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

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.

![Diagram of MOS Transistors](image)
MOS Networks

what is the relationship between x and y?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 volts</td>
<td>3 volts</td>
</tr>
</tbody>
</table>

Two Input Networks

what is the relationship between x, y and z?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z1</th>
<th>z2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 volts</td>
<td>0 volts</td>
<td>3 volts</td>
<td>0 volts</td>
</tr>
<tr>
<td>0 volts</td>
<td>3 volts</td>
<td>0 volts</td>
<td>3 volts</td>
</tr>
</tbody>
</table>

2 to 1 Boolean Functions

There are 16 possible two bit input one bit output

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>16 possible functions (F0–F15)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 1 0 0 0 0 1 1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 0 0 0 0 1 1 0 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 1 1 1 0 1 0 1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>X and Y</th>
<th>X or Y</th>
<th>X xor Y</th>
<th>X = Y</th>
<th>not X</th>
<th>not (X or Y)</th>
<th>X nor Y</th>
<th>not (X and Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Y</td>
<td>X or Y</td>
<td>X = Y</td>
<td>not X</td>
<td>not (X or Y)</td>
<td>X nor Y</td>
<td>not (X and Y)</td>
</tr>
<tr>
<td>1</td>
<td>X and Y</td>
<td>X or Y</td>
<td>X xor Y</td>
<td>X = Y</td>
<td>not X</td>
<td>not (X and Y)</td>
<td>X nor Y</td>
<td>not (X and Y)</td>
</tr>
</tbody>
</table>

| 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

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Boolean Functions and Algebra**

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

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**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:

- $X' = X \text{ NAND } X$
- $X \text{ AND } Y = (X \text{ NAND } Y)'$
- $X \text{ OR } Y = (X' \text{ NAND } Y')$

But we can express AND, OR, and NOT in terms of NOR:

- $X' = X \text{ NOR } X$
- $X \text{ OR } Y = (X' \text{ NOR } Y')$
- $X \text{ AND } Y = (X' \text{ NOR } Y')$

**Duality**

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

Example: $X + 0 = 0$ is dual to $X \cdot 1 = 1$

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

**Axioms and Theorems**

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

13. Consensus: \( (X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = X \cdot Y + X' \cdot Z \)
Dual: \( (X + Y) \cdot (Y + Z) \cdot (X' + Z) = (X + Y) \cdot (X' + Z) \)

14. DeMorgan’s Law: \( (X + Y + ...)' = X' \cdot Y' \cdot ... \)
Dual: \( (X \cdot Y \cdot ...) = X' + Y' + ... \)

15. Generalized DeMorgan’s Laws: \( f(X_1,X_2,...,X_n,0,1,+,\cdot) = f(X_1',X_2',...,X_n',1,0,\cdot,+)^\)

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

16. Generalized Duality:
\[ f(X_1,X_2,...,X_n,0,1,+,\cdot) \iff f(X_1',X_2',...,X_n',1,0,\cdot,+)^\]

Proving Theorems

- Example 1: Prove the unifying theorem-- \( X \cdot Y + X \cdot Y' = X \)
  Distributive \( X \cdot Y + X \cdot Y' = (X \cdot Y + Y' \cdot Y) \)
  Complementarity \( = X \cdot 1 \)
  Identity \( = X \)

- Example 2: Prove the absorption theorem-- \( X + X \cdot Y = X \)
  Identity \( X + X \cdot Y = (X \cdot 1) + (X \cdot Y) \)
  Distributive \( = X \cdot (1 + Y) \)
  Null \( = X \cdot (1) \)
  Identity \( = X \)

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = A'B'Cin + A'BCin' + AB'Cin' + ABCin \]

\[ Cout = A'BCin + AB'Cin + ABCin' + ABCin \]

Simplification of Carry Out

\[ Cout = A'BCin + AB'Cin + ABCin' + ABCin \]
\[ = A'BCin + A'B'cin + ABCin' + ABCin + ABCin \]
\[ = (A' + A)BCin + AB'Cin + ABCin' + ABCin \]

associative
\[ = (1)BCin + AB'Cin + ABCin' + ABCin \]
\[ = BCin + AB'Cin + ABCin + ABCin + ABCin \]
\[ = BCin + A(B' + B)Cin + ABCin' + ABCin \]
\[ = BCin + A(1)Cin + ABCin' + ABCin \]
\[ = BCin + A Cin + AB(Cin' + Cin) \]
\[ = BCin + ACin + AB(1) \]
\[ = BCin + ACin + AB \]