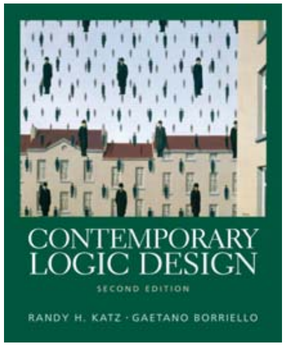
CSE 370 Spring 2006
Introduction to Digital Design
Lecture 3: Boolean Algebra



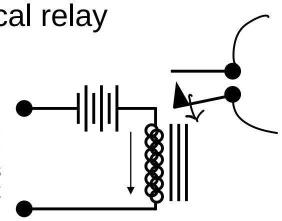
Last Lecture
- Binary and other bases
- Negative binary numbers
- Switches/CMOS

Today
- Basic Boolean Functions
- Boolean Algebra

Switching Networks

- Switch settings determine whether a conducting network to a light bulb
- Larger computations?
  - Use a light bulb (output) to set other switches (input)
  - Example: Mechanical relay

conducting
path composed
of switches
closes circuit



current flowing through coil
magnetizes core and causes normally
closed (nc) contact to be pulled open

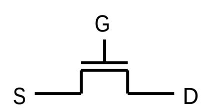
when no current flows, the spring of the contact
returns it to its normal position

Transistor Networks

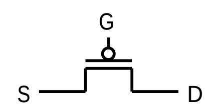
- Relays no more: slow and big
- Modern digital electronics predominately uses CMOS technology
  - MOS: metal-oxide semiconductor
  - C: complementary (both p and n type transistors arranged so that power is dissipated during switching.)

MOS Transistors

- MOS transistors have three terminals: drain, gate, and source
  - Act as switches: if the voltage on the gate terminal is (some amount) higher/lower than the source terminal then a conducting path will be established between the drain and source terminals.

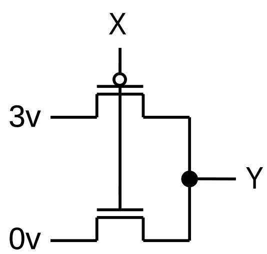


n-channel
open when voltage at G is low
closes when:
voltage(G) > voltage (S) + ε



p-channel
closed when voltage at G is low
opens when:
voltage(G) < voltage (S) – ε

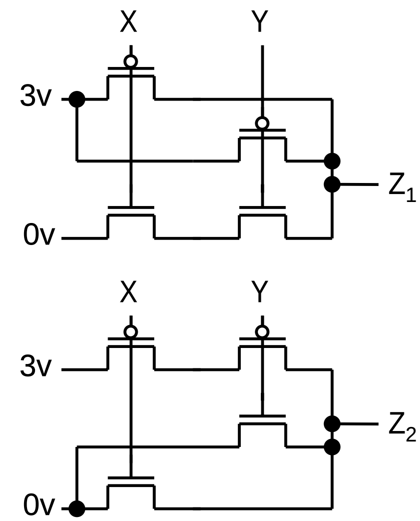
MOS Networks



what is the relationship between x and y?

x	y
0 volts	
3 volts	

Two Input Networks

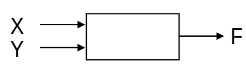


what is the relationship between x, y and z?

x	y	z1	z2
0 volts	0 volts		
0 volts	3 volts		
3 volts	0 volts		
3 volts	3 volts		

2 to 1 Boolean Functions

There are 16 possible two bit input one bit output



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

(General: k input bits, one output bit: 2^k such functions)

Costs

- 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

NOTs, NANDs, NORs cost the least

NOT, NOR, NANDS, Oh My!

- Can we implement all logic functions from NOT, NOR, NANDs?
- Example: Implementing NOT(X NAND Y)

  is the same as implementing (X AND Y)
- In fact we can implement a NOT using a NAND or a NOR:

  NOT(X) = X NAND X   NOT(X)= Y NOR Y

- In fact NAND and NOR can be used to implement each other:

  X NAND Y=NOT(NOT(X) NOR NOT(Y))

  X NOR Y=NOT(NOT(X) NAND NOT(Y))
- To sort through the mess of what we have created we will construct a mathematical framework: Boolean Algebra

Boolean Algebra

- A set of elements B together with a two binary operations, addition, {+}, and multiplication, {•} which satisfy the *axioms*:
  - B contains at least two nonequal elements
  - (closure) For every a,b in B

    a+b is in B   a•b is in B
  - (commutative) For every a,b in B

    a+b=b+a   a•b=b•a
  - (associative) For every a,b,c in B

    (a+b)+c=a+(b+c)   a•(b•c)=(a•b)•c
  - (identity) There exists identity elements for + and •, such that for every a in B

    a+0=a   a•1=a
  - (distributive) For every a,b,c in B

    a+(b•c)=(a+b)•(a+c)   a•(b+c)=(a•b)+(a•c)
  - (complement) For each a in B there exists an element a' in B, such that a+a'=1 and a•a'=0

A Boolean Algebra

- A Boolean Algebra:
  - the set B={0,1}
  - binary operation + = logical OR
  - binary operation • = logical AND
  - complement ' = logical NOT

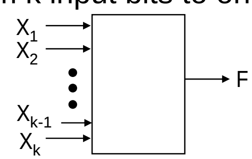
- These satisfy the above axioms

- We will often deal with variable representing an element from the set:

  Example: (X+Y)•(X+Z)

Boolean Functions

- Boolean Function
  - function from k input bits to one output bit



  - All such functions can be represented by a truth table

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

Boolean Functions and Algebra

- All Boolean Functions can be represented by an expression in Boolean Algebra using ANDs, ORs, and NOTs:

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

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

Universality of NAND/NOR

- All Boolean Functions can be represented by an expression in Boolean Algebra using ANDs, ORs, and NOTs.

