19. Combinational Circuits
19. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder circuit
- Arithmetic/logic unit
Q. What is a combinational circuit?
A. A digital circuit (all signals are 0 or 1) with no feedback (no loops).

Q. Why combinational circuits?
A. Accurate, reliable, general purpose, fast, cheap.

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

Applications. Smartphone, tablet, game controller, antilock brakes, *microprocessor*, ...
Wires

Wires propagate on/off values

- ON (1): connected to power.
- OFF (0): not connected to power.
- Any wire connected to a wire that is ON is also ON.
- Drawing convention: "flow" from top, left to bottom, right.

power connection

thick wires are ON

thin wires are OFF
Switches control propagation of on/off values through wires.

- Simplest case involves two connections: control (input) and output.
- control OFF: output ON
- control ON: output OFF
Controlled Switch

Switches control propagation of on/off values through wires.
- General case involves three connections: control input, data input and output.
- control OFF: output is connected to input
- control ON: output is disconnected from input

Idealized model of pass transistors found in real integrated circuits.
A *relay* is a physical device that controls a switch with a magnet

- 3 connections: input, output, control.
- Magnetic force pulls on a contact that cuts electrical flow.
First level of abstraction

Switches and wires model provides separation between physical world and logical world.
  • We assume that switches operate as specified.
  • That is the only assumption.
  • Physical realization of switch is irrelevant to design.

Physical realization dictates *performance*
  • Size.
  • Speed.
  • Power.

New technology *immediately* gives new computer.


Basis of Moore's law.
### Switches and wires: a first level of abstraction

<table>
<thead>
<tr>
<th>technology</th>
<th>“information”</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>pneumatic</td>
<td>air pressure</td>
<td><img src="pneumatic.png" alt="Image" /></td>
</tr>
<tr>
<td>fluid</td>
<td>water pressure</td>
<td><img src="fluid.png" alt="Image" /></td>
</tr>
<tr>
<td>relay (now)</td>
<td>electric potential</td>
<td><img src="relay.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Amusing attempts that do not scale but prove the point

<table>
<thead>
<tr>
<th>technology</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>relay (1940s)</td>
<td><img src="relay_1940s.png" alt="Image" /></td>
</tr>
<tr>
<td>vacuum tube</td>
<td><img src="vacuum_tube.png" alt="Image" /></td>
</tr>
<tr>
<td>transistor</td>
<td><img src="transistor.png" alt="Image" /></td>
</tr>
<tr>
<td>“pass transistor” in integrated circuit</td>
<td><img src="pass_transistor.png" alt="Image" /></td>
</tr>
<tr>
<td>atom-thick transistor</td>
<td><img src="atom_thick_transistor.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Real-world examples that prove the point
Switches and wires: a first level of abstraction

VLSI = Very Large Scale Integration

Technology
Deposit materials on substrate.

Key properties
- Lines are wires.
- Certain crossing lines are controlled switches.

Key challenge in physical world
Fabricating physical circuits with billions of wires and controlled switches

Key challenge in “abstract” world
Understanding behavior of circuits with billions of wires and controlled switches

Bottom line. Circuit = Drawing (!)
Circuit anatomy

connected wires

crossing wires

switch

Need more levels of abstraction to understand circuit behavior
Image sources

http://electronics.howstuffworks.com/relay.htm
19. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder circuit
- Arithmetic/logic unit
Boolean algebra

Developed by George Boole in 1840s to study logic problems
- Variables represent true or false (1 or 0 for short).
- Basic operations are AND, OR, and NOT (see table below).

Widely used in mathematics, logic and computer science.

