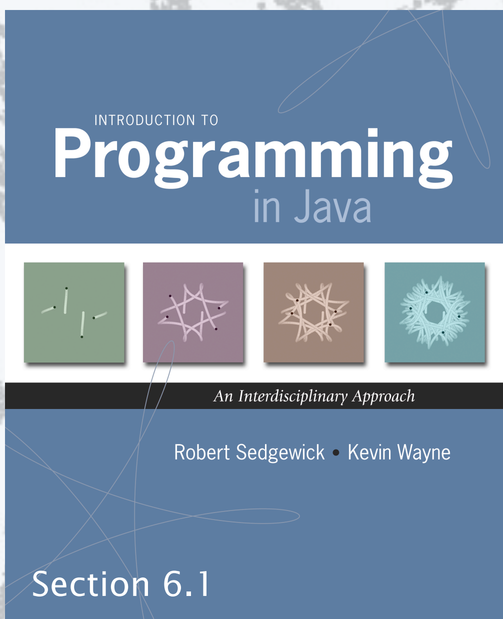


http://introcs.cs.princeton.edu

20. Combinational Circuits

Combinational circuits

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?

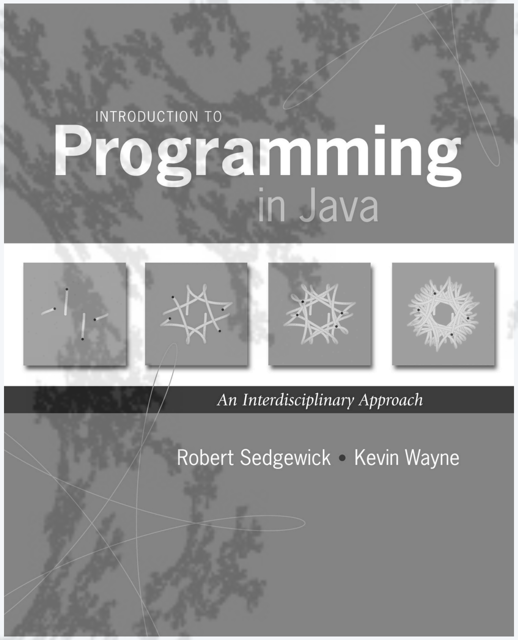
A. Accurate, reliable, general purpose, fast, cheap.

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*, ...

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E



http://introcs.cs.princeton.edu

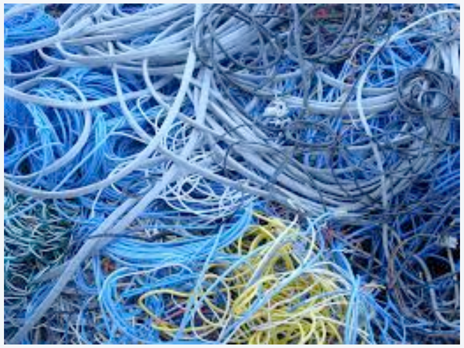
20. Combinational Circuits

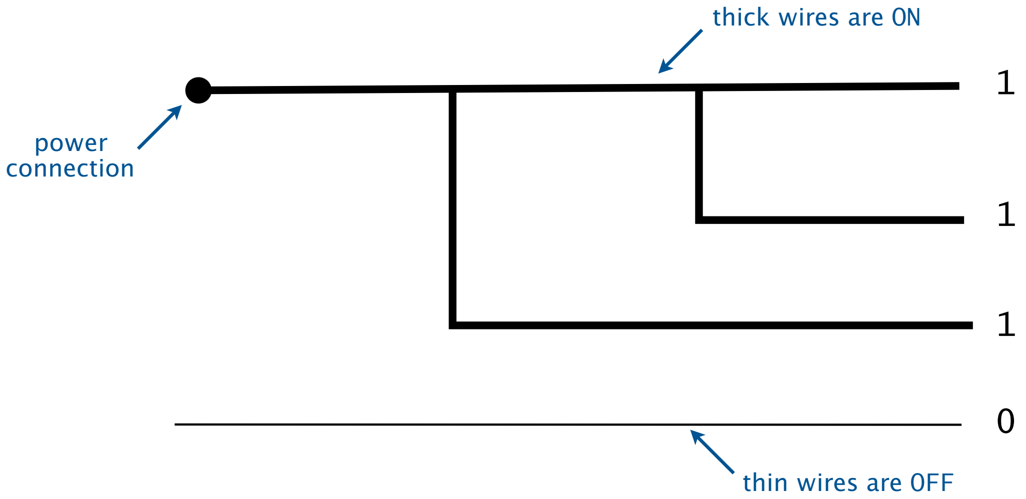
- **Building blocks**
- Boolean algebra
- Digital circuits
- Adder

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.

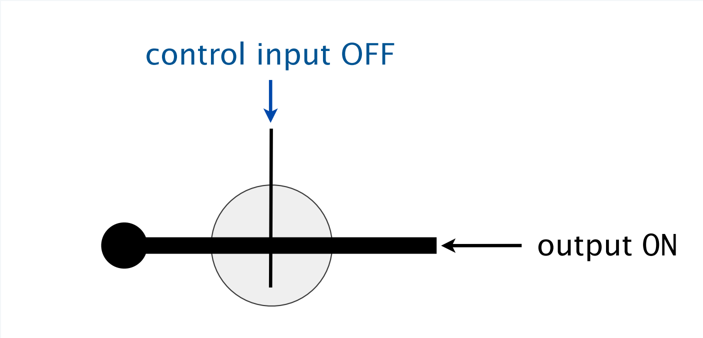


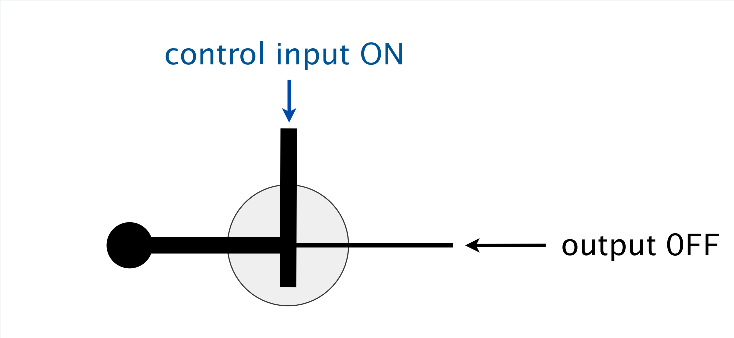


Controlled Switch

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

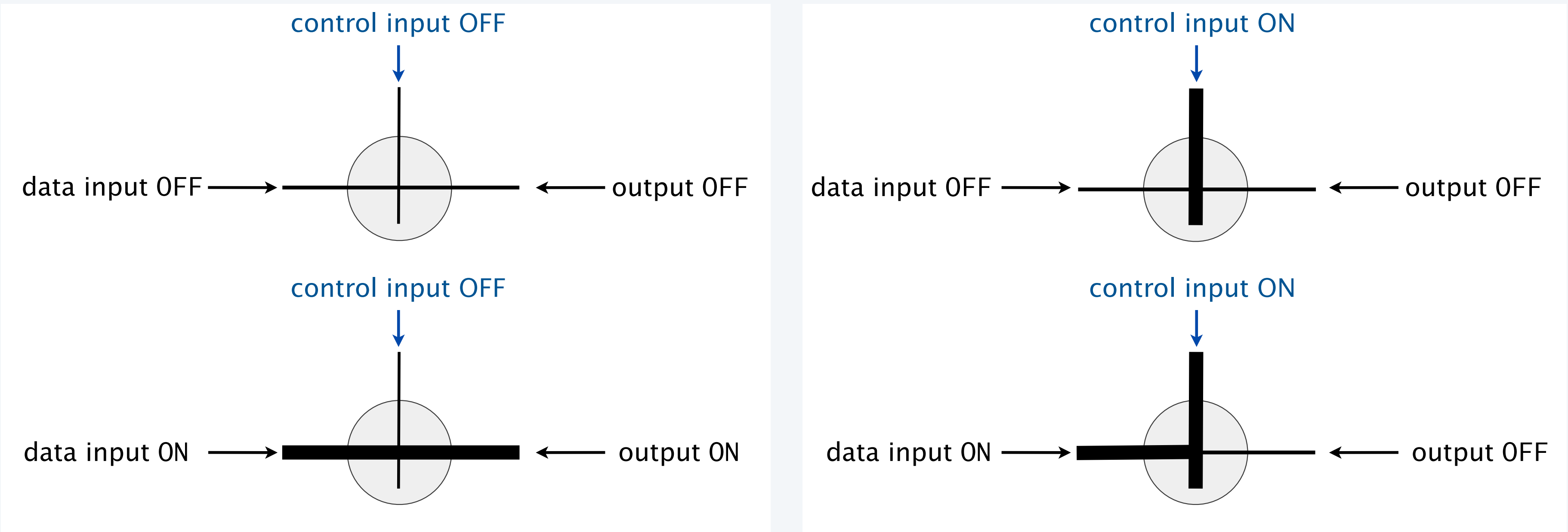




Controlled Switch

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

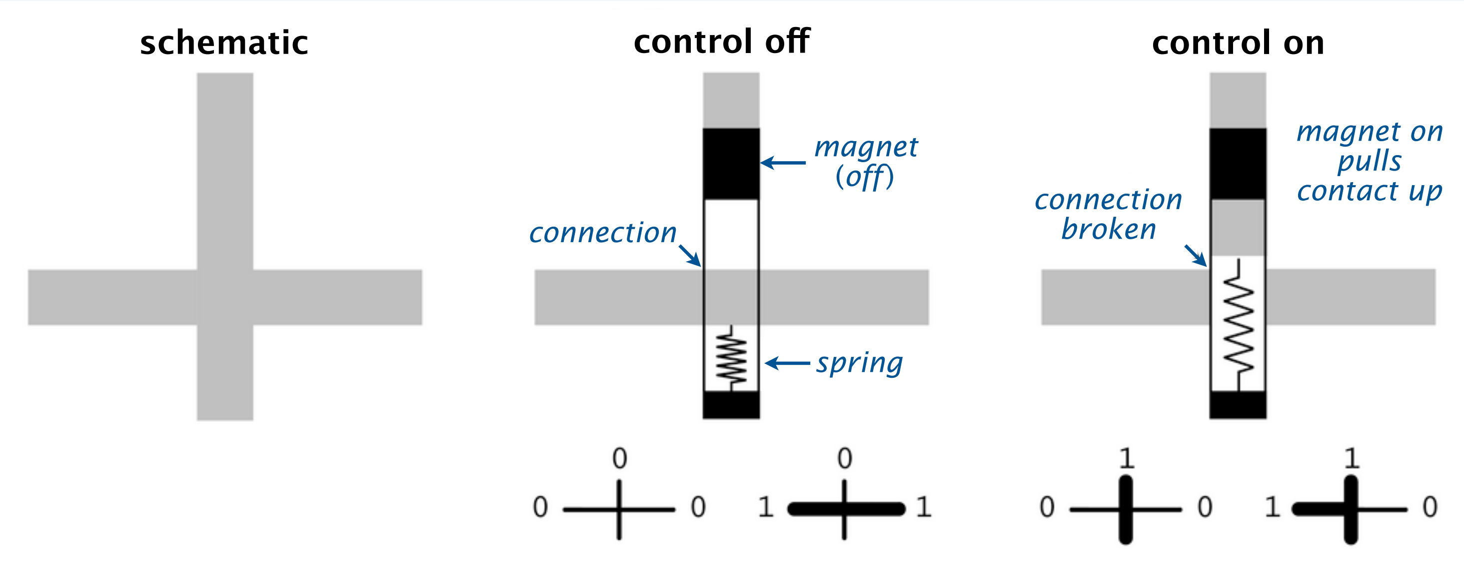


Idealized model of *pass transistors* found in real integrated circuits.

