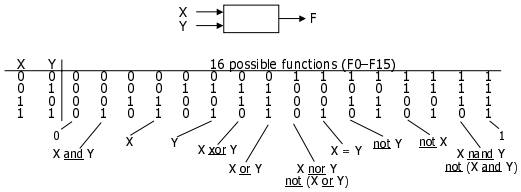
Combinational logic

- Logic functions, truth tables, and switches
  - NOT, AND, OR, NAND, NOR, XOR, . . .
  - Minimal set
- Axioms and theorems of Boolean algebra
  - Proofs by re-writing
  - Proofs by perfect induction
- Gate logic
  - Networks of Boolean functions
  - Time behavior
- Canonical forms
  - Two-level
  - Incompletely specified functions

Possible logic functions of two variables

- 16 possible functions of 2 input variables:
  - 2**(2**n) functions of n inputs



Cost of different logic functions

- Some are easier, others 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
  - Because NOT, NOR, and NAND are the cheapest they are the functions we implement the most in practice

Minimal set of functions

- Implement any logic functions from NOT, NOR, and NAND?
  - For example, implementing X and Y is the same as implementing not (X nand Y)
- 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", i.e., 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))

- Based on the mathematical foundations of logic: Boolean Algebra

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

Boolean algebra

- Boolean algebra
  - B = {0, 1}
  - + is logical OR, • is logical AND
  - ' is logical NOT
- All algebraic axioms hold

Logic functions and Boolean algebra

- Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: ', +, and ·

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 · Y) + (X' · Y') ⇔ 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

Axioms and theorems of Boolean algebra

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

Axioms and theorems of Boolean algebra (cont'd)

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

Axioms and theorems of Boolean algebra (cont')

- de Morgan's:
  - 14. (X + Y + ...)' = X' · Y' · ... 14D. (X · Y · ...)' = X' + Y' + ...
- generalized de Morgan's:
  - 15. f'(X1,X2,...,Xn,0,1,+,·) = f(X1',X2',...,Xn',1,0,·,+)
- establishes relationship between · and +

Axioms and theorems of Boolean algebra (cont')

- Duality
  - 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!
  - Meta-theorem (a theorem about theorems)
- duality:
  - 16. X + Y + ... ⇔ X · Y · ...
- generalized duality:
  - 17. f (X1,X2,...,Xn,0,1,+,·) ⇔ f(X1,X2,...,Xn,1,0,·,+)
- Different than deMorgan's Law
  - this is a statement about theorems
  - this is not a way to manipulate (re-write) expressions

Proving theorems (rewriting)

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

distributivity (8)	X · Y + X · Y'	= X · (Y + Y')
complementarity (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
distributivity (8)	X · 1 + X · Y	= X · (1 + Y)
identity (2)	X · (1 + Y)	= X · (1)
identity (1D)	X · (1)	= X ✓

Proving theorems (perfect induction)

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

$(X + Y)' = X' \cdot Y'$
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

$(X \cdot Y)' = X' + Y'$
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

A simple example

- 1-bit binary adder
  - inputs: A, B, Carry-in
  - outputs: Sum, Carry-out

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

S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin

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

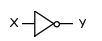
Apply the theorems to simplify expressions

- The theorems of Boolean algebra can simplify Boolean expressions
  - e.g., full adder's carry-out function (same rules apply to any function)

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

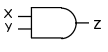
From Boolean expressions to logic gates

- NOT X' $\bar{X}$ ~X



X	Y
0	1
1	0

- AND X · Y XY $X \wedge Y$



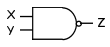
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

- OR X + Y $X \vee Y$

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

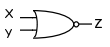
From Boolean expressions to logic gates (cont'd)

- 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")

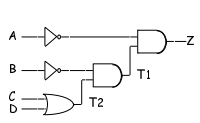
- XNOR X = Y

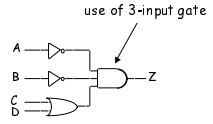
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

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

From Boolean expressions to logic gates (cont'd)

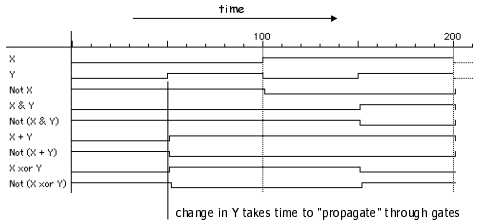
- More than one way to map expressions to gates
  - e.g., $Z = A' \cdot B' \cdot (C + D) = (A' \cdot (B' \cdot \overline{(C + D)}))$ with $\overline{(C + D)}$ = T2, T1





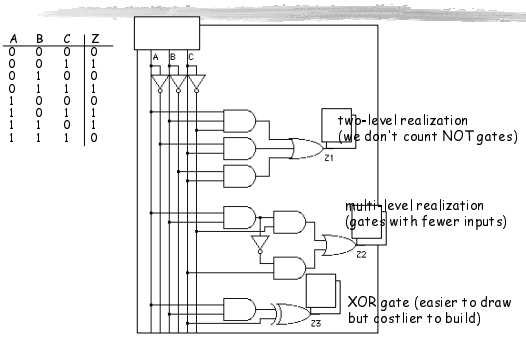
Waveform view of logic functions

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



Choosing different realizations of a 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



Which realization is best?

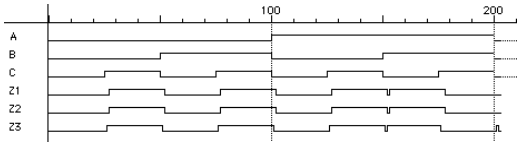
- Reduce number of inputs
  - literal: input variable (complemented or not)
    - can approximate cost of logic gate as 2 transistors per literal
    - why not count inverters?
  - Fewer literals means less transistors
    - smaller circuits
  - Fewer inputs implies faster gates
    - gates are smaller 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? (cont'd)

- 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 inputs, the alternative implementations have almost the same waveform behavior
  - delays are different
  - glitches (hazards) may arise
  - variations due to differences in number of gate levels and structure
- 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
- Many alternative gate realizations may have the same truth table
- 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

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, in true or inverted form (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'
= 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, in true or inverted form (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) = ΠM(0,2,4)
= M0 • M2 • M4
= (A + B + C) (A + B' + C) (A' + B + C)

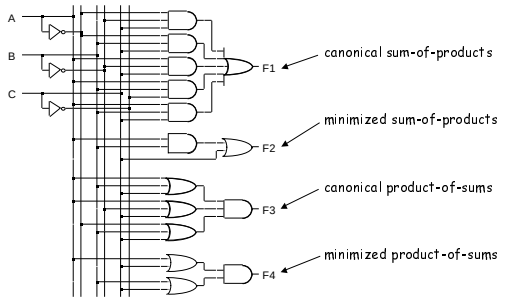
canonical form ≠ minimal form

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)

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

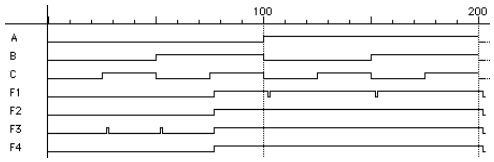
- Sum-of-products
  - F' = A'B'C' + A'BC' + AB'C'
- Apply de Morgan's
  - (F')' = (A'B'C' + A'BC' + AB'C')'
  - F = (A + B + C) (A + B' + C) (A' + B + C)
- Product-of-sums
  - F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')
- Apply de Morgan's
  - (F')' = ((A + B + C')(A + 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



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)



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

Incompleteley specified functions

- Example: binary coded decimal increment by 1
  - BCD digits encode decimal digits 0 – 9 in 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:
  - $Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15$
  - $Z = \Sigma\ [\ m(0,2,4,6,8) + d(10,11,12,13,14,15)\ ]$
  - $Z = M1 \cdot M3 \cdot M5 \cdot M7 \cdot M9 \cdot D10 \cdot D11 \cdot D12 \cdot D13 \cdot D14 \cdot D15$
  - $Z = \Pi\ [\ M(1,3,5,7,9) \cdot D(10,11,12,13,14,15)\ ]$

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)

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
- Later
  - Two-level simplification using K-maps
  - Automation of simplification
  - Multi-level logic
  - Design case studies
  - Time behavior