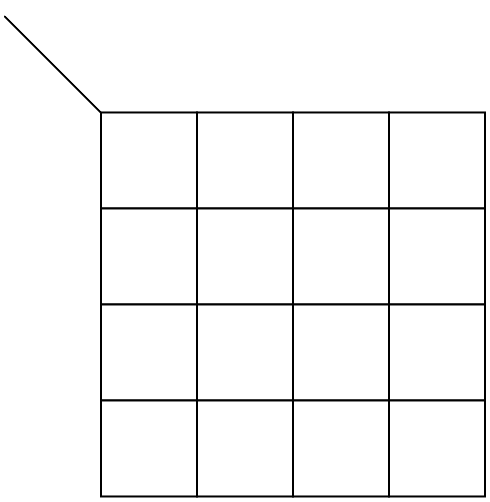
$F(A,B,C) = A\,B' + A\,B + B'C'$



Ex2, pg. 126: Simplify $F(a,b,c) = a'b'c' + a'b\,c' + a'b\,c + a\,b'c'$ in SOP form

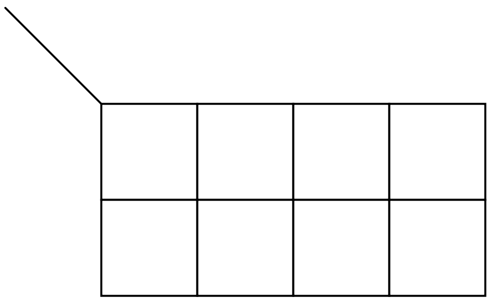


Ex3, pg. 127: Simplify $F(a,b,c,d) = \Sigma m(0,2,4,6,7,8,9,11,12,14)$ in SOP form

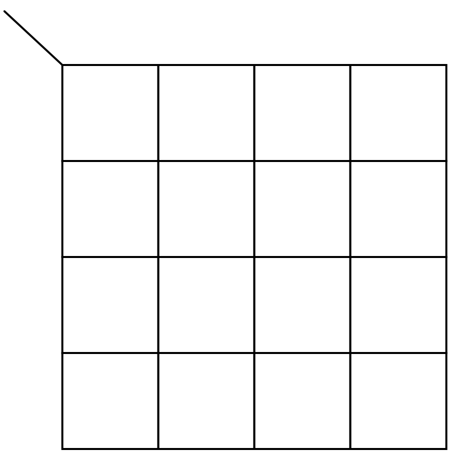


Sometimes, more than one minimum (“simplest”) solution exists.

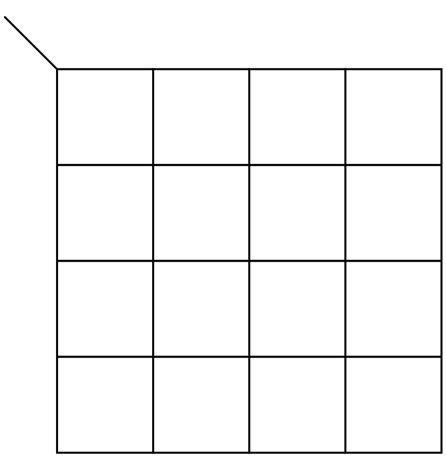
Ex4, pg. 128: Simplify $G(x,y,z) = x'y\,z' + x'y\,z + x\,y'z' + x\,y'\,z + x\,y\,z$ in SOP form



Ex5, pg. 128: Simplify $G(w,x,y,z) = \Sigma m(2,5,6,7,9,10,11,13,15)$ in SOP form



Ex6, pg. 129: Simplify $H(A,B,C,D) = A'BC' + A'CD + ABC + AC'D + BD$



Look at textbook examples 3.12, 3.13, 3.14, 3.15, 3.16 (pages 130-134).

3.3: Incompletely Specified Functions

In some circuits, the value of the output is specified only for some input conditions, and don't care about the remaining conditions. *Don't cares* are represented by "**x**" in truth tables and K-maps (Note: some textbooks use "**d**").

