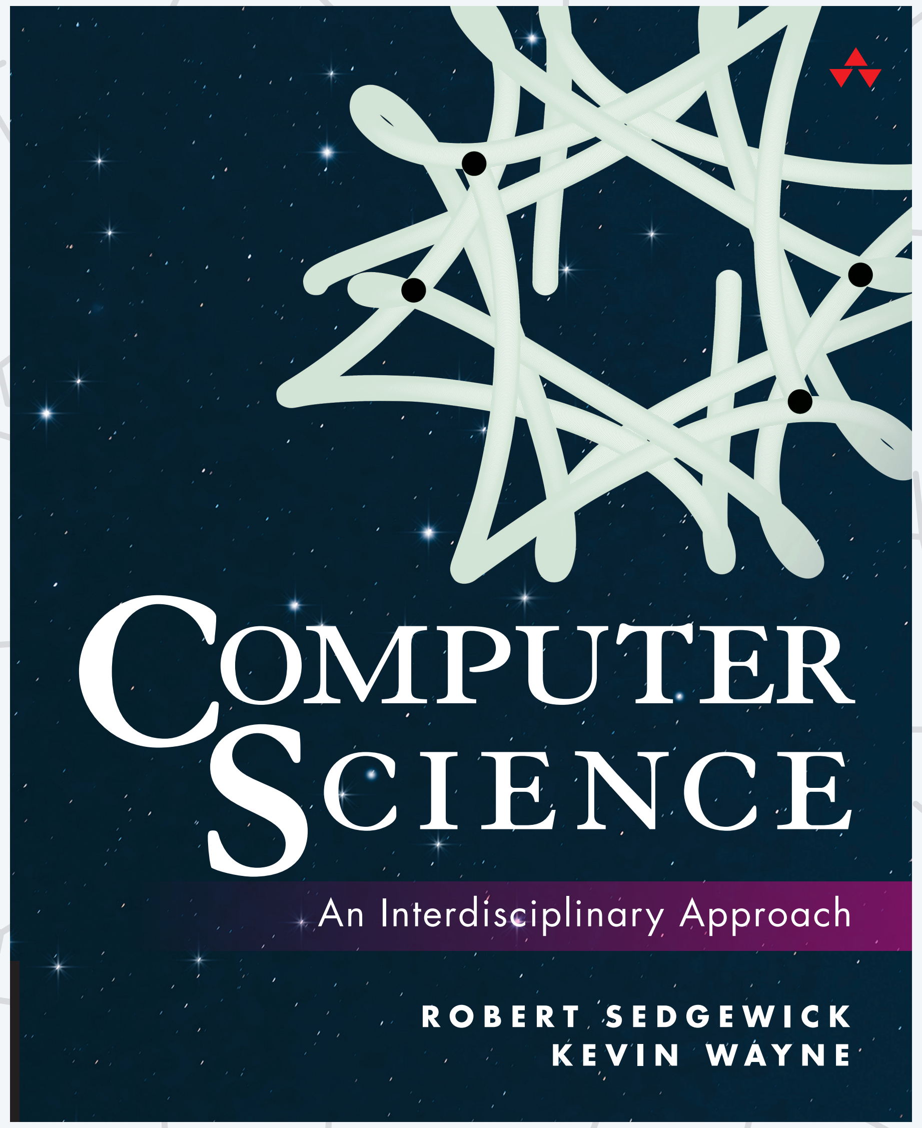



https://introcs.cs.princeton.edu

# 7. Digital Circuits

- boolean algebra
- logic gates
- sum-of-products
- adder circuit

# Context

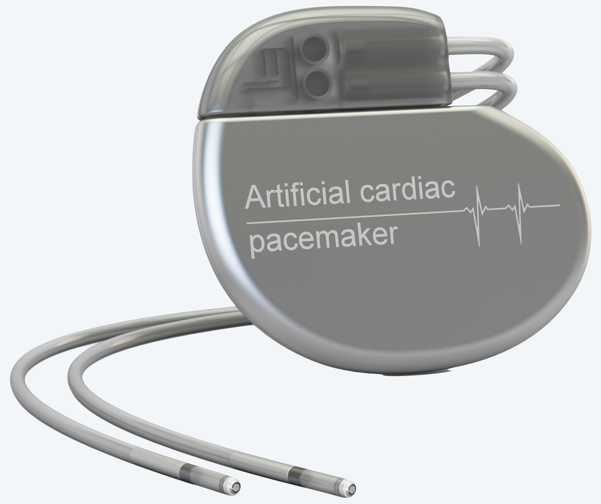
Q. How are computers built?

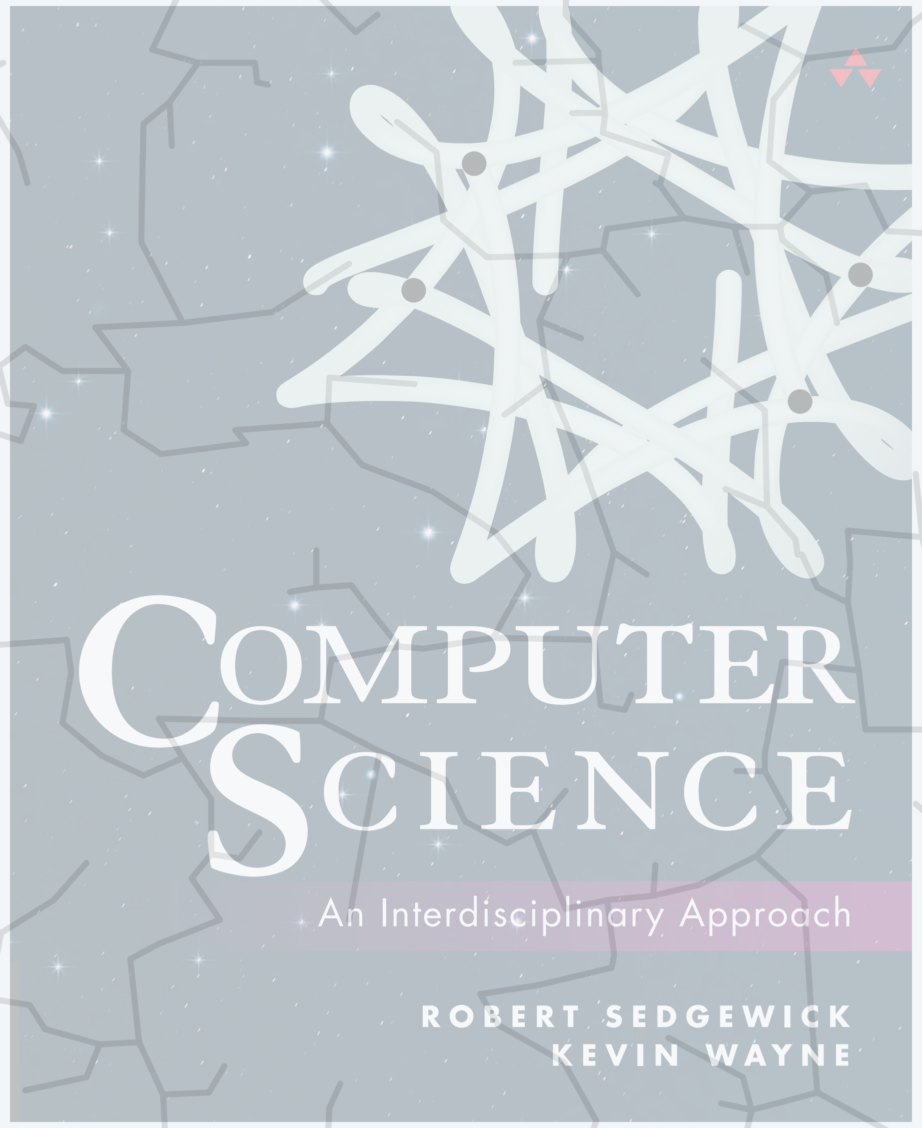
A. Not nearly as complicated as you might think.


This lecture. Introduction to digital circuits.

- Digital = all signals are either $0$ or $1$.
- Analog = signals vary continuously.
- Advantages of digital: accurate, reliable, fast, cheap, scalable, …

Applications. Laptop, smartphone, gaming console, pacemaker, …






https://introcs.cs.princeton.edu

# 7. DIGITAL CIRCUITS

- *boolean algebra*
- *logic gates*
- *sum-of-products*
- *adder circuit*

# Boolean algebra

Boolean algebra. Developed by George Boole in 1840s to study logic problems.

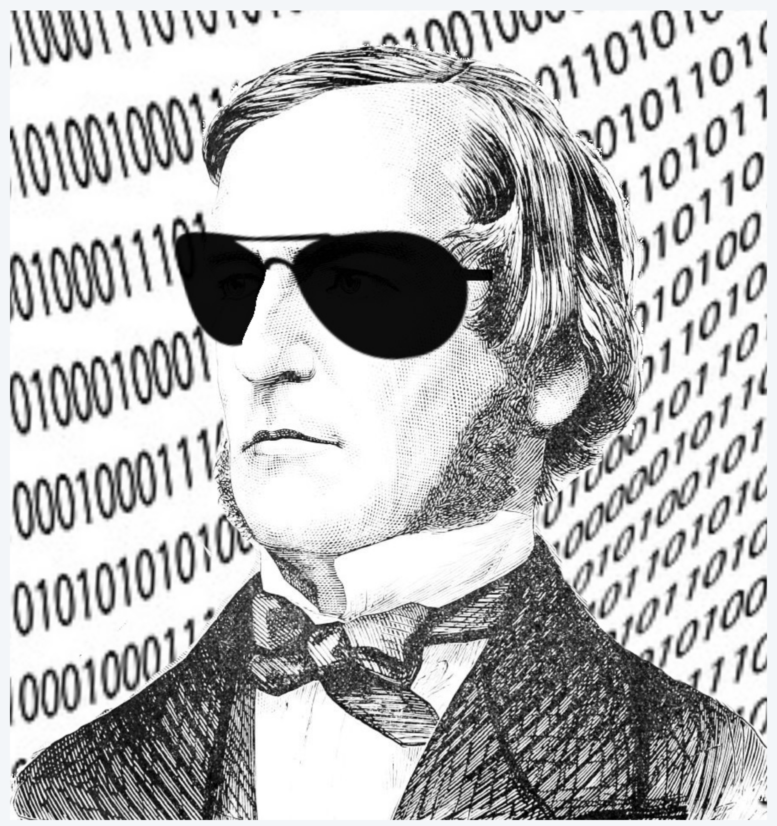
- Values of variables are *true* (1) or *false* (0).
- Primitive operations are *NOT*, *AND*, and *OR*.
- Widely used in mathematics, logic, computer science, …

| operation | logic notation | Java syntax | circuit notation | precedence |
|---|---|---|---|---|
| *NOT* | ¬ x | !x | $x'$ | *highest* |
| *AND* | x ∧ y | x && y | $x \cdot y$ | *middle* |
| *OR* | x ∨ y | x \|\| y | $x + y$ | *lowest* |

$\bar{x}$ (*alternative*) → $x'$

$x\ y$ (*shorthand*) → $x \cdot y$

*this lecture* → circuit notation



George Boole is Coole

Relevance to circuits. Provides the mathematical foundation.

# Truth tables

Boolean function. A function whose arguments and result assume the values 0 and 1.

Truth table. A systematic way to define a boolean function.

- One row for each possible assignment of arguments.
- Each row gives the function value for the specified arguments.
- The truth table of a boolean function of $n$ variables has $2^n$ rows.

| $x$ | $x'$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

NOT

| $x$ | $y$ | $x \cdot y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND

| $x$ | $y$ | $x + y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

*count in binary from 0 to $2^n - 1$*

# Boolean algebra properties

Boolean algebra shares many properties with elementary algebra. ← *justifies use of · and + for AND and OR*

| property | AND | OR |
|---|---|---|
| *commutative* | $x \cdot y = y \cdot x$ | $x + y = y + x$ |
| *associative* | $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ | $x + (y + z) = (x + y) + z$ |
| *identity* | $x \cdot 1 = x$ | $x + 0 = x$ |
| *distributive* | $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ | $x + (y \cdot z) = (x + y) \cdot (x + z)$ |
| *complementary* | $x \cdot x' = 0$ | $x + x' = 1$ |
| *idempotent* | $x \cdot x = x$ | $x + x = x$ |
| *De Morgan* | $(x \cdot y)' = x' + y'$ | $(x + y)' = x' \cdot y'$ |
| *duality* | *for any property, can interchange + and ·, along with 0 and 1* | |
| ⋮ | ⋮ | |

*same as elementary algebra* (commutative, associative, identity)

*different from elementary algebra* (distributive through duality)

# Proving a theorem in Boolean algebra

Q. How to prove a theorem, such as De Morgan's law?

A1. Apply sequence of known theorems.

