Karnaugh maps (K-Maps)

- Karnaugh maps are an alternative way (compared to a truth table) to describe logic functions; they are useful for displaying logic functions with up to 4 or 5 inputs.
- Whereas truth tables describe a function in a tabular format, Karnaugh maps describe a function in a *grid-like* format.
- Benefit of Karnaugh maps lies in their ability to allow one to minimize logic functions graphically — tends to be much simpler compared to using Boolean algebra to find minimized SOP and POS expressions.

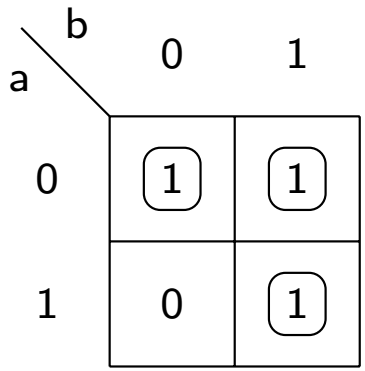
Two-variable Karnaugh maps for minimum SOP

- Consider the following logic function (via its truth table). The canonical SOP for this function is $f = a'b' + a'b + ab$.

a	b	f
0	0	1
0	1	1
1	0	0
1	1	1

- The K-Map for the function: It is a 2×2 grid of entries.

$$f = a'b' + a'b + ab$$

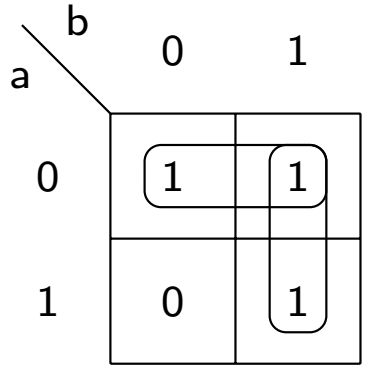


  - One input variable is used to index the rows of the K-Map; the other input variable is used to index the columns of the K-Map.
  - The entries correspond to the required values of f with the input variables are set to particular values.

Two-variable Karnaugh maps for minimum SOP

- The K-Map for the function...

$$f = a'b' + a'b + ab$$
$$f = a'b' + a'b + a'b + ab$$
$$f = a'(b' + b) + (a' + a)b = a' + b$$



- Each 1×1 rectangle in the Karnaugh map corresponds to a minterm.
- Adjacent minterms differ by a change in only one variable — adjacent rectangles with 1s can be merged together — the result is no longer a minterm, but rather a product term which encloses (includes) all minterms contained within.
- The K-Map provide a useful way to: 1) decide which terms should be duplicated; and 2) which terms should be grouped and factored to simply the Boolean expression.
- The minimum SOP in this example if $f = a' + b$.

Three-variable Karnaugh maps for minimum SOP

- Consider the following logic function (via its truth table). The canonical SOP for this function is $f = a'b'c + ab'c + abc' + abc$.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Three-variable Karnaugh maps for minimum SOP

- The K-Map for this function...

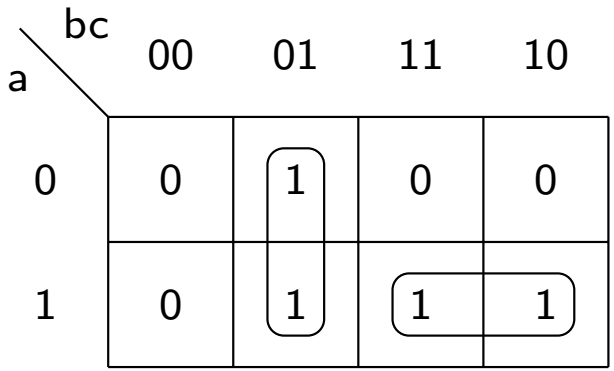
$$f = a'b'c + ab'c + abc' + abc$$

a \ bc	00	01	11	10
0	0	(1)	0	0
1	0	(1)	(1)	(1)

- It's a table with 8 entries — one for each possible value of the function.
- One variable is used to index the rows while two variables are used to index the columns.
  - Notice that the numbering of columns is not consecutive... **The numbering is done to ensure that between any two adjacent columns, only one variable changes its value.**

Three-variable Karnaugh maps for minimum SOP

- The K-Map for the function...

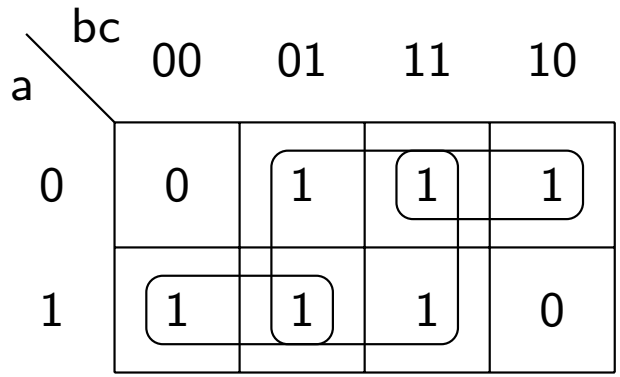


$$f = a'b'c + ab'c + abc' + abc$$
$$f = (a' + a)b'c + ab(c' + c)$$
$$f = b'c + ab$$

- Each 1×1 rectangle in the Karnaugh map corresponds to a minterm.
- Adjacent minterms differ by a change in only one variable — adjacent rectangles with 1s can be merged together — the result is no longer a minterm, but rather a product term which encloses (includes) all minterms contained within.
- We can merge adjacent rectangles in the Karnaugh map to determine that the minimum SOP is $f = b'c + ab$.

