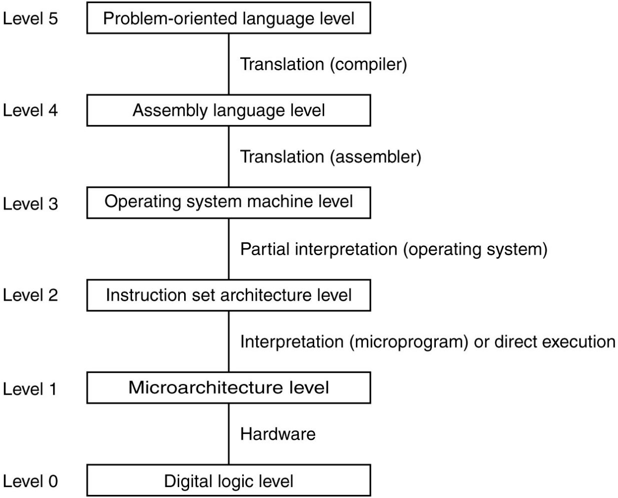
Boolean Algebra, Gates and Circuits

Kasper Brink
November 14, 2018

(Images taken from Tanenbaum, Structured Computer Organization, Fifth Edition, (c) 2006 Pearson Education, Inc.)

Radboud Universiteit

Multi-level Architecture



Analog vs. Digital Logic

Digital computers process *discrete* information.

All information is represented by two logical values:

logic 0 → e.g. voltage between 0.0 and 0.5 V

logic 1 → e.g. voltage between 1.0 and 1.5 V

To design and analyze digital circuits we make use of *Boolean Algebra*.

Boolean Values and Operators

Boolean values $B = \{0, 1\}$

Boolean variables $x \in B$

Boolean operator		corresponds to Logic operator	
$\overline{x}$	complement (NOT)	$\neg p$	negation
$x \cdot y$	product (AND)	$p \wedge q$	conjunction
$x + y$	sum (OR)	$p \vee q$	disjunction

x	$\overline{x}$
0	1
1	0

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Derived Operators

$\overline{xy}$	NAND
$\overline{x+y}$	NOR
$x \oplus y$	XOR (exclusive or)

x	y	$\overline{xy}$	$\overline{x+y}$	$x \oplus y$
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

Boolean Functions

Boolean function of degree n: $\quad f : B^n \to B$

We can describe each Boolean function by its *truth table*, for example this function $f : B^3 \to B$

x	y	z	f
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	0

The function f is true iff exactly one of its arguments is true.

How many distinct Boolean functions of degree n exist? $\quad 2^{2^n}$

Boolean Expressions

Boolean expression: expression using 0, 1, boolean variables and operators.

Every Boolean expression represents a Boolean function.

Example: $f(x,y) = x + \overline{y}$

x	y	$f(x,y)$
0	0	1
0	1	0
1	0	1
1	1	1

Conversely: can every Boolean function be written as a Boolean expression?

Deriving a Boolean Expression for a Function

Example:

x	y	$f(x,y)$	
0	0	1	$\overline{x}\,\overline{y}$
0	1	0	
1	0	1	$x\,\overline{y}$
1	1	1	$x\,y$

This gives us $f(x,y) = \overline{x}\,\overline{y} + x\overline{y} + xy$

Literal: x_i or $\overline{x_i}$
Minterm: product of literals where each variable occurs exactly once
Expression for f: sum of the minterms where $f = 1$

This is called the *(full) Disjunctive Normal Form* (DNF), or the *"sum of products"* representation.

Equivalence of Boolean Expressions

Two Boolean expressions that represent the same Boolean function are called *equivalent*.

Example: $x + \overline{y} = \overline{x}\,\overline{y} + x\overline{y} + xy$

You can prove this by showing that both expressions have the same truth table.

Often more convenient: rewriting expressions using the laws of Boolean algebra.

Laws of Boolean Algebra

Name	Multiplicative form	Additive form
Complement	$\overline{\overline{x}} = x$	
Identity	$x \cdot 1 = x$	$x + 0 = x$
Null	$x \cdot 0 = 0$	$x + 1 = 1$
Idempotent	$x\,x = x$	$x + x = x$
Inverse	$x\,\overline{x} = 0$	$x + \overline{x} = 1$
Commutative	$xy = yx$	$x + y = y + x$
Associative	$x(yz) = (xy)z$	$x + (y + z) = (x + y) + z$
Distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Absorption	$x(x + y) = x$	$x + xy = x$
De Morgan	$\overline{xy} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x}\,\overline{y}$

See also Tanenbaum, figure 3-6 (p. 156).

Rewriting Expressions (1)

Example: rewrite the following expression to (full) DNF.

$$
\begin{aligned}
(x+z)(\overline{x+\overline{y}}) &= (x+z)(\overline{x}\cdot\overline{\overline{y}}) && \text{(de Morgan)} \\
&= x\overline{x}y + z\overline{x}y && \text{(distributivity, double complement)} \\
&= 0 + z\overline{x}y && \text{(inverse, null)} \\
&= \overline{x}yz && \text{(identity, commutativity)}
\end{aligned}
$$

Rewriting Expressions (2)

Example: derive an expression for the following function, and simplify this.

x	y	z	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$\begin{aligned}
f(x,y,z) &= \overline{x}y\overline{z} + x\overline{y}\,\overline{z} + xy\overline{z} && \text{(DNF)}\\
&= (\overline{x}y + x\overline{y} + xy)\overline{z} && \text{(distributivity)}\\
&= (\overline{x}y + x\overline{y} + (xy + xy))\overline{z} && \text{(idempotent)}\\
&= (\overline{x}y + xy + x\overline{y} + xy)\overline{z} && \text{(commutativity)}\\
&= ((\overline{x} + x)y + x(\overline{y} + y))\overline{z} && \text{(distributivity)}\\
&= (y + x)\overline{z} && \text{(inverse \& null)}\\
&= y\overline{z} + x\overline{z} && \text{(distributivity)}
\end{aligned}$$

Karnaugh Maps

Karnaugh Map is a graphical method for simplifying a Boolean expression.

Basic idea:

- Two minterms that differ in exactly one literal can be combined into a simpler term:

$$A\overline{B}CD + A\,\overline{B}\,\overline{C}\,D \longrightarrow A\overline{B}D$$

- Put all minterms in a table, ordered such that neighboring cells differ in exactly one literal.

- Visually locate blocks of minterms that can be combined, which are as large as possible.

Karnaugh Maps: Example

A	*B*	*C*	*D*	*F*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

AB \ *CD*	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	0	0
10	1	1	0	1

$$F = \overline{A}D + \overline{C}D + A\,\overline{B}\,\overline{D}$$

Karnaugh Maps: Rules

- Choose blocks to cover all the ones, but none of the zeros.
- Blocks are allowed to overlap.
- Blocks must:
  - be rectangular
  - have lengths of 1, 2 or 4
  - be as large as possible
- Use as few blocks as possible.
- Wrap around: top – bottom, left – right, corners

Karnaugh Maps: Example 2

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	0
1	1	1	0	1

$$F = \overline{B} + \overline{A}C + A\overline{C}$$

Functional Completeness

- Every Boolean function can be expressed as a sum of products of literals (DNF).

  → The set of operators $\{\overline{}, \cdot, +\}$ is *functionally complete*.

- The sets $\{\overline{}, \cdot\}$ and $\{\overline{}, +\}$ are also functionally complete.

- The operators NAND ($\overline{xy}$) and NOR ($\overline{x+y}$) are each functionally complete on their own.

Gates en Circuits

Digital Circuits

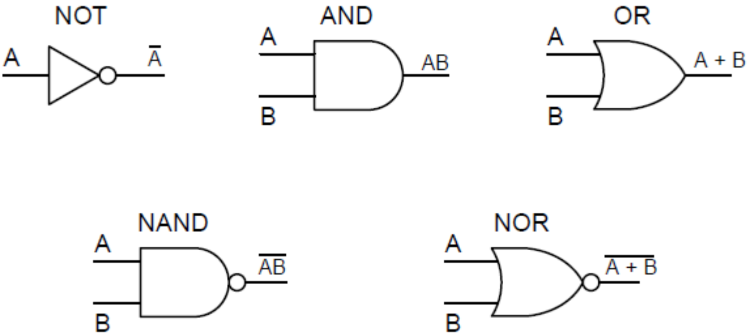
Physical implementation of digital logic:

Boolean values	$\rightarrow$	signals in electronic circuit
Boolean operators	$\rightarrow$	gates

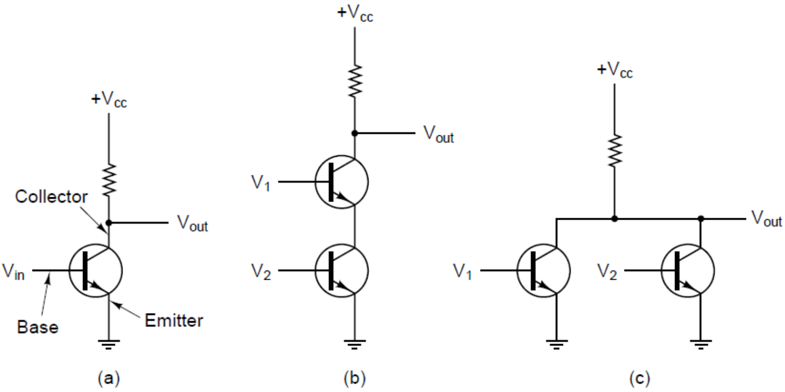
Completeness: every Boolean function can be expressed with the help of a complete set of operators, for example $\{\bar{\ }, \cdot, +\}$ or $\{\text{NAND}\}$.

Therefore: every Boolean function can be implemented as a *circuit* by connecting gates from a complete set together!

Gates


NOT
A
$\overline{A}$
AND
A
B
AB
OR
A
B
A + B
NAND
A
B
$\overline{AB}$
NOR
A
B
$\overline{A + B}$

How are Gates Implemented?

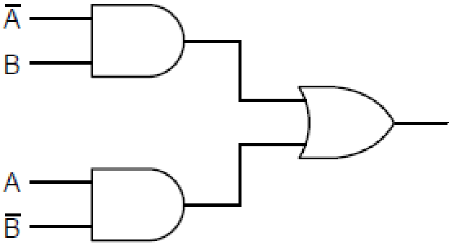


NOT-gate NAND-gate NOR-gate

XOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

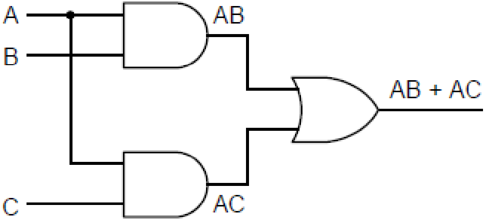
$$A \oplus B = \overline{A}B + A\overline{B}$$



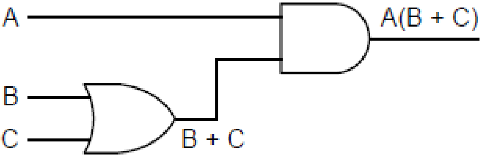
$\overline{A}B + A\overline{B}$

Equivalence of Circuits

$AB + AC$



$A(B + C)$



Combinational and Sequential Logic

Combinational circuit

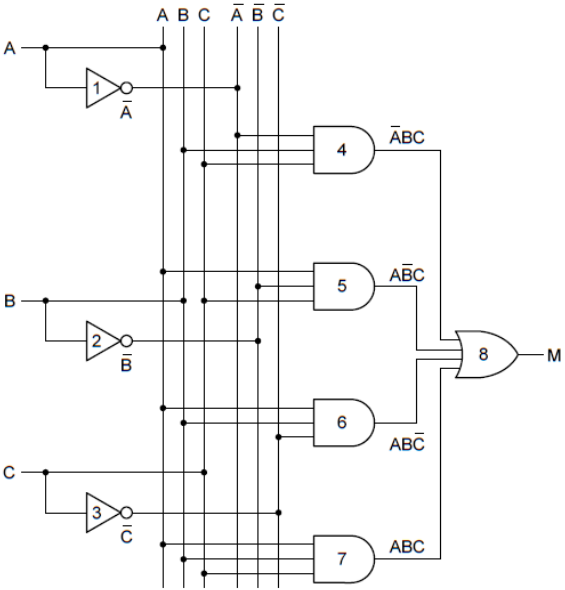
- output of circuit only depends on input (at the current time)
- implements a Boolean function
- acyclic circuit (mostly)

Sequential circuit

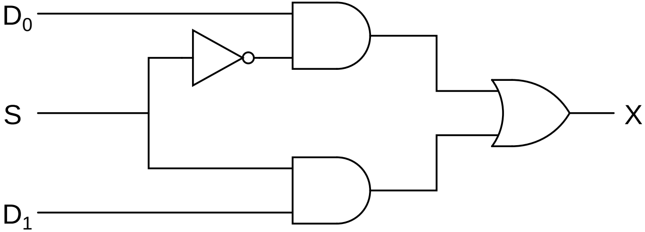
- output of circuit depends on input *and* internal state
- implements a finite state machine
- cyclic circuit: feedback
- "circuit with memory"

Majority

A	*B*	*C*	*M*
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



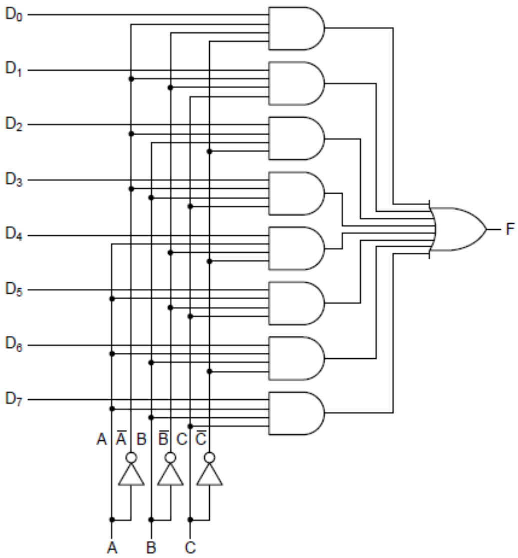
Multiplexer: choose between signals



Circuit with data inputs D_0 en D_1 and control input S.
If $S = 0$, the signal from D_0 is passed to the output, else D_1 is.