A2. For each possible assignment of truth values to variables, evaluate the purported theorem; confirm that it is *true*. ← *"method of perfect induction"*

Ex. De Morgan's law: $(x \cdot y)' = (x' + y')$.

| $x$ | $y$ | $x \cdot y$ | $(x \cdot y)'$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**truth table for LHS**

| $x$ | $y$ | $x'$ | $y'$ | $x' + y'$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

**truth table for RHS**

# Boolean functions of two variables

Boolean function. A function whose arguments and result assume the values 0 and 1.

| $x$ | $y$ | *AND* | *OR* | *XOR* | *NAND* | *NOR* | *XNOR* |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**commonly used boolean functions of 2 variables**

# Boolean functions of three (and more) variables

**Boolean function.** A function whose arguments and result assume the values 0 and 1.

| $x$ | $y$ | $z$ | *AND* | *OR* | *MAJ* | *ODD* |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**commonly used boolean functions of 3 variables**

| function | shorthand | description |
|---|---|---|
| *logical AND* | *AND* | all inputs are 1 |
| *logical OR* | *OR* | at least one input is 1 |
| *majority* | *MAJ* | more inputs are 1 than 0 |
| *odd parity* | *ODD* | odd number of inputs are 1 |

*these functions extends to n variables*

**Which of the following does not represent majority function?**

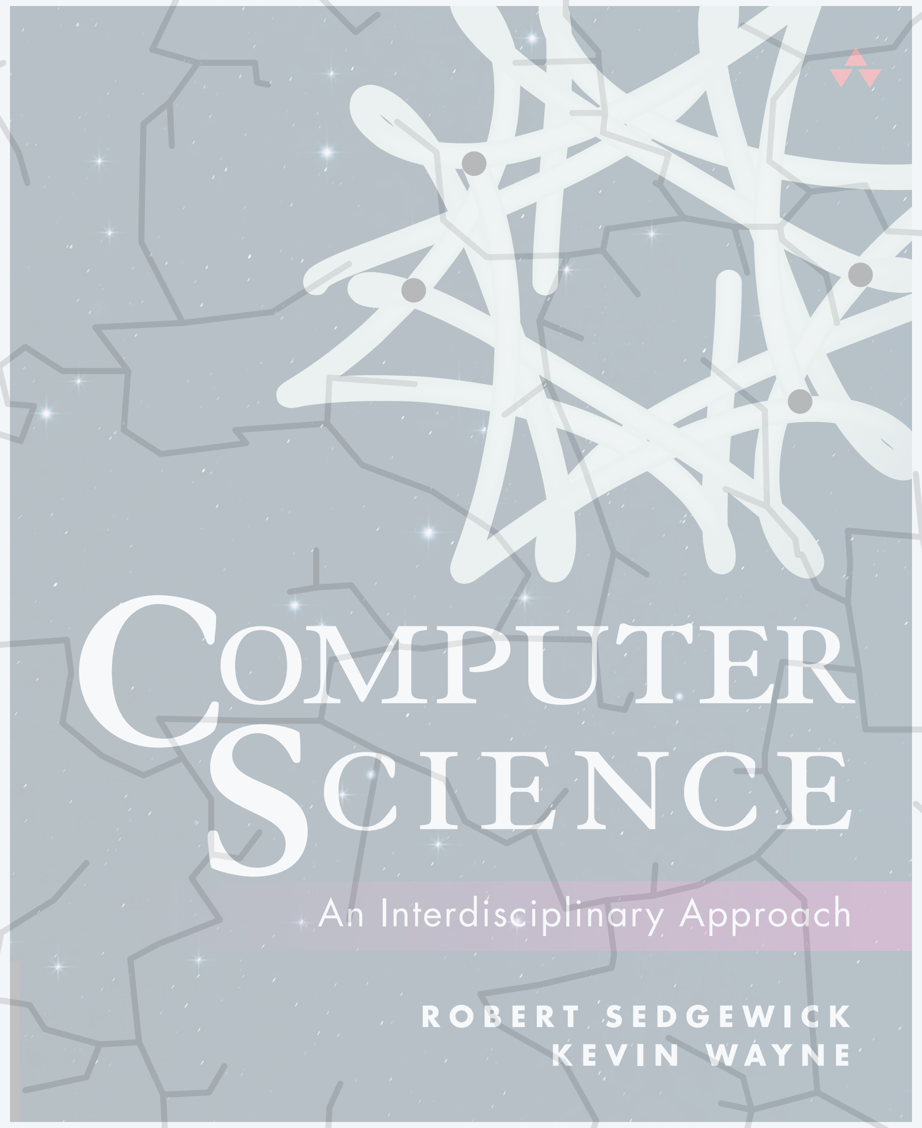
**A.** $(x \cdot y) + (y \cdot z)$

**B.** $(x \cdot y) + (y \cdot z) + (x \cdot z)$

**C.** $z \cdot (x' \cdot y + x \cdot y') + x \cdot y$

**D.**

```
public static boolean majority(boolean x, boolean y, boolean z) {
   int count = 0;
   if (x) count++;
   if (y) count++;
   if (z) count++;
   return count >= 2;
}
```




https://introcs.cs.princeton.edu

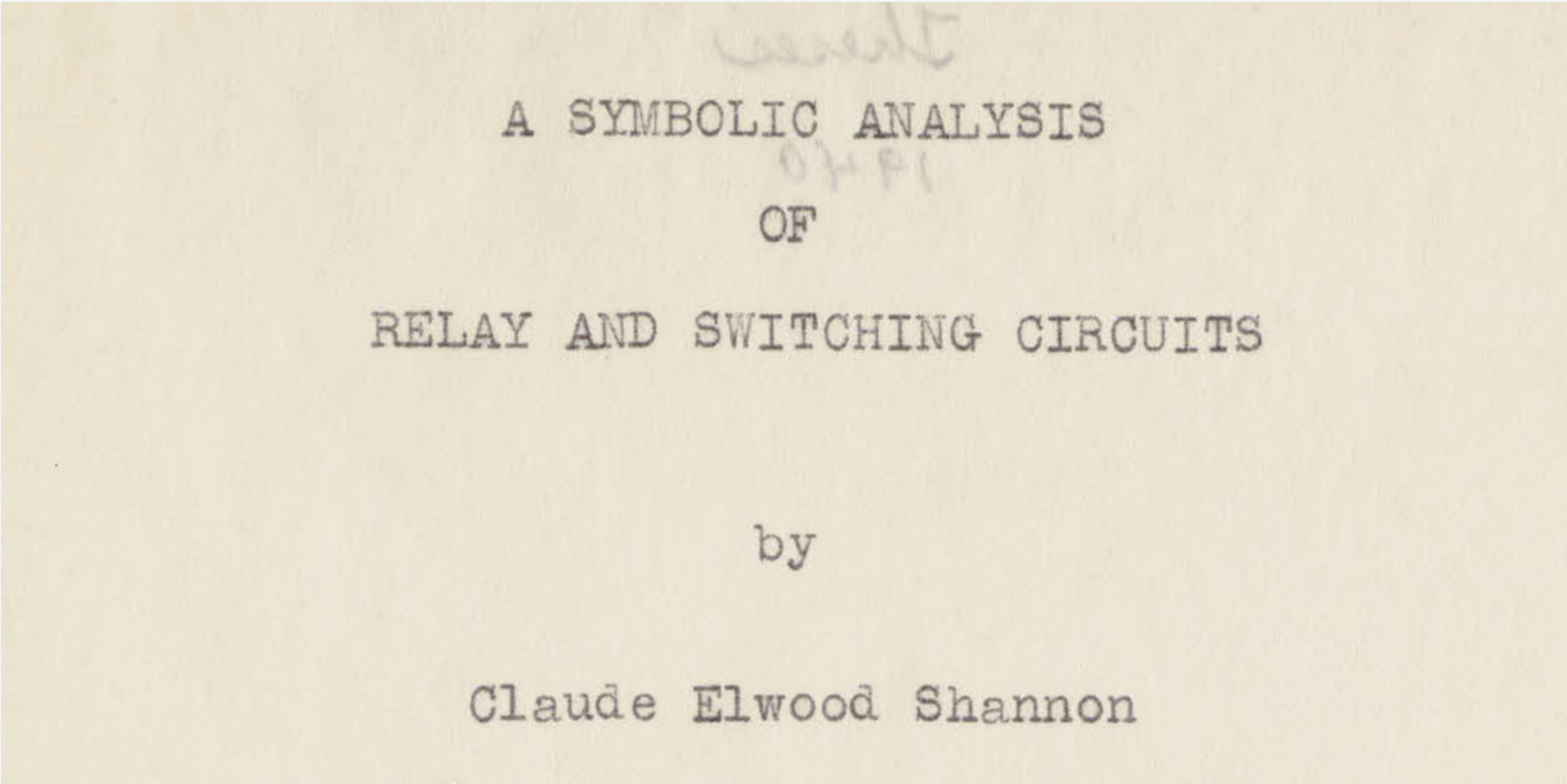
# 7. Digital Circuits

- boolean algebra
- logic gates
- sum-of-products
- adder circuit

# A basis for digital devices

Claude Shannon. Identified the deep connection between Boolean algebra and circuits.

- Demonstrated how circuits could be analyzed using Boolean algebra.
- Designed circuits to perform mathematical operations on binary numbers. ← *add, subtract, multiply, factor, …*

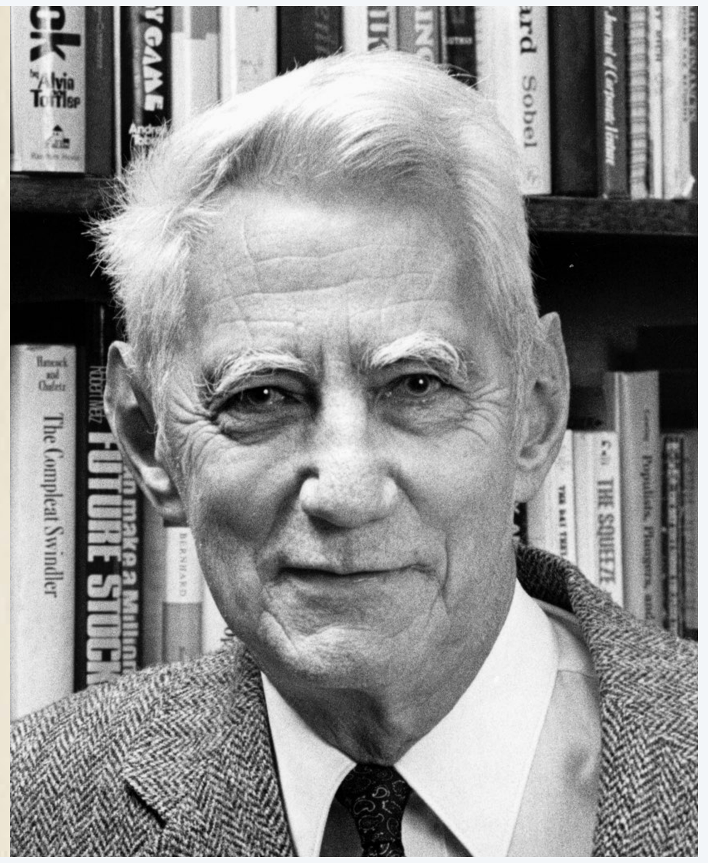

A SYMBOLIC ANALYSIS

OF

RELAY AND SWITCHING CIRCUITS

by

Claude Elwood Shannon

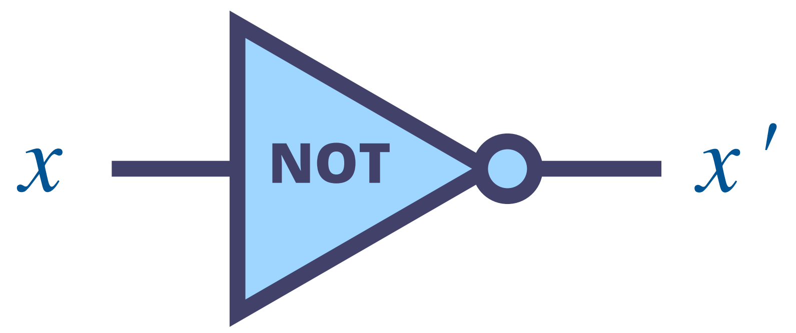
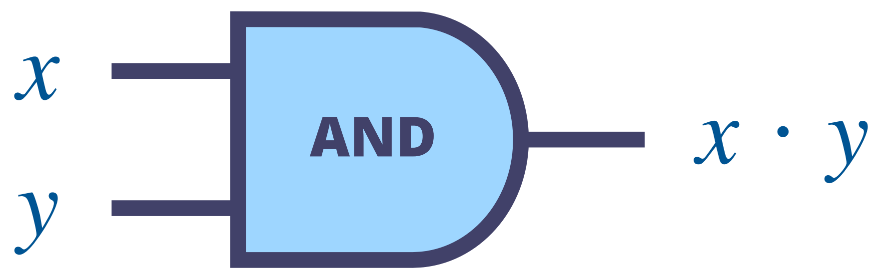
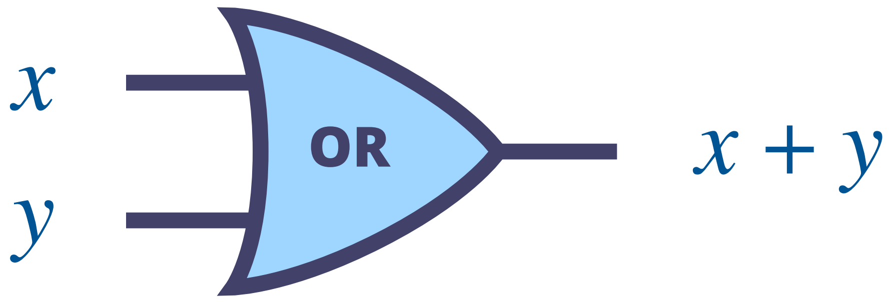


**Claude Shannon's master's thesis at MIT (1937)**

Impact. Every electronic device we use today is based upon Shannon's foundational work.

