Digital Logic Design

4190.201.001

2010 Spring Semester

3. Combinational Logic

Naehyuck Chang
Dept. of EECS/CSE
Seoul National University
naehyuck@snu.ac.kr

ELPL Embedded Low-Power Laboratory

Introduction

- Combinational logic
  - Whose output is solely determined by their inputs
- Representation of a function
  - Truth table
  - Boolean equation
    - Sum of products, a two-level form
    - Unique way to represent a logic like a finger print
    - Alternative form is product of sums
  - Can be done in many ways
    - Highly desirable to find the simplest implementation
    - Gates and wires
- Boolean minimization
  - Karnaugh map, etc.
  - Fundamental tradeoff between time and space (speed and area: gates and wires)
  - Two-level logic and multi-leveled logic

Introduction

- Time response of in digital network
  - Non-zero propagation delay

Introduction

- Definition
  - Output behavior depends on the current input
  - Memoryless
  - Example: adder
    - The output changes shortly after the input changes, but the previous input has nothing to do with the current output
- Comparison with sequential logic
  - There is memory or state
  - Whose output depends both the input and the state
  - Example: traffic light
- Simple combinational circuits representing with truth tables

X	Y	Equal
0	0	1
0	1	0
1	0	0
1	1	1

Comparator

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

Tally circuit

Introduction

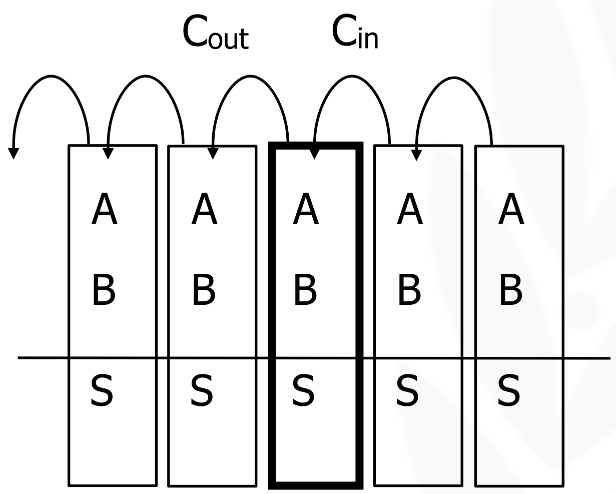
- More examples
  - Half adder: output the carry but cannot be chained
  - Full adder: can be chained
- Truth table
  - Suitable with a modest number of inputs
  - 2n number of rows where n is the number of inputs

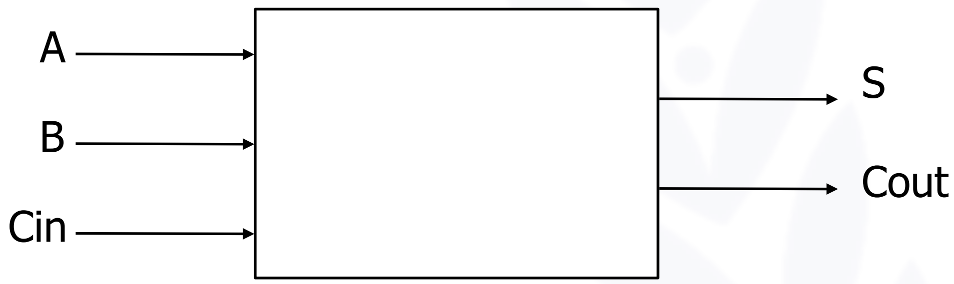
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half adder

Introduction

- Full adder





A	A	Cin	Carry	Sum
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

An Algebraic Structure

- An algebraic structure consists of
  - A set of elements B
  - Binary operations { + , • }
  - And a unary operation { ’ }
  - Such that the following axioms hold:

1. The set B contains at least two elements a, b such that a is not equal to b	
2. Closure:	$a + b \in B$, $a \bullet b \in B$
3. Commutativity:	$a + b = b + a$, $a \bullet b = b \bullet a$
4. Associativity:	$a + (b + c) = (a + b) + c$, $a \bullet (b \bullet c) = (a \bullet b) \bullet c$
5. Identity:	$a + 0 = a$, $a \bullet 1 = a$
6. Distributivity:	$a + (b \bullet c) = (a + b) \bullet (a + c)$, $a \bullet (b + c) = (a \bullet b) + (a \bullet c)$
7. Complementarity:	$a + a' = 1$, $a \bullet a' = 0$

  - Order of operations
    - Complement, AND and then OR
  - AND and OR are not the same to the arithmetic operations MULTIPLY and PLUS

Truth Tables

- Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: ′, +, and •
- Equivalence of Boolean expressions and truth tables
  - Can be readily derived from each other

X	Y	X • Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X′	X′ • Y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

X	Y	X′	Y′	X • Y	X′ • Y′	(X • Y) + (X′ • Y′)
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

$$(X \bullet Y) + (X' \bullet Y') \equiv X = Y$$

Boolean expression that is true when the variables X and Y have the same value and false, otherwise

X, Y are Boolean algebra variables

Embedded Low-Power Laboratory

Tuesday, March 23, 2010

Truth Tables

- Reduced carry out full adder expression
  - $C_{out} = (AC_{in}) + (BC_{in}) + (AB)$

A	A	C_{in}	AC_{in}	BC_{in}	AB	C_{out}
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	1
1	1	1	1	1	1	1

Truth Tables

- Deriving expressions from truth tables
  - S =
  - C_{out} =

A	B	C_{in}	C_{out}	Sum
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

Theorems of Boolean Algebra

- Duality
  - A dual of a Boolean expression is derived by replacing
    • by +, + by •, 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)
- Duality:
  - X + Y + ... ⇔ X • Y • ...
- Generalized duality:
  - f (X1,X2,...,Xn,0,1,+,•) ⇔ f(X1,X2,...,Xn,1,0,•,+)
- Different from deMorgan's Law
  - This is a statement about theorems
  - This is not a way to manipulate (re-write) expressions

Theorems of Boolean Algebra

- Identity
  - 1. $X + 0 = X$ 1D. $X \bullet 1 = X$
- Null
  - 2. $X + 1 = 1$ 2D. $X \bullet 0 = 0$
- Idempotency:
  - 3. $X + X = X$ 3D. $X \bullet X = X$
- Involution:
  - 4. $(X')' = X$
- Complementarity:
  - 5. $X + X' = 1$ 5D. $X \bullet X' = 0$
- Commutativity:
  - 6. $X + Y = Y + X$ 6D. $X \bullet Y = Y \bullet X$
- Associativity:
  - 7. $(X + Y) + Z = X + (Y + Z)$ 7D. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$

Theorems of Boolean Algebra

- Distributivity:
  - 8. $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$ 8D. $X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$
- Uniting:
  - 9. $X \bullet Y + X \bullet Y' = X$ 9D. $(X + Y) \bullet (X + Y') = X$
- Absorption:
  - 10. $X + X \bullet Y = X$ 10D. $X \bullet (X + Y) = X$
  - 11. $(X + Y') \bullet Y = X \bullet Y$ 11D. $(X \bullet Y') + Y = X + Y$
- Factoring:
  - 12. $(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$ 12D. $X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$
- Concensus:
  - 13. $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$ 13D. $(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$

Proving Theorems (Rewriting)

- Using the axioms of Boolean algebra:
  - e.g., prove the theorem: $X \bullet Y + X \bullet Y' = X$

distributivity (8)	$X \bullet Y + X \bullet Y'$	$= X \bullet (Y + Y')$
complementarity (5)	$X \bullet (Y + Y') = X \bullet (1)$	
identity (1D)	$X \bullet (1)$	$= X$ ü

  - e.g., prove the theorem: $X + X \bullet Y = X$

identity (1D)	$X + X \bullet Y$	$= X \bullet 1 + X \bullet Y$
distributivity (8)	$X \bullet 1 + X \bullet Y$	$= X \bullet (1 + Y)$
identity (2)	$X \bullet (1 + Y)$	$= X \bullet (1)$
identity (1D)	$X \bullet (1)$	$= X$ ü

Activity

- Prove the following using the laws of Boolean algebra:
  - $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$

	$(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z)$
identity	$(X \bullet Y) + (1) \bullet (Y \bullet Z) + (X' \bullet Z)$
complementarity	$(X \bullet Y) + (X' + X) \bullet (Y \bullet Z) + (X' \bullet Z)$
distributivity	$(X \bullet Y) + (X' \bullet Y \bullet Z) + (X \bullet Y \bullet Z) + (X' \bullet Z)$
commutativity	$(X \bullet Y) + (X \bullet Y \bullet Z) + (X' \bullet Y \bullet Z) + (X' \bullet Z)$
commutativity	$(X \bullet Y) + (X \bullet Y) \bullet Z + (X' \bullet Z) + (X' \bullet Z) \bullet Y$
absorption	$(X \bullet Y) + (X' \bullet Z)$

Activity

- Full adder example
  - Cout = A′BCin + AB′Cin + ABCin′ + ABCin
  - Idempotent (introduces a redundant term)
    - Cout = A′BCin + AB′Cin + ABCin′ + ABCin + ABCin
  - Commutative (rearranges terms)
    - Cout = A′BCin + ABCin + AB′Cin + ABCin′ + ABCin
  - Distributed (factors out common literals)
    - Cout = (A′ + A)BCin + AB′Cin + ABCin′ + ABCin
  - Complementarity (replaces A′ + A to 1)
    - Cout = (1)BCin + AB′Cin + ABCin′ + ABCin
  - Identity (replaces 1X to X)
    - Cout = BCin + AB′Cin + ABCin′ + ABCin
  - Finally
    - Cout = BCin + ACin + AB

DeMorgan's Law

- DeMOrgan's law
  - Establishes relationship between • and +
- Theorem:
  - 14. $(X + Y + \ldots)' = X' \bullet Y' \bullet \ldots$ 14D. $(X \bullet Y \bullet \ldots)' = X' + Y' + \ldots$
- Generalized DeMorgan's:
  - 15. $f'(X1,X2,\ldots,Xn,0,1,+,\bullet) = f(X1',X2',\ldots,Xn',1,0,\bullet,+)$
- Purpose
  - Negative logic

Proving Theorems (Perfect Induction)

- Using perfect induction (complete truth table):
  - e.g., de Morgan's:

$(X + Y)' = X' \bullet Y'$
NOR is equivalent to AND
with inputs complemented

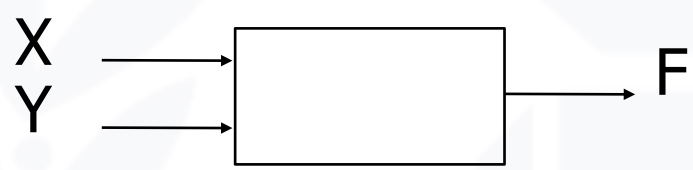
X	Y	X′	Y′	(X + Y)′	X′ • Y′
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

$(X \bullet Y)' = X' + Y'$
NAND is equivalent to OR
with inputs complemented

X	Y	X′	Y′	(X • Y)′	X′ + 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

Possible Logic Functions of Two Variables

- There are 16 possible functions of 2 input variables:
  - In general, there are 2^{2^n} functions of n inputs



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		X		Y	X xor Y	X or Y	X nor Y not (X or Y)	X = Y	not Y		not X		X nand Y not (X and Y)	1

Cost of Different Logic Functions

- Different functions are easier or harder to implement
  - 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, because NOT, NOR, and NAND are the cheapest they are the functions we implement the most in practice
- But if we consider the target technology (logic structure), the story is different
  - To be learned from electronics circuit, semiconductor and advanced digital system design

Minimal Set of Functions

- Can we implement all logic functions from NOT, NOR, and NAND?
  - For example, implementing X and Y
    is the same as implementing not (X nand Y)
- In fact, we can do it with only NOR or only NAND
  - NOT is just a NAND or a 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

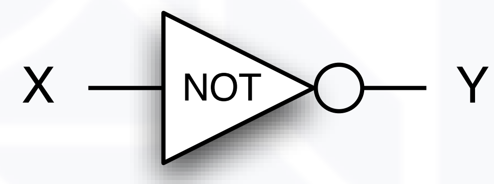
  - And NAND and NOR are "duals",
    that is, its easy to implement one using the other

X nand Y ≡ not ((not X) nor (not Y))
X nor Y ≡ not ((not X) nand (not Y))

- But lets not move too fast . . .
  - Let's look at the mathematical foundation of logic

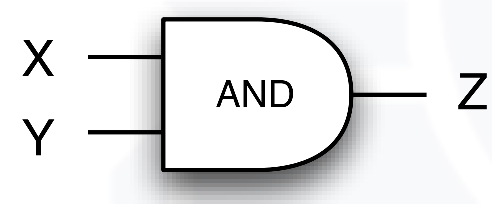
From Boolean Expressions to Logic Gates

- NOT — X' — $\overline{X}$ — $\sim X$



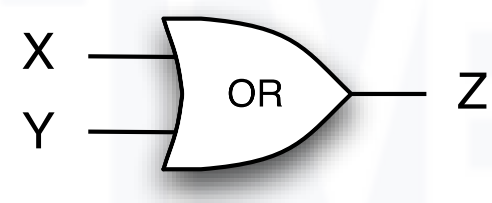
X	Y
0	1
1	0

- AND — $X \bullet Y$ — XY — $X \wedge Y$



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

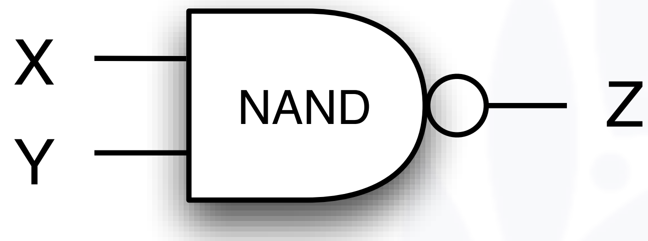
- OR — $X + Y$ — $X \vee Y$



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

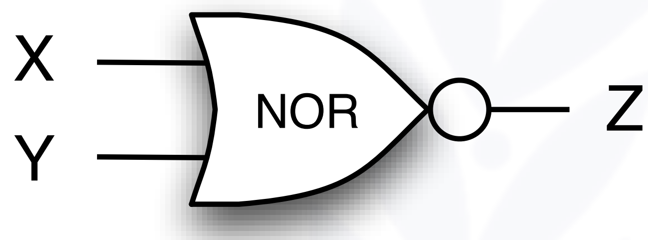
From Boolean Expressions to Logic Gates

- NAND



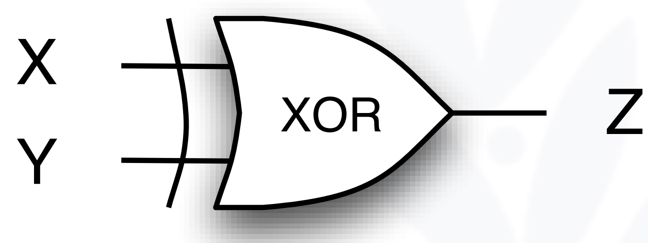
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR



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

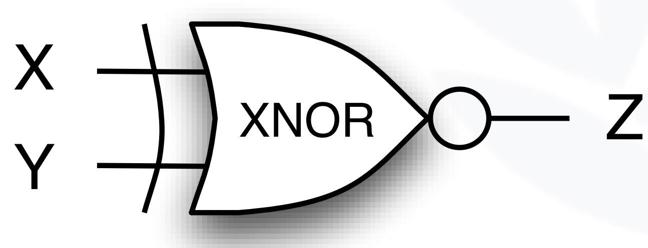
- XOR
  $X \oplus Y$



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

X xor Y = X Y’ + X’ Y
X or Y but not both
("inequality", "difference")

- XNOR
  $X \equiv Y$



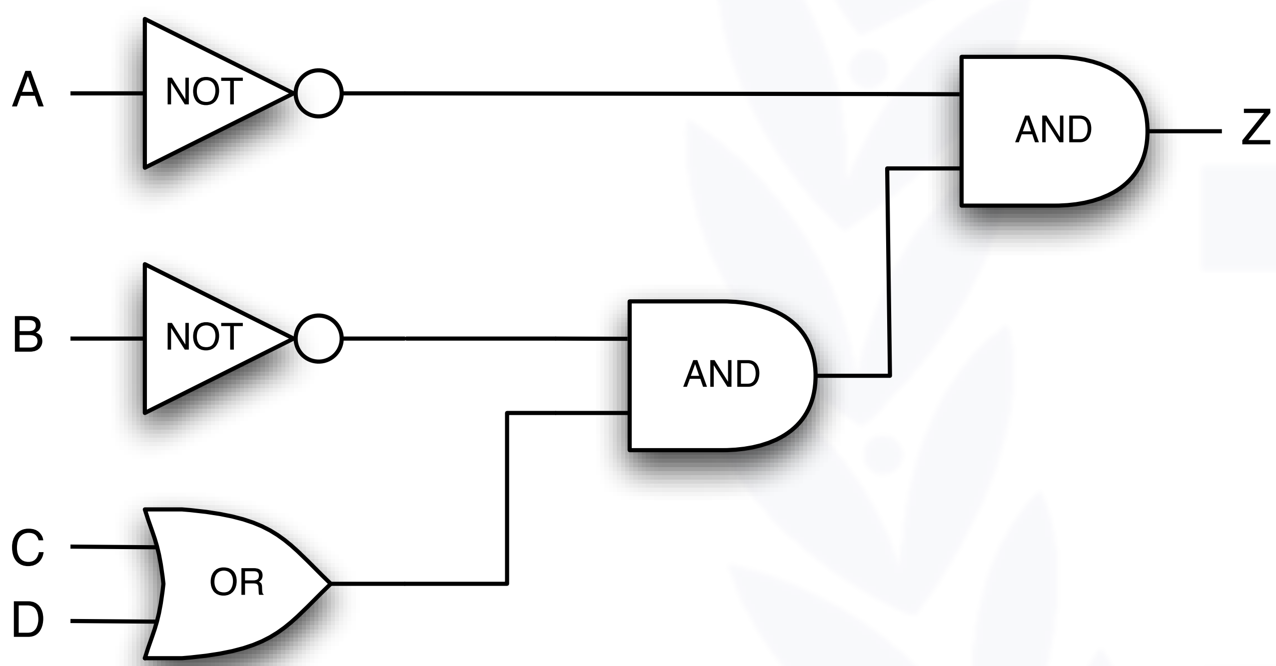
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

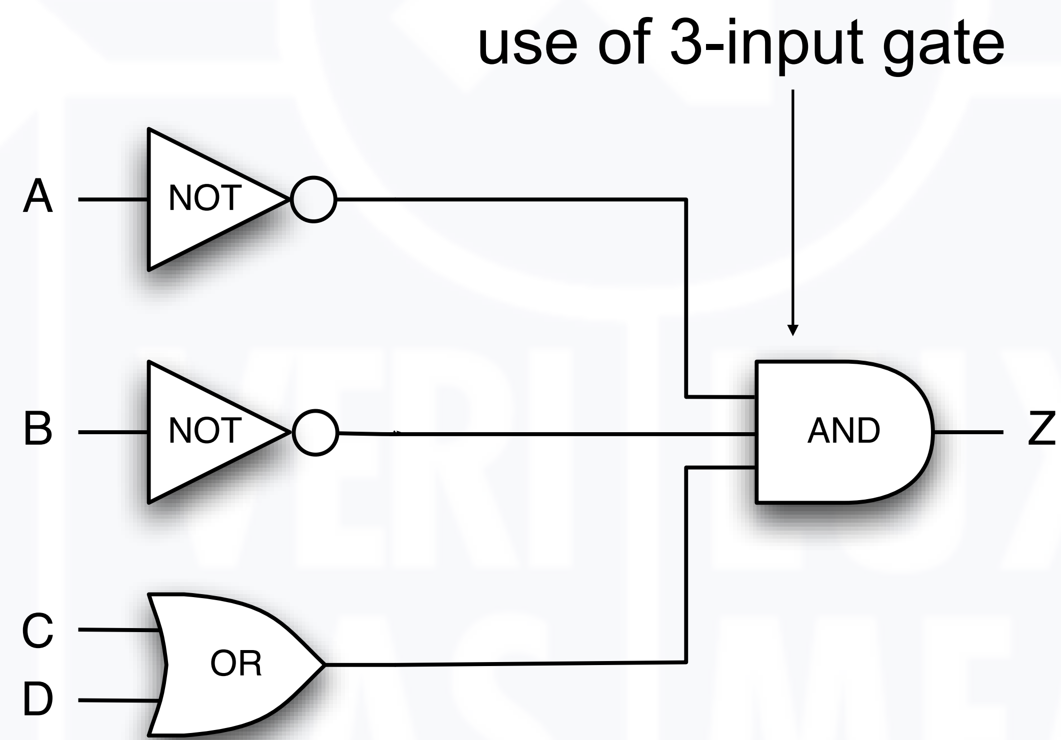
X xnor Y = X Y + X’ Y’
X and Y are the same
("equality", "coincidence")