But we can express AND, OR, and NOT in terms of NAND:

X' = X NAND X
X AND Y = (X NAND Y)'
X OR Y = (X' NAND Y')

But we can express AND, OR, and NOT in terms of NOR:
X' = X NOR X
X OR Y = (X NOR Y)'
X AND Y = (X' NOR Y')

Duality

- All Boolean expressions have logical duals
- Any theorem that can be proved is also proved for its dual
- Replace: • with +, + with •, 0 with 1, and 1 with 0
- Leave the variables unchanged

Example: X+0 = 0 is dual to X•1=1

Do not confuse Duality with de'Morgan's theorem.

Axioms and Theorems

1. Identity: X + 0 = X — Dual: X • 1 = X
2. Null: X + 1 = 1 — Dual: X • 0 = 0
3. Idempotent: X + X = X — Dual: X • X = X
4. Involution: (X')' = X
5. Complementarity: X + X' = 1 — Dual: X • X' = 0
6. Commutative: X + Y = Y + X — Dual: X • Y = Y • X
7. Associative: (X+Y)+Z=X+(Y+Z) — Dual: (X•Y)•Z=X•(Y•Z)
8. Distributive: X•(Y+Z)=(X•Y)+(X•Z) — Dual: X+(Y•Z)=(X+Y)•(X+Z)
9. Uniting: X•Y+X•Y'=X — Dual: (X+Y)•(X+Y')=X
10. Absorption: X + X • Y = X — Dual: X • (X + Y) = X
11. Absorption2: (X + Y') • Y = X • Y — Dual: (X • Y') + Y = X + Y
12. Factoring: (X + Y) • (X' + Z) = X • Z + X' • Y — Dual: X • Y + X' • Z = (X + Z) • (X' + Y)

Axioms and Theorems

13. Concensus: (X • Y) + (Y • Z) + (X' • Z) = X • Y + X' • Z

Dual: (X + Y) • (Y + Z) • (X' + Z) = (X + Y) • (X' + Z)

14. DeMorgan's Law: (X + Y + ...)' = X' • Y' • ...

Dual: (X • Y • ...)' = X' + Y' + ...

15. Generalized DeMorgan's Laws: f'(X1,X2,...,Xn,0,1,+,•) = f(X1',X2',...,Xn',1,0,•,+)

Notice the DeMorgan is not Duality: Duality is not a way to rewrite an expression, it is a meta-theorem:

16. Generalized Duality:

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

Proving Theorems

- Example 1: Prove the uniting theorem-- X•Y+X•Y'=X

Distributive	X•Y+X•Y' = X•(Y+Y')
Complementarity	= X•(1)
Identity	= X

- Example 2: Prove the absorption theorem-- X+X•Y=X

Identity	X+X•Y = (X•1)+(X•Y)
Distributive	= X•(1+Y)
Null	= X•(1)
Identity	= X

Exercise

- Example 3: Prove the consensus theorem-- (XY)+(YZ)+(X'Z)= XY+X'Z

Exercise

- Example 3: Prove the consensus theorem-- (XY)+(YZ)+(X'Z)= XY+X'Z

Complementarity	XY+YZ+X'Z = XY+(X+X')YZ + X'Z
Distributive	= XYZ+XY+X'YZ+X'Z

↙ *Use absorption {AB+A=A} with A=XY and B=Z*

	= XY+X'YZ+X'Z
Rearrange terms	= XY+X'ZY+X'Z

↙ *Use absorption {AB+A=A} with A=X'Z and B=Y*

XY+YZ+X'Z = XY+X'Z

Logic Simplification

- Use the axioms to simplify logical expressions
  - Why? To use less hardware
- Example: A two-level logic expression

Z = A'BC + AB'C' + AB'C + ABC' + ABC
= AB'C + AB'C' + A'BC + ABC' + ABC

rearrange

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

distributive

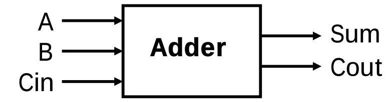
= AB' + A'BC + AB   comp.
= AB' + AB + A'BC   rearrange
= A(B' + B) + A'BC   distributive
= A + A'BC   comp.

Absorption #2D {(X •Y')+Y=X+Y} with X=BC and Y=A

Z = A + BC

Example: Full Adder

- 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'BCin' + AB'Cin' + ABCin

Cout = A'BCin + AB'Cin + ABCin' + ABCin

Simplification of Carry Out

Cout = A'BCin + AB'Cin + ABCin' + ABCin
= A'BCin + AB'Cin + ABCin' + ABCin + ABCin
= A'BCin + ABCin + AB'Cin + ABCin' + ABCin
= (A'+A)BCin + AB'Cin + ABCin' + ABCin
= (1)BCin + AB'Cin + ABCin' + ABCin
= BCin + AB'Cin + ABCin' + ABCin + ABCin
= BCin + AB'Cin + ABCin + ABCin' + ABCin
= BCin + A(B'+B)Cin + ABCin' + ABCin
= BCin + A(1)Cin + ABCin' + ABCin
= BCin + ACin + AB(Cin'+Cin)
= BCin + ACin + AB(1)
= BCin + ACin + AB

associative

idempotent