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.

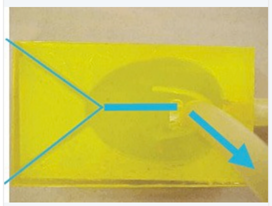
New technology immediately gives new computer.

Better switch? Better computer.

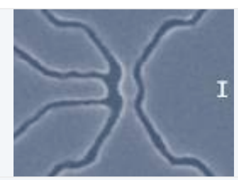
Basis of Moore's law.

all built with
"switches and wires"

Switches and wires: a first level of abstraction

technology	*"information"*	*switch*
pneumatic	air pressure	
fluid	water pressure	
relay	electric potential	

Amusing attempts that do not scale but prove the point

technology	*switch*
relay	
vacuum tube	
transistor	
"pass transistor" in integrated circuit	
atom-thick transistor	

Real-world examples that prove the point

Switches and wires: a first level of abstraction

VLSI = Very Large Scale Integration

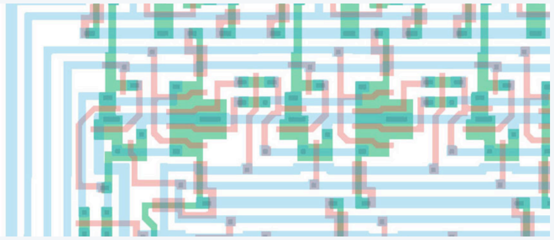
Technology
Deposit materials on substrate.

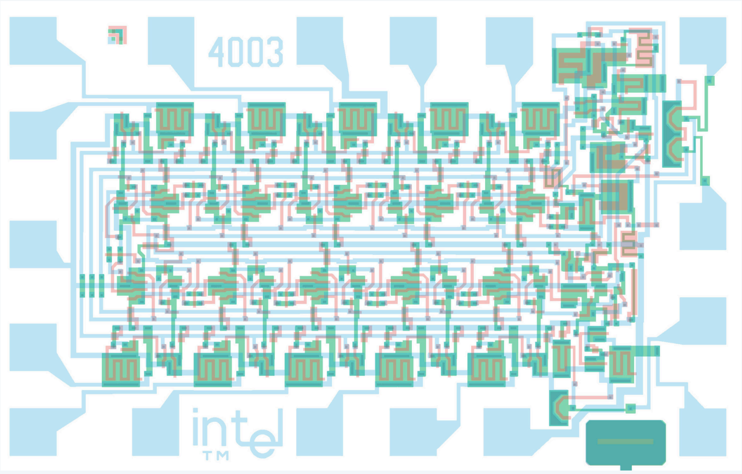
Key properties
Lines are wires.
Certain crossing lines are controlled switches.

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

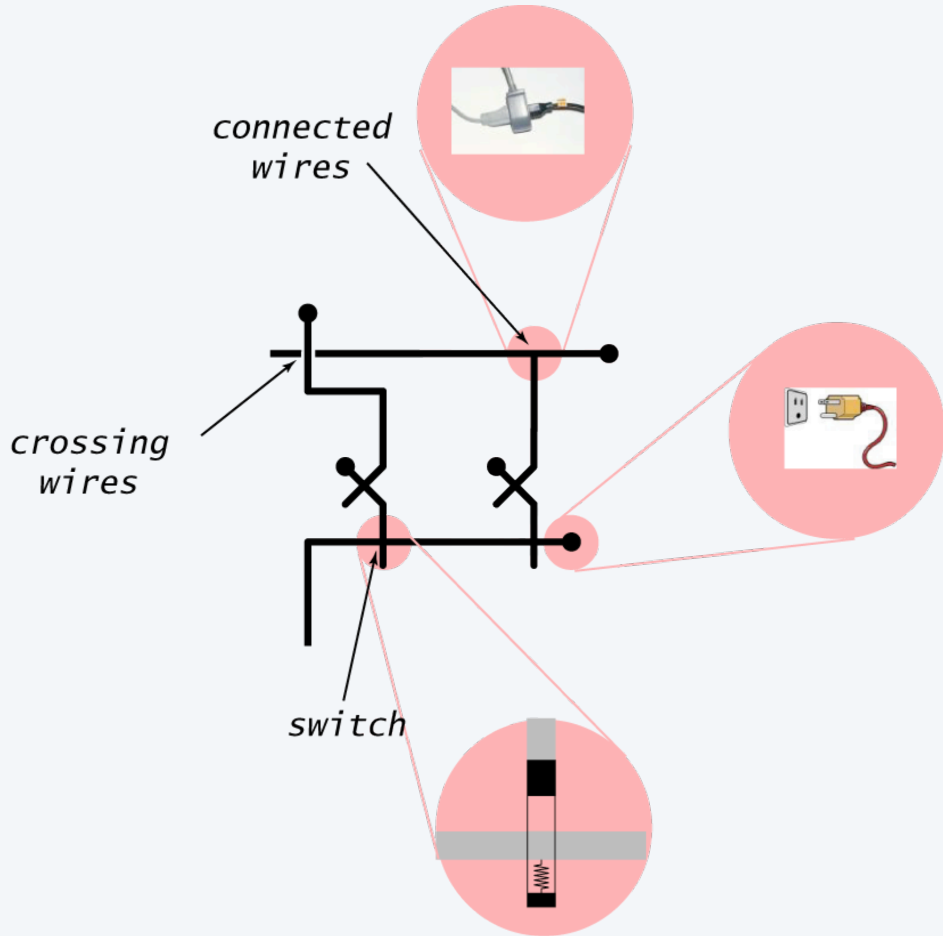
Key challenge in “abstract” world
Understanding behavior of circuits with
billions of wires and controlled switches

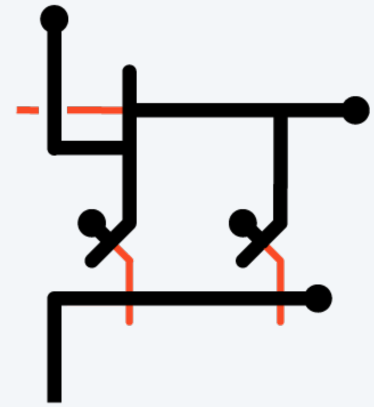
Bottom line. Circuit = Drawing (!)





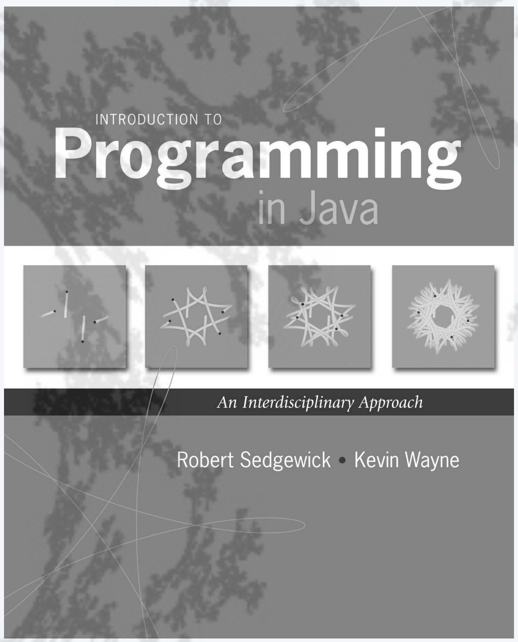
Circuit anatomy





Need more levels of abstraction to understand circuit behavior

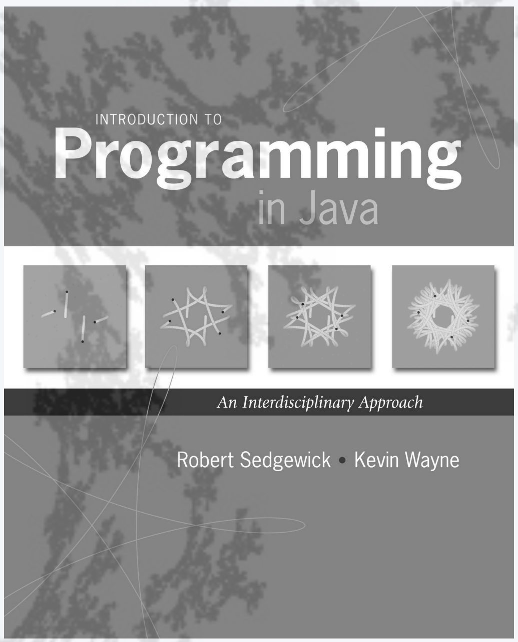
COMPUTER SCIENCE
S E D G E W I C K / W A Y N E



http://introcs.cs.princeton.edu

20. Combinational Circuits

- **Building blocks**
- Boolean algebra
- Digital circuits
- Adder



http://introcs.cs.princeton.edu

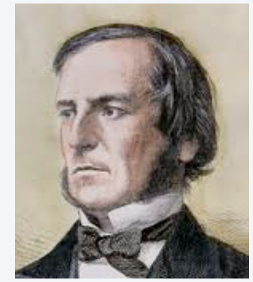
20. Combinational Circuits

- Building blocks
- **Boolean algebra**
- Digital circuits
- Adder

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.



George Boole
1815–1864

operation	*Java notation*	*logic notation*	*circuit design (this lecture)*
AND	x && y	$x \wedge y$	xy
OR	x || y	$x \vee y$	$x + y$
NOT	! x	$\neg x$	x'

various notations in common use

DeMorgan's Laws

Example: (stay tuned for proof)

$$(xy)' = (x' + y')$$
$$(x + y)' = x'y'$$

Relevance to circuits. Basis for next level of abstraction.

Truth tables

A truth table is a systematic way to define a Boolean function

- One row for each possible set of argument values.
- Each row gives the function value for the specified argument values.
- N inputs: 2^N rows needed.

x	x'
0	1
1	0

NOT

x	y	xy
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	*NOR*
0	0	1
0	1	0
1	0	0
1	1	0

NOR

x	y	*XOR*
0	0	0
0	1	1
1	0	1
1	1	0

XOR

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

x	y	xy	$(xy)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

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

$(xy)' = (x' + y')$

x	y	$x + y$	$(x + y)'$ NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x	y	x'	y'	$x'y'$ NOR
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

$(x + y)' = x'y'$

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).

Truth tables for all Boolean functions of 2 variables

x	*y*	*ZERO*	*AND*		*x*		*y*	*XOR*	*OR*	*NOR*	*EQ*	*¬y*		*¬x*		*NAND*	*ONE*
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

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).

x	*y*	*z*	*AND*	*OR*	*NOR*	*MAJ*	*ODD*
0	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1
0	1	0	0	1	0	0	1
0	1	1	0	1	0	1	0
1	0	0	0	1	0	0	1
1	0	1	0	1	0	1	0
1	1	0	0	1	0	1	0
1	1	1	1	1	0	1	1

Some Boolean functions of 3 variables

all extend to *N* variables

Examples

AND	logical AND	0 iff *any* inputs is 0 (1 iff all inputs 1)
OR	logical OR	1 iff *any* input is 1 (0 iff all inputs 0)
NOR	logical NOR	0 iff *any* input is 1 (1 iff all inputs 0)
MAJ	majority	1 iff more inputs are 1 than 0
ODD	odd parity	1 iff an odd number of inputs are 1

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

A. 2^{2^N}

N	*number of Boolean functions with N variables*
2	$2^4 = 16$
3	$2^8 = 256$
4	$2^{16} = 65{,}536$
5	$2^{32} = 4{,}294{,}967{,}296$
6	$2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$

Universality of AND, OR and NOT

Every Boolean function can be represented as a sum of products

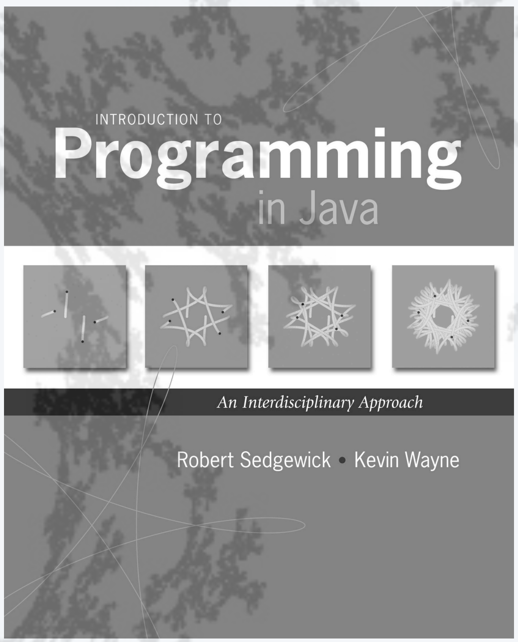
- Form an AND term for each 1 in Boolean function.
- OR all the terms together.

x	y	z	MAJ	$x'yz$	$xy'z$	xyz'	xyz	$x'yz + xy'z + xyz' + xyz = MAJ$
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 as a sum of products

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

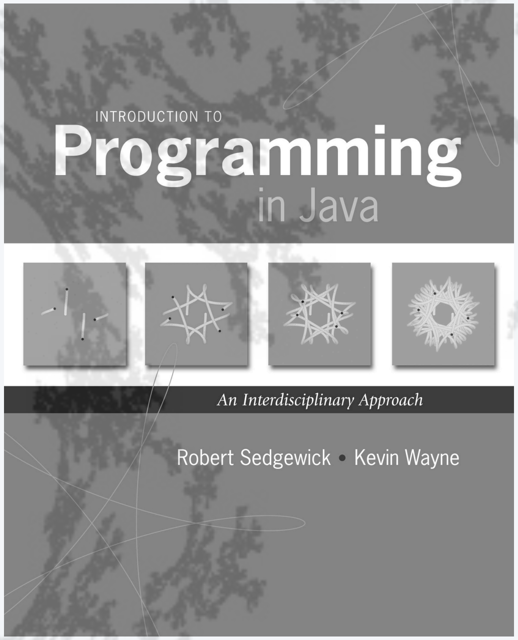
Fact. { AND, OR, NOT } is universal.



http://introcs.cs.princeton.edu

20. Combinational Circuits

- Building blocks
- **Boolean algebra**
- Digital circuits
- Adder



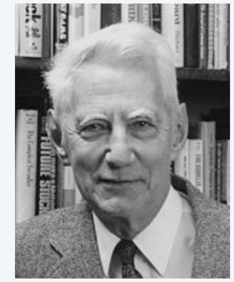
http://introcs.cs.princeton.edu

20. Combinational Circuits

- Building blocks
- Boolean algebra
- **Digital circuits**
- Adder

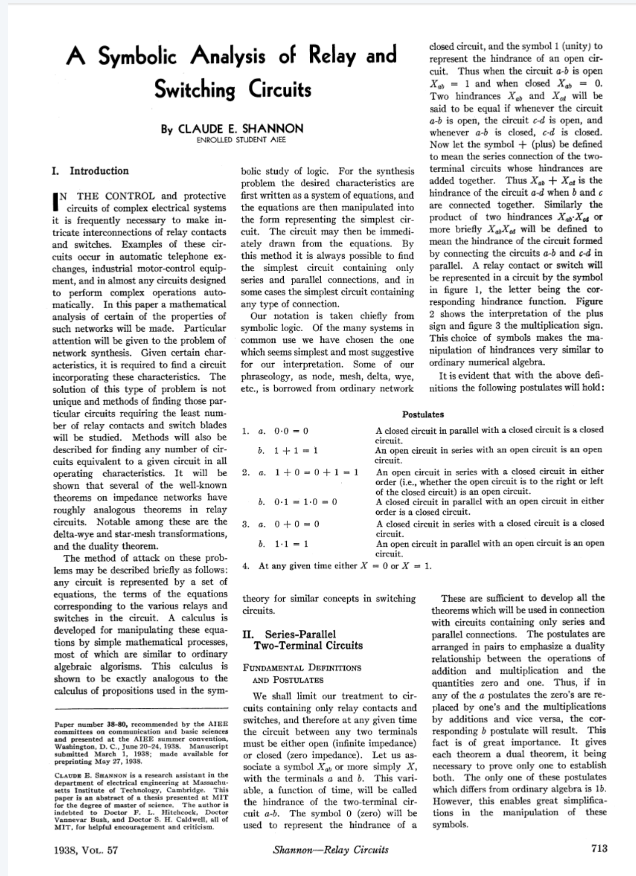
A basis for digital devices

Claude Shannon connected circuit design with *boolean algebra* in 1937.



Claude Shannon
1916–2001

> *"Possibly the most important, and also the most famous, master's thesis of the [20th] century."*
>
> *– Howard Gardner*



A Symbolic Analysis of Relay and Switching Circuits

By CLAUDE E. SHANNON
ENROLLED STUDENT AIEE

I. Introduction

IN THE CONTROL and protective circuits of complex electrical systems it is frequently necessary to make intricate interconnections of relay contacts and switches. Examples of these circuits occur in automatic telephone exchanges, industrial motor-control equipment, and in almost any circuits designed to perform complex operations automatically. In this paper a mathematical analysis of certain of the properties of such networks will be made. Particular attention will be given to the problem of network synthesis. Given certain characteristics, it is required to find a circuit incorporating these characteristics. The solution of this type of problem is not unique and methods of finding those particular circuits requiring the least number of relay contacts and switch blades will be studied. Methods will also be described for finding any number of circuits equivalent to a given circuit in all operating characteristics. It will be shown that several of the well-known theorems on impedance networks have roughly analogous theorems in relay circuits. Notable among these are the delta-wye and star-mesh transformations, and the duality theorem.

The method of attack on these problems may be described briefly as follows: any circuit is represented by a set of equations, the terms of the equations corresponding to the various relays and switches in the circuit. A calculus is developed for manipulating these equations by simple mathematical processes, most of which are similar to ordinary algebraic algorisms. This calculus is shown to be exactly analogous to the calculus of propositions used in the symbolic study of logic. For the synthesis problem the desired characteristics are first written as a system of equations, and the equations are then manipulated into the form representing the simplest circuit. The circuit may then be immediately drawn from the equations. By this method it is always possible to find the simplest circuit containing only series and parallel connections, and in some cases the simplest circuit containing any type of connection.

Our notation is taken chiefly from symbolic logic. Of the many systems in common use we have chosen the one which seems simplest and most suggestive for our interpretation. Some of our phraseology, as node, mesh, delta, wye, etc., is borrowed from ordinary network theory for similar concepts in switching circuits.

II. Series-Parallel Two-Terminal Circuits

Fundamental Definitions and Postulates

We shall limit our treatment to circuits containing only relay contacts and switches, and therefore at any given time the circuit between any two terminals must be either open (infinite impedance) or closed (zero impedance). Let us associate a symbol X_{ab} or more simply X, with the terminals a and b. This variable, a function of time, will be called the hindrance of the two-terminal circuit a-b. The symbol 0 (zero) will be used to represent the hindrance of a closed circuit, and the symbol 1 (unity) to represent the hindrance of an open circuit. Thus when the circuit a-b is open $X_{ab} = 1$ and when closed $X_{ab} = 0$. Two hindrances X_{ab} and X_{cd} will be said to be equal if whenever the circuit a-b is open, the circuit c-d is open, and whenever a-b is closed, c-d is closed. Now let the symbol $+$ (plus) be defined to mean the series connection of the two-terminal circuits whose hindrances are added together. Thus $X_{ab} + X_{cd}$ is the hindrance of the circuit a-d when b and c are connected together. Similarly the product of two hindrances $X_{ab} \cdot X_{cd}$ or more briefly $X_{ab}X_{cd}$ will be defined to mean the hindrance of the circuit formed by connecting the circuits a-b and c-d in parallel. A relay contact or switch will be represented in a circuit by the symbol in figure 1, the letter being the corresponding hindrance function. Figure 2 shows the interpretation of the plus sign and figure 3 the multiplication sign. This choice of symbols makes the manipulation of hindrances very similar to ordinary numerical algebra.

It is evident that with the above definitions the following postulates will hold:

Postulates

1.	*a.* $0 \cdot 0 = 0$	A closed circuit in parallel with a closed circuit is a closed circuit.
	b. $1 + 1 = 1$	An open circuit in series with an open circuit is an open circuit.
2.	*a.* $1 + 0 = 0 + 1 = 1$	An open circuit in series with a closed circuit in either order (i.e., whether the open circuit is to the right or left of the closed circuit) is an open circuit.
	b. $0 \cdot 1 = 1 \cdot 0 = 0$	A closed circuit in parallel with an open circuit in either order is a closed circuit.
3.	*a.* $0 + 0 = 0$	A closed circuit in series with a closed circuit is a closed circuit.
	b. $1 \cdot 1 = 1$	An open circuit in parallel with an open circuit is an open circuit.
4.	At any given time either $X = 0$ or $X = 1$.	

These are sufficient to develop all the theorems which will be used in connection with circuits containing only series and parallel connections. The postulates are arranged in pairs to emphasize a duality relationship between the operations of addition and multiplication and the quantities zero and one. Thus, if in any of the a postulates the zero's are replaced by one's and the multiplications by additions and vice versa, the corresponding b postulate will result. This fact is of great importance. It gives each theorem a dual theorem, it being necessary to prove only one to establish both. The only one of these postulates which differs from ordinary algebra is 1b. However, this enables great simplifications in the manipulation of these symbols.

Paper number **38-80**, recommended by the AIEE committees on communication and basic sciences and presented at the AIEE summer convention, Washington, D. C., June 20–24, 1938. Manuscript submitted March 1, 1938; made available for preprinting May 27, 1938.

Claude E. Shannon is a research assistant in the department of electrical engineering at Massachusetts Institute of Technology, Cambridge. This paper is an abstract of a thesis presented at MIT for the degree of master of science. The author is indebted to Doctor F. L. Hitchcock, Doctor Vannevar Bush, and Doctor S. H. Caldwell, all of MIT, for helpful encouragement and criticism.

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

A second level of abstraction: logic gates

boolean function	notation	proof
NOT	x'	1 iff x is 0
NOR	$(x + y)'$	1 iff x and y are both 0
OR	$x + y$	$x+y = ((x + y)')'$
AND	xy	$xy = (x' + y')'$

truth table

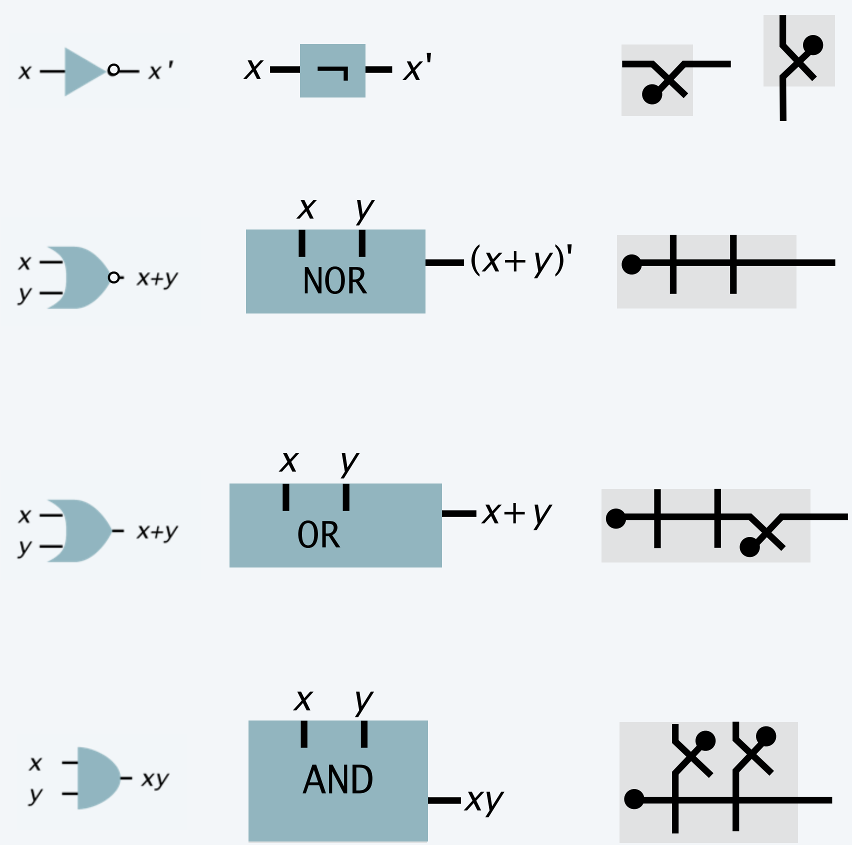
x	x'
0	1
1	0

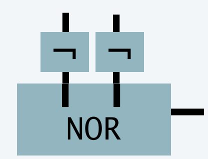
x	y	*NOR*
0	0	1
0	1	0
1	0	0
1	1	0

x	y	*OR*
0	0	0
0	1	1
1	0	1
1	1	1

x	y	*AND*
0	0	0
0	1	0
1	0	0
1	1	1

classic symbol — **our symbol** — **under the cover circuit (gate)**

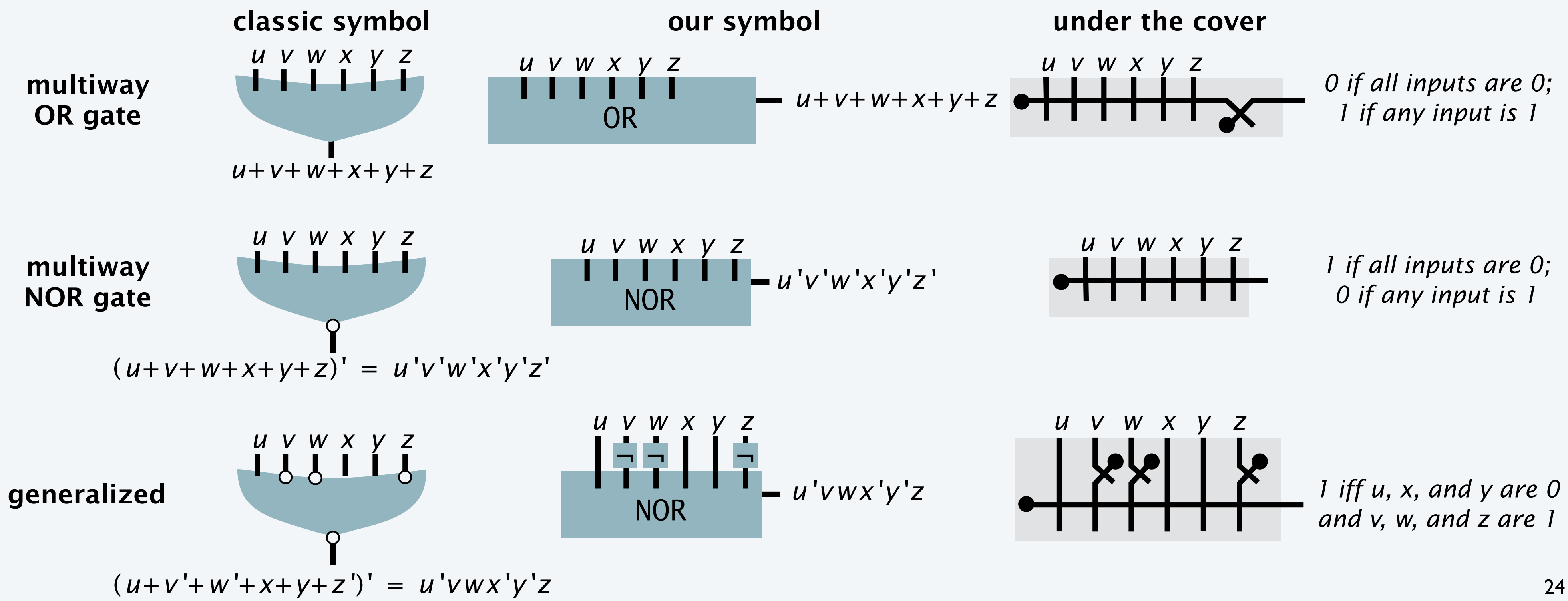




Gates with arbitrarily many inputs

Multiway gates.

- OR: 1 if any input is 1; 0 if all inputs are 0.
- NOR: 0 if any input is 1; 1 if all inputs are 0.
- Generalized: Negate some inputs.



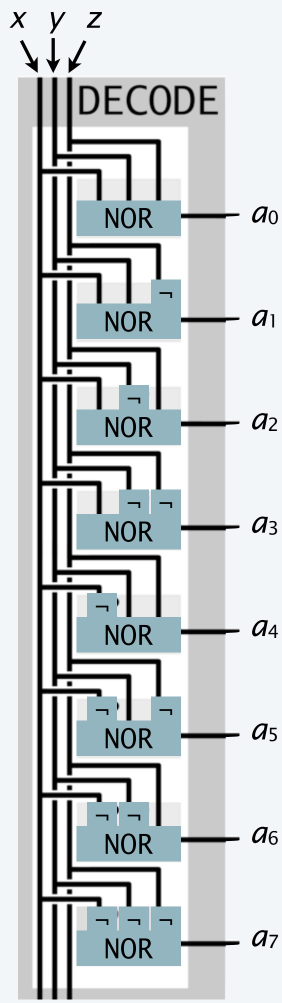
Generalized NOR gate application: Decoder

A *decoder* uses a binary address to switch on a single output line

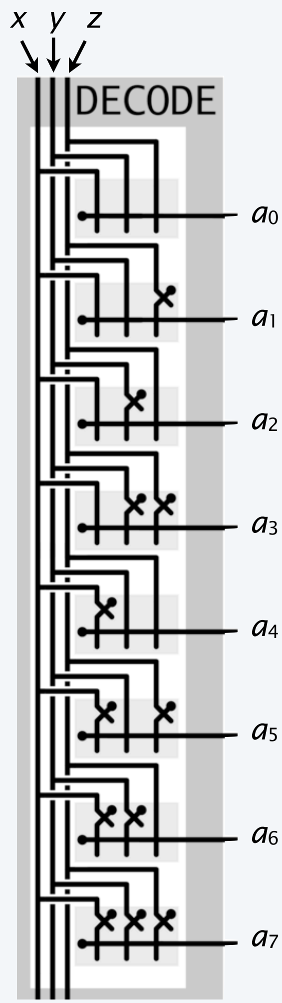
- n address inputs, 2^n outputs.
- Uses all 2^n different generalized *NOR* gates.
- Addressed output line is 1; all others are 0.

x	y	z	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
			$x'y'z'$	$x'y'z$	$x'yz'$	$x'yz$	$xy'z'$	$xy'z$	xyz'	xyz
			$(x+y+z)'$	$(x+y+z')'$	$(x+y'+z)'$	$(x+y'+z')'$	$(x'+y+z)'$	$(x+y'+z)'$	$(x'+y'+z)'$	$(x'+y'+z')'$
0	0	0	**1**	0	0	0	0	0	0	0
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	0
0	1	1	0	0	0	**1**	0	0	0	0
1	0	0	0	0	0	0	**1**	0	0	0
1	0	1	0	0	0	0	0	**1**	0	0
1	1	0	0	0	0	0	0	0	**1**	0
1	1	1	0	0	0	0	0	0	0	**1**

Next. Circuits for *any* boolean function.

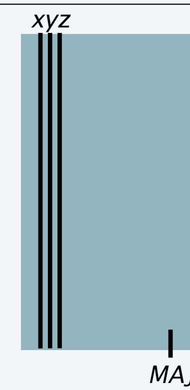


3-bit decoder



under the covers

Creating a digital circuit that computes a boolean function: majority



Use the truth table

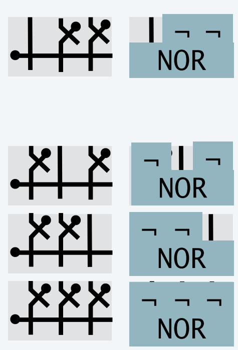
- Identify rows where the function is 1.
- Use a generalized NOR gate for each.
- OR the results together.

Example 1: Majority function

x	y	z	MAJ
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

$MAJ = x'yz + xy'z + xyz' + xyz$

generalized NORs
implement AND terms
in sum-of -products

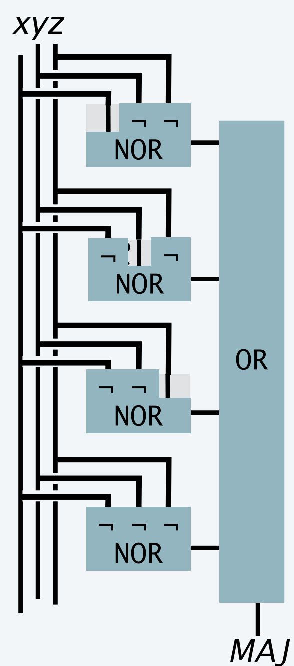


$x'yz = (x + y' + z')'$

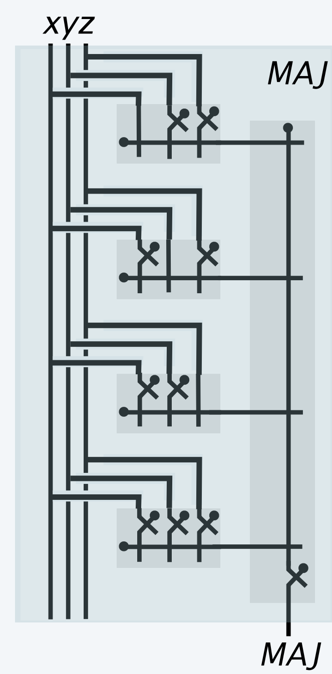
$xy'z = (x' + y + z')'$

$xyz' = (x' + y' + z)'$

$xyz = (x' + y' + z')'$



majority circuit

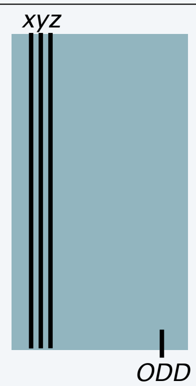


under the covers

Creating a digital circuit that computes a boolean function: odd parity

Use the truth table

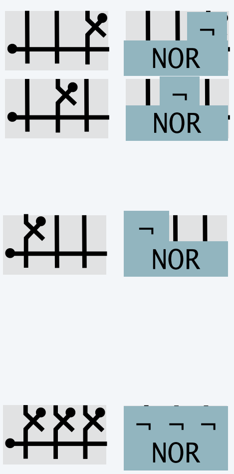
- Identify rows where the function is 1.
- Use a generalized NOR gate for each.
- OR the results together.



Example 2: Odd parity function

x	y	z	ODD
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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

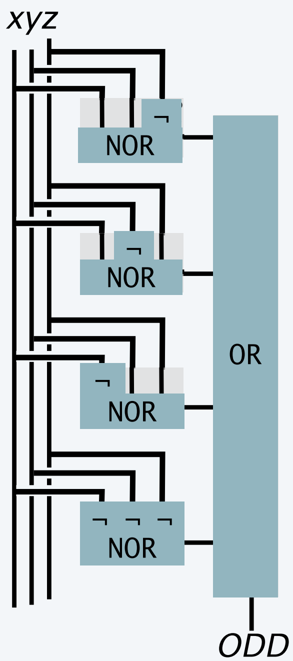


$x'y'z = (x + y + z')'$

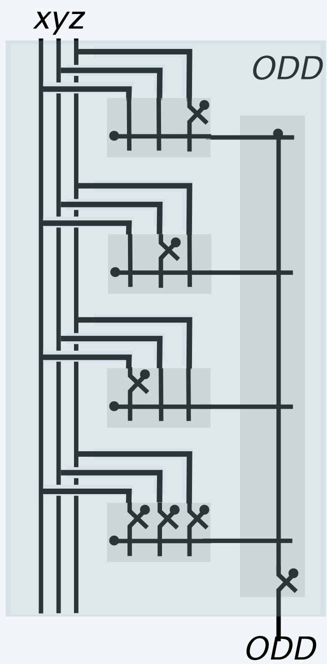
$x'yz' = (x' + y' + z)'$

$xy'z' = (x' + y + z)'$

$xyz = (x' + y' + z')'$



odd parity circuit



under the covers

Combinational circuit design: Summary

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

Ingredients

- OR gates.
- NOT gates.
- NOR gates.
- Wire.

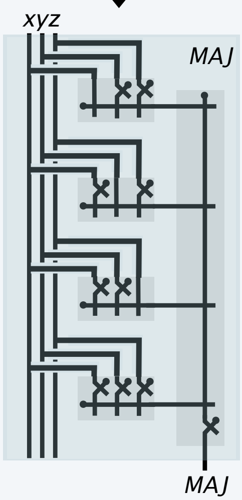
Method

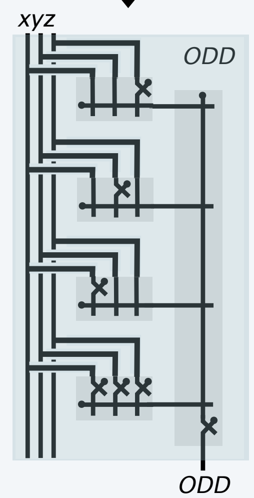
- Step 1: Represent input and output with Boolean variables.
- Step 2: Construct truth table to define the function.
- Step 3: Identify rows where the function is 1.
- Step 4: Use a generalized NOR for each and OR the results.

Bottom line (profound idea): Yields a circuit for ANY function.
Caveat (stay tuned): Circuit might be huge.

x	*y*	*z*	*MAJ*
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

x	*y*	*z*	*ODD*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1





TEQ on combinational circuit design

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

← not really a TEQ because we usually frame these as multiple choice

TEQ on combinational circuit design

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

← not really a TEQ because we usually frame these as multiple choice

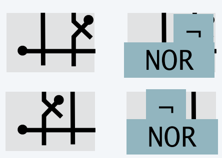
A. Use the truth table

- Identify rows where the function is 1.
- Use a generalized NOR gate for each.
- OR the results together.

XOR function

x	*y*	*XOR*
0	0	0
0	1	1
1	0	1
1	1	0

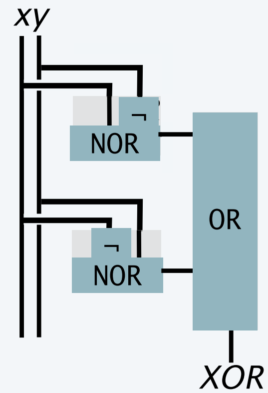
$XOR = x'y + xy'$



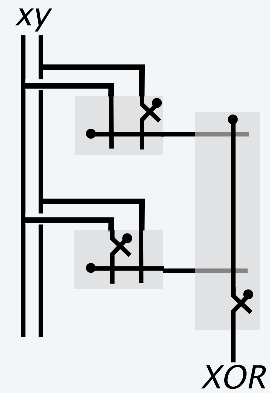
$x'y = (x + y')'$

$xy' = (x' + y)'$

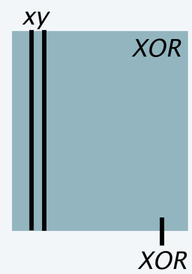
circuit (gates)



circuit



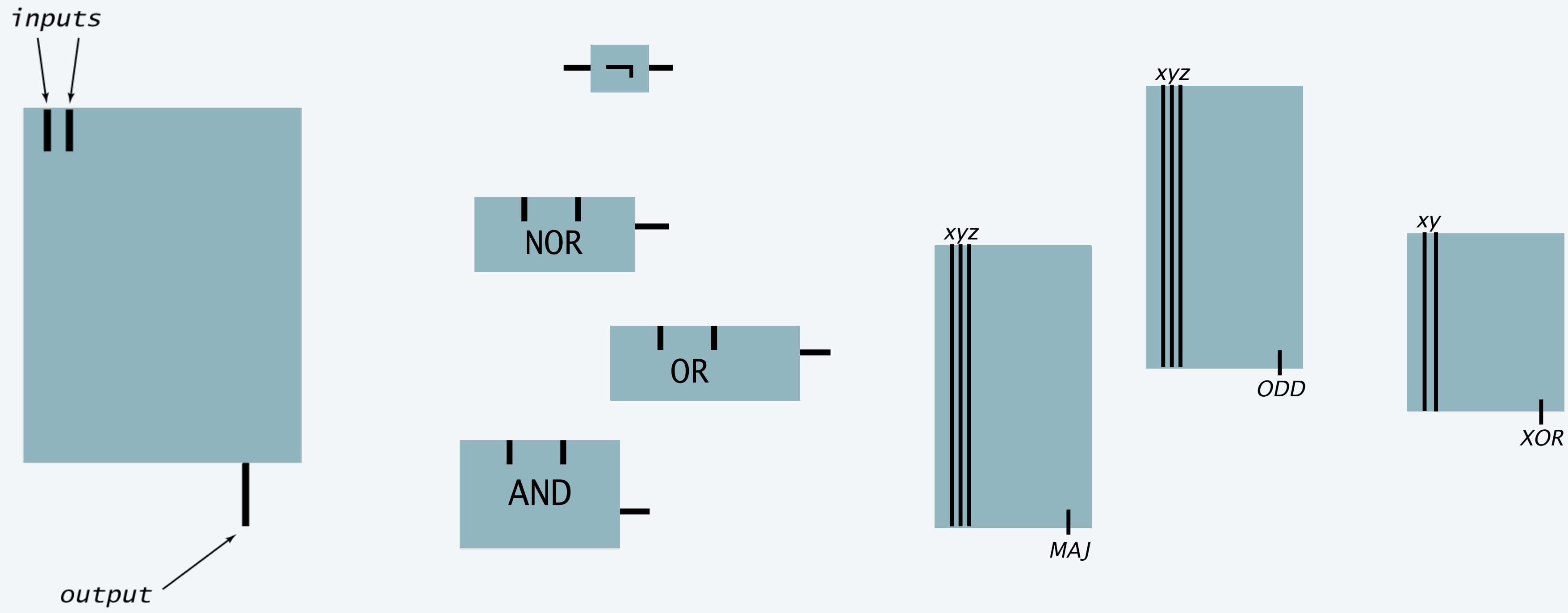
interface



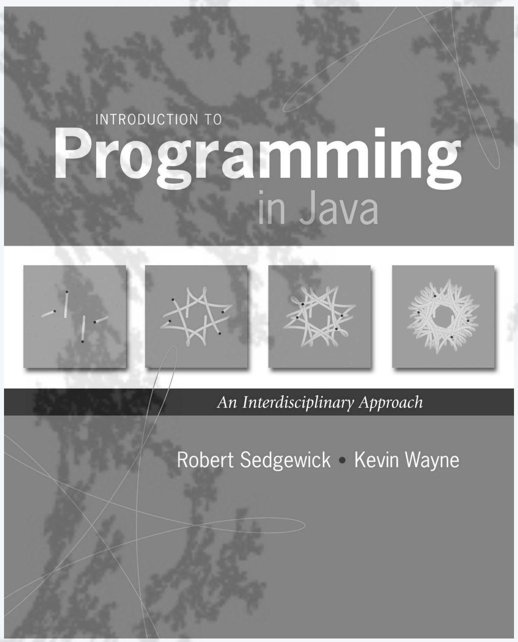
Encapsulation

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 and outputs is the *API*.
- We control complexity by *encapsulating* circuits as we do with *ADTs*.



COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

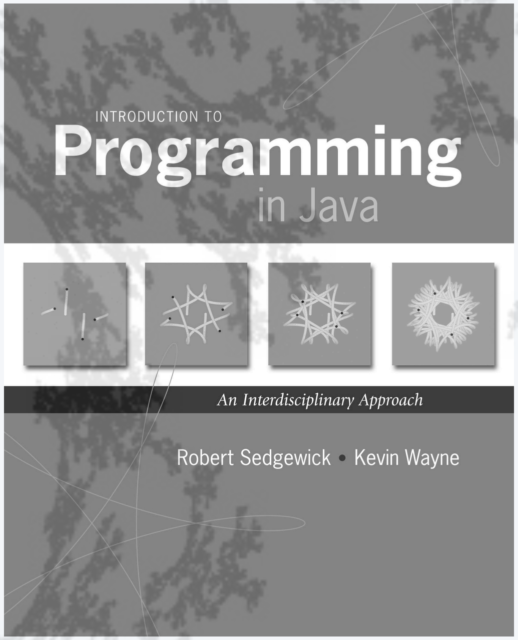


http://introcs.cs.princeton.edu

20. Combinational Circuits

- Building blocks
- Boolean algebra
- **Digital circuits**
- Adder

COMPUTER SCIENCE
SEDGEWICK / WAYNE



http://introcs.cs.princeton.edu

20. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder

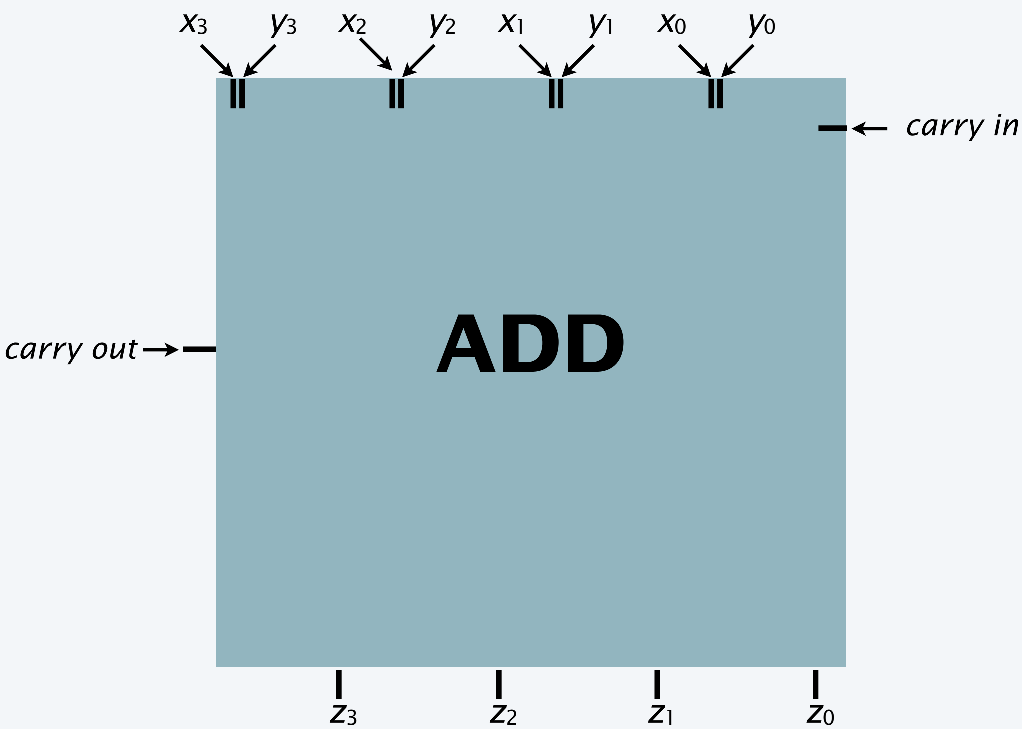
Let's make an adder circuit

Goal. $x + y = z$ for 4-bit binary integers. ← same ideas scale to 64-bit adder in your computer

- 4-bit adder: 9 inputs, 5 outputs.
- Each output is a boolean function of the inputs.

1	0	0	1	
	2	4	7	7
+	9	5	1	9
1	1	9	9	6

	1	1	0	0
	0	0	1	0
+	0	1	1	1
	1	0	0	1



carry out →	c_4	c_3	c_2	c_1	c_0	← carry in
		x_3	x_2	x_1	x_0	
	+	y_3	y_2	y_1	y_0	
		z_3	z_2	z_1	z_0	

Let's make an adder circuit

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

Strawman solution: Build truth tables for each output bit.

c_4	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

4-bit adder truth table

c_0	x_3	x_2	x_1	x_0	y_3	y_2	y_1	y_0	c_4	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	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	1
						. . .							
1	1	1	1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1

$2^{8+1} = 512$ rows!

Q. Why is this a bad idea?

A. 128-bit adder: 2^{256+1} rows >> # electrons in universe!

Let's make an adder circuit

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

c_4	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

carry 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

sum bit

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

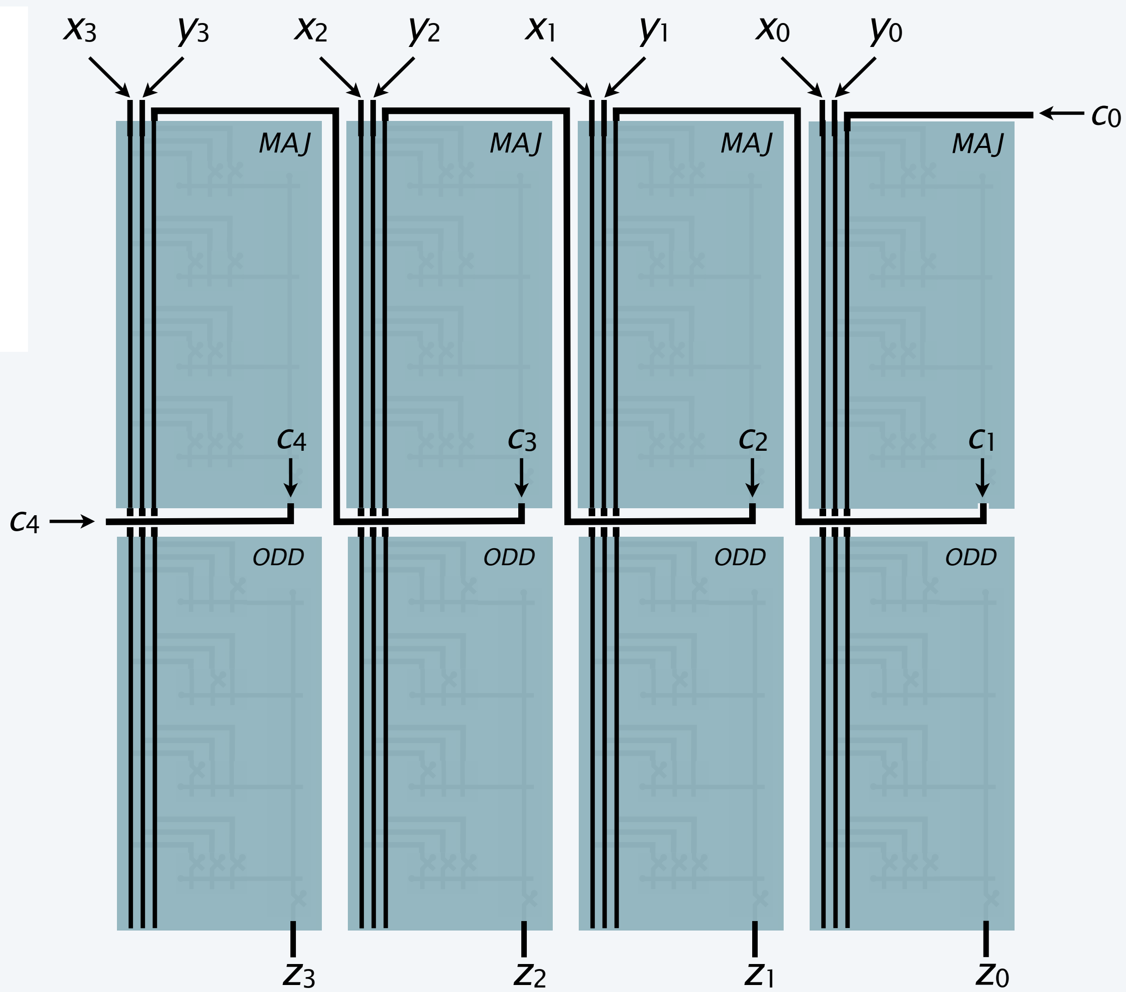
Let's make an adder circuit

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

Do one bit at a time.

- Use MAJ and ODD circuits.
- Chain together 1-bit adders to "ripple" carries.

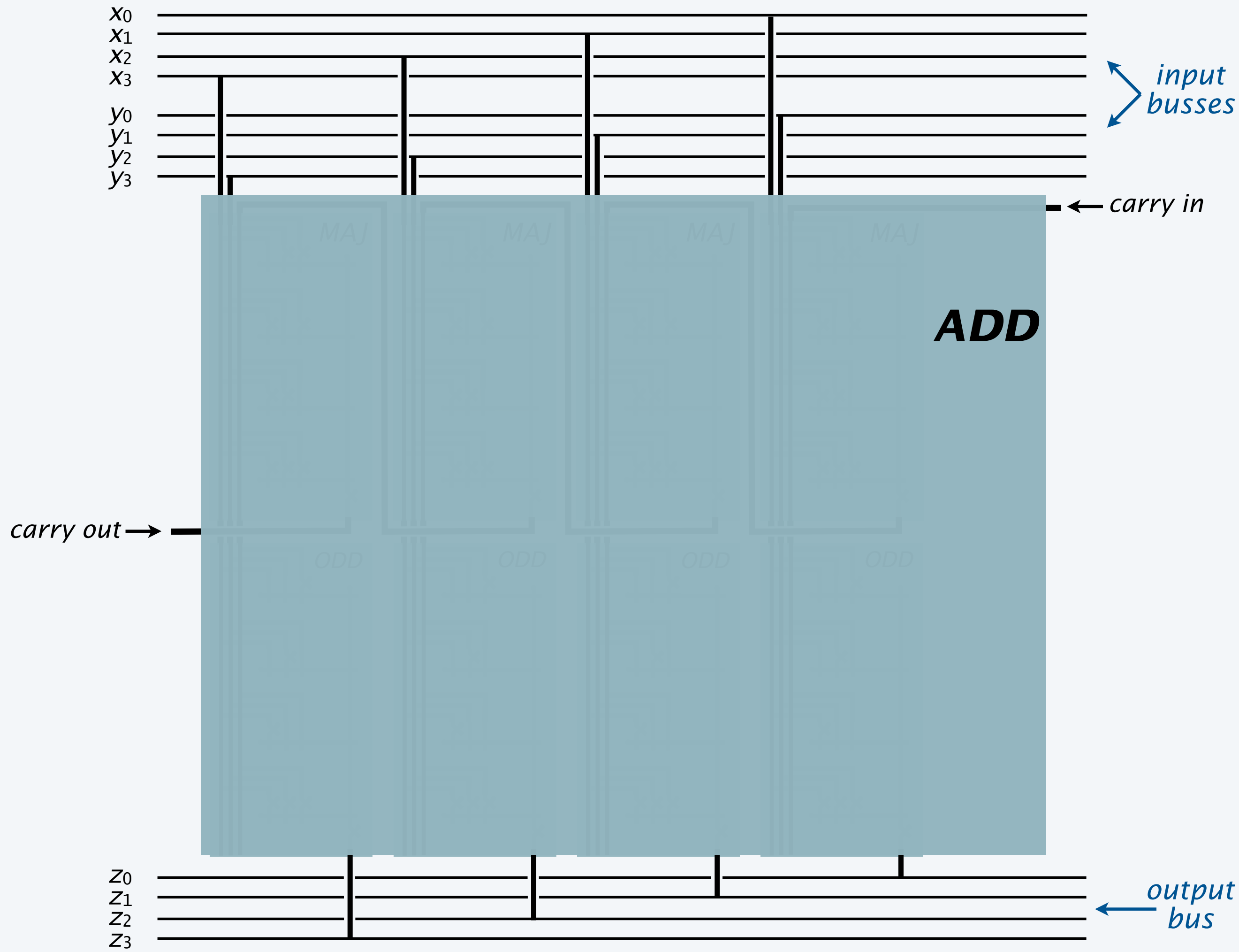
c_4	c_3	c_2	c_1	c_0
	x_3	x_2	x_1	x_0
+	y_3	y_2	y_1	y_0
	z_3	z_2	z_1	z_0



Adder interface

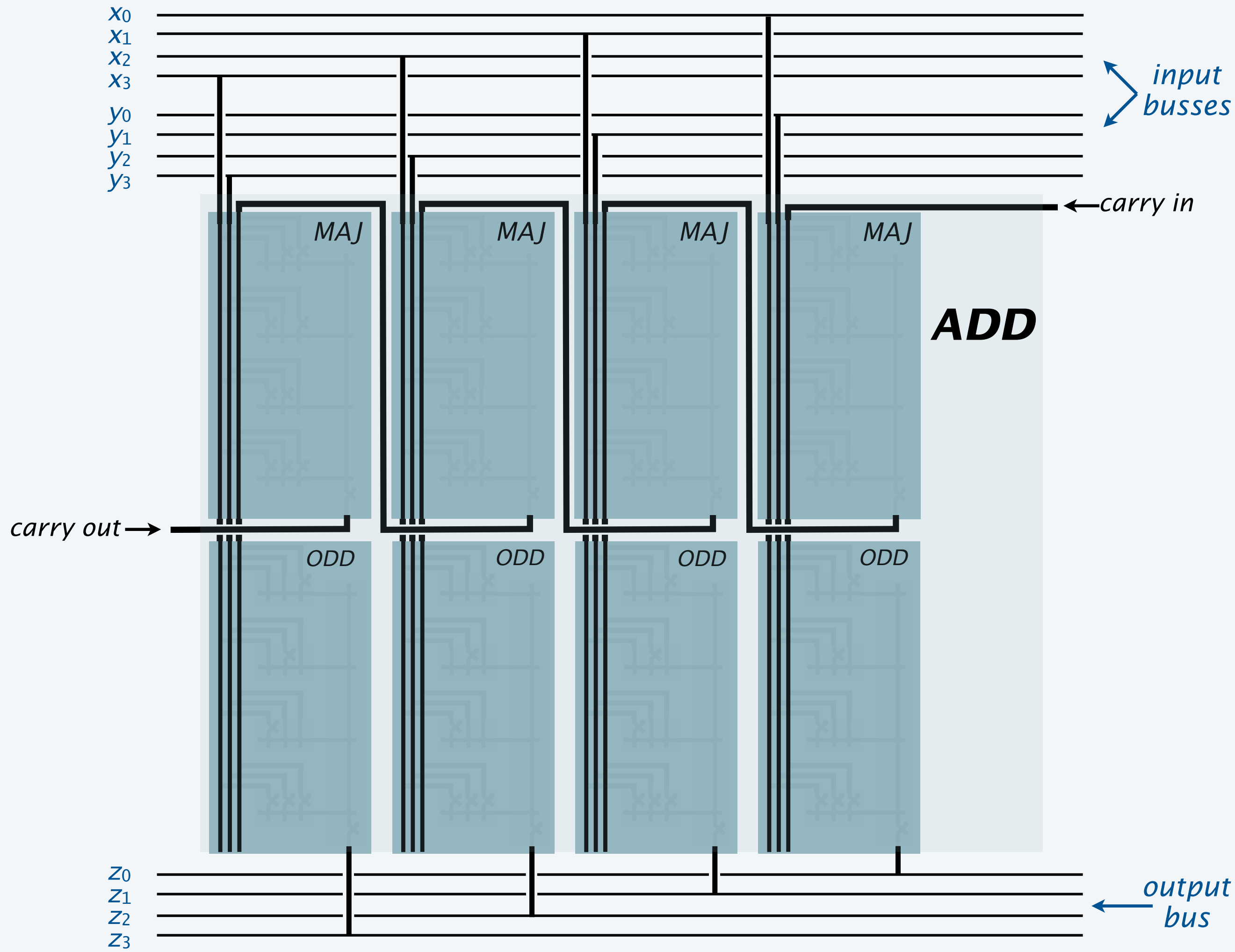
A bus is a group of wires that connect components (carrying data values).