Embedded Low-Power Laboratory

From Boolean Expressions to Logic Gates

- More than one way to map expressions to gates
  - e.g., Z = A′ • B′ • (C + D) = (A′ • (B′ • (C + D)))



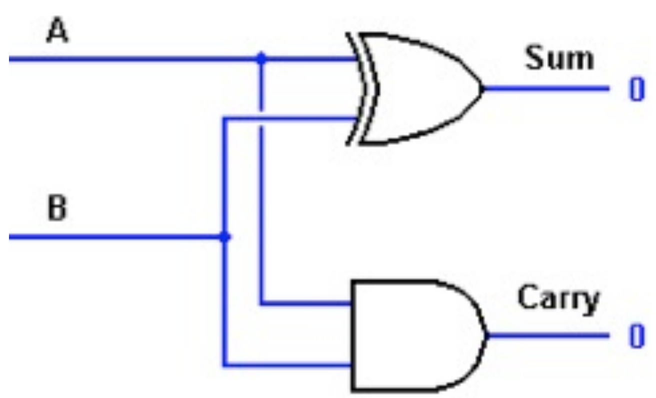


From Boolean Expressions to Logic Gates

- Implication
  - X implies Y: $X \Rightarrow Y$
  - Is false only when X is true and Y is false

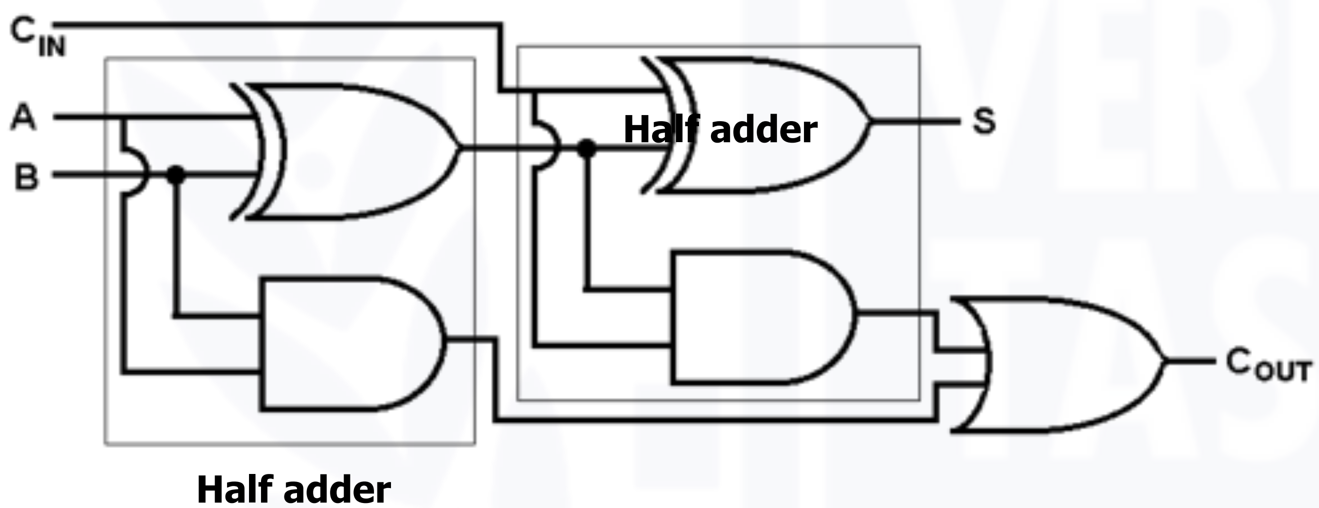
Logic Blocks and Hierarchy

- Half adder



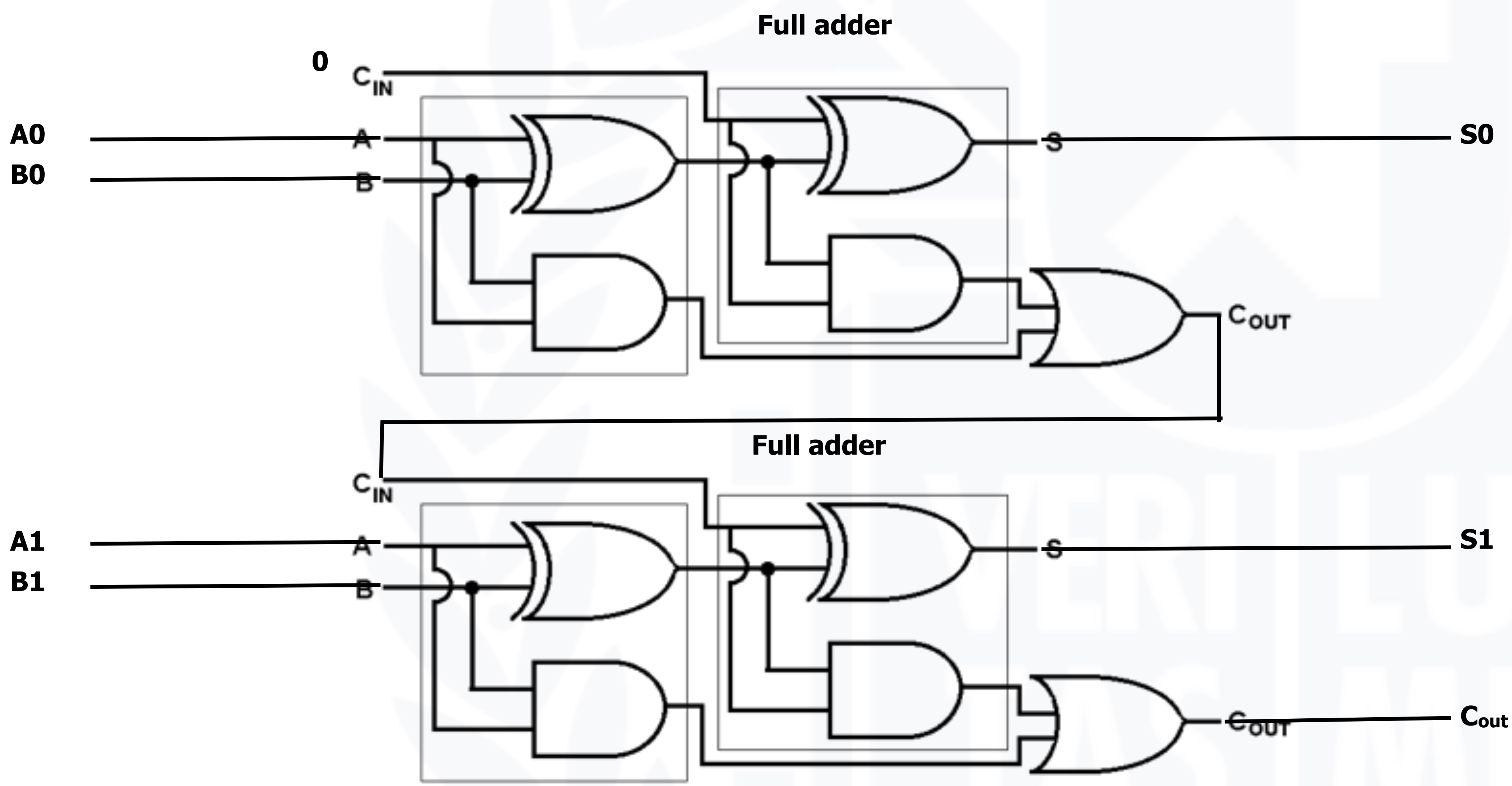
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Full adder