Ex: A circuit has 3 inputs (x_1, x_2, x_3) and one output $\boldsymbol{Z}$. The inputs represent a 3-bit binary number between 1 and 6 (assume that zero and seven never appears at the input). The output is to be **1** if the input number is divisible by 3. Show the minterm list, truth table, and plot its corresponding K-map.

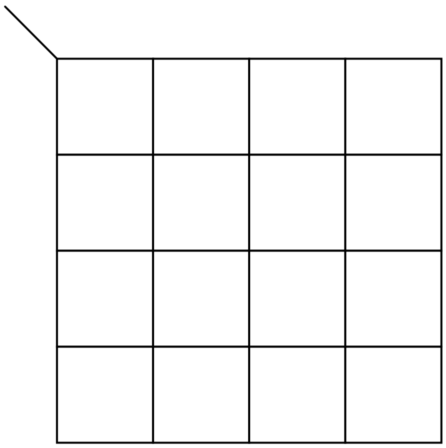
x_1	x_2	x_3	Z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Simplification of Functions that Include Don't Cares

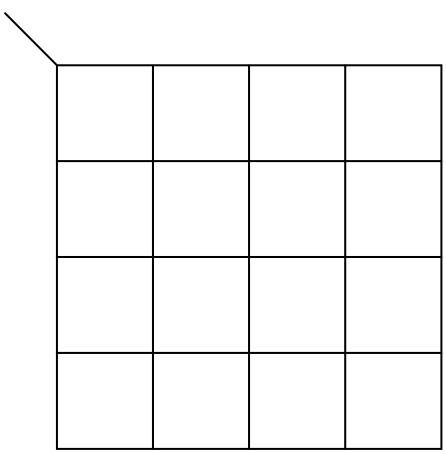
Use same technique previously described. However, treat don't cares as 1's **if** that helps make larger groupings.

Ex: Simplify the function described above: (i) Do not use the *don't cares*; (ii) Do use the *don't cares*. Which function is simpler?

Ex 3.17 (pg. 135). Simplify $F(A,B,C,D)=\Sigma m(1,7,10,11,13)+\Sigma d(5,8,15)$ in SOP.
[Only one minimum solution]



Ex 3.19 (pg. 137). Simplify $F(a,b,c,d) = \Sigma m(0,2,3,4,8,10,12\text{-}15)+\Sigma d(7,9)$ in SOP.
[Several minimum solutions]

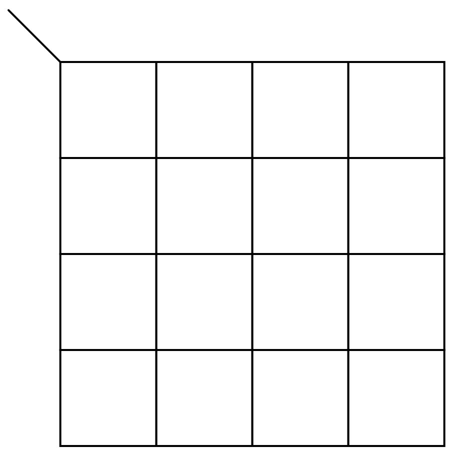


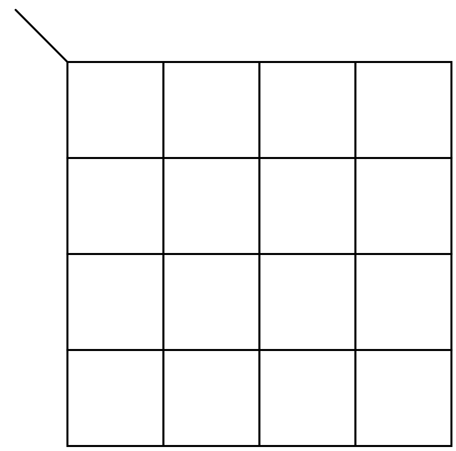
3.4: Minimum Product of Sum (POS) Solutions

Given a function ***F*** (algebraic or minterm list) → Simplify it in POS form
1. Plot the **zeros** of the function in a K-map. Don't cares remain unchanged.
2. Group the **zeros** as if they were minterms. Write each product term with a prime, and apply DeMorgan's property.
3. Write the minimum (simplified) POS form of the function.

