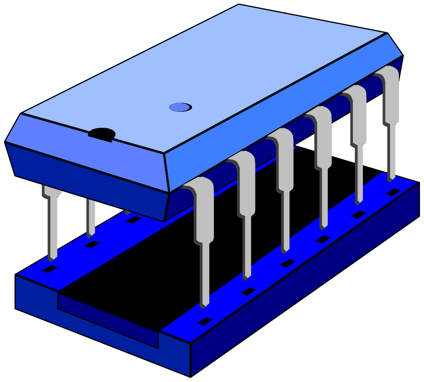


BOOLEAN ALGEBRA AND LOGIC

Bernhard Kainz (with thanks to **A. Gopalan, N. Dulay** and **E. Edwards**)

b.kainz@imperial.ac.uk

Learning Objectives

- At the end of this lecture you should:
  - Understand how logic relates to computing problems
  - Be able to represent Boolean logic problems as:
    - Truth tables
    - Logic circuits
    - Boolean algebra

What is Logic?

- Dictionary definitions (dictionary.com - reduced!)
  - reason or sound judgement
  - a system of principles of reasoning
  - the science that investigates the principles governing correct or reliable inference
- Branch of philosophy
  - Principles of inference
- You use logic all the time in your everyday life

Propositional Logic

- The Ancient Greek philosophers created a system to formalise arguments called propositional logic
- A proposition is a statement that can be TRUE or FALSE
- Propositions can be compounded by means of the operators AND, OR and NOT

Propositional Logic Example

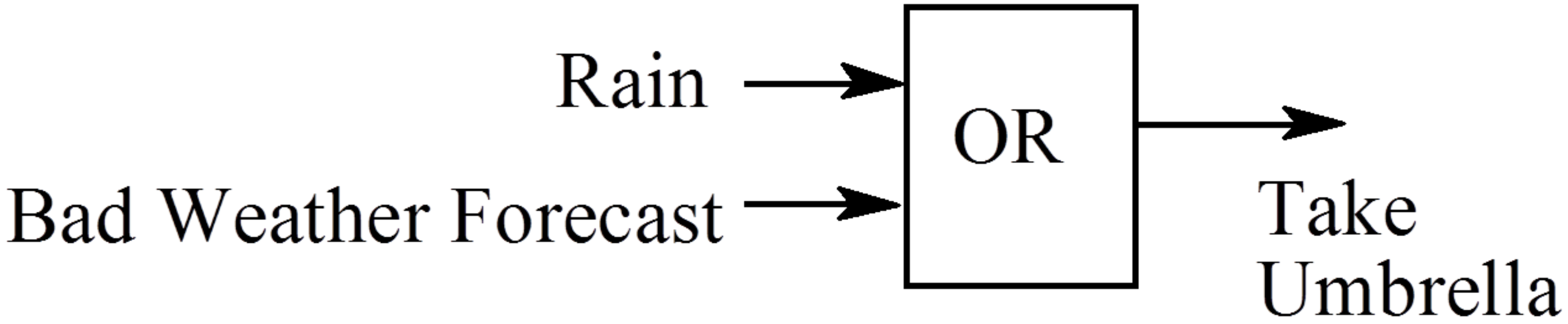
- Propositions may be TRUE or FALSE, for example:
  - It is raining
  - The weather forecast is bad
- A combined proposition example is:
  - It is raining OR the weather forecast is bad

Propositional Logic Example

- Can assign values to propositions, for example: I will take an umbrella if it is raining OR the weather forecast is bad
  - Means that the proposition “I will take an umbrella” is the result of the Boolean combination (OR) between raining and weather forecast being bad. In fact we could write:
    - I will take an umbrella = it is raining OR the weather forecast is bad

Diagrammatic Representation

- Can think of the umbrella proposition as a result that we calculate from the weather forecast and the fact that it is raining by means of a logical OR



