Ch 2. Combinational Logic

Combinational logic

- Define
  - The kind of digital system whose output behavior depends only on the current 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 property 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 :

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

$$2)\ \overline{A} + B \bullet C = \left(\overline{\mathrm{A}}\right) + \left(B \bullet C\right)$$

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):

1. the set B contains at least two elements: a, b		
2. closure:	a + b is in B	a • b is in B
3. identity:	$a + 0 = a$	$a \bullet 1 = a$
4. complementarity:	$a + a' = 1$	$a \bullet a' = 0$
5. commutativity:	$a + b = b + a$	$a \bullet b = b \bullet a$
6. distributivity:	$a + (b \bullet c) = (a + b) \bullet (a + c)$	$a \bullet (b + c) = (a \bullet b) + (a \bullet c)$
7. associativity:	$a + (b + c) = (a + b) + c$	$a \bullet (b \bullet c) = (a \bullet b) \bullet c$
(redundant)	$= a + b + c$	$= a \bullet b \bullet c$

Axioms/theorems of Boolean algebra

- Operations with 0 and 1:
  1. $X \bullet 1 = X$ — 1D. $X + 0 = X$
  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. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z) = X \bullet Y \bullet Z$

Axioms/theorems of Boolean algebra (cont'd)

- 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. $\{f(X_1, X_2, \ldots, X_n, 0, 1, +, \bullet)\}' = \{f(X_1', X_2', \ldots, X_n', 1, 0, \bullet, +)\}$

Axioms/theorems of Boolean algebra (cont'd)

- Duality:

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

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

- General form:

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

- Theorem for multiplying and factoring

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

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

- Consensus theorem:

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

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

Axioms/theorems of Boolean algebra (cont'd)

- Verifying the Boolean theorems using the axioms of Boolean algebra:
  - e.g., the Uniting theorem(9): X • Y + X • Y' = X?

Distributive law (8)	X • (Y + Y')	=	X
Complementarity theorem (5)	X • (1)	=	X
Identity (1D)	X	=	X ✓

  - e.g., the Simplification theorem(10): X + X • Y = X?

Identity (1D)	X • 1 + X • Y	=	X
Distributive law (8)	X(1 + Y)	=	X
Commutative law (6)	X(Y + 1)	=	X
Identity (2)	X(1)	=	X
Identity (1)	X	=	X ✓

Duality and DeMorgan's law

- 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)
  - allow to derive new theorems

    : (e.g.) the dual of the Uniting theorem(9), X • Y + X • Y' = X, is

    (X + Y) • (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) • (X + Y') = X?	
X + (Y •Y') = X	Distributive law (8D)
X + 0 = X	Complementarity theorem (5D)
X = X ✓	Identity (1)

Duality and DeMorgan's law

- DeMorgan's law
  - 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 $Z = \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}$
    $\overline{Z} = (\overline{\overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + AB\overline{C}})$
    $\overline{Z} = \overline{\overline{A}\,\overline{B}C} \bullet \overline{\overline{A}BC} \bullet \overline{A\overline{B}C} \bullet \overline{AB\overline{C}}$
    $\overline{Z} = (A + B + \overline{C})\ (A + \overline{B} + \overline{C})\ (\overline{A} + B + \overline{C})\ (\overline{A} + \overline{B} + C)$

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 not (X or Y)	X = Y	not Y	Y implies X	not X	X implies Y	X nand Y not (X and Y)	1

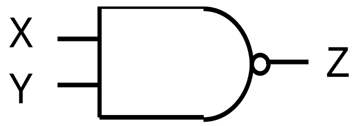
Cost of different logic functions

- 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 $\oplus$ Y (F6): require 16 switches

  - thus, because NOT, NOR, and NAND are the cheapest they are the functions we implement the most in practice

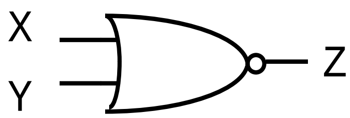
Realizing Boolean formulas (logic gates)

- NAND



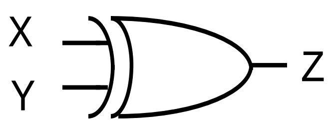
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR



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

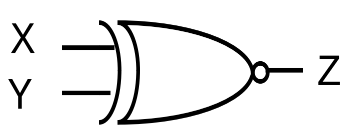
- XOR
  $X \oplus Y$



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

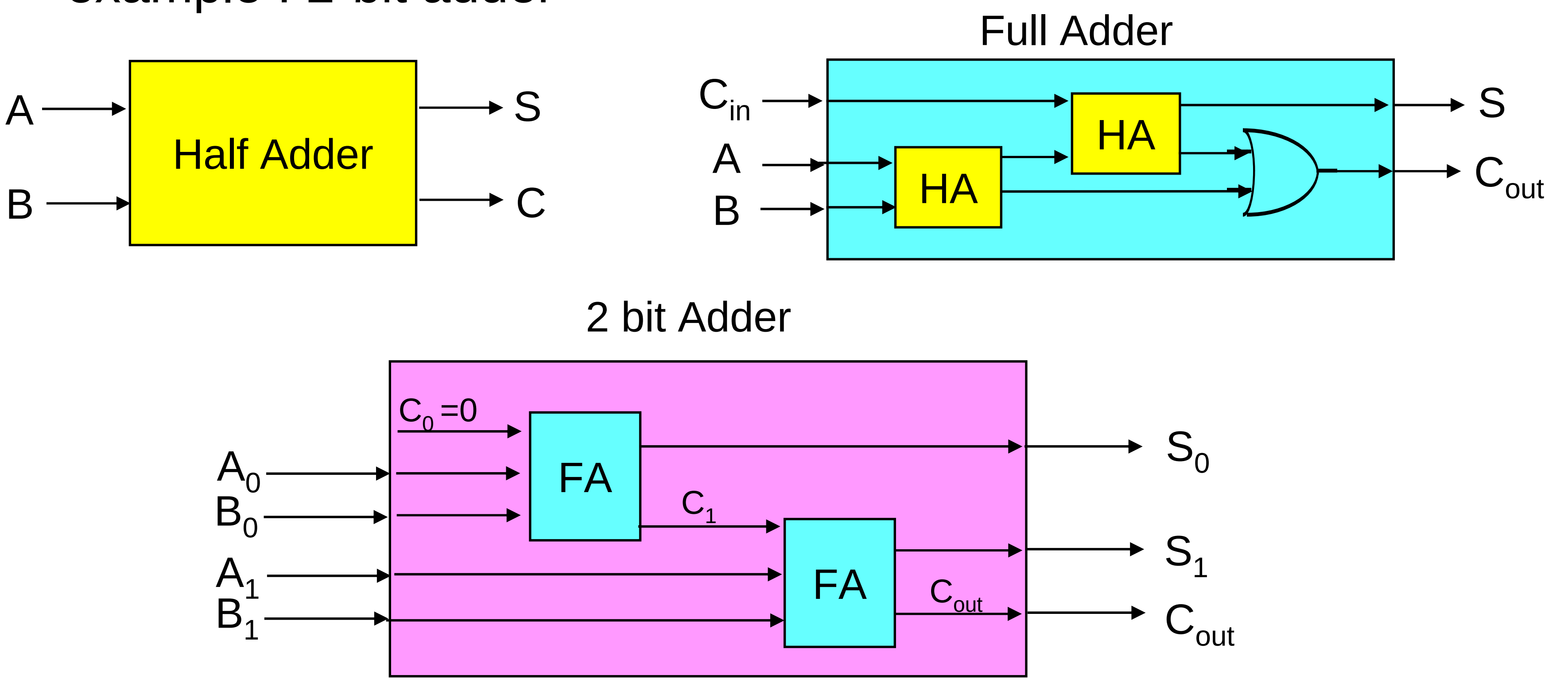


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

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

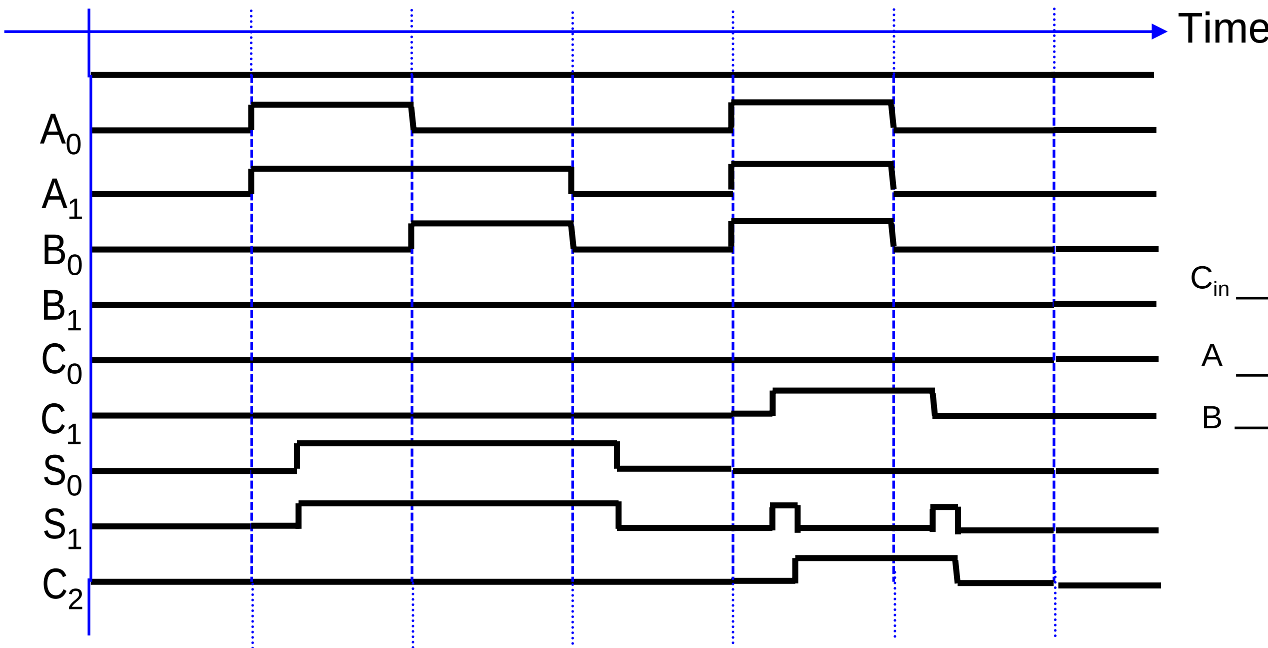
Realizing Boolean formulas (logic blocks and hierarchy)

