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.

**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

Two Input Networks

- What is the relationship between x and y?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2 to 1 Boolean Functions

- There are 16 possible two bit input one bit output functions.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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 \( \oplus \) Y (F6): require 16 switches

\[ \text{NOTs, NANDs, NORs cost the least} \]

### NOT, NOR, NANDS, Oh My!
- Can we implement all logic functions from NOT, NOR, NANDs?
- Example: Implementing \( \text{NOT}(X \text{ NAND } Y) \) is the same as implementing \( (X \text{ AND } Y) \)
- In fact we can implement a NOT using a NAND or a NOR:
  \[ \text{NOT}(X) = X \text{ NAND } X \quad \text{NOT}(X) = Y \text{ NOR } Y \]
- In fact NAND and NOR can be used to implement each other:
  \[ X \text{ NAND } Y = \overline{\text{NOT}(X) \text{ OR } \overline{\text{NOT}(Y))}} \]
  \[ X \text{ NOR } Y = \overline{\text{NOT}(X) \text{ NAND } \overline{\text{NOT}(Y))}} \]
- 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 two binary operations, addition, \( + \), and multiplication, \( \cdot \), which satisfy the axioms:
  - \( B \) contains at least two nonequal elements
  - (closure) For every \( a, b \in B \):
    \[ a + b \in B \quad a \cdot b \in B \]
  - (commutative) For every \( a, b \in B \):
    \[ a + b = b + a \quad a \cdot b = b \cdot a \]
  - (associative) For every \( a, b, c \in B \):
    \[ (a + b) + c = a + (b + c) \quad a \cdot (b \cdot c) = (a \cdot b) \cdot c \]
  - (identity) There exists identity elements for + and \( \cdot \), such that for every \( a \in B \):
    \[ a + 0 = a \quad a \cdot 1 = a \]
  - (distributive) For every \( a, b, c \in B \):
    \[ a + (b \cdot c) = (a + b) \cdot (a + c) \quad a \cdot (b + c) = (a \cdot b) + (a \cdot c) \]
  - (complement) For each \( a \in B \) there exists an element \( a' \) in \( B \), such that \( a + a' = 1 \) and \( a \cdot a' = 0 \)

### A Boolean Algebra
- A Boolean Algebra:
  - \( B = \{0, 1\} \)
  - binary operation \( + = \text{logical OR} \)
  - binary operation \( \cdot = \text{logical AND} \)
  - complement \( ' = \text{logical NOT} \)

- These satisfy the above axioms

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

  Example: \( (X + Y) \cdot (X + Z) \) is the same as \( (X \text{ OR } Y) \text{ AND } (X \text{ OR } Z) \)
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>X1</th>
<th>X2</th>
<th>X3</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>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>0</td>
<td>1</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>

Boolean Functions and Algebra

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

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>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>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth Table Example:

F = X \cdot Y + Y \cdot \overline{Z}

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

Proving Theorems

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

Distributive \(X \cdot Y + X \cdot Y' = X \cdot (Y + Y')\)
Complementarity \(Y + Y' = 1\)
Identity \(X \cdot 1 = X\)

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 \(X\)

Example 3: Prove the consensus theorem-- \((X \cdot Y) + (Y \cdot Z) + (X' \cdot Z) = X \cdot Y + X' \cdot Z\)

Activity

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 \[= XY+X'YZ+X'Z\]

管控 absorption \{AB+A=A\} with A=XY and B=Z
\[= XY+X'YZ+X'Z\]

Rearrange terms \[= XY+X'ZY+X'Z\]
管控 absorption \{AB+A=A\} with A=X'Z and B=Y
\[= XY+X'ZY+X'Z\]

Proving Theorems

Prove by using “Perfect Induction” also called “Enumeration”

Cumbersome for very large expressions

\[X \land Y' = X' \lor Y'\]

NOR is equivalent to AND

\[X \lor Y' = X' \land Y'\]

NAND is equivalent to OR

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X'</th>
<th>Y'</th>
<th>(X + Y')'</th>
<th>X' • Y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Logic Simplification

Use the axioms to simplify logical expressions

- Why? To use less hardware

Example: A two-level logic expression
\[Z = \overline{A'BC} + \overline{AB'C} + \overline{AB'C} + ABC' + ABC\]
\[= \overline{AB'C} + \overline{AB'C} + \overline{AB'C} + ABC' + ABC\] rearrange
\[= \overline{AB'C} + \overline{AB'C} + \overline{AB'C} + ABC + AB(C' + C)\] distributive
\[= \overline{AB'} + A'BC + AB\] comp.
\[= \overline{AB'} + AB + A'BC\] 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

<table>
<thead>
<tr>
<th>Cin</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A'B</td>
<td>B'C</td>
<td>B + C</td>
<td>A'B'C + A'B'C' + A'B'C' + ABC' + ABC</td>
</tr>
</tbody>
</table>

Boolean expression
- \[S = A'B'Cin + A'B'C' + A'B'C' + ABC' + ABC\]
- \[Cout = A'B'Cin + AB'Cin + ABCin' + ABCin\]
Simplification of Carry Out

\[ \text{Cout} = A'B\text{Cin} + AB'C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = A'B\text{Cin} + AB'C\text{in} + ABC\text{in'} + ABC\text{in} + ABC\text{in} \]
\[ = (A'+A)BC\text{in} + AB'C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = (1)BC\text{in} + AB'C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = BC\text{in} + AB'C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = BC\text{in} + A(B+B)C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = BC\text{in} + A(1)C\text{in} + ABC\text{in'} + ABC\text{in} \]
\[ = BC\text{in} + AC\text{in} + AB(\text{Cin'+Cin}) \]
\[ = BC\text{in} + AC\text{in} + AB(1) \]
\[ = BC\text{in} + AC\text{in} + AB \]