# Digital Logic Design

Ch 2

## Combinational Logic

# Topics

- Combinational logic definition
- Axioms and Theorems of Boolean algebra
- Realizing Boolean formulas
- Levels of logic expressions
  - Two-level logic
  - Canonical forms
  - Multi-level logic
- Simplifying two-level logic expressions
  - Boolean cubes
  - Karnaugh maps

# Combinational logic

- Define
  - the kind of digital system whose output behavior depends only on the current inputs ➔ each output is defined as a function (combination) of inputs
  - memoryless: its outputs are independent of the historical sequence of values presented to it as inputs
  - (cf.) sequential logic
- Many ways to describe combination logic
  - Boolean algebra expression
  - wired up logic gates
  - truth tables tabulating input and output combinations
  - graphical maps
  - program statements in a hardware description language

# Examples of combinational logic

- The equivalence circuit

| X | Y | Equal |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

- The tally circuit

| X | Y | Zero | One | Two |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

- Binary Adder

| X | Y | Cout | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

< Half-adder>

| X | Y | Cin | Cout | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

< Full-adder>

# Laws and theorems of Boolean logic

- Basic concept
  - Boolean algebra is the mathematical foundation of digital systems
  - laws (axioms) : the properties to which the operations of Boolean algebra must adhere
  - axioms can be used to prove more general laws
- Boolean operations
  - operation order

    COMPLEMENT → AND → OR
  - parentheses : change the default order of evaluation
  - examples :

$$1)\ \overline{A} \bullet B + C = \left(\left(\overline{\mathrm{A}}\right) \bullet B\right) + C$$

# Axioms of Boolean algebra

- A Boolean algebra consists of
  - a set of elements B
  - binary operations { + , • }
  - and a unary operation { ’ }
  - such that the following axioms hold (Huntington's postulates):
- Axioms

A1. the set B contains at least two elements: a, b

| | | |
|---|---|---|
| A2. closure: | a + b is in B | a • b is in B |
| A3. commutativity: | a + b = b + a | a • b = b • a |
| A4. associativity: | a + (b + c) = (a + b) + c | a • (b • c) = (a • b) • c |
| | = a + b + c | = a • b • c |
| A5. identity: | a + 0 = a | a • 1 = a |
| A6. distributivity: | a + (b • c) = (a + b) • (a + c) | a • (b + c) = (a • b) + (a • c) |
| A7. complementarity: | a + a’ = 1 | a • a’ = 0 |

# Theorems of Boolean algebra

- Operations with 0 and 1 (Unit/Zero properties)
  1. $X + 0 = X$ — 1D. $X \bullet 1 = X$
  2. $X + 1 = 1$ — 2D. $X \bullet 0 = 0$
- Idempotent theorem:
  3. $X + X = X$ — 3D. $X \bullet X = X$
- Involution theorem: 4. $(X')' = X$
- Theorem of complementarity:
  5. $X + X' = 1$ — 5D. $X \bullet X' = 0$
- Commutative law:
  6. $X + Y = Y + X$ — 6D. $X \bullet Y = Y \bullet X$
- Associative law:
  7. $(X + Y) + Z = X + (Y + Z) = X + Y + Z$ — 7D. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z) = X \bullet Y \bullet Z$

# Theorems of Boolean algebra

- Distributive law:
  - 8. $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$  8D. $X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$
- Simplification theorems:
  - 9. $X \bullet Y + X \bullet Y' = X$  9D. $(X + Y) \bullet (X + Y') = X$
  - 10. $X + X \bullet Y = X$  10D. $X \bullet (X + Y) = X$
  - 11. $(X + Y') \bullet Y = X \bullet Y$  11D. $(X \bullet Y') + Y = X + Y$
- DeMorgan's law:
  - 12. $(X + Y + Z + \ldots)' = X' \bullet Y' \bullet Z' \bullet \ldots$  12D. $(X \bullet Y \bullet Z \bullet \ldots)' = X' + Y' + Z' + \ldots$
- General form:
  - 13. $\{f(X1, X2, \ldots, Xn, 0, 1, +, \bullet)\}' = \{f(X1', X2', \ldots, Xn', 1, 0, \bullet, +)\}$

# Theorems of Boolean algebra

- Duality:

  14. $(X + Y + Z + \ldots)^D = X \bullet Y \bullet Z \bullet \ldots$

  14D. $(X \bullet Y \bullet Z \bullet \ldots)^D = X + Y + Z + \ldots$

- General form:

  15. $\{f(X_1,X_2,\ldots,X_n,0,1,+,\bullet)\}^D = f(X_1,X_2,\ldots,X_n,1,0,\bullet,+)$

- Theorem for multiplying and factoring

  16. $(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$

  $= XX'+\underline{XZ+X'Y+YZ}$

  $X \Rightarrow Z$

  16D. $X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$

  $= (X+X')(X+Z)(X'+Y)(Y+Z)$

- Consensus theorem:

  17. $X \bullet Y + Y \bullet Z + X' \bullet Z = X \bullet Y + X' \bullet Z$

  17D. $(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$

Proof ➔ $XY+X'Z+YZ = XY+X'Z+(X+X')YZ = XY+X'Z+XYZ+X'YZ = X(Y+YZ)+X'(Z+YZ)$

# Verifying the Boolean theorems

- Verifying the theorems using the axioms of Boolean algebra
- Proving the uniting theorem(9): $\mathbf{X \bullet Y + X \bullet Y' = X}$

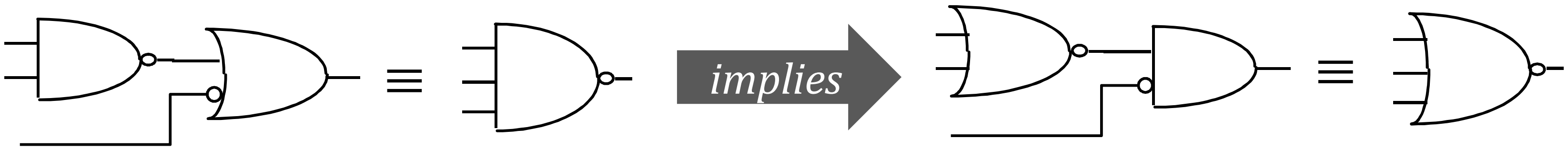
| | | |
|---|---|---|
| Distributive law (8) | $X \bullet (Y + Y')$ | $= X$ |
| Complementarity theorem (5) | $X \bullet (1)$ | $= X$ |
| Identity (1D) | $X$ | $= X$ |

- Proving the simplification theorem(10): $\mathbf{X + X \bullet Y = X}$

| | | |
|---|---|---|
| Identity (1D) | $X \bullet 1 + X \bullet Y$ | $= X$ |
| Distributive law (8) | $X(1 + Y)$ | $= X$ |
| Identity (2) | $X(1)$ | $= X$ |
| Identity (1) | $X$ | $= X$ |

# Duality vs. DeMorgan's law

- Duality
  - a dual of a Boolean expression is derived by replacing $\bullet$ by +, + by $\bullet$, 0 by 1, and 1 by 0, and leaving variables unchanged
  - Any theorem that can be proven is thus also proven for its dual!



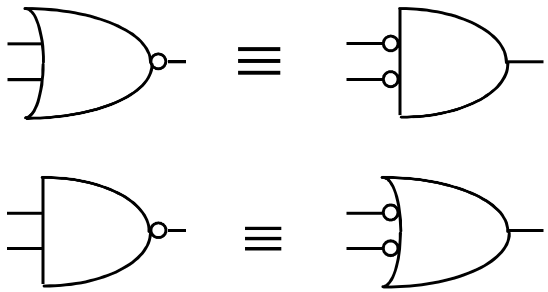

  - A meta-theorem (a theorem about theorems) that allow to derive new theorems: (e.g.) the dual of the uniting theorem(9), $X \bullet Y + X \bullet Y' = X$, is $(X + Y) \bullet (X + Y') = X$. ➔ *The proof of the dual follows step-by-step, simply using the duals of the laws used in the original proof.*

**$(X + Y) \bullet (X + Y') = X?$**

| | | |
|---|---|---|
| $X + (Y \bullet Y')$ | $= X$ | Distributive law (8D) |
| $X + 0$ | $= X$ | Complementarity theorem (5D) |
| $X$ | $= X$ | Identity (1) |

# Duality vs. DeMorgan's law

- DeMorgan's law
  - $(a+b)' \equiv a'b'$
    $(ab)' \equiv a'+b'$



  - give a procedure for complementing a complex function
  - the complemented expression is derived by replacing all literals by their complements, 0 by 1, 1 by 0, • by + and + by •
  - (e.g.) the complement of $Z = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C}$
    $\bar{Z} = (\overline{\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C}})$
    $\bar{Z} = \overline{\bar{A}\bar{B}C} \bullet \overline{\bar{A}BC} \bullet \overline{A\bar{B}C} \bullet \overline{AB\bar{C}}$
    $\bar{Z} = (A + B + \bar{C})\ (A + \bar{B} + \bar{C})\ (\bar{A} + B + \bar{C})\ (\bar{A} + \bar{B} + C)$

# Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:

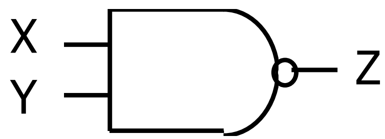

| X | Y | 16 possible functions (F0–F15) | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 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 and Y | NOT (X implies Y) | X | NOT (Y implies X) | Y | X xor Y | X or Y | X nor Y<br>not (X or Y) | X = Y | not Y | Y implies X | not X | X implies Y | X nand Y<br>not (X and Y) | 1 |

# Cost of different logic functions

- Different functions are cheaper or more expensive to implement in CMOS technology
  - Each has a cost associated with the number of switches needed
  - 0(F0) and 1(F15): require **0** switches, directly connect output to low/high
  - X(F3) and Y(F5): require **0** switches, output is one of inputs
  - X'(F12) and Y'(F10): require **2** switches for "inverter" or NOT-gate
  - X nor Y (F4) and X nand Y (F14): require **4** switches
  - X or Y (F7) and X and Y (F1): require **6** switches →
  - X = Y (F9) and X ⊕ Y (F6): require **16** switches →
- Thus…
  - A simpler expression for a Boolean function does not necessarily minimize the number of transistors for its realization.
  - Because NOT, NOR, and NAND are the cheapest, they are the functions that we implement the most in practice.

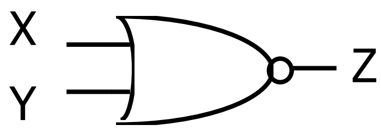
# Realizing Boolean formulas: logic gates

- NAND



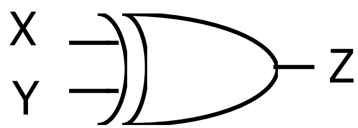

| X | Y | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- NOR




| X | Y | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

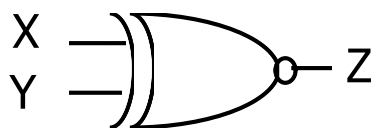
- XOR
  $X \oplus Y$




| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

X <u>xor</u> Y = X Y′ + X′ Y
X or Y but not both
("inequality", "difference")

- XNOR
  $X = Y$

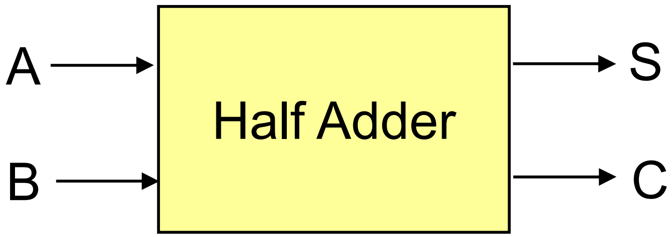


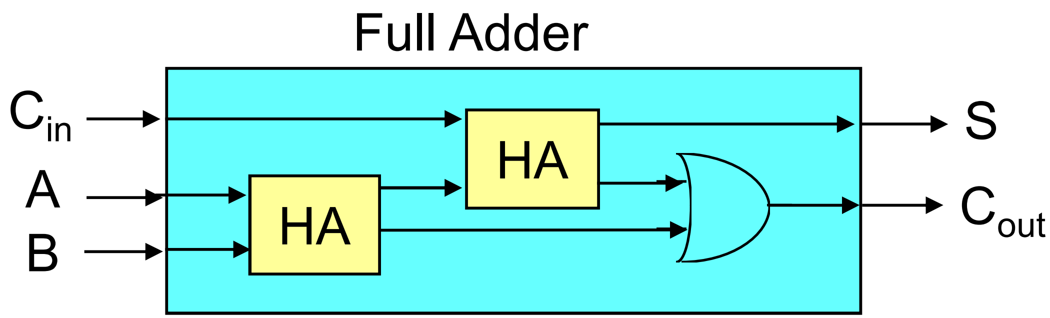

| X | Y | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

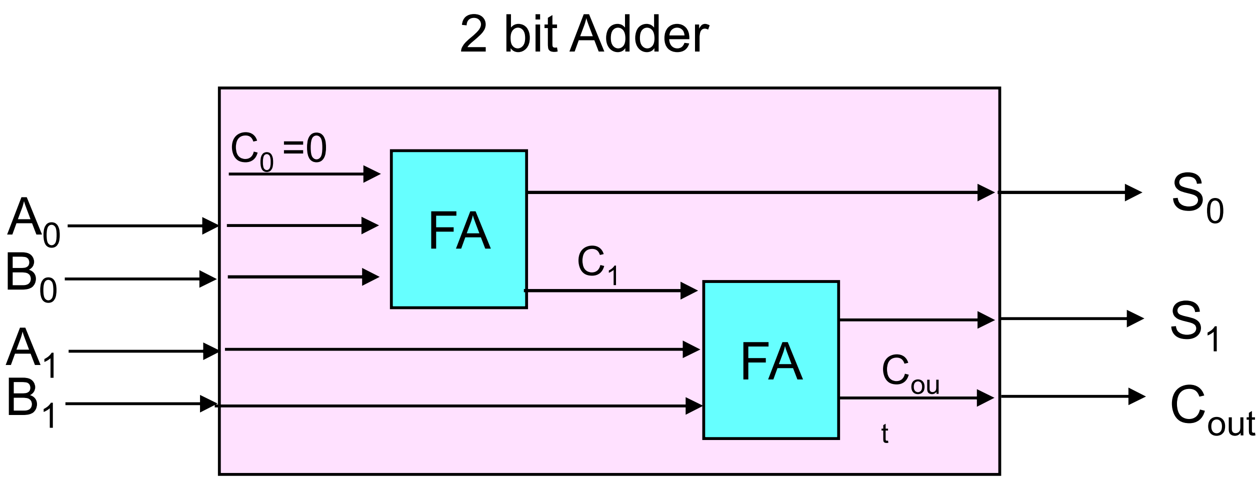
X <u>xnor</u> Y = X Y + X′ Y′
X and Y are the same
("equality", "coincidence")

# Realizing Boolean formulas: logic blocks & hierarchy

- Complex logic function can be constructed from more primitive functions by wiring up logic gates
- example : 2-bit adder

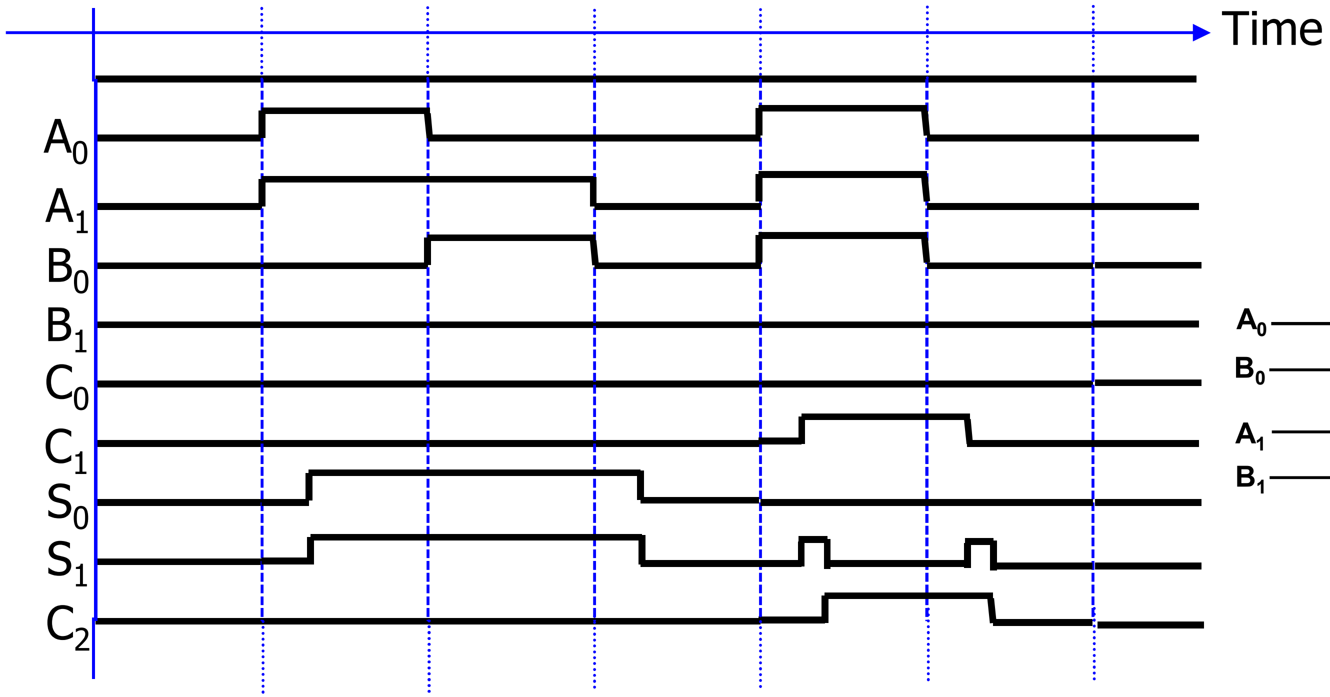


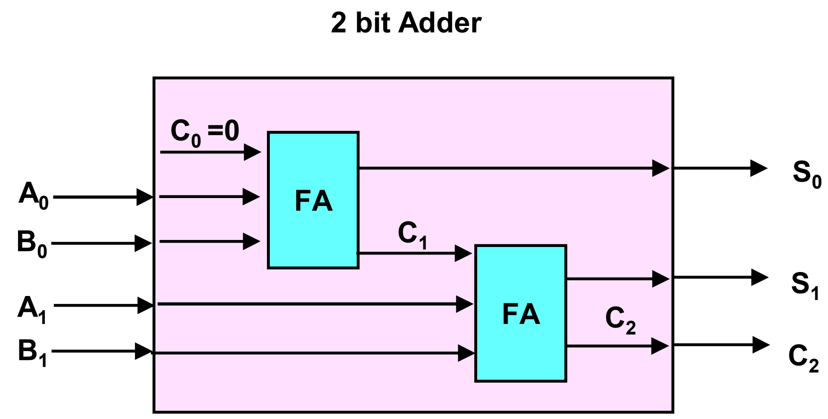



# Time behavior and waveforms

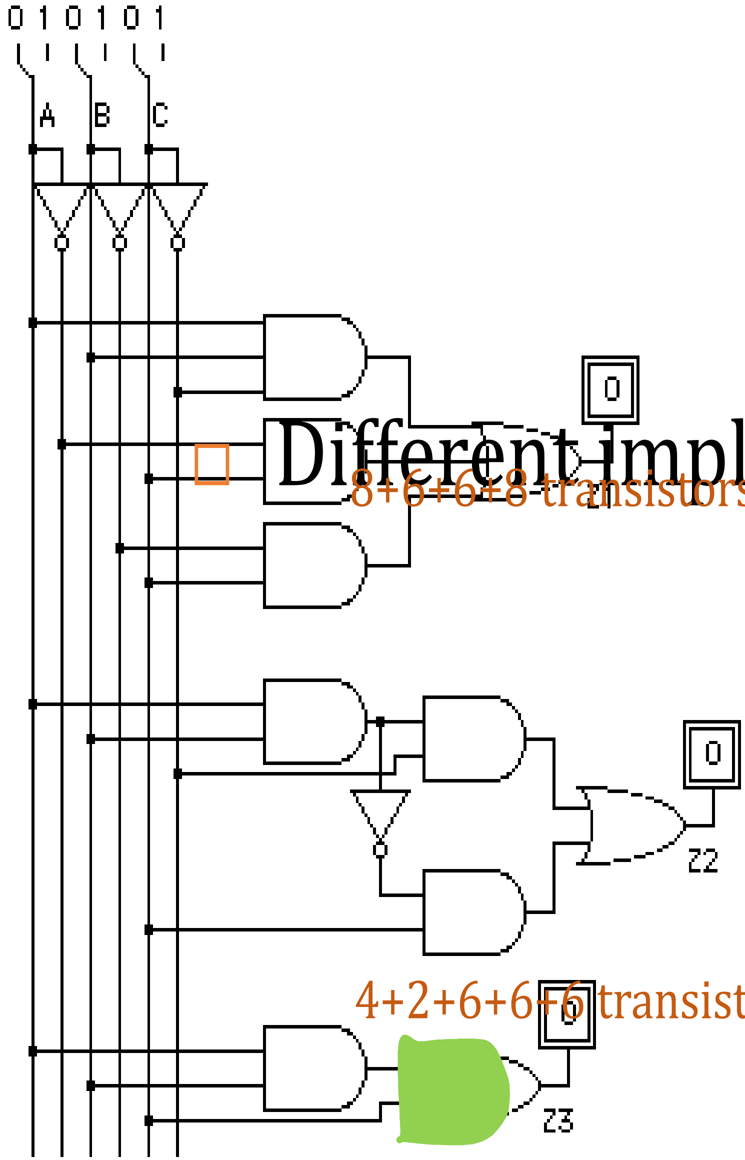
- Waveform: represent signal propagation over time
  - x-axis: the time step
  - y-axis: the logical value
- Unit delay model: considering the delay through any gate as taking exactly one time unit for a simplifying assumption



# Minimizing the number of gates & wires

| 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 |




- Different implementations of one f

8+6+6+8 transistors

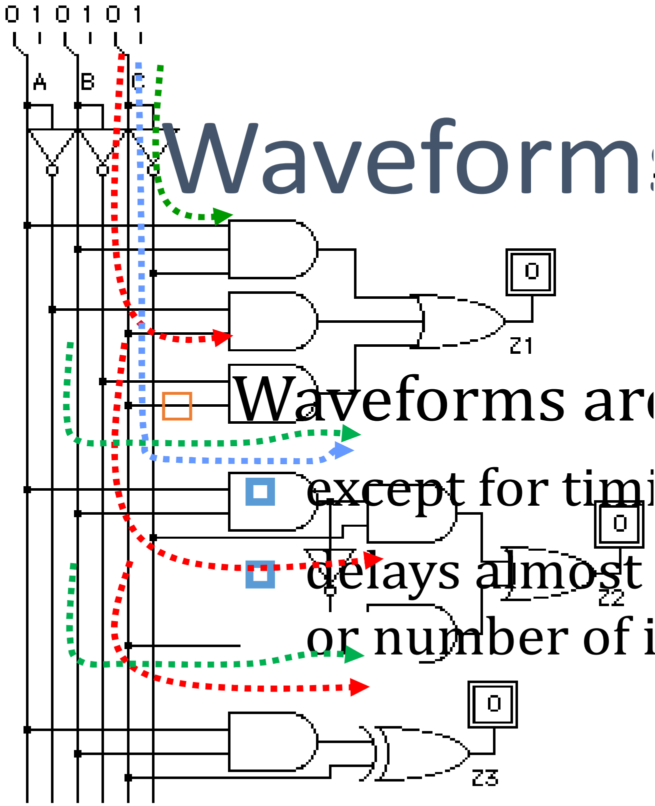
4+2+6+6+6 transistors

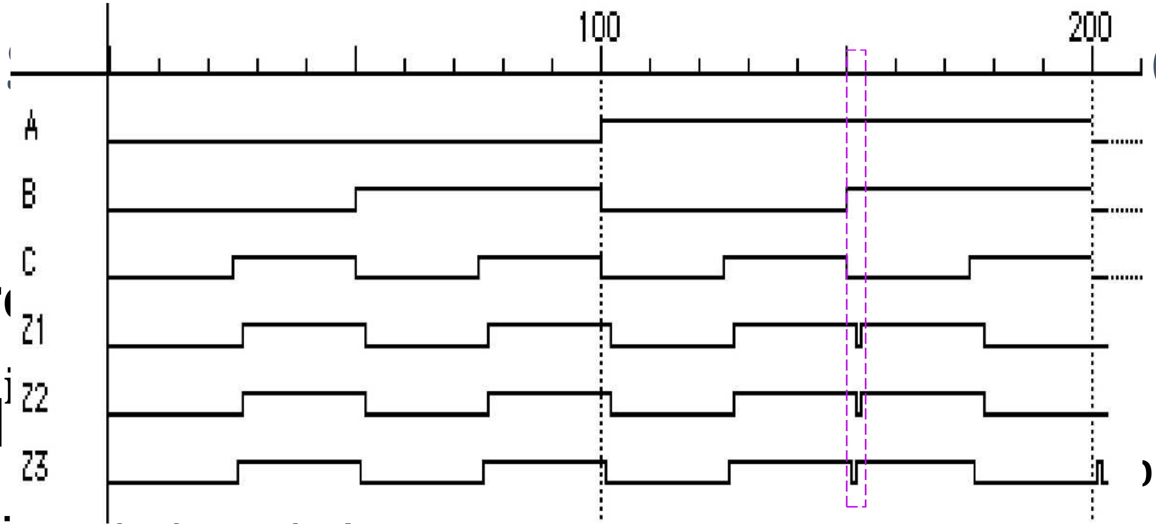
$= ABC' + (A' + B')C$

$= ABC' + (AB)'C$

$= T \oplus C$

# Waveform on




- Waveforms ar
  - except for timi
  - delays almost pe of g
    or number of inputs to gate)

# Two-level logic

- Nested levels to express Boolean logics
  - One-level: $Z = A + B$, $F = AB'$, $G = A'B$
  - Two-level: $Y = F + G = AB' + A'B$
- Two-level are sufficient to represent any Boolean expression.
  - **Canonical form**: standard form to represent a Boolean expression
  - unique algebraic signature of the function
- Two alternative canonical forms
  - sum-of-products: $AB' + A'B$
  - product-of-sums: $(A + B')(A' + B)$
- Incompletely specified function
  - In the original form, all $2^n$ possible input combinations must be considered for a function with $n$ inputs.
  - For flexibility, we consider one more set: don't care set

# Sum-of-products canonical forms

- a.k.a. *minterm expansion* or a *disjunctive normal form* (**DNF**)
- An expression in DNF consists of an ORed list of minterms.

F = *001* *011* *101* *110* *111*

F = A′B′C + A′BC + AB′C + ABC′ + ABC

| A | B | C | F | F′ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

F′ = A′B′C′ + A′BC′ + AB′C′

# Sum-of-products canonical forms

- Product term (or minterm)
  - ANDed product of literals – input combination for which output is true
  - each variable appears exactly once, true or inverted (but not both)

| A | B | C | minterms | |
|---|---|---|---|---|
| 0 | 0 | 0 | A′B′C′ | m0 |
| 0 | 0 | 1 | A′B′C | m1 |
| 0 | 1 | 0 | A′BC′ | m2 |
| 0 | 1 | 1 | A′BC | m3 |
| 1 | 0 | 0 | AB′C′ | m4 |
| 1 | 0 | 1 | AB′C | m5 |
| 1 | 1 | 0 | ABC′ | m6 |
| 1 | 1 | 1 | ABC | m7 |

short-hand notation for minterms of 3 variables

F in canonical form:

$$
\begin{aligned}
F(A, B, C) &= \Sigma m(1,3,5,6,7) \\
&= m1 + m3 + m5 + m6 + m7 \\
&= A'B'C + A'BC + AB'C + ABC' + ABC
\end{aligned}
$$

canonical form $\neq$ minimal form

$$
\begin{aligned}
F(A, B, C) &= A'B'C + A'BC + AB'C + ABC + ABC' \\
&= (A'B' + A'B + AB' + AB)C + ABC' \\
&= ((A' + A)(B' + B))C + ABC' \\
&= C + ABC' \\
&= ABC' + C \\
&= AB + C
\end{aligned}
$$

# Product-of-sums canonical forms

- a.k.a, *maxterm expansion* or a *conjunctive normal form* (CNF)
- An expression in CNF consists of an ANDed list of maxterms

| A | B | C | F | F′ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

F = *000* *010* *100*

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′)

# Product-of-sums canonical forms

- Sum term (or maxterm)
  - ORed sum of literals – input combination for which output is false
  - each variable appears exactly once, true or inverted (but not both)

| A | B | C | maxterms | |
|---|---|---|---|---|
| 0 | 0 | 0 | A+B+C | M0 |
| 0 | 0 | 1 | A+B+C′ | M1 |
| 0 | 1 | 0 | A+B′+C | M2 |
| 0 | 1 | 1 | A+B′+C′ | M3 |
| 1 | 0 | 0 | A′+B+C | M4 |
| 1 | 0 | 1 | A′+B+C′ | M5 |
| 1 | 1 | 0 | A′+B′+C | M6 |
| 1 | 1 | 1 | A′+B′+C′ | M7 |

short-hand notation for maxterms of 3 variables

F in canonical form:

$$
\begin{aligned}
F(A, B, C) &= \Pi M(0,2,4) \\
&= M0 \bullet M2 \bullet M4 \\
&= (A + B + C)\,(A + B' + C)\,(A' + B + C)
\end{aligned}
$$

canonical form $\neq$ minimal form

$$
\begin{aligned}
F(A, B, C) &= (A + B + C)\,(A + B' + C)\,(A' + B + C) \\
&= (A + B + C)\,(A + B' + C) \\
&\quad\ (A + B + C)\,(A' + B + C) \\
&= (A + C)\,(B + C)
\end{aligned}
$$

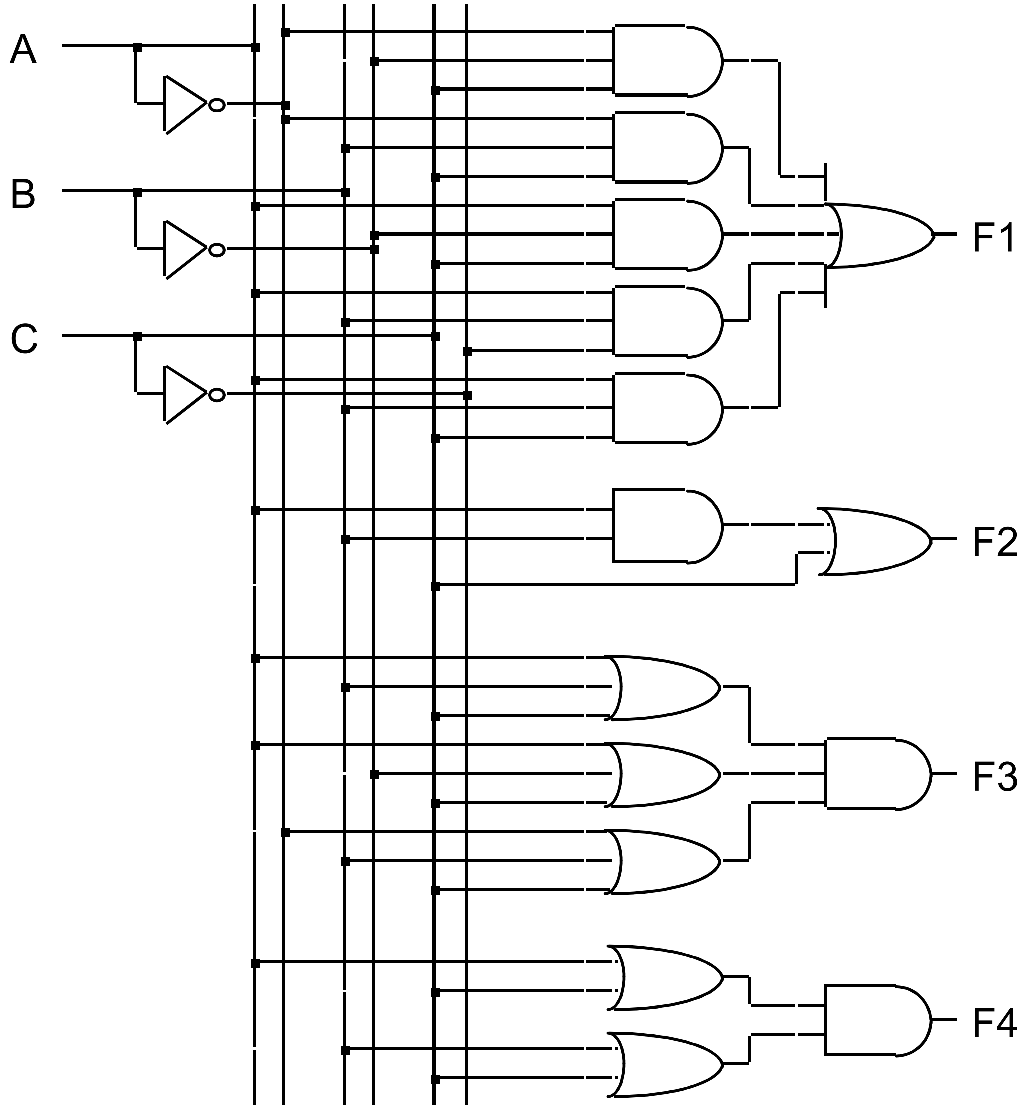
# S-o-P, P-o-S and DeMorgan's law

- Sum-of-products
  - $F' = A'B'C' + A'BC' + AB'C'$
- Apply DeMorgan's
  - $(F')' = (A'B'C' + A'BC' + AB'C')'$
  - $F = (A + B + C)\ (A + B' + C)\ (A' + B + C)$

- Product-of-sums
  - $F' = (A + B + C')\ (A + B' + C')\ (A' + B + C')\ (A' + B' + C)\ (A' + B' + C')$
- Apply DeMorgan's
  - $(F')' = (\ (A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C')\ )'$
  - $F = A'B'C + A'BC + AB'C + ABC' + ABC$

# 4 alternative two-level implementations of F

Note
A + A' = 1
A'B'+A'B+AB'+AB = 1




canonical sum-of-products

F1 = A'B'C + A'BC + AB'C + ABC' + ABC

minimized sum-of-products

F2 = AB + C

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

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

canonical product-of-sums

F3 = (A + B + C)(A + B' + C)(A' + B + C)

minimized product-of-sums

F4 = (A + C)(B + C)

# Conversion between canonical forms

- Minterm to maxterm conversion
  - use maxterms whose indices do not appear in DNF
  - e.g., $F(A,B,C) = \Sigma m(1,3,5,6,7) = \Pi M(0,2,4)$
- Maxterm to minterm conversion
  - use minterms whose indices do not appear in CNF
  - e.g., $F(A,B,C) = \Pi M(0,2,4) = \Sigma m(1,3,5,6,7)$
- DNF of F to DNF of F'
  - use minterms whose indices do not appear
  - e.g., $F(A,B,C) = \Sigma m(1,3,5,6,7)$ $F'(A,B,C) = \Sigma m(0,2,4)$
- CNF of F to CNF of F'
  - use maxterms whose indices do not appear
  - e.g., $F(A,B,C) = \Pi M(0,2,4)$ $F'(A,B,C) = \Pi M(1,3,5,6,7)$

# Incompletely specified functions

- Example: *binary coded decimal* (BCD) increment by 1
  - BCD digits encode the decimal digits 0 – 9 in the bit patterns 0000 – 1001

| 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 inputs patterns should never be encountered in practice – **"don't care"** about associated output values, can be exploited in minimization

# Notation for incompletely specified functions

- Don't cares and canonical forms
  - so far, only represented on-set
  - also represent don't-care-set
  - need two of the three sets (on-set, off-set, dc-set)
- Canonical representations of the BCD increment by 1 function:
  - Z = m0 + m2 + m4 + m6 + m8
    + d10 + d11 + d12 + d13 + d14 + d15
    = Σ [ m(0,2,4,6,8) + d(10,11,12,13,14,15) ]
  - Z = M1 • M3 • M5 • M7 • M9
    • D10 • D11 • D12 • D13 • D14 • D15
    = Π [ M(1,3,5,7,9) • D(10,11,12,13,14,15) ]
  - For realization of Z, don't care terms will be eventually assigned 0 or 1 in a way to simplify the circuit of Z as much as possible.

| 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 |

# Simplification: 2-level combinational logic

- Finding a minimal S-of-P or P-of-S realization
- Algebraic simplification
  - not an algorithmic/systematic procedure
  - how do you know when the minimum realization has been found?
- Computer-aided design tools
  - precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - to understand automatic tools and their strengths and weaknesses
  - ability to check results (on small examples)

# Essence of Boolean simplification

- Key tool to simplification: the **Uniting** theorem

  → $AB' + AB = A(B' + B) = A$

- Essence of simplification of two-level logic with uniting
  - Find two element subsets of the ON-set where only one variable changes its value
  - This single varying variable can be eliminated and a single product term used to represent both elements

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

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

B has the same value in both on-set rows – B remains

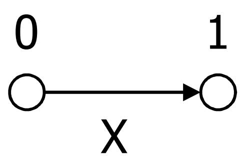
A has a different value in the two rows – A is eliminated

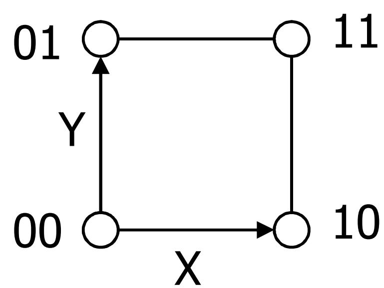
# Boolean cubes

*abbreviated as* ***n-cube***

- Visual technique for identifying when the uniting theorem can be applied
- truth table with n input variables = *n-dimensional cube*

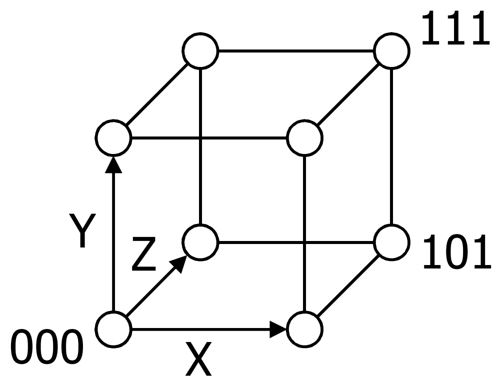
1-cube

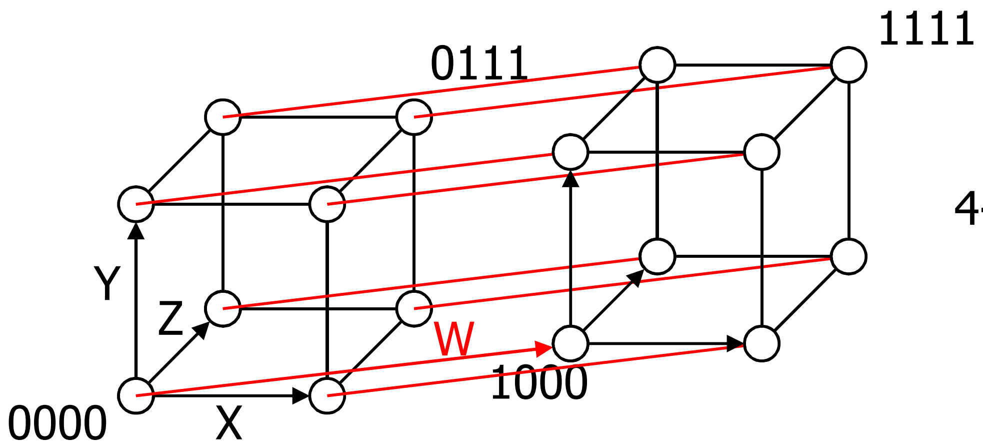



2-cube

3-cube



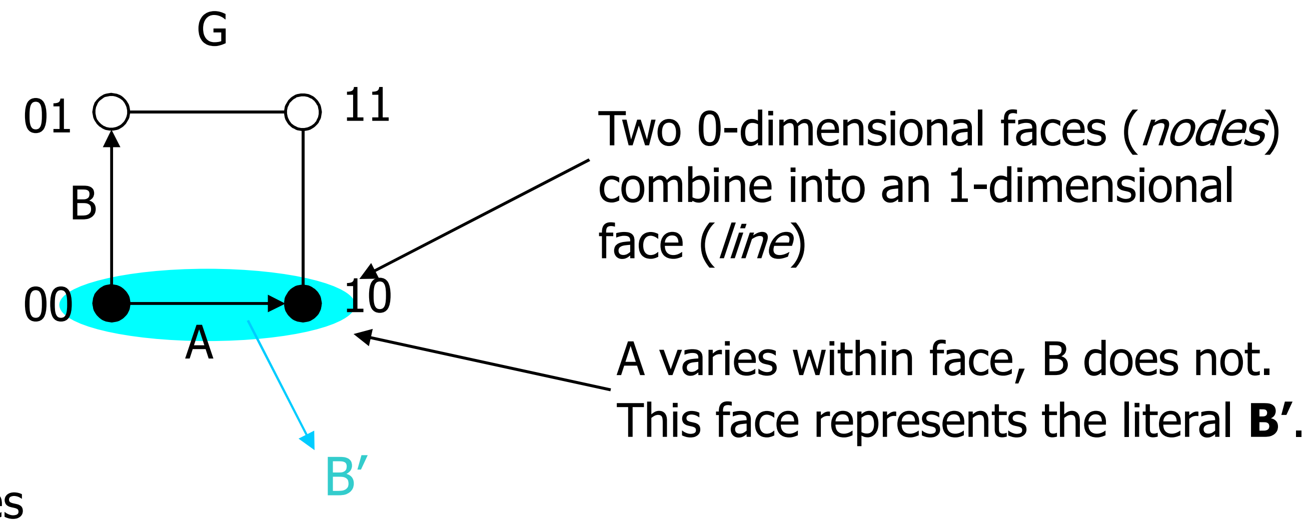

4-cube

# Mapping truth tables onto Boolean cubes

- The uniting theorem combines two **faces** of a cube into one larger face
- Adjacency plane
  - circled elements of the on-set that are directly adjacent
  - each adjacency plane corresponds to a product term
- Example:

| A | B | G |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

ON-set = solid nodes
OFF-set = empty nodes
DC-set = $\times$'d nodes

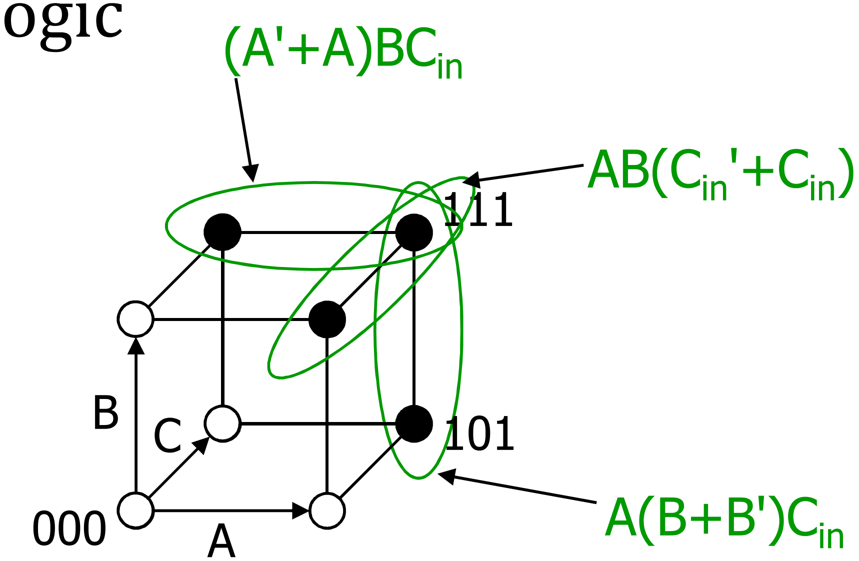


# Three variable example

*Three 1-dimensional adjacency planes*

- Binary full-adder carry-out logic

| A | B | $C_{in}$ | $C_{out}$ |
|---|---|---|---|
| 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 |




The on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times.

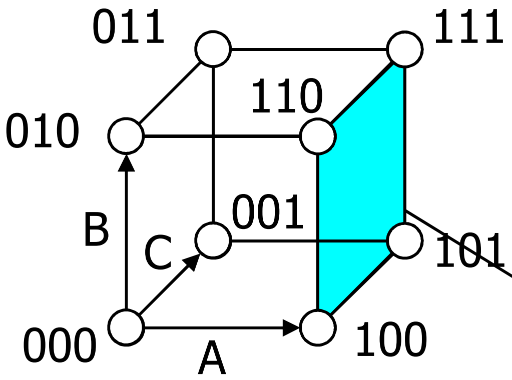
$C_{out} = BC_{in}+AB+AC_{in}$

*Four nodes (0-dimensional) are combined into three lines (1-dimensional)*

# Higher dimensional cubes

- Sub-cubes of higher dimension than 1
- Example: 2-dimensional adjacency plane in a 3-cube

F(A,B,C) = Σm(4,5,6,7)




On-set forms a square
i.e., a cube of dimension 2

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

A is asserted (true) and unchanged
B and C vary

This subcube represents the literal A

*Four nodes (dimension 0) are combined into four lines (dimension 1), which are then combined into one 2-dimensional sub-cube (square)*

AB′C′+ABC′+AB′C+ABC
= AB + AB′ + AC + AC′
= A

# m-cubes in n-dimensional Boolean space

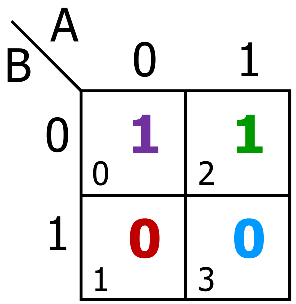
- In a 3-cube (three variables):
  - 0-dimensional plane, i.e., a single **node**, yields a term in 3 literals
    - example : 101 = AB'C
  - 1-dimensional plane, i.e., a **line** of two nodes, yields a term in 2 literals
    - example : 100 — 101 = AB'
  - 2-dimensional plane, i.e., a **squire** plane of four nodes, yields a term in 1 literal
    - example :

      100 — 101

      110 — 111 = A
  - 3-dimensional plane, i.e., a **cube** of eight nodes, yields 0 literal meaning a constant logic "1"
- In general, an ***m***-dimensional adjacency plane within an ***n***-dimensional cube ($m < n$) yields a term with ***n* – *m*** literals

# Karnaugh maps

- The cube notation provides visual clues as to where the uniting theorem is applied to elements of on-set.
- But, humans have the difficulty of visualizing adjacencies in more than 3 dimensional cubes.
- Karnaugh maps alternative reformulation of the truth table
  - for expressions at least up to six variables
  - wrap–around at edges
  - On-set elements with only one variable changing value are adjacent unlike the situation in a linear truth-table



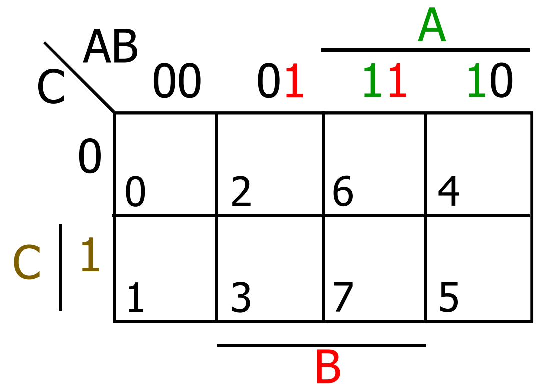

| A | B | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

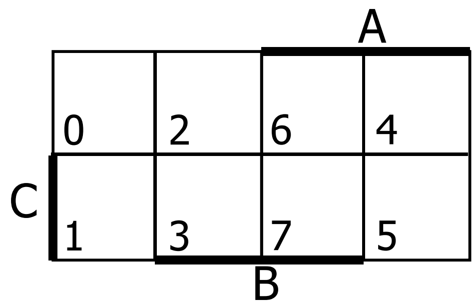
# Karnaugh maps (K-Maps)

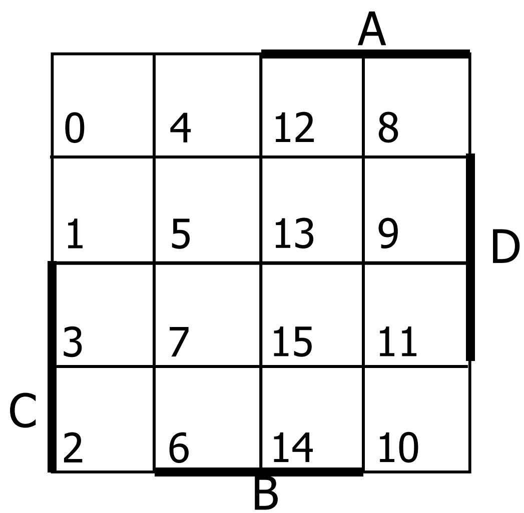
- Numbering scheme based on <u>Gray–code</u>
  - e.g., 00, 01, 11, 10 (*for every advancement, a single bit changes*)
  - only a single bit changes in code for adjacent map cells

000
001
011
010
110
111
101
100

*K-map for six (=3+3) variables is possible*
*How about 7 or 8 variables?*



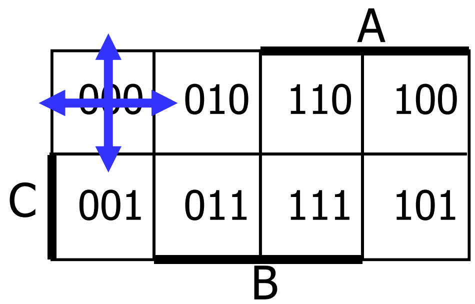



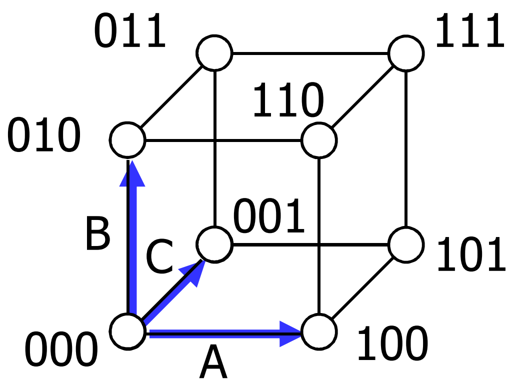



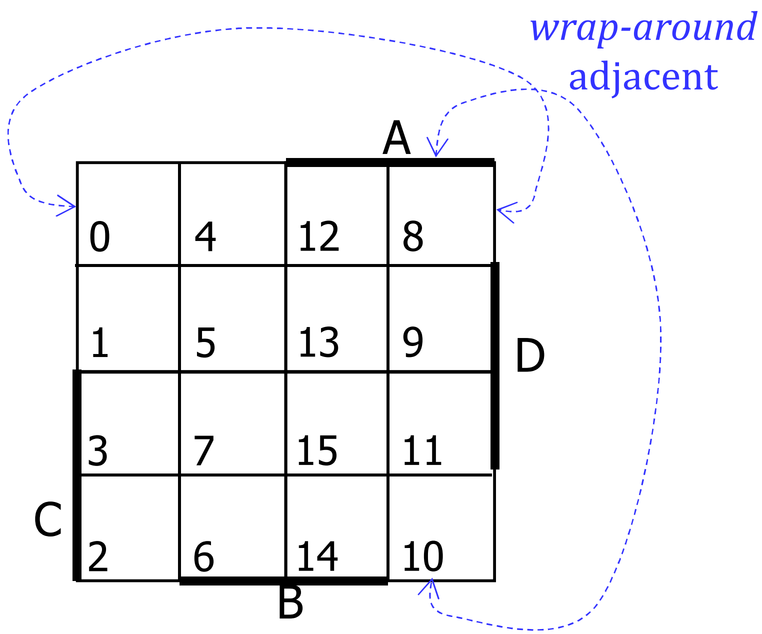

13 = 1101= ABC′D

# Adjacencies in K-Maps

- Wrap from first to last column
- Wrap top row to bottom row



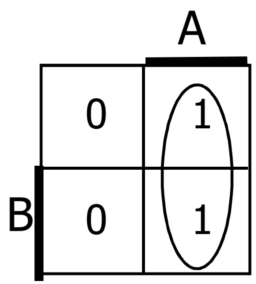



# Karnaugh map examples

- 2-variable maps
  - F = AB′ + AB = A

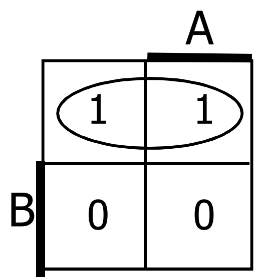
| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




A

  - G = A′B′ + AB′ = B′

| A | B | G |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

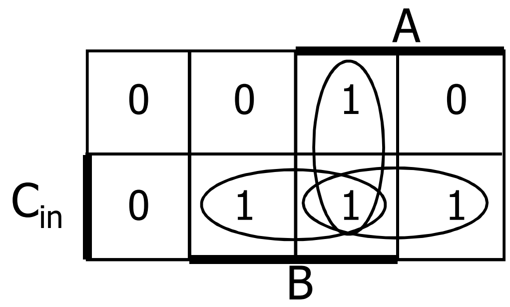



B′

# Karnaugh map examples

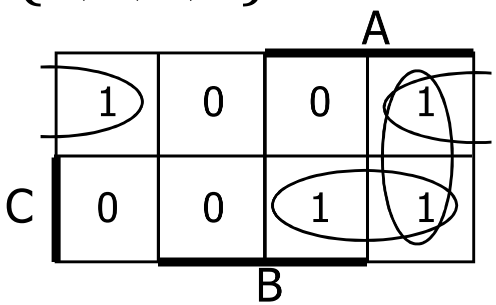
- 3-variable maps
  - full adder

| A | B | $C_{in}$ | $C_{out}$ |
|---|---|---|---|
| 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 |




$AB + Bc_{in} + AC_{in}$

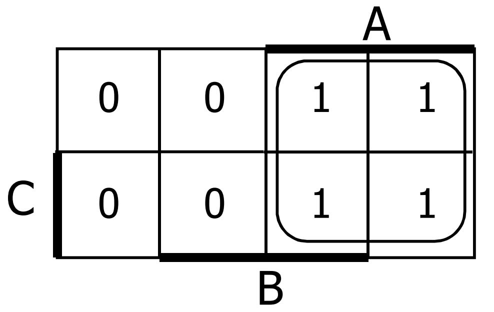
- $F(A,B,C) = \Sigma m(0,4,5,7)$



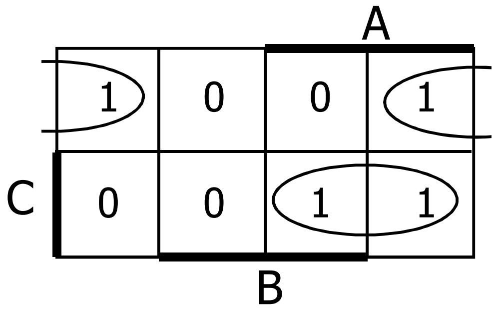

$AC + B'C' + \cancel{AB'}$

obtain the complement of the function by covering 0s with subcubes (see next page)

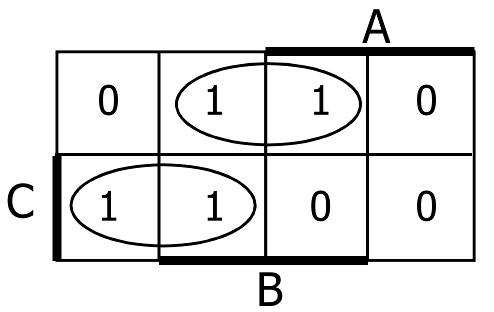
# More K-Map examples




G(A,B,C) = A




F(A,B,C) = $\Sigma$m(0,4,5,7) = AC + B′C′



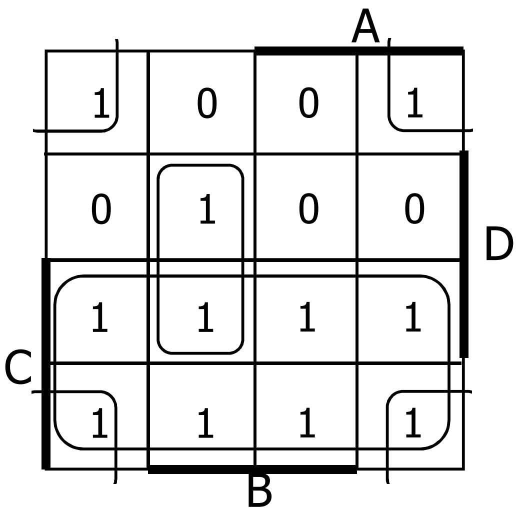

Complement of F(A,B,C) = m(0,4,5,7)
F' simply replace 1's with 0's and vice versa

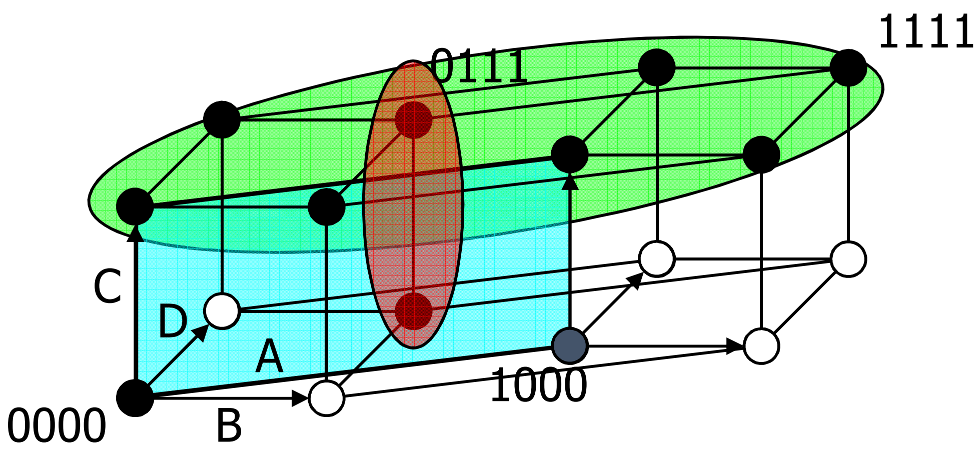
F'(A,B,C) = $\Sigma$ m(1,2,3,6) = BC′ + A′C

# Karnaugh map: 4-variable example

- F(A,B,C,D) = $\Sigma m(0,2,3,5,6,7,8,10,11,14,15)$

  F = C + A′BD + B′D′

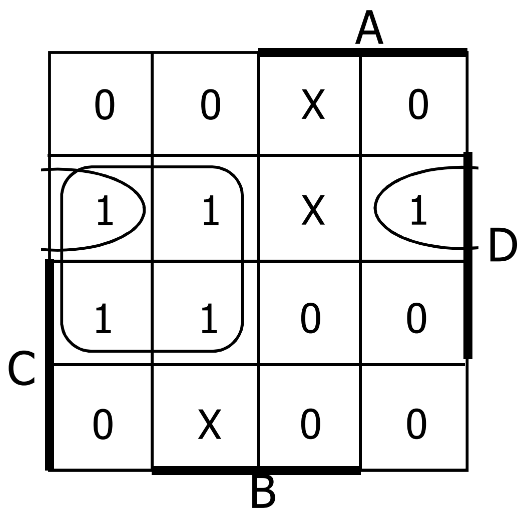



find the smallest number of the largest possible subcubes to cover the On-set (fewer terms with fewer inputs per term)

# Karnaugh maps: don't cares

- $f(A,B,C,D) = \Sigma\, m(1,3,5,7,9) + d(6,12,13)$
  - without don't cares
    - $f = A'D + B'C'D$



# Karnaugh maps: don't cares

- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)
  - f = A'D + B'C'D — without don't cares
  - f = A'D + C'D — with don't cares

A

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | X | 0 |
| | 1 | 1 | X | 1 |
| C | 1 | 1 | 0 | 0 |
| | 0 | X | 0 | 0 |

D

B

by using don't care as a "1" a 2-cube can be formed rather than a 1-cube to cover this node

<u>don't cares</u> can be treated as 1s or 0s depending on which is more advantageous

# Multilevel Logic

- Comparison with 2-level logic
  - gain: reduce the number of wires, gates and inputs to each gate
  - lose: add up more combined delay due to the increased levels of logic
- Example
  - 2-level logic

    $Z = ADF + AEF + BDF + BEF + CDF + CEF + G$

    → six 3-input AND gates, one 7-input OR gate
  - multilevel logic

    $Z = (AD + AE + BD + BE + CD + CE)F + G$

    $Z = [(A + B + C)D + (A + B + C)E]F + G$

    $Z = (A + B +C)(D + E)F + G$

    → one 3-input OR gate, two 2-input OR gates, one 3-input AND gate

# Chapter review

- Variety of primitive logic building blocks
  - NOT, AND, OR, NAND, NOR, XOR and XNOR gates
- Axioms and theorems of Boolean algebra
  - proofs by re-writing and perfect induction
- Two-level logic
  - canonical forms: sum-of-products and product-of-sums
  - incompletely specified functions
- Simplification
  - a start at understanding two-level simplification
  - Boolean cubes
  - K-Map