Electronics

Introducction to Synthetic Biology

E Navarro
A Montagud
P Fernandez de Cordoba
JF Urchueguía

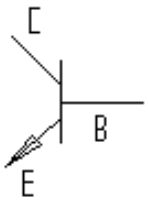
InterTech

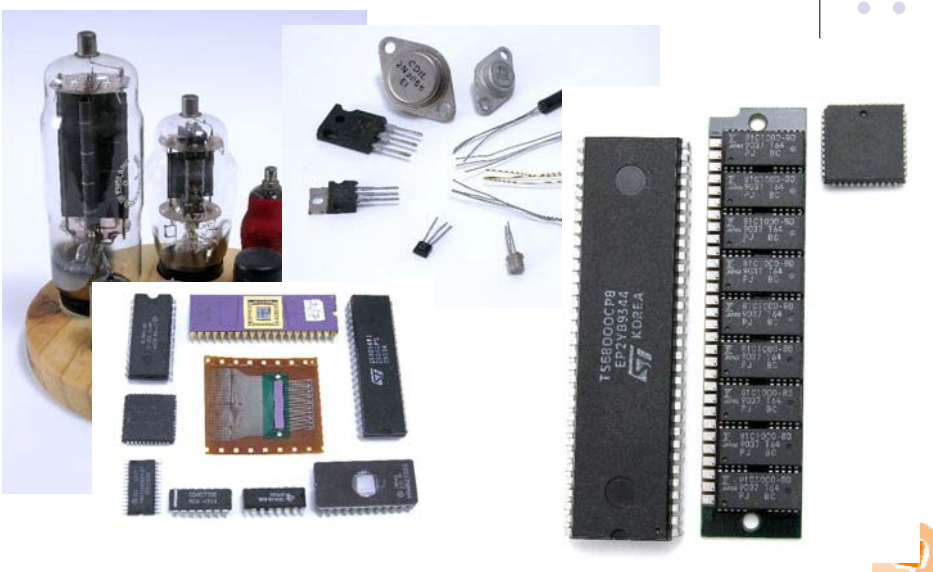
Overview

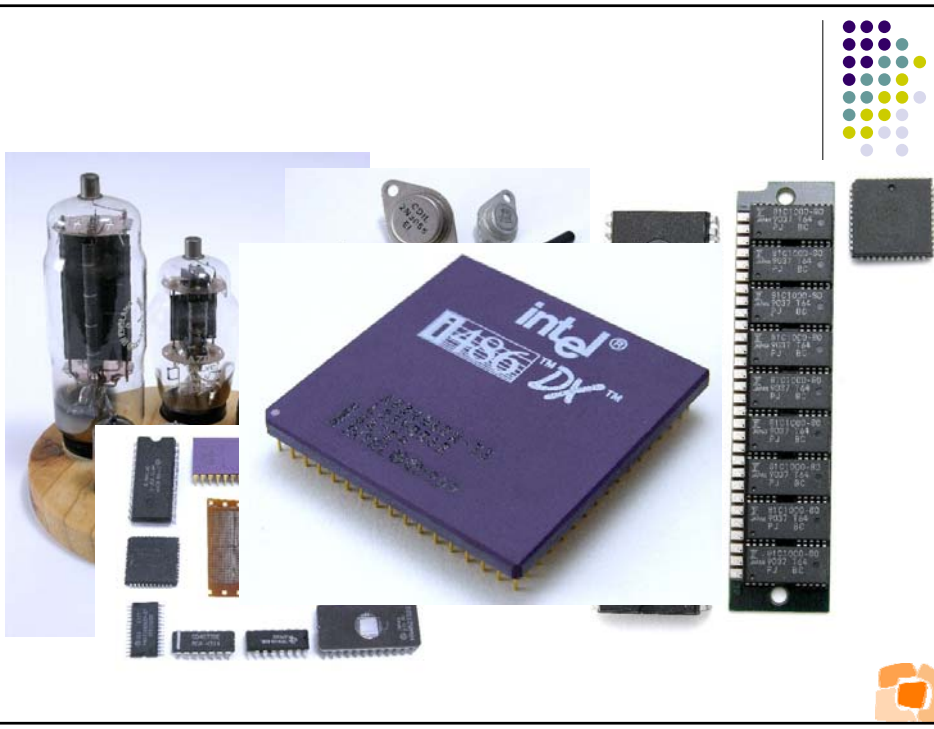
- Introduction
- Boolean algebras
- Logical gates
- Representation of boolean functions
- Karnaugh maps

