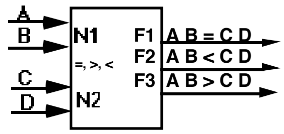
Gate logic: two-level simplification

- **Design example: two-bit comparator**



Block Diagram and Truth Table

A	B	C	D	F1	F2	F3
0	0	0	0	1	0	0
		0	1	0	1	0
		1	0	0	1	0
		1	1	0	1	0
0	1	0	0	0	0	1
		0	1	1	0	0
		1	0	0	1	0
		1	1	0	1	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	1	0	0
		1	1	0	1	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	1	0	0

A 4-Variable K-map for each of the 3 output functions

Gate logic: two-level simplification

- **Design example: two-bit comparator**

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

K-map for F1

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

K-map for F2

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

K-map for F3

F1 =

F2 =

F3 =

Gate logic: two-level simplification

- **Design example: two-bit comparator**

K-map for F1

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

K-map for F2

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

K-map for F3

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

F1 = A' B' C' D' + A' B C' D + A B C D + A B' C D'

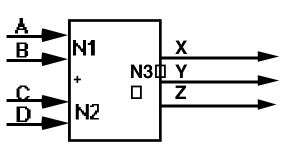
= A' C' (B xnor D) + A C (B xnor D) = (A xnor C) • (B xnor D)

F2 = A' B' D + A' C + B' C D

F3 = B C' D' + A C' + A B D'

Gate logic: two-level simplification

- **Design example: two-bit adder**



Block Diagram and Truth Table

A	B	C	D	X	Y	Z
0	0	0	0	0	0	0
		0	1	0	0	1
		1	0	0	1	0
		1	1	0	1	1
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	0	1	1
		1	1	1	0	0
1	0	0	0	0	1	0
		0	1	0	1	1
		1	0	1	0	0
		1	1	1	0	1
1	1	0	0	0	1	1
		0	1	1	0	0
		1	0	1	0	1
		1	1	1	1	0

A 4-variable K-map for each of the 3 output functions

Gate logic: two-level simplification

- Design example: two-bit adder

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

K-map for X

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

K-map for Y

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

K-map for Z

X =

Y =

Z =

Gate logic: two-level simplification

- Design example: two-bit adder

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

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

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

X = A C + B C D + A B D

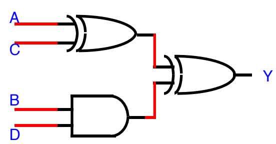
Y = A' B' C + A B' C' + A' B C' D + A' C D' + A C' D' + A B C D

Z = B D' + B' D = B xor D

Gate logic: two-level simplification

- Further simplification of Y using the Reed-Muller Algebra (AND-XOR)

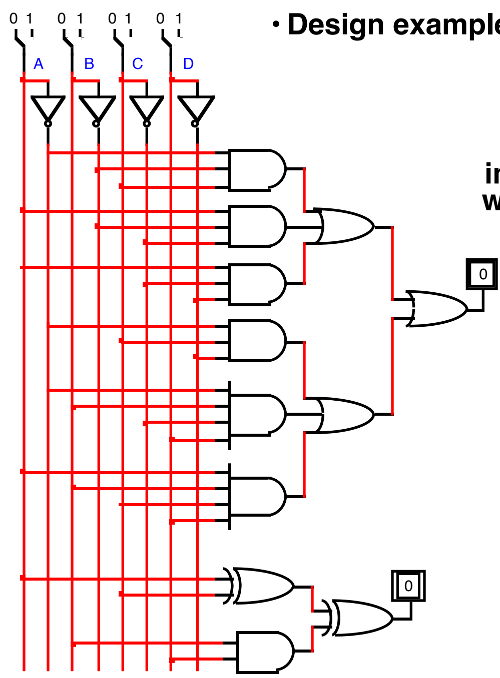
- Y = A' B' C + A B' C' + A' B C' D + A' C D' + A C' D' + A B C D
  = A' B' C + A B' C' + A' B C' D + A' (B+B') C D' + A (B+B') C' D' +A B C D
  = A' B' C + A B' C' + A' B C' D + A' B C D' + A B C' D' + A B C D
  = B' (A xor C) + A' B (C xor D) + A B (C xnor D)
  = B' (A xor C) + B (A xor C xor D)
  = B' (A xor C) xor B (A xor C xor D)
  = B' (A xor C) xor B (A xor C) xor B D
  = (B' xor B) (A xor C) xor B D
  = 1 (A xor C) xor B D
  = A xor C xor B D



