Chapter #2: Two-Level Combinational Logic

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley

May 1993

Motivation

Further Amplification on the Concepts of Chapter #1:

- *Rapid prototyping technology*

  Use of computer aided design tools: espresso

- *Design Techniques that Spanning Multiple Technologies*

  Transistor-Transistor Logic (TTL)

  Complementary Metal on Oxide Silicon (CMOS)

- *Multiple Design Representations*

  Truth Tables

  Static gate descriptions

  Dynamic waveform descriptions

Chapter Overview

- *Logic Functions and Switches*

  Not, AND, OR, NAND, NOR, XOR, XNOR

- *Gate Logic*

  Laws and Theorems of Boolean Algebra

  Two Level Canonical Forms

  Incompletely Specified Functions

- *Two Level Simplification*

  Boolean Cubes

  Karnaugh Maps

  Quine-McClusky Method

  Espresso Methos

Logic Functions: Boolean Algebra

Algebraic structure consisting of:

set of elements *B*

binary operations {+, •}

unary operation {'}

such that the following axioms hold:

1. *B* contains at least two elements, *a*, *b*, such that *a* *b*

2. *Closure* *a*,*b* in *B*,
   (i) $a + b$ in *B*
   (ii) $a \bullet b$ in *B*

3. *Commutative Laws*: *a*,*b* in *B*,
   (i) $a + b = b + a$
   (ii) $a \bullet b = b \bullet a$

4. *Identities*: 0, 1 in *B*
   (i) $a + 0 = a$
   (ii) $a \bullet 1 = a$

5. *Distributive Laws*:
   (i) $a + (b \bullet c) = (a + b) \bullet (a + c)$
   (ii) $a \bullet (b + c) = a \bullet b + a \bullet c$

6. *Complement*:
   (i) $a + a' = 1$
   (ii) $a \bullet a' = 0$

Logic Functions: Boolean Algebra

B = {0,1}, + = OR, • = AND, ' = NOT is a Boolean Algebra

must verify that the axioms hold:

E.g., Commutative Law:

0 + 1 = 1 + 0?   0 • 1 = 1 • 0?
1 = 1   0 = 0

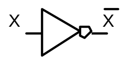
Theorem: any Boolean function that can be expressed as a truth table can be written as an expression in Boolean Algebra using ', +, •

Review from Chapter 1

NOT

Description
If X = 0 then X' = 1
If X = 1 then X' = 0

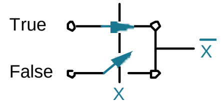
Gates



Truth Table

X	$\overline{X}$
0	1
1	0

Switches



AND

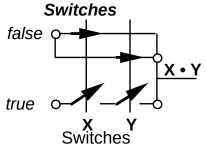
Description
Z = 1 if X and Y are both 1

Gates

Truth Table

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

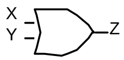
Switches



OR

Description
Z = 1 if X or Y (or both) are 1

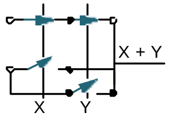
Gates



Truth Table

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Switches



Logic Functions: From Expressions to Gates

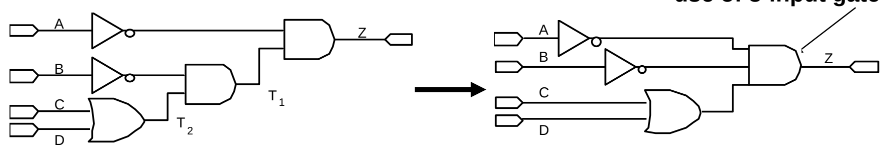
More than one way to map an expression to gates

T2

E.g., Z = A' • B' • (C + D) = (A' • (B' • (C + D)))

T1

use of 3-input gate



***Literal*: each appearance of a variable or its complement in an expression**

E.g., Z = A B' C + A' B + A' B C' + B' C

3 variables, 10 literals

Logic Functions: NAND, NOR, XOR, XNOR

16 functions of two variables:

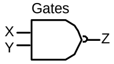
X	Y	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	X • Y		X		Y		X + Y			$\overline{Y}$		$\overline{X}$			1

X, X', Y, Y', X•Y, X+Y, 0, 1 only half of the possible functions

NAND

Description
Z = 1 if X is 0 or Y is 0

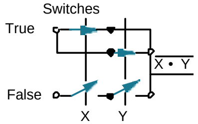
Gates



Truth Table

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

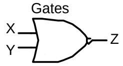
Switches



NOR

Description
Z = 1 if both X and Y are 0

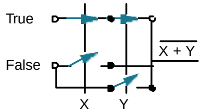
Gates



Truth Table

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Switches



Logic Functions: NAND, NOR Implementation

NAND, NOR gates far outnumber AND, OR in typical designs

easier to construct in the underlying transistor technologies

Any Boolean expression can be implemented by NAND, NOR, NOT gates

In fact, NOT is superfluous
(NOT = NAND or NOR with both inputs tied together)

X	Y	X NOR Y
0	0	1
1	1	0

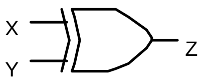
X	Y	X NAND Y
0	0	1
1	1	0

Logic Functions: XOR, XNOR

XOR: X or Y but not both ("inequality", "difference")
XNOR: X and Y are the same ("equality", "coincidence")

Description
Z = 1 if X has a different value than Y

Gates



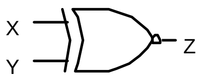
Truth Table

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

(a) XOR

Description
Z = 1 if X has the same value as Y

Gates



Truth Table

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

(b) XNOR

$$X \oplus Y = X Y' + X' Y$$

$$\overline{X \oplus Y} = X Y + X' Y'$$

Logic Functions: Waveform View

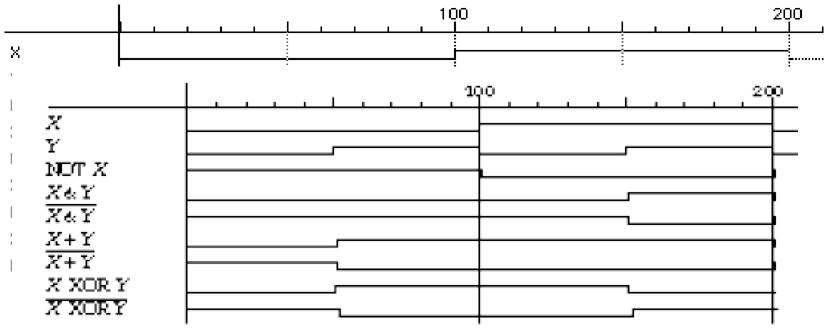


Figure 2.9 Timing waveforms for NOT, AND, NAND, OR, NOR, XOR, XNOR

Logic Functions: Rationale for Simplification

Logic Minimization: reduce complexity of the gate level implementation

- **reduce number of literals (gate inputs)**
- **reduce number of gates**
- **reduce number of levels of gates**