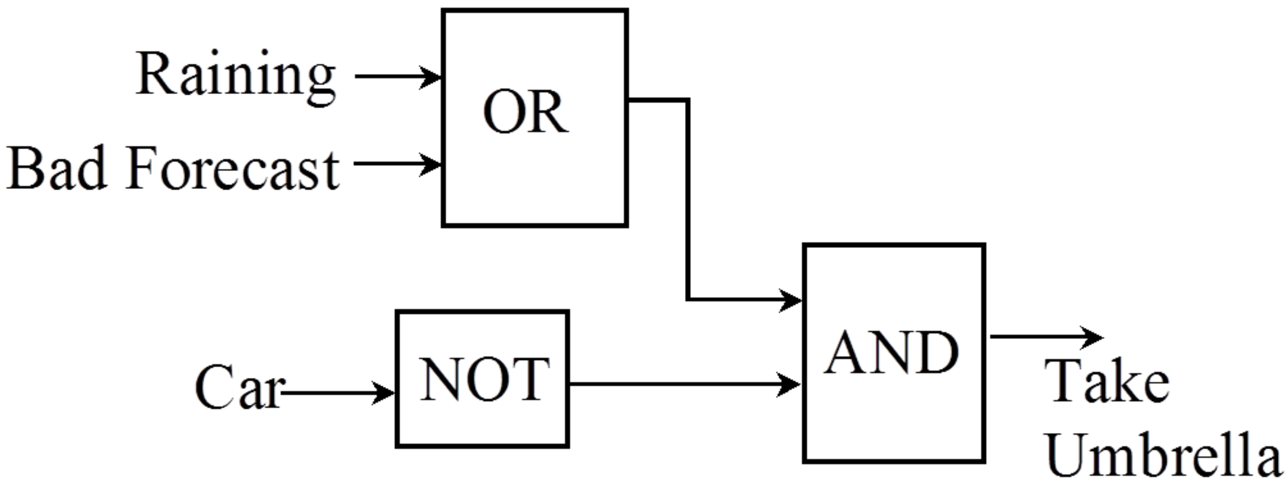
Truth Tables

- Since propositions can only take two values, we can express all possible outcomes of the umbrella proposition by a table

Raining	Bad Forecast	Umbrella
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Complex Propositions

- Can make our propositions more complex, for example:
  - (Take Umbrella) = (NOT (Take Car)) AND ((Bad Forecast) OR (Raining))
- Diagrammatical representation



Boolean Logic

- To perform calculations quickly and efficiently we need a more succinct notation than propositional logic
- Need to have well-defined semantics for all the “operators”, or connectives that we intend to use

- Boolean Algebra satisfies the criterion above
- Named after George Boole
- Provides a system of logical operations
- Rules for combining operations
- Describes their application to binary numbers

George Boole: 1815-1864

Boolean Algebra – Fundamentals

- The truth values are replaced by 1 and 0
  - 1 = TRUE   0 = FALSE
- Propositions are replaced by variables
  - R = it is raining   W = The weather forecast is bad
- Operators are replaced by symbols
  - ' = NOT   + = OR   • = AND

Boolean Algebra – Simplify Propositions

- Recall:
  - (Take Umbrella) = (NOT (Take Car)) AND ((Bad Forecast) OR (Raining))
- Using notations notations, we get:
  - $U = (C') \cdot (W + R)$

Boolean Algebra – Precedence

- Operator Precedence
  - Highest precedence operator is evaluated first

OPERATOR	SYMBOL	PRECEDENCE
NOT	' (¬)	Highest
AND	• (∧)	Middle
OR	+ (∨)	Lowest

math (logic) symbol

- Note that: (C') • (W + R) is not the same as C' • W + R
- Logic operators in, e.g., C:
  - Logical: AND: && OR: || NOT: !
  - (Binary: AND: & OR: | NOT: ~)

Boolean Algebra – Truth Tables

- All possible outcomes of the operators can be written as truth tables

AND

•

A	B	R
0	0	0
0	1	0
1	0	0
1	1	1

OR

+

A	B	R
0	0	0
0	1	1
1	0	1
1	1	1

NOT

'