# Primitive logic gates: AND, OR, and NOT

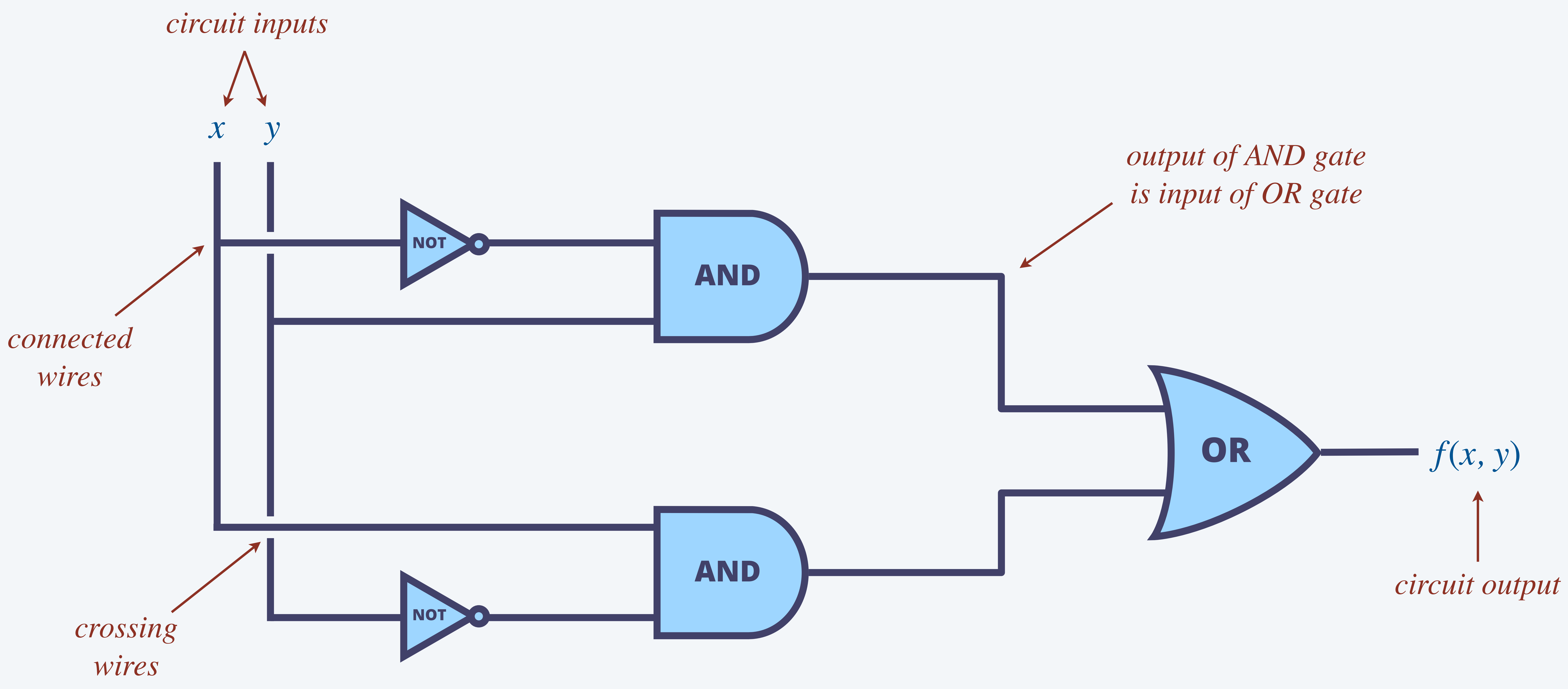
**Logic gate.** Physical device that implement a boolean function with one output.

<table>
<tr><th>gate</th><th>truth table</th><th>notation</th><th>symbol</th></tr>
<tr><td>NOT<br>(inverter)</td><td><table><tr><th>$x$</th><th>NOT</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td>$x'$</td><td></td></tr>
<tr><td>AND</td><td><table><tr><th>$x$</th><th>$y$</th><th>AND</th></tr><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></table></td><td>$x \cdot y$</td><td></td></tr>
<tr><td>OR</td><td><table><tr><th>$x$</th><th>$y$</th><th>OR</th></tr><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></table></td><td>$x + y$</td><td></td></tr>
</table>

# Digital circuits

Digital circuit. A network of logic gates connected by wires.

- Every wire is either *on* (1) or *off* (0).
- Can connect output of one gate to input of another gate.
- Any wire connected to a wire that is *on* is also *on* (and same for *off*).

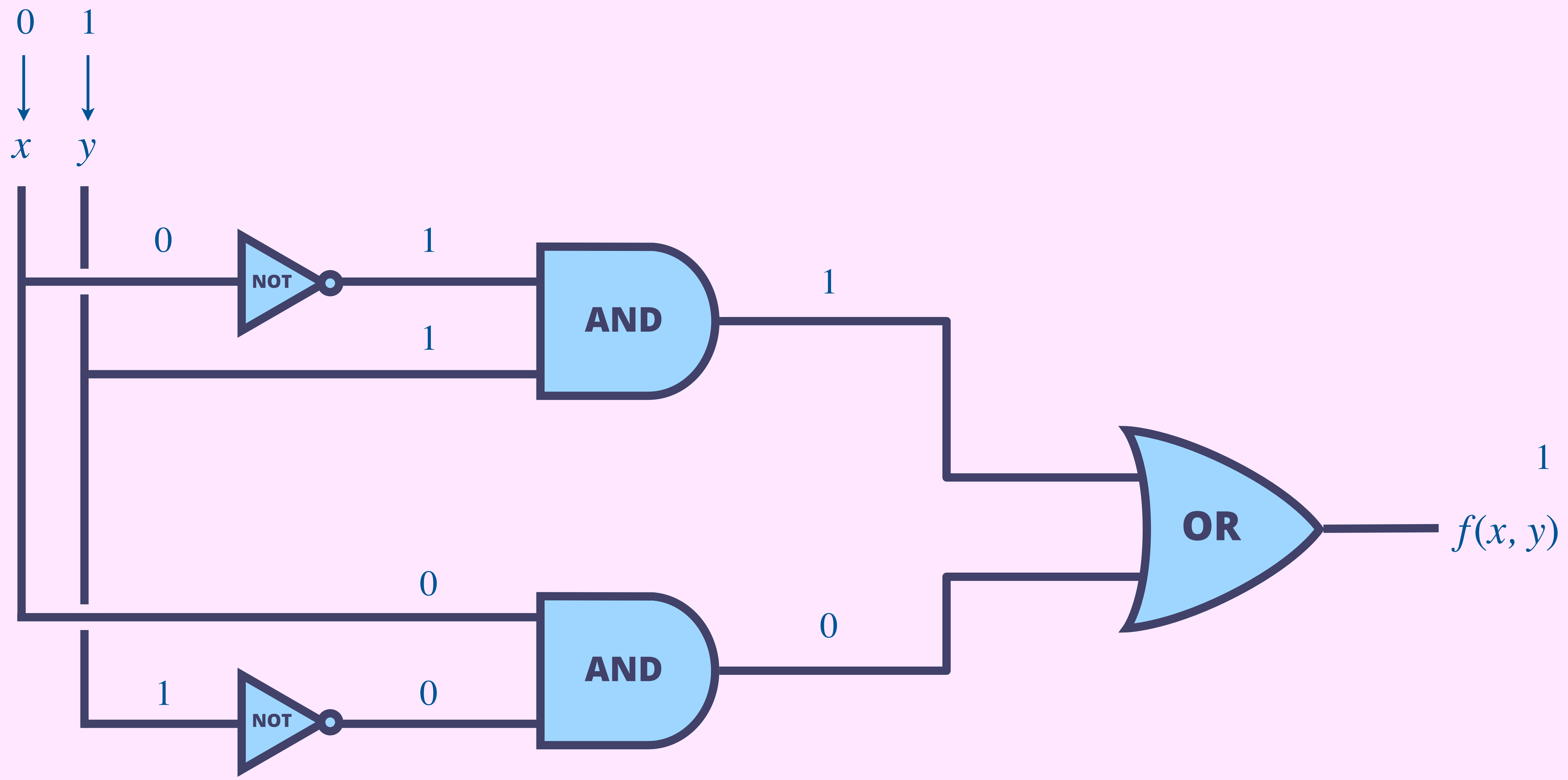



| $x$ | $y$ | $XOR$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Digital circuits

Digital circuit. A network of logic gates connected by wires.

- Every wire is either *on* (1) or *off* (0).
- Can connect output of one gate to input of another gate.
- Any wire connected to a wire that is *on* is also *on* (and same for *off*).




| $x$ | $y$ | $XOR$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**For which values of $x$ and $y$ does the following circuit output $1$ ?**

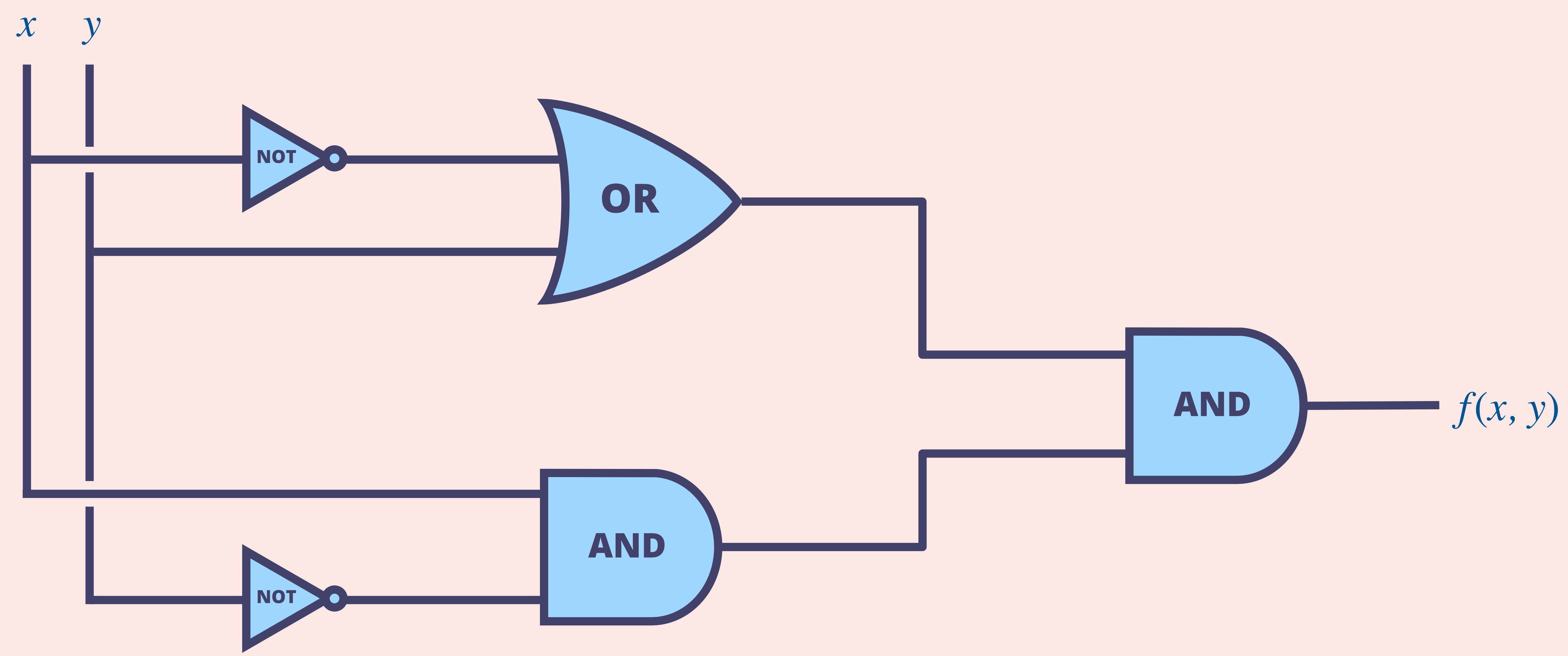
**A.** $x = 0,\ y = 0$

**B.** $x = 0,\ y = 1$

**C.** $x = 1,\ y = 0$

**D.** $x = 1,\ y = 1$

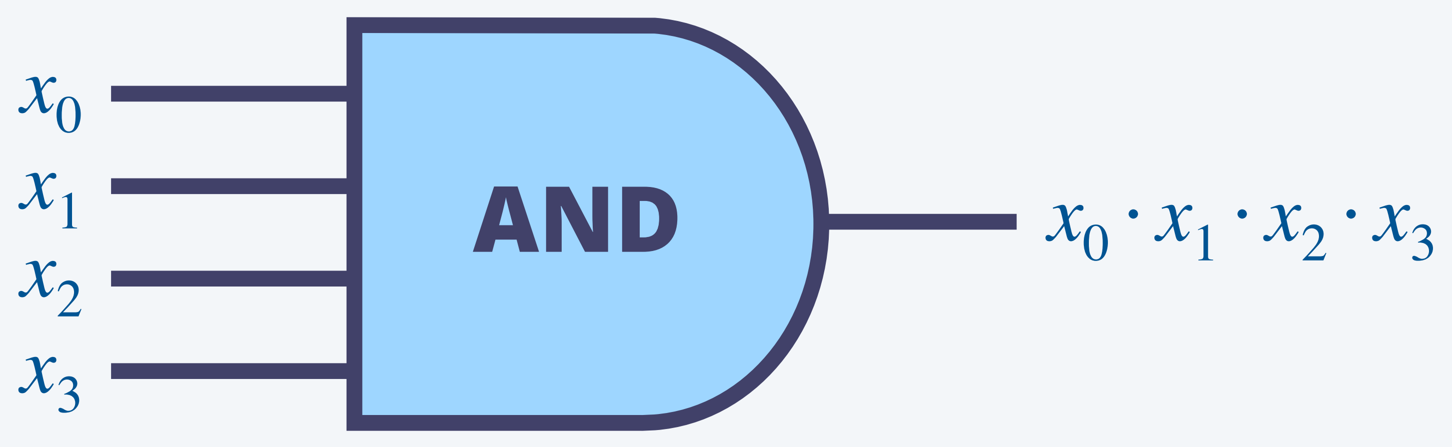
**E.** None of the above.