<table>
<thead>
<tr>
<th>operation</th>
<th>Java notation</th>
<th>logic notation</th>
<th>circuit design (this lecture)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>x &amp;&amp; y</td>
<td>( x \land y )</td>
<td>( xy )</td>
</tr>
<tr>
<td>OR</td>
<td>x</td>
<td></td>
<td>y</td>
</tr>
<tr>
<td>NOT</td>
<td>! x</td>
<td>( \neg x )</td>
<td>( x' )</td>
</tr>
</tbody>
</table>

Example: (stay tuned for proof)

\[
(xy)' = (x' + y') \\
(x + y)' = x'y'
\]

DeMorgan’s Laws

Relevance to circuits. Basis for next level of abstraction.

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com
A truth table is a systematic way to define a Boolean function
- One row for each possible set of arguments.
- Each row gives the function value for the specified arguments.
- $N$ inputs: $2^N$ rows needed.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x'$</th>
<th>$x$</th>
<th>$y$</th>
<th>$xy$</th>
<th>$x$</th>
<th>$y$</th>
<th>$x + y$</th>
<th>$x$</th>
<th>$y$</th>
<th>$NOR$</th>
<th>$x$</th>
<th>$y$</th>
<th>$XOR$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth table proofs

Truth tables are convenient for establishing identities in Boolean logic
- One row for each possibility.
- Identity established if columns match.

**Proofs of DeMorgan's laws**

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

\[(xy)' = (x' + y')\]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x + y</th>
<th>(x + y)'</th>
<th>x</th>
<th>y</th>
<th>x'</th>
<th>y'</th>
<th>x'y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[(x + y)' = x'y'\]
All Boolean functions of two variables

Q. How many Boolean functions of two variables?
A. 16 (all possibilities for the 4 bits in the truth table column).

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>ZERO</th>
<th>AND</th>
<th>x</th>
<th>y</th>
<th>XOR</th>
<th>OR</th>
<th>NOR</th>
<th>EQ</th>
<th>¬y</th>
<th>¬x</th>
<th>NAND</th>
<th>ONE</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>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Functions of three and more variables

Q. How many Boolean functions of *three* variables?

A. 256 (all possibilities for the 8 bits in the truth table column).

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>AND</th>
<th>OR</th>
<th>NOR</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>1</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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<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>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Some Boolean functions of 3 variables

Examples

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>logical AND</td>
<td>0 iff <em>any</em> inputs is 0 (1 iff all inputs 1)</td>
</tr>
<tr>
<td>OR</td>
<td>logical OR</td>
<td>1 iff <em>any</em> input is 1 (0 iff all inputs 0)</td>
</tr>
<tr>
<td>NOR</td>
<td>logical NOR</td>
<td>0 iff <em>any</em> input is 1 (1 iff all inputs 0)</td>
</tr>
<tr>
<td>MAJ</td>
<td>majority</td>
<td>1 iff more inputs are 1 than 0</td>
</tr>
<tr>
<td>ODD</td>
<td>odd parity</td>
<td>1 iff an odd number of inputs are 1</td>
</tr>
</tbody>
</table>

Q. How many Boolean functions of *N* variables?

A. $2^{(2^N)}$

<table>
<thead>
<tr>
<th>N</th>
<th>number of Boolean functions with N variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$2^4 = 16$</td>
</tr>
<tr>
<td>3</td>
<td>$2^8 = 256$</td>
</tr>
<tr>
<td>4</td>
<td>$2^{16} = 65,536$</td>
</tr>
<tr>
<td>5</td>
<td>$2^{32} = 4,294,967,296$</td>
</tr>
<tr>
<td>6</td>
<td>$2^{64} = 18,446,744,073,709,551,616$</td>
</tr>
</tbody>
</table>
Universality of AND, OR and NOT

Every Boolean function can be represented as a **sum of products**
- Form an AND term for each 1 in Boolean function.
- OR all the terms together.

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

Expressing MAJ as a sum of products:

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

**Def.** A set of operations is *universal* if every Boolean function can be expressed using just those operations.

**Fact.** \{ AND, OR, NOT \} is universal.
Image sources
http://en.wikipedia.org/wiki/George_Boole#/media/File:George_Boole_color.jpg
19. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder circuit
- Arithmetic/logic unit
A basis for digital devices

Claude Shannon connected circuit design with Boolean algebra in 1937.

