Gates and Logic:
From Transistors to Logic Gates and Logic Circuits

Prof. Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

[Weatherspoon, Bala, Bracy, and Sirer]
Goals for Today

• From Switches to Logic Gates to Logic Circuits
• Logic Gates
  • From switches
  • Truth Tables
• Logic Circuits
  • From Truth Tables to Circuits (Sum of Products)
  • Identity Laws
• Logic Circuit Minimization
  • Algebraic Manipulations
  • Truth Tables (Karnaugh Maps)
• Transistors (electronic switch)
A switch
Acts as a *conductor* or *insulator*.
Can be used to build amazing things…

The Bombe used to break the German Enigma machine during World War II
Basic Building Blocks: Switches to Logic Gates

Truth Table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>

Truth Table

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>
Basic Building Blocks: Switches to Logic Gates

- Either (OR)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>

- Both (AND)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>
Basic Building Blocks: Switches to Logic Gates

- Either (OR)

Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ON</td>
</tr>
</tbody>
</table>

0 = OFF
1 = ON

- Both (AND)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Light</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ON</td>
</tr>
</tbody>
</table>
Basic Building Blocks: Switches to Logic Gates

- **Did you know?**
  - **George Boole**: Inventor of the idea of logic gates. He was born in Lincoln, England and he was the son of a shoemaker in a low class family.

George Boole (1815-1864)
Takeaway

• Binary (two symbols: true and false) is the basis of Logic Design
Building Functions: Logic Gates

- **NOT:**
  - digital circuit that either allows a signal to pass through it or not.
  - Used to build logic functions
  - There are seven basic logic gates: **AND, OR, NOT, NAND (not AND), NOR (not OR), XOR, and XNOR (not XOR)** [later]
  - **Truth Table for NOT:**
    | A | Out |
    |---|-----|
    | 0 | 1   |
    | 1 | 0   |
  - **Truth Table for AND:**
    | A | B | Out |
    |---|---|-----|
    | 0 | 0 | 0   |
    | 0 | 1 | 0   |
    | 1 | 0 | 0   |
    | 1 | 1 | 1   |
  - **Truth Table for NAND:**
    | A | B | Out |
    |---|---|-----|
    | 0 | 0 | 1   |
    | 0 | 1 | 1   |
    | 1 | 0 | 1   |
    | 1 | 1 | 0   |
  - **Truth Table for OR:**
    | A | B | Out |
    |---|---|-----|
    | 0 | 0 | 0   |
    | 0 | 1 | 1   |
    | 1 | 0 | 1   |
    | 1 | 1 | 1   |
  - **Truth Table for NOR:**
    | A | B | Out |
    |---|---|-----|
    | 0 | 0 | 1   |
    | 0 | 1 | 0   |
    | 1 | 0 | 0   |
    | 1 | 1 | 0   |
Goals for Today

• From Switches to Logic Gates to Logic Circuits
• Logic Gates
  • From switches
  • Truth Tables
• Logic Circuits
  • From Truth Tables to Circuits (Sum of Products)
  • Identity Laws
• Logic Circuit Minimization
  • Algebraic Manipulations
  • Truth Tables (Karnaugh Maps)
• Transistors (electronic switch)
Next Goal

• Given a Logic function, create a Logic Circuit that implements the Logic Function…
• …and, with the minimum number of logic gates
• Fewer gates: A cheaper ($$$$) circuit!
Logic Gates

NOT:

\[
\begin{array}{c|c}
A & \text{Out} \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

AND:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

NAND:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

OR:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

NOR:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
\end{array}
\]

XOR:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

XNOR:

\[
\begin{array}{c|c|c}
A & B & \text{Out} \\
\hline
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

A B Out
0 0 1
0 1 0
1 0 0
1 1 1
Logic Implementation

- How to implement a desired logic function?

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</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>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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Logic Implementation

• How to implement a desired logic function?