Three-variable Karnaugh maps for minimum SOP (another example)

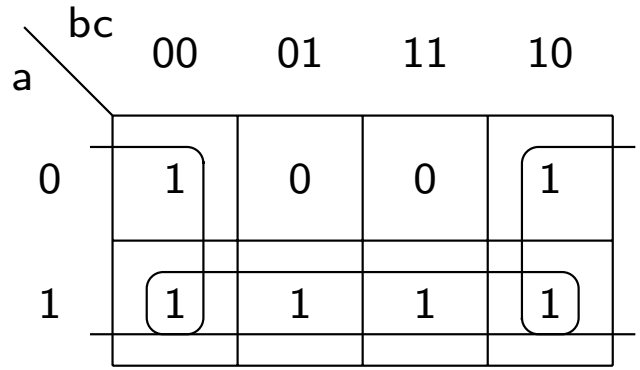
- We can obtain larger rectangles...



$$f = a'b'c + a'bc' + a'bc + ab'c' + ab'c + abc$$
$$f = a'b'c + a'bc' + a'bc + a'bc + ab'c' + ab'c + ab'c + abc$$
$$f = (a' + a)b'c + a'b(c' + c) + (a' + a)bc + ab'(c' + c)$$
$$f = b'c + a'b + bc + ab'$$
$$f = (b' + b)c + a'b + ab'$$
$$f = c + a'b + ab'$$

Three-variable Karnaugh maps for minimum SOP (another example)

- We can wrap around the edges of the K-Map...



$$f = a'b'c' + a'bc' + ab'c' + ab'c + abc' + abc$$
$$f = a'b'c' + a'bc' + ab'c' + ab'c' + ab'c + abc' + abc' + abc$$
$$f = (a' + a)b'c' + (a' + a)bc' + ab'(c' + c) + ab(c' + c)$$
$$f = b'c' + bc' + ab' + ab$$
$$f = (b' + b)c' + a(b' + b)$$
$$f = c' + a$$

Three-variable Karnaugh maps for minimum SOP (summary)

- Identify rectangles of 1s in the Karnaugh map. Rectangles will always have sizes which are powers of 2 (every time we double the size of a rectangle, we are removing a variable — merging rectangles corresponds to Boolean algebra operations!).
  - Larger rectangles correspond to product terms with fewer inputs.
- Select the fewest rectangles (product terms) as required in order to make sure all the 1s (minterms) of the logic function are included (covered) in the final SOP (by making sure that each minterm of the function is in at least 1 of the selected rectangles).

Four-variable Karnaugh maps for minimum SOP

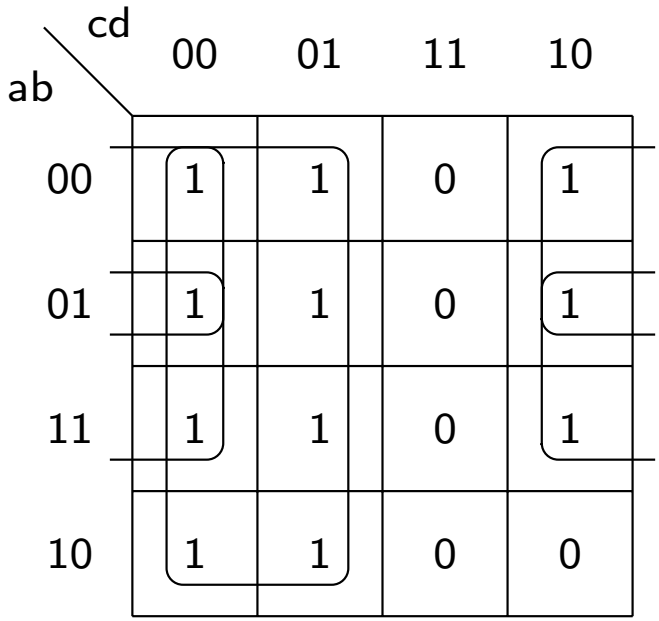
- K-Map for a 4-input function $f = f(a, b, c, d)$... (truth table omitted due to its size)...

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	1
11	1	1	0	1
10	1	1	0	0

- Two inputs to label the rows and two input variables to label the columns.
- Numbering of the rows and columns is done to ensure that adjacent rows (or columns) different by only one changing variable.

Four-variable Karnaugh maps for minimum SOP

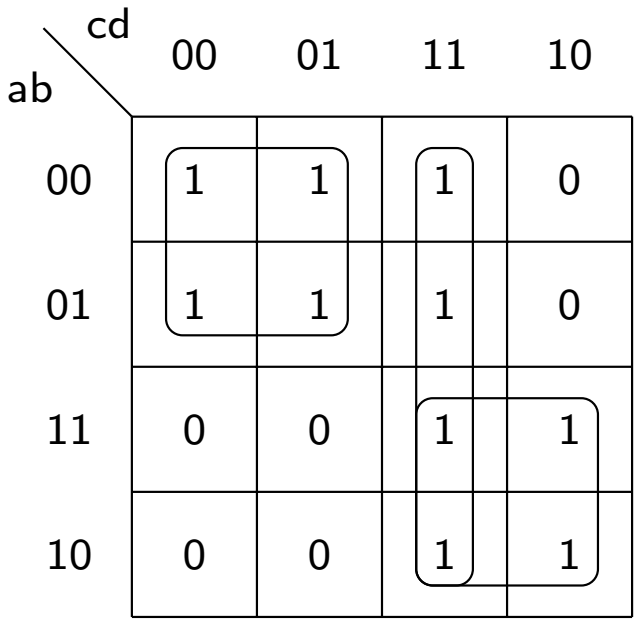
- Optimized K-Map...