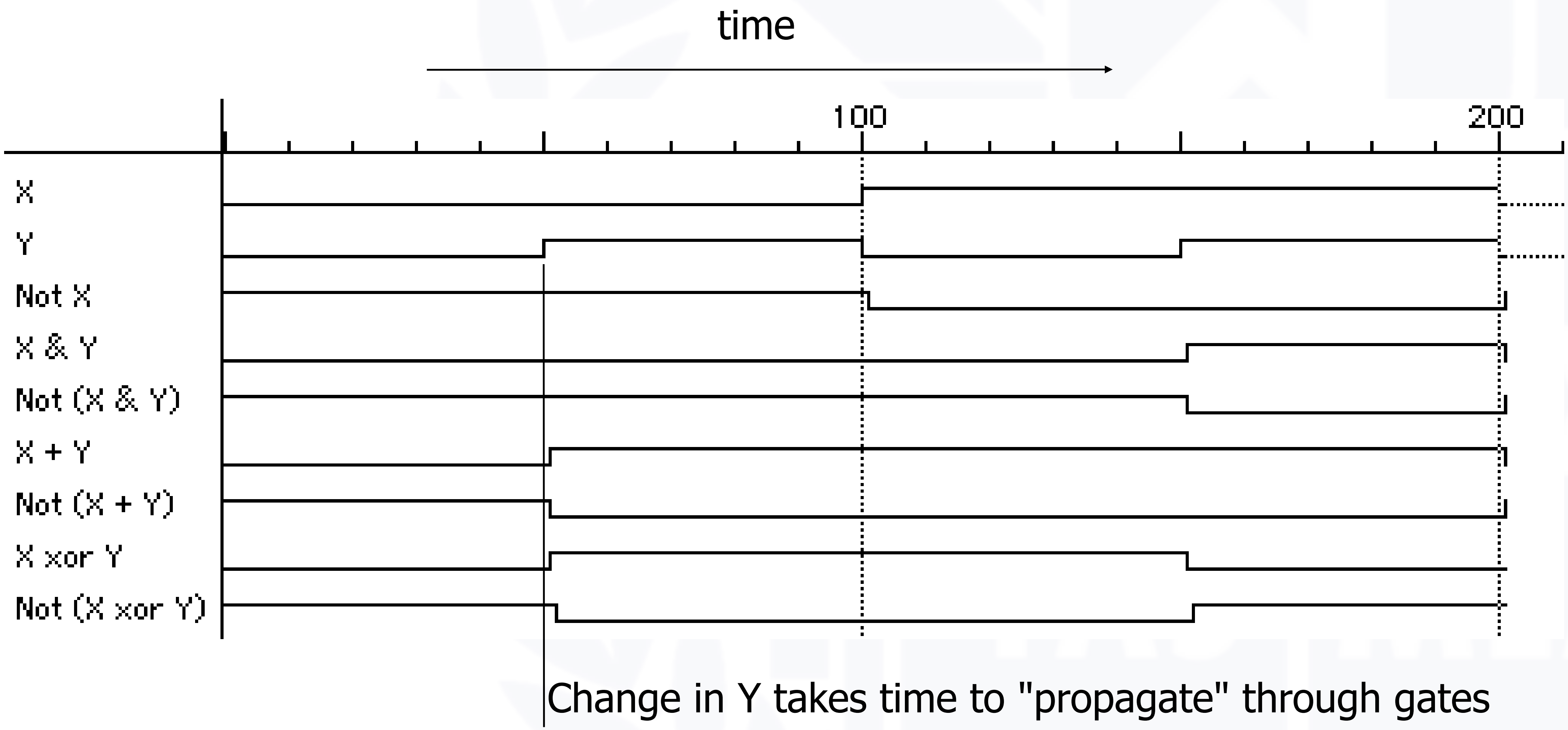
Logic Blocks and Hierarchy

- 2 bit full adder



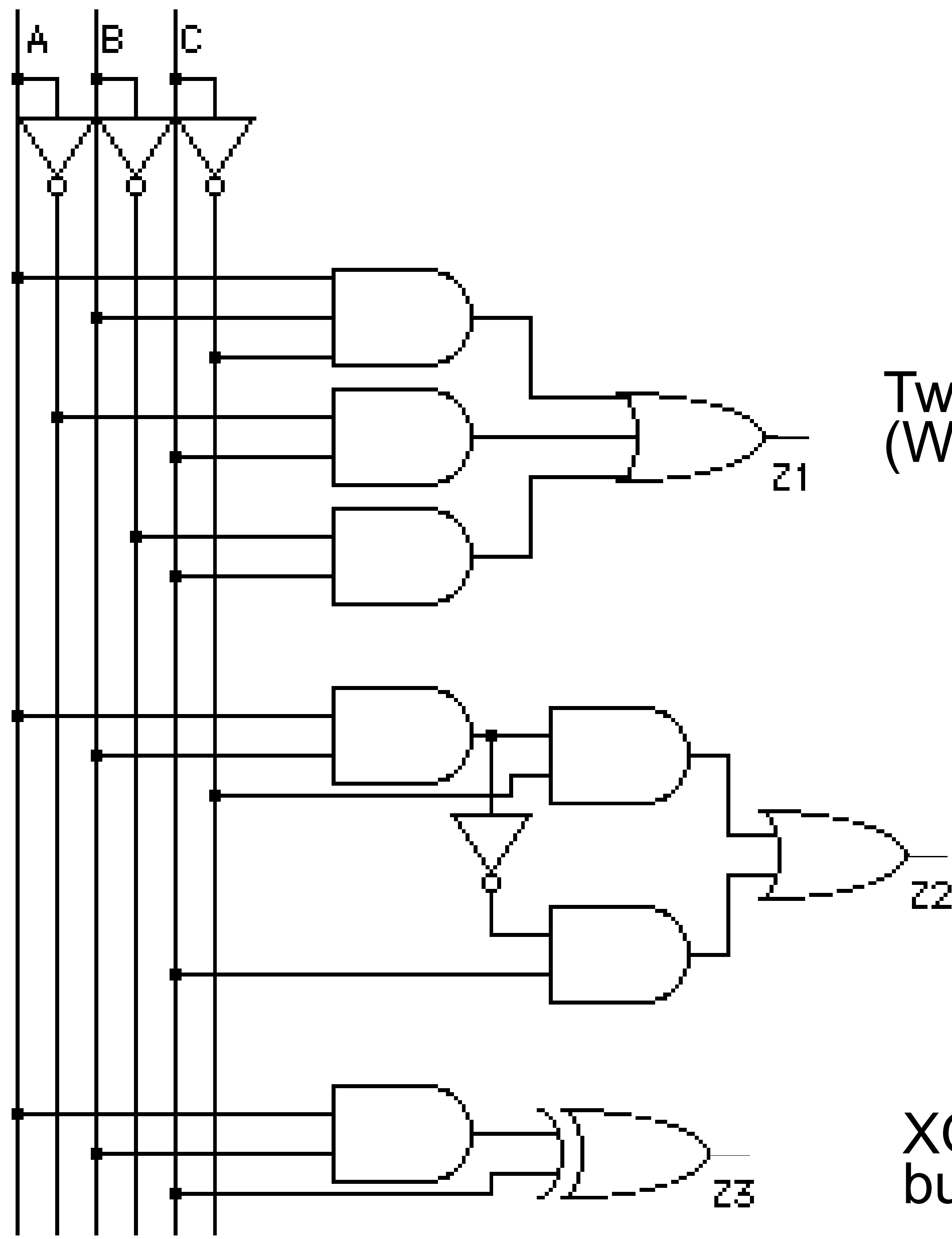
Waveform View of Logic Functions

- Just a sideways truth table
  - But note how edges don't line up exactly
  - It takes time for a gate to switch its output!



Time and Space Tradeoff

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
(We don't count NOT gates)

Multi-level realization
(Gates with fewer inputs)

XOR gate (easier to draw
but costlier to build)

Which Realization is Best?

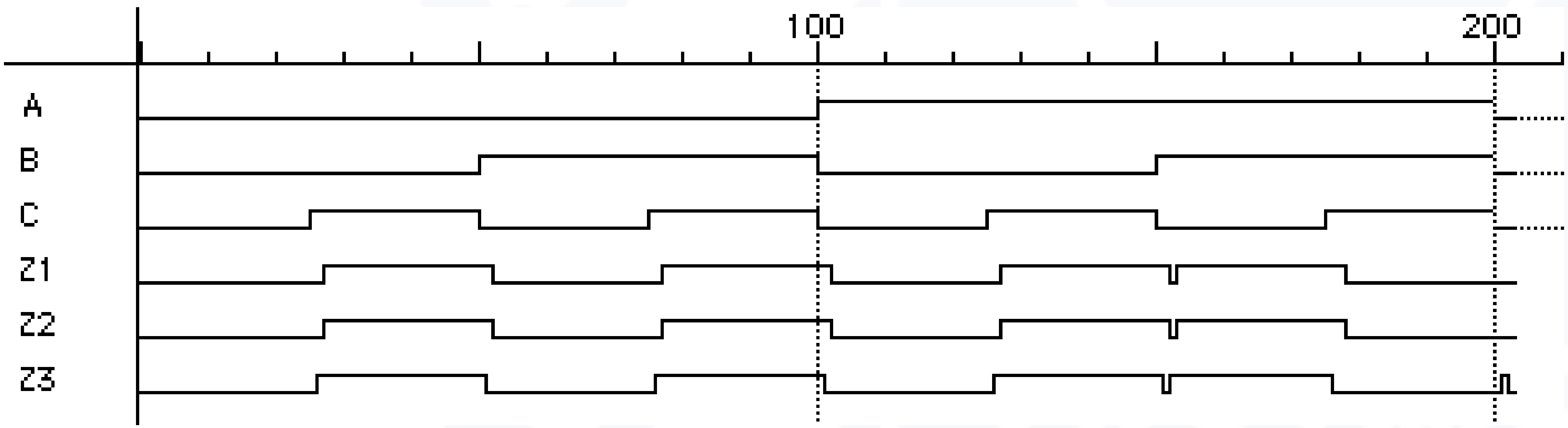
- Reduce number of inputs
  - Literal: input variable (complemented or not)
    - Can approximate cost of logic gate as 2 transistors per literal
    - Why not count inverters?
  - Fewer literals means less transistors
    - Smaller circuits
  - Fewer inputs implies faster gates
    - Gates are smaller and thus also faster
  - Fan-ins (# of gate inputs) are limited in some technologies
- Reduce number of gates
  - Fewer gates (and the packages they come in) means smaller circuits
    - Directly influences manufacturing costs

Which Is the Best Realization?

- Reduce number of levels of gates
  - Fewer level of gates implies reduced signal propagation delays
  - Minimum delay configuration typically requires more gates
    - Wider, less deep circuits
- How do we explore tradeoffs between increased circuit delay and size?
  - Automated tools to generate different solutions
  - Logic minimization: reduce number of gates and complexity
  - Logic optimization: reduction while trading off against delay

Are All Realizations Equivalent?

- Under the same input stimuli, the three alternative implementations have almost the same waveform behavior
  - Delays are different
  - Glitches (hazards) may arise – these could be bad, it depends
  - Variations due to differences in number of gate levels and structure
- The three implementations are functionally equivalent



Implementing Boolean Functions

- Technology independent
  - Canonical forms
  - Two-level forms
  - Multi-level forms
- Technology choices
  - Packages of a few gates
  - Regular logic
  - Two-level programmable logic
  - Multi-level programmable logic

Canonical Forms

- Truth table is the unique signature of a Boolean function
- The same truth table can have many gate realizations
- Canonical forms
  - Standard forms for a Boolean expression
  - Provides a unique algebraic signature

Sum-of-Products Canonical Forms

- Also known as disjunctive normal form
- Also known as minterm expansion
  - Minterm contains one version of every literal
  - Each minterm covers only one row
  - Minterms are ORed together to form the complete function

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 Form

- Product term and minterm
  - AB is a product term but not a minterm
  - ABC is a product term and a 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:

F(A, B, C) = Σm(1,3,5,6,7)
= m1 + m3 + m5 + m6 + m7
= A′B′C + A′BC + AB′C + ABC′ + ABC

Canonical form ≠ minimal form

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

Product-of-Sums Canonical Form

- Also known as conjunctive normal form
- Also known as maxterm expansion
  - Maxterm contains one version of every literal
  - Each maxterm covers all but one row
  - Maxterms are ANDed together and form the complete function

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

A+B+C

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

A+B′+C

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

A′+B+C

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+B′+C)(A′+B+C)

Product-of-Sums Canonical Form (cont'd)

- Sum term and maxterm
  - A+B is a sum term but not a maxterm
  - A+B+C is a sum term and a 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:

$$F(A, B, C) = \Pi M(0,2,4) = M0 \bullet M2 \bullet M4 = (A + B + C)\,(A + B' + C)\,(A' + B + C)$$