1) Write minterms
2) sum of products:
   • OR of all minterms where out=1

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</th>
<th>minterm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( \bar{a} \bar{b} \bar{c} )</td>
</tr>
</tbody>
</table>
Logic Equations

• NOT:
  - out = \overline{a} = !a = \neg a

• AND:
  - out = a \cdot b = a \& b = a \land b

• OR:
  - out = a + b = a | b = a \lor b

• XOR:
  - out = a \oplus b = a\overline{b} + \overline{a}b

• NAND:
  - out = \overline{a \cdot b} = !(a \& b) = \neg (a \land b)

• NOR:
  - out = \overline{a + b} = !(a \lor b) = \neg (a \lor b)

• XNOR:
  - out = \overline{a \oplus b} = ab + \overline{ab}

• Logic Equations
  - Constants: true = 1, false = 0
  - Variables: a, b, out, ...
  - Operators (above): AND, OR, NOT, etc.
Identities
Identities useful for manipulating logic equations
  - For optimization & ease of implementation

\[
a + 0 = a
\]
\[
a + 1 = a
\]
\[
a + \bar{a} = 1
\]

\[
a \cdot 0 = 0
\]
\[
a \cdot 1 = a
\]
\[
a \cdot \bar{a} = 0
\]
Identities

Identities useful for manipulating logic equations

- For optimization & ease of implementation

\[(a + b) = \]

\[(a \cdot b) = \]

\[a + a \cdot b = \]

\[a(b+c) = \]

\[a(b + c) = \]
Goals for Today

• From Switches to Logic Gates to Logic Circuits
• Logic Gates
  • From switches
  • Truth Tables
• Logic Circuits
  • From Truth Tables to Circuits (Sum of Products)
  • Identity Laws
• **Logic Circuit Minimization – why?**
  • Algebraic Manipulations
  • Truth Tables (Karnaugh Maps)
• Transistors (electronic switch)
Checking Equality w/Truth Tables

circuits ↔ truth tables ↔ equations

Example: \((a+b)(a+c) = a + bc\)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Takeaway

• Binary (two symbols: true and false) is the basis of Logic Design

• More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.
Goals for Today

• From Switches to Logic Gates to Logic Circuits
• Logic Gates
  • From switches
  • Truth Tables
• Logic Circuits
  • From Truth Tables to Circuits (Sum of Products)
  • Identity Laws
• Logic Circuit Minimization
  • Algebraic Manipulations
    • Truth Tables (Karnaugh Maps)
  • Transistors (electronic switch)
Karnaugh Maps
How does one find the most efficient equation?
– Manipulate algebraically until…?
– Use Karnaugh Maps (optimize visually)
– Use a software optimizer

For large circuits
– Decomposition & reuse of building blocks
Minimization with Karnaugh maps (1)

Sum of minterms yields

out = \( abc + \overline{a}bc + a\overline{b}c + a\overline{b}\overline{c} \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>out</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>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>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Minimization with Karnaugh maps (2)

- **Sum of minterms yields**
  - \( \text{out} = \overline{a} \overline{b} c + \overline{a} b c + a \overline{b} c + a \overline{b} \overline{c} \)

- **Karnaugh map minimization**
  - Cover all 1’s
  - Group adjacent blocks of \( 2^n \) 1’s that yield a rectangular shape
  - Encode the common features of the rectangle
  - \( \text{out} = a \overline{b} + \overline{a} c \)
Karnaugh Minimization Tricks (1)

- Minterms can overlap
  - $\text{out} =$

- Minterms can span 2, 4, 8 or more cells
  - $\text{out} =$
Karnaugh Minimization Tricks (2)

• The map wraps around
  - \( \text{out} = \)

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cd</th>
<th>ab</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
“Don’t care” values can be interpreted individually in whatever way is convenient

- Assume all x’s = 1
- Out =

- Assume middle x’s = 0
- Assume 4th column x = 1
- Out =
Minimization with K-Maps

(1) Circle the 1’s (see below)
(2) Each circle is a logical component of the final equation

\[ = a\overline{b} + \overline{a}c \]

Rules:
- Use fewest circles necessary to cover all 1’s
- Circles must cover only 1’s
- Circles span rectangles of size power of 2 (1, 2, 4, 8…)
- Circles should be as large as possible (all circles of 1?)
- Circles may wrap around edges of K-Map
- 1 may be circled multiple times if that means fewer circles
Multiplexer

- A multiplexer selects between multiple inputs
  - out = a, if d = 0
  - out = b, if d = 1

- Build truth table
- Minimize diagram
- Derive logic diagram
Takeaway

• Binary (two symbols: true and false) is the basis of Logic Design

• More than one Logic Circuit can implement same Logic function. Use Algebra (Identities) or Truth Tables to show equivalence.