“Possibly the most important, and also the most famous, master’s thesis of the [20th] century.”

– Howard Gardner

Key idea. Can use Boolean algebra to systematically analyze circuit behavior.

A Symbolic Analysis of Relay and Switching Circuits

By CLAIRE SHANNON

I. Introduction

In this analysis, we will examine the behavior of relay and switching circuits, which are fundamental components in the design of digital devices. By representing these circuits using Boolean algebra, we can systematically analyze their behavior and predict their performance.

II. Basic Concepts

A. Variables

In Boolean algebra, variables represent the states of the switches in a circuit. These states can be either on (1) or off (0).

B. Operations

1. OR Operation

   \[ A + B = C \]

2. AND Operation

   \[ A \cdot B = C \]

3. NOT Operation

   \[ \neg A = B \]

C. lpfuncal Equations

In the following section, we will derive the symbolic equations that describe the behavior of relay and switching circuits.

1. Steady State

   \[ A_0 = \neg B_0 \]

2. Stable State

   \[ A_s = B_s \]

3. Transition State

   \[ A_t = \neg B_t \]

4. Symmetric State

   \[ A_y = B_y \]

D. Circuit Analysis

By applying the symbolic equations to the circuit, we can determine the behavior of the circuit at any given time.

E. Conclusion

In this paper, we have presented a systematic approach to analyzing relay and switching circuits. By using Boolean algebra, we can predict the behavior of these circuits and design more efficient and reliable digital devices.

Claude Shannon 1916–2001
### A second level of abstraction: logic gates

<table>
<thead>
<tr>
<th>boolean function</th>
<th>notation</th>
<th>truth table</th>
<th>classic symbol</th>
<th>our symbol</th>
<th>under the cover circuit (gate)</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOT</strong></td>
<td>$x'$</td>
<td><img src="image" alt="Truth Table" /></td>
<td><img src="image" alt="NOR Symbol" /></td>
<td><img src="image" alt="NOR Symbol" /></td>
<td><img src="image" alt="NOR Gate" /></td>
<td>1 iff $x$ is 0</td>
</tr>
<tr>
<td><strong>NOR</strong></td>
<td>$(x + y)'$</td>
<td><img src="image" alt="Truth Table" /></td>
<td><img src="image" alt="NOR Symbol" /></td>
<td><img src="image" alt="NOR Symbol" /></td>
<td><img src="image" alt="NOR Gate" /></td>
<td>1 iff $x$ and $y$ are both 0</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>$x + y$</td>
<td><img src="image" alt="Truth Table" /></td>
<td><img src="image" alt="NOT Symbol" /></td>
<td><img src="image" alt="NOR Symbol" /></td>
<td><img src="image" alt="NOR Gate" /></td>
<td>$x + y = ((x + y)')'$</td>
</tr>
<tr>
<td><strong>AND</strong></td>
<td>$xy$</td>
<td><img src="image" alt="Truth Table" /></td>
<td><img src="image" alt="AND Symbol" /></td>
<td><img src="image" alt="AND Symbol" /></td>
<td><img src="image" alt="AND Gate" /></td>
<td>$xy = (x' + y')'$</td>
</tr>
</tbody>
</table>
Multiway OR gates

OR gates with multiple inputs.
- 1 if any input is 1.
- 0 if all inputs are 0.

Multiway OR gates are oriented vertically in our circuits. Learn to recognize them!
Multiway generalized AND gates.

- 1 for exactly 1 set of input values.
- 0 for all other sets of input values.