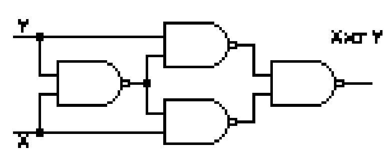
CD \ AB	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

Gate logic: two-level simplification

- Design example: two-bit adder



Two alternative implementations of Y with and without XOR



Note: XOR typically requires 4 NAND gates to implement!

Gate logic: two-level simplification

- **Design example: BCD increment by 1**

W

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

X

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

Y

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

Z

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

Gate logic: two-level simplification

- **Design example: BCD increment by 1**

W

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

X

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

W = BCD + AD'

X = BC' + BD' + B'CD

Y = A'C'D + CD'

Z = D'

Y

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

Z

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

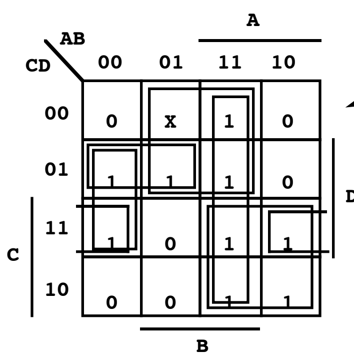
Gate logic: definition of terms for two-level simplification

- **Implicant – single element of the ON-set or DC-set or any group of these elements that can be combined to form a subcube**
- **Prime implicant – implicant that cannot be combined with another to form a larger subcube**
- **Essential prime implicant – a prime implicant is essential if it is the only one to cover an element of the ON-set (will participate in ALL possible covers of the ON-set)**

Note: Don't cares are used to form prime implicants
but cannot make the prime implicant essential

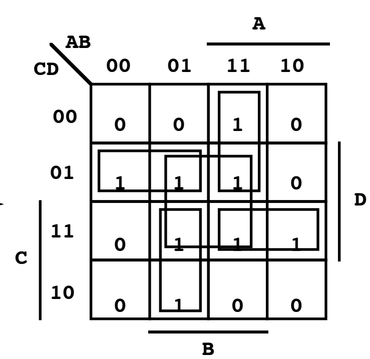
- **Objective:**
  - **grow implicant into prime implicants (minimize literals per term)**
  - **cover the ON-set with as few prime implicants as possible**
    (minimize number of product terms)
  - **essential primes participate in all possible covers**

Gate logic: examples to illustrate terms



6 prime implicants:
A'B'D, BC', AC, A'C'D, AB, B'CD

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



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

essential
minimum cover: essential implicants

Gate logic: two-level simplification

- **Algorithm: Minimum Sum of Products Expression from a K-Map**

  Step 1: Choose an element of the ON-set.

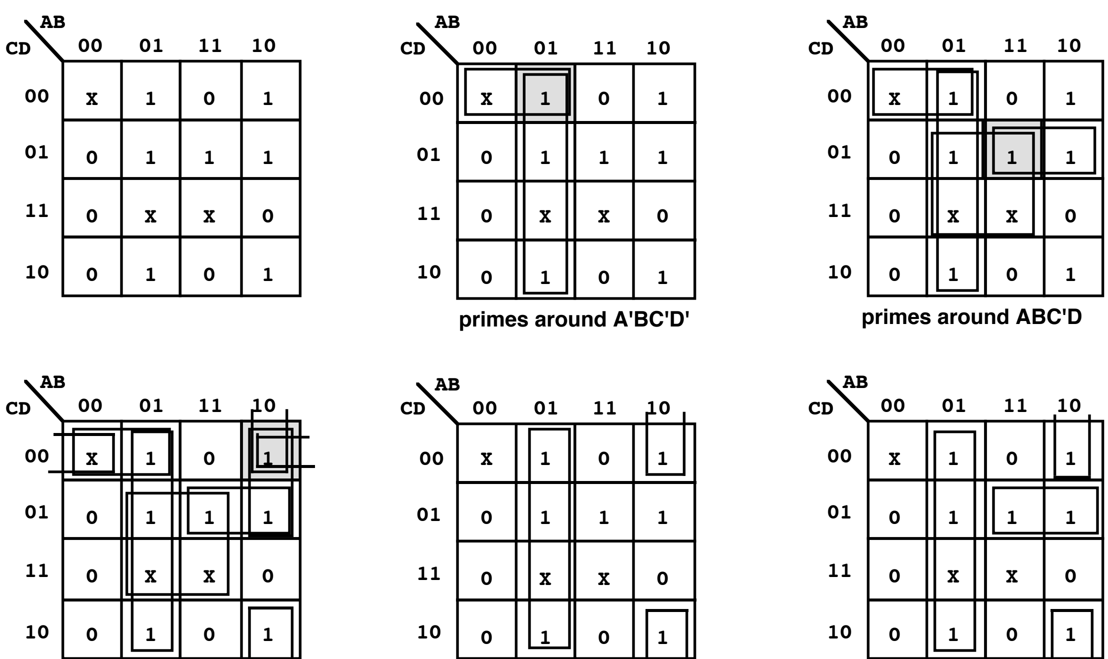
  Step 2: Find "maximal" groupings of 1s and Xs adjacent to that element. Remember to consider top/bottom row, left/right column, and corner adjacencies. This forms prime implicants (number of elements always a power of 2).