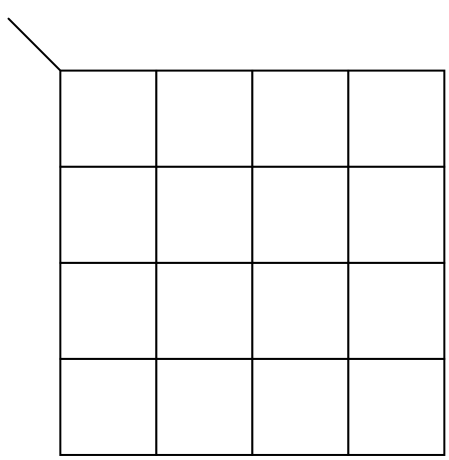
Procedure is different from textbook, but produces same result.

Ex 3.21 (pg. 139). Simplify $F(a,b,c,d) = \Sigma m(0,1,4,5,10,11,14)$ in SOP and POS.

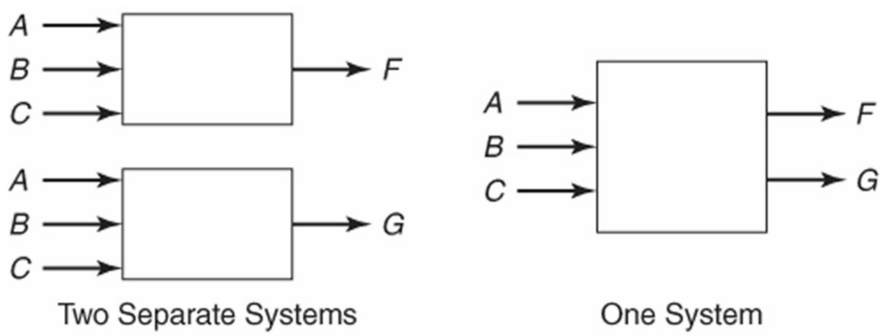




Ex 3.22 (pg. 140). Simplify $F(w,x,y,z)=\Sigma m(1,3,4,6,11)+\Sigma d(0,8,10,12,13)$ in POS.



3.6: Combinational Logic Circuits with Multiple Outputs

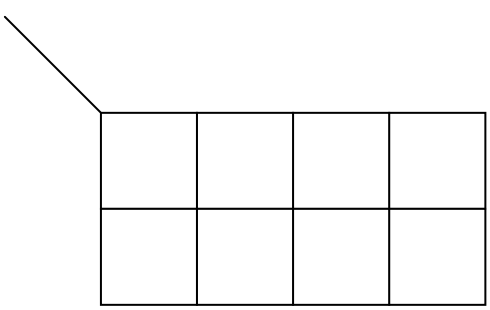


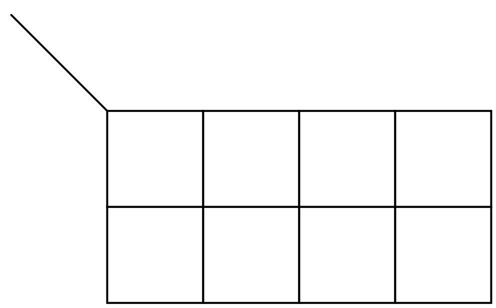
"Simplistic" Design Process:
1. Describe each function: truth table, or minterm list, or maxterm list.
2. Simplify each function independently.
3. Draw logic circuits for each output. Share logic gates if possible.

Objective: design circuit for each output while maximizing gate sharing (maximize re-use of logic gates) between all functions.

Ex (p. 147): Design a circuit with the functions described. Use the fewest number of gates. Assume complemented variables are readily available.