**Gate** | **Function** | **Inputs that output 1** | **Another set of inputs**
--- | --- | --- | ---
AND | $uvwxyz$ | 1 | 0

generalized | $u'vwx'y'z$ | 1 | 0

NOR | $u'v'w'x'y'z'$ | 1 | 0

same as $(u + v + w + x + y + z)'$

Might also call these "generalized NOR gates"; we consistently use AND.
Pop quiz on generalized AND gates

Q. Give the Boolean function computed by these gates.

Q. Also give the inputs for which the output is 1.
Pop quiz on generalized AND gates

Q. Give the Boolean function computed by these gates.

Q. Also give the inputs for which the output is 1.

\[ u'v'wxy'z \quad 101101 \]

\[ u'vwxy'z \quad 011001 \]

Get the idea? If not, replay this slide, like flash cards.

Note. From now on, we will not label these gates.
A useful combinational circuit: decoder

Decoder
- \( n \) input lines (address).
- \( 2^n \) outputs.
- Addressed output is 1.
- All other outputs are 0.

Example: 3-to-8 decoder

- 110 = 6
- Outputs 0-5 and 7 are 0
- Output 6 is 1
A useful combinational circuit: decoder

Decoder
- $n$ input lines (address).
- $2^n$ outputs.
- Addressed output is 1.
- All other outputs are 0.

Implementation
- Use all $2^n$ generalized AND gates with $n$ inputs.
- Only one of them matches the input address.

Application (next lecture)
- Select a memory word for read/write.
- [Use address bits of instruction from IR.]
Another useful combinational circuit: demultiplexer (demux)

Demultiplexer

- $n$ address inputs.
- 1 data input with value $x$.
- $2^n$ outputs.
- Addressed output has value $x$.
- All other outputs are 0.

Example: 3-to-8 demux

Example diagram showing how the demultiplexer works with an address input of 101, which corresponds to output 5. Output 5 has value $x$, and the other outputs 0-4 and 6-7 are 0.
Another useful combinational circuit: demultiplexer (demux)

**Demultiplexer**
- $n$ address inputs.
- 1 data input with value $x$.
- $2^n$ outputs.
- Addressed output has value $x$.
- All other outputs are 0.

**Implementation**
- Start with decoder.
- Add AND $x$ to each gate.

**Application (next lecture)**
- Turn on control wires to implement instructions.
- [Use opcode bits of instruction in IR.]

*Example: 3-to-8 demux*
Decoder/demux

- $n$ address inputs.
- 1 data input with value $x$.
- $2^n$ output pairs.
- Addressed output pair has value $(1, x)$.
- All other outputs are 0.

Example: 3-to-8 decoder/demux

101 = 5

output pairs 0-4 and 6-7 are (0, 0)
output pair 5 has value (1, x)
Decoder/demux

Decoder/demux
- \( n \) address inputs.
- 1 data input with value \( x \).
- \( 2^n \) output pairs.
- Addressed output pair has value \((1, x)\).
- All other outputs are 0.

Implementation
- Add decoder output to demux.

Example: 3-to-8 decoder/demux

Application (next lecture)
- Access and control write of memory word
- [Use addr bits of instruction in IR.]
Creating a digital circuit that computes a boolean function: majority

Use the truth table
- Identify rows where the function is 1.
- Use a generalized AND gate for each.
- OR the results together.

**Example 1: Majority function**

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

\[ MAJ = x'y'z + xy'z + xyz' + xyz \]
Creating a digital circuit that computes a boolean function: odd parity

**Use the truth table**
- Identify rows where the function is 1.
- Use a generalized AND gate for each.
- OR the results together.

**Example 2: Odd parity function**

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

\[ ODD = x'y'z + x'yz' + xy'z' + xyz \]

**odd parity circuit**

**Example**

110

**multiway OR gate**

**ODD is 0**
Combinational circuit design: Summary

**Problem:** Design a circuit that computes a given boolean function.

**Ingredients**
- OR gates.
- NOT gates.
- NOR gates.
- Wire.

**Method**
- Step 1: Represent input and output with Boolean variables.
- Step 2: Construct truth table to define the function.
- Step 3: Identify rows where the function is 1.
- Step 4: Use a generalized AND for each and OR the results.