• Any logic function can be implemented as “sum of products”. Karnaugh Maps minimize number of gates.
Goals for Today

- From Switches to Logic Gates to Logic Circuits
- Logic Gates
  - From switches
  - Truth Tables
- Logic Circuits
  - From Truth Tables to Circuits (Sum of Products)
  - Identity Laws
- Logic Circuit Minimization
  - Algebraic Manipulations
  - Truth Tables (Karnaugh Maps)
- Transistors (electronic switch)
Silicon Valley & the Semiconductor Industry

• Transistors:
• Youtube video “How does a transistor work”
  https://www.youtube.com/watch?v=lcrBqCFLHIY
• Break: show some Transistor, Fab, Wafer photos
Transistors 101

**N-Type Silicon:** negative free-carriers (electrons)

**P-Type Silicon:** positive free-carriers (holes)

**P-Transistor:** negative charge on gate generates electric field that creates a (+ charged) p-channel connecting source & drain

**N-Transistor:** works the opposite way

Metal-Oxide Semiconductor (Gate-Insulator-Silicon)

- Complementary MOS = **CMOS** technology uses both p- & n-type transistors
CMOS Notation

N-type

Gate input controls whether current can flow between the other two terminals or not.

P-type

*Hint:* the “o” bubble of the p-type tells you that this gate wants a 0 to be turned on.
2-Transistor Combination: NOT

- Logic gates are constructed by combining transistors in complementary arrangements.
- Combine p&n transistors to make a NOT gate:

**CMOS Inverter:**

- **power source (1)**
- **input**
- **p-gate**
- **output**
- **n-gate**
- **ground (0)**

- **power source (1)**
- **n-gate stays open**
- **p-gate closes**
- **ground (0)**

- **power source (1)**
- **p-gate stays open**
- **n-gate closes**
- **ground (0)**
In Inverter

Function: NOT

Symbol:

Truth Table:

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
NOR Gate

Function: NOR
Symbol:

Truth Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Building Functions (Revisited)

- **NOT:**
  - ![Not gate diagram]

- **AND:**
  - ![And gate diagram]

- **OR:**
  - ![Or gate diagram]

NAND and NOR are universal
- Can implement *any* function with NAND or just NOR gates
- Useful for manufacturing
Logic Gates

- One can buy gates separately
  - ex. 74xxx series of integrated circuits
  - cost ~$1 per chip, mostly for packaging and testing

- Cumbersome, but possible to build devices using gates put together manually
Then and Now

**The first transistor**
- One workbench at AT&T Bell Labs
- 1947
- Bardeen, Brattain, and Shockley

**Intel Haswell**
- 1.4 billion transistors, 22nm
- 177 square millimeters
- Four processing cores

https://en.wikipedia.org/wiki/Transistor_count
Then and Now

- **The first transistor**
  - One workbench at AT&T Bell Labs
  - 1947
  - Bardeen, Brattain, and Shockley

- **Intel Broadwell**
  - 7.2 billion transistors, 14nm
  - 456 square millimeters
  - Up to 22 processing cores

https://en.wikipedia.org/wiki/Transistor_count
Big Picture: Abstraction

• Hide complexity through simple abstractions
  ▪ Simplicity
    • Box diagram represents inputs and outputs
  ▪ Complexity
    • Hides underlying NMOS- and PMOS-transistors and atomic interactions
Summary

• Most modern devices made of billions of transistors
  • You will build a processor in this course!
  • Modern transistors made from semiconductor materials
  • Transistors used to make logic gates and logic circuits

• We can now implement any logic circuit
  • Use P- & N-transistors to implement NAND/NOR gates
  • Use NAND or NOR gates to implement the logic circuit
  • Efficiently: use K-maps to find required minimal terms