$F(A,B,C) = \Sigma m(0, 2, 6, 7)$ $G(A,B,C) = \Sigma m(1, 3, 6, 7)$

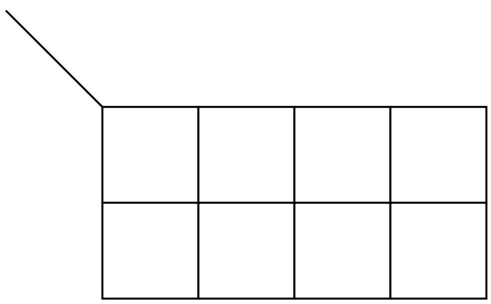


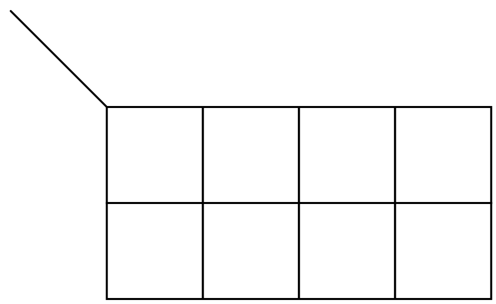


"Advanced" Design Process:
1. Describe each function: truth table, or minterm list, or maxterm list.
2. Simplify each function while being aware of other functions. The "simplified" functions may not necessarily be in simplest form, but allow maximum sharing of gates.
3. Draw logic circuits for each output. Share logic gates if possible.

Ex 1 (p. 148): Design a circuit with the functions described. Use the fewest number of gates. Assume complemented variables are readily available.

$F(A,B,C) = \Sigma m(0, 1, 6)$ $G(A,B,C) = \Sigma m(2, 3, 6)$



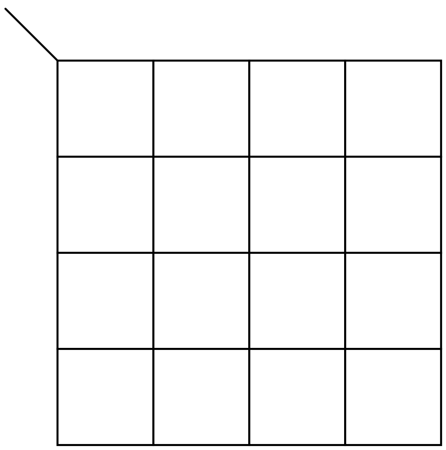


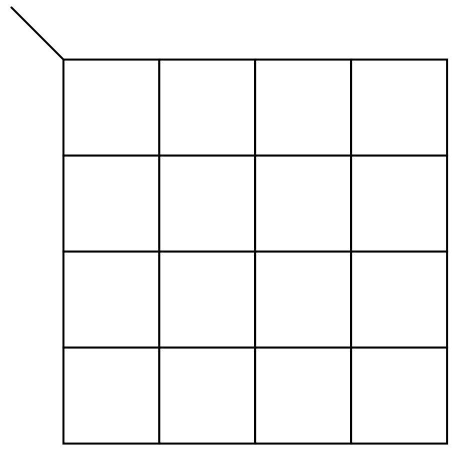
See also Example 3.28 on p. 149.

Ex 2 (p. 151): Design a circuit with the functions described. Use the fewest number of gates. Assume complemented variables are readily available.

$$F(A,B,C,D) = \Sigma m(0, 2, 3, 4, 6, 7, 10, 11)$$

$$G(A,B,C,D) = \Sigma m(0, 4, 8, 9, 10, 11, 12, 13)$$





Chapter 3 Summary:

- Use K-map method to simplify functions in SOP or POS.
  - For SOP: Group the 1's, find product terms, and OR them together.
  - For POS: Group the 0's, find sum terms, and AND them together.
  - Both describe the <u>same</u> function, i.e., produce same truth table. They are <u>**not**</u> complement functions of each other.
- Use *don't cares* if they help make a group larger.
- For multiple outputs, need to be aware of other functions when simplifying, in order to maximize gate sharing.