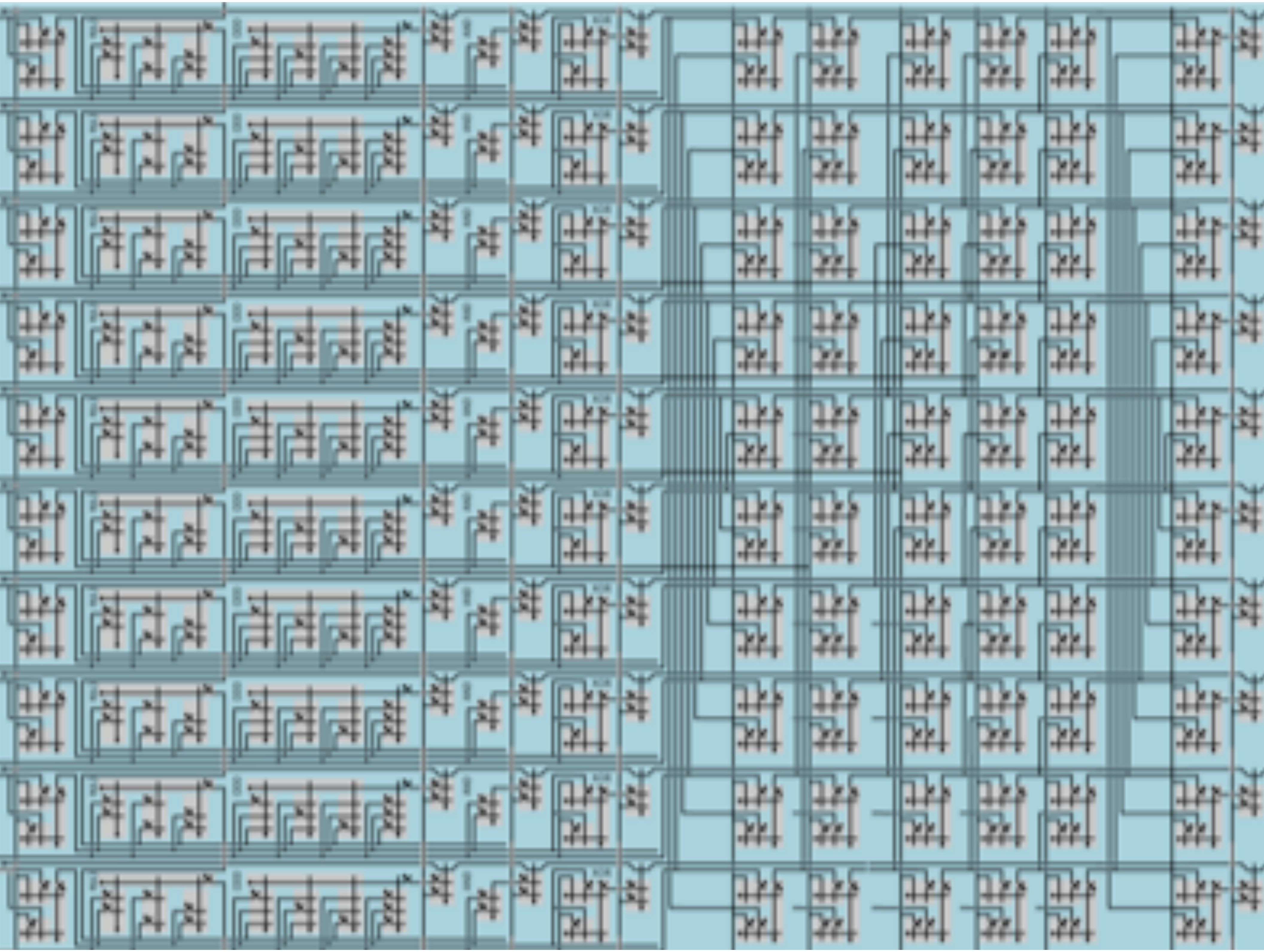


Combinational Circuits

Q. What is a combinational circuit?

A. Digital: signals are 0 or 1. ← analog circuits: signals vary continuously

A. No feedback: no loops. ← sequential circuits: loops allowed (stay tuned)

Q. Why combinational circuits?

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

Basic abstractions.

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

Applications. Cell phone, iPod, antilock brakes, microprocessors, …



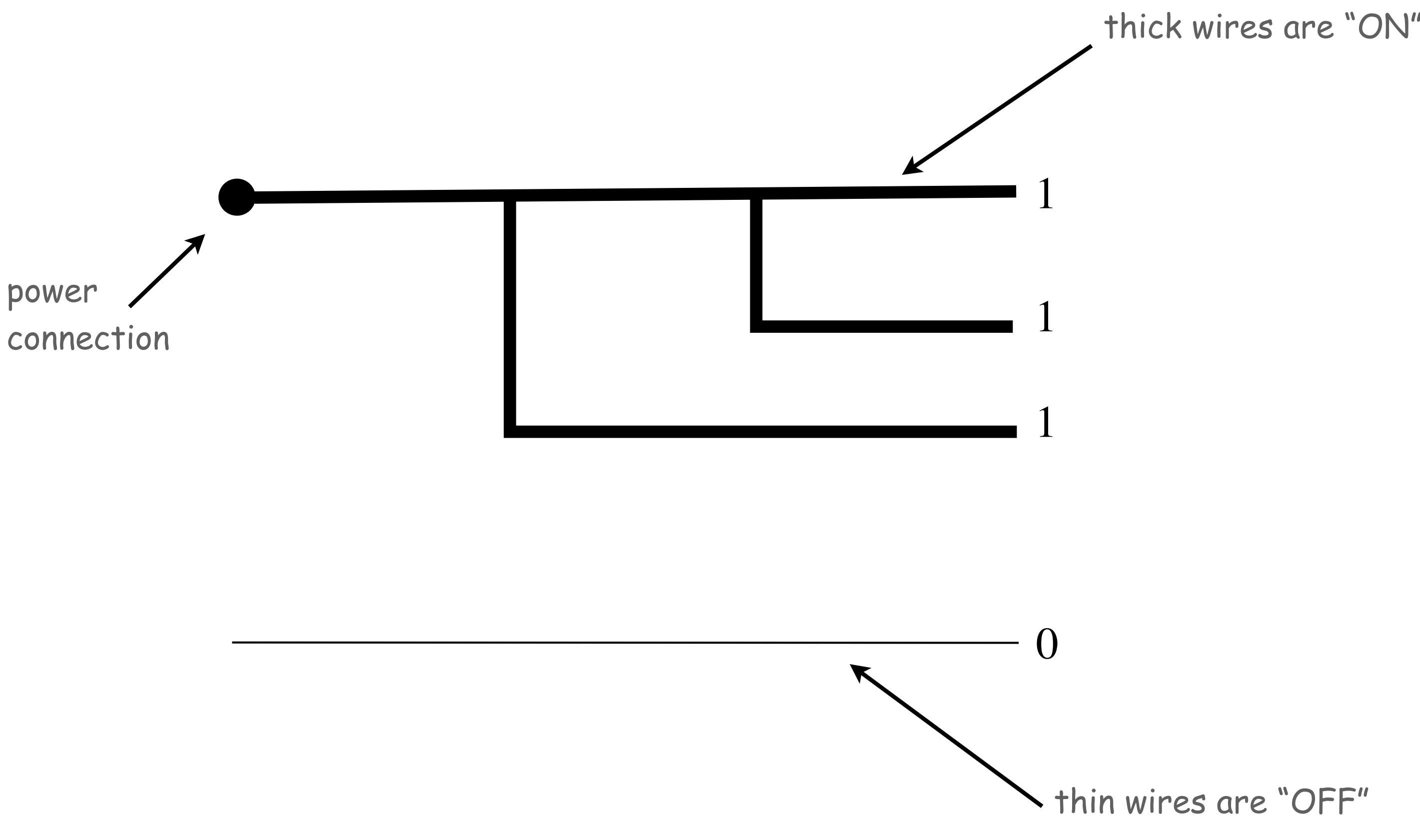


Building Blocks

Wires

Wires.

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



Controlled Switch

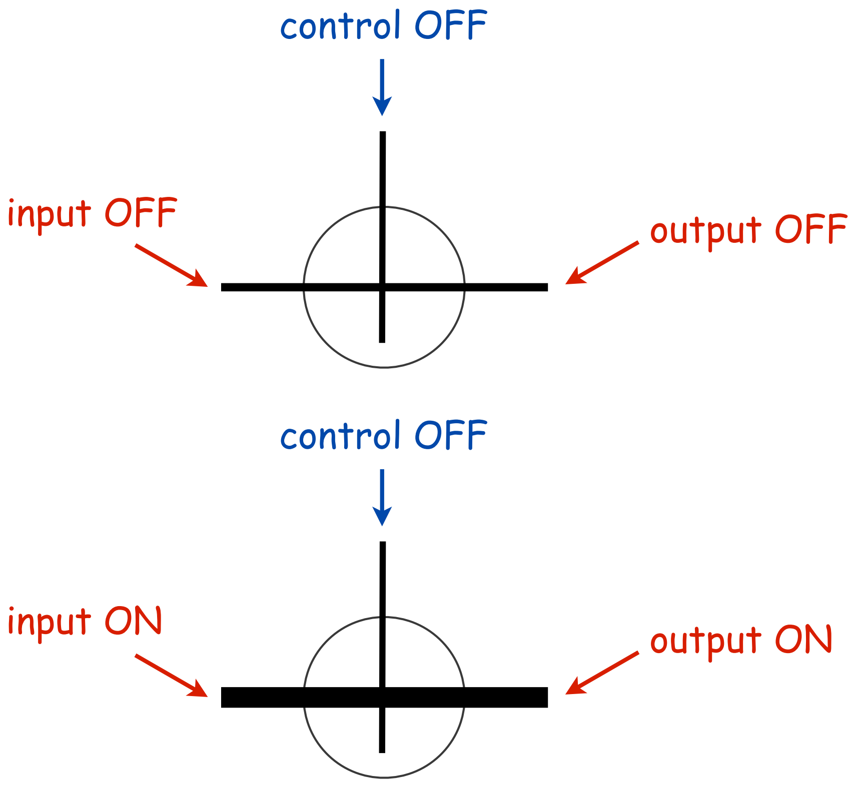
Controlled switch.

- 3 connections: input, output, control.

Controlled Switch

Controlled switch.