Canonical form ≠ minimal form

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

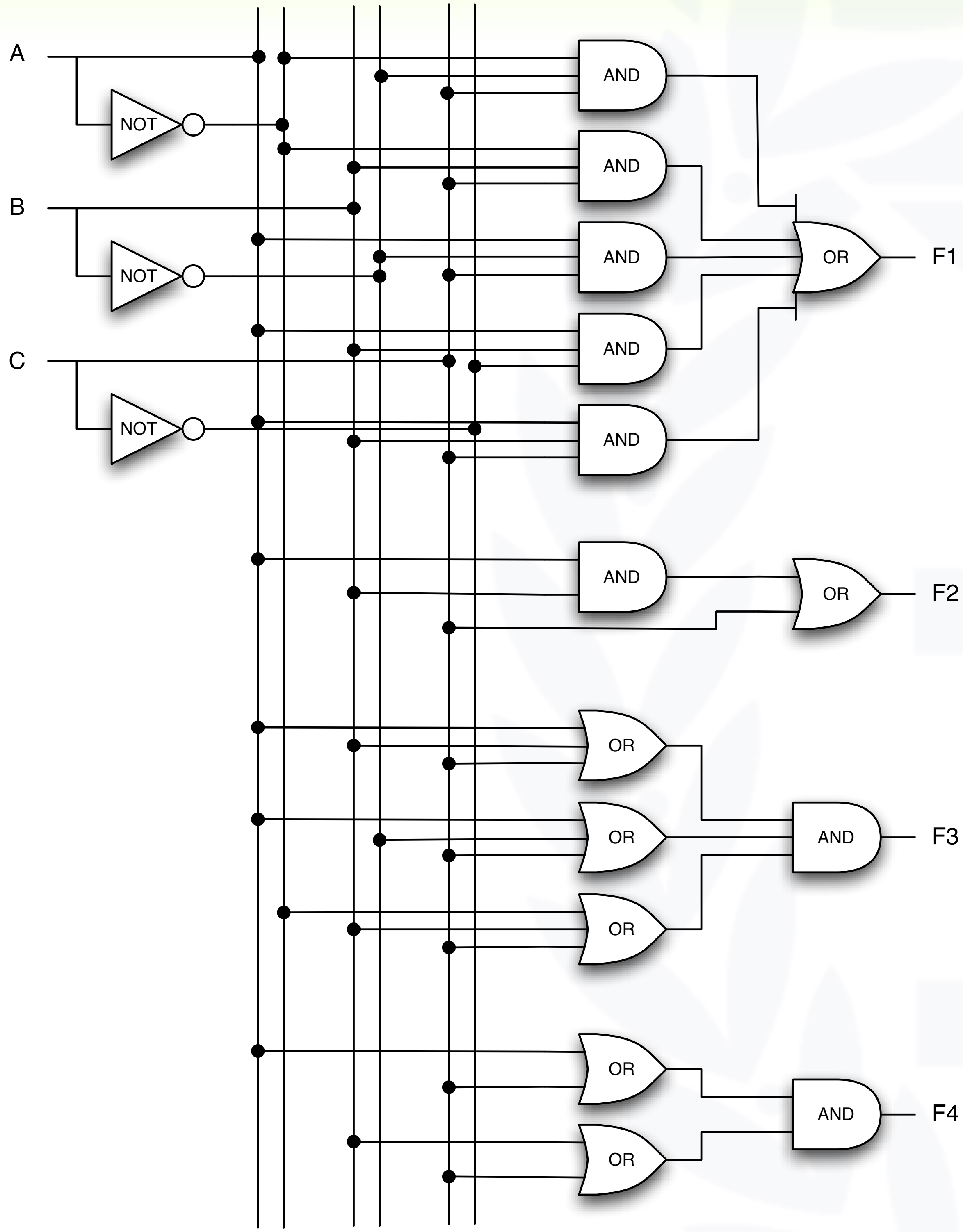
Embedded Low-Power Laboratory

S-o-P, P-o-S, and de Morgan's Theorem

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

Four Alternative Two-Level Implementations of F = AB + C



Canonical sum-of-products

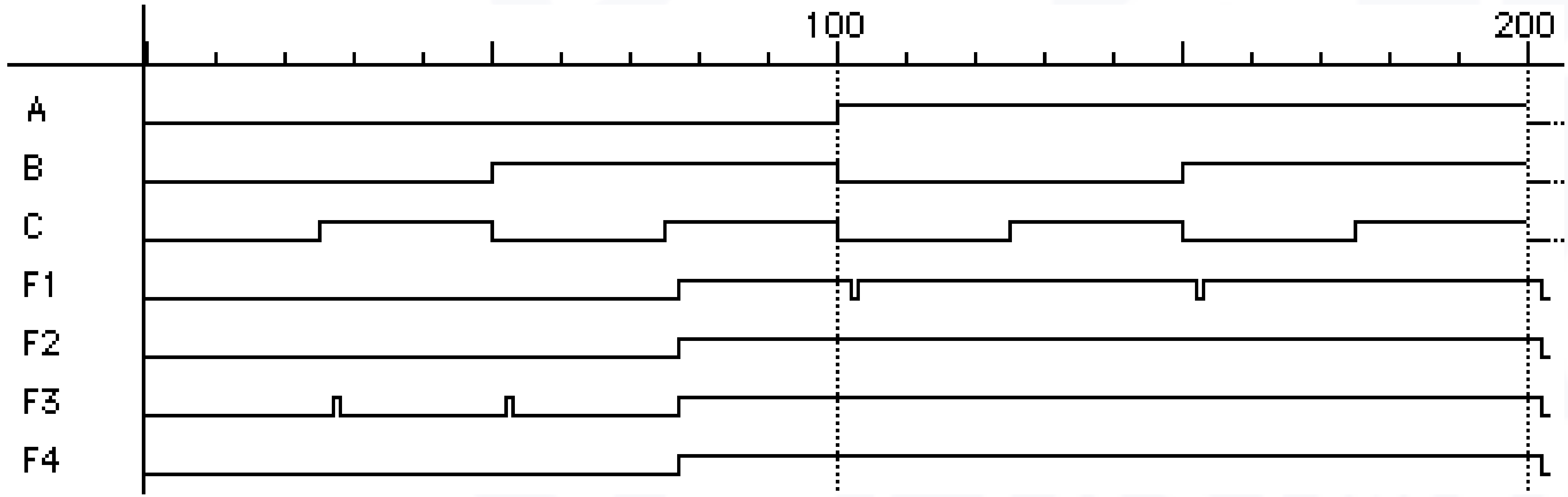
Minimized sum-of-products

Canonical product-of-sums

Minimized product-of-sums

Waveforms for the Four Alternatives

- Waveforms are essentially identical
  - Except for timing hazards (glitches)
  - Delays almost identical (modeled as a delay per level, not type of gate or number of inputs to gate)



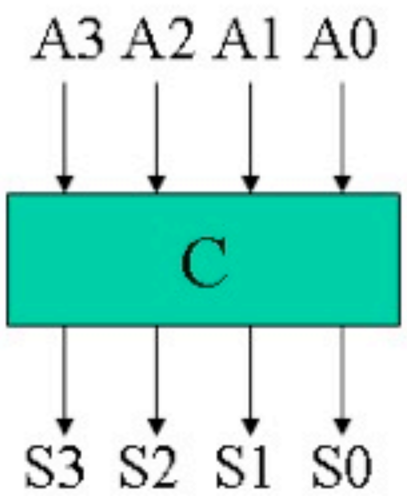
Embedded Low-Power Laboratory

Mapping Between Canonical Forms

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

- Binary and BCD (decimal) representation



A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	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	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	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

Incompletely Specified Functions

- Binary coded decimal 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:

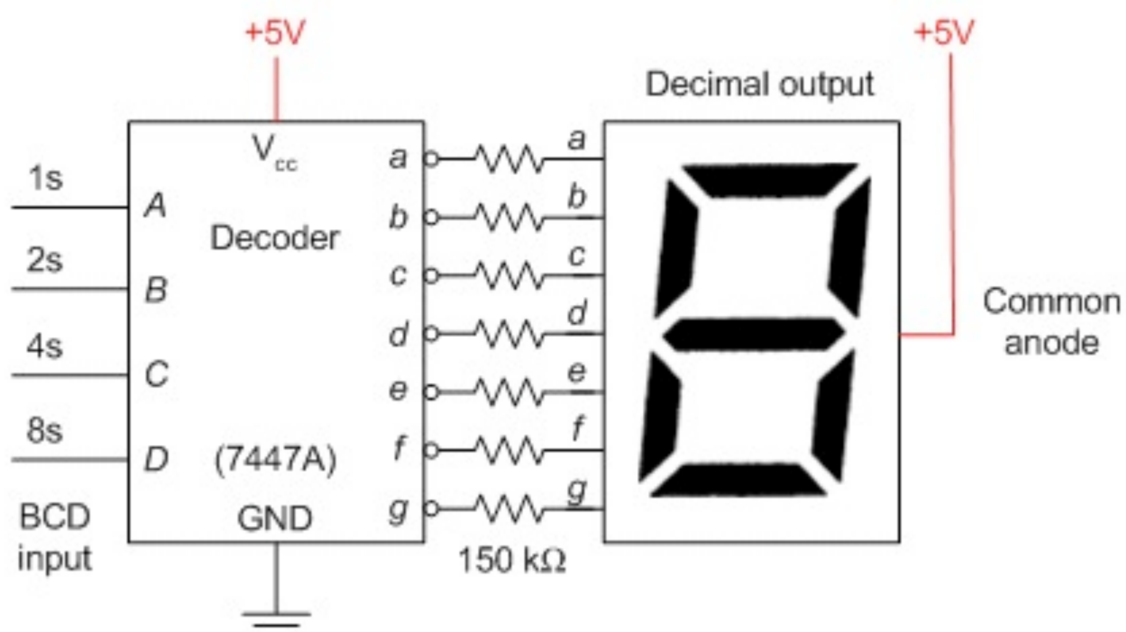
  - $Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15$
  - $Z = \Sigma$ [m(0,2,4,6,8) + d(10,11,12,13,14,15)]

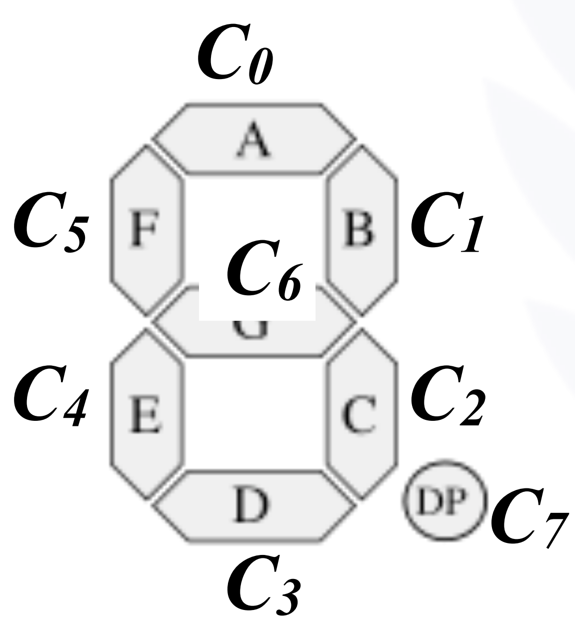
  - $Z = M1 \bullet M3 \bullet M5 \bullet M7 \bullet M9 \bullet D10 \bullet D11 \bullet D12 \bullet D13 \bullet D14 \bullet D15$
  - $Z = \Pi$ [M(1,3,5,7,9) • D(10,11,12,13,14,15)]

