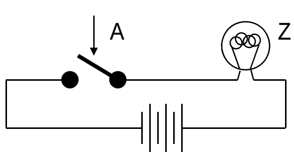
Combinational logic

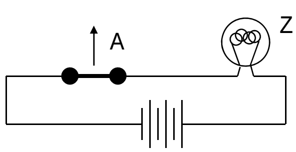
- Switches
- Basic logic and truth tables
- Logic functions
- Boolean algebra
- Proofs by re-writing and by perfect induction

Switches: basic element of physical implementations

- Implementing a simple circuit (arrow shows action if wire changes to "1"):



close switch (if A is "1" or asserted)
and turn on light bulb (Z)



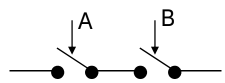
open switch (if A is "0" or unasserted)
and turn off light bulb (Z)

$Z \equiv A$

Switches (cont'd)

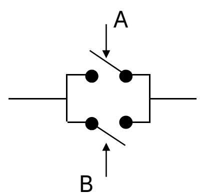
- Compose switches into more complex ones (Boolean functions):

AND



Z ≡ A <u>and</u> B

OR



Z ≡ A <u>or</u> B

Switching networks

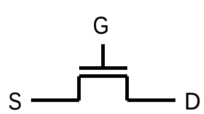
- Switch settings
  - determine whether or not a conducting path exists to light the light bulb
- To build larger computations
  - use the light bulb (output of the network)
    to set other switches (inputs to another network)

Transistor networks

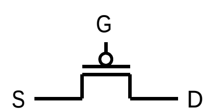
- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and
    normally-closed switches
- MOS transistors act as voltage-controlled switches
  - similar, though easier to work with than relays.

MOS transistors

- MOS transistors have three terminals: drain, gate, and source
  - they act as switches in the following way:
    if the voltage on the gate terminal is (some amount) higher/lower than the source terminal then a conducting path will be established between the drain and source terminals

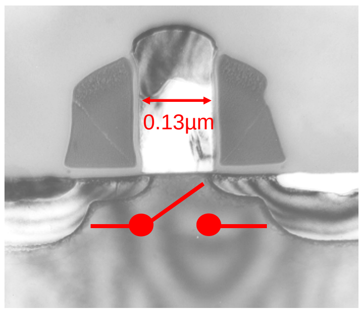


n-channel
open when voltage at G is low
closes when:
voltage(G) > voltage (S) + ε

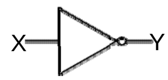


p-channel
closed when voltage at G is low
opens when:
voltage(G) < voltage (S) – ε

Most digital logic is CMOS

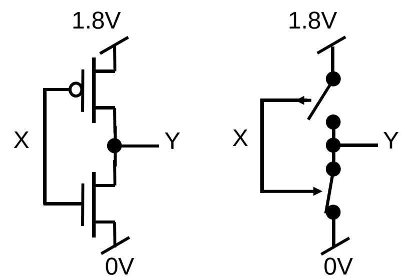


Mark Bohr Intel



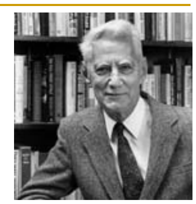
0V ≡ Logic 0
1.8V ≡ Logic 1

X	Y
0V	1.8V
1.8V	0V

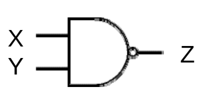


Multi-input logic gates

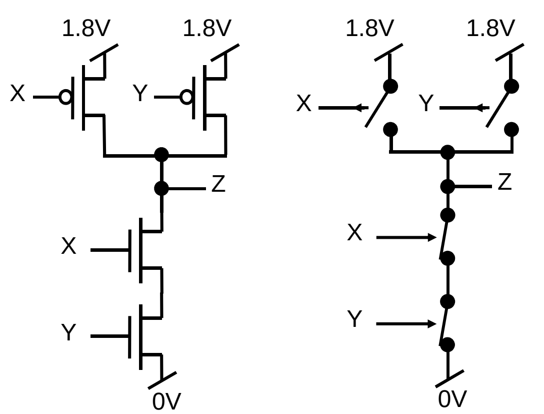
- CMOS logic gates are inverting
  - Easy to implement NAND, NOR, NOT while AND, OR, and Buffer are harder



