6. Combinational Circuits

Introduction to Computer Science   •   Robert Sedgewick and Kevin Wayne   •   Copyright © 2005   •   http://www.cs.Princeton.EDU/IntroCS

George Boole (1815 – 1864)  Claude Shannon (1916 – 2001)

Digital Circuits

What is a digital system?
- Digital: signals are 0 or 1.
- Analog: signals vary continuously.

Why digital systems?
- Accuracy and reliability.
- Staggeringly fast and cheap.

Basic abstractions.
- On, off.
- Wire: propagates on/off value.
- Switch: controls propagation of on/off values through wires.

Digital circuits and you.
- Computer microprocessors.
- Antilock brakes, cell phones, iPods, etc.

Computer Architecture

TOY lectures. von Neumann machine.

This lecture. Boolean circuits.

Ahead. Putting it all together and building a TOY machine.

Digital Circuits

Wires.
- On (1): connected to power.
- Off (0): not connected to power.
- If a wire is connected to a wire that is on, that wire is also on.
- Typical drawing convention: “flow” from top, left to bottom, right.
Controlled Switch

Controlled switch. [relay implementation]
- 3 connections: input, output, control.
- Magnetic force pulls on a contact that cuts electrical flow.
- Control wire affects output wire, but output does not affect control; establishes forward flow of information over time.

Layers of Abstraction

Layers of abstraction.
- Build a circuit from wires and switches.
- Define a circuit by its inputs and outputs.
- To control complexity, encapsulate circuits.
Logic Gates: Fundamental Building Blocks

\[ \text{NOT} = x' \]

\begin{array}{c|c|c}
 x & \text{NOT} & x' \\
 0 & 1 & \\
 1 & 0 & \\
\end{array}

\[ \text{OR} = x \lor y \]

\begin{array}{c|c|c|c}
 x & y & \text{OR} & x \lor y \\
 0 & 0 & 0 & \\
 0 & 1 & 1 & \\
 1 & 0 & 1 & \\
 1 & 1 & 1 & \\
\end{array}

\[ \text{AND} = x \land y \]

\begin{array}{c|c|c|c}
 x & y & \text{AND} & x \land y \\
 0 & 0 & 0 & \\
 0 & 1 & 0 & \\
 1 & 0 & 0 & \\
 1 & 1 & 1 & \\
\end{array}

Multiway Gates

- \text{OR}: \ 1 \text{ if any input is 1}; \ 0 \text{ otherwise}.
- \text{AND}: \ 1 \text{ if all inputs are 1}; \ 0 \text{ otherwise}.
- \text{Generalized}: \ negate \ some \ inputs.
Boolean Algebra

History.
- Developed by Boole to solve mathematical logic problems (1847).
- Shannon master’s thesis applied it to digital circuits (1937).

Basics.
- Boolean variable: value is 0 or 1.
- Boolean function: function whose inputs and outputs are 0, 1.

Relationship to circuits.
- Boolean variables: signals.
- Boolean functions: circuits.

Truth Table

Truth table.
- Systematic method to describe Boolean function.
- One row for each possible input combination.
- \( N \) inputs \( \rightarrow 2^N \) rows.

Truth Table for Functions of 2 Variables

Truth table.
- 16 Boolean functions of 2 variables.
- every 4-bit value represents one
Truth Table for Functions of 3 Variables

Truth table.
- 16 Boolean functions of 2 variables.
- 256 Boolean functions of 3 variables.
- $2^n$ Boolean functions of $n$ variables!

Sum-of-Products
Systematic procedure for representing a Boolean function using AND, OR, NOT.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>AND</th>
<th>OR</th>
<th>MAJ</th>
<th>ODD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<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>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Some Functions of 3 Variables

Universality of AND, OR, NOT

Fact. Any Boolean function can be expressed using AND, OR, NOT.

- $\{\text{AND}, \text{OR}, \text{NOT}\}$ are universal.
- Ex: $\text{XOR}(x, y) = xy' + x'y$.

Expressing XOR Using AND, OR, NOT

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x</th>
<th>y'</th>
<th>xy'</th>
<th>xy</th>
<th>xy' + x'y'</th>
<th>XOR y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Exercise. Show $\{\text{AND}, \text{NOT}\}$, $(\text{OR}, \text{NOT})$, $(\text{NAND})$, $(\text{NOR})$ are universal.

Hint. DeMorgan’s law: $(x'y')' = x + y$.