**Bottom line (profound idea):** Yields a circuit for ANY function.

**Caveat:** Circuit might be huge (stay tuned).
Pop quiz on combinational circuit design

Q. Design a circuit to implement XOR(x, y).
**Pop quiz on combinational circuit design**

**Q.** Design a circuit to implement $XOR(x, y)$.

**A. Use the truth table**
- Identify rows where the function is 1.
- Use a generalized AND gate for each.
- OR the results together.

**XOR function**

<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>

$XOR = x'y + xy'$

![Circuit diagram](image)
Encapsulation in hardware design mirrors familiar principles in software design

- Building a circuit from wires and switches is the *implementation*.
- Define a circuit by its inputs, controls, and outputs is the *API*.
- We control complexity by *encapsulating* circuits as we do with *ADTs*.
Image sources

19. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder circuit
- Arithmetic/logic unit
Let's make an adder circuit!

Adder
- Compute $z = x + y$ for $n$-bit binary integers.
- $2n$ inputs.
- $n$ outputs.
- Ignore overflow.

Example: 8-bit adder

$$
\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
+ & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
$$

$0 + 49 = 72$
Let's make an adder circuit!

**Adder**
- Compute $z = x + y$ for $n$-bit binary integers.
- 2$n$ inputs.
- $n$ outputs.
- Ignore overflow.

**Example: 8-bit adder**

<table>
<thead>
<tr>
<th>$C_8$</th>
<th>$C_7$</th>
<th>$C_6$</th>
<th>$C_5$</th>
<th>$C_4$</th>
<th>$C_3$</th>
<th>$C_2$</th>
<th>$C_1$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_7$</td>
<td>$x_6$</td>
<td>$x_5$</td>
<td>$x_4$</td>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>$y_7$</td>
<td>$y_6$</td>
<td>$y_5$</td>
<td>$y_4$</td>
<td>$y_3$</td>
<td>$y_2$</td>
<td>$y_1$</td>
<td>$y_0$</td>
</tr>
<tr>
<td>$z_7$</td>
<td>$z_6$</td>
<td>$z_5$</td>
<td>$z_4$</td>
<td>$z_3$</td>
<td>$z_2$</td>
<td>$z_1$</td>
<td>$z_0$</td>
<td></td>
</tr>
</tbody>
</table>
Let's make an adder circuit!

**Goal**: \( z = x + y \) for 8-bit integers.

**Strawman solution**: Build truth tables for each output bit.

### 8-bit adder truth table

<table>
<thead>
<tr>
<th>( x_7 )</th>
<th>( x_6 )</th>
<th>( x_5 )</th>
<th>( x_4 )</th>
<th>( x_3 )</th>
<th>( x_2 )</th>
<th>( x_1 )</th>
<th>( x_0 )</th>
<th>( y_7 )</th>
<th>( y_6 )</th>
<th>( y_5 )</th>
<th>( y_4 )</th>
<th>( y_3 )</th>
<th>( y_2 )</th>
<th>( y_1 )</th>
<th>( y_0 )</th>
<th>( z_7 )</th>
<th>( z_6 )</th>
<th>( z_5 )</th>
<th>( z_4 )</th>
<th>( z_3 )</th>
<th>( z_2 )</th>
<th>( z_1 )</th>
<th>( z_0 )</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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<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>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>
<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>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<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>

\( 2^{16} = 65536 \) rows!