  Repeat Steps 1 and 2 to find all prime implicants

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

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

Gate logic: two-level simplification example



primes around A'BC'D'

primes around ABC'D

primes around AB'C'D'

essential primes

minimum cover of 1s

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method**
  - – tabular method to systematically find all prime implicants

Stage 1: Find all prime implicants

Step 1: Fill Column I with ON-set and DC-set minterm indices. Group by number of 1s.

Implication Table

Column I	
0000	
0100	
1000	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

$f(A,B,C,D) = \sum m(4,5,6,8,9,10,13)+\sum d(0,7,15)$

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method**
  - – tabular method to systematically find all prime implicants

Stage 1: Find all prime implicants

Step 1: Fill Column I with ON-set and DC-set minterm indices. Group by number of 1s.

Step 2: Apply Uniting Theorem— Compare elements of group w/ N 1s against those with N+1 1s. Differ by one bit implies adjacent. Eliminate variable and place in next column.

e.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Implication Table

Column I	Column II	
0000 √	0-00	
	-000	
0100 √		
1000 √	010-	
	01-0	
0101 √	100-	
0110 √	10-0	
1001 √		
1010 √	01-1	
	-101	
0111 √	011-	
1101 √	1-01	
1111 √	-111	
	11-1	

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method**
  - – tabular method to systematically find all prime implicants

Stage 1: Find all prime implicants

Step 1: Fill Column I with ON-set and DC-set minterm indices. Group by number of 1s.

Step 2: Apply Uniting Theorem— Compare elements of group w/ N 1s against those with N+1 1s. Differ by one bit implies adjacent. Eliminate variable and place in next column.

e.g., 0000 vs. 0100 yields 0-00
0000 vs. 1000 yields -000

When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.

Implication Table

Column I	Column II	Column III
0000 √	0-00 *	01-- *
	-000 *	
0100 √		-1-1 *
1000 √	010- √	
	01-0 √	
0101 √	100- *	
0110 √	10-0 *	
1001 √		
1010 √	01-1 √	
	-101 √	
0111 √	011- √	
1101 √	1-01 *	
1111 √	-111 √	
	11-1 √	

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method continued**

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

Prime Implicants:

0-00 = A' C' D'

100- = A B' C'

1-01 = A C' D

-1-1 = B D

-000 = B' C' D'

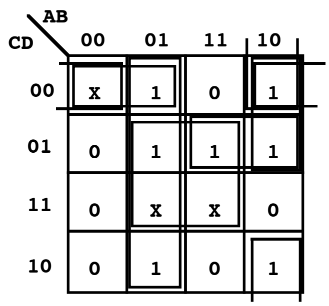
10-0 = A B' D'

01-- = A' B

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method continued**

Stage 2: find smallest set of prime implicants that cover the ON-set
recall that essential prime implicants must be in all cover
another tabular method– the prime implicant chart



Prime Implicants:

0-00 = A' C' D'

100- = A B' C'

1-01 = A C' D

-1-1 = B D

-000 = B' C' D'