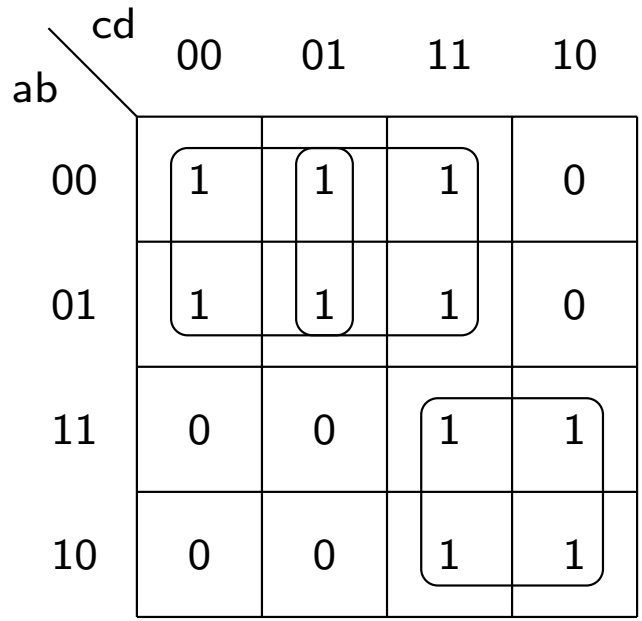
- Optimized SOP is $f = c' + a'd' + bd'$.

Four-variable Karnaugh maps for minimum SOP

- We often have choices and can obtain equally good (but different) implementations of a function f.



$f_1 = a'c' + cd + ac$

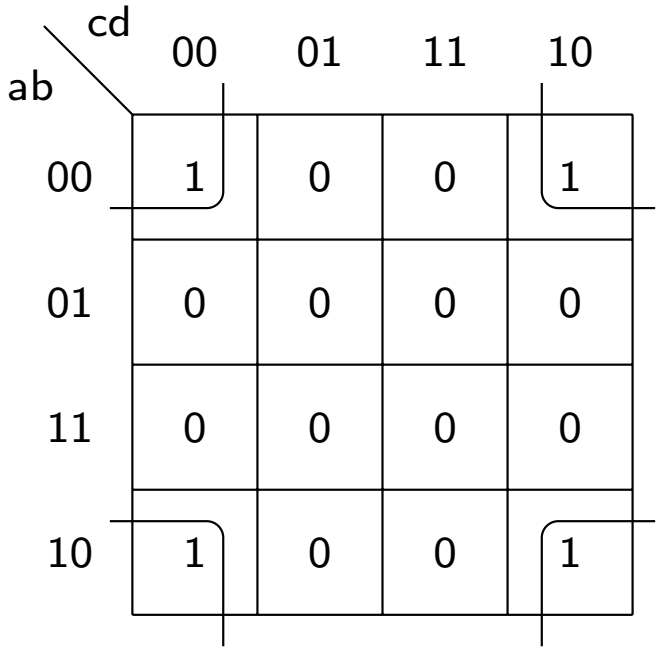


$f_1 = a'c' + a'd + ac$

- Both f_1 and f_2 implement f and have equal cost. It can be proven that $f_1 = f_2$.

Four-variable Karnaugh maps for minimum SOP

- Watch out for corners in 4-variable K-Maps...

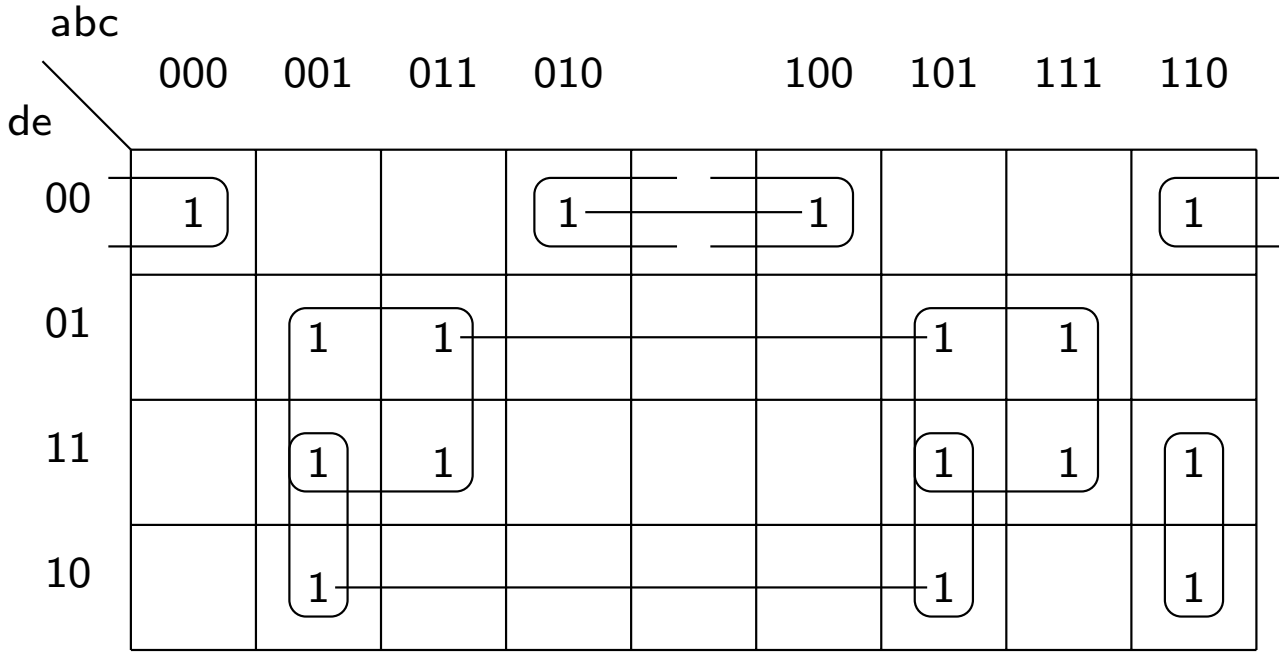


$$f_1 = a'b'c'd' + a'b'cd' + ab'c'd' + ab'cd'$$
$$f_1 = a'b'(c' + c)d' + ab'(c' + c)d'$$
$$f_1 = a'b'd' + ab'd'$$
$$f_1 = (a' + a)b'd'$$
$$f_1 = b'd'$$

- Left and right columns are adjacent. Top and bottom rows are adjacent. Therefore, the corners are also adjacent.

Five-variable Karnaugh maps for minimum SOP

- Five variable Karnaugh map can be drawn by using two, four variable maps.
- Pick one input — each four variable K-Map map corresponds to one of the two possible values of the selected input.
- Example... left four variable map corresponds to $a = 0$ while the right four variable map corresponds to $a = 1$.



Five-variable Karnaugh maps for minimum SOP

- Can compute f by considering each four variable map which gives

$$f = \overbrace{a'ce + a'b'cd + a'c'd'e'}^{\text{when } a=0} + \overbrace{ace + ab'cd + ac'd'e' + abc'd}^{\text{when } a=1}$$

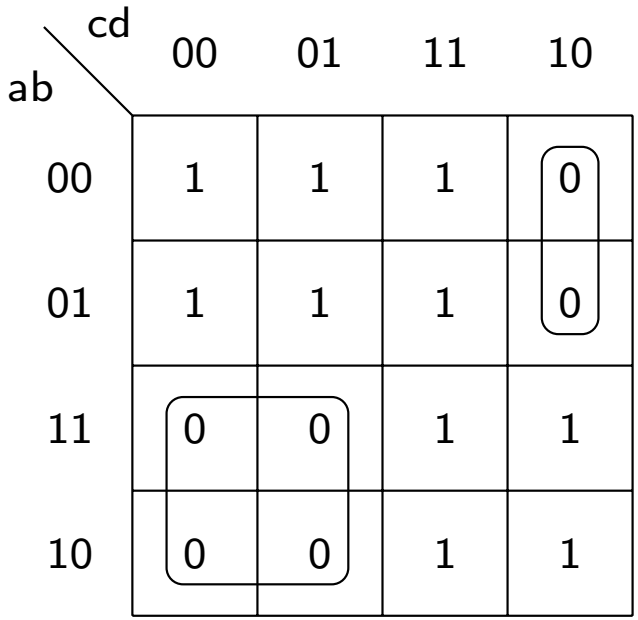
$$= \underbrace{ce + b'cd + c'd'e'}_{\text{identical terms in each map}} + \underbrace{abc'd}_{\text{unique terms in each map}}$$

Larger Karnaugh maps for minimum SOP

- Without going into details (or an example), a six variable Karnaugh map can be constructed by using four, 4 variable Karnaugh maps.
  - In this case, the four maps are arranged in a 2×2 grid.
- K-Maps for functions with more than 6 variables become impractical to draw.

Karnaugh maps for minimum POS

- More or less the same thing as with SOP...
- Rather than focusing on the 1s of the function and writing product terms, we focus on the 0s of the function and write sum terms.
- Example...



$$
\begin{aligned}
f &= (a+b+c'+d)(a+b'+c'+d)(a'+b'+c+d) \\
&\qquad (a'+b'+c+d')(a'+b+c+d)(a'+b+c+d') \\
f &= [(a+c'+d)+(bb')][(a'+b'+c)+(dd')][(a'+b+c)+(dd')] \\
f &= (a+c'+d)[(a'+b'+c)(a'+b+c)] \\
f &= (a+c'+d)[(a'+c)+(b'b)] \\
f &= (a+c'+d)(a'+c)
\end{aligned}
$$

Karnaugh maps and don't cares

- Sometimes we might be given a logic function and will be told that certain input combinations will not appear.
- For these input combinations, we *don't care* what the output of the logic function will be (since it will never happen).
- Might be tempted to set the output to either 1 or 0, but if we consider the output to be a don't care (denoted by and "X"), then we can exploit the fact that this output can be either 0 or 1 when we minimize the function.
- Depending on how we set the don't cares, we might end up with a more simplified expression.

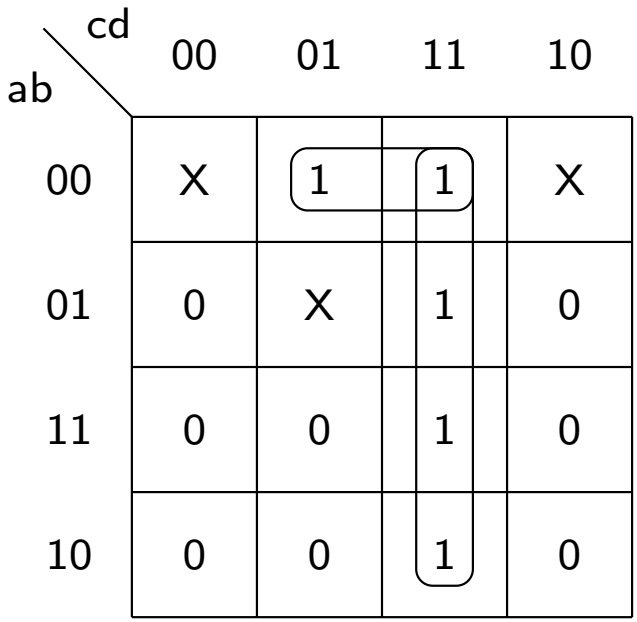
Karnaugh maps and don't cares

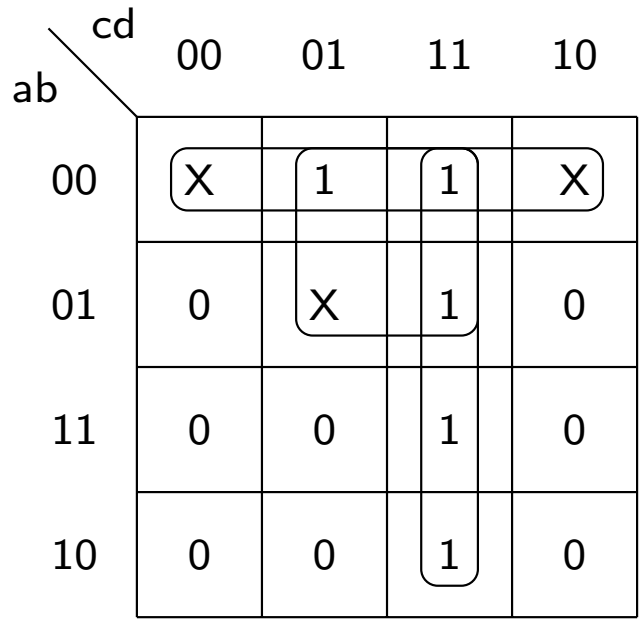
- Example... (three don't cares indicated with "X")...

ab \ cd	00	01	11	10
00	X	1	1	X
01	0	X	1	0
11	0	0	1	0
10	0	0	1	0

Karnaugh maps and don't cares

- Consider forcing all the don't cares to be "0" and then all "1"...

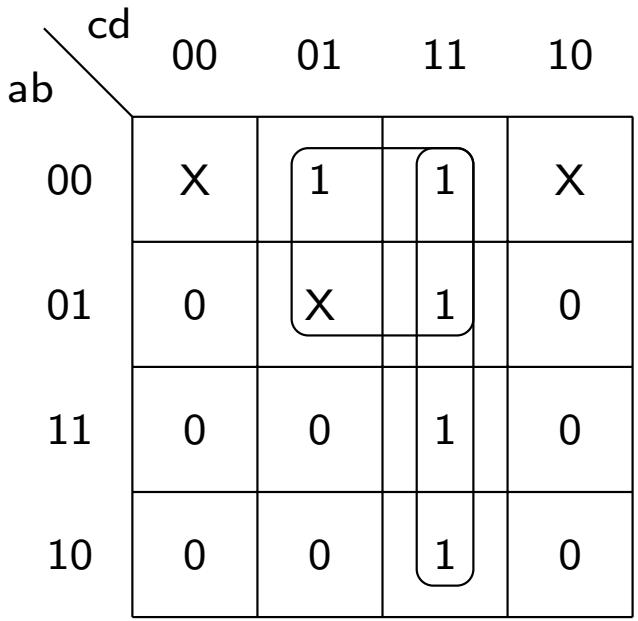




- If the don't cares are forced to be all 0, then we find that $f = a'b'd + c'd'$ (cost is 10).
- Similarly, if the don't cares are forced to be all 1, then we find that $f = a'b' + cd + a'd$ (cost is 13).

Karnaugh maps and don't cares

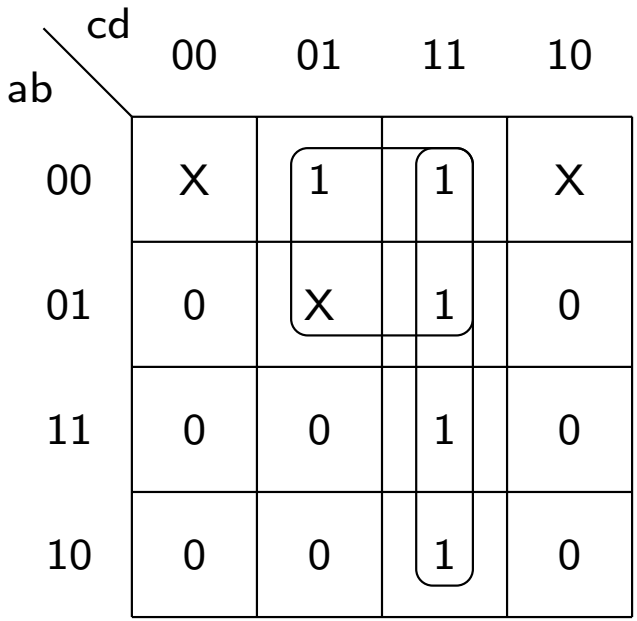
- Let the optimization guide us...

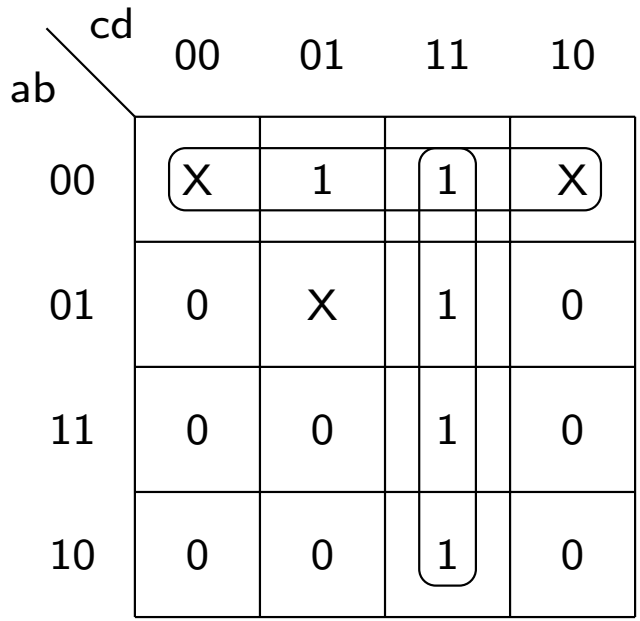


- If we set some don't cares to be 0 while others to be 1, then we can obtain $f = a'd + cd$ (cost is 9).
- Those don't cares not enclosed in a rectangle are set to 0 while those don't cares which are enclosed in a rectangle are set to 1.

Logical and algebraic equivalence

- With don't cares, we can end up with two different (but equally good) implementations which exploit the don't cares in different ways.





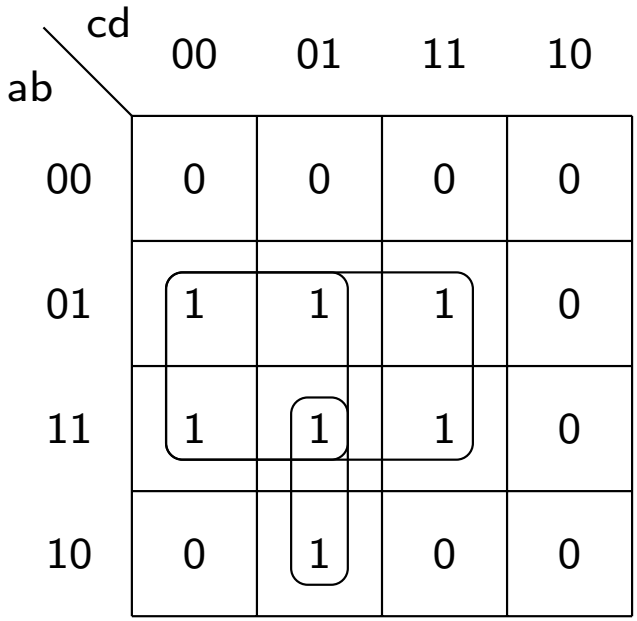
$f_1 = a'd + cd$

$f_2 = a'b' + cd$

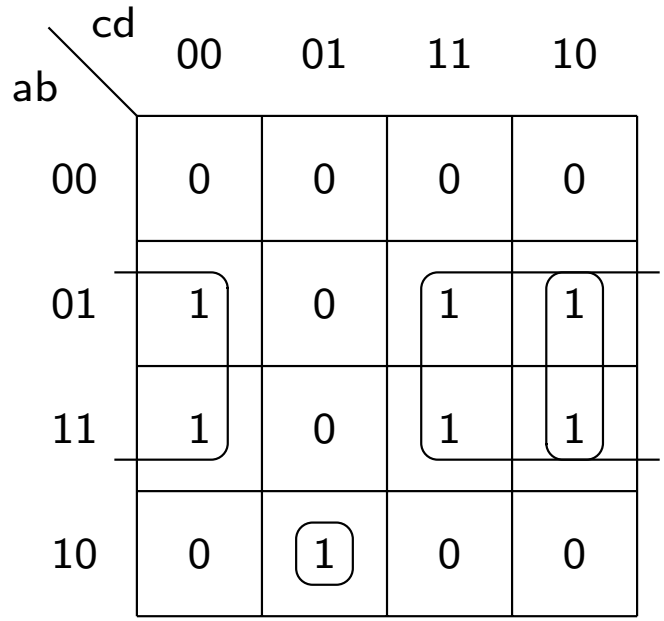
- Both f_1 and f_2 are valid implementations (produce proper values when required), but differ in how don't cares are handled.
- Functions f_1 and f_2 **are functionally equivalent** but **are not algebraically equivalent**. In other words, in is impossible to show that f_1 is equal to f_2 because they are not.

Multiple output functions and product term sharing

- Might have multiple functions which share the same inputs.



f

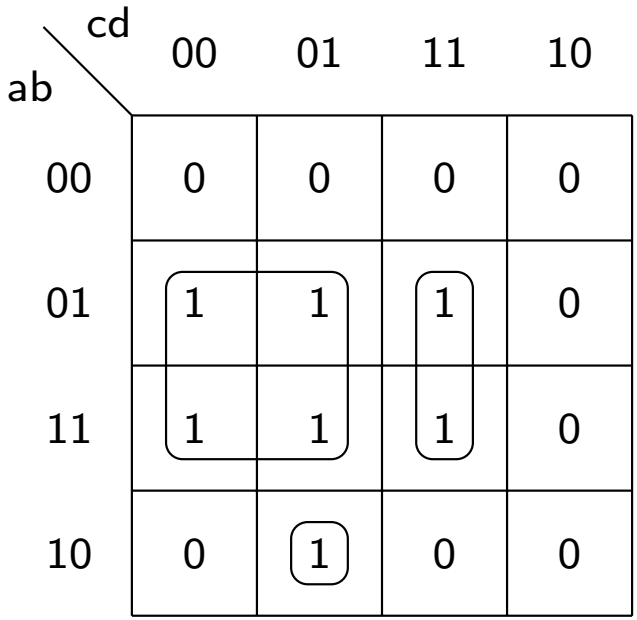


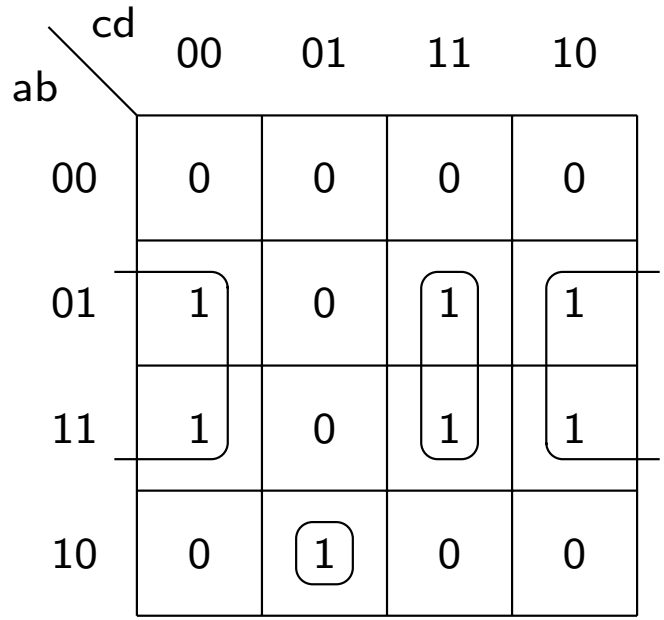
g

- Optimized separately, $f = bc' + bd + ac'd$ and $g = bd' + bc + ab'c'd$.
- The total cost of f is 14 and the total cost of g is 15.
- Functions f and g do not share any common logic — total circuit cost is 29.

Multiple output functions and product term sharing

- If we consider both f and g at the same time...

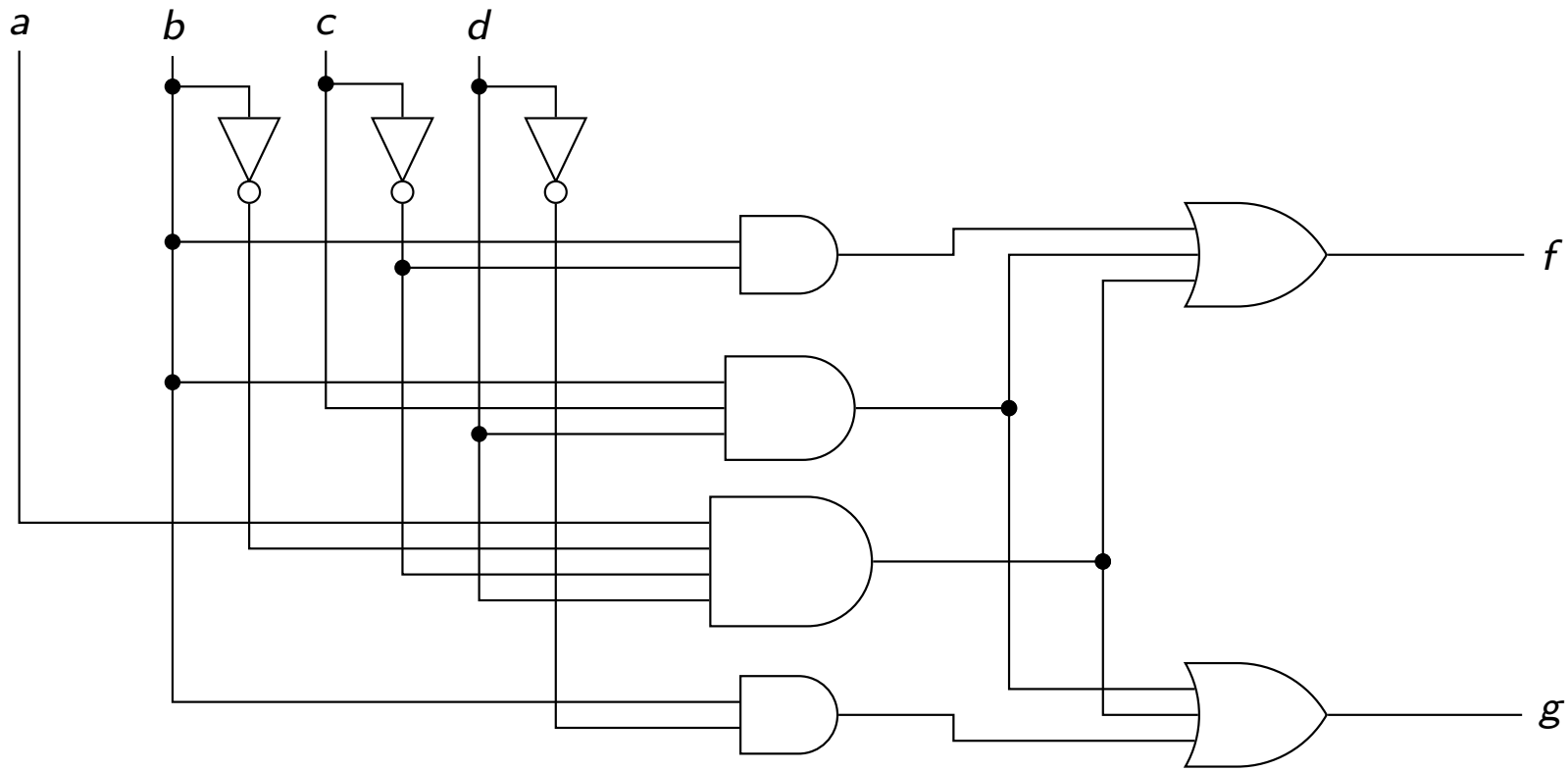




- Identified product terms to implement f and g are not necessarly the best for minimization of f and g separately.
- We would find $f = bc' + bcd + ab'c'd$ and $g = bd' + bcd + ab'c'd$ for a total cost of 32.
- However, both f and g require the product terms bcd and $ab'c'd$ — these product terms are referred to as **shared product terms**.
- Shared product terms **do not get counted twice!**. The total circuit cost is only 23.

Multiple output functions and product term sharing

- The circuit with shared product terms...



- It can often be better to accept **non-minimal implementations of individual functions** if product terms can be shared between multiple functions.

Implicants, prime implicants, essential implicants and covers

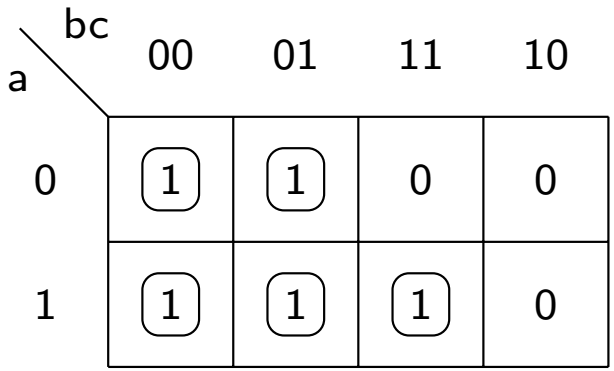
- If we are given an n variable logic function f and are told that the output of f is 0 or 1 for every 2^n possible input patterns, then f is said to be **completely specified**.
- If some outputs of f are don't cares, then f is said to be **incompletely specified**.
- For any f minterms are split into 3 sets: (a) The **on set** (minterms for which f is 1); (b) The **off set** (minterms for which f is 0); and (c) The **don't care set** (minterms for which f is a don't care).
- A product term is called an **implicant** of a function if the function is 1 for all minterms contained within the product term. **In terms of a Karnaugh map, all rectangles which include only 1s correspond to implicants of a function**.
- An implicant is called a **prime implicant** if the removal of any input from the implicate results in a product term which is **no longer** an implicant. **In terms of a Karnaugh map, a prime implicant corresponds to a rectangle of 1s which cannot be made any larger (e.g., by merging with another rectangle) without enclosing a 0**.
- An **essential prime implicant** is a prime implicant which includes a minterm that is not found in any other prime implicant.

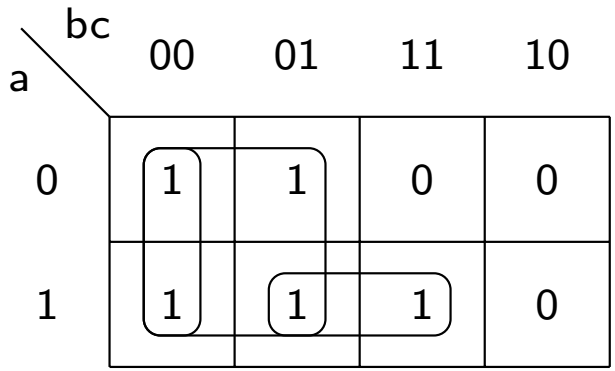
Implicants, prime implicants, essential implicants and covers

- A **cover** of a function is a collection of enough implicates to account for all situations where a logic function is a 1.
- **We can always form a cover of a function using only prime implicants if we so chose.**
- A cover that consists of only prime implicants **must** include all of the essential prime implicants — if not, then some minterms for which f produces a 1 would not be included and the implementation would be wrong.

Implicants, prime implicants, essential implicants and covers

- An illustration with K-Maps...





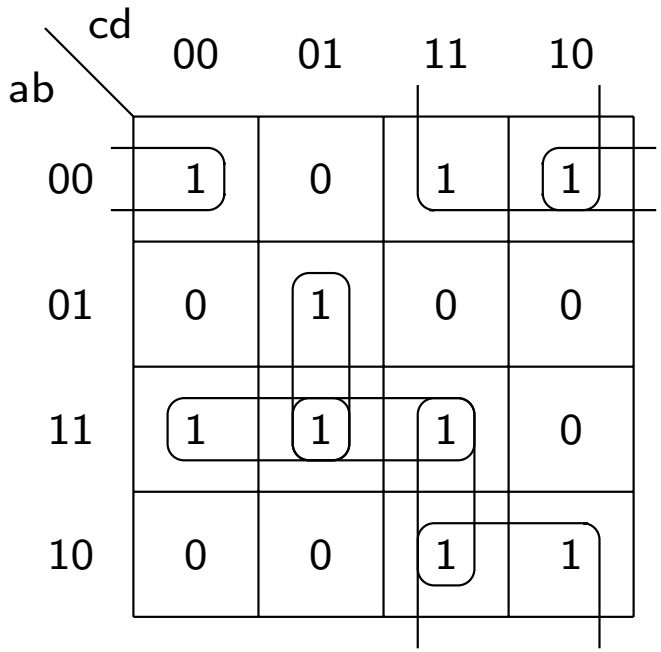
- The minterms $a'b'c'$, $a'b'c$, $ab'c'$, $ab'c$, abc are implicants. None of the minterms are prime implicants.
- The product terms $b'c'$ is an implicant, but is not a prime implicant.
- Terms b' and ac are prime implicants. Both b' and ac also happen to be essential prime implicants.

Implicants, prime implicants, essential implicants and covers

- All this terminology relates to how we optimize with K-Maps... Implementations with prime implicants tend to have low cost...
- Therefore, to optimize...
  - Generate all prime implicates;
  - Identify the essential prime implicants;
  - Include all the essential prime implicants in the cover for f. If the function is completely covered, then stop.
  - Otherwise, include as few non-essential prime implicants as possible to complete the cover of f.
- This is precisely what we try to do with Karnaugh maps: We attempt to find rectangles which are as large as possible — this corresponds to identifying prime implicants. We then select the rectangles which includes minterms not included by other rectangles (we include the essential prime implicants). Finally, we include as few additional rectangles as possible to properly implement f.

Implicants, prime implicants, essential implicants and covers

- Example...



- Two different minimal implementations: $f = a'b'd' + b'c + abc' + bc'd + abd$ or $f = a'b'd' + b'c + abc' + bc'd + acd$.
- The essential primes $a'b'd', b'c, abc'$ and $bc'd$ are included in both.
- Still need to cover minterm $abcd$ — we can either pick abd or acd which are both prime (but not essential). Only need one of them.