A	R
0	1
1	0

Boolean Algebra – Truth Tables

- Given any Boolean expression e.g.: U = C' • (W + R)
- We can calculate a truth table for every possible value of the variables on the right hand side
- **For n variables there are 2^n possibilities**

Boolean Algebra – Truth Tables

- Truth table for “Umbrella”
  - U = C' • (W + R)

R	W	C	X1=R+W	X2=C'	U=X1•X2
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	0
Inputs			Partial Results		Outputs

Boolean Algebra – Rules

- **Note: A and B can be any Boolean Expression**

Negation:	Associative:	Commutative:
$(A')' = A$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$	$A \cdot B = B \cdot A$
$A \cdot A' = 0$	$(A + B) + C = A + (B + C)$	$A + B = B + A$
$A + A' = 1$		

Distributive:

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

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

Note the precedence

Boolean Algebra – Rules

Single variables (Idempotent law):

$A \bullet A = A$

$A + A = A$

Simplification rules with 1 and 0:

$A \bullet 0 = 0$

$A \bullet 1 = A$

$A + 0 = A$

$A + 1 = 1$

Boolean Algebra – de Morgan's Rule

$(A + B)' = A' \bullet B'$

$(A \bullet B)' = A' + B'$

as before, A and B can be any Boolean expression

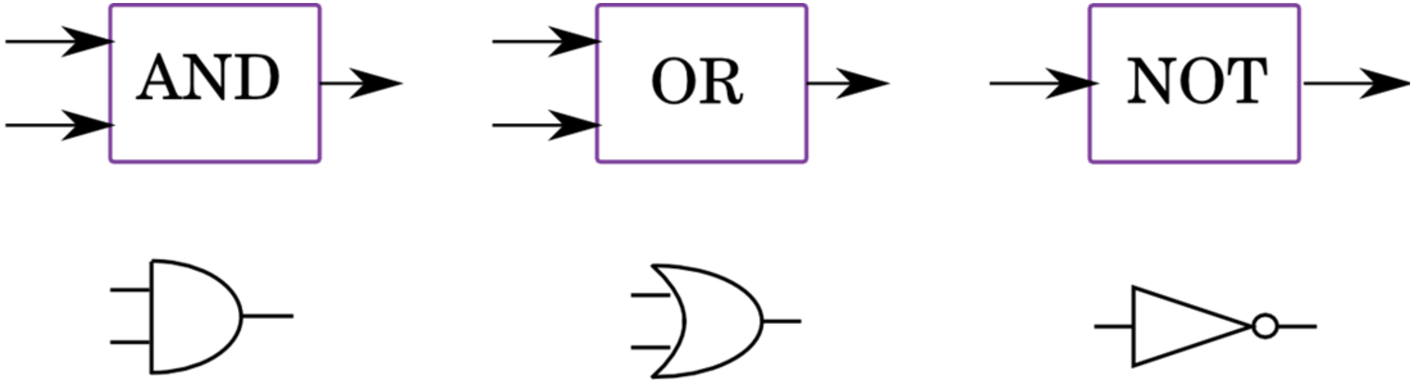
Can generalise to n Boolean variables:

$(A + B + C + D + ...)' = A' \bullet B' \bullet C' \bullet D' \bullet ...$

$(A \bullet B \bullet C \bullet D \bullet ... \bullet X)' = A' + B' + C' + D' + ... + X'$

Boolean Functions – Schematic Representation

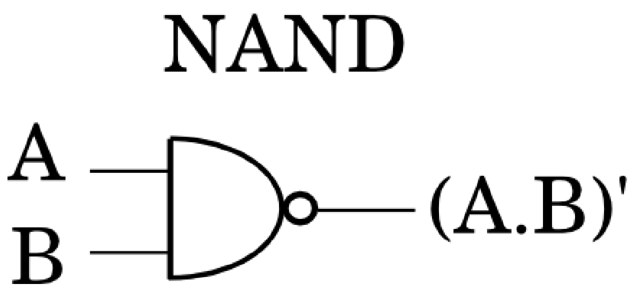
- A standard set of easy-to-recognise symbols is used to represent Boolean functions