c_4	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



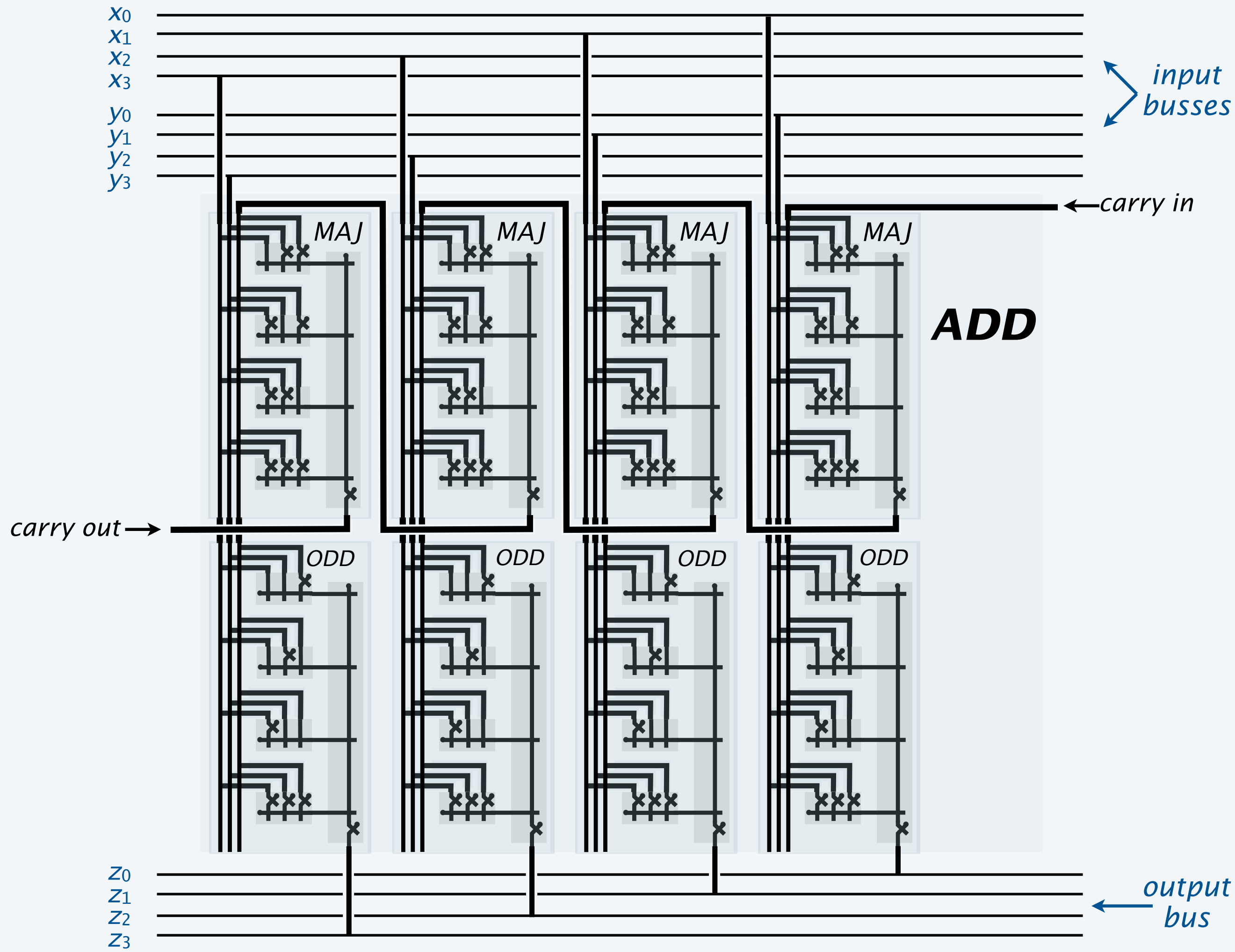
Adder component-level view

c_4	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



Adder switch-level view

c_4	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



Arithmetic and logic unit (ALU)

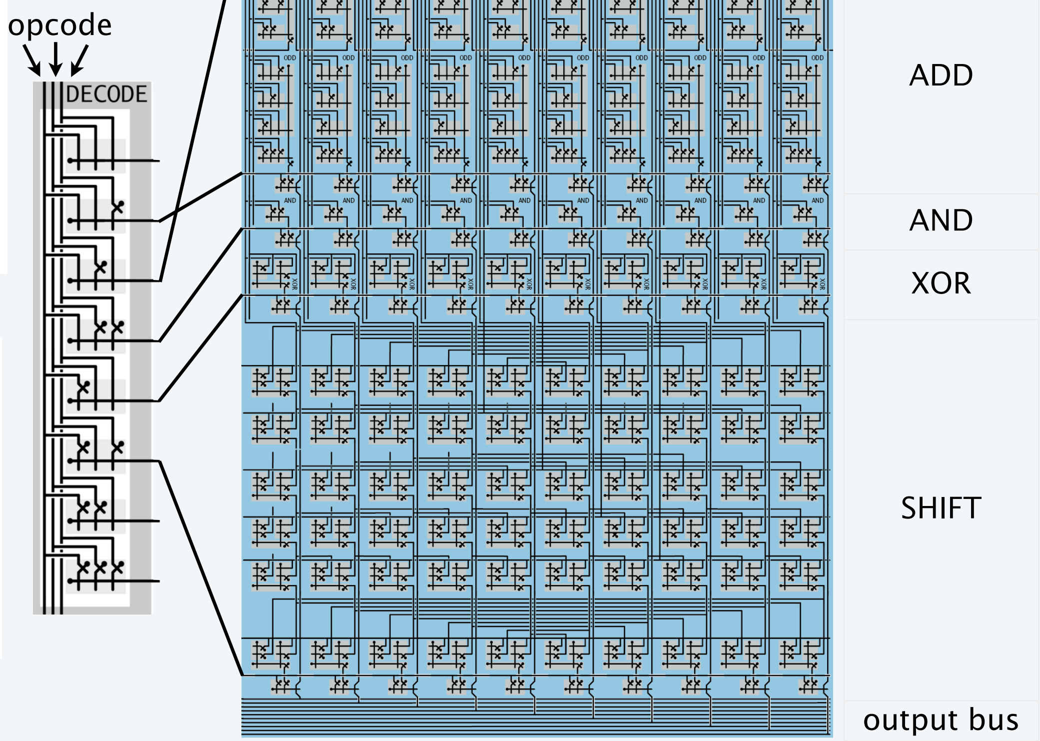
ALU: A large combinatorial circuit—the calculator at the heart of your computer

- Add $x+y$.
- Subtract (by first negating y).
- Bitwise AND (trivial).
- Bitwise XOR (TEQ).
- Shift left and right (details omitted).
- ...

Key component: A decoder!

- All circuits compute a result.
- Decoder uses opcode to select exactly one of the results for the output bus (many details omitted).

Example: tinyTOY ALU (see next lecture)



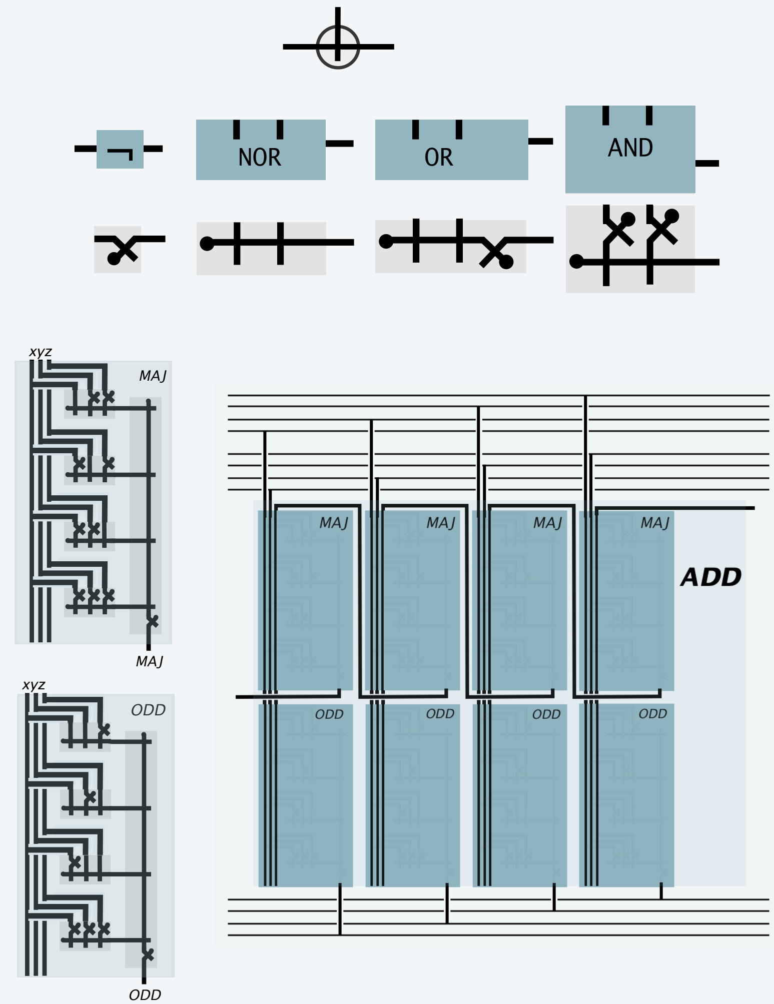
Summary

Lessons for software design apply to hardware!

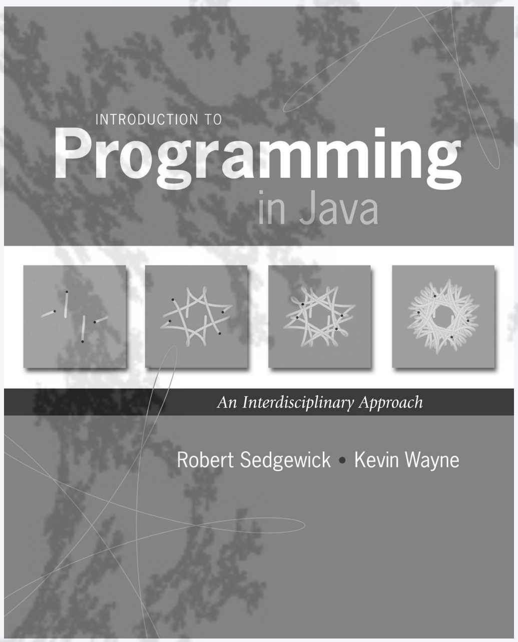
- Interface describes behavior of circuit.
- Implementation gives details of how to build it.
- Boolean logic gives understanding of behavior.

Layers of abstraction apply with a vengeance!

- On/off.
- Controlled switch. [relay, pass transistor]
- Gates. [NOT, NOR, OR, AND]
- Boolean functions. [MAJ, ODD]
- Adder.
- ...
- ALU.
- ...
- TOY machine (stay tuned).
- Your computer.



COMPUTER SCIENCE
S E D G E W I C K / W A Y N E

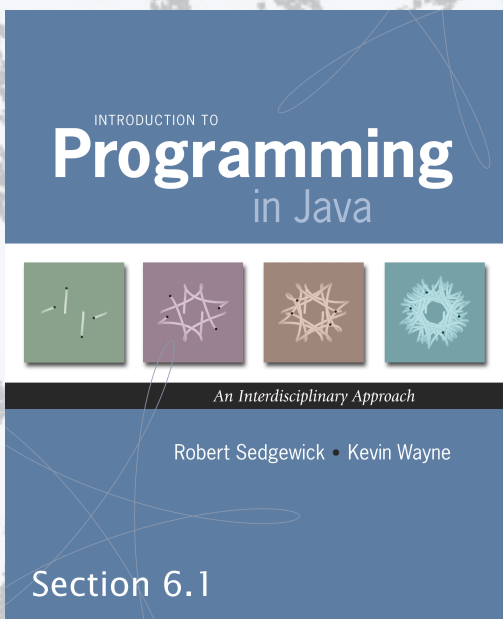


http://introcs.cs.princeton.edu

20. Combinational Circuits

- Building blocks
- Boolean algebra
- Digital circuits
- Adder

COMPUTER SCIENCE
S E D G E W I C K / W A Y N E



http://introcs.cs.princeton.edu

20. Combinational Circuits