10-0 = A B' D'

01-- = A' B

Gate logic: computer aids for two-level simplification

- **Quine-McCluskey method continued**
  - prime implicant chart

prime implicants:
- 0–00 = A'C'D'
- 100– = AB'C'
- 1–01 = AC'D
- –1–1 = BD
- –000 = B'C'D'
- 10–0 = AB'D'
- 01–– = A'B

prime implicant chart

ON-set elements

primes	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
8,10 (10–0)				X		X	
9,13 (1–01)					X		X
4,5,6,7 (01––)	X	X	X				
5,7,13,15 (–1–1)		X					X

Gate logic: computer aids for two-level simplification

- If column has a single X then the associated prime is essential
  – it must appear in minimum cover

	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
~~8,10 (10–0)~~				X		X	
9,13 (1–01)					X		X
~~4,5,6,7 (01—)~~	X	X	X				
5,7,13,15 (–1–1)		X					X

essential:

8,10 (10–0)
4,5,6,7 (01—)

Gate logic: computer aids for two-level simplification

- Eliminate all columns covered by essential primes

	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
~~8,10 (10–0)~~				X		X	
9,13 (1–01)					X		X
~~4,5,6,7 (01—)~~	X	X	X				
5,7,13,15 (–1–1)		X					X

essential:

8,10 (10–0)
4,5,6,7 (01—)

Gate logic: computer aids for two-level simplification

- Find minimum set of rows that covers remaining columns
  - – in general, this is a very hard problem (NP-complete)

	4	5	6	8	9	10	13
0,4 (0–00)	X						
0,8 (–000)				X			
8,9 (100–)				X	X		
~~8,10 (10–0)~~				X		X	
~~9,13 (1–01)~~					X		X
~~4,5,6,7 (01—)~~	X	X	X				
5,7,13,15 (–1–1)		X					X

essential:

8,10 (10–0)
4,5,6,7 (01—)

to complete cover:

9,13 (1–01)

$$\boxed{a'b + ab'd' + ac'd}$$

Gate logic: computer aids for two-level simplification

- Multiple-output functions
- Consider prime implicants for all outputs and whether they can be shared
- Consider conjunction of outputs (all combinations)

$f_1(a,b,c,d) = \Sigma m(2,4,10,11,12,13)$
$f_2(a,b,c,d) = \Sigma m(4,5,10,11,13)$
$f_3(a,b,c,d) = \Sigma m(1,2,3,10,11,12)$

original three functions

$f_{12}(a,b,c,d) = \Sigma m(4,10,11,13)$
$f_{13}(a,b,c,d) = \Sigma m(2,10,11,12)$
$f_{23}(a,b,c,d) = \Sigma m(10,11)$

terms that can be shared between pairs of function

$f_{123}(a,b,c,d) = \Sigma m(10,11)$

terms that can be shared between all three

Gate logic: computer aids for two-level simplification

		2	4	10	11	12	13	4	5	10	11	13	1	2	3	10	11	12
a	3													X	X	X	X	
b	1		X			X												
c	1					X	X											
d	2							X	X									
e	2								X			X						
f	3												X		X			
g	13	X		X										X		X		
h	123			X	X					X	X					X	X	
i	12		X					X										
j	13					X												X
k	12						X					X						

(b+i)(c+k)(d+i)(d+e)(e+k) = 1 every column must be covered

cei+bcde+eik+dik+bdk = 1 multiply out

bcde is clearly out of consideration – it requires one more product term
any one of cei, eik, dik, or bdk will do for minimum number of product
terms or for minimum number of literals (they are all equal cost)

f_1 = c+g+h+i f_2 = e+h+i f_3 = f+g+h+j

Gate logic: computer aids for two-level simplification

- Problem with Quine-McCluskey:
  - number of prime implicants grows rapidly with the number of inputs
    - upper bound: 3**n / n, where n is the number of inputs
  - finding a minimum cover is NP-complete, i.e., a computational expensive process not likely to yield to any efficient algorithm
- Solution: use heuristics, trade minimality of answer for speed
  - Espresso: don't generate all prime implicants (QM stage 1)
    - judiciously select a subset of primes that still covers ON-set
    - operate as a human would in finding primes in a K-map

