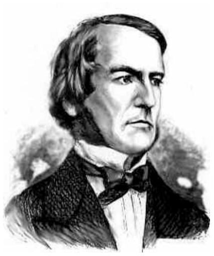
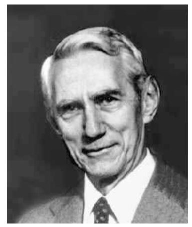
6. Combinational Circuits



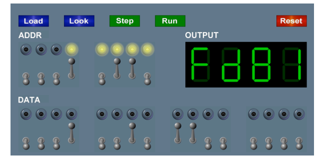
George Boole (1815 – 1864)



Claude Shannon (1916 – 2001)

Computer Architecture

TOY lectures. von Neumann machine.



This lecture. Boolean circuits.

Digital Circuits

Q. What is a digital system?
A. Digital: signals are 0 or 1.

analog: signals vary continuously

Q. Why digital systems?
A. Accurate, reliable, fast, cheap.

Basic abstractions.

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

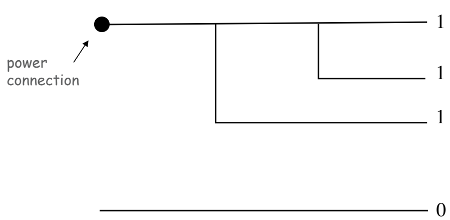
Applications. Cell phone, iPod, antilock breaks, microprocessors, …

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.



Logic Gates

NOT = x'

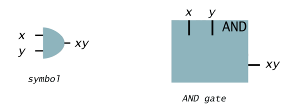
x	NOT
0	1
1	0

OR = x+y

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

AND = xy

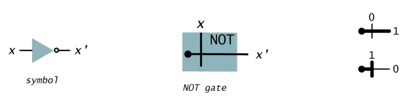
x y	AND
0 0	0
0 1	0
1 0	0
1 1	1



Logic Gates

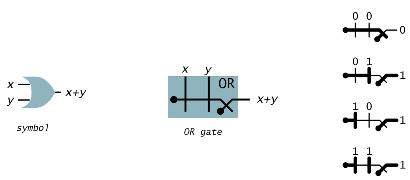
NOT = x'

x	NOT
0	1
1	0



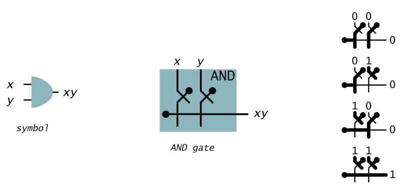
OR = x+y

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



AND = xy

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

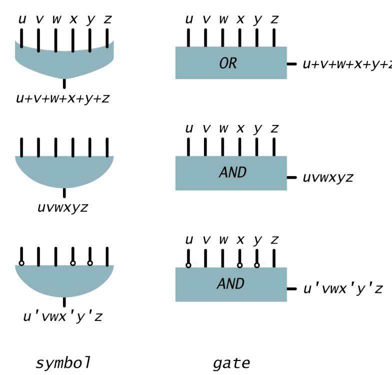


implementations with switches

Multiway Gates

Multiway gates.

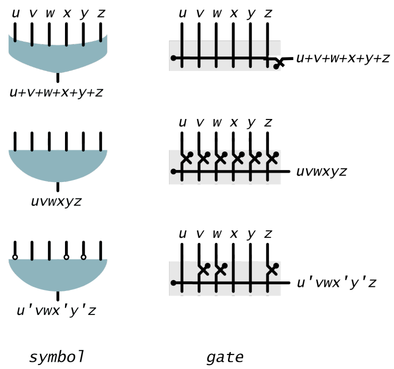
- OR: 1 if any input is 1; 0 otherwise.
- AND: 1 if all inputs are 1; 0 otherwise.
- Generalized: negate some inputs.



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.



Boolean Algebra

Boolean Algebra

History.

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

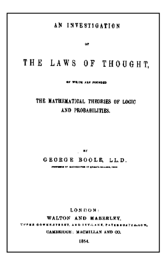
"possibly the most important, and also the most famous, master's thesis of the [20th] century" — Howard Gardner

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.



AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

THE MATHEMATICAL THEORIES OF LOGIC AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

LONDON:
WALTON AND MABERLEY,
CAMBRIDGE: MACMILLAN AND CO.
1854.

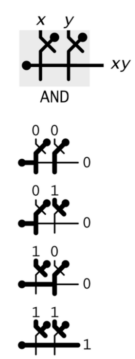
Truth Table

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

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
- $2^\wedge(2^\wedge n)$ 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*}, {*OR*, *NOT*}, {*NAND*} are universal.

Hint. DeMorgan's law: $(x' y')' = x + y$.

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

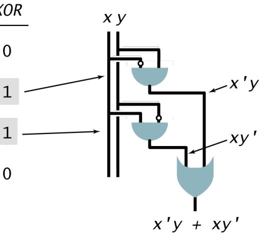
Translate Boolean Formula to Boolean Circuit

Sum-of-products. *XOR*.

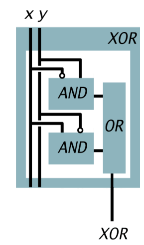
XOR = x'y + xy'

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

Truth table



Abstract circuit



Circuit

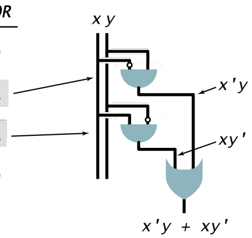
Translate Boolean Formula to Boolean Circuit

Sum-of-products. *XOR*.

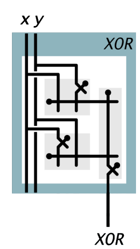
XOR = x'y + xy'

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

Truth table



Abstract circuit



Circuit

Translate Boolean Formula to Boolean Circuit

Sum-of-products. 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

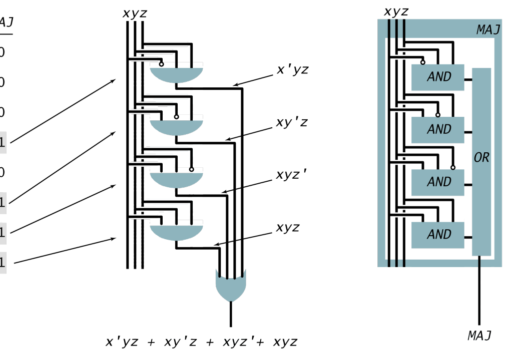
Translate Boolean Formula to Boolean Circuit

Sum-of-products. 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

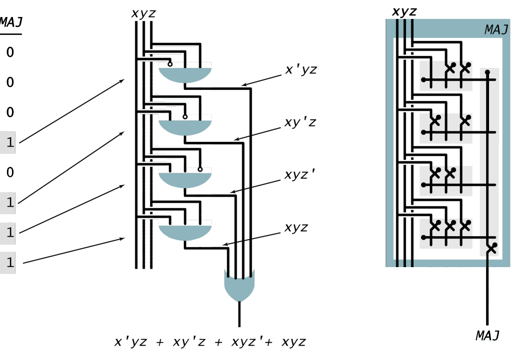
Translate Boolean Formula to Boolean Circuit

Sum-of-products. 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

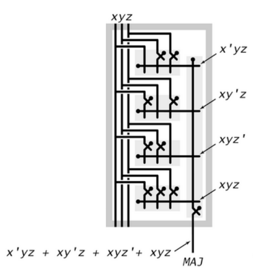
Circuit

Simplification Using Boolean Algebra

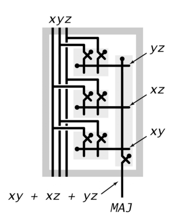
Many possible circuits for each Boolean function.

- 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

Expressing a Boolean Function Using AND, OR, NOT

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.

ODD Parity Circuit

ODD(x, y, z).

- 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

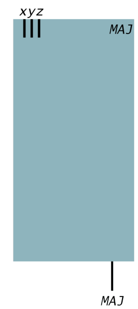
ODD Parity Circuit

ODD(x, y, z).

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

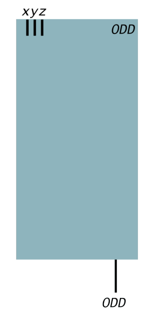
MAJ = x'yz + xy'z + xyz'+ xyz

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



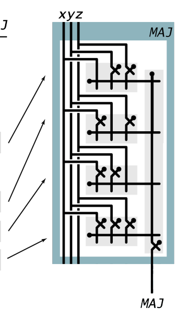
ODD Parity Circuit

ODD(x, y, z).

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

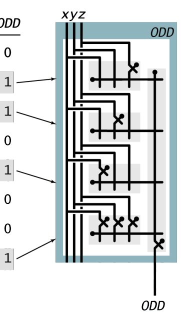
MAJ = x'yz + xy'z + xyz'+ xyz

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

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

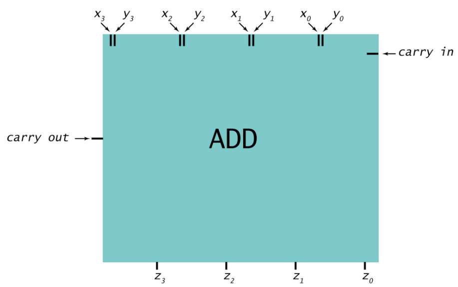
- We build 4-bit adder: 9 inputs, 4 outputs.
- Same idea scales to 128-bit adder.
- Key computer component.

Step 1. Represent input and output in binary.

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

	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}
	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	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.

Step 2. [do one bit at a time]

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

	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}
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.

- Derive (simplified) Boolean expression.

	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

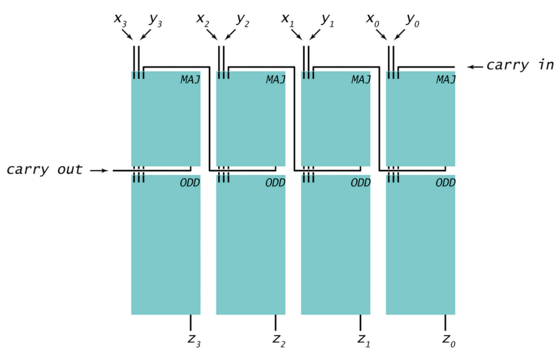
x_i	y_i	c_i	z_i	ODD
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Let's Make an Adder Circuit

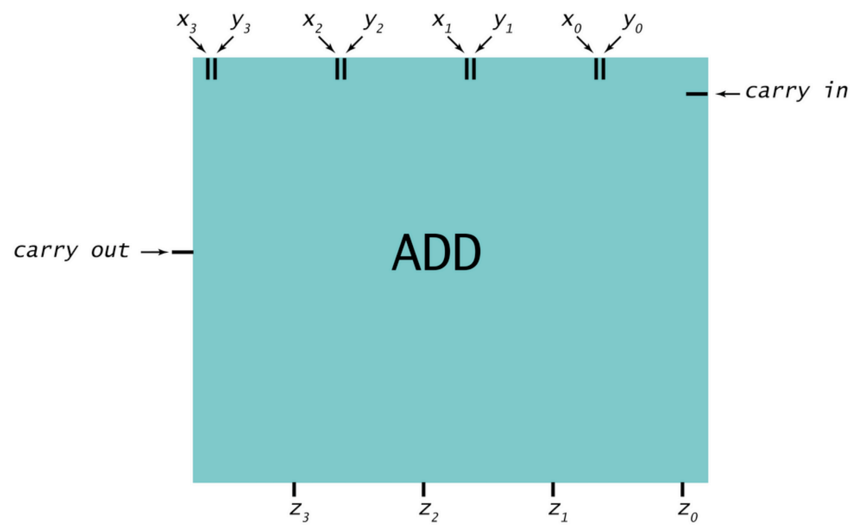
Goal. $x + y = z$ for 4-bit integers.

Step 4.

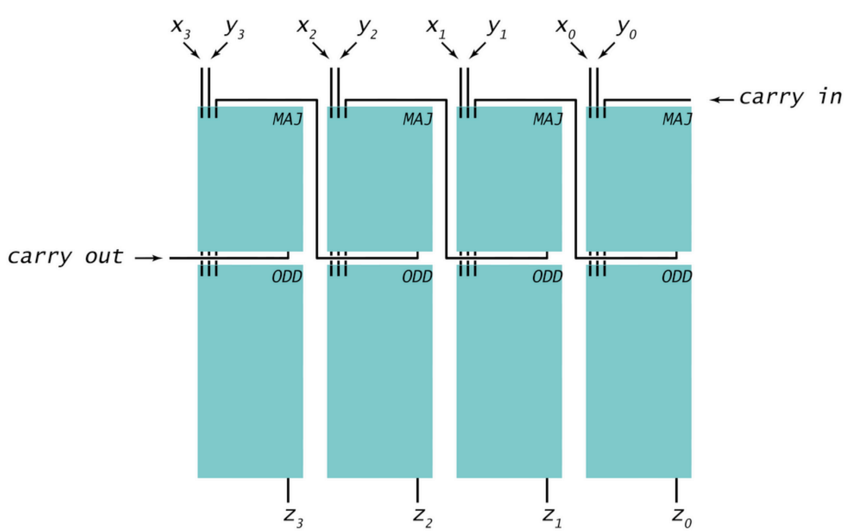
- Transform Boolean expression into circuit.
- Chain together 1-bit adders.



Adder: Interface



Adder: Component Level View



Adder: Switch Level View