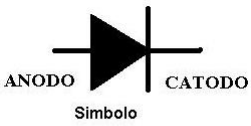
A BRIEF HISTORY

Dryden Flight Research Center E49-0053 Photographed 10/49
Early "computers" at work. NASA photo

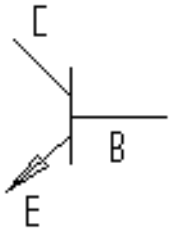


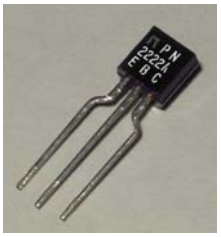

TS68000CP8
EP2V8935A4
KOREA


intel
i386 DX



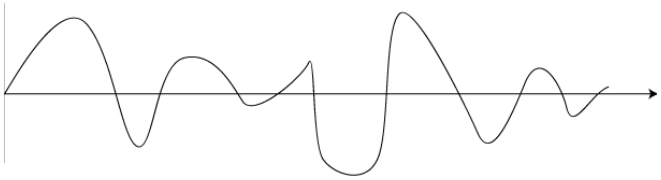
Simbolo



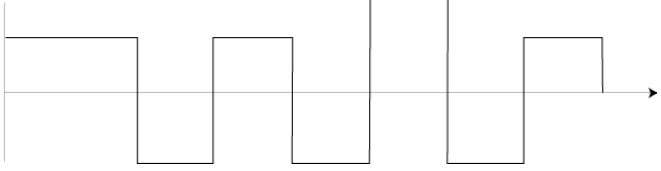


ANALOG SYSTEMS

Analog Signal



Digital Signal

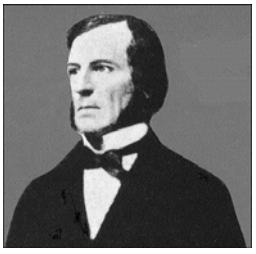


- Analog
  - Continuous
  - Can take on any value in a given range
  - Very susceptible to noise
- Digital
  - Discrete
  - Can only take on certain values in a given range
  - Can be less susceptible to noise

BOOLEAN ALGEBRAS

- George Boole:

An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities (1854)



Algebra: a strange mathematical structure between groups and vectors

Claude Elwood Shannon was the first to use Boolean Algebra to solve problems in electronic circuit design. (1938)

Basis of this algebra

- All the elements of this algebra have values of 0 or 1.

- Three operators:
  - -OR written as +, as in $A + B$
  - - AND written as •, as in $A \cdot B$
  - -NOT written as an overline, as in $\overline{A}$

Operators: OR

$f_{or}(a,b)=a+b$

- The result of the OR operator is 1 if either of the operands is a 1.
- The only time the result of an OR is 0 is when both operands are 0s.
- OR is like our old pal *addition*, but operates only on binary values

a	*b*	$f_{or}(a,b)$
0	0	0
0	1	1
1	0	1
1	1	1

Truth table

Operators: AND

- The result of an AND is a 1 only when both operands are 1s.
- If either operand is a 0, the result is 0.
- AND is like our old nemesis *multiplication*, but operates on binary values.

$f_{and}(a,b)=a \cdot b$

a	b	$f_{and}(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

Operators: NOT

- NOT is a *unary* operator – it operates on only one operand.
- NOT *negates* it's operand.
- If the operand is a 1, the result of the NOT is a 0.

$f_{not}(a)= \overline{a}$

a	b	$f_{not}(a)$
0	-	1
0	-	1
1	-	0
1	-	0

Truth table

Equations

Boolean algebra uses equations to express relationships. For example:

$$X = A \cdot (\overline{B} + C)$$

This equation expressed a relationship between the value of ***X*** and the values of ***A***, ***B*** and ***C***.

Quiz (already?)

What is the value of each X:

$$X_1 = 1 \cdot (0 + 1)$$

$$X_2 = A + \overline{A}$$

$$X_3 = A \cdot \overline{A}$$

$$X_4 = X_4 + 1$$

Laws of Boolean Algebra

Just like in *good old algebra*, Boolean Algebra has postulates and identities.

We can often use these laws to reduce expressions or put expressions in to a more desirable form.

Basic Postulates of Boolean Algebra

- Using just the basic postulates – everything else can be derived.

Commutative laws
Distributive laws
Identity
Inverse

Identity Laws

$$A + 0 = A$$

$$A \cdot 1 = A$$

Inverse Laws

$$A + \overline{A} = 1$$

$$A \cdot \overline{A} = 0$$

Commutative Laws

$$A + B = B + A$$

$$A \cdot B = B \cdot A$$

Distributive Laws

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

$$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$$

Other Identities

Can be derived from the basic postulates.

Laws of Ones and Zeros

Associative Laws

DeMorgan's Theorems

Zero and One Laws

$A + 1 = 1$ Law of Ones

$A \cdot 1 = A$

$A + 0 = A$ Law of Zeros

$A \cdot 0 = 0$

Associative Laws

$$A+(B+C)=(A+B)+C$$

$$A\cdot(B\cdot C)=(A\cdot B)\cdot C$$

DeMorgan's Theorems

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

A	B	A+B	$\overline{A}$	$\overline{B}$	$\overline{A+B}$	$\overline{A}\,.\,\overline{B}$
0	0	0	1	1	**1**	**1**
0	1	1	1	0	**0**	**0**
1	0	1	0	1	**0**	**0**
1	1	1	0	0	**0**	**0**

DeMorgan's Theorems

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

A	B	AB	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	0	**1**	1	1	**1**
0	1	0	**1**	1	0	**1**
1	0	0	**1**	0	1	**1**
1	1	1	**0**	0	0	**0**

Other Operators

- Boolean Algebra is defined over the 3 operators AND, OR and NOT.
  - this is a *functionally complete set*.
- There are other useful operators:
  - NOR: is a 0 if either operand is a 1
  - NAND: is a 0 only if both operands are 1
  - XOR: is a 1 if the operands are different.
- NOTE: NOR or NAND is (by itself) a functionally complete set!

Boolean Functions

- Boolean functions are functions that operate on a number of Boolean variables.
- The result of a Boolean function is itself either a 0 or a 1.
- Example: $f(a,b) = a+b$

Question

- How many Boolean functions of 1 variable are there?
- We can answer this by listing them all!

$$f_1(x) = x$$
$$f_2(x) = \overline{x}$$
$$f_3(x) = 0$$
$$f_4(x) = 1$$

Tougher Question

- How many Boolean functions of 2 variables are there?
- It's much harder to list them all, but it is still possible…

Alternative Representation

- We can define a Boolean function showing it by means of using algebraic operations.
- We can also define a Boolean function by listing the value of the function for all possible inputs.

Truth Tables

a	*b*	OR	AND	NOR	NAND	XOR
0	0	0	0	1	1	0
0	1	1	0	0	1	1
1	0	1	0	0	1	1
1	1	1	1	0	0	0

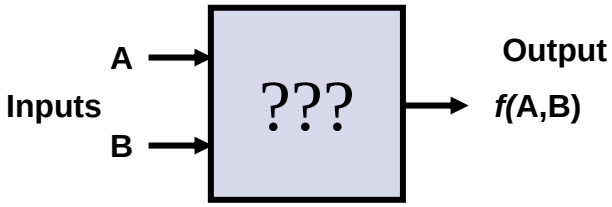
Truth Table for (X+Y)·Z

X	Y	Z	(X+Y)·Z
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	0
1	1	1	1

Gates

- Digital logic circuits are electronic circuits that are implementations of some Boolean function(s).
- A circuit is built up of *gates*, each *gate* implements some simple logic function.

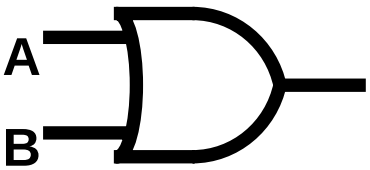
A Gate



Gates compute something!

- The output depends on the inputs.
- If the input changes, the output might change.
- If the inputs don't change – the output does not change.

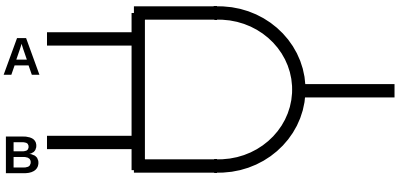
An OR gate



A+B

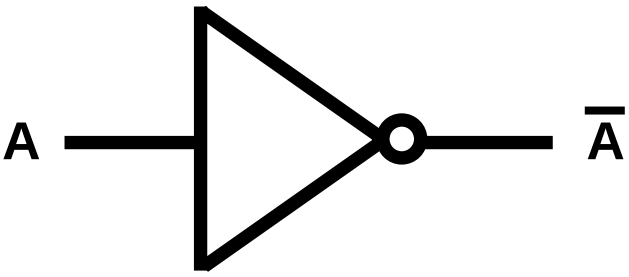
a	*b*	$f_{or}(a,b)$
0	0	0
0	1	1
1	0	1
1	1	1

An AND gate

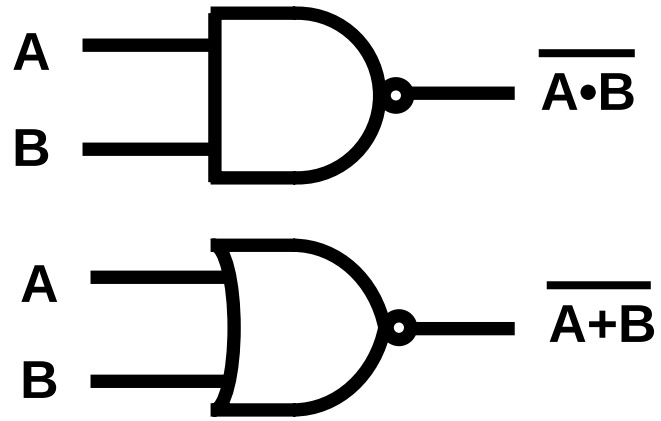


a	*b*	$f_{and}(a,b)$
0	0	0
0	1	0
1	0	0
1	1	1

A Not gate



A NAND and NOR gate



XOR and XNOR

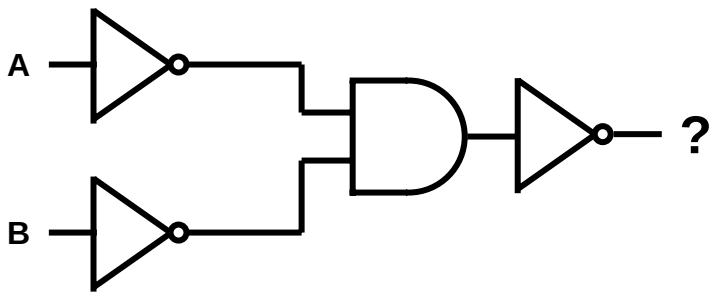
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Combinational Circuits

- We can put gates together into circuits
  - output of some gates are inputs of other ones.
- We can design a circuit that represents any Boolean function!

A Simple Circuit



Truth Table for our circuit

a	b	$\overline{a}$	$\overline{b}$	$\overline{a} \bullet \overline{b}$	$\overline{\overline{a} \bullet \overline{b}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Alternative Representations

- Any of these can express a Boolean function:

Boolean Equation

Circuit (Logic Diagram)

Truth Table

Implementation

- A logic diagram is used to design an *implementation* of a function.
- The implementation involves the specific gates and the way they are connected.
- We can buy a bunch of gates, put them together (along with a power source) and build up a machine.

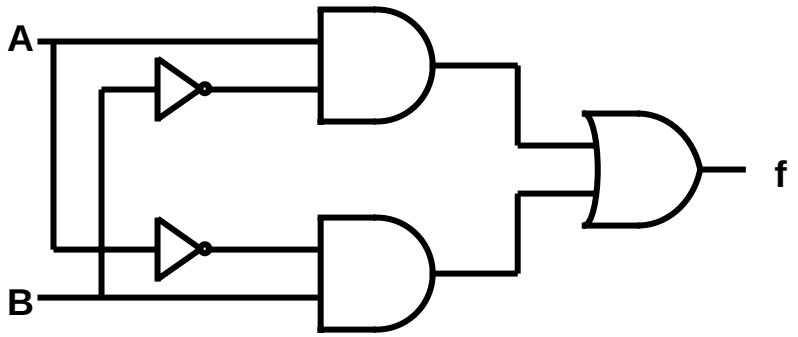
Function Implementation

- Given a Boolean function expressed as a truth table or Boolean Equation, there are many possible implementations.
- The actual implementation depends on what kind of gates are available.
- In general it would be convenient to minimize the number of gates.

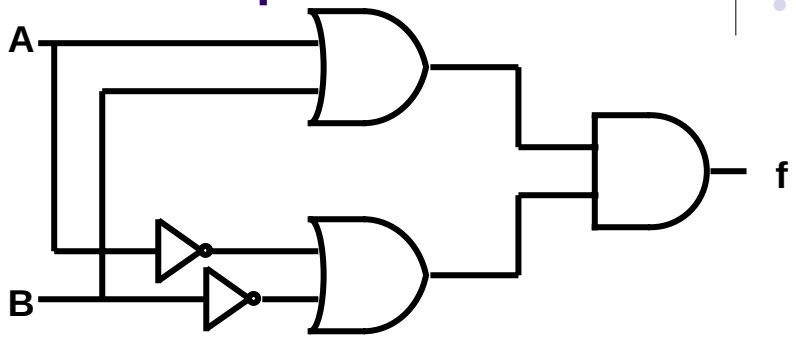
Example: $f = A \bullet \overline{B} + \overline{A} \bullet B$

A	B	$A \bullet \overline{B}$	$\overline{A} \bullet B$	f
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

One Implementation $f = A \bullet \overline{B} + \overline{A} \bullet B$



Another Implementation



$$f = A \bullet \overline{B} + \overline{A} \bullet B = (A + B) \bullet (\overline{A} + \overline{B})$$

Proof it's the same function

$A \bullet \overline{B} + \overline{A} \bullet B =$	
$\overline{(A \bullet \overline{B})} \bullet \overline{(\overline{A} \bullet B)} =$	DeMorgan's Law
$\overline{(\overline{A} + B)} \bullet \overline{(A + \overline{B})} =$	DeMorgan's Laws
$\overline{((\overline{A} + B) \bullet A)} + \overline{((\overline{A} + B) \bullet \overline{B})} =$	Distributive
$\overline{(\overline{A} \bullet A + B \bullet A)} + \overline{(\overline{A} \bullet \overline{B} + B \bullet \overline{B})} =$	Distributive
$\overline{(B \bullet A)} + \overline{(\overline{A} \bullet \overline{B})} =$	Inverse, Identity
$\overline{(B \bullet A)} \bullet \overline{(\overline{A} \bullet \overline{B})} =$	DeMorgan's Law
$(\overline{B} + \overline{A}) \bullet (A + B)$	DeMorgan's Laws

Karnaugh maps

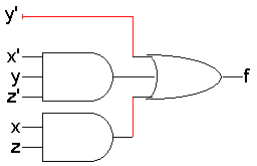
- The Karnaugh map was invented in 1952 by Edward W. Veitch and developed further 1953 by Maurice Karnaugh.
- Karnaugh maps make use of the human brain's excellent pattern-matching capability to decide which terms should be combined to get the simplest expression.
- Karnaugh maps permit the rapid identification and elimination of potential race hazards, something that boolean equations alone cannot do.
- A Karnaugh map is an excellent aid for simplification of up to six variables, but with more variables it becomes hard even for our brain to discern optimal patterns.
- For problems involving more than six variables, solving the boolean expressions is preferred to use of a Karnaugh map.

Review: Standard forms of expressions

- Expressions can be written in many ways, but some ways are more useful than others
- A sum of products (SOP) expression contains:
  - Only OR (sum) operations at the "outermost" level
  - Each term that is summed must be a product

$$f(x,y,z) = y' + x'yz' + xz$$

- The advantage is that any sum of product expression can be implemented using a two-level circuit
  - literals and their complements at the "0th" level
  - AND gates at the first level
  - a single OR gate at the second level
- This diagram uses some shorthands…
  - NOT gates are implicit
  - literals are reused
  - this is *not* okay in LogicWorks!



Terminology: Minterms

- A minterm is a special product of literals, in which each input variable appears exactly once.
- A function with n variables has 2^n minterms (since each variable can appear complemented or not)
- A three-variable function, such as f(x,y,z), has $2^3 = 8$ minterms:

x'y'z'	x'y'z	x'yz'	x'yz
xy'z'	xy'z	xyz'	xyz

- Each minterm is true for exactly one combination of inputs:

Minterm	Is true when…	Shorthand
x'y'z'	x=0, y=0, z=0	m_0
x'y'z	x=0, y=0, z=1	m_1
x'yz'	x=0, y=1, z=0	m_2
x'yz	x=0, y=1, z=1	m_3
xy'z'	x=1, y=0, z=0	m_4
xy'z	x=1, y=0, z=1	m_5
xyz'	x=1, y=1, z=0	m_6
xyz	x=1, y=1, z=1	m_7

Terminology: Sum of minterms form

- Every function can be written as a sum of minterms, which is a special kind of sum of products form
- The sum of minterms form for any function is *unique*
- If you have a truth table for a function, you can write a sum of minterms expression just by picking out the rows of the table where the function output is 1.

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

f = x'y'z' + x'y'z + x'yz' + x'yz + xyz'
  = $m_0 + m_1 + m_2 + m_3 + m_6$
  = $\Sigma m(0,1,2,3,6)$

f' = xy'z' + xy'z + xyz
  = $m_4 + m_5 + m_7$
  = $\Sigma m(4,5,7)$

f' contains all the minterms not in f

Can we know the number of functions that we have if we know the number of variables?

Re-arranging the truth table

- A two-variable function has four possible minterms. We can re-arrange these minterms into a Karnaugh map.

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

- Now we can easily see which minterms contain common literals.
  - Minterms on the left and right sides contain y' and y respectively.
  - Minterms on the top and bottom rows contain x' and x respectively.

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

x \ y	y'	y
x'	x'y'	x'y
x	xy'	xy

Karnaugh map simplifications

- Imagine a two-variable sum of minterms:

  x'y' + x'y

- Both of these minterms appear in the top row of a Karnaugh map, which means that both of them contain the literal x'.

		y
	x'y'	x'y
X	xy'	xy

- What happens if you simplify this expression using Boolean algebra?

x'y' + x'y	= x'(y' + y)	[Distributive]
	= x' • 1	[y + y' = 1]
	= x'	[x • 1 = x]

More two-variable examples

- Other example expression is x'y + xy.
  - Both minterms appear in the right side, where y is uncomplemented.
  - Thus, we can reduce x'y + xy just to y.

		y
	x'y'	x'y
X	xy'	xy

- How about x'y' + x'y + xy?
  - We have x'y' + x'y in the top row, corresponding to x'.
  - There's also x'y + xy in the right side, corresponding to y.
  - This whole expression can be reduced to x' + y.

		y
	x'y'	x'y
X	xy'	xy

A three-variable Karnaugh map

- For a three-variable expression with inputs x, y, z, the arrangement of minterms is more tricky:

X \ YZ	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

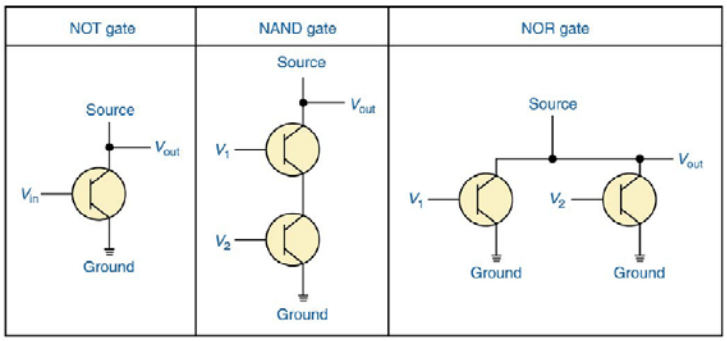
X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

- Another way to label the K-map (use whichever you like):

X \ Y	Z			
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'

X \ Y	Z			
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6

Not all the gates are equally easy to build



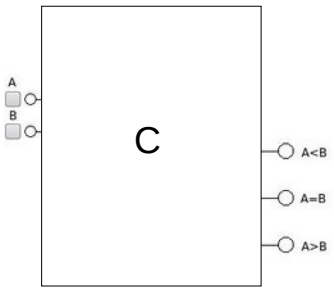
Digital Circuits

- Combinatorial logic
  - Results of an operation depend *only* on the present inputs of the operation
  - Uses: perform arithmetic, control data movement, comparison of values for decision making
- Sequential logic
  - Results depend on both the inputs of the operation *and* the result of the previous operation
  - Uses: counter

Comparators

It compares two input signals.

Inputs		Outputs		
A	*B*	*A* < *B*	*A* = *B*	*A* > *B*
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

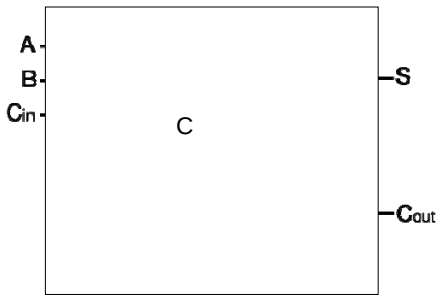


Based on this it can built a filter, or other more important devices like a analog digital converter or controllers

Adder

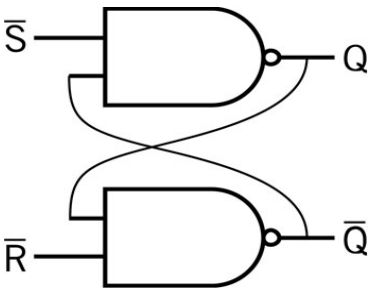
It performs the additioning operation of three binary digits.

Input			Output	
A	B	C_i	C_o	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		



Sequential Logic Circuits

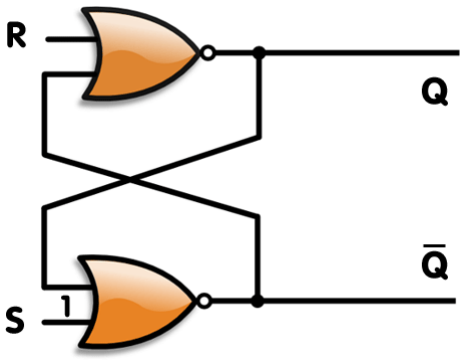
- Output depends on
  - Input
  - Previous state of the circuit
- Flip-flop: basic memory element
- State table: output for all combinations of input and previous states



SR Latch

SR latch operation

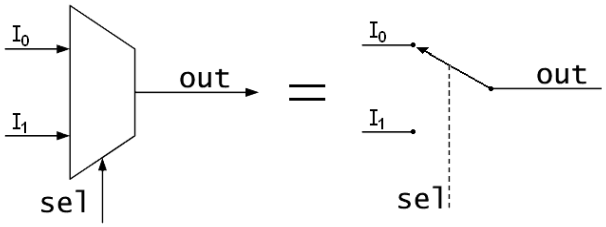
S	R	Action
0	0	Keep state
0	1	Q = 0
1	0	Q = 1
1	1	Unstable combination,



SR Latch

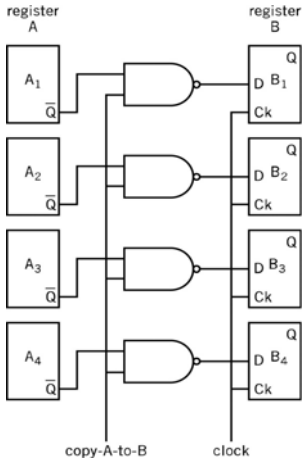
Other electronic devices

- Multiplexer:



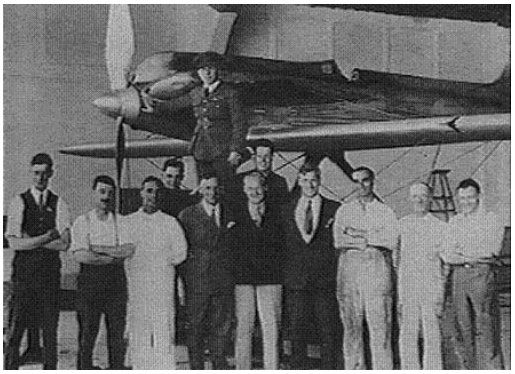
Register COPY Operation

- Uses both sequential and combinational logic



Where are we?

We are



Bibliography

- Integrated Electronics: Analog and Digital Circuits and Systems. J Millman, CC Halkias. McGraw-Hill.
- Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets. JR Gregg. IEEE Press Understanding Science & Technology Series.
- Introduction to Logic Design. AB Marcovitz. McGraw-Hill.

InterTech