fewer inputs implies faster gates in some technologies

fan-ins (number of gate inputs) are limited in some technologies

fewer levels of gates implies reduced signal propagation delays

minimum delay configuration typically requires more gates

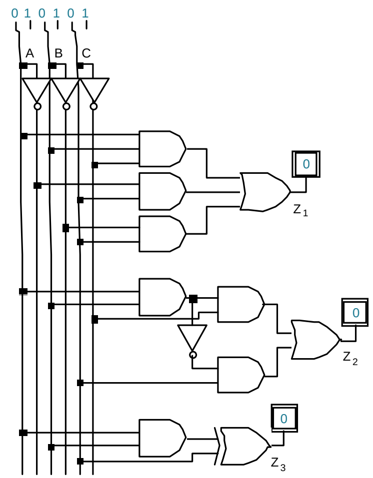
number of gates (or gate packages) influences manufacturing costs

Traditional methods:
reduce delay at expense of adding gates

New methods:
trade off between increased circuit delay and reduced gate count

Logic Functions: Alternative Gate Realizations

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Two-Level Realization (inverters don't count)

Multi-Level Realization
***Advantage:* Reduced Gate Fan-ins**

Complex Gate: XOR
***Advantage:* Fewest Gates**

TTL Package Counts:
Z1 - three packages (1x 6-inverters, 1x 3-input AND, 1x 3-input OR)
Z2 - three packages (1x 6-inverters, 1x 2-input AND, 1x 2-input OR)
Z3 - two packages (1x 2-input AND, 1x 2-input XOR)

Logic Functions: Waveform Verification

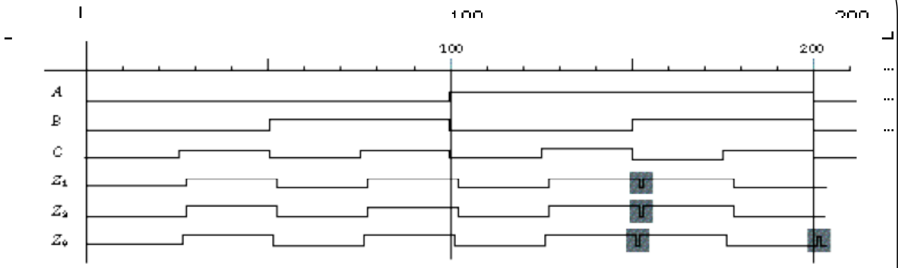


Figure 2.11 Waveform behavior of three implementations of the truth table of Figure 2.9(a).

Under the same input stimuli, the three alternative implementations have essentially the same waveform behavior.

Slight variations due to differences in number of gate levels

The three implementations are equivalent

Gate Logic: Laws of Boolean Algebra

Duality: a dual of a Boolean expression is derived by replacing AND operations by ORs, OR operations by ANDs, constant 0s by 1s, and 1s by 0s (literals are left unchanged).

Any statement that is true for an expression is also true for its dual!

Useful Laws/Theorems of Boolean Algebra:

Operations with 0 and 1:

1. X + 0 = X — 1D. X • 1 = X
2. X + 1 = 1 — 2D. X • 0 = 0

Idempotent Law:

3. X + X = X — 3D. X • X = X

Involution Law:

4. (X')' = X

Laws of Complementarity:

5. X + X' = 1 — 5D. X • X' = 0

Commutative Law:

6. X + Y = Y + X — 6D. X • Y = Y • X

Gate Logic: Laws of Boolean Algebra (cont)

Associative Laws:

7. (X + Y) + Z = X + (Y + Z) = X + Y + Z — 7D. (X • Y) • Z = X • (Y • Z) = X • Y • Z

Distributive Laws:

8. X • (Y+ Z) = (X • Y) + (X •Z) — 8D. X + (Y• Z) = (X + Y) • (X + Z)

Simplification Theorems:

9. X • Y + X • Y' = X — 9D. (X + Y) • (X + Y') = X
10. X + X • Y = X — 10D. X • (X + Y) = X
11. (X + Y') • Y = X • Y — 11D. (X • Y') + Y = X + Y

DeMorgan's Law:

12. (X + Y + Z + ...)' = X' • Y' • Z' • ... — 12D. (X • Y • Z • ...) ' = X' + Y' + Z' + ...
13. {F(X1,X2,...,Xn,0,1,+,•)}' = {F(X1',X2',...,Xn',1,0,•,+)}

Duality:

14. $(X + Y + Z + ...)^D$ = X • Y • Z • ... — 14D. $(X •FY • Z • ...)^D$ = X + Y + Z + ...
15. $\{F(X1,X2,...,Xn,0,1,+,•)\}^D$ = {F(X1,X2,...,Xn,1,0,•,+)}

Theorems for Multiplying and Factoring:

16. (X + Y) • (X' + Z) = X • Z + X' • Y — 16D. X • Y + X' • Z = (X + Z) • (X' + Y)

Consensus Theorem:

17. (X • Y) + (Y • Z) + (X' • Z) = X • Y + X' • Z — 17D. (X + Y) • (Y + Z) • (X' + Z) = (X + Y) • (X' + Z)

Gate Logic: Laws of Boolean Algebra

Proving theorems via axioms of Boolean Algebra:

E.g., prove the theorem: X • Y + X • Y' = X

E.g., prove the theorem: X + X • Y = X

Gate Logic: Laws of Boolean Algebra

Proving theorems via axioms of Boolean Algebra:

E.g., prove the theorem: X • Y + X • Y' = X

distributive law (8)	X • Y + X •Y' = X • (Y + Y')
complementary law (5)	X • (Y + Y') = X • (1)
identity (1D)	X • (1) = X

E.g., prove the theorem: X + X • Y = X

identity (1D)	X + X • Y = X • 1 + X • Y
distributive law (8)	X • 1 + X • Y = X • (1 + Y)
identity (2)	X • (1 + Y) = X • (1)
identity (1)	X • (1) = X

Gate Logic: Laws of Boolean Algebra

DeMorgan's Law

(X + Y)' = X' • Y'

NOR is equivalent to AND with inputs complemented

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X+Y}$	$\overline{X}\bullet\overline{Y}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

(X • Y)' = X' + Y'

NAND is equivalent to OR with inputs complemented

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X\bullet Y}$	$\overline{X}+\overline{Y}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

DeMorgan's Law can be used to convert AND/OR expressions to OR/AND expressions

Example:

Z = A' B' C + A' B C + A B' C + A B C'

Z' = (A + B + C') • (A + B' + C') • (A' + B + C') • (A' + B' + C)

Gate Logic: Laws of Boolean Algebra

Apply the laws and theorems to simplify Boolean equations

Example: full adder's carry out function

Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin

Gate Logic: Laws of Boolean Algebra

Apply the laws and theorems to simplify Boolean equations

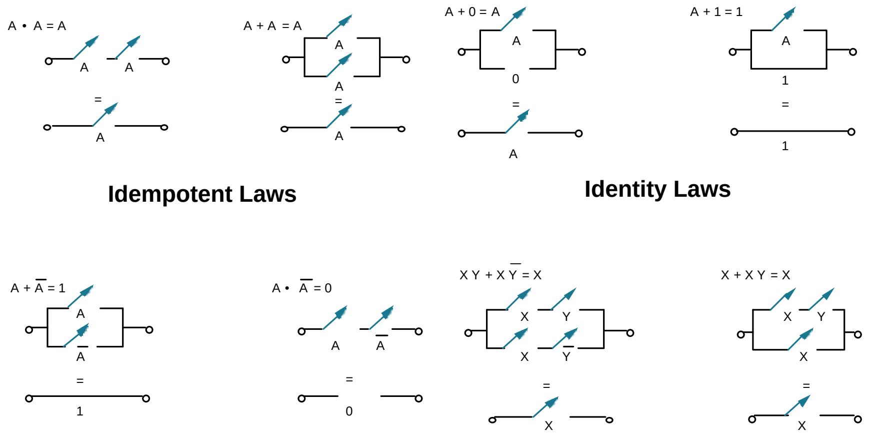
Example: full adder's carry out function

identity

Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin
= A' B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= A' B Cin + A B Cin + A B' Cin + A B Cin' + A B Cin
= (A' + A) B Cin + A B' Cin + A B Cin' + A B Cin
= (1) B Cin + A B' Cin + A B Cin' + A B Cin
= B Cin + A B' Cin + A B Cin' + A B Cin + A B Cin
= B Cin + A B' Cin + A B Cin + A B Cin' + A B Cin
= B Cin + A (B' + B) Cin + A B Cin' + A B Cin
= B Cin + A (1) Cin + A B Cin' + A B Cin
= B Cin + A Cin + A B (Cin' + Cin)
= B Cin + A Cin + A B (1)
= B Cin + A Cin + A B

associative

Gate Logic: Switching Equivalents



Idempotent Laws **Identity Laws**

Complementarity Laws **Simplification Theorems**

Gate Logic: 2-Level Canonical Forms

Truth table is the unique signature of a Boolean function

Many alternative expressions (and gate realizations) may have the same truth table

Canonical form: standard form for a Boolean expression provides a unique algebraic signature

Sum of Products Form

also known as disjunctive normal form, minterm expansion

0 1 1 1 0 0 1 0 1 1 1 0 1 1 1

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

A	B	C	F	$\overline{F}$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

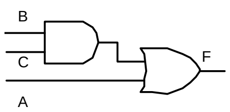
F' = A' B' C' + A' B' C + A' B C'

Gate Logic: Two Level Canonical Forms

Sum of Products

A	B	C	Minterms
0	0	0	$\overline{A}\,\overline{B}\,\overline{C} = m_0$
0	0	1	$\overline{A}\,\overline{B}\,C = m_1$
0	1	0	$\overline{A}\,B\,\overline{C} = m_2$
0	1	1	$\overline{A}\,B\,C = m_3$
1	0	0	$A\,\overline{B}\,\overline{C} = m_4$
1	0	1	$A\,\overline{B}\,C = m_5$
1	1	0	$A\,B\,\overline{C} = m_6$
1	1	1	$A\,B\,C = m_7$

Shorthand Notation for Minterms of 3 Variables



2-Level AND/OR Realization

product term / minterm:

ANDed product of literals in which each variable appears exactly once, in true or complemented form (but not both!)

F in canonical form:

F(A,B,C) = Σm(3,4,5,6,7)
= m3 + m4 + m5 + m6 + m7
= A' B C + A B' C' + A B' C + A B C' + A B C

canonical form/minimal form

F = A B' (C + C') + A' B C + A B (C' + C)

= A B' + A' B C + A B

= A (B' + B) + A' B C

= A + A' B C

= A + B C

F = (A + B C)' = A' (B' + C') = A' B' + A' C'

Gate Logic: 2 Level Canonical Forms

Product of Sums / Conjunctive Normal Form / Maxterm Expansion

A	B	C	Maxterms
0	0	0	$A + B + \underline{C} = M_0$
0	0	1	$A + \underline{B} + C = M_1$
0	1	0	$A + \underline{B} + \underline{C} = M_2$
0	1	1	$\underline{A} + B + C = M_3$
1	0	0	$\underline{A} + B + \underline{C} = M_4$
1	0	1	$\underline{A} + \underline{B} + C = M_5$
1	1	0	$\underline{A} + \underline{B} + \underline{C} = M_6$
1	1	1	$A + B + C = M_7$

Maxterm Shorthand Notation for a Function of Three Variables

Maxterm:

ORed sum of literals in which each variable appears exactly once in either true or complemented form, but not both!

Maxterm form:

Find truth table rows where F is 0
0 in input column implies true literal
1 in input column implies complemented literal

F(A,B,C) = ΠM(0,1,2)
= (A + B + C) (A + B + C') (A + B' + C)

F'(A,B,C) = ΠM(3,4,5,6,7)
= (A + B' + C') (A' + B + C) (A' + B + C') (A' + B' + C) (A' + B' + C')

Gate Logic: Two Level Canonical Forms

Sum of Products, Products of Sums, and DeMorgan's Law

$F' = A' B' C' + A' B' C + A' B C'$

Apply DeMorgan's Law to obtain F:

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

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

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

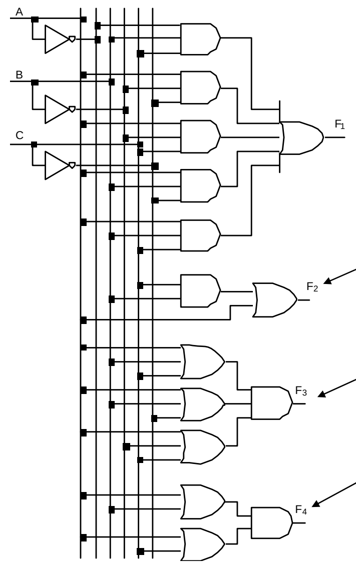
Apply DeMorgan's Law to obtain F:

$(F')' = \{(A + B' + C') (A' + B + C) (A' + B + C') (A' + B' + C) (A' + B' + C')\}'$

$F = A' B C + A B' C' + A B' C + A B C' + A B C$

Gate Logic: Two-Level Canonical Forms

Four Alternative Implementations of F:



Canonical Sum of Products

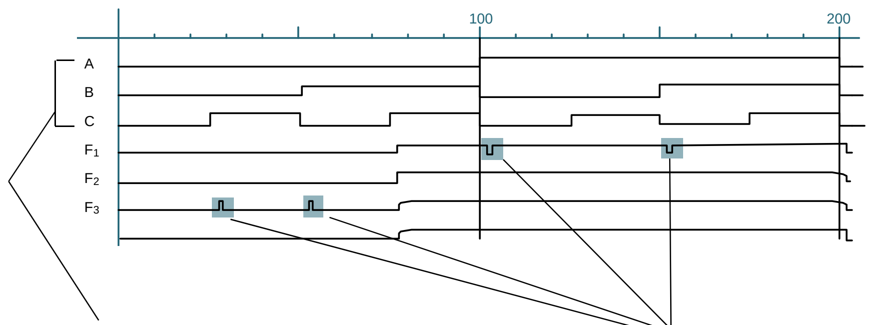
Minimized Sum of Products

Canonical Products of Sums

Minimized Products of Sums

Gate Logic: Two-Level Canonical Forms

Waveform Verification of the Four Alternatives



Eight Unique Combinations of Three Inputs

Except for *timing glitches*, output waveforms of the four implementations are essentially identical

Gate Logic: Two-Level Canonical Forms

Mapping Between Forms

1. Minterm to Maxterm conversion:
   rewrite minterm shorthand using maxterm shorthand
   replace minterm indices with the indices not already used

   E.g., $F(A,B,C) = \Sigma m(3,4,5,6,7) = \Pi M(0,1,2)$

2. Maxterm to Minterm conversion:
   rewrite maxterm shorthand using minterm shorthand
   replace maxterm indices with the indices not already used

   E.g., $F(A,B,C) = \Pi M(0,1,2) = \Sigma m(3,4,5,6,7)$

3. Minterm expansion of F to Minterm expansion of F':
   in minterm shorthand form, list the indices not already used in F

   E.g., $F(A,B,C) = \Sigma m(3,4,5,6,7) \longrightarrow F'(A,B,C) = \Sigma m(0,1,2)$
   $= \Pi M(0,1,2) \longrightarrow = \Pi M(3,4,5,6,7)$

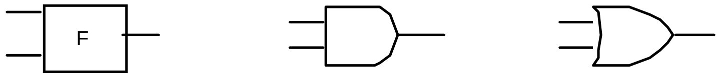
4. Minterm expansion of F to Maxterm expansion of F':
   rewrite in Maxterm form, using the same indices as F

   E.g., $F(A,B,C) = \Sigma m(3,4,5,6,7) \longrightarrow F'(A,B,C) = \Pi M(3,4,5,6,7)$
   $= \Pi M(0,1,2) \longrightarrow = \Sigma m(0,1,2)$

Gate Logic: Positive vs. Negative Logic

Normal Convention: Postive Logic/Active High
Low Voltage = 0; High Voltage = 1

Alternative Convention sometimes used: Negative Logic/Active Low



Voltage Truth T able — Positive Logic — Negative Logic

A	B	F
low	low	low
low	high	low
high	low	low
high	high	high

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

Behavior in terms of Electrical Levels

Two Alternative Interpretations
Positive Logic AND
Negative Logic OR

Dual Operations

Gate Logic: Positive vs. Negative Logic

Conversion from Positive to Negative Logic

Voltage Truth T able — Positive Logic — Negative Logic

A	B	F
low	low	high
low	high	low
high	low	low
high	high	low

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

A	B	F
1	1	0
1	0	1
0	1	1
0	0	1

Positive Logic NOR: $\overline{A + B} = \overline{A} \bullet \overline{B}$

Negative Logic NAND: $\overline{A \bullet B} = \overline{A} + \overline{B}$

Dual operations:
AND becomes OR, OR becomes AND
Complements remain unchanged

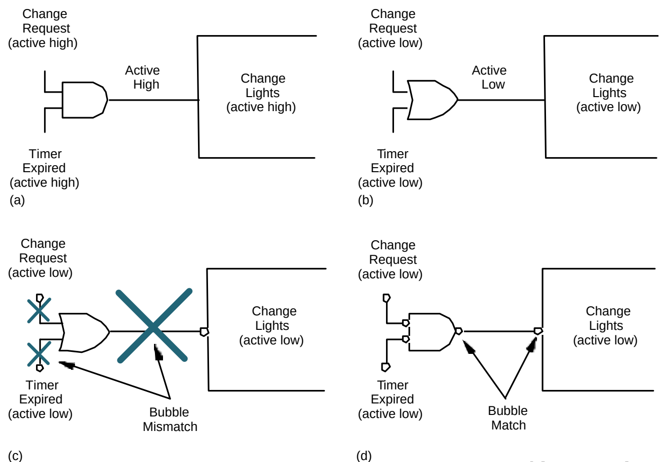
Gate Logic: Positive vs. Negative Logic

Practical Example

Use OR gate if input polarities are neg. logic

Use AND gate if active high

Mismatch between input and output logic polarities



Use NAND w/ inverted inputs if negative logic

Gate Logic: Incompletely Specified Functions

n input functions have 2^n possible input configurations

for a given function, not all input configurations may be possible

this fact can be exploited during circuit minimization!

E.g., Binary Coded Decimal Digit Increment by 1
BCD digits encode the decimal digits 0 - 9 in the bit patterns 0000_2 - 1001_2

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

Off-set of W

On-set of W

Don't care (DC) set of W

These input patterns should never be encountered in practise associated output values are "Don't Cares"

Gate Logic: Incompletely Specified Functions

Don't Cares and Canonical Forms

Canonical Representations of the BCD Increment by 1 Function:

Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15

Z = Σm(0, 2, 4, 6, 8) + d(10, 11, 12 ,13, 14, 15)

Z = M1 • M3 • M5 • M7 • M9 • D10 • D11 • D12 • D13 • D14 • D15

Z= ΠM(1, 3, 5, 7, 9) • D(10, 11, 12, 13, 14 ,15)

Gate Logic: Two-Level Simplification

Algebraic Simplification:

not an algorithm/systematic procedure

how do you know when the minimum realization has been found?