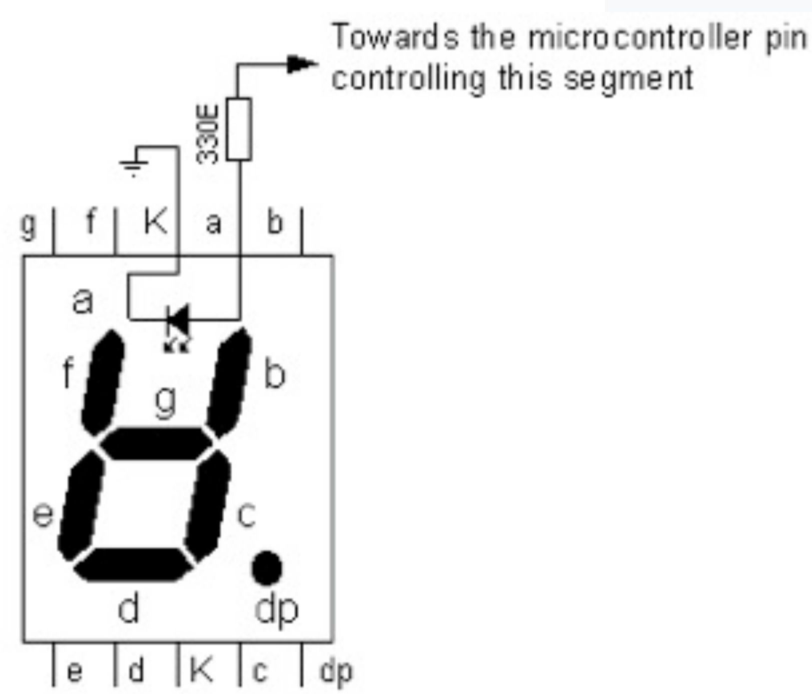
Embedded Low-Power Laboratory

Examples

- BCD (decimal) to seven segment decoder







Seven segment LED

Examples

- BCD to seven segment decoder
  - Truth table

Embedded Low-Power Laboratory

Simplification of Two-Level Combinational Logic

- Motivation: is the following optimization is easy and systematic?
  - Cout = A'BCin + AB'Cin + ABCin' + ABCin
  - Idempotent (introduces a redundant term)
    - Cout = A'BCin + AB'Cin + ABCin' + ABCin + ABCin
  - Commutative (rearranges terms)
    - Cout = A'BCin + ABCin + AB'Cin + ABCin' + ABCin
  - Distributed (factors out common literals)
    - Cout = (A' + A)BCin + AB'Cin + ABCin' + ABCin
  - Complementarity (replaces A' + A to 1)
    - Cout = (1)BCin + AB'Cin + ABCin' + ABCin
  - Identity (replaces 1X to X)
    - Cout = BCin + AB'Cin + ABCin' + ABCin
  - Finally
    - Cout = BCin + ACin + AB

Simplification of Two-Level Combinational Logic

- Finding a minimal sum of products or product of sums realization
  - Exploit don't care information in the process
- 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)

The Uniting Theorem

- Key tool to simplification: A (B′ + B) = A
- Essence of simplification of two-level logic
  - 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

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

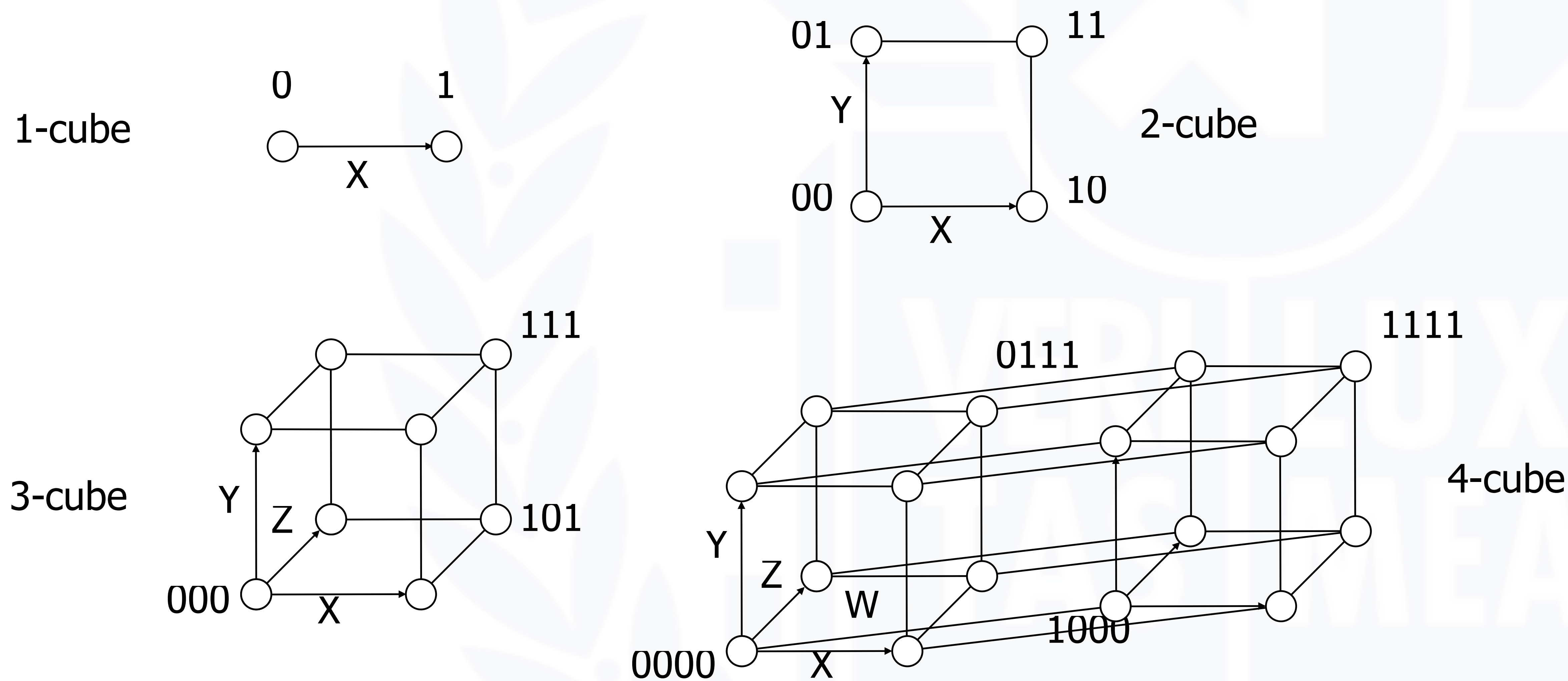
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

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

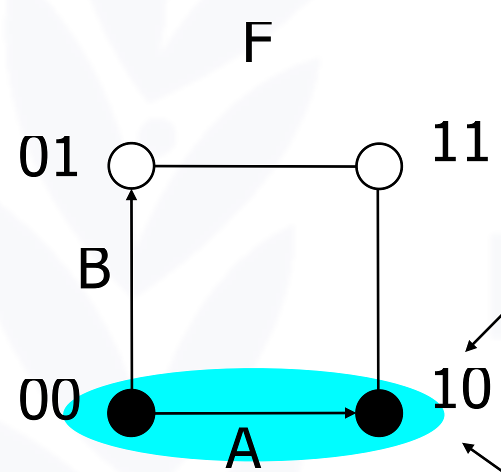
- Visual technique for identifying when the uniting theorem can be applied
- n input variables = n-dimensional "cube"



Mapping Truth Tables onto Boolean Cubes

- Uniting theorem combines two "faces" of a cube into a larger "face"
- Example:

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



Two faces of size 0 (nodes)
Combine into a face of size 1(line)

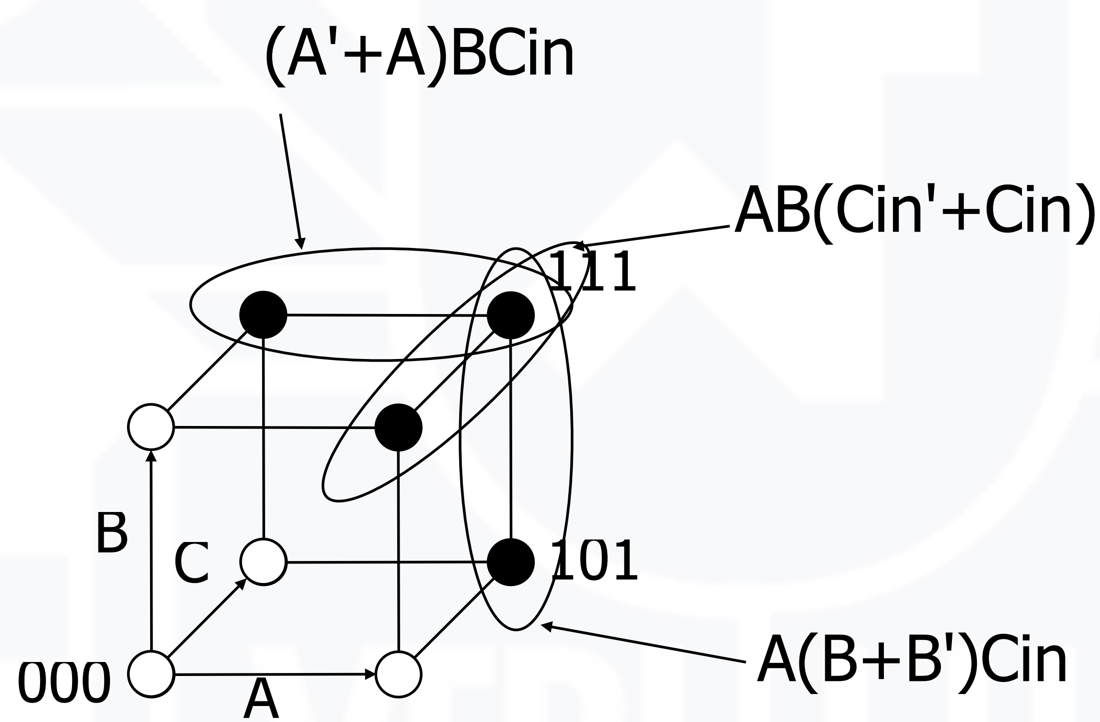
A varies within face, B does not
this face represents the literal B'