Claude Shannon – 1938

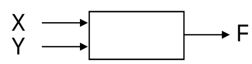


X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:
  - in general, there are 2**(2**n) functions of n inputs



X	Y	16 possible functions (F_0–F_{15})															
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
		0	X and Y		X		Y	X xor Y	X or Y	X nor Y not (X or Y)	X = Y	not Y		not X		X nand Y not (X and Y)	1

Minimal set of functions

- Can we implement all logic functions from NOT, NOR, and NAND?
  - For example, implementing X and Y
    is the same as implementing not (X nand Y)
- In fact, we can do it with only NOR or only NAND
  - NOT is just a NAND or a NOR with both inputs tied together

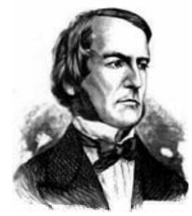
X	Y	X nor Y
0	0	1
1	1	0

X	Y	X nand Y
0	0	1
1	1	0

  - and NAND and NOR are "duals",
    that is, its easy to implement one using the other

X nand Y ≡ not ((not X) nor (not Y))
X nor Y ≡ not ((not X) nand (not Y))

Boolean algebra



George Boole – 1854

- An algebraic structure consists of
  - a set of elements B
  - binary operations { + , • }
  - and a unary operation { ' }
  - such that the following axioms hold:

1. the set B contains at least two elements: a, b		
2. closure:	a + b is in B	a • b is in B
3. commutativity:	a + b = b + a	a • b = b • a
4. associativity:	a + (b + c) = (a + b) + c	a • (b • c) = (a • b) • c
5. identity:	a + 0 = a	a • 1 = a
6. distributivity:	a + (b • c) = (a + b) • (a + c)	a • (b + c) = (a • b) + (a • c)
7. complementarity:	a + a' = 1	a • a' = 0

Logic functions and Boolean algebra

- Any logic function that can be expressed as a truth table can be written as an expression in Boolean algebra using the operators: ', +, and •

X, Y are Boolean algebra variables

X	Y	X • Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X'	X' • Y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

X	Y	X'	Y'	X • Y	X' • Y'	(X • Y) + (X' • Y')
0	0	1	1	0	1	1
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	1

(X • Y) + (X' • Y') → X = Y

Boolean expression that is true when the variables X and Y have the same value and false, otherwise¾¾

Axioms and theorems of Boolean algebra

- identity
  1. $X + 0 = X$ — 1D. $X \bullet 1 = X$
- null
  2. $X + 1 = 1$ — 2D. $X \bullet 0 = 0$
- idempotency:
  3. $X + X = X$ — 3D. $X \bullet X = X$
- involution:
  4. $(X')' = X$
- complementarity:
  5. $X + X' = 1$ — 5D. $X \bullet X' = 0$
- commutativity:
  6. $X + Y = Y + X$ — 6D. $X \bullet Y = Y \bullet X$
- associativity:
  7. $(X + Y) + Z = X + (Y + Z)$ — 7D. $(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$
- distributivity:
  8. $X \bullet (Y + Z) = (X \bullet Y) + (X \bullet Z)$ — 8D. $X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$

Axioms and theorems of Boolean algebra (cont'd)

- uniting:
  9. $X \bullet Y + X \bullet Y' = X$ — 9D. $(X + Y) \bullet (X + Y') = X$
- absorption:
  10. $X + X \bullet Y = X$ — 10D. $X \bullet (X + Y) = X$
  11. $(X + Y') \bullet Y = X \bullet Y$ — 11D. $(X \bullet Y') + Y = X + Y$
- factoring:
  12. $(X + Y) \bullet (X' + Z) = X \bullet Z + X' \bullet Y$ — 12D. $X \bullet Y + X' \bullet Z = (X + Z) \bullet (X' + Y)$
- concensus:
  13. $(X \bullet Y) + (Y \bullet Z) + (X' \bullet Z) = X \bullet Y + X' \bullet Z$ — 13D. $(X + Y) \bullet (Y + Z) \bullet (X' + Z) = (X + Y) \bullet (X' + Z)$
- de Morgan's:
  14. $(X + Y + ...)' = X' \bullet Y' \bullet ...$ — 14D. $(X \bullet Y \bullet ...)' = X' + Y' + ...$
- generalized de Morgan's:
  15. $f'(X_1,X_2,...,X_n,0,1,+,\bullet) = f(X_1',X_2',...,X_n',1,0,\bullet,+)$

