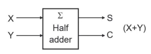
Simplification

- Fewer logic gates = cheaper, use less power, faster.
- Algebraic Simplification:
  - Minimise no. of literals, and no. of terms.
  - Sometimes conflicting
  - requires good algebraic manipulation skills.
  - eg.
    - Example 1: Simplify (x+y)·(x+y')·(x'+z)

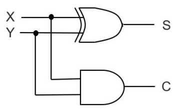
(x+y)·(x+y')·(x'+z)	
= (x·x + x·y' + x·y + y·y') · (x'+z)	(distributivity)
= (x + x·y' + x·y + y·y') · (x'+z)	(idempotency)
= (x + x·(y'+y) + y·y') · (x'+z)	(distributivity)
= (x + x·(1) + 0) · (x'+z)	(complement)
= (x + x) · (x'+z)	(identity)
= x · (x'+z)	(idempotency)
= x·x' + x·z	(distributivity)
= 0 + x·z	(complement)
= x·z	(identity)

no. of literals reduced from 6 to 2.

- Half adder
  - A circuit that adds 2 single bits (X, Y) to produce a result of 2 bits (C, S)
  - 

Inputs		Outputs		DEC
X	Y	C	S	
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	2

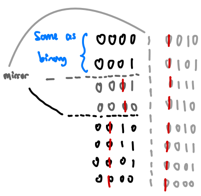
  - $C = X \cdot Y$
    $S = X' \cdot Y + X \cdot Y'$ } Sum of minterms.
    $= X \oplus Y$



- Gray Code. (Reflected Binary Code)
  - unweighted (not arithmetic)
  - Only a single bit change
  - not restricted to decimal digits: n bits → 2^n values.
  - error detection
  - no duplicate codes.

Decimal	Binary	Gray Code	Decimal	Binary	Gray code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

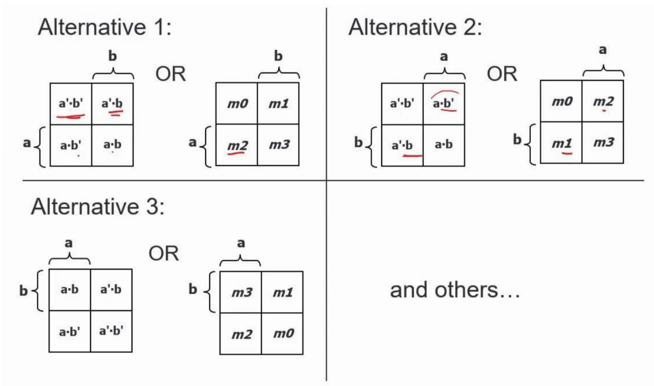
  - Algorithm for Standard Gray code sequence:

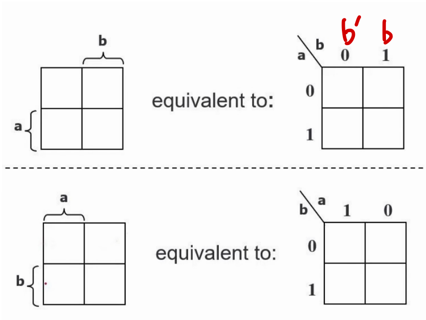


K-maps.

- Systematic method to obtain simplified sum-of-products (SOP) expressions.
- Fewest possible product terms and literals
- Easy to use.
- Limited to 5/6 variables.

- A matrix of squares – each square represents a minterm
  - 2 adj. sqr. rep. minterms that differ by exactly one literal.

- 



- K-map for a function is filled by putting.
  1. "1" in the square that corresponds to a minterm of the function.
  2. "0" otherwise

- Another way of drawing a truth-table.

eg. Half adder

	b'	b
a'	0	0
a	0	1

C = a·b

	b'	b
a'	0	1
a	1	0

S = a·b' + a'·b

eg. 3 variables

a \ bc	00	01	11	10
0	a'·b'·c'	a'·b'·c	a'·b·c	a'·b·c'
1	a·b'·c'	a·b'·c	a·b·c	a·b·c'

OR

	m0	m1	m3	m2
a	m4	m5	m7	m6

· ensure that minterms of adjacent cells differ by only one literal. (i.e. graycode sequence)

a \ bc	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

also a neighbour!

* There is wrap-around in the K-map

∴ every cell in an n-variable K-map has n adjacent neighbours.