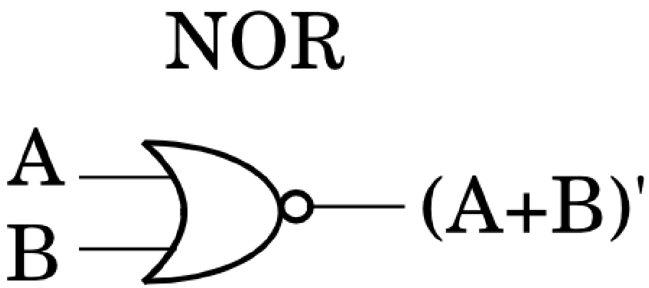


- A circle is all that is required to indicate NOT. The triangle is just to indicate Input/Output direction

Inverting Functions

- A circle can be added to the AND and OR symbol outputs to create their inverted functions – NotAnd (NAND) and NotOr (NOR) gates



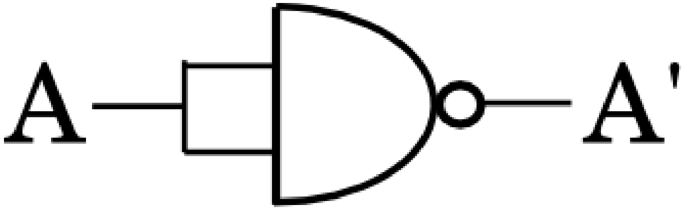


Building Blocks for Circuits

- NAND/NOR are the commonly used building blocks for most circuits
  - NAND / NOR can easily be constructed from transistors
  - NAND is complete
    - A set of Boolean functions f1,f2,… is “complete” if and only if any Boolean function can be generated by a combination these functions
    - Also called “universal gate”

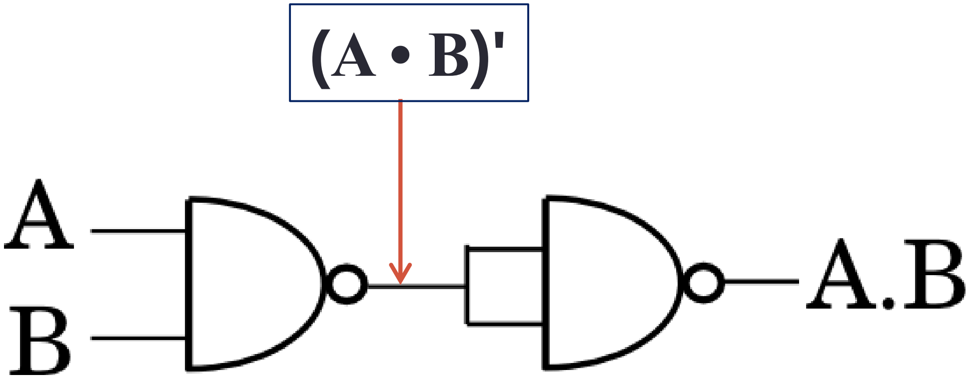
NAND Gate – NOT

- It is possible to build all other gates out of NAND gates
- Create a NOT gate using the Idempotent law:
  - A • A = A therefore (A • A)' = A'



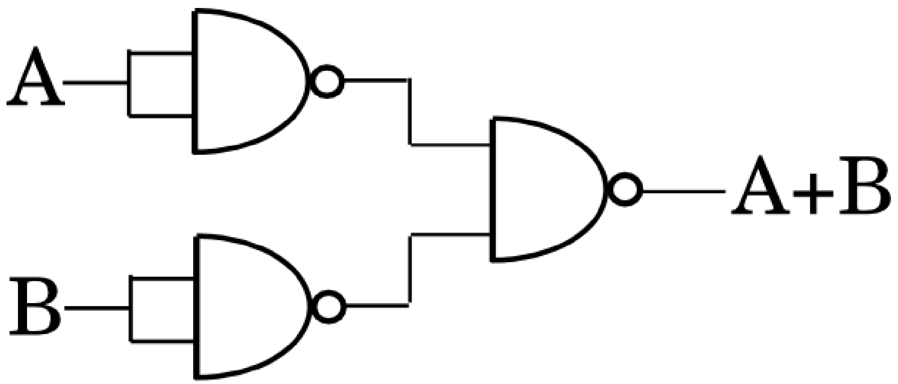
NAND Gate – AND

- Create an AND gate using the Involution law:
  - (A')' = A



NAND Gate – OR / NOR

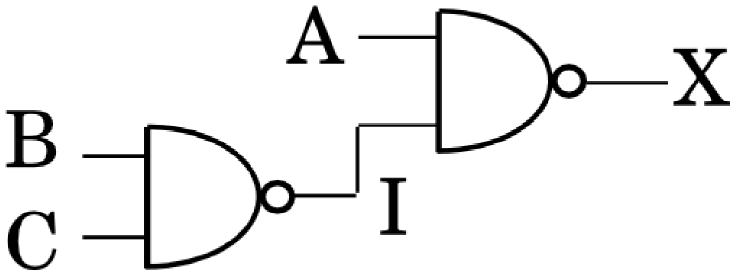
- To make an OR gate we need to apply de Morgan's theorem:
  - A + B = (A' • B')'



- Just invert output to get a **NOR** gate ☺

NAND – Complex Circuits

- Consider two cascading NAND Gates



- What circuit have we created?
  - Use Boolean Algebra to find out
  - $I = (B \cdot C)'$
  - $X = (A \cdot I)' = (A \cdot (B \cdot C)')'$
  - Apply de Morgan's law, we get
  - $X = A' + ((B \cdot C)')' = A' + (B \cdot C)$

NAND – Complex Circuits

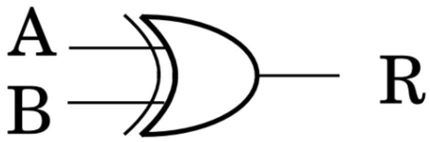
- Truth table for X = A' + (B • C)

A	B	C	B • C	X = A' + (B • C)
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

XOR and XNOR Gates

- Very useful gates

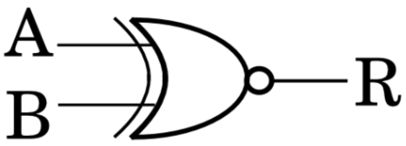
Exclusive Or (XOR)



$$R = A.B' + A'.B$$

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive Nor (XNOR)

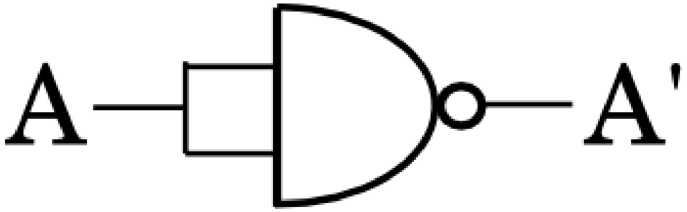


$$R = A'.B' + A.B$$

A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

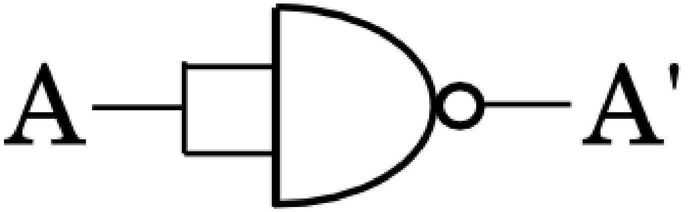
Behind the scenes

- Instantaneous on IC?