Gate logic: computer aids for two-level simplification

- Espresso method overview

- 1. Expands implicants to their maximum size
  - Implicants covered by an expanded implicant are removed from further consideration
  - Quality of result depends on order of implicant expansion
  - Heuristic methods used to determine order
  - Step is called EXPAND

- 2. Irredundant cover – no proper subset is also a cover – is extracted from the expanded primes
  - Just like the Quine-McCluskey Prime Implicant Chart
  - Step is called IRRED-COVER

- 3. Solution usually pretty good, but sometimes can be improved
  - Might exist another cover with fewer terms or fewer literals
  - Shrink prime implicants to smallest size that still covers ON-set
  - Step is called REDUCE

- 4. Repeat sequence REDUCE/EXPAND/IRRED-COVER to find alternative prime implicants
  - Keep doing this as long as new covers improve on last solution

- 5. A number of optimizations are tried
  - e.g., identify and remove essential primes early in the process

Gate logic: computer aids for two-level simplification

- Espresso input and output example:
  - f(A,B,C,D) = Σm(4,5,6,8,9,10,13) + Σd(0,7,15)

Input

```
.i 4          -- # inputs
.o 1          -- # outputs
.ilb a b c d  -- input names
.ob f         -- output name
.p 10         -- number of product terms
0100  1       -- A'BC'D'
0101  1       -- A'BC'D
0110  1       -- A'BCD'
1000  1       -- AB'C'D'
1001  1       -- AB'C'D
1010  1       -- AB'CD'
1101  1       -- ABC'D
0000  -       -- A'B'C'D' don't care
0111  -       -- A'BCD don't care
1111  -       -- ABCD don't care
.e            -- end of list
```

Output

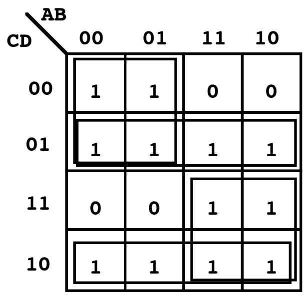
```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01  1
10-0  1
01--  1
.e
```

f = AC'D+AB'D'+A'B

© 1992 Gaetano Borriello, 1994 Steven Burns

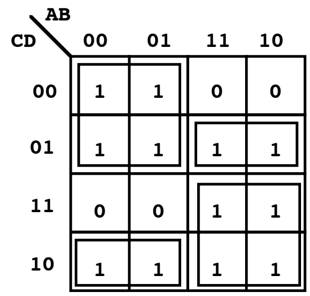
Gate logic: computer aids for two-level simplification

- Espresso algorithm:
  - why iterate (REDUCE–IRREDUNDANT COVER–EXPAND) ?



initial set of primes found in step 1 and 2 of the algorithm

4 primes, irredundant cover but not a minimal cover

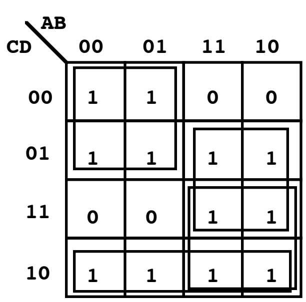


result of REDUCE is to shrink primes while still covering the ON-set

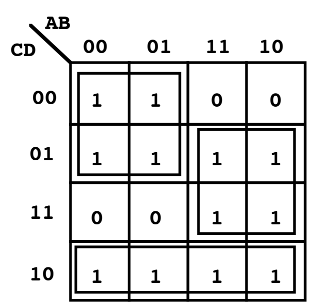
order in which shrink is performed is important

Gate logic: computer aids for two-level simplification

- Espresso iteration (cont'd)



second EXPAND generates a different set of prime implicants



IRREDUNDANT COVER found by final step solution now is only 3 prime implicants

Gate logic: chapter summary

• Primitive logic building blocks

- INVERTER, AND, OR, NAND, NOR, XOR, XNOR

• Canonical Forms Sum of Products, Products of Sums

- Incompletely specified functions/don't cares

• Logic Minimization

- Goal: two-level logic realizations with fewest gates and fewest number of gate inputs

- Obtained via Laws and Theorems of Boolean Algebra
- or Boolean Cubes and the Uniting Theorem
- or K-map Methods
- or Quine-McCluskey Algorithm
- or Espresso CAD Tool