ON-set = solid nodes
OFF-set = empty nodes
DC-set = ×'d nodes

Three Variable Example

- Binary full-adder carry-out logic

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

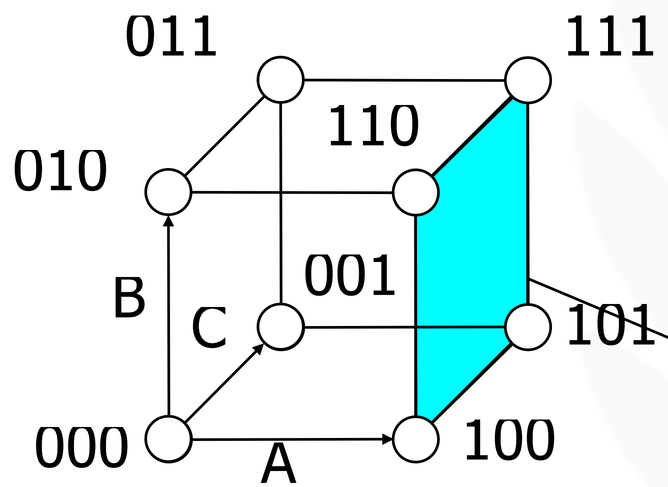


The on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that "111" is covered three times

Cout = BCin+AB+ACin

Higher Dimensional Cubes

- Sub-cubes of higher dimension than 2



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

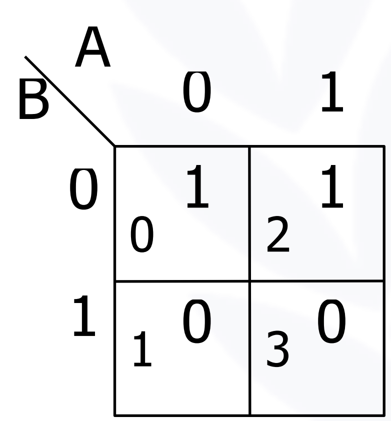
This subcube represents the literal A

m-Dimensional Cubes in a n-Dimensional Boolean Space

- In a 3-cube (three variables):
  - A 0-cube, i.e., a single node, yields a term in 3 literals
  - A 1-cube, i.e., a line of two nodes, yields a term in 2 literals
  - A 2-cube, i.e., a plane of four nodes, yields a term in 1 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

Karnaugh Maps

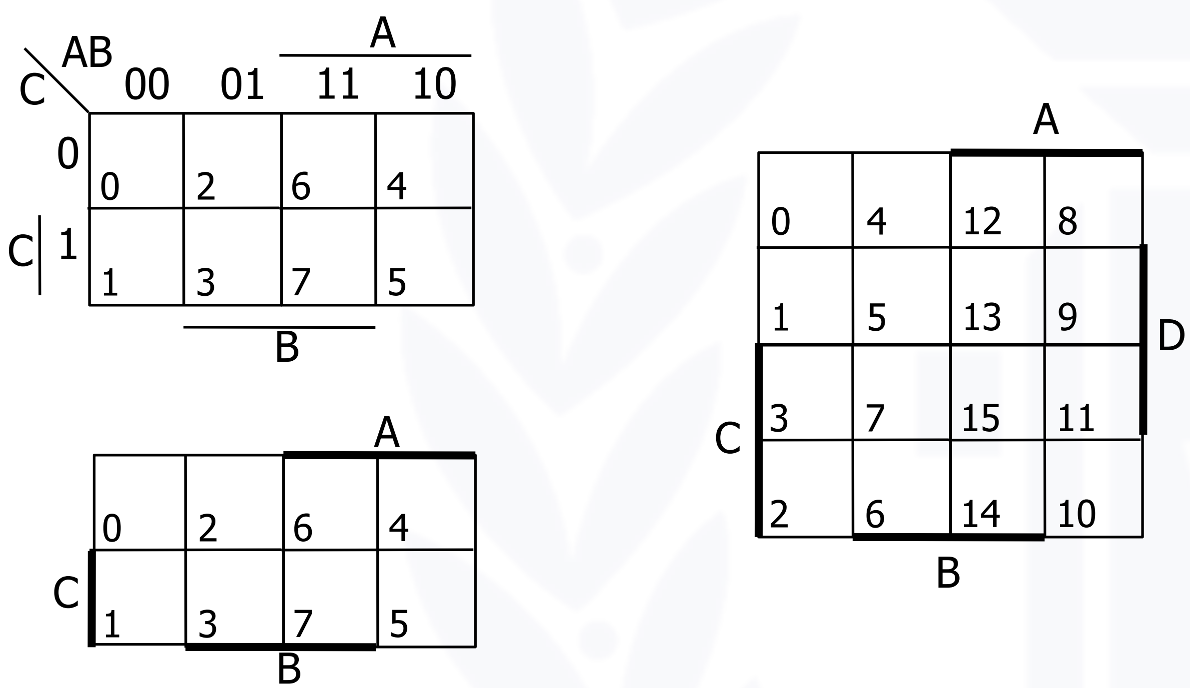
- Flat map of Boolean cube
  - Wrap–around at edges
  - Hard to draw and visualize for more than 4 dimensions
  - Virtually impossible for more than 6 dimensions
- Alternative to truth-tables to help visualize adjacencies
  - Guide to applying the uniting theorem
  - On-set elements with only one variable 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 (cont'd)

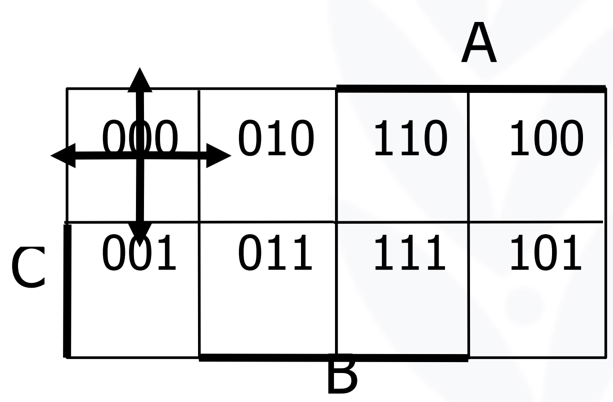
- Numbering scheme based on Gray–code
  - e.g., 00, 01, 11, 10
  - Only a single bit changes in code for adjacent map cells

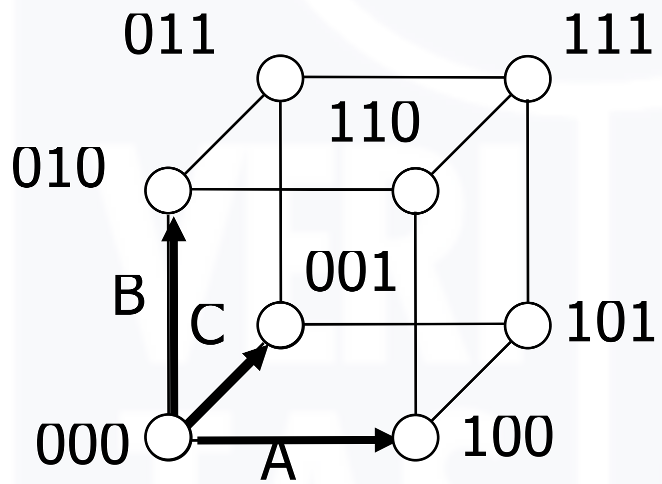


13 = 1101= ABC'D

Adjacencies in Karnaugh Maps

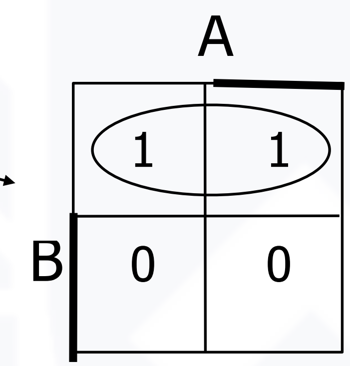
- Wrap from first to last column
- Wrap top row to bottom row



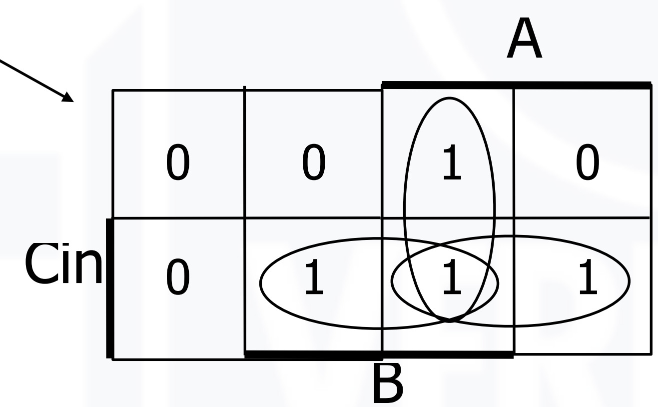


Karnaugh Map Examples

- F =
- Cout =
- f(A,B,C) = Σm(0,4,5,7)

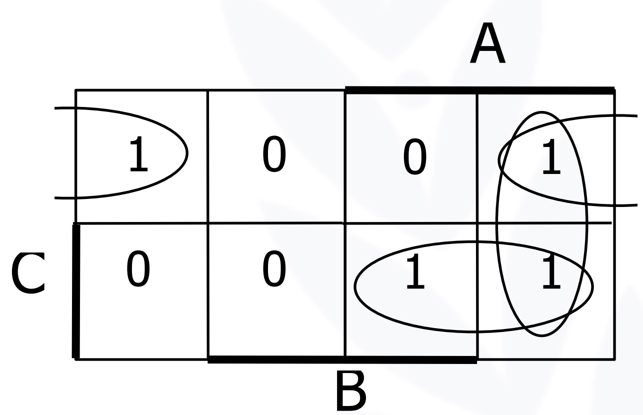


B′



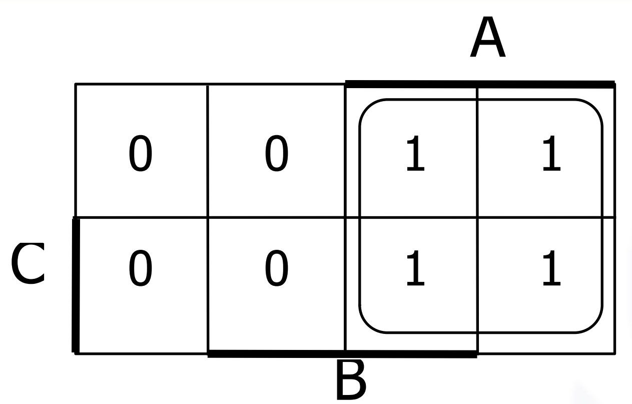
AB + ACin + BCin

obtain the complement of the function by covering 0s with subcubes

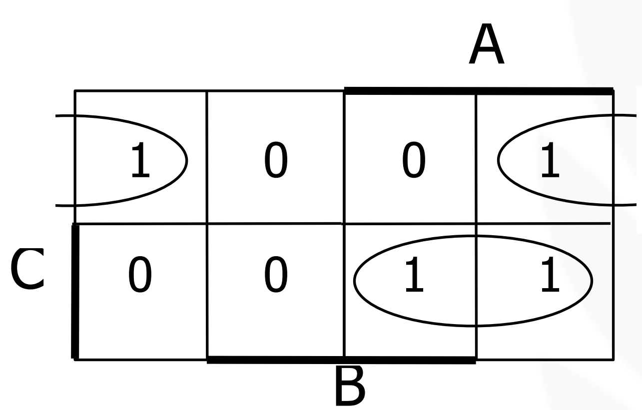


AC + B′C′ + ~~AB′~~

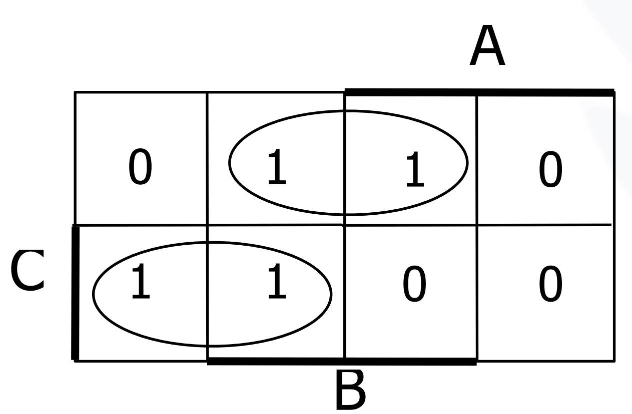
More Karnaugh Map Examples



G(A,B,C) = A



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



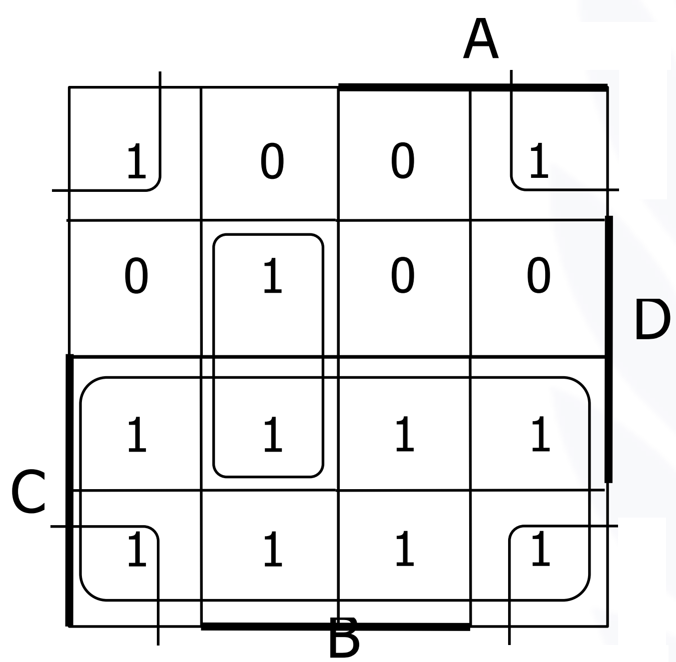
F' simply replace 1's with 0's and vice versa

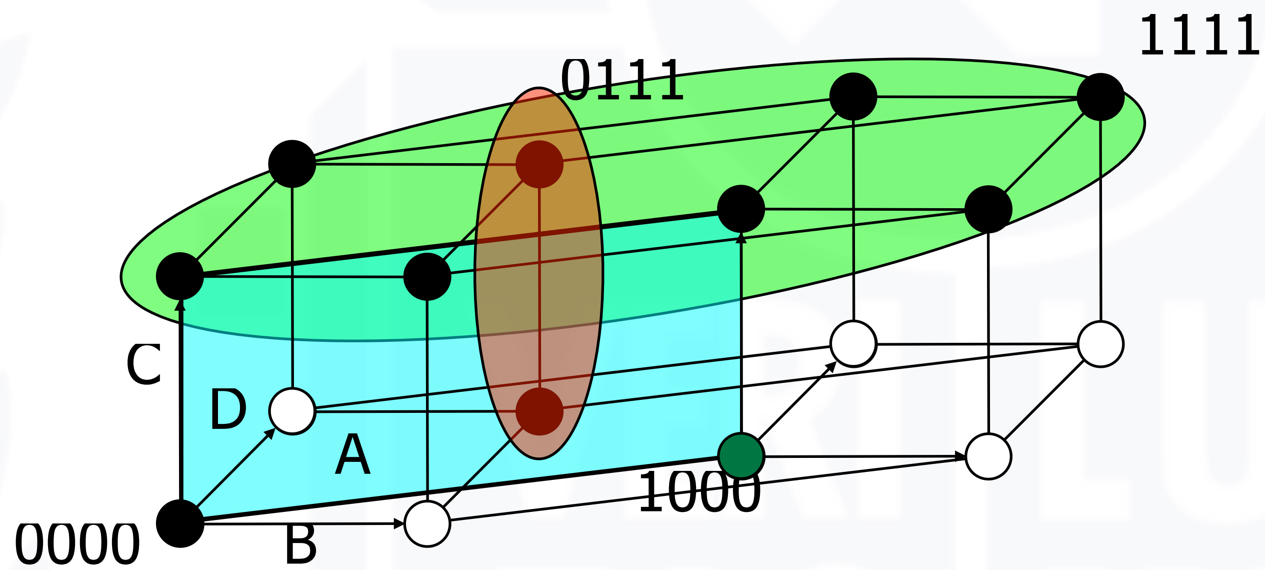
F'(A,B,C) = Σ m(1,2,3,6) = BC' + A'C

Karnaugh Map: 4-Variable Example

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

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

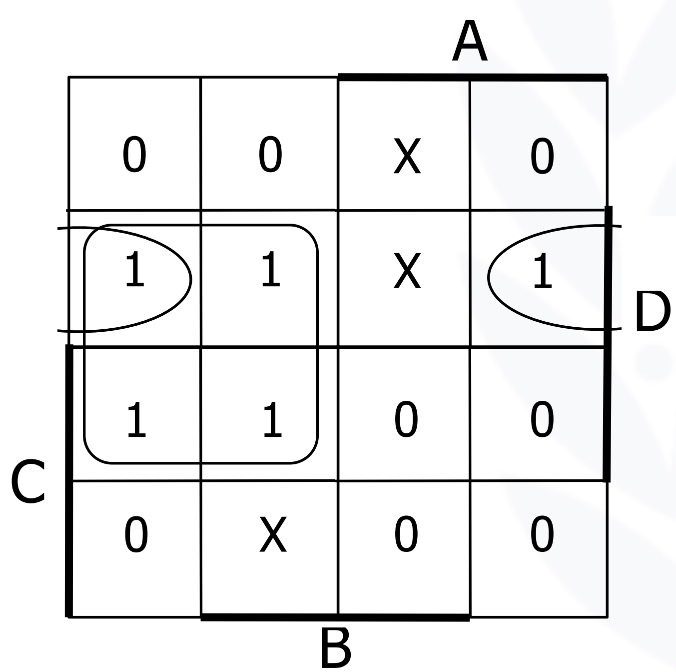




Find the smallest number of the largest possible subcubes to cover the ON-set (fewer terms with fewer inputs per term)

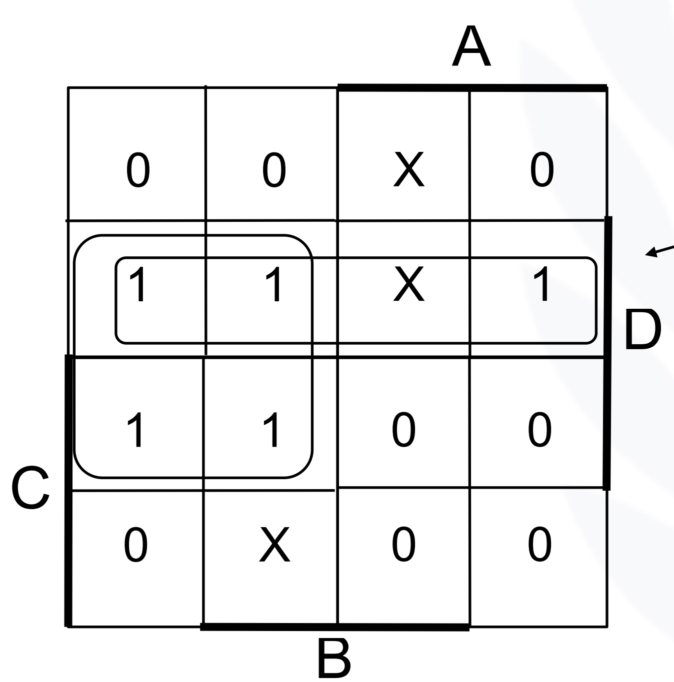
Karnaugh Maps: Don’t Cares

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



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

Don't cares can be treated as 1s or 0s depending on which is more advantageous

Activity

- Minimize the function $F = \Sigma\, m(0, 2, 7, 8, 14, 15) + d(3, 6, 9, 12, 13)$

Combinational Logic Summary

- Logic functions, truth tables, and switches
  - NOT, AND, OR, NAND, NOR, XOR, . . ., minimal set
- Axioms and theorems of Boolean algebra
  - Proofs by re-writing and perfect induction
- Gate logic
  - Networks of Boolean functions and their time behavior
- Canonical forms
  - Two-level and incompletely specified functions
- Simplification
  - A start at understanding two-level simplification
- Later
  - Automation of simplification
  - Multi-level logic
  - Time behavior
  - Hardware description languages
  - Design case studies