- qns: sometimes need to do "Simplification"

i.e. $A(x,y,z) = x\cdot y + y\cdot z' + x'\cdot y'\cdot z$

$x\cdot y\cdot z' + x\cdot y\cdot z$

inside y but outside z region

overlapped region.

eg. 4 variables.

wx \ yz	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

y, x, w, z

eg. 5 variables = 2 4variables kmap

$V\cdot w'\cdot x'\cdot y'\cdot z'$

10000

V'

wx \ yz	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

V

wx \ yz	00	01	11	10
00	m16	m17	m19	m18
01	m20	m21	m23	m22
11	m28	m29	m31	m30
10	m24	m25	m27	m26

eg. 6 variables?!

b

a'·b'

cd \ ef	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

a'·b

ef / cd	10	11	01	00
00	m18	m19	m17	m16
01	m22	m23	m21	m20
11	m30	m31	m29	m28
10	m26	m27	m25	m24

a

a·b'

cd \ ef	00	01	11	10
10	m40	m41	m43	m42
11	m44	m45	m47	m46
01	m36	m37	m39	m38
00	m32	m33	m35	m34

a·b

cd / ef	10	11	01	00
10	m58	m59	m57	m56
11	m62	m63	m61	m60
01	m54	m55	m53	m52
00	m50	m51	m49	m48

- How to use K-maps?
  - Unifying Theorem: $A + A' = 1$ (aka complement law)
    - each cell containing a "1" corresponds to a minterm of a given function F where the output is 1.

      look for valid grouping of adjacent cells containing "1" then corresponds to a simpler product term of F.

      size in powers of 2.

      bigger group = smaller product term ⇒ eliminating some variables. 2^n cell eliminates n variables

· ∴ group as many cell as possible

· Select as few groups as possible to cover all the cells (minterm) of the function.

⇒ shorter product term, smaller no. of product term.

A = w'.x.y'.z' + w'.x.y'.z = w'.x.y'.(z' + z) = **w'.x.y'** ← inside x, but outside w and y.

B = w.x'.y.z' + w.x'.y.z + w.x.y.z' + w.x.y.z

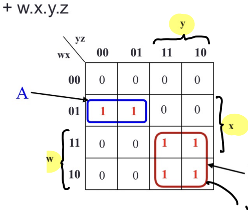
= w.x'.y.(z' + z) + w.x.y.(z' + z)

= w.x'.y + w.x.y

= w.(x'+x).y

= **w.y**

inside w and y.

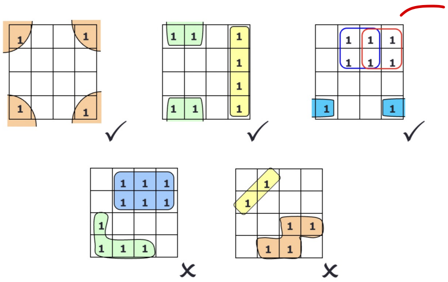


longer group = smaller product term.

$\therefore F(w,x,y,z) = A + B$

$= w'\cdot x\cdot y' + w\cdot y$ // (simplified Sum of product (SOP))

· Valid groupings



note: groups are in powers of 2

i.e. 1, 2, 4, 8, 16...

· K-map must have function in Sum-of-minterms form.

· otherwise,

1. Convert it into Sum-of-products (SOP) form.

2. Expand the SOP expression into Sum-of-minterms expression, or fill in the K-map directly based on the SOP expression.

· eg $F(A,B,C,D) = A\cdot(C+D)'\cdot(B'+D') + C\cdot(B+C'+A'\cdot D)$

$= A\cdot(C'\cdot D')\cdot(B'+D') + B\cdot C + \cancel{C\cdot C'} + A'\cdot C\cdot D$

$= \underbrace{A\cdot B'\cdot C'\cdot D'}_{1\text{ pos}} + \underbrace{A\cdot C'\cdot D'}_{2\text{ pos}} + \underbrace{B\cdot C}_{3\text{ pos}} + \underbrace{A'\cdot C\cdot D}_{2\text{ pos.}}$ → (no need to convert to Sum-of-minterms form)

Prime Implicants & Essential Prime Implicants.

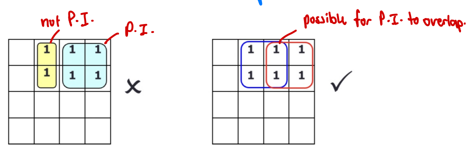
- To find the simplest SOP expression from a K-map,
  1. Min no. of literals per product term
  2. Min no. of product terms.