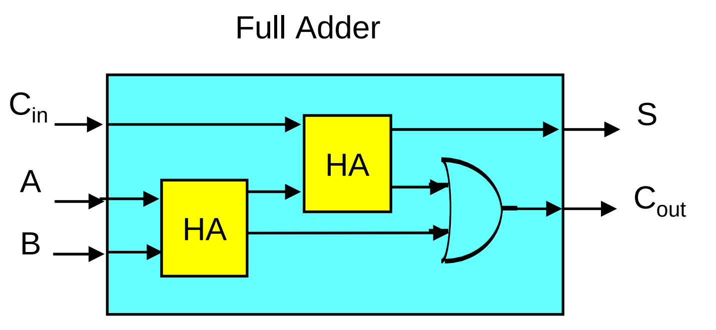
- Complex logic function can be constructed from more primitive functions by wiring up logic gates
- example : 2-bit adder



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



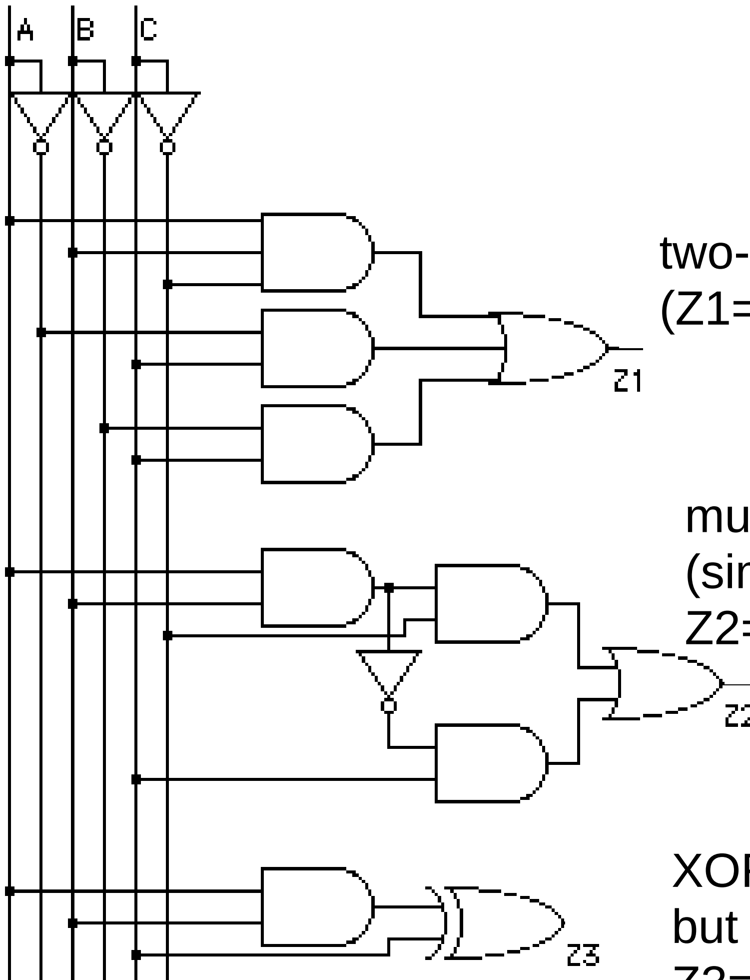


Minimizing the number of gates and wires

- Different implementations of one function

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

Z=A’B’C+A’BC+AB’C+ ABC’



two-level realization (Z1=ABC’+A’C+B’C)

multi-level realization (simple gates but long path; Z2=TC’+T’C, T=AB)

XOR gate (lowest gate count but the worst delay; Z3= (AB)⊕C

Two-level logic

- Canonical form
  - Standard form for a Boolean expression
  - Unique algebraic signature of the function
  - Two alternative forms
    - sum-of-products
    - product-of-sums
- Incompletely specified function
  - consider one more set: don’t care set

Sum-of-products canonical forms

- Also known as disjunctive normal form
- Also known as minterm expansion

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 (cont'd)

- 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:

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

F = *000* *010* *100*

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

Product-of-sums canonical form (cont'd)

- 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

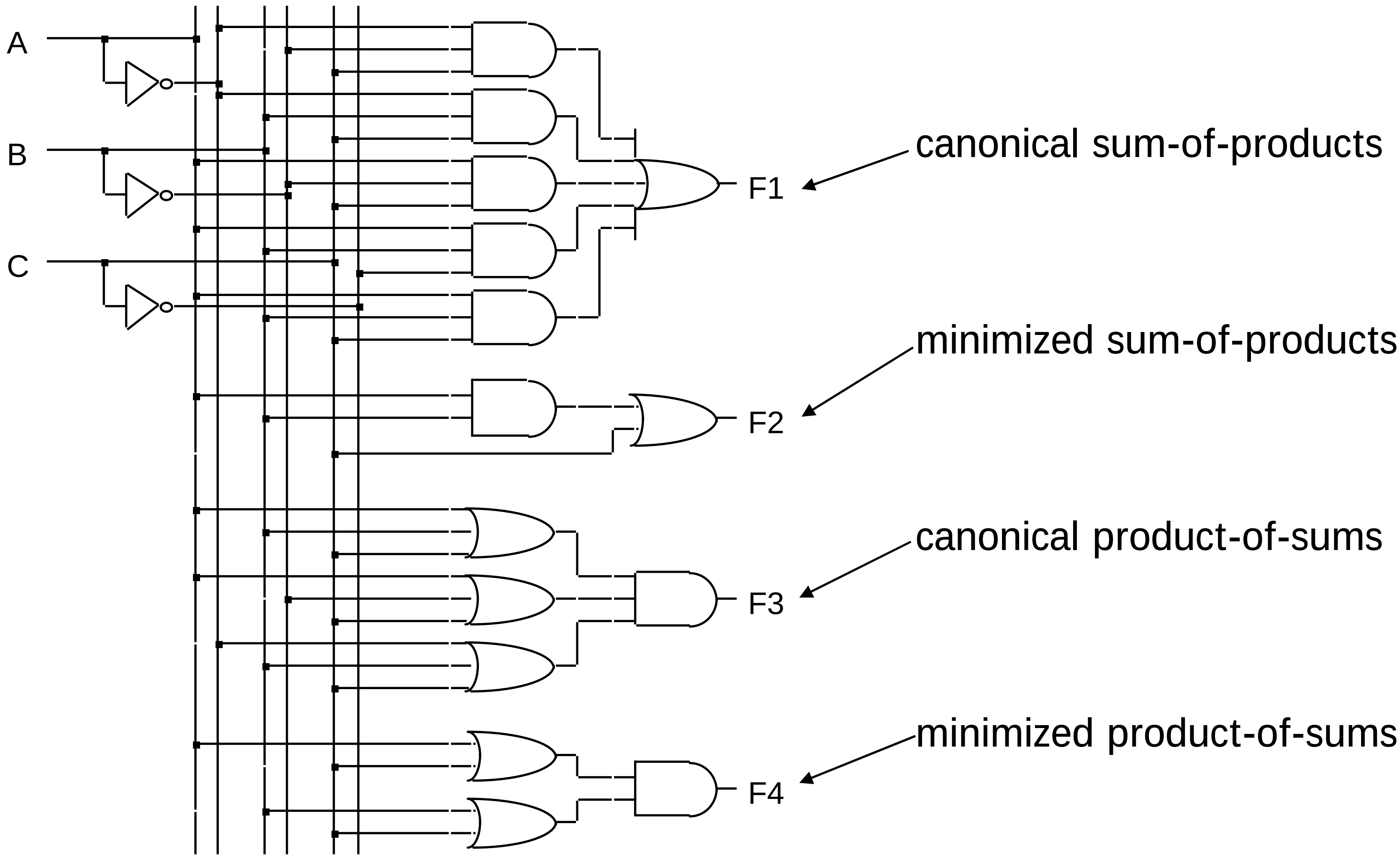
F in canonical form:

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

canonical form $\neq$ minimal form

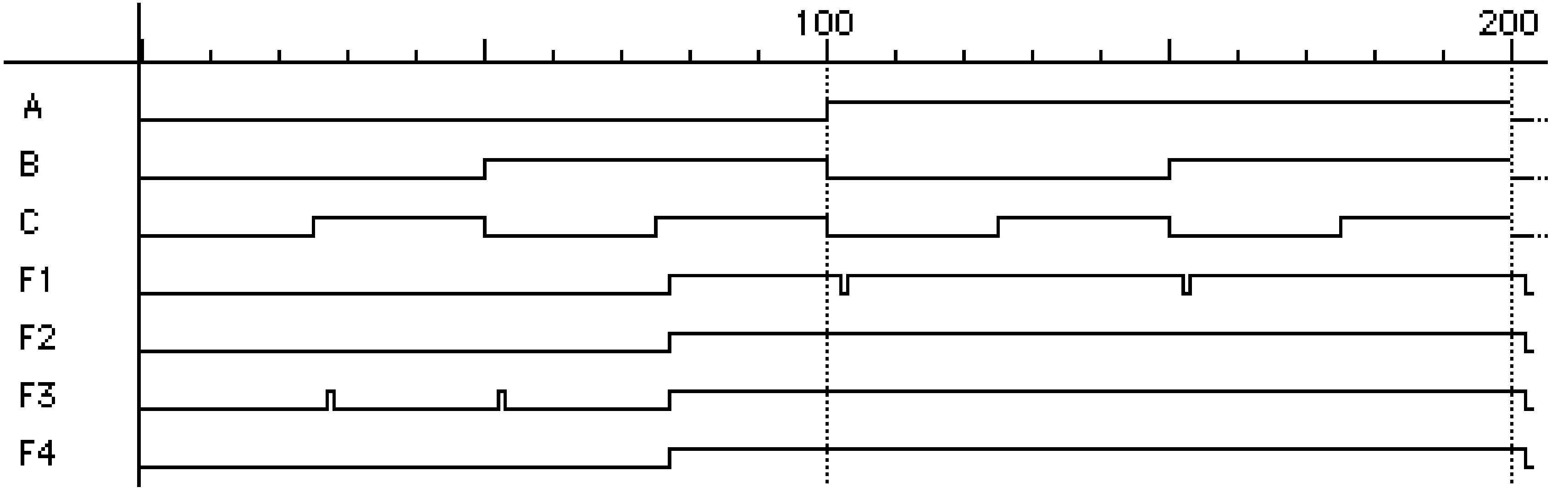
$$
\begin{aligned}
F(A, B, C) &= (A + B + C)\ (A + B' + C)\ (A' + B + C) \\
&= (A + B + C)\ (A + B' + C) \\
&\quad\ (A + B + C)\ (A' + B + C) \\
&= (A + C)\ (B + C)
\end{aligned}
$$

Four alternative two-level implementations of F



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)



S-o-P, P-o-S, and DeMorgan's theorem

- 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

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

Incompletely specified functions

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

Simplification of two-level combinational logic

- Finding a minimal sum of products or product of sums 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)

The essence of Boolean simplification

- Key tool to simplification: the Uniting theorem

  $\rightarrow A\,(B' + B) = A$

- 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

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

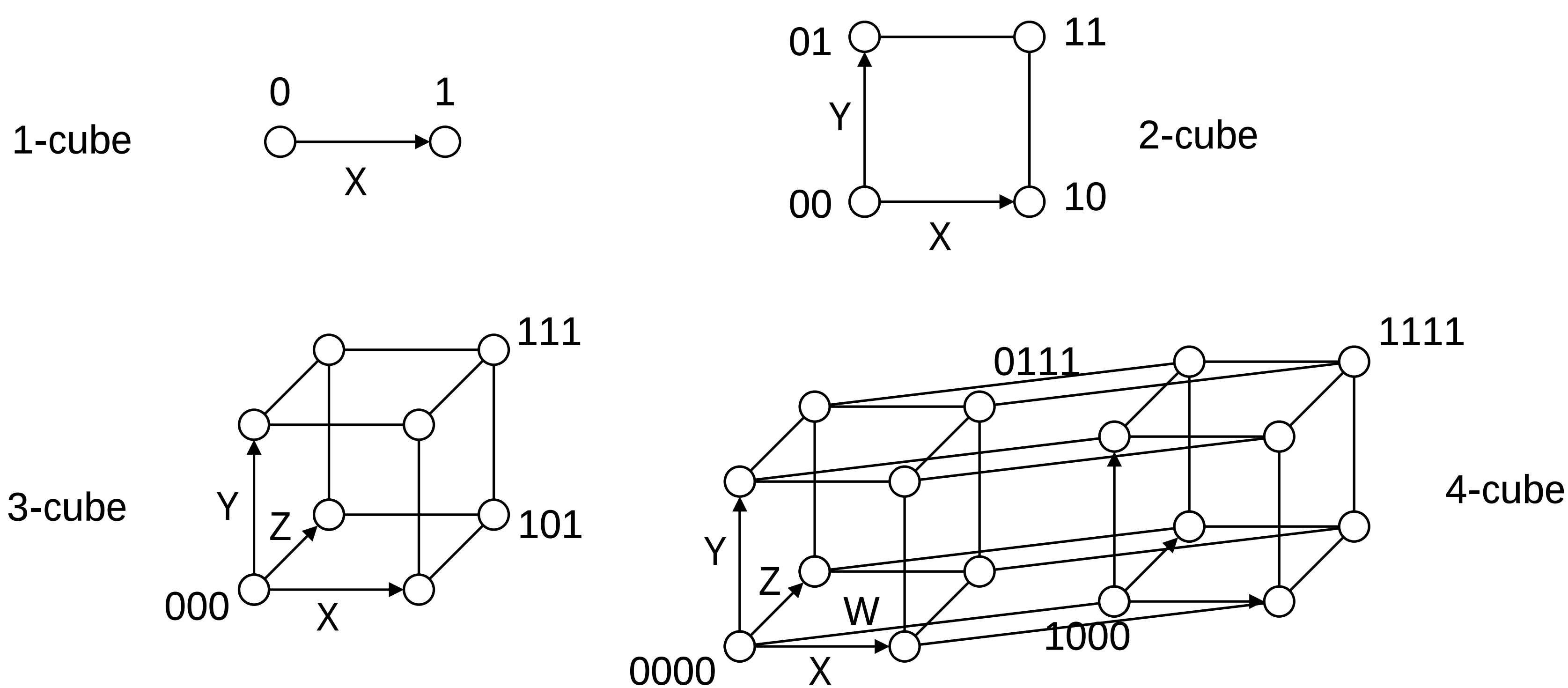
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

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

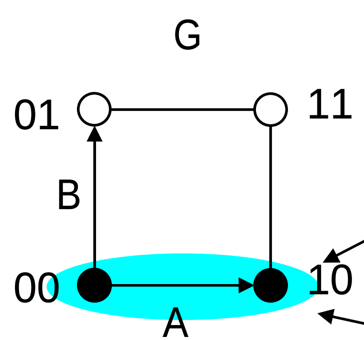
- Visual technique for identifying when the uniting theorem can be applied
- n input variables = n-dimensional "cube"



Mapping truth tables onto Boolean cubes

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



two faces of size 0 (nodes) combine into a face of size 1(line)

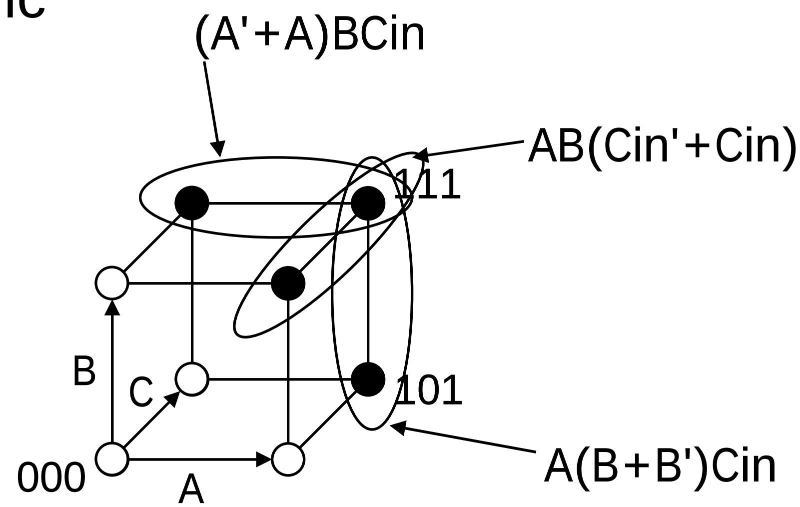
A varies within face, B does not
this face represents the literal B'

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

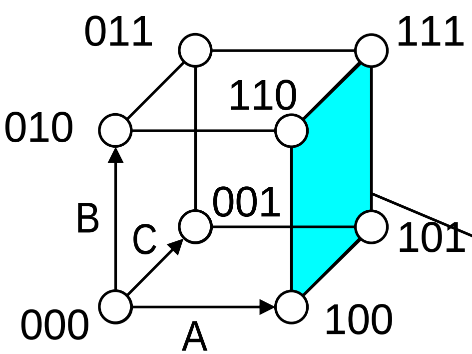


the on-set is completely covered by the combination (OR) of the subcubes of lower dimensionality - note that “111” is covered three times

Cout = BCin+AB+ACin

Higher dimensional cubes

- Sub-cubes of higher dimension than 1



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

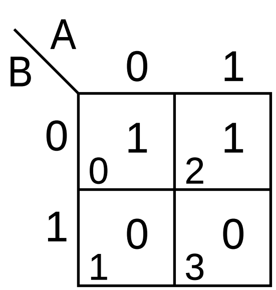
This subcube represents the literal A

m-dimensional cubes in an n-dimensional Boolean space

- In a 3-cube (three variables):
  - a 0-dimensional plane, i.e., a single node, yields a term in 3 literals
    - example : 101 = AB’C
  - a 1-dimensional plane, i.e., a line of two nodes, yields a term in 2 literals
    - example : 100-101 = AB’
  - a 2-dimensional plane, i.e., a plane of four nodes, yields a term in 1 literal
    - example : 100-101-111-110 = A
  - a 3-dimensional plane, i.e., a cube of eight nodes, yields a constant logic "1"
- In general,
  - an m-dimensional adjacency plane within an n-cube (m < n) yields a term with n – m literals

Karnaugh maps

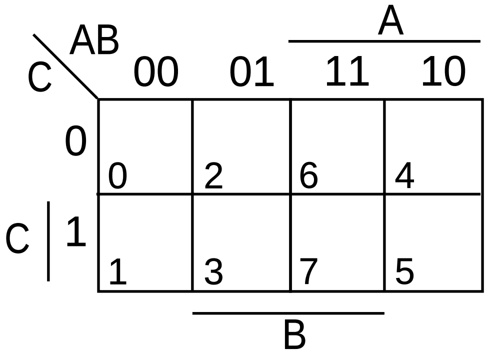
- The problem for humans
  - difficulty of visualizing adjacencies in more than 3 dimensional cubes
- Karnaugh maps
  - Alternative reformulation of the truth table
  - at least for expressions up to six variables
  - wrap–around at edges
  - on-set elements with only one variable changing value are adjacent unlike the situation in a linear truth-table

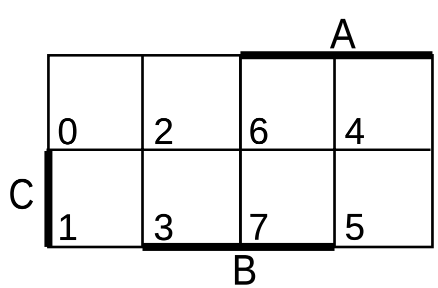


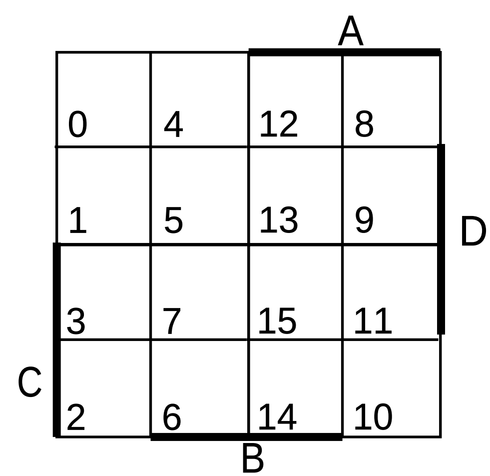
A	B	F
0	0	1
0	1	0
1	0	1
1	1	0

Karnaugh maps (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



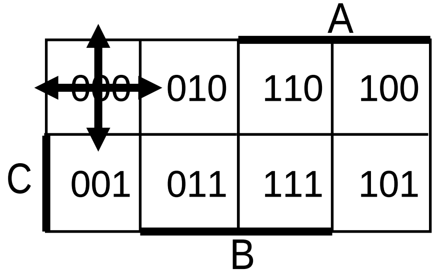


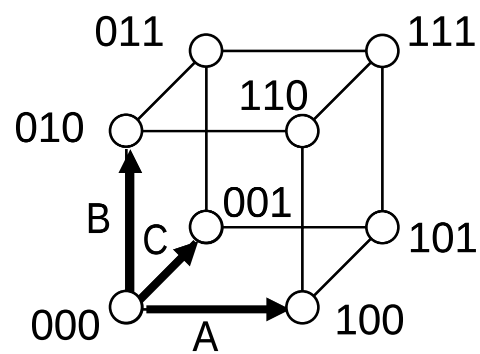


13 = 1101= ABC'D

Adjacencies in Karnaugh maps

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

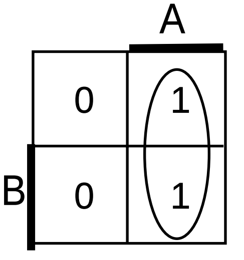




Karnaugh map examples

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

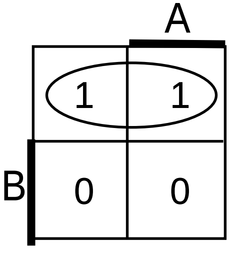
A	B	F
0	0	0
0	1	0
1	0	1
1	1	1



A

  - G = A’B’ + AB’ = B’

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

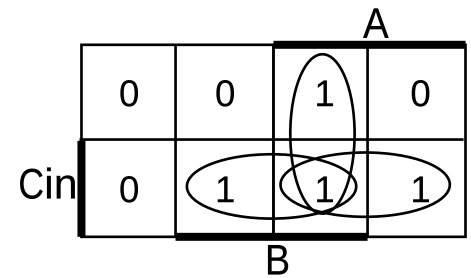


B’

Karnaugh map examples (cont'd)

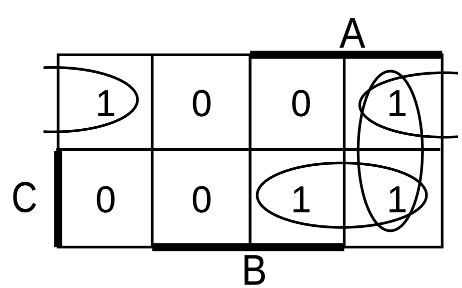
- 3-variable maps
  - Full adder

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



AB + Bcin + ACin

- F(A,B,C) = Σm(0,4,5,7)



AC + B'C' + ~~AB'~~

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

More Karnaugh map examples

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

G(A,B,C) = A

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

F(A,B,C) = Σm(0,4,5,7) = AC + B’C’

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

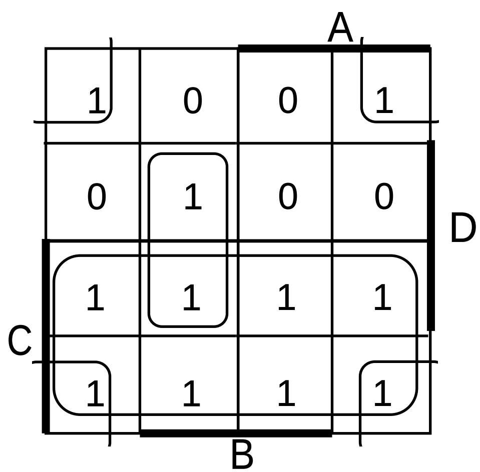
Complement of F(A,B,C) = m(0,4,5,7)
F' simply replace 1's with 0's and vice versa

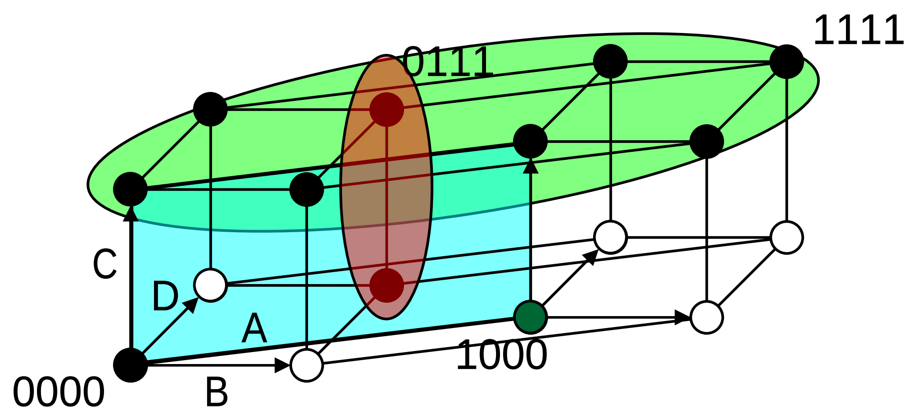
F'(A,B,C) = Σ m(1,2,3,6) = BC’ + A’C

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’

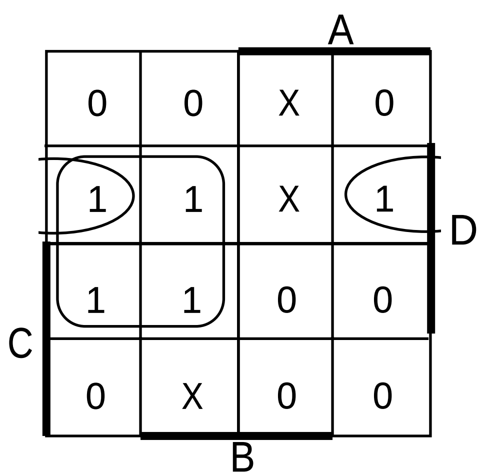




find the smallest number of the largest possible subcubes to cover the ON-set
(fewer terms with fewer inputs per term)

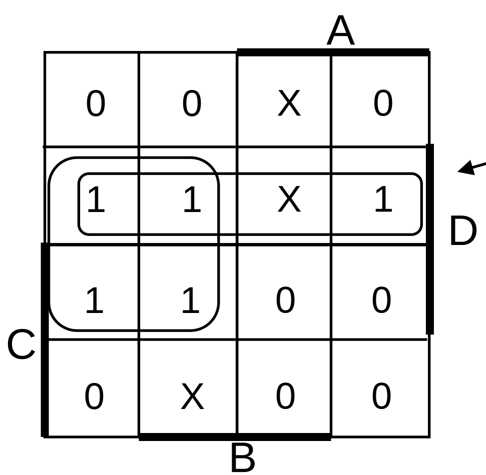
Karnaugh maps: don’t cares

- f(A,B,C,D) = Σ 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 (cont'd)

- 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

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 because of the increased levels of logic
- Example
  - 2-level logic

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

    → six 3-input AND gates and 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 and 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