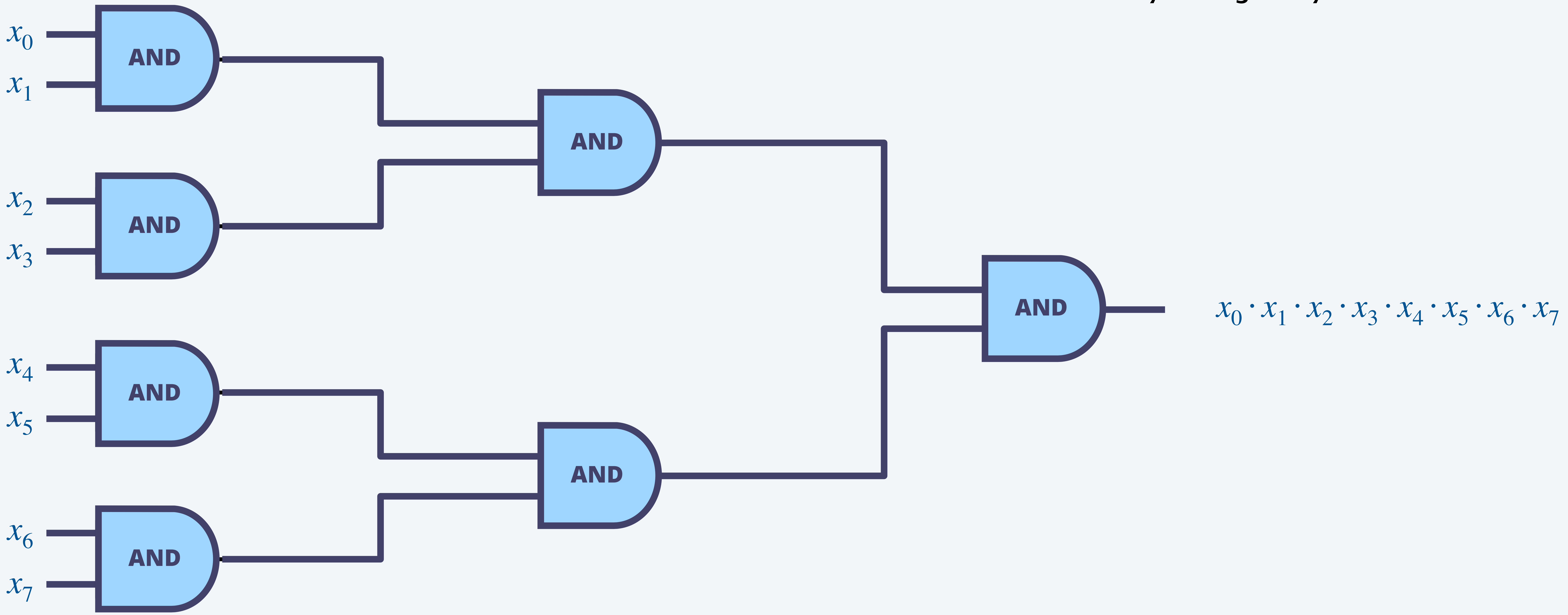
# Multiway *AND* gates

Multiway *AND* gate.

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




4-way AND gate symbol

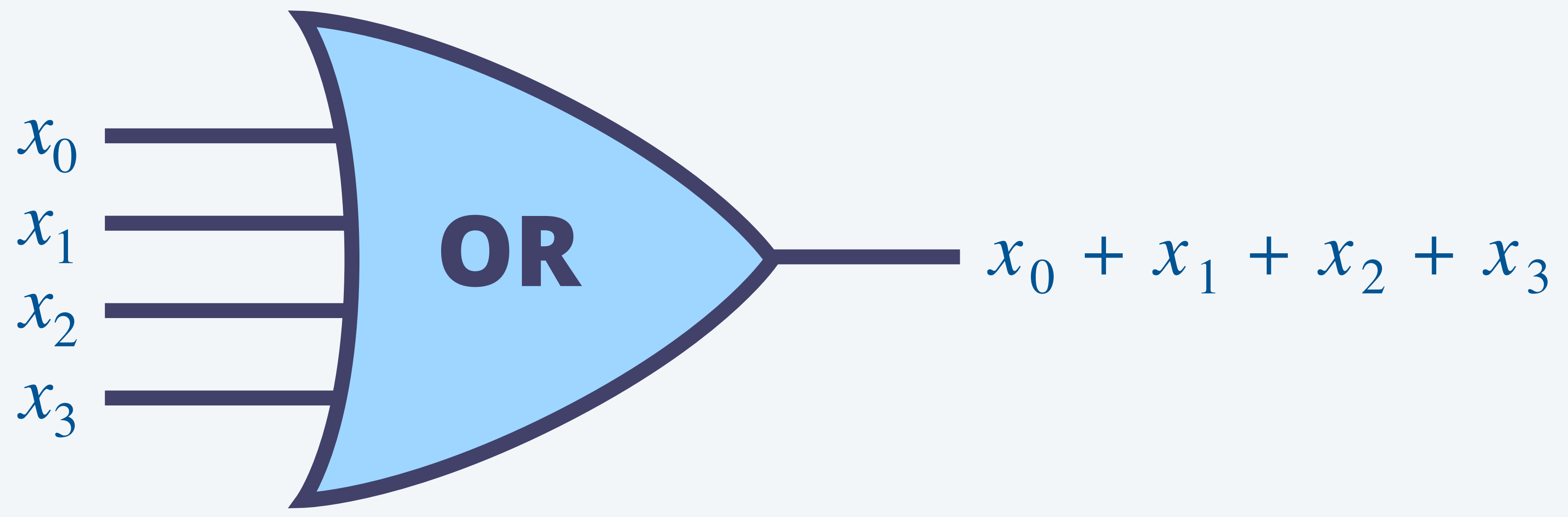



8-way AND gate implementation
(tree of 2-way AND gates)

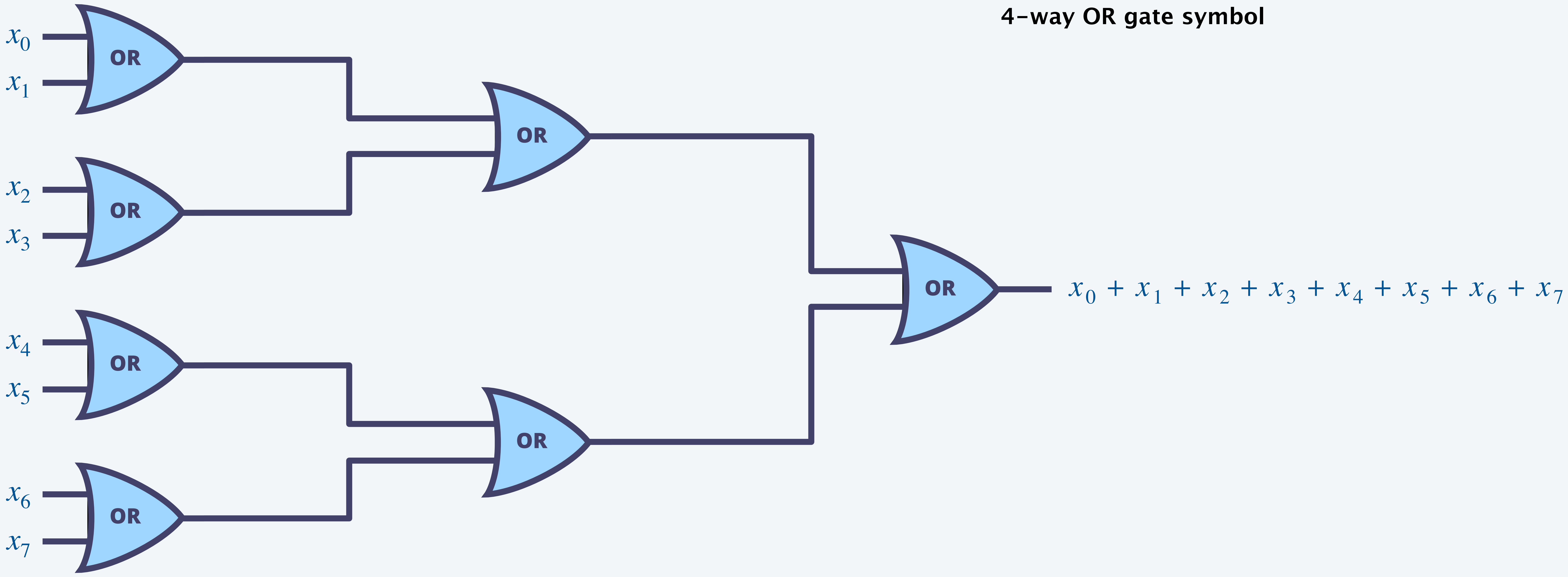
# Multiway $OR$ gates

## Multiway $OR$ gate.

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




4-way OR gate symbol

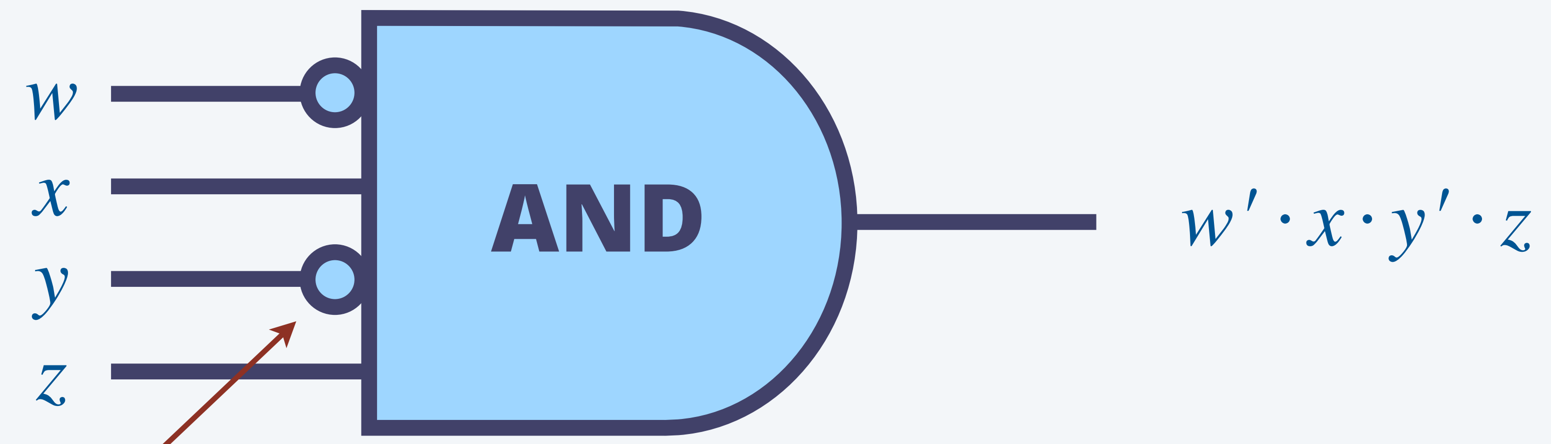



8-way OR gate implementation
(tree of 2-way OR gates)

# Generalized *AND* gates

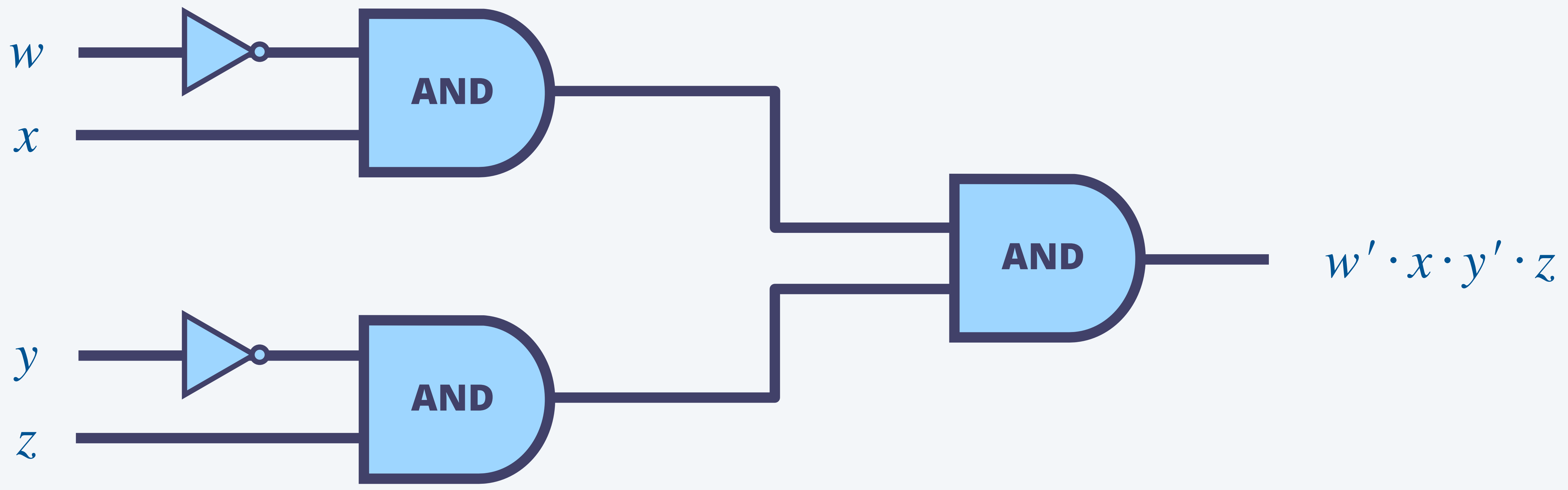
Generalized *AND* gate.

