19. Combinational Circuits

**Context**

**Q.** What is a combinational circuit?

**A.** A digital circuit (all signals are 0 or 1) with no feedback (no loops).

- analog circuit: signals vary continuously
- sequential circuit: loops allowed (stay tuned)

**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.

**Diagram:**

- Thick wires are ON
- Thin wires are OFF

1

0
**Controlled Switch**

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

- 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.

---

**Controlled switch: example implementation**

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

Amusing attempts that do not scale but prove the point

Real-world examples that prove the point

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 (I)

Circuit anatomy

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</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>x &amp; y</td>
<td>x &amp; y</td>
<td>xy</td>
</tr>
<tr>
<td>OR</td>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>NOT</td>
<td>!x</td>
<td>!x</td>
<td>x'</td>
</tr>
</tbody>
</table>

DeMorgan’s Laws

Example: (stay tuned for proof)

- \((xy)' = (x' + y')\)
- \((x + y)' = x'y'\)

Truth tables

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

NOR

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

AND

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

OR

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

XOR

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

- \((xy)' = (x' + y')\)
- \((x + y)' = x'y'\)
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).

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).

Examples

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

Some Boolean functions of 3 variables

Expressing MAJ as a sum of products

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

Fact. \{ AND, OR, NOT \} is universal.
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 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>NOT</td>
<td>$x'$</td>
<td></td>
<td>[1, 0]</td>
<td>[x', x]</td>
<td>$x \rightarrow x'$</td>
<td>1 iff $x$ is 0</td>
</tr>
<tr>
<td>NOR</td>
<td>$(x + y)'$</td>
<td>[0, 1]</td>
<td>[x', y']</td>
<td>$x \neg\lor y'$</td>
<td>$(x + y)'$</td>
<td>1 iff $x$ and $y$ are both 0</td>
</tr>
<tr>
<td>OR</td>
<td>$x + y$</td>
<td>[0, 1]</td>
<td>[x, y]</td>
<td>$x \lor y$</td>
<td>$x + y$</td>
<td>1 if any input is 1</td>
</tr>
<tr>
<td>AND</td>
<td>$xy$</td>
<td>[0, 1]</td>
<td>[x, y]</td>
<td>$x \land y$</td>
<td>$xy$</td>
<td>1 if all inputs are 0</td>
</tr>
</tbody>
</table>

### Multiway OR gates

**Multiway OR gates** with multiple inputs.

- 1 if **any** input is 1.
- 0 if **all** inputs are 0.

<table>
<thead>
<tr>
<th>classic symbol</th>
<th>our symbol</th>
<th>under the cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w \lor x \lor y \lor z$</td>
<td>$w \lor x \lor y \lor z$</td>
<td>[0, 1]</td>
</tr>
</tbody>
</table>

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

- For exactly 1 set of input values:
  - Inputs that output 1:
    - AND gate: \( u'vwx'yz' \)
    - NOR gate: \( u'v'w'x'y'z' \)

- For all other sets of input values:
  - Another set of inputs:
    - AND gate: \( 01100 \)
    - NOR gate: \( 01010 \)

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.

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

Outputs 0-5 and 7 are 0
Output 6 is 1

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.

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.

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.]

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.
Decoder/demux

Implementation
- Add decoder output to 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>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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

MAJ = x'y'z + xy'z' + 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>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>

ODD = x'y'z + xy'z' + xyz

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).

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>

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

Encapsulation

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.
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>( x )</th>
<th>( y )</th>
<th>( z )</th>
<th>( c )</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>

---

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

**A.** 128-bit adder: \( 2^{128} \) 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>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>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>
<td>0</td>
</tr>
</tbody>
</table>

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.)

Arithmetic and logic unit (ALU) module

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.

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