- 3 connections: input, output, control.
- control OFF: output is connected to input

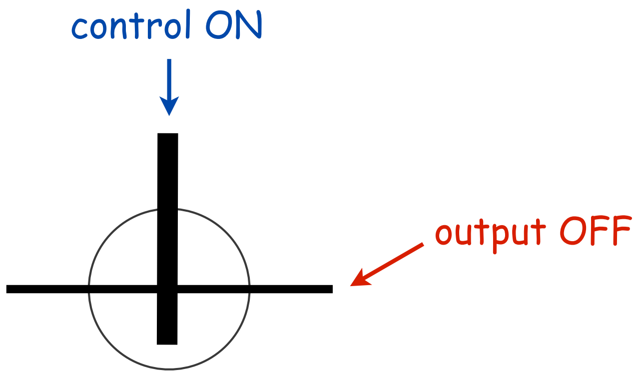


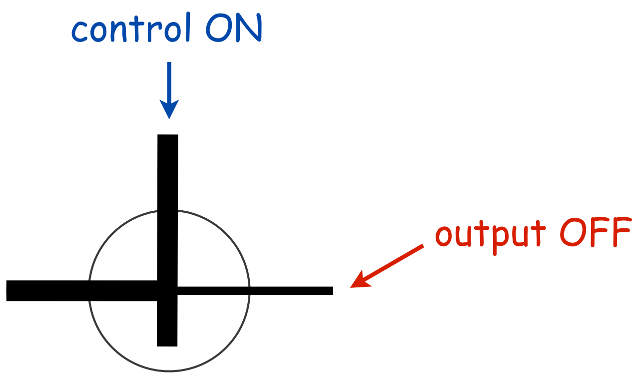
Controlled Switch

Controlled switch.

- 3 connections: input, output, control.

- control ON: output is disconnected from input

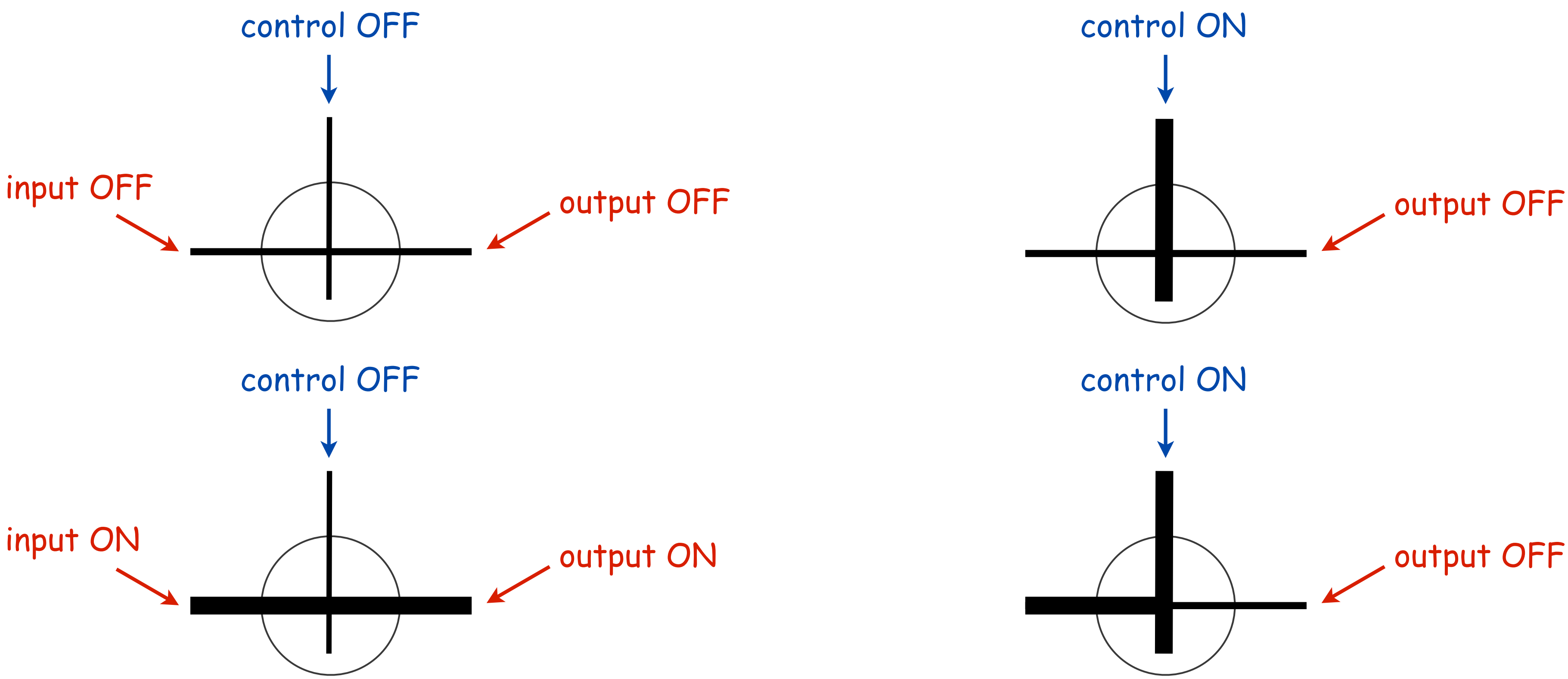




Controlled Switch

Controlled switch.

- 3 connections: input, output, control.
- control OFF: output is connected to input
- control ON: output is disconnected from input

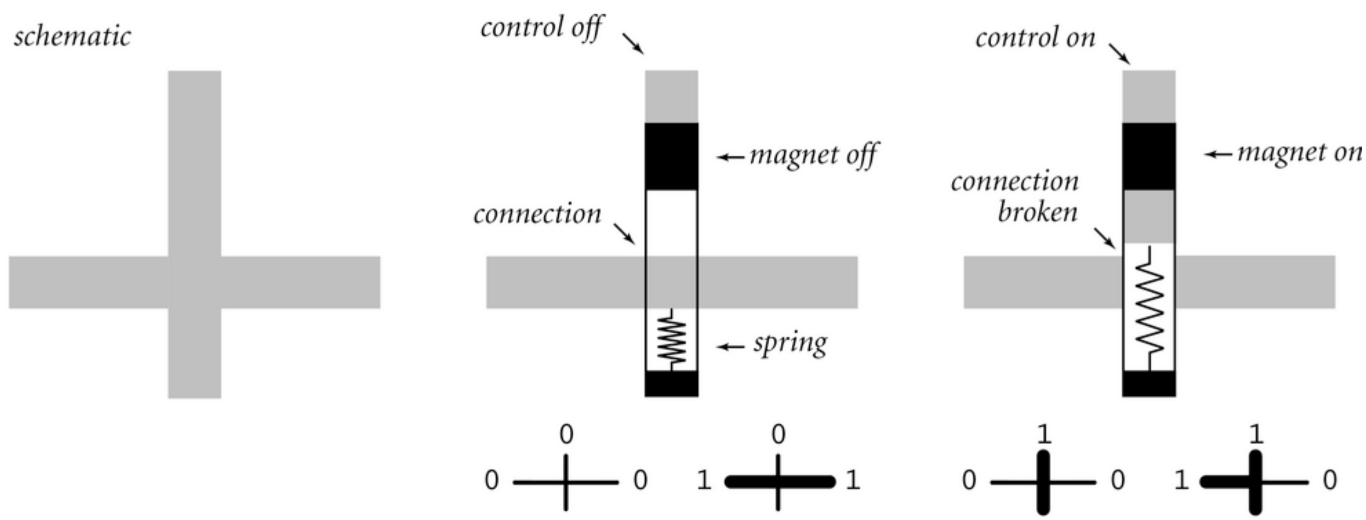


idealized model of "pass transistors" found in real integrated circuits

Implementing a Controlled Switch

Relay implementation.

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



Anatomy of a relay (controlled switch)

First Level of Abstraction

Separates physical world from 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

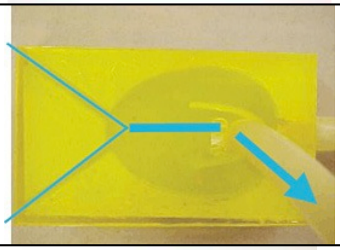
- size
- speed
- power

New technology immediately gives new computer.

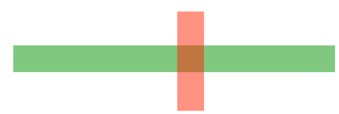
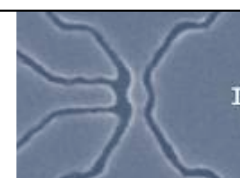
Better switch? Better computer.

Controlled Switches: A First Level of Abstraction

Some amusing attempts to prove the point:

Technology	"Information"	Switch
pneumatic	air pressure	
fluid	water pressure	
relay	electric potential	

Real-world examples that prove the point:

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

Controlled Switches: A First Level of Abstraction ?

VLSI = Very Large Scale Integration

Technology:
Deposit materials on substrate.

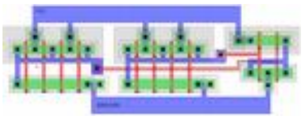
Key property:
Crossing lines are controlled switches.

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

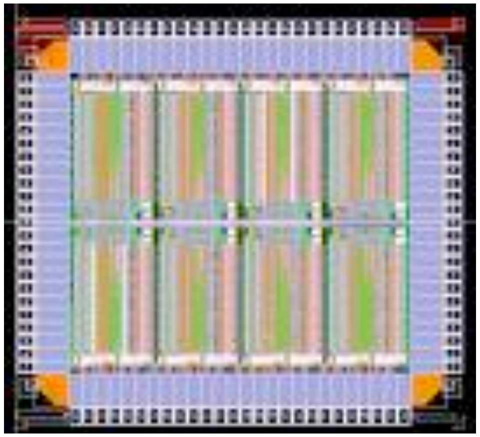
Key challenge in "abstract" world:
Understanding behavior of circuits
with billions of controlled switches

Bottom line: Circuit = Drawing (!)