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

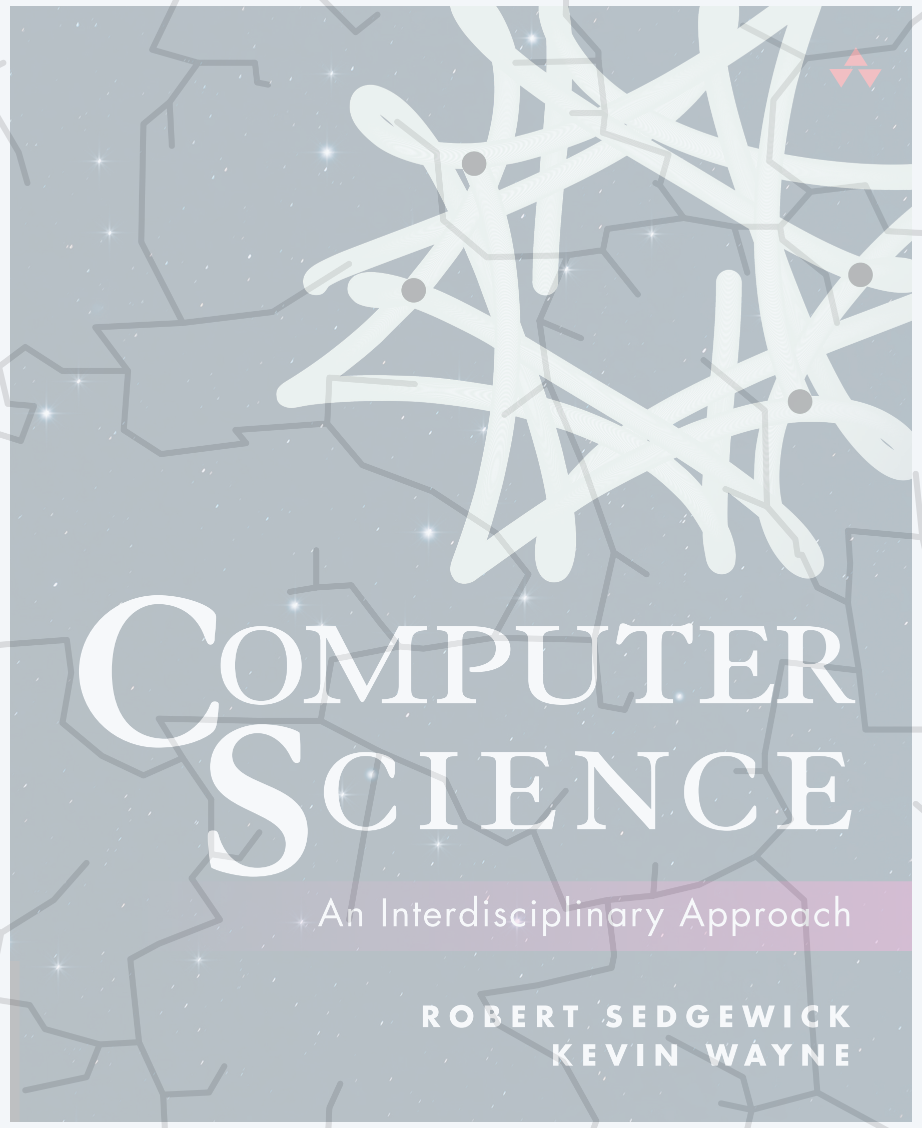



4-way generalized AND gate symbol

*each "inversion bubble" denotes a NOT gate*




4-way generalized AND gate implementation
(tree of 2-way AND gates, plus NOT gates)




https://introcs.cs.princeton.edu

# 7. DIGITAL CIRCUITS

- *boolean algebra*
- *logic gates*
- *sum-of-products*
- *adder circuit*

# Sum-of-products

Sum-of-products. Every boolean function can be represented as a sum of products.

- Products: form an *AND* term for each 1 in truth table.
- Sum: combine the terms with the *OR* function.

*also known as "disjunctive normal form"*

$(x' \ y \ z) + (x \ y' \ z) + (x \ y \ z') + (x \ y \ z) \ = \ MAJ$

| $x$ | $y$ | $z$ | $MAJ$ | $x' \ y \ z$ | $x \ y' \ z$ | $x \ y \ z'$ | $x \ y \ z$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

**Expressing MAJ(x, y, z) as a sum of products**

# Universality

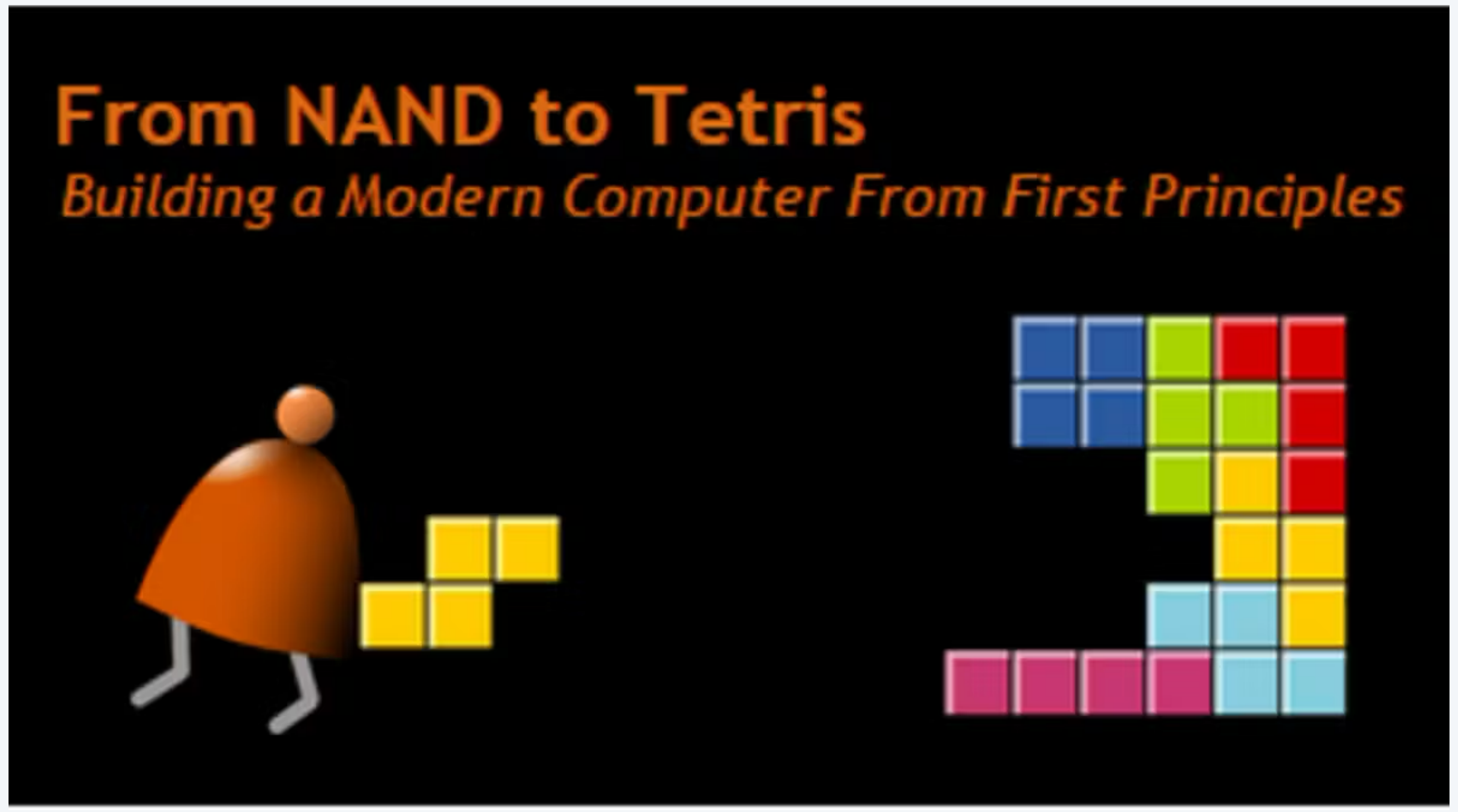
Def. A set of operations is universal if every boolean function can be expressed using just those operations.

Proposition. $\{ AND, OR, NOT \}$ is a universal set of operations.
Pf. Sum-of-products construction on previous slide.

Proposition. $\{ NAND \}$ is a universal set of operations.
Pf. $\{ AND, OR, NOT \}$ can be constructed from $NAND$. ← *see precept*

| $x$ | $y$ | $NAND$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

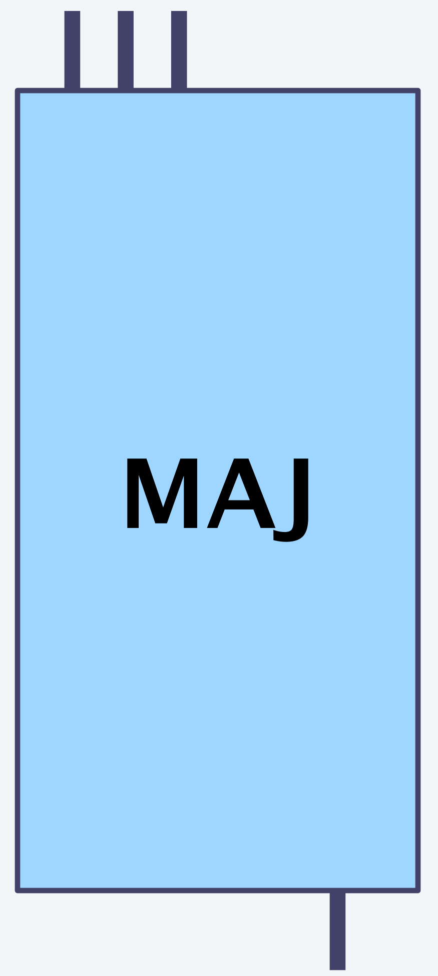
NAND



# Majority function

Sum-of-products construction.

- Identify rows of truth table where the function is $1$.
- Use a generalized $AND$ gate for each term.
- Combine the terms using an $OR$ gate.

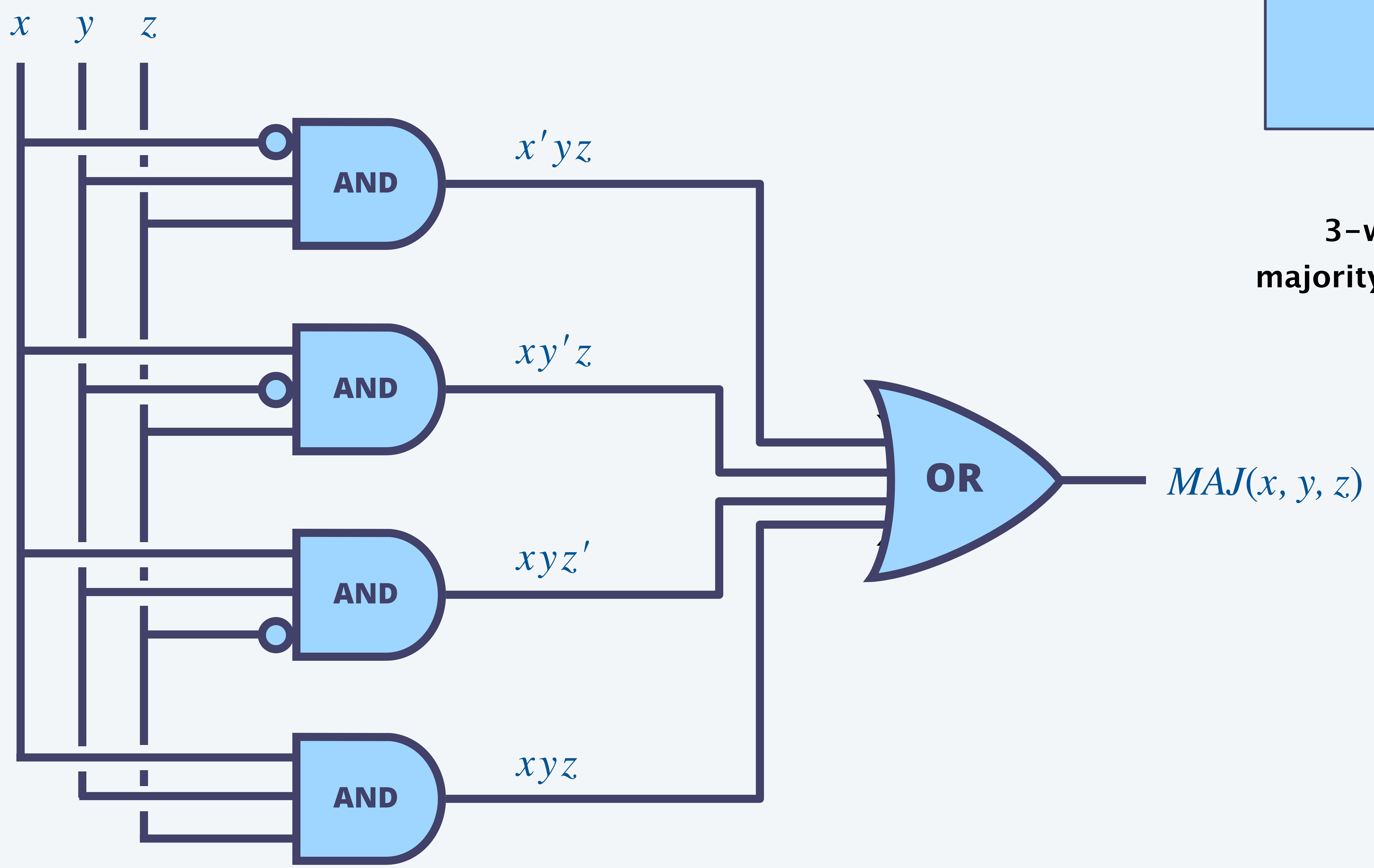



3-way majority circuit

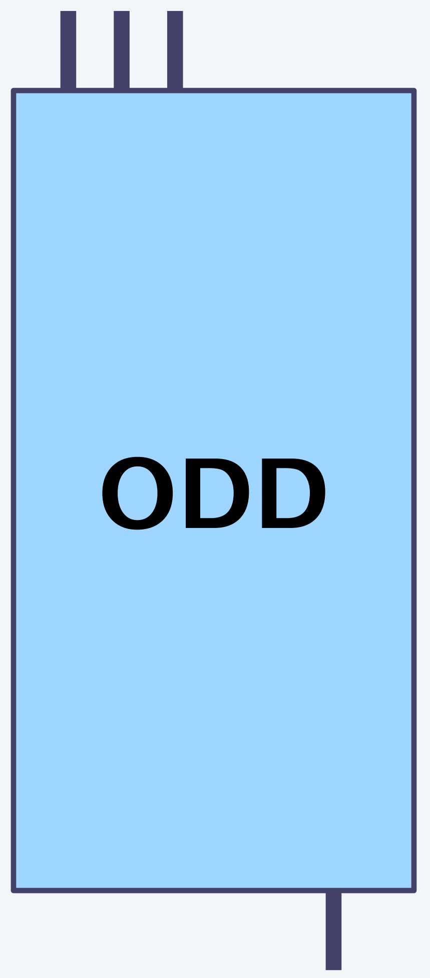
Ex 1. Majority function.