**Q.** Not convinced this a bad idea?

**A.** 128-bit adder: \( 2^{256} \) rows >> # electrons in universe!
Let's make an adder circuit!

**Goal:** $z = x + y$ for 8-bit integers.

**Do one bit at a time.**
- Build truth table for carry bit.
- Build truth table for sum bit.

**A surprise!**
- Carry bit is MAJ.
- Sum bit is ODD.

### Carry bit

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$c_i$</th>
<th>$c_{i+1}$</th>
<th>MAJ</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>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>0</td>
<td>0</td>
</tr>
<tr>
<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>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Sum bit

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$c_i$</th>
<th>$z_i$</th>
<th>ODD</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</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>
</tr>
</tbody>
</table>

### Truth tables

<table>
<thead>
<tr>
<th>$c_8$</th>
<th>$c_7$</th>
<th>$c_6$</th>
<th>$c_5$</th>
<th>$c_4$</th>
<th>$c_3$</th>
<th>$c_2$</th>
<th>$c_1$</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_7$</td>
<td>$x_6$</td>
<td>$x_5$</td>
<td>$x_4$</td>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$y_7$</td>
<td>$y_6$</td>
<td>$y_5$</td>
<td>$y_4$</td>
<td>$y_3$</td>
<td>$y_2$</td>
<td>$y_1$</td>
<td>$y_0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$z_7$</td>
<td>$z_6$</td>
<td>$z_5$</td>
<td>$z_4$</td>
<td>$z_3$</td>
<td>$z_2$</td>
<td>$z_1$</td>
<td>$z_0$</td>
<td></td>
</tr>
</tbody>
</table>
Let's make an adder circuit!

Goal: $z = x + y$ for 4-bit integers.

Do one bit at a time.
- Carry bit is MAJ.
- Sum bit is ODD.
- Chain 1-bit adders to "ripple" carries.
An 8-bit adder circuit
Layers of abstraction

Lessons for software design apply to hardware
- Interface describes behavior of circuit.
- Implementation gives details of how to build it.
- Exploit understanding of behavior at each level.

Layers of abstraction apply with a vengeance
- On/off.
- Controlled switch. [relay, pass transistor]
- Gates. [NOT, OR, AND]
- Boolean functions. [MAJ, ODD]
- Adder.
- Arithmetic/Logic unit (next).
- CPU (next lecture, stay tuned).

Vastly simplifies design of complex systems and enables use of new technology at any layer
19. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder circuit
- Arithmetic/logic unit
Next layer of abstraction: modules, busses, and control lines

Basic design of our circuits
- Organized as *modules* (functional units of TOY: ALU, memory, register, PC, and IR).
- Connected by *busses* (groups of wires that propagate information between modules).
- Controlled by *control lines* (single wires that control circuit behavior).

Conventions
- Bus inputs are at the top, input connections are at the left.
- Bus outputs are at the bottom, output connections are at the right.
- Control lines are blue.

These conventions *make circuits easy to understand.*
(Like style conventions in coding.)
Ex. Three functions on 8-bit words
- Two input busses (arguments).
- One output bus (result).
- Three control lines.
Arithmetic and logic unit (ALU) module

Ex. Three functions on 8-bit words
- Two input busses (arguments).
- One output bus (result).
- Three control lines.
- Left-right shifter circuits omitted (see book for details).

Implementation
- One circuit for each function.
- Compute all values in parallel.

Q. How do we select desired output?
A. "One-hot muxes" (see next slide).

"Calculator" at the heart of your computer.
A simple and useful combinational circuit: one-hot multiplexer

One-hot multiplexer
- $m$ selection lines
- $m$ data inputs
- 1 output.
- At most one selection line is 1.
- Output has value of selected input.

This is a precondition unlike other circuits we consider
A simple and useful combinational circuit: one-hot multiplexer

One-hot multiplexer
- \( m \) selection lines
- \( m \) data inputs
- 1 output.
- At most one selection line is 1.
- Output has value of selected input.

Implementation
- AND corresponding selection and data inputs.
- OR all results (at most one is 1).

Applications
- Arithmetic-logic unit (previous slide).
- Main memory (next lecture).

Important to note. No direct connection from input to output.
Summary: Useful combinational circuit modules

Next: Registers, memory, connections, and control.
19. Combinational Circuits