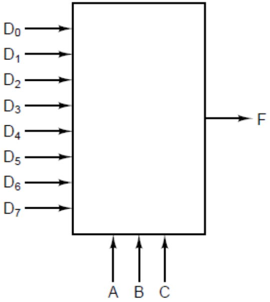
Output $X = \overline{S} \cdot D_0 + S \cdot D_1$

Multiplexer with 8 inputs



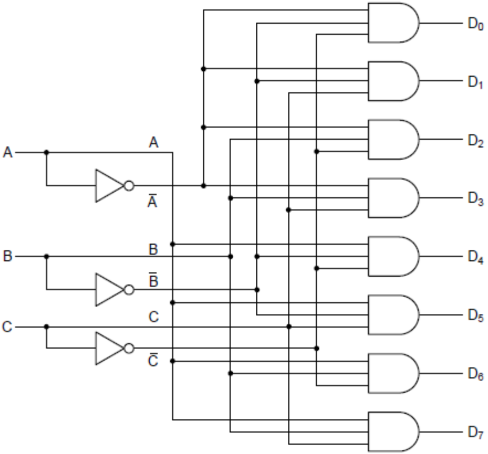
General Multiplexer

In general: a multiplexer with n control inputs selects one of the input signals $D_0, \ldots, D_{2^n-1}$ and passes this to the output.



Decoder

The binary number on the *n* inputs determines which of the 2^n outputs will become active (output a "1").



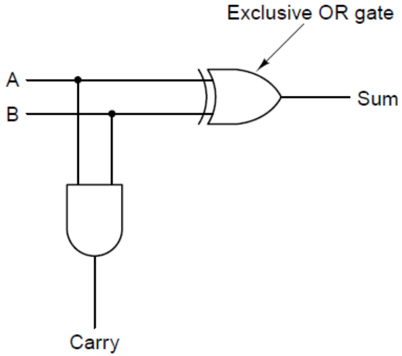
Half Adder

Adding two 1-bit numbers *A* en *B*:

A	0	1	0	1
B	0 +	0 +	1 +	1 +
	00	01	01	10

$$\text{Sum} = A \oplus B$$

$$\text{Carry} = AB$$

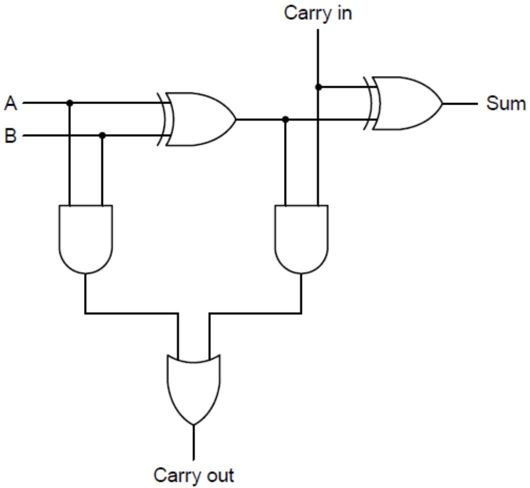


Full Adder

Adding 1-bit numbers with carry-in.

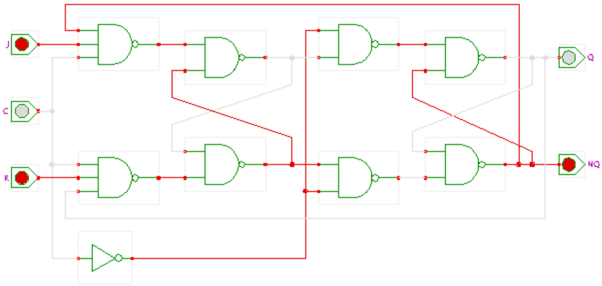
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = A \oplus B \oplus C_{in}$$
$$\text{Carry} = AB + (A \oplus B) \cdot C_{in}$$



Hades

- Practical assignment will be done in *Hades*, the *Hamburg Design System*.
- “Framework for interactive simulation”
- Can be downloaded from https://tams.informatik.uni-hamburg.de/applets/hades/



Summary

Boolean Algebra

- Boolean functions and expressions
- Disjunctive normal form
- Laws of Boolean algebra
- Karnaugh maps
- Functional completeness

Gates and circuits

- Physical implementation of digital logic
- Combinational and sequential circuits
- Gate symbols
- Circuits: xor, majority, multiplexer, decoder, half adder, full adder.