| $x$ | $y$ | $z$ | $MAJ$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | ← $x'yz$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | ← $xy'z$ |
| 1 | 1 | 0 | 1 | ← $xyz'$ |
| 1 | 1 | 1 | 1 | ← $xyz$ |

$$MAJ(x, y, z) = x'yz + xy'z + xyz' + xyz$$



# Odd-parity function




3-way odd parity circuit

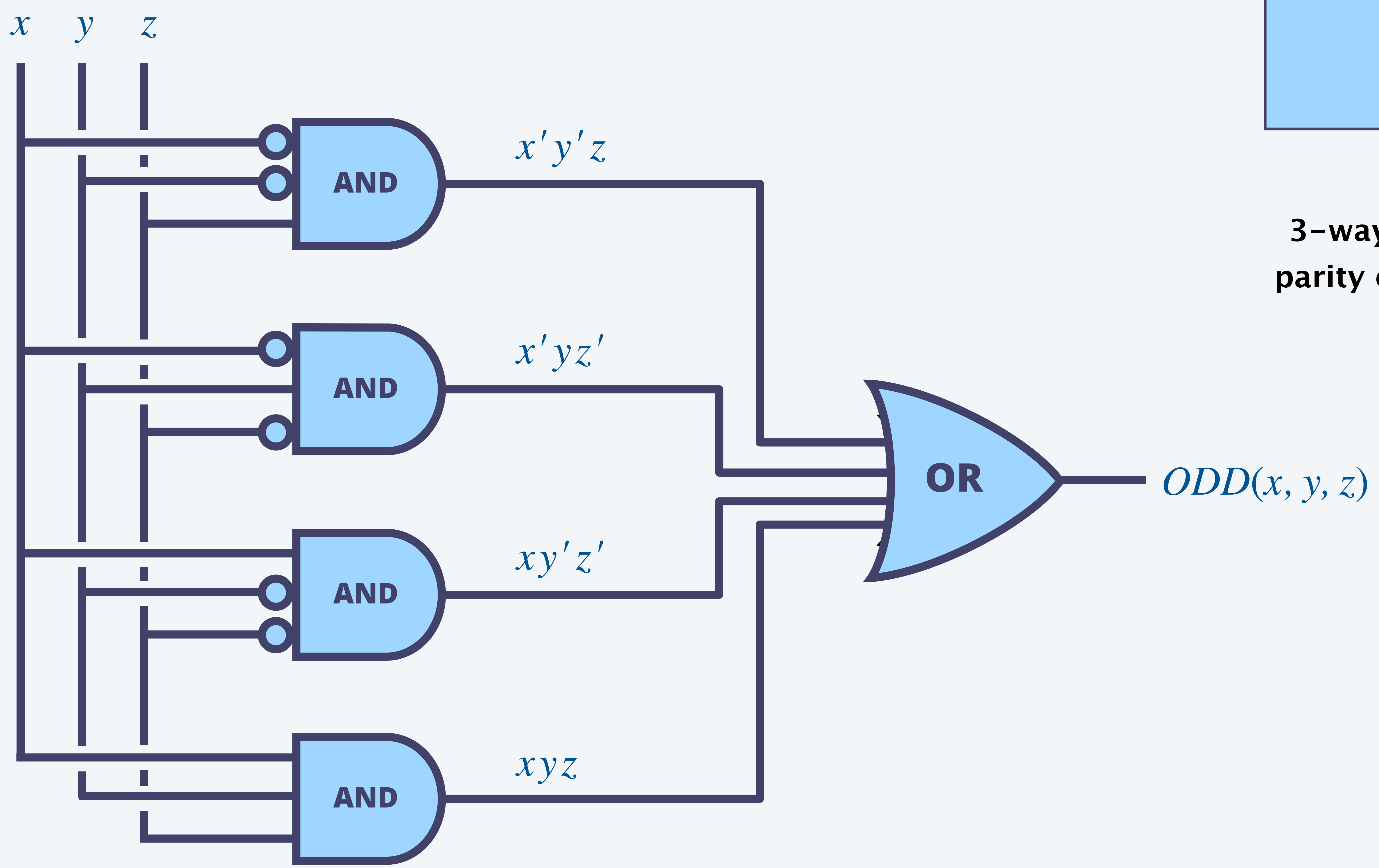
**Sum-of-products construction.**

- Identify rows of truth table where the function is $1$.
- Use a generalized *AND* gate for each term.
- Combine the terms using an *OR* gate.

**Ex 2.** Odd-parity function.

| $x$ | $y$ | $z$ | $ODD$ | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 1 | ← $x'y'z$ |
| 0 | 1 | 0 | 1 | ← $x'yz'$ |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | ← $xy'z'$ |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | ← $xyz$ |

$$ODD(x, y, z) = x'y'z + x'yz' + xy'z' + xyz$$



# Sum-of-products construction (summary)

Goal. Design a digital circuit that computes a given boolean function.

Recipe.

- 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 $AND$ gate for each row, and $OR$ the results.

Profound consequence. Can design a digital circuit for ANY boolean function.

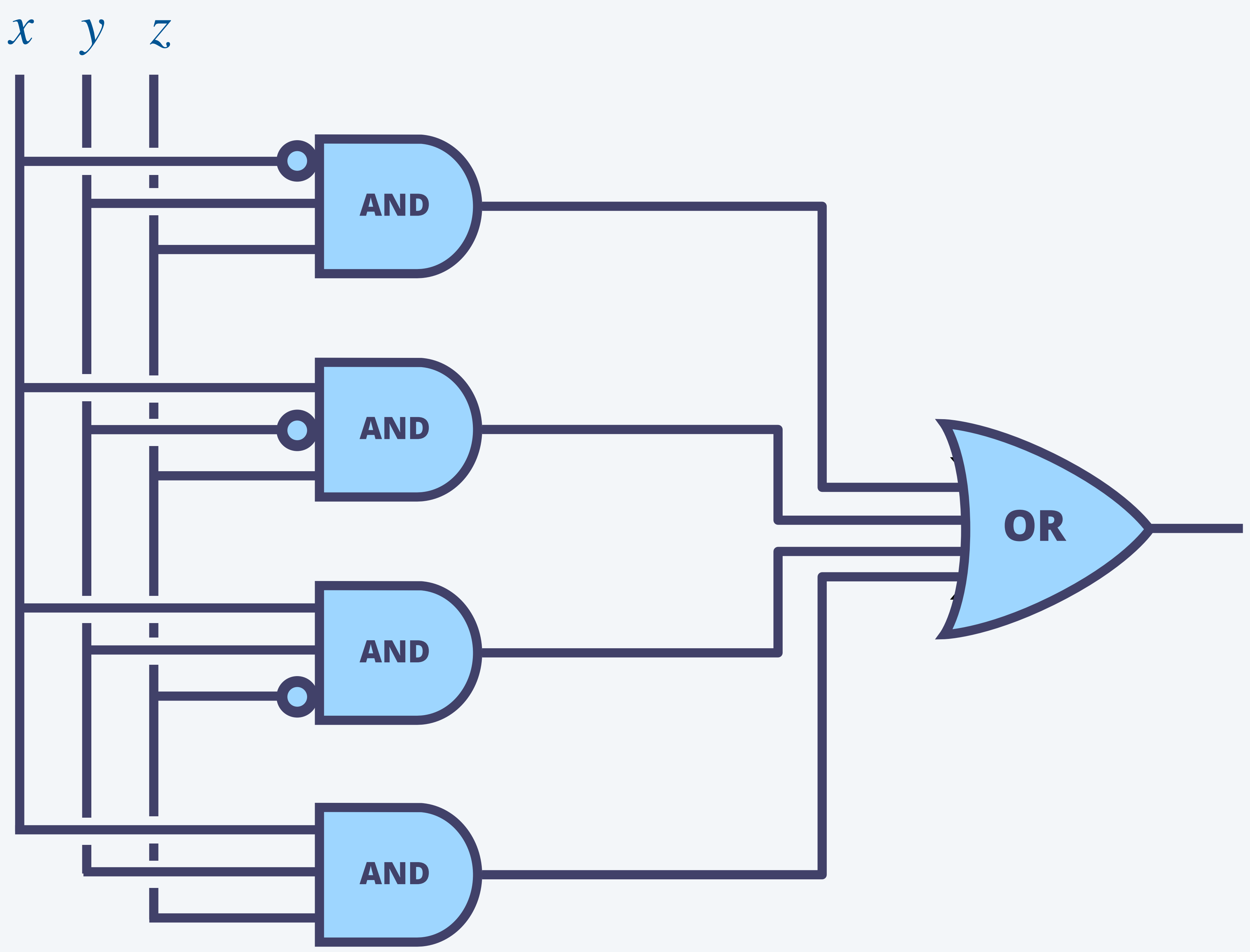
# Optimized digital circuits

Caveat. Sum-of-products construction is not optimal in terms of:

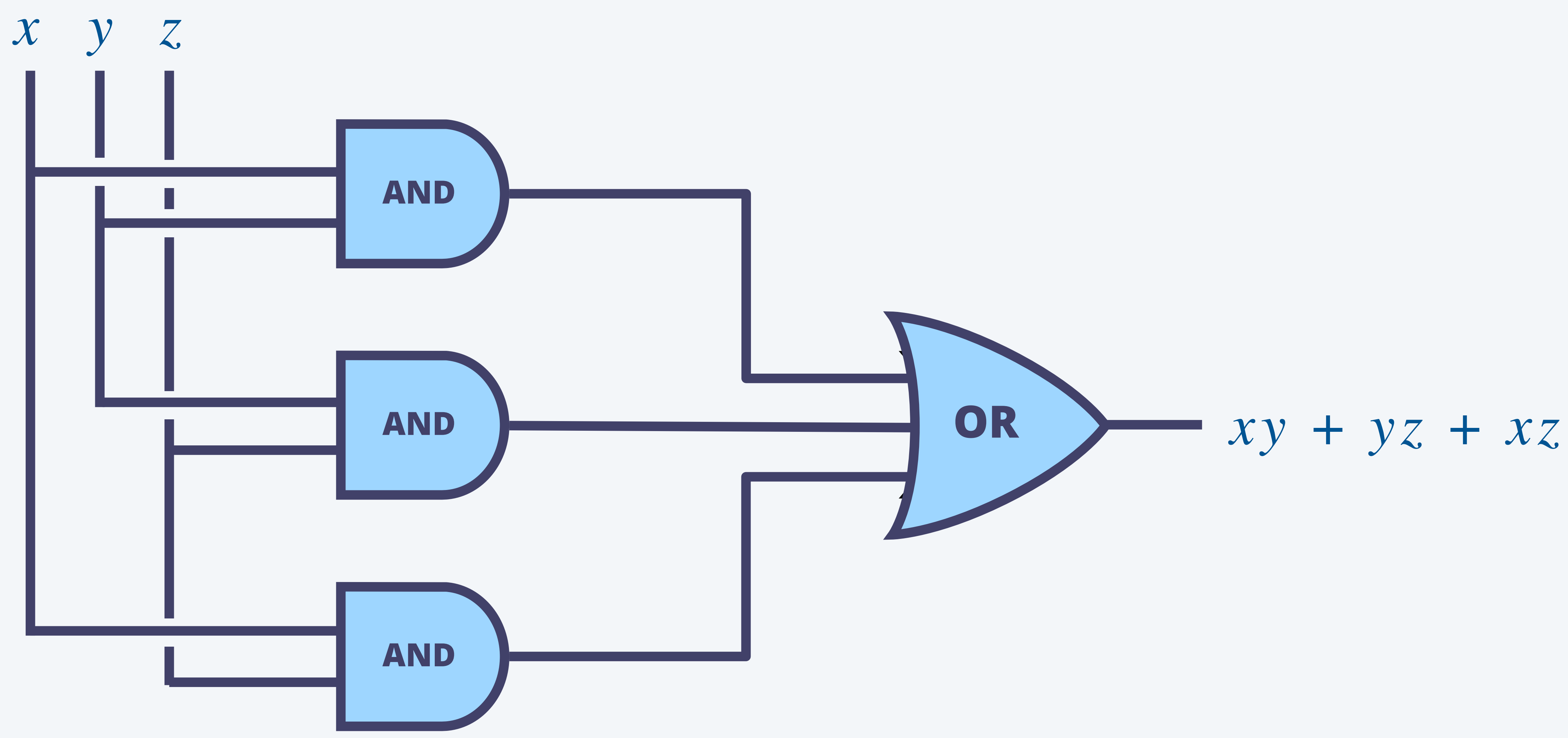
- Space = number of gates.
- Time = depth of circuit.

*this course: we'll ignore such low-level optimization*

Ex. Majority function (3-bit).




**3-way majority circuit (sum-of-products)**




**3-way majority circuit (optimized)**

# Digital circuits: quiz 3

**How many 3-way generalized AND gates are needed to build the sum-of-products circuit for the following truth table?**

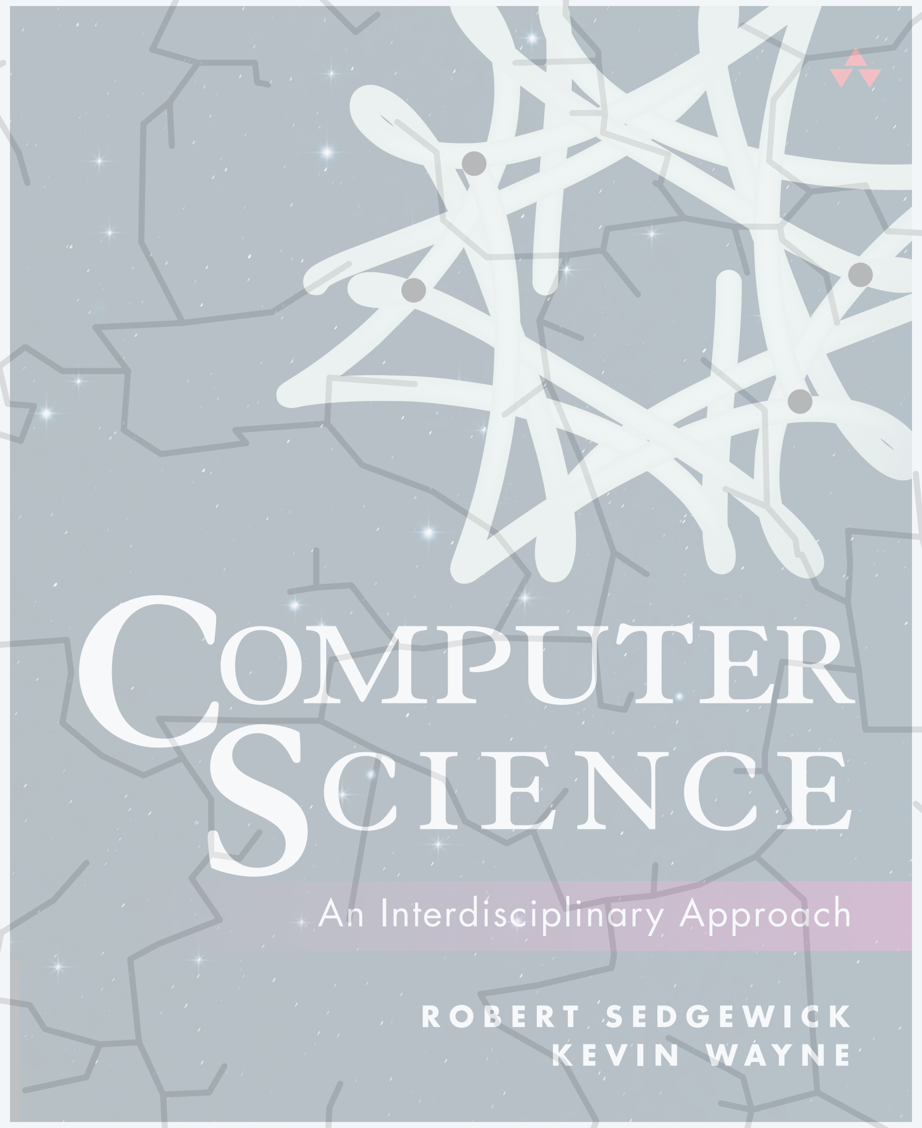
**A.** 1

**B.** 2

**C.** 3

**D.** 4

| $x$ | $y$ | $z$ | $EQ$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




https://introcs.cs.princeton.edu

# 7. DIGITAL CIRCUITS

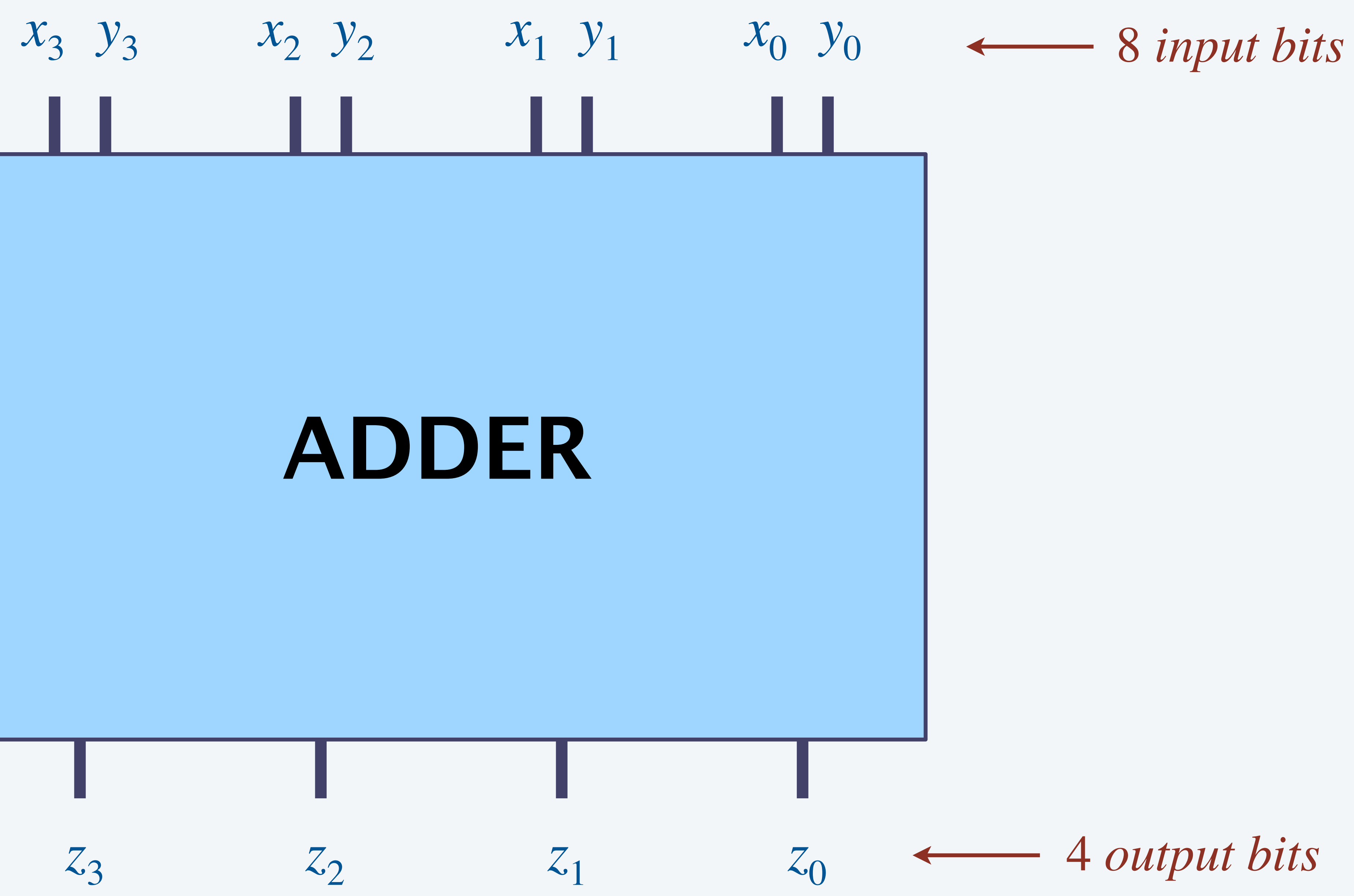
- boolean algebra
- logic gates
- sum-of-products
- adder circuit

# Let's make an adder circuit!

**Adder circuit.** Compute $z = x + y$ for 4-bit binary integers. ← *ignore integer overflow (as in TOY and Java)*

**First step.** Represent inputs and outputs in binary.

$$\begin{array}{ccccc} & x_3 & x_2 & x_1 & x_0 \\ + & y_3 & y_2 & y_1 & y_0 \\ \hline & z_3 & z_2 & z_1 & z_0 \end{array}$$

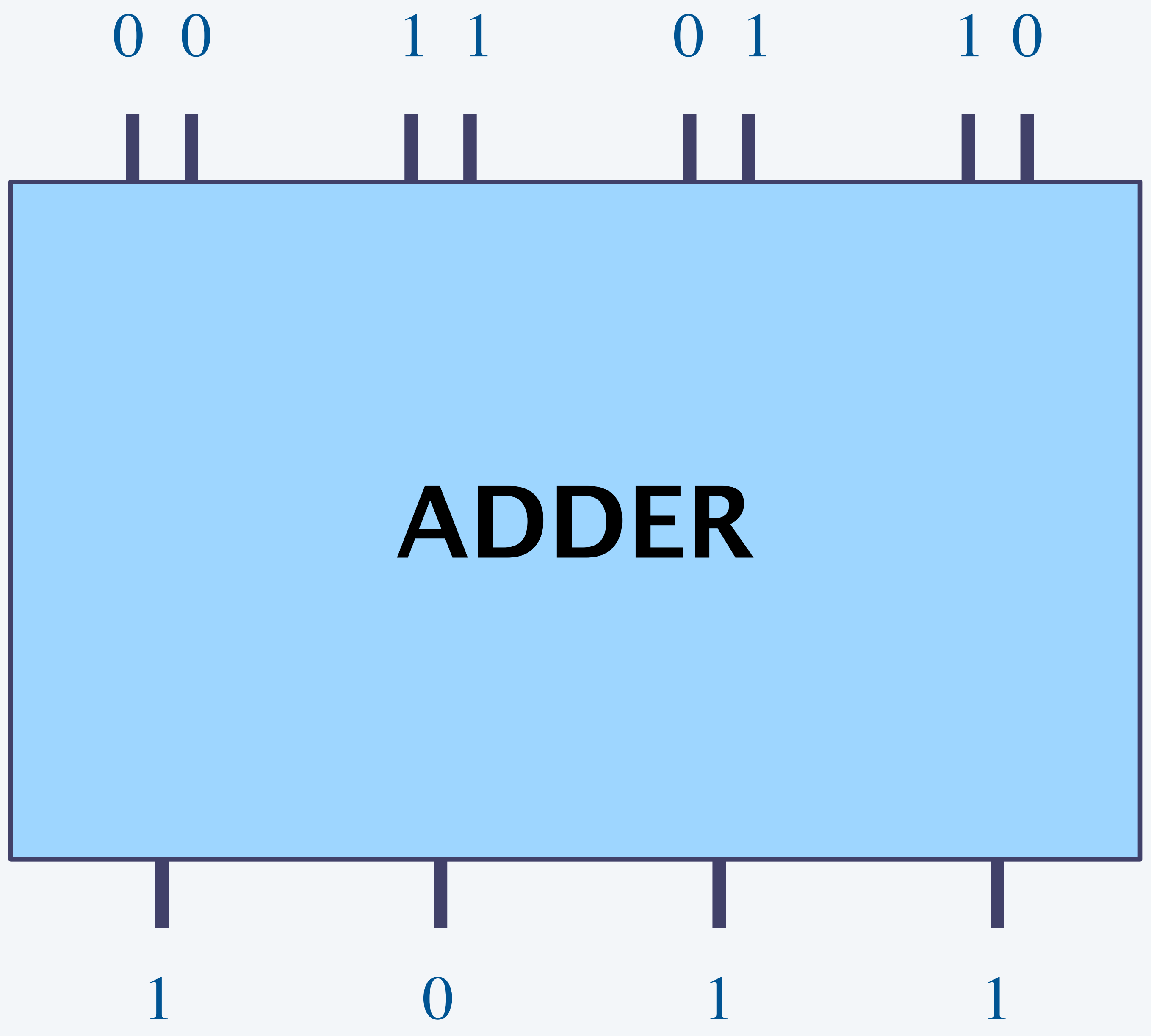


# Let's make an adder circuit!

**Adder circuit.** Compute $z = x + y$ for 4-bit binary integers. ← *ignore integer overflow (as in TOY and Java)*

**First step.** Represent inputs and outputs in binary.

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 0 | 1 |
| + | 0 | 1 | 1 | 0 |
| | 1 | 0 | 1 | 1 |



# Let's make an adder circuit!

Adder circuit. Compute $z = x + y$ for 4-bit binary integers.

*exceeds number of electrons in universe (!)*

Straw-person solution. Build a truth table for each output bit.

Approach is not scalable! Truth table for 128-bit adder would have $2^{256}$ rows.

$$\begin{array}{ccccc} & x_3 & x_2 & x_1 & x_0 \\ + & y_3 & y_2 & y_1 & y_0 \\ \hline & z_3 & z_2 & z_1 & z_0 \end{array}$$

| $x_3$ | $x_2$ | $x_1$ | $x_0$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ | $z_3$ | $z_2$ | $z_1$ | $z_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

*$2^8 = 512$ rows*

**truth table for 4-bit adder**

# Let's make an adder circuit!

Adder circuit. Compute $z = x + y$ for 4-bit binary integers.

$c_0 = 0$

$$\begin{array}{ccccc} & c_3 & c_2 & c_1 & c_0 \\ & x_3 & x_2 & x_1 & x_0 \\ + & y_3 & y_2 & y_1 & y_0 \\ \hline & z_3 & z_2 & z_1 & z_0 \end{array}$$

Efficient solution. Do one bit at a time.

- Build truth table for each carry bit. ← *3-bit majority function* (!)
- Build truth table for each sum bit.

| $x_i$ | $y_i$ | $c_i$ | $c_{i+1}$ | $MAJ$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**truth table for carry bit**

$c_{i+1} = MAJ(x_i, y_i, c_i)$

# Let's make an adder circuit!

**Adder circuit.** Compute $z = x + y$ for 4-bit binary integers.

$c_0 = 0$

| | | | | |
|---|---|---|---|---|
| | $c_3$ | $c_2$ | $c_1$ | $c_0$ |
| | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
| + | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| | $z_3$ | $z_2$ | $z_1$ | $z_0$ |

**Efficient solution.** Do one bit at a time.

- Build truth table for each carry bit ← *3-bit majority function* (!)
- Build truth table for each sum bit. ← *3-bit odd-parity function* (!)

| $x_i$ | $y_i$ | $c_i$ | $z_i$ | $ODD$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**truth table for sum bit**

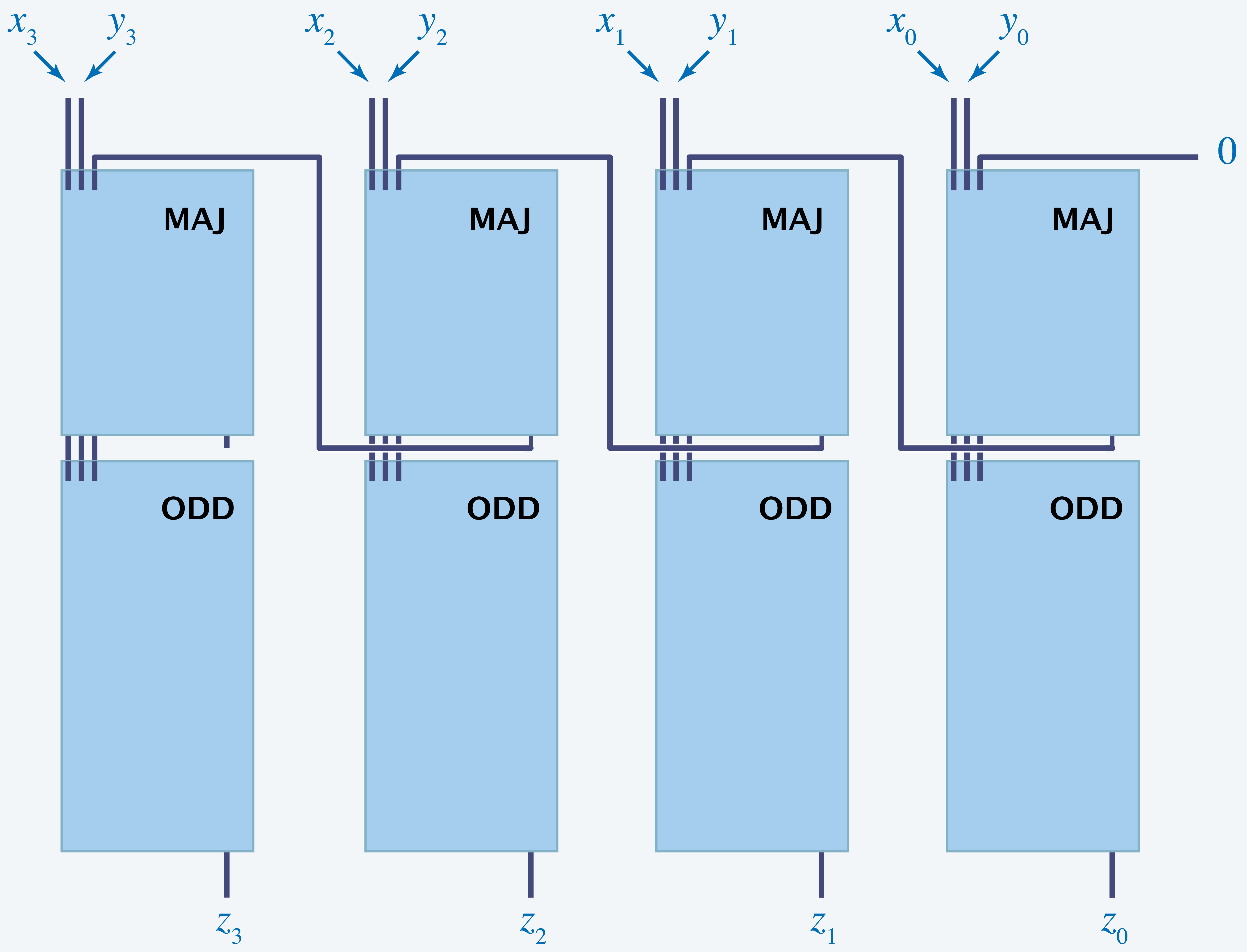
$z_i = ODD(x_i, y_i, c_i)$

# Let's make an adder circuit!

Adder circuit. Compute $z = x + y$ for 4-bit binary integers.

$c_0 = 0$

$$
\begin{array}{rcccc}
 & c_3 & c_2 & c_1 & c_0 \\
 & x_3 & x_2 & x_1 & x_0 \\
+ & y_3 & y_2 & y_1 & y_0 \\
\hline
 & z_3 & z_2 & z_1 & z_0
\end{array}
$$




Efficient solution. Do one bit at a time.

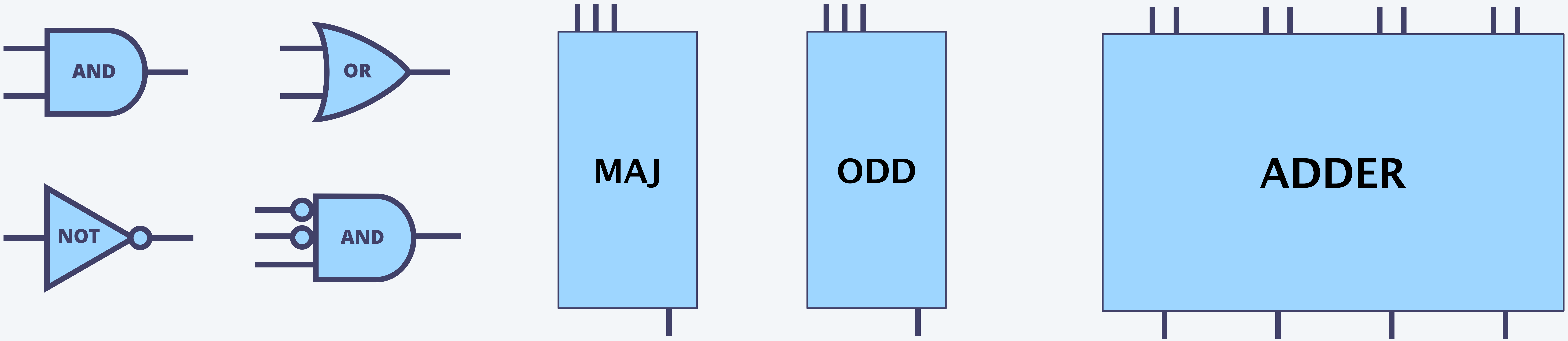
- Carry bit is *MAJ*.
- Sum bit is *ODD*.
- Chain 1-bit adders to “ripple” carries.

Size of circuit. $\Theta(n)$ gates for $n$-bit adder.

# Encapsulation

Encapsulation in circuit design mirrors familiar software design principle.

- API describes behavior (input and outputs) of circuit.
- Implementation gives details of how to build it from wires and gates.
- Client uses circuit as a black box.



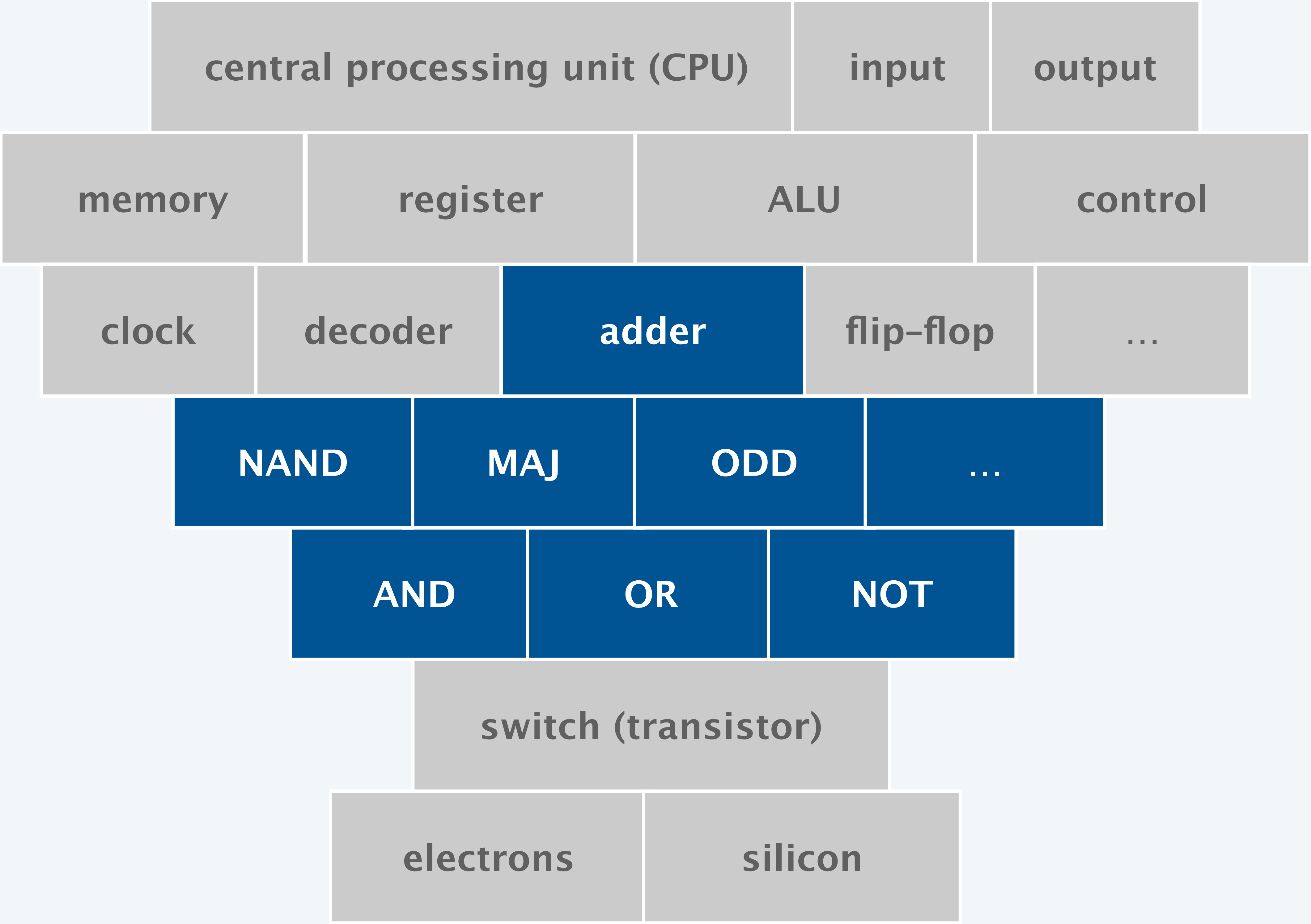

Bottom line. We manage complexity by encapsulating circuits.

# Layers of abstraction

Layers of abstraction apply with a vengeance.

- On/off.
- Switch.
- Primitive gates (*AND*, *OR*, *NOT*).
- Composite gates (multiway *AND*/*OR*, *MAJ*, *ODD*).
- Adder circuit.
- Memory.
- Arithmetic logic unit (ALU).
- Central processing unit (CPU).
- Input and output.
- Your computer.

Want to learn more? See ECE 206 and ECE 365.



# Credits

Co-instructors, course admin, and graduate student preceptors.

Undergrad graders, precept assistants, and lab TAs.

☞ Apply to be one next semester!

## Senior Staff


Kobi Kaplan


Donna Gabai


Alan Kaplan

## Assistant Instructors


Tanvi Namjoshi


Ruyu Yan


Owen Zhang


Nobline Yoo


Max Gonzalez-Saez


Kylie Zhang


Kathryn Wantlin


Jane Castleman


Beza Desta


Nicholas Alexander Sudarsky


Berlin Chen


Abhishek Joshi

# A final thought

# Credits

| image | source | license |
| --- | --- | --- |
| *Retro Telephone and Smartphone* | Adobe Stock | education license |
| *Macbook Pro* | Apple | |
| *Samsung Galaxy S23* | Samsung | |
| *Xbox One* | Microsoft | |
| *Cardiac Pacemaker* | Adobe Stock | education license |
| *Apple A16 Bionic Chip* | Apple | |
| *Boole is Coole* | IrishPhilosophy | CC BY-NC-SA 2.0 |
| *Boole Orders Lunch* | Sidney Harris | |
| *Trick OR Treat* | IFL Science | |
| *From NAND to Tetris* | nand2tetris.org | |

# Credits

| image | source | license |
| --- | --- | --- |
| *Claude Shannon* | Lucent Technologies | |
| *Bit Player Theatrical Poster* | thebitplayer.com | |
| *John Hutton as Claude Shannon* | thebitplayer.com | |
| *Logic Gate Symbols* | Adobe Stock | education license |
| *Apple MacBook Pro* | Adobe Stock | education license |