Sum-of-Products

Translate Boolean Formula to Boolean Circuit

$\text{XOR} = xy' + x'y$
Translate Boolean Formula to Boolean Circuit

**Sum-of-products. XOR.**

\[ \text{XOR} = x'y + xy' \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>XOR</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>

Truth table

Abstract circuit

Circuit

Translate Boolean Formula to Boolean Circuit

**Sum-of-products. Majority.**

\[ \text{MAJ} = x'y'z + xy'z + xyz' + xyz \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>MAJ</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>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>

Truth table

Circuit
Translate Boolean Formula to Boolean Circuit

**Sum-of-products. Majority.**

\[ \text{MAJ} = x'y'z + x'y'z + x'y'z + xyz \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>MAJ</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>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  Abstract circuit  Circuit

Simplification Using Boolean Algebra

**Many possible circuits for each Boolean function.**

- Sum-of-products not necessarily optimal in:
  - number of switches (space)
  - depth of circuit (time)

**Ex.** MAJ(x, y, z) = x'y'z + x'y'z + x'y'z + xyz = xy + yz + xz.

Expressing a Boolean Function Using AND, OR, NOT

**Ingredients.**
- AND gates.
- OR gates.
- NOT gates.
- Wire.

**Instructions.**
- Step 1: represent input and output signals with Boolean variables.
- Step 2: construct truth table to carry out computation.
- Step 3: derive (simplified) Boolean expression using sum-of-products.
- Step 4: transform Boolean expression into circuit.

ODD Parity Circuit

**ODD(x, y, z).**
- 1 if odd number of inputs are 1.
- 0 otherwise.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>ODD</th>
<th>x'y'z</th>
<th>x'y'z</th>
<th>xy'z'</th>
<th>xyz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Expressing ODD using sum-of-products
ODD Parity Circuit

ODD(x, y, z).
- 1 if odd number of inputs are 1.
- 0 otherwise.

Let's Make an Adder Circuit

Goal. \( x + y = z \) for 4-bit integers.
- We build 4-bit adder: 9 inputs, 4 outputs.
- Same idea scales to 128-bit adder.
- Key computer component.

Step 1. Represent input and output in binary.

Step 2. [first attempt]
- Build truth table.

Q. Why is this a bad idea?
A. 128-bit adder: \( 2^{512-1} \) rows >> # electrons in universe!
Let's Make an Adder Circuit

**Goal.** \( x + y = z \) for 4-bit integers.

**Step 2.** [do one bit at a time]
- Build truth table for carry bit.
- Build truth table for summand bit.

<table>
<thead>
<tr>
<th>Carry Bit</th>
<th>Summand Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i, y_i, c_i, c_{i+1} )</td>
<td>( x_i, y_i, c_i, z_i )</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
</tr>
</tbody>
</table>

**Step 3.**
- Derive (simplified) Boolean expression.

<table>
<thead>
<tr>
<th>Carry Bit</th>
<th>Summand Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i, y_i, c_i, c_{i+1}, MAJ )</td>
<td>( x_i, y_i, c_i, z_i, ODD )</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>0 0 1 1 1</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>0 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 1 1</td>
<td>0 1 1 0 0</td>
</tr>
<tr>
<td>1 0 0 0 0</td>
<td>1 0 0 1 1</td>
</tr>
<tr>
<td>1 0 1 1 1</td>
<td>1 0 1 0 0</td>
</tr>
<tr>
<td>1 1 0 1 1</td>
<td>1 1 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>1 1 1 1 1</td>
</tr>
</tbody>
</table>

**Step 4.**
- Transform Boolean expression into circuit.
- Chain together 1-bit adders.

**Adder: Interface**
Adder: Component Level View

Adder: Switch Level View

Shifter

Decoder

Decoder: [n-bit]
- n address inputs, 2^n data outputs.
- Addressed output bit is 1; others are 0 register.
2-Bit Decoder Controlling 4-Bit Shifter

**Ex.** Put in a binary amount to shift.

![Diagram of 2-bit decoder controlling 4-bit shifter]

Arithmetic Logic Unit

Arithmetic logic unit (ALU). Computes all operations in parallel.
- Add and subtract.
- Xor.
- And.
- Shift left or right.

Q. How to select desired answer?

1 Hot OR

1 hot OR.
- All devices compute their answer; we pick one.
- Exactly one select line is on.
- Implies exactly one output line is relevant.

![Diagram of 1-hot OR]

Arithmetic logic unit.
- Add and subtract.
- Xor.
- And.
- Shift left or right.

Arithmetic logic unit.
- Computes all operations in parallel.
- Uses 1-hot OR to pick each bit answer.
**Device Interface Using Buses**

**Device.** Processes a word at a time.  → 16-bit words for TOY memory

**Input bus.** Wires on top.

**Output bus.** Wires on bottom.

**Control.** Individual wires on side.

---

**Summary**

**Lessons for software design apply to hardware design!**

- Interface describes behavior of circuit.
- Implementation gives details of how to build it.

**Layers of abstraction apply with a vengeance!**

- On/off.
- **Controlled switch.** [relay, transistor]
- **Gates.** [AND, OR, NOT]
- **Boolean circuit.** [MAJ, ODD]
- Adder.
- Shifter.
- Arithmetic logic unit.
- ...
- **TOY machine.**