Computer-Aided Tools:

precise solutions require very long computation times, especially for functions with many inputs (>10)

heuristic methods employed —
"educated guesses" to reduce the amount of computation
good solutions not best solutions

Still Relevant to Learn Hand Methods:

insights into how the CAD programs work, and their strengths and weaknesses

ability to check the results, at least on small examples

don't have computer terminals during exams

Gate Logic: Two-Level Simplification

***Key Tool: The Uniting Theorem* — A (B' + B) = A**

A	B	F
0	0	0
0	1	0
1	0	1
1	1	1

F = A B' + A B = A (B' + B) = A

B's values change within the on-set rows

B is eliminated, A remains

A's values don't change within the on-set rows

A	B	G
0	0	1
0	1	0
1	0	1
1	1	0

G = A' B' + A B' = (A' + A) B' = B'

B's values stay the same within the on-set rows

A is eliminated, B remains

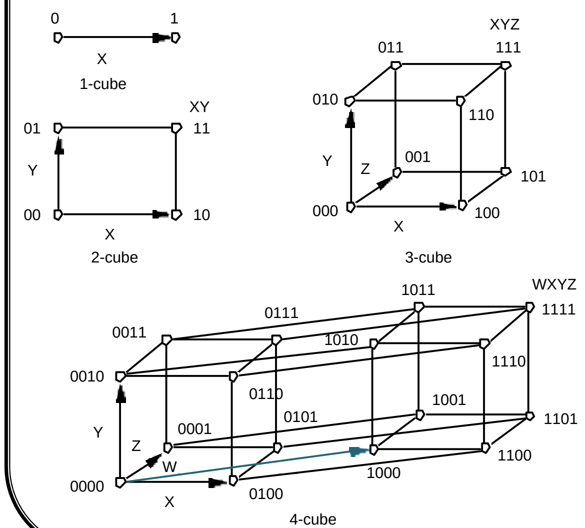
A's values change within the on-set rows

Essence of Simplification:
find two element subsets of the ON-set where only one variable changes its value. This single varying variable *can be eliminated!*

Gate Logic: Two-Level Simplification

Boolean Cubes

Visual technique for identifying when the Uniting Theorem can be applied



Just another way to represent the truth table

n input variables = n dimensional "cube"

Gate Logic: Two-Level Simplification

Mapping Truth Tables onto Boolean Cubes

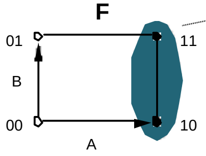
ON-set = filled-in nodes

OFF-set = empty nodes

DC-set = X'd nodes

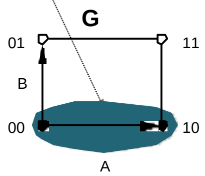
Cube of n-1 dimensions

Reduced expression
contains n-1 variables



A asserted and unchanged

B varies within loop



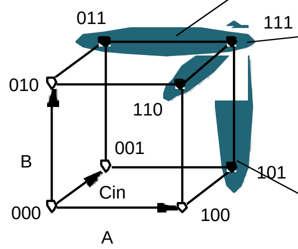
A varies within loop

B complemented and unchanged

Gate Logic: Two-Level Simplification

Three variable example: Full Adder Carry Out

A	B	Cin	Cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



(A' + A) B Cin

A B (Cin' + Cin)

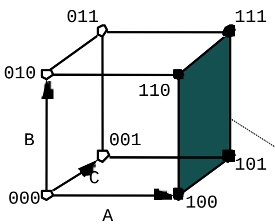
A (B + B') Cin

The ON-set is covered
by the OR of the subcubes
of lower dimensionality

Cout = B Cin + A B + A Cin

Gate Logic: Two-Level Simplification

Subcubes of Higher Dimensions than 2



F(A,B,C) = Σm(4,5,6,7)

On-set forms a rectangle,
i.e., a cube of two dimensions

represents an expression in one variable
i.e., 3 dimensions - 2 dimensions

A is asserted and unchanged
B and C vary

This subcube represents the
literal A

Gate Logic: Two-Level Simplification

In a 3-cube:

- a 0-cube, i.e., a single node, yields a term in three literals
- a 1-cube, i.e., a line of two nodes, yields a term in two literals
- a 2-cube, i.e., a plane of four nodes, yields a term in one literal
- a 3-cube, i.e., a cube of eight nodes, yields a constant term "1"

In general,

an m-subcube within an n-cube (m < n) yields a term with
n - m literals

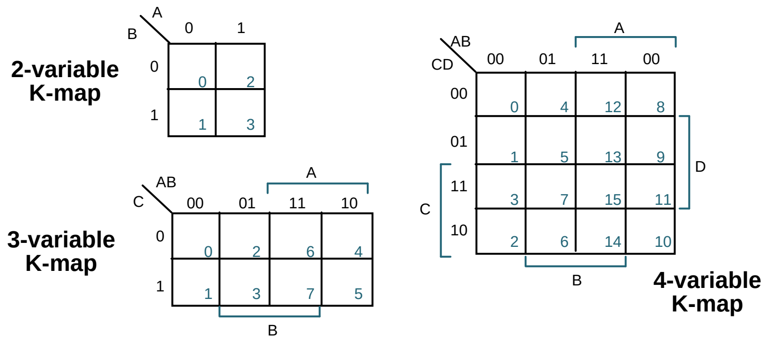
Gate Logic: Two-Level Simplification

Karnaugh Map Method

hard to draw cubes of more than 4 dimensions

K-map is an alternative method of representing the truth table that helps visualize adjacencies in up to 6 dimensions

Beyond that, computer-based methods are needed

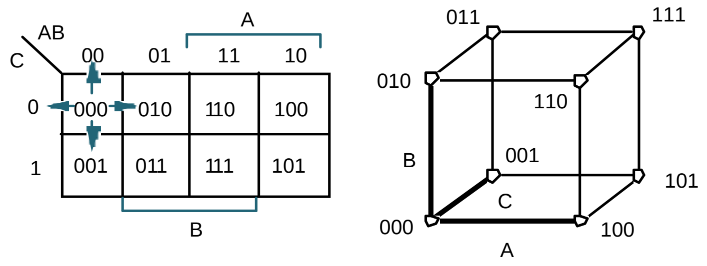


Numbering Scheme: 00, 01, 11, 10
Gray Code — only a single bit changes from code word to next code word

Gate Logic: Two-Level Simplification

Karnaugh Map Method

Adjacencies in the K-Map

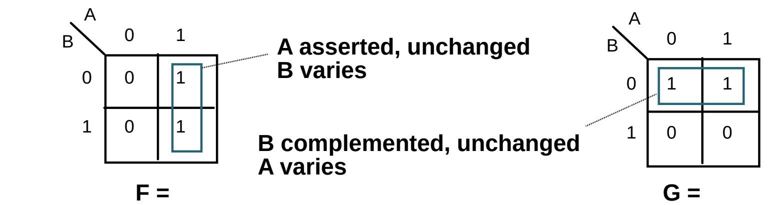


Wrap from first to last column

Top row to bottom row

Gate Logic: Two-Level Simplification

K-Map Method Examples

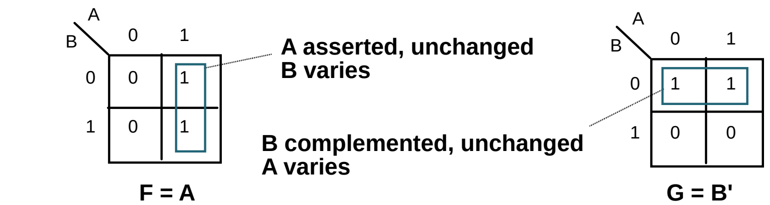


Cout =

F(A,B,C) =

Gate Logic: Two-Level Simplification

K-Map Method Examples



Cout = A B + B Cin + A Cin

F(A,B,C) = A

Gate Logic: Two-Level Simplification

More K-Map Method Examples, 3 Variables

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

F(A,B,C) = Σm(0,4,5,7)

F =

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

F' simply replace 1's with 0's and vice versa

F'(A,B,C) = Σm(1,2,3,6)

F' =

Gate Logic: Two-Level Simplification

More K-Map Method Examples, 3 Variables

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

F(A,B,C) = Σm(0,4,5,7)

F = B' C' + A C

In the K-map, adjacency wraps from left to right and from top to bottom

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

F' simply replace 1's with 0's and vice versa

F'(A,B,C) = Σm(1,2,3,6)

F' = B C' + A' C

Compare with the method of using DeMorgan's Theorem and Boolean Algebra to reduce the complement!

Gate Logic: Two-Level Simplification

K-map Method Examples: 4 variables

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

F(A,B,C,D) = Σm(0,2,3,5,6,7,8,10,11,14,15)

F =

Gate Logic: Two-Level Simplification

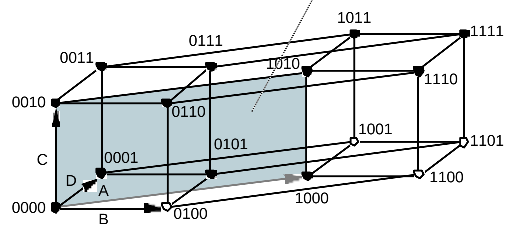
K-map Method Examples: 4 variables

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

F(A,B,C,D) = Σm(0,2,3,5,6,7,8,10,11,14,15)

F = C + A' B D + B' D'

Find the smallest number of the largest possible subcubes that cover the ON-set



K-map Corner Adjacency Illustrated in the 4-Cube

Gate Logic: Two-Level Simplification

K-map Method: Circling Zeros

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

$$F = (\overline{B} + C + D)\ (\overline{A} + C + \overline{D})\ (B + C + \overline{D})$$

Replace F by $\overline{F}$, 0's become 1's and vice versa

$$\overline{F} = B\,\overline{C}\,\overline{D} + A\,\overline{C}\,D + B\,C\,D$$

$$\overline{\overline{F}} = \overline{B\,\overline{C}\,\overline{D} + A\,\overline{C}\,D + B\,C\,D}$$

$$F = (\overline{B} + C + D)\ (\overline{A} + C + \overline{D})\ (B + C + \overline{D})$$

Gate Logic: Two-Level Simplification

K-map Example: Don't Cares

Don't Cares can be treated as 1's or 0's if it is advantageous to do so

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

$F(A,B,C,D) = \Sigma m(1,3,5,7,9) + \Sigma d(6,12,13)$

F = w/o don't cares

F = w/ don't cares

Gate Logic: Two-Level Simplification

K-map Example: Don't Cares

Don't Cares can be treated as 1's or 0's if it is advantageous to do so

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

$F(A,B,C,D) = \Sigma m(1,3,5,7,9) + \Sigma d(6,12,13)$

F = A'D + B' C' D w/o don't cares

F = C' D + A' D w/ don't cares

By treating this DC as a "1", a 2-cube can be formed rather than one 0-cube

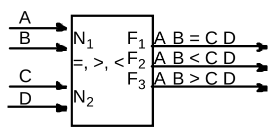
CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

In PoS form: F = D (A' + C')

Same answer as above, but fewer literals

Gate Logic: Two-Level Simplification

Design Example: Two Bit Comparator



A	B	C	D	F_1	F_2	F_3
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

Block Diagram and Truth Table

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 F_1

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 F_2

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 F_3

F1 =

F2 =

F3 =

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 F_1

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 F_2

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 F_3

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

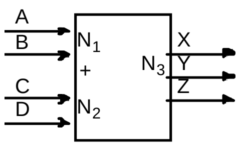
F2 = A' B' D + B' C D + A' C

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

A xnor B xnor C xnor D

Gate Logic: Two-Level Simplification

Design Example: Two Bit Adder



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

Block Diagram and Truth Table

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

Gate Logic: Two-Level Simplification

Design Example (Continued)

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 =

Z =

Y =

Gate Logic: Two-Level Simplification

Design Example (Continued)

K-map for X

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 Y

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 Z

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

1's on diagonal suggest XOR!
Y K-Map not minimal as drawn

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

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

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

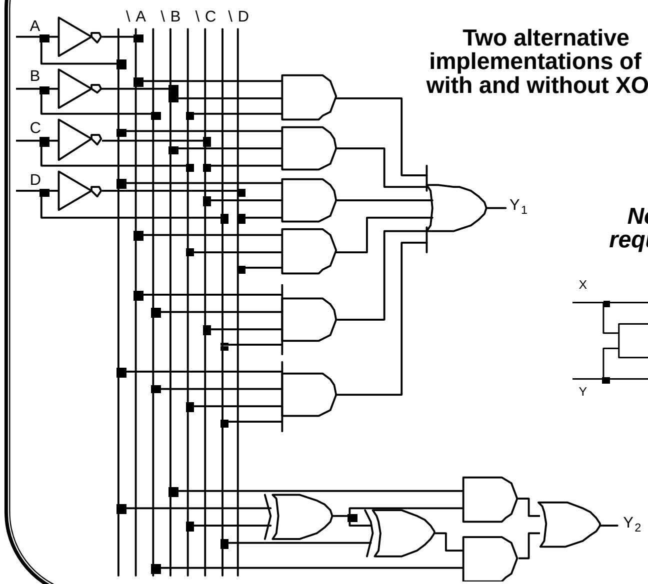
= B' (A xor C) + A' B (C xor D) + A B (C xnor D)

= B' (A xor C) + B (A xor B xor C)

gate count reduced if XOR available

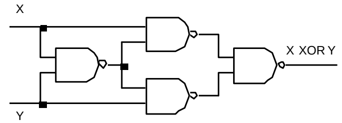
Gate Logic: Two-Level Simplification

Design Example (Continued)



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

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

X

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

Y

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

Z

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

W =

X =

Y =

Z =

Gate Logic: Two-Level Simplification

W

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

X

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

Y

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

Z

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

W = B C D + A D'

X = B C' + B D' + B' C D

Y = A' C' D + C D'

Z = D'

Gate Logic: Two Level Simplification

Definition of Terms

implicant: single element of the ON-set or any group of elements that can be combined together in a K-map

prime implicant: implicant that cannot be combined with another implicant to eliminate a term

essential prime implicant: if an element of the ON-set is covered by a single prime implicant, it is an essential prime

Objective:

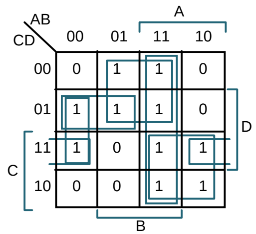
grow implicants into prime implicants

cover the ON-set with as few prime implicants as possible

essential primes participate in ALL possible covers

Gate Logic: Two Level Simplification

Examples to Illustrate Terms

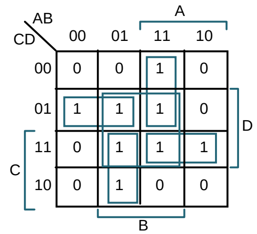


6 Prime Implicants:

A' B' D, B C', A C, A' C' D, A B, B' C D

essential

Minimum cover = B C' + A C + A' B' D



5 Prime Implicants:

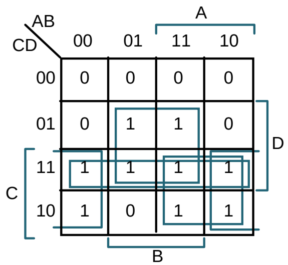
B D, A B C', A C D, A' B C, A' C' D

essential

Essential implicants form minimum cover

Gate Logic: Two Level Simplification

More Examples



Prime Implicants:

B D, C D, A C, B' C

essential

Essential primes form the minimum cover

Gate Logic: Two-Level Simplification

Algorithm: Minimum Sum of Products Expression from a 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, and corner adjacencies. This forms ***prime implicants*** (always a power of 2 number of elements).

Repeat Steps 1 and 2 to find all prime implicants

Step 3: Revisit 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

Gate Logic: Two Level Simplification

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

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

Initial K-map

Gate Logic: Two Level Simplification

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

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

Initial K-map

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

Primes around A' B C' D'

Gate Logic: Two Level Simplification

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

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

Initial K-map

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

Primes around A' B C' D'

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

Primes around A B C' D

Gate Logic: Two-Level Simplification

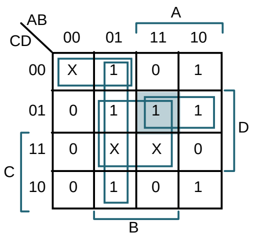
Example Continued

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

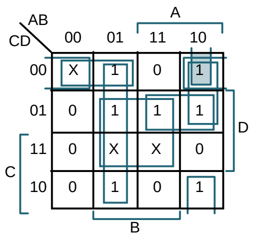
Primes around A B C' D

Gate Logic: Two-Level Simplification

Example Continued



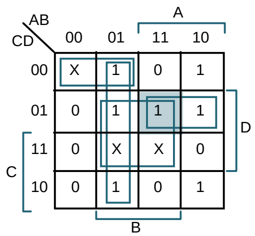
Primes around A B C' D



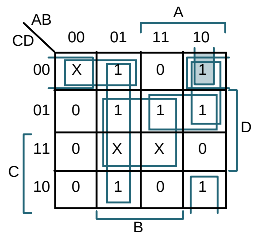
Primes around A B' C' D'

Gate Logic: Two-Level Simplification

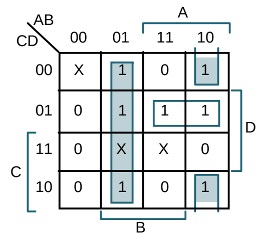
Example Continued



Primes around A B C' D



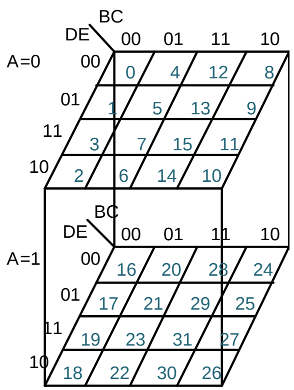
Primes around A B' C' D'

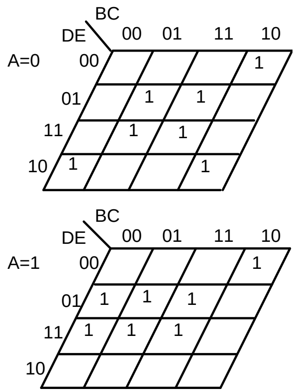


Essential Primes with Min Cover

Gate Logic: Two-Level Simplification

5-Variable K-maps



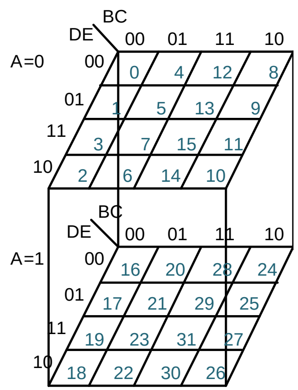


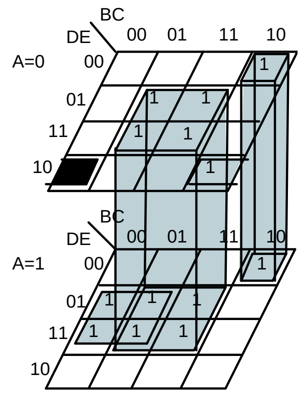
f(A,B,C,D,E) = Σm(2,5,7,8,10, 13,15,17,19,21,23,24,29 31)

=

Gate Logic: Two-Level Simplification

5-Variable K-maps



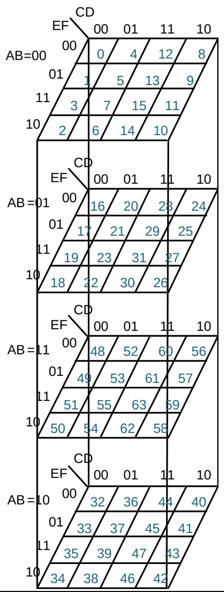


f(A,B,C,D,E) = Σm(2,5,7,8,10, 13,15,17,19,21,23,24,29 31)

= C E + A B' E + B C' D' E' + A' C' D E'