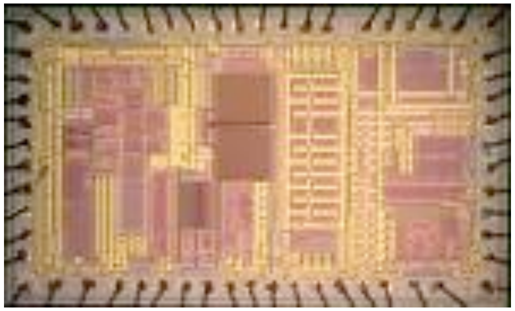
drawing



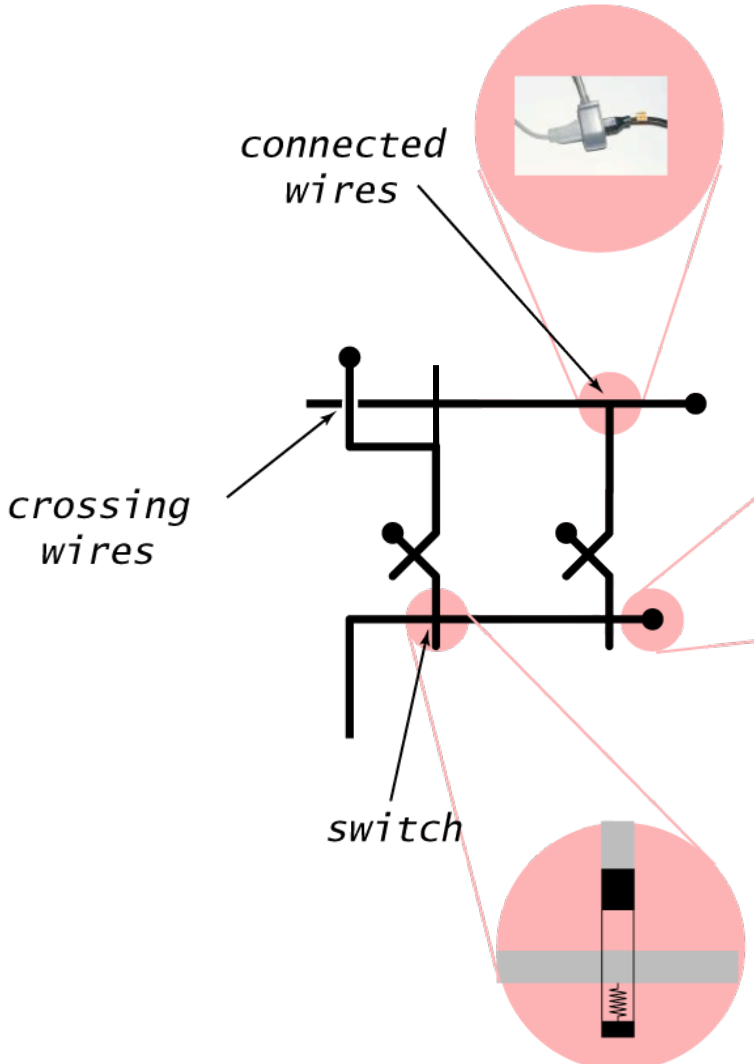
drawing

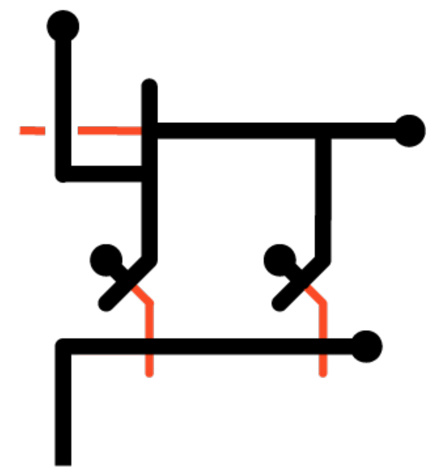


circuit



Circuit Anatomy



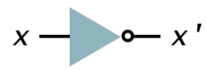


need more "levels of abstraction" to understand circuit behavior

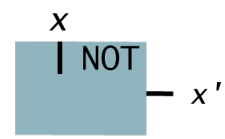
Second Level of Abstraction: Logic Gates

NOT = x'

x	NOT
0	1
1	0



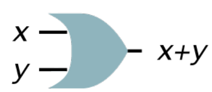
symbol



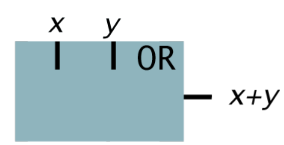
NOT gate

OR = x+y

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



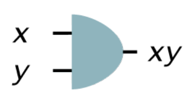
symbol



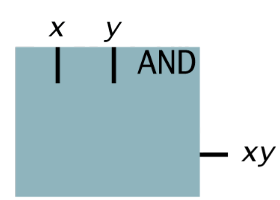
OR gate

AND = xy

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



symbol

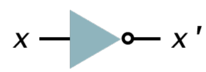


AND gate

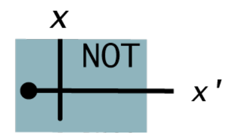
Second Level of Abstraction: Logic Gates

NOT = x'

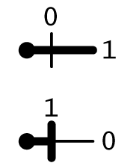
x	NOT
0	1
1	0



symbol

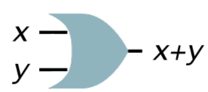


NOT gate

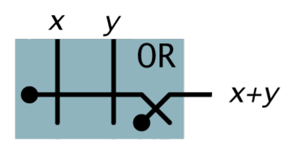


OR = x+y

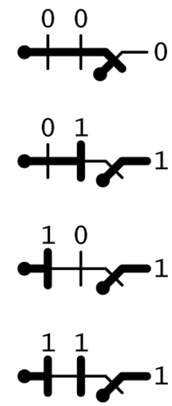
x y	OR
0 0	0
0 1	1
1 0	1
1 1	1



symbol

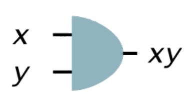


OR gate

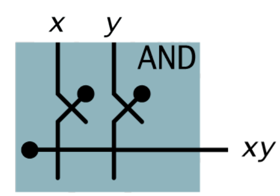


AND = xy

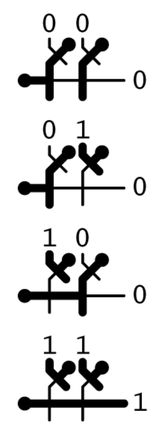
x y	AND
0 0	0
0 1	0
1 0	0
1 1	1



symbol



AND gate

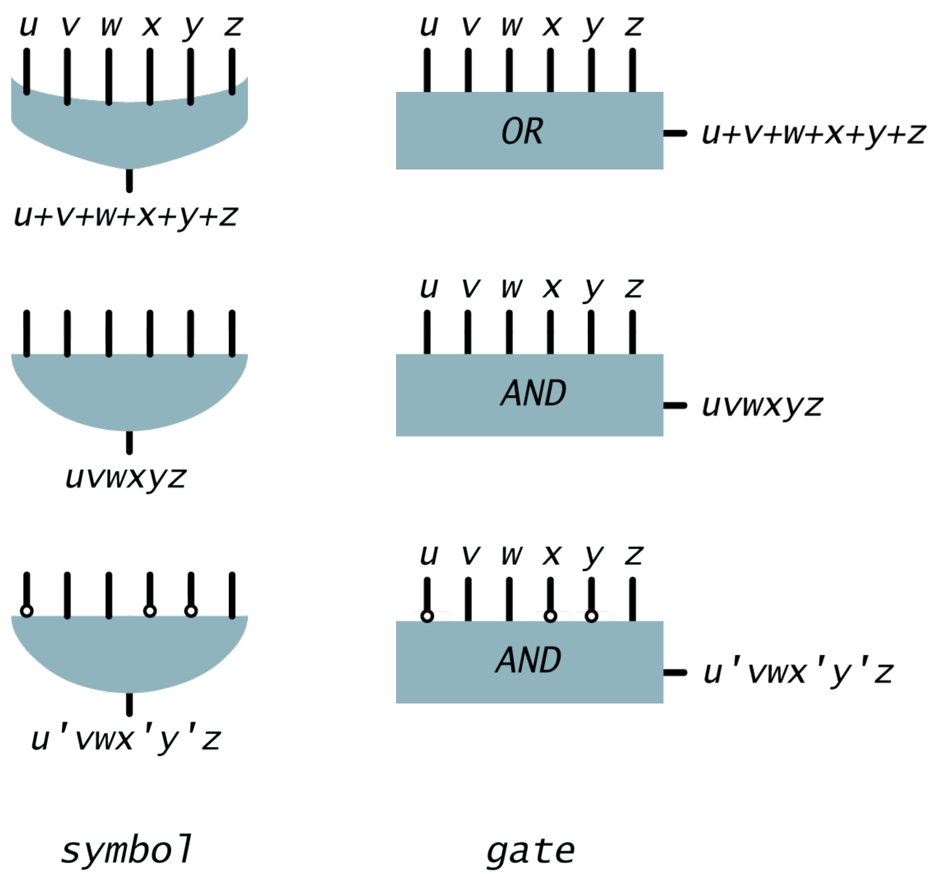


implementations with switches

Multiway Gates

Multiway gates.

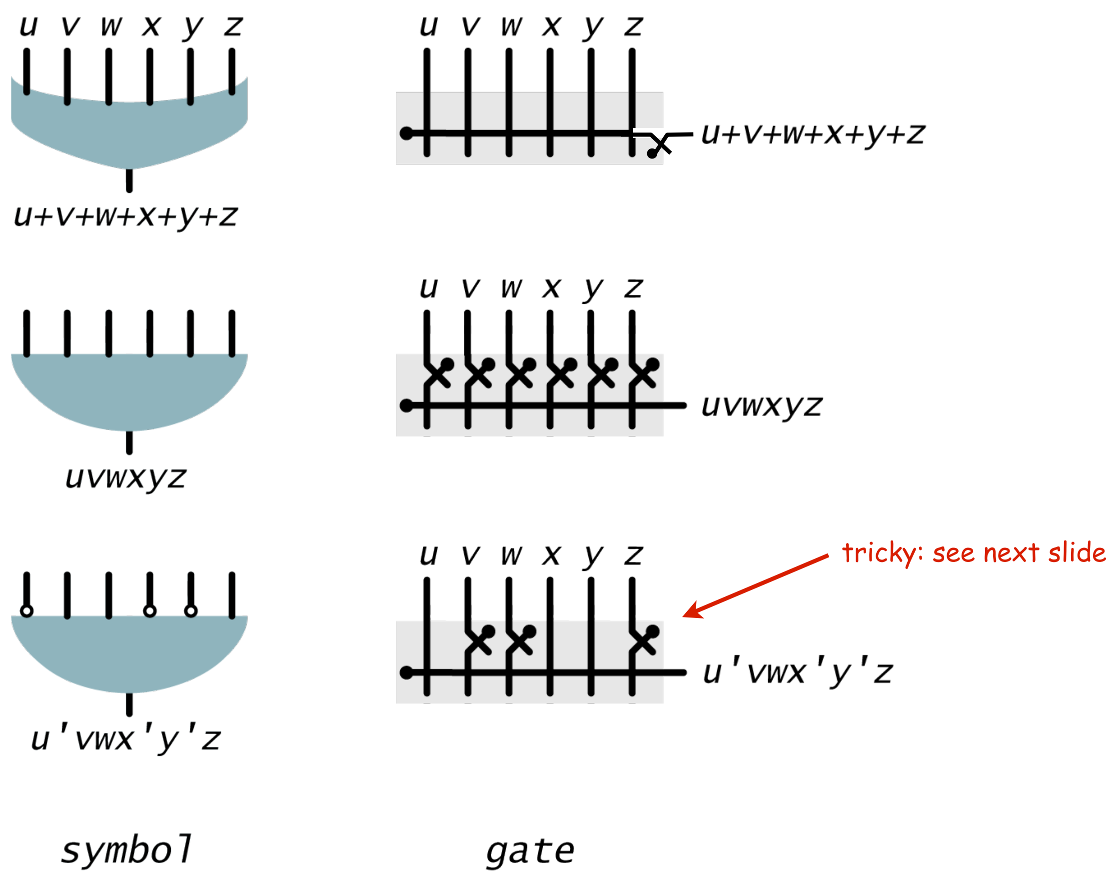
- OR: 1 if any input is 1; 0 otherwise.
- AND: 1 if all inputs are 1; 0 otherwise.
- Generalized: negate some inputs.