Axioms and theorems of Boolean algebra (cont'd)

- Duality
  - a dual of a Boolean expression is derived by replacing • by +, + by •, 0 by 1, and 1 by 0, and leaving variables unchanged
  - any theorem that can be proven is thus also proven for its dual!
  - a meta-theorem (a theorem about theorems)
- duality:
  16. $X + Y + \ldots \Leftrightarrow X \bullet Y \bullet \ldots$
- generalized duality:
  17. $f(X_1,X_2,\ldots,X_n,0,1,+,\bullet) \Leftrightarrow f(X_1,X_2,\ldots,X_n,1,0,\bullet,+)$
- Different than deMorgan's Law
  - this is a statement about theorems
  - this is not a way to manipulate (re-write) expressions

Proving theorems (rewriting)

- Using the laws of Boolean algebra:
  - e.g., prove the theorem: $X \bullet Y + X \bullet Y' = X$

distributivity (8)	$X \bullet Y + X \bullet Y'$	=	$X \bullet (Y + Y')$
complementarity (5)	$X \bullet (Y + Y')$	=	$X \bullet (1)$
identity (1D)	$X \bullet (1)$	=	X

  - e.g., prove the theorem: $X + X \bullet Y = X$

identity (1D)	$X + X \bullet Y$	=	$X \bullet 1 + X \bullet Y$
distributivity (8)	$X \bullet 1 + X \bullet Y$	=	$X \bullet (1 + Y)$
identity (2)	$X \bullet (1 + Y)$	=	$X \bullet (1)$
identity (1D)	$X \bullet (1)$	=	X

Activity

- Prove consensus theorem using the laws of Boolean algebra:
  - (X • Y) + (Y • Z) + (X' • Z) = X • Y + X' • Z

	(X • Y) + (Y • Z) + (X' • Z)
identity	(X • Y) + (1) • (Y • Z) + (X' • Z)
complementarity	(X • Y) + (X' + X) • (Y • Z) + (X' • Z)
distributivity	(X • Y) + (X' • Y • Z) + (X • Y • Z) + (X' • Z)
commutativity	(X • Y) + (X • Y • Z) + (X' • Y • Z) + (X' • Z)
factoring	(X • Y) • (1 + Z) + (X' • Z) • (1 + Y)
null	(X • Y) • (1) + (X' • Z) • (1)
identity	(X • Y) + (X' • Z)

identity	1. X + 0 = X	1D. X • 1 = X
null	2. X + 1 = 1	2D. X • 0 = 0
complementarity:	5. X + X' = 1	5D. X • X' = 0
commutativity:	6. X + Y = Y + X	6D. X • Y = Y • X
associativity:	7. (X + Y) + Z = X + (Y + Z)	7D. (X • Y) • Z = X • (Y • Z)
distributivity:	8. X • (Y + Z) = (X • Y) + (X • Z)	8D. X + (Y • Z) = (X + Y) • (X + Z)
factoring:	12. (X + Y) • (X' + Z) = X • Z + X' • Y	12D. X • Y + X' • Z = (X + Z) • (X' + Y)

Proving theorems (perfect induction)

- Using perfect induction (complete truth table):
  - e.g., de Morgan's:

(X + Y)' = X' • Y'
NOR is equivalent to AND
with inputs complemented

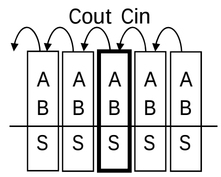
X	Y	X'	Y'	(X + Y)'	X' • Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

(X • Y)' = X' + Y'
NAND is equivalent to OR
with inputs complemented

X	Y	X'	Y'	(X • Y)'	X' + Y'
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

A simple example: 1-bit binary adder

- Inputs: A, B, Carry-in
- Outputs: Sum, Carry-out



A	B	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin

Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin