Combinational logic

- Basic logic
  - Boolean algebra, proofs by re-writing, proofs by perfect induction
  - logic functions, truth tables, and switches
  - NOT, AND, OR, NAND, NOR, XOR, . . ., minimal set
- Logic realization
  - two-level logic and canonical forms
  - incompletely specified functions
- Simplification
  - uniting theorem
  - grouping of terms in Boolean functions
- Alternate representations of Boolean functions
  - cubes
  - Karnaugh maps

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

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

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 . . .
  - lets look at the mathematical foundation of logic

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		
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}
  - variables
  - + 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. $X \bullet 1 = X$
- 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. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$

Axioms and theorems of Boolean algebra (cont'd)

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

Axioms and theorems of Boolean algebra (cont'd)

- de Morgan's:
  14. $(X + Y + \ldots)' = X' \bullet Y' \bullet \ldots$ 14D. $(X \bullet Y \bullet \ldots)' = X' + Y' + \ldots$
- generalized de Morgan's:
  15. $f'(X_1,X_2,\ldots,X_n,0,1,+,\bullet) = f(X_1',X_2',\ldots,X_n',1,0,\bullet,+)$

- establishes relationship between • and +

Axioms and theorems of Boolean algebra (cont'd)

- 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:

  16. $X + Y + \ldots \Leftrightarrow X \bullet Y \bullet \ldots$
- generalized duality:

  17. $f(X_1, X_2, \ldots, X_n, 0, 1, +, \bullet) \Leftrightarrow f(X_1, X_2, \ldots, X_n, 1, 0, \bullet, +)$

- 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: $X \cdot Y + X \cdot Y' = X$

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

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

identity (1D)	$X + X \cdot Y$	$= X \cdot 1 + X \cdot Y$
distributivity (8)	$X \cdot 1 + X \cdot Y$	$= X \cdot (1 + Y)$
identity (2)	$X \cdot (1 + Y)$	$= X \cdot (1)$
identity (1D)	$X \cdot (1)$	$= X$ ✓

Activity

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

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

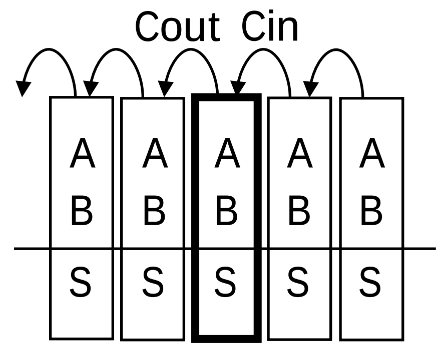
$(X \bullet Y)' = X' + Y'$

NAND is equivalent to OR with inputs complemented

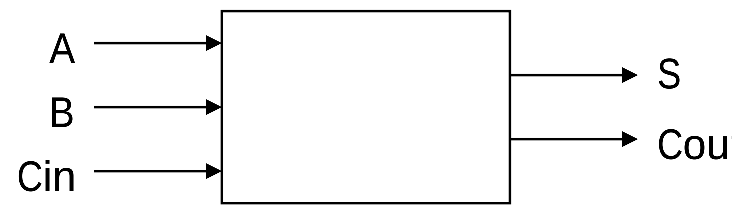
X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



A	B	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



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)

$$
\begin{aligned}
\text{Cout} &= A' B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= A' B\,Cin + A B' Cin + A B\,Cin' + \boxed{A B\,Cin + A B\,Cin} \\
&= A' B\,Cin + A B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= (A' + A) B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= (1) B\,Cin + A B' Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A B' Cin + A B\,Cin' + \boxed{A B\,Cin + A B\,Cin} \\
&= B\,Cin + A B' Cin + A B\,Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A (B' + B) Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A (1) Cin + A B\,Cin' + A B\,Cin \\
&= B\,Cin + A\,Cin + A B (Cin' + Cin) \\
&= B\,Cin + A\,Cin + A B (1) \\
&= B\,Cin + A\,Cin + A B
\end{aligned}
$$

adding extra terms creates new factoring opportunities

Activity

- Fill in the truth-table for a circuit that checks that a 4-bit number is divisible by 2, 3, or 5

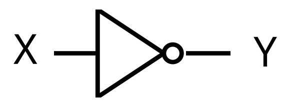
X8	X4	X2	X1	By2	By3	By5
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0

- Write down Boolean expressions for By2, By3, and By5

Activity

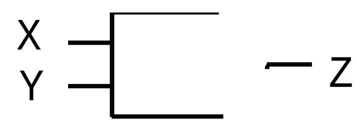
From Boolean expressions to logic gates

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



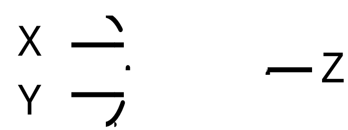
X	Y
0	1
1	0

- AND   X • Y   XY   X ∧ Y



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

- OR   X + Y   X ∨ Y



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

From Boolean expressions to logic gates (cont'd)

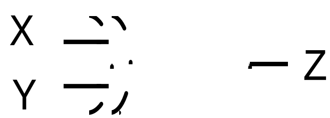
- NAND

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

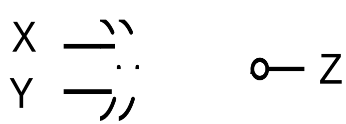
- XOR
  $X \oplus Y$



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

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

- XNOR
  X = Y

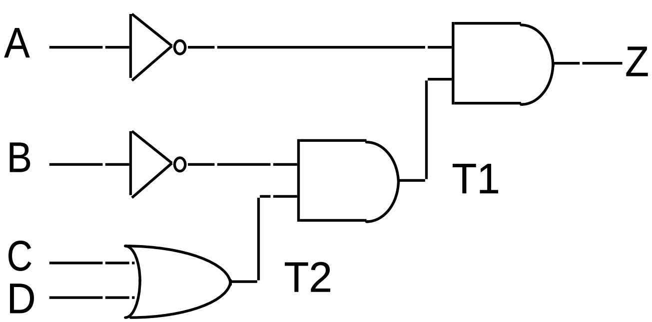


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

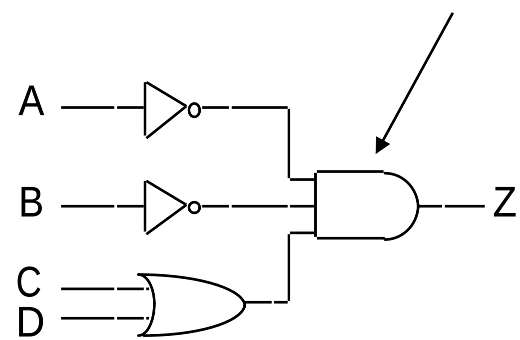
X <u>xnor</u> Y = X Y + X' Y'
X and Y are the same
("equality", "coincidence")

From Boolean expressions to logic gates (cont'd)

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



use of 3-input gate



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!