Multiway Gates

Multiway gates.

- OR: 1 if any input is 1; 0 otherwise.
- AND: 1 if all inputs are 1; 0 otherwise.
- Generalized: negate some inputs.



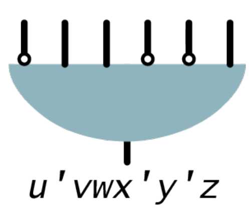
Building blocks (summary)

Wires

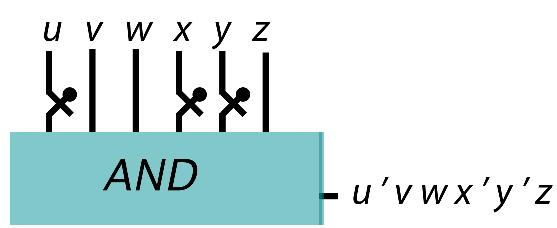
Controlled switches

Gates

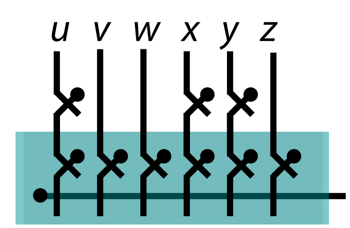
Generalized multiway gates



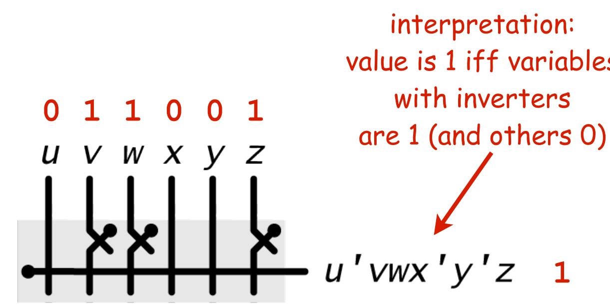
abstract



AND gate implementation



underlying circuit



simpler version

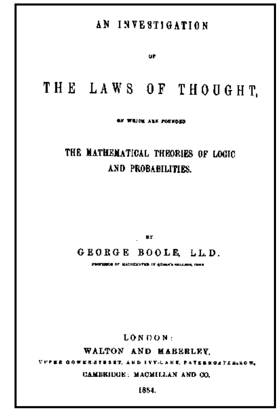
Boolean Algebra

Boolean Algebra

History.

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

"possibly the most important, and also the most famous, master's thesis of the [20th] century" — Howard Gardner



AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC
AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

LONDON:
WALTON AND MABERLEY,
UPPER GOWER-STREET, AND IVY-LANE, PATERNOSTER-ROW.
CAMBRIDGE: MACMILLAN AND CO.
1854.

Boolean algebra.

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

Relationship to circuits.

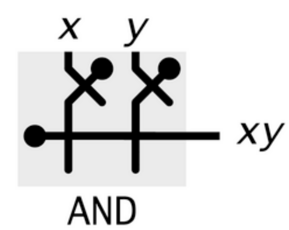
- Boolean variable: signal.
- Boolean function: circuit.

Copyright 2004, Sidney Harris
http://www.sciencecartoonsplus.com

Truth Table

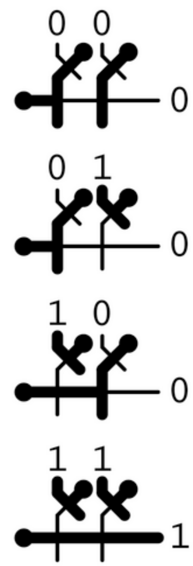
Truth table.

- Systematic method to describe Boolean function.
- One row for each possible input combination.
- n inputs $\Rightarrow$ 2^n rows.



x	y	$x\,y$
0	0	0
0	1	0
1	0	0
1	1	1

AND truth table



Truth Table for Functions of 2 Variables

Truth table.

- 16 Boolean functions of 2 variables.

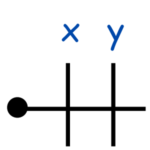
← every 4-bit value represents one

x	*y*	*ZERO*	*AND*		*x*		*y*	*XOR*	*OR*
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

truth table for all Boolean functions of 2 variables

x	*y*	*NOR*	*EQ*	*y'*		*x'*		*NAND*	*ONE*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

truth table for all Boolean functions of 2 variables



x NOR y

Truth Table for Functions of 3 Variables

Truth table.

- 16 Boolean functions of 2 variables. ← every 4-bit value represents one
- 256 Boolean functions of 3 variables. ← every 8-bit value represents one
- 2^(2^*n*) Boolean functions of *n* variables! ← every 2^n-bit value represents one

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

some functions of 3 variables

Universality of AND, OR, NOT

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

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

notation	meaning
x'	$NOT\ x$
$x\ y$	$x\ AND\ y$
$x + y$	$x\ OR\ y$

Expressing XOR Using AND, OR, NOT

x	y	x'	y'	$x'y$	xy'	$x'y + xy'$	$x\ XOR\ y$
0	0	1	1	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	1
1	1	0	0	0	0	0	0

Exercise. Show $\{AND, NOT\}$ are universal. (Hint: DeMorgan's law: $(x'\ y')' = x + y.$)

Exercise. Show $\{NOR\}$ is universal. (Stay tuned for easy proof)

Sum-of-Products

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

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

proves that { AND, OR, NOT } are universal

x	y	z	MAJ	$x'yz$	$xy'z$	xyz'	xyz	$x'yz + xy'z + xyz' + xyz$
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 using sum-of-products

Translate Boolean Formula to Boolean Circuit

Sum-of-products. *XOR*.

$$XOR = x'y + xy'$$

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

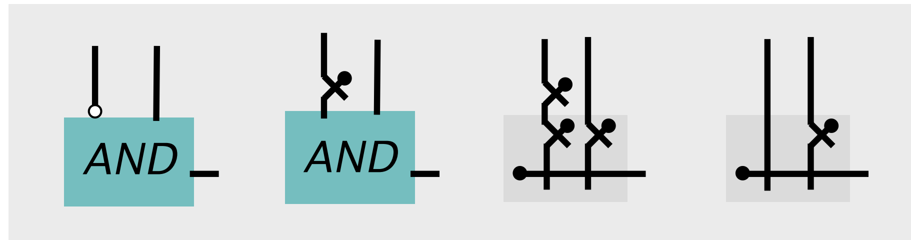
Truth table

Circuit

Translate Boolean Formula to Boolean Circuit

Sum-of-products. *XOR.*

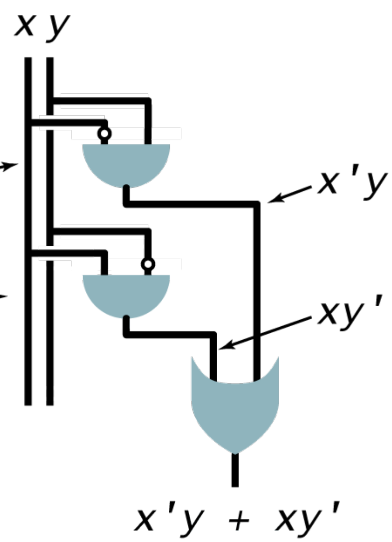
Key transformation from abstract to real circuit



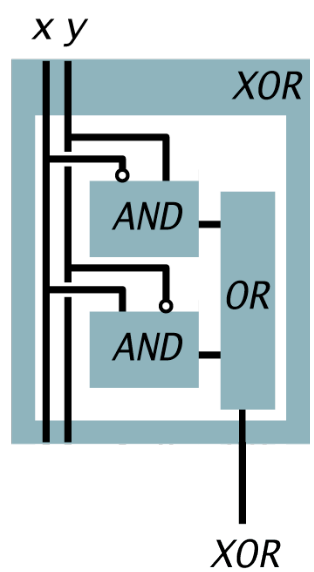
$XOR = x'y + xy'$

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

Truth table



Abstract circuit

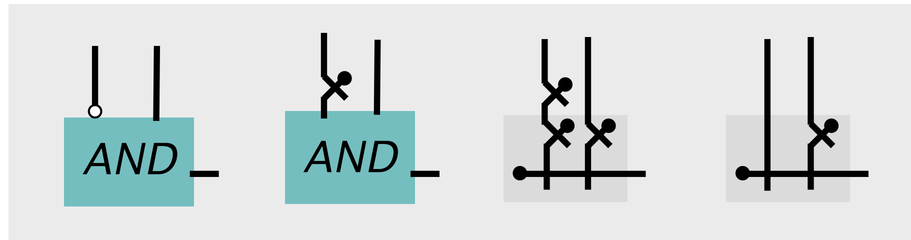


Circuit

Translate Boolean Formula to Boolean Circuit

Example 1. *XOR.*

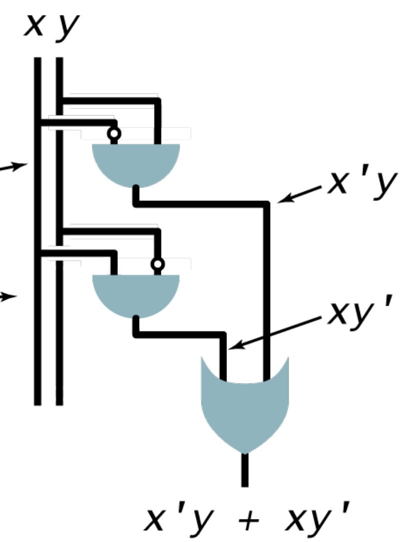
Key transformation from abstract to real circuit



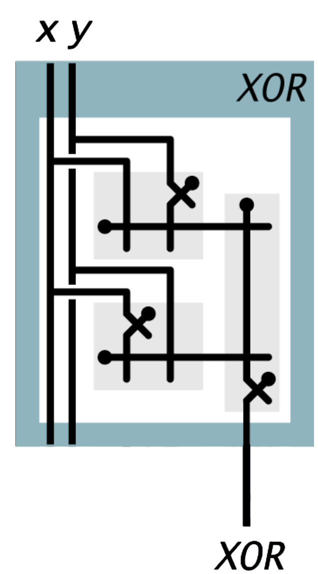
XOR = x'y + xy'

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

Truth table



Abstract circuit



Circuit

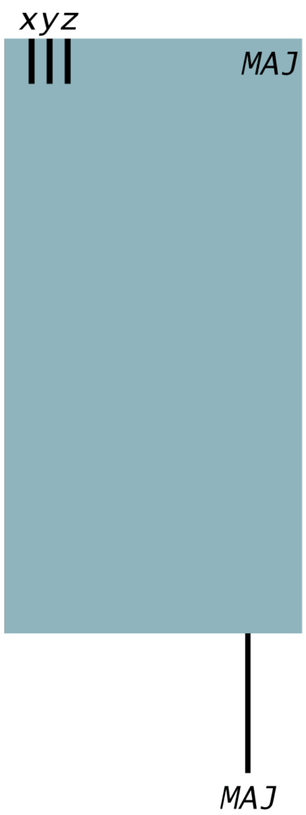
Translate Boolean Formula to Boolean Circuit

Example 2. Majority.

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

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

Truth table



Circuit

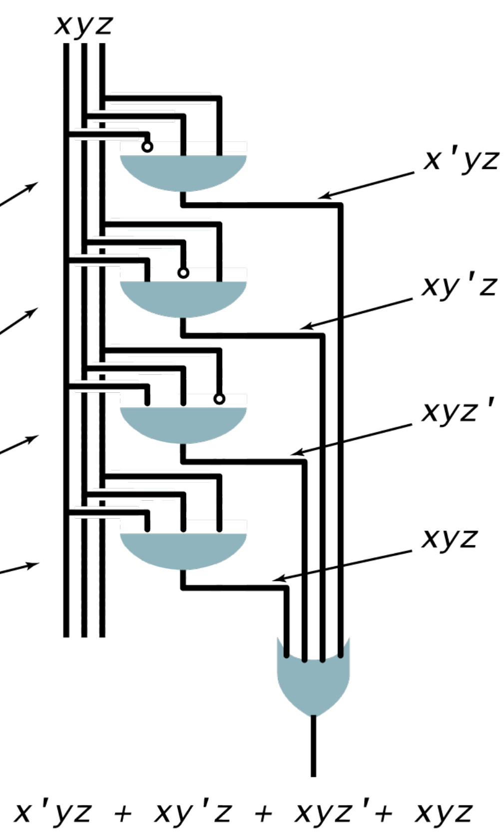
Translate Boolean Formula to Boolean Circuit

Example 2. Majority.

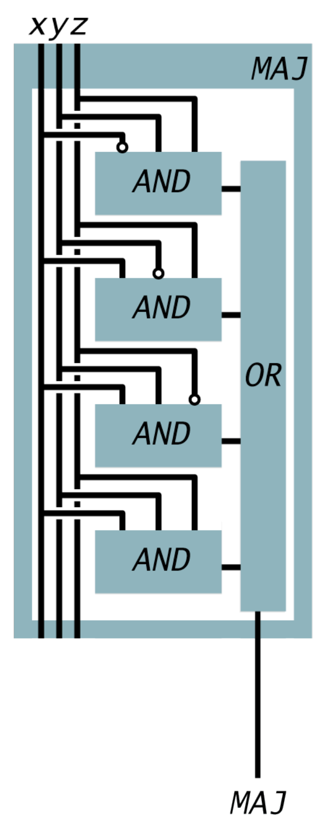
MAJ = x'yz + xy'z + xyz'+ xyz

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

Truth table



Abstract circuit



Circuit

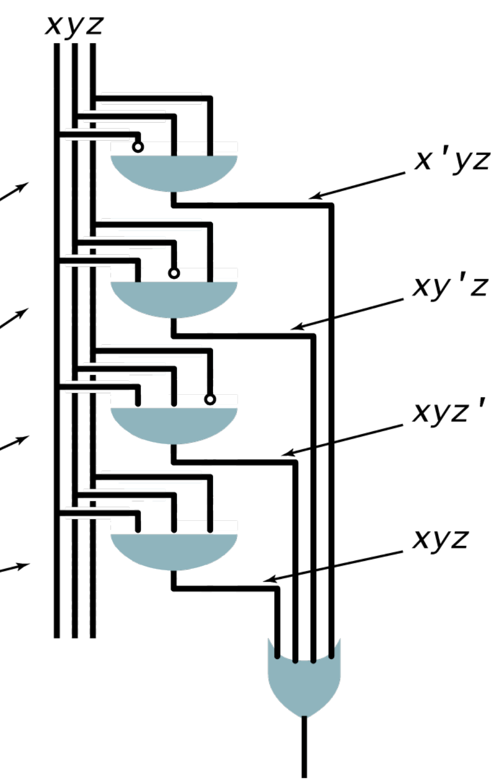
Translate Boolean Formula to Boolean Circuit

Example 2. Majority.

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

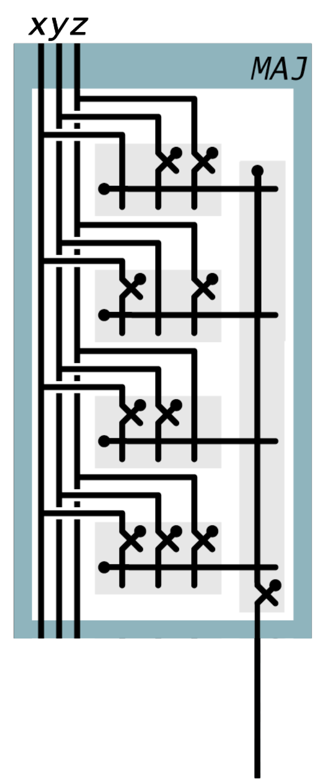
x	y	z	MAJ
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Truth table



x'yz + xy'z + xyz'+ xyz

Abstract circuit



MAJ

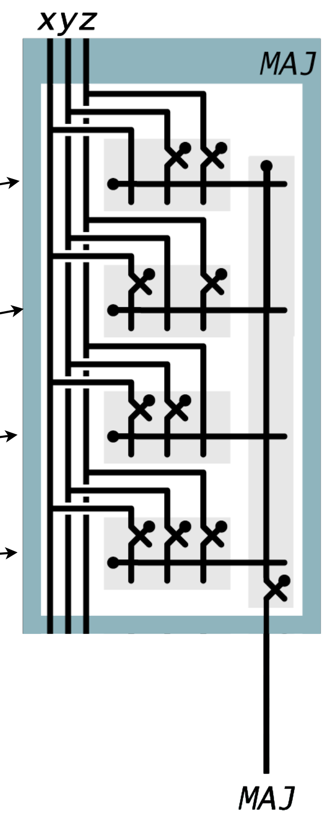
Circuit

Translate Boolean Formula to Boolean Circuit

Example 2. Majority.

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

x	y	z	MA.
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



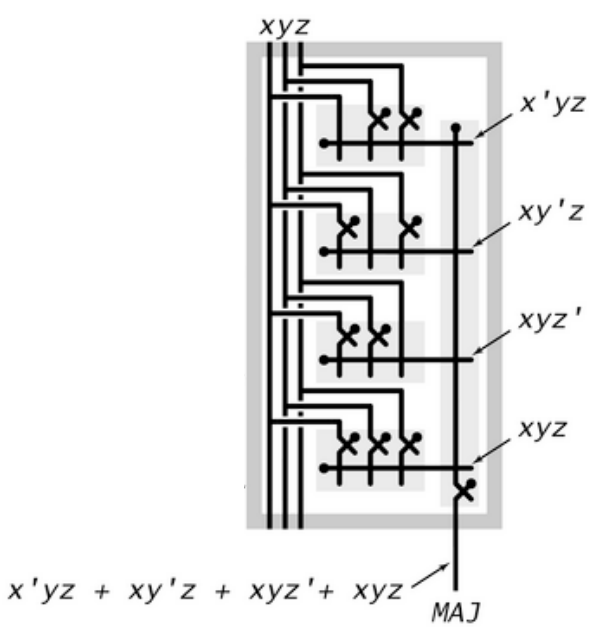
Truth table *Abstract circuit* *Circuit*

Simplification Using Boolean Algebra

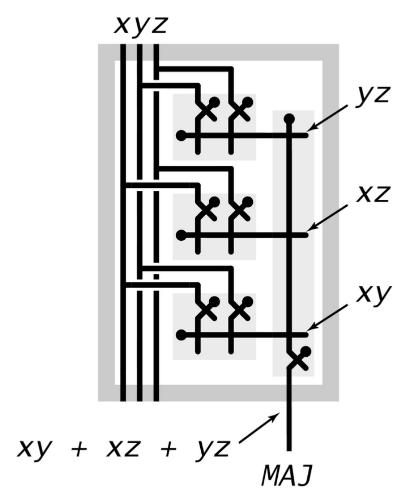
Many possible circuits for each Boolean function.

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

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



size = 10, depth = 2



size = 7, depth = 2

Combinational Circuit Design: Summary

Problem: Compute the value of a boolean function

Ingredients.

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

Instructions.

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

Bottom line (profound idea):

It is easy to design a circuit to compute ANY boolean function.

Caveat (stay tuned): Circuit might be huge.

Translate Boolean Formula to Boolean Circuit

Example 3. Odd parity

- 1 if odd number of inputs are 1.
- 0 otherwise.

x	y	z	ODD	$x'y'z$	$x'yz'$	$xy'z'$	xyz	$x'y'z + x'yz' + xy'z' + xyz$
0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	1
0	1	0	1	0	1	0	0	1
0	1	1	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	1	1

Expressing ODD using sum-of-products

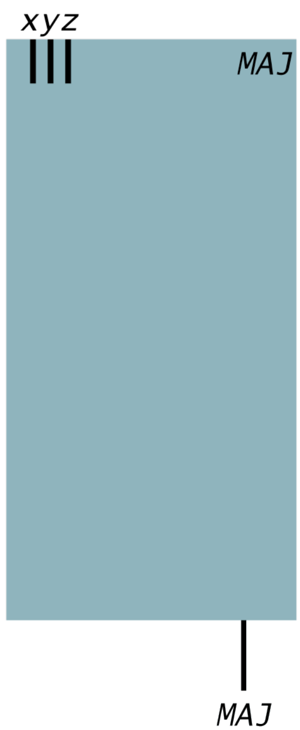
Translate Boolean Formula to Boolean Circuit

Example 3. Odd parity

- 1 if odd number of inputs are 1.
- 0 otherwise.

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

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



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

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

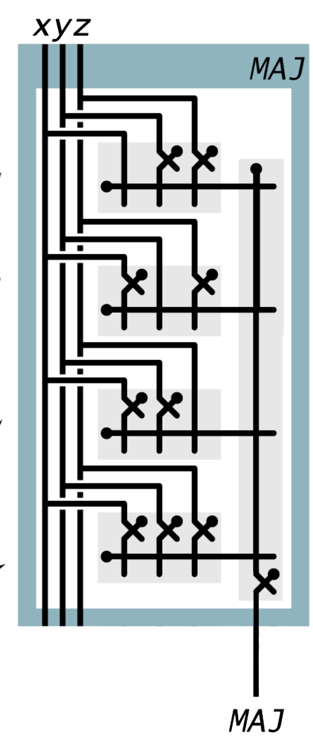
Translate Boolean Formula to Boolean Circuit

Example 3. Odd parity

- 1 if odd number of inputs are 1.
- 0 otherwise.

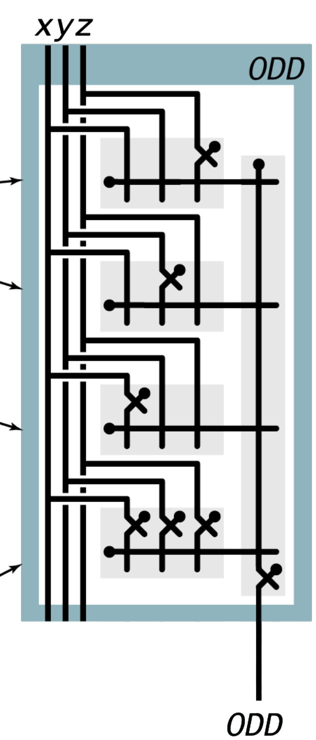
$MAJ = x'yz + xy'z + xyz' + xyz$

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



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

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



Adder Circuit

Let's Make an Adder Circuit

same idea scales to 64-bit adder in your computer

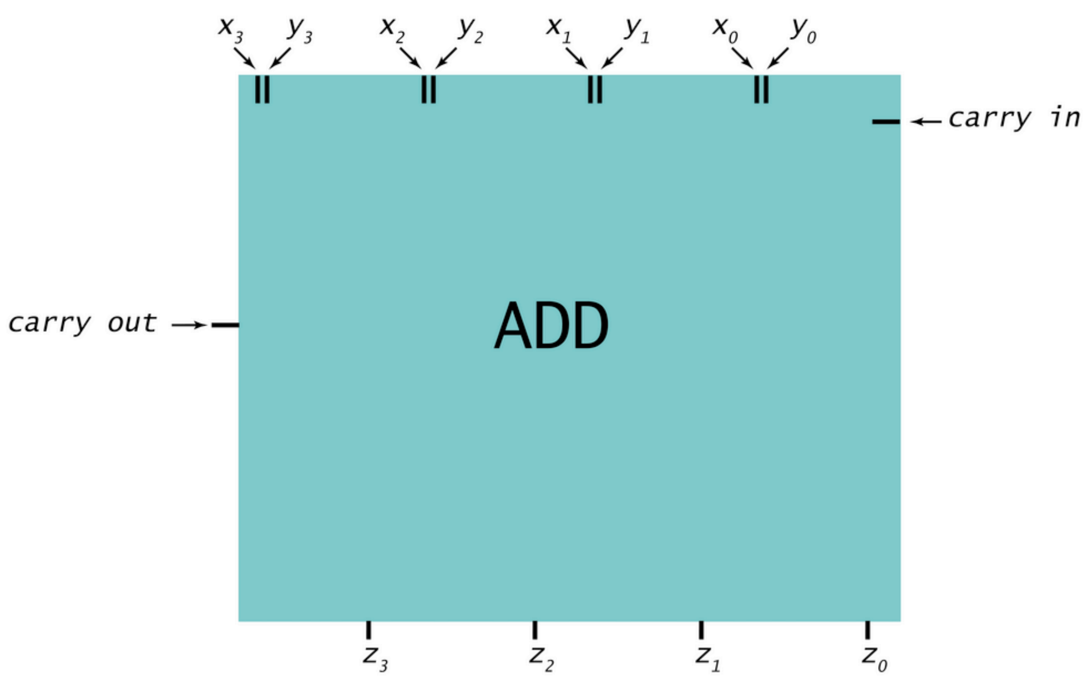
Goal. $x + y = z$ for 4-bit integers.

- We build 4-bit adder: 9 inputs, 4 outputs.
- Each output bit is a boolean function of the inputs.
- Standard method applies.

	1	1	1	0
	2	4	8	7
+	3	5	7	9
	6	0	6	6

Step 1. Represent input and output in binary.

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



	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.

Step 2. [first attempt]

- Build truth table.

c_{out} c_{in}

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

4-bit adder truth table

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
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	1
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	0	0	0	1	0	0	0	1	0	0
.	.	.	.	.	.	.	.	.	.	.	.	.
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.

c_{out}	c_3	c_2	c_1	c_0	$= 0$
	x_3	x_2	x_1	x_0	
+	y_3	y_2	y_1	y_0	
	z_3	z_2	z_1	z_0	

Step 2. Do one bit at a time!

- Build truth table for carry bit.
- Build truth table for summand bit.

carry bit

x_i	y_i	c_i	c_{i+1}
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

summand bit

x_i	y_i	c_i	z_i
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

Let's Make an Adder Circuit

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

Step 3. A surprise!

- carry bit is majority function
- summand bit is odd parity function.

c_{out}	c_3	c_2	c_1	$c_0 = 0$
	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

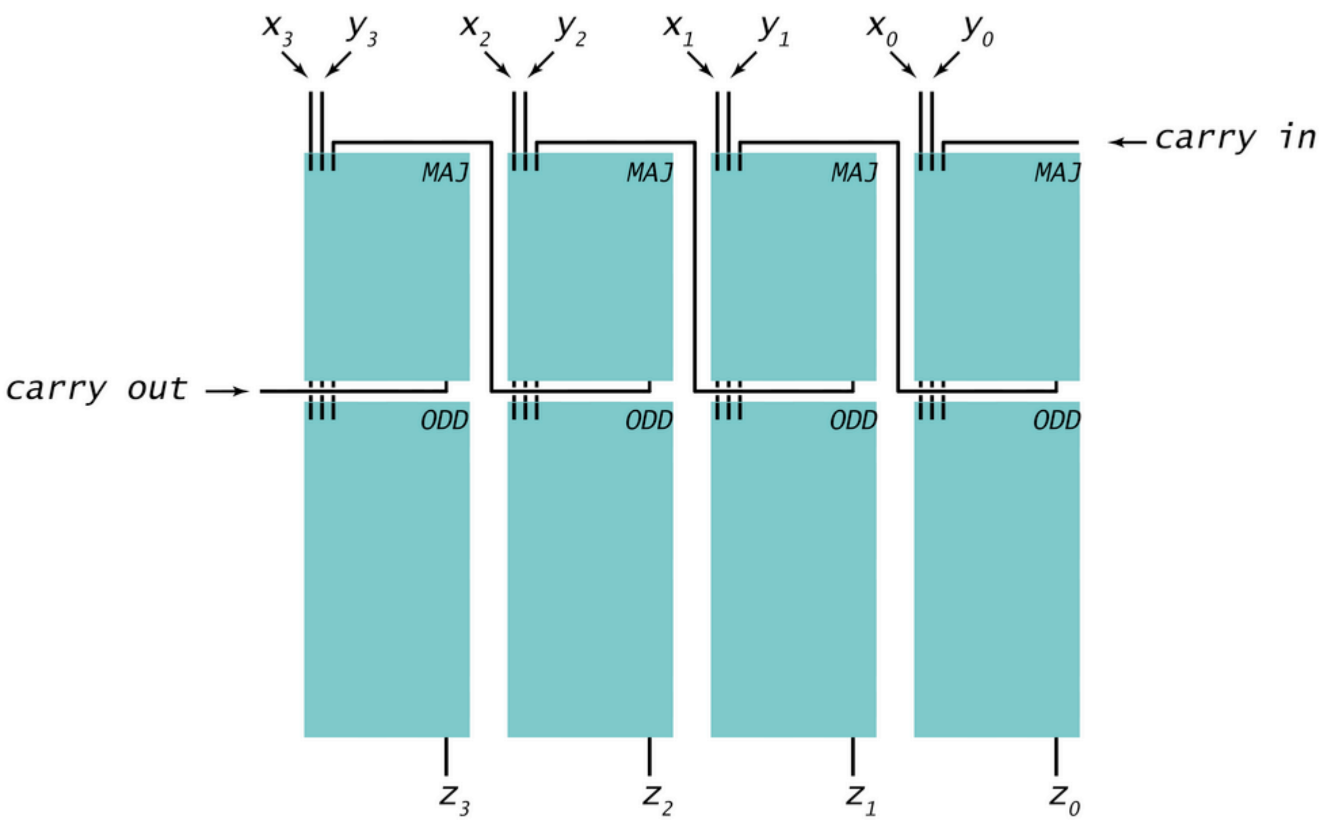
summand bit

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

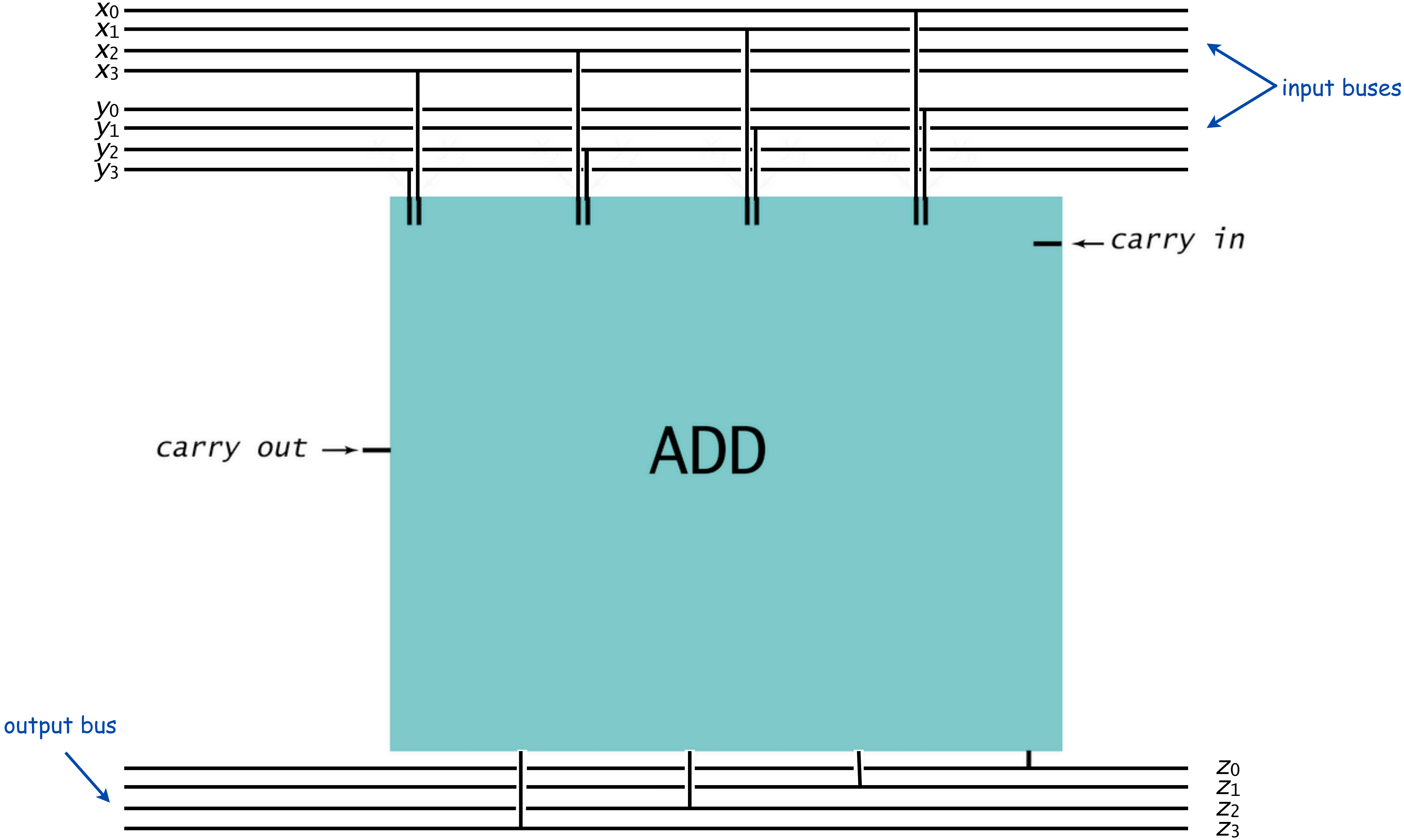
Let's Make an Adder Circuit

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

Step 4.
- Transform Boolean expression into circuit (use known components!).
- Chain together 1-bit adders.
- That's it!

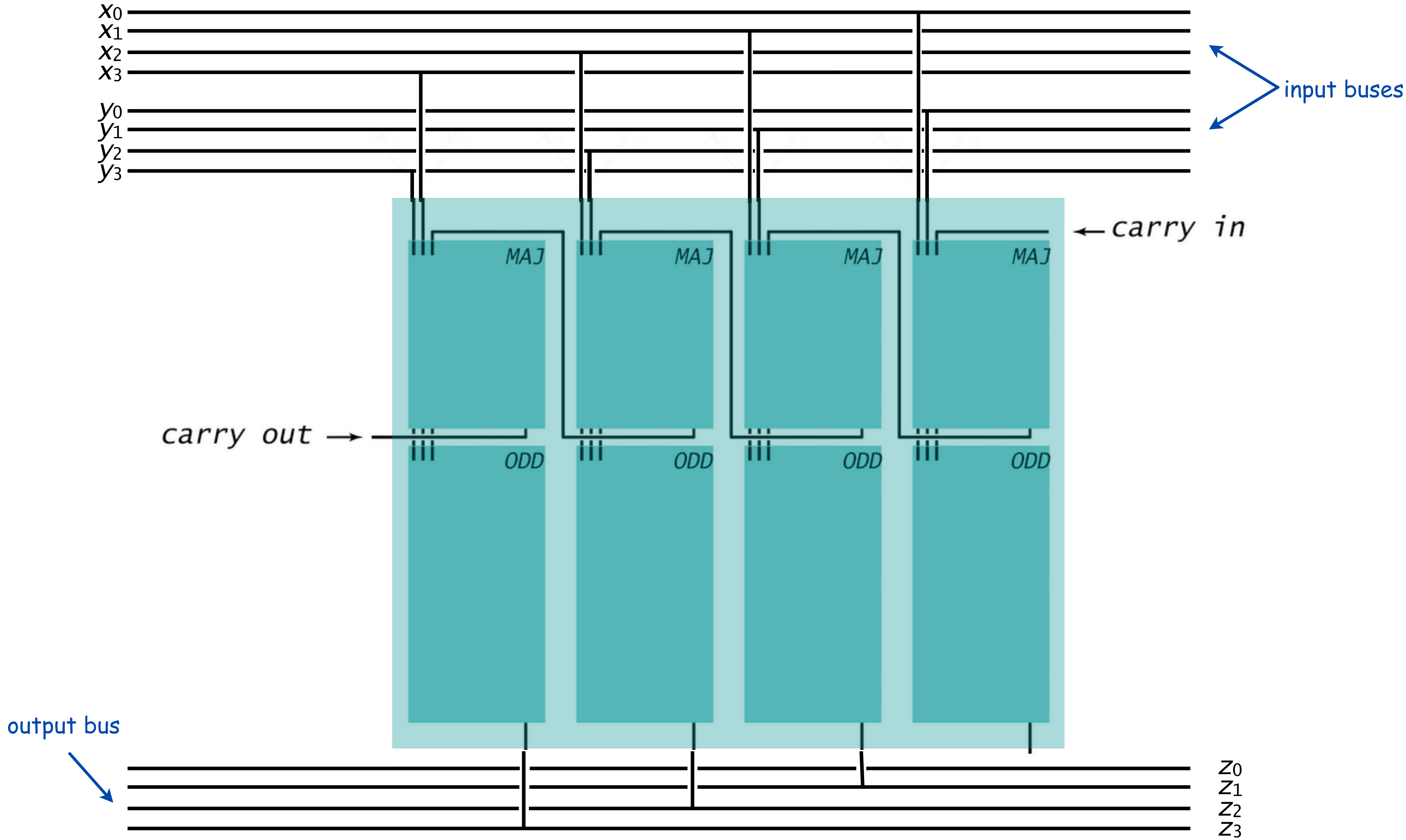


Adder: Interface

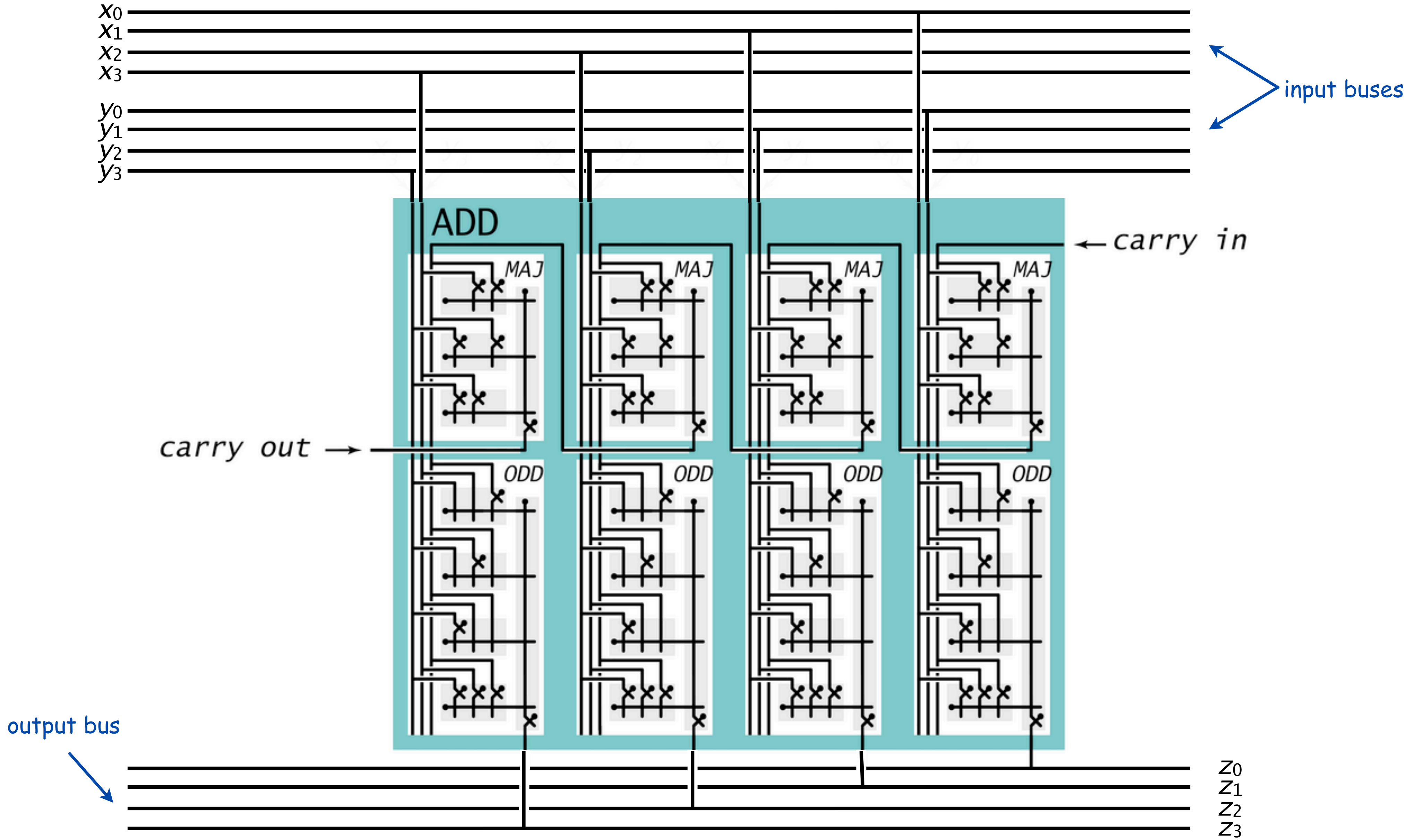


A bus is a group of wires that connect (carry data values) to other components.

Adder: Component Level View

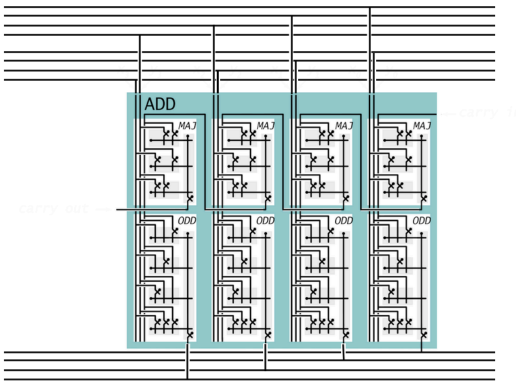


Adder: Switch Level View

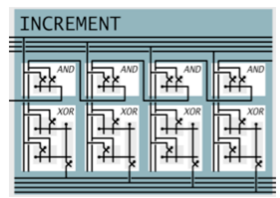


Useful Combinational Circuits

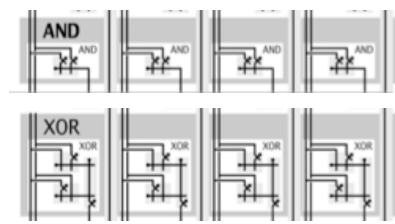
Adder



Incrementer (easy, add 0001)

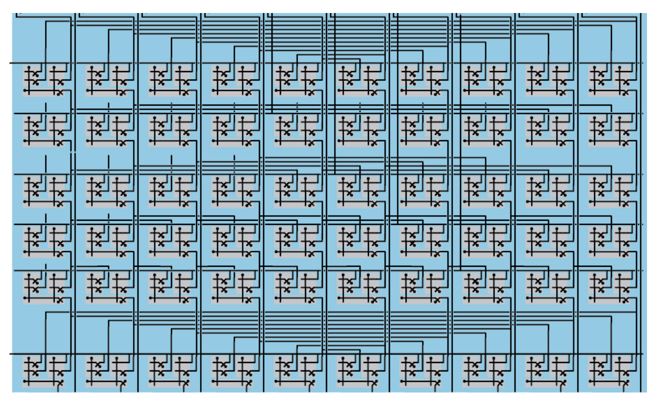


Bitwise AND, XOR (easy)



Decoder [next slide]

Shifter (clever, but we'll skip details)



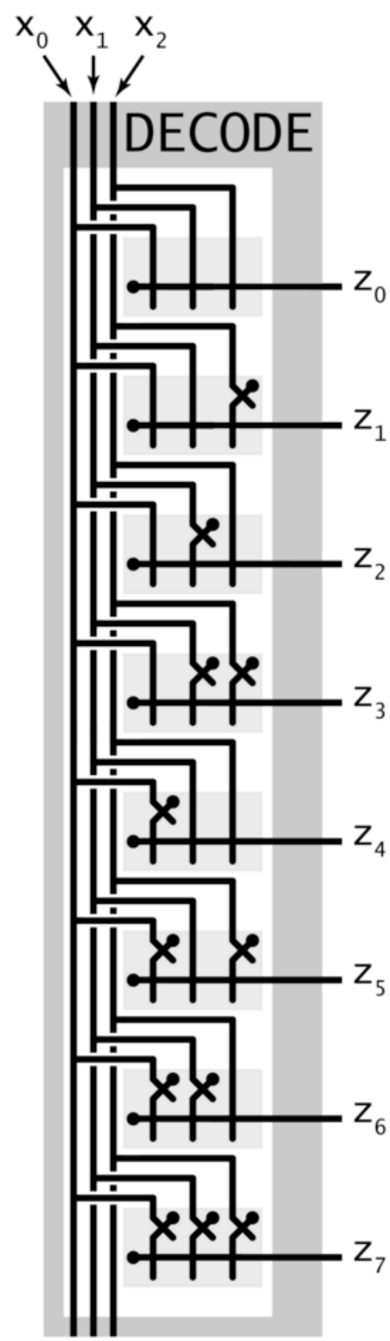
Multiplexer [next lecture]

Decoder

Decoder. [n-bit]

- n address inputs, 2^n data outputs.
- Addressed output bit is 1; others are 0.
- Compact implementation of n Boolean functions

x_0	x_1	x_2	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
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



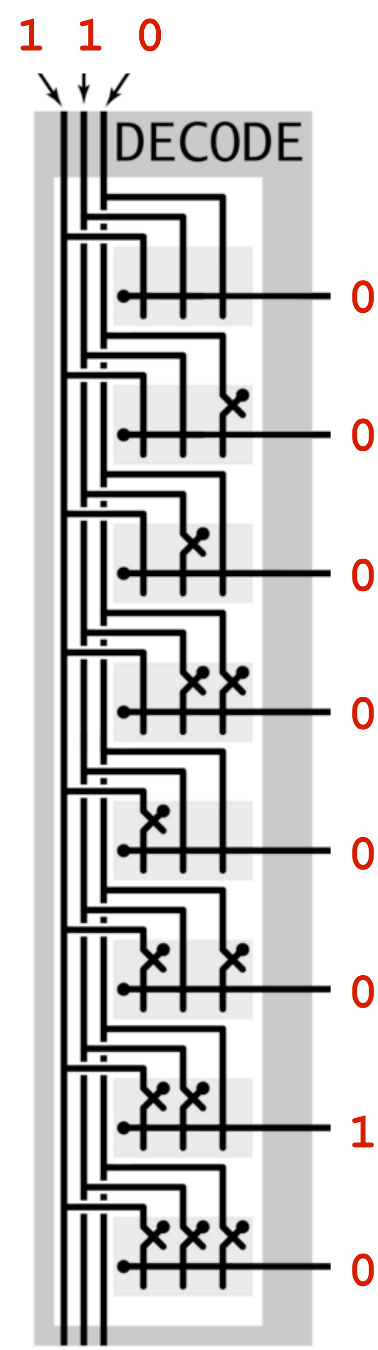
3-bit Decoder

Decoder

Decoder. [n-bit]

- n address inputs, 2^n data outputs.
- Addressed output bit is 1; others are 0.
- Compact implementation of n Boolean functions

x_0	x_1	x_2	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
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



3-bit Decoder

Decoder application: Your computer's ALU!

ALU: Arithmetic and Logic Unit

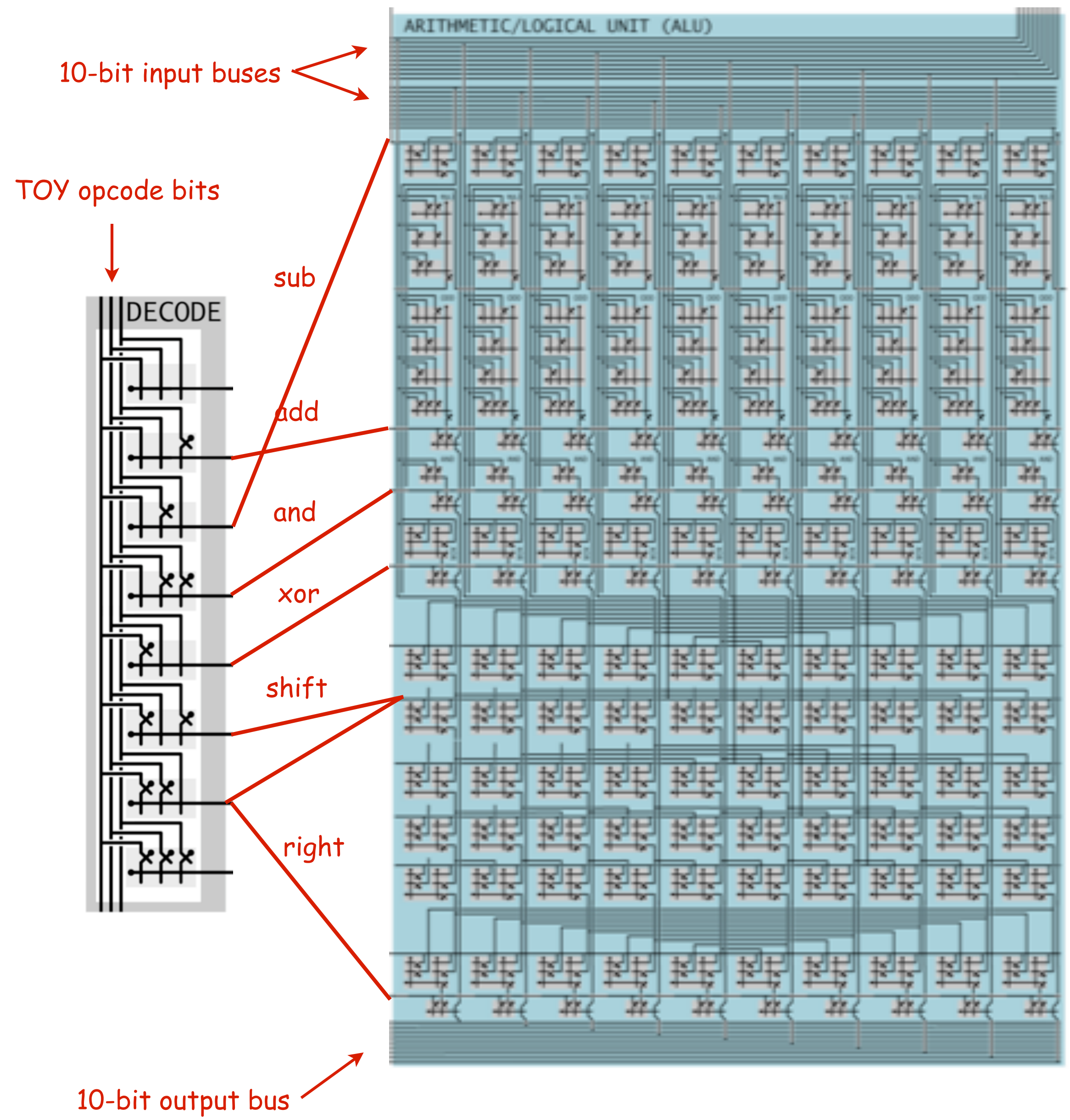
- implements instructions
- input, output connects to registers via buses

Ex: TOY-Lite (10 bit words)

1: add
2: subtract
3: and
4: xor
5: shift left
6: shift right

Details:

- All circuits compute their result.
- Decoder lines AND all results.
- "one-hot" OR collects answer.



Summary

Lessons for software design apply to hardware design!

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

Layers of abstraction apply with a vengeance!

- On/off.
- Controlled switch. [relay, transistor]
- Gates. [AND, OR, NOT]
- Boolean circuit. [MAJ, ODD]
- Adder.
- Shifter.
- Arithmetic logic unit.
- …
- TOY machine (stay tuned).
- Your computer.