- Achieved through K-map via:
  1. Bigger groupings of minterms (prime implicants) where possible.
  2. No redundant groupings (look for essential prime implicants)

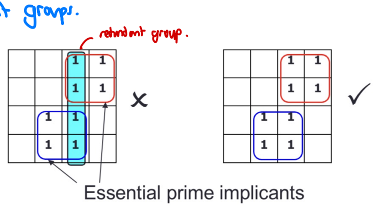
- Implicant : a product term that could be used to cover minterms of a function.

- Prime implicant : a product term obtained by combining the maximum possible number of minterms from adjacent squares in the map.
  (i.e. biggest grouping possible)

Always look for prime implicants in a K-map.



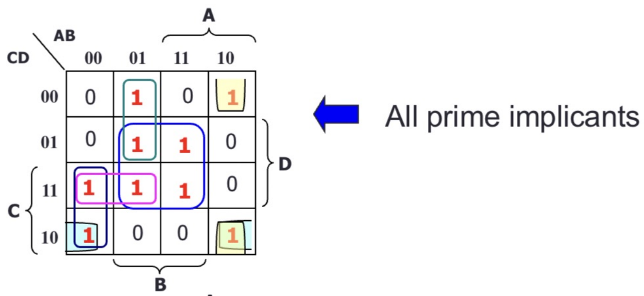
No redundant groups.



Sometimes its hard to look for redundant group.

- Essential prime implicant (EPI) : a prime implicant that includes at least one minterm that is not covered by any other prime implicant.

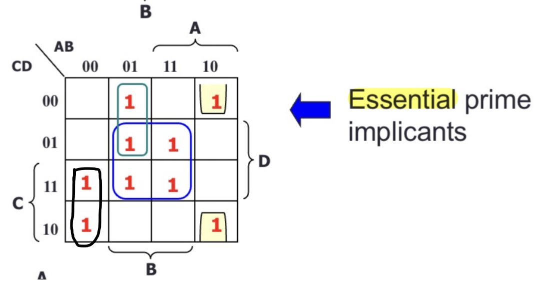
put essential groupings first, then the rest.

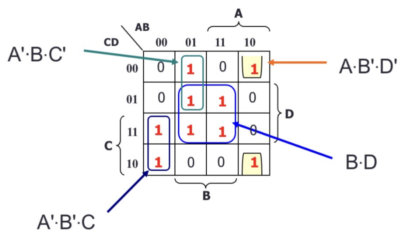


∴ To find Simplified SOP Expression

Algorithm

1. Circle all prime implicants on the K-map.
2. Identify and select all essential prime implicants for the cover.
3. Select a minimum subset of the remaining prime implicants to complete the cover, that is, to cover those minterms not covered by the essential prime implicants.





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

Simplified Sum of product Expression.

· Also can find simplified POS Expression from K-map.

→ Obtained by grouping maxterms (i.e. 0s) of the function.

K-map of F

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

K-map of F'

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

∴ SOP of F' :

$$F' = B \cdot D' + A \cdot B$$

∴ To get POS of F:

$$F = (B \cdot D' + A \cdot B)' \quad \text{— complement both sides.}$$

$$= (B \cdot D')' \cdot (A \cdot B)'$$

$$= (B' + D) \cdot (A' + B')$$

· Don't-Care Conditions

· In certain problems, some outputs are not specified or are invalid.

∴ These outputs can either be '1' or '0'

· They are called don't-care conditions, denoted by X (or d)

· Can be used to help simplify Boolean expression further in K-maps

⇒ Could be chosen to be either '1' or '0' ⇒ depending on simplification

· Σd to denote set of don't-care minterms.

Sidenote:

half-adders will never give 1,1 as output.

· Comparison

Without don't-cares:

F(A,B,C) = Σm(3, 5, 6, 7)

G(A,B,C) = Σm(1, 2, 4, 7)

F

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

F = A·C + A·B + B·C

G

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

F = A·B'·C' + A'·B'·C + A·B·C + A'·B·C'

With don't-cares:

F(A,B,C) = Σm(3, 5, 6) + Σd(0, 7)

G (A,B,C)= Σm(1, 2, 4) + Σd(0, 7)

F

A \ BC	00	01	11	10
0	X (treat as 0)	0	1	0
1	0	1	X (treat as 1)	1

F = A·C + A·B + B·C

G

A \ BC	00	01	11	10
0	X (treat as 1)	1	0	1
1	1	0	X (treat as 0)	0

G = B'·C' + A'·B' + A'·C'

Shorter