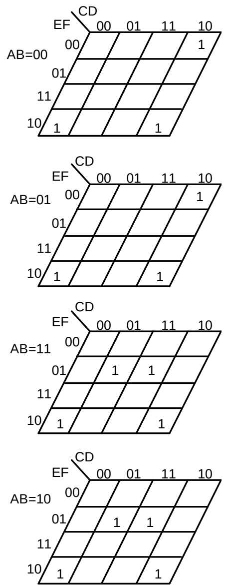
Gate Logic: Two Level Simplification

6- Variable K-Maps



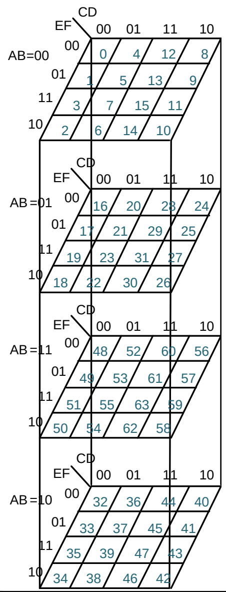
ƒ(A,B,C,D,E,F) =
Σm(2,8,10,18,24,
26,34,37,42,45,50,
53,58,61)

=



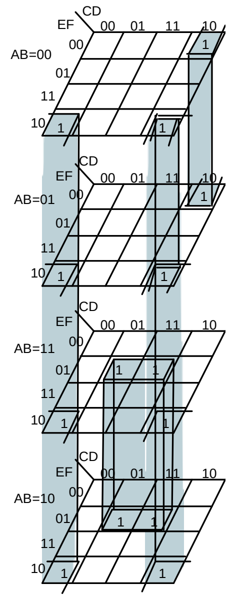
Gate Logic: Two Level Simplification

6- Variable K-Maps



ƒ(A,B,C,D,E,F) =
Σm(2,8,10,18,24,
26,34,37,42,45,50,
53,58,61)

= D' E F' + A D E' F
+ A' C D' F'



Gate Logic: CAD Tools for Simplification

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

ƒ(A,B,C,D) = Σm(4,5,6,8,9,10,13) + Σd(0,7,15)

Stage 1: Find all prime implicants

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

Implication Table

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

Gate Logic: CAD Tools for Simplification

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

ƒ(A,B,C,D) = Σm(4,5,6,8,9,10,13) + Σd(0,7,15)

Stage 1: Find all prime implicants

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

Step 2: Apply Uniting Theorem— Compare elements of group w/ N 1's against those with N+1 1's. 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	
	- 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	

Repeat until no further combinations can be made.

Gate Logic: CAD Tools for Simplification

Quine-McCluskey Method

Tabular method to systematically find all prime implicants

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

Stage 1: Find all prime implicants

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

Step 2: Apply Uniting Theorem— Compare elements of group w/ N 1's against those with N+1 1's. 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.

Repeat until no further combinations can be made.

Implication Table

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

Gate Logic: CAD Tools for 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'     -000 = B' C' D'
100- = A B' C'      10-0 = A B' D'
1-01 = A C' D       01-- = A' B
-1-1 = B D
```


Gate Logic: CAD Tools for 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'     -000 = B' C' D'
100- = A B' C'      10-0 = A B' D'
1-01 = A C' D       01-- = A' B
-1-1 = B D
```

Stage 2: find smallest set of prime implicants that cover the ON-set

recall that essential prime implicants must be in all covers

another tabular method– the prime implicant chart

Gate Logic: CAD Tools for Simplification

Prime Implicant Chart

	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

rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

Gate Logic: CAD Tools for Simplification

Prime Implicant Chart

	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

rows = prime implicants
columns = ON-set elements
place an "X" if ON-set element is covered by the prime implicant

	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

If column has a single X, than the implicant associated with the row is essential. It must appear in minimum cover

Gate Logic: CAD Tools for Simplification

Prime Implicant Chart (Continued)

	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

Eliminate all columns covered by essential primes

Gate Logic: CAD Tools for Simplification

Prime Implicant Chart (Continued)

	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

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

Find minimum set of rows that cover the remaining columns

f = A B' D' + A C' D + A' B

Gate Logic: CAD Tools for Simplification

ESPRESSO Method

Problem with Quine-McCluskey: the 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

Espresso: trades solution speed for minimality of answer

don't generate *all* prime implicants (Quine-McCluskey Stage 1)

judiciously select a subset of primes that still covers the ON-set

operates in a fashion not unlike a human finding primes in a K-map

Gate Logic: CAD Tools for 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 (i.e., no proper subset is also a cover) is extracted from the expanded primes
   Just like the Quine-McCluskey Prime Implicant Chart
   Step is called IRREDUNDANT 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/IRREDUNDANT 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: CAD Tools for Simplification

Espresso Inputs and Outputs

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

Espresso 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
```

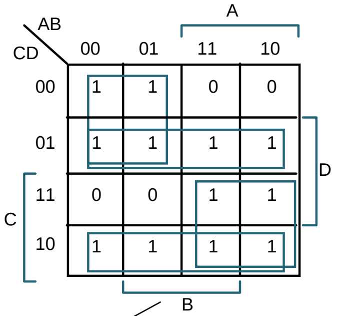
Espresso Output

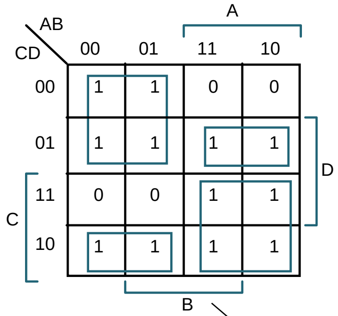
```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01   1
10-0   1
01--   1
.e
```

f = A C' D + A B' D' + A' B

Gate Logic: CAD Tools for Simplification

Espresso: Why Iterate on Reduce, Irredundant Cover, Expand?





Initial Set of Primes found by Steps1 and 2 of the Espresso Method

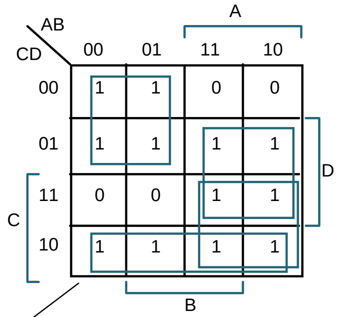
4 primes, irredundant cover, but not a minimal cover!

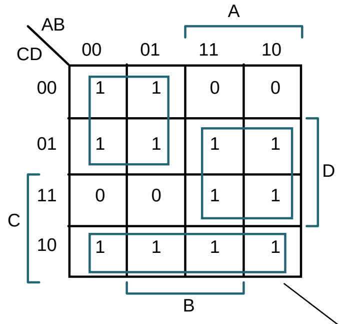
Result of REDUCE: Shrink primes while still covering the ON-set

Choice of order in which to perform shrink is important

Gate Logic: CAD Tools for Simplification

Espresso Iteration (Continued)





Second EXPAND generates a different set of prime implicants

IRREDUNDANT COVER found by final step of espresso

Only three prime implicants!

Two-Level Logic: 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 up to 6 variables

or Quine-McCluskey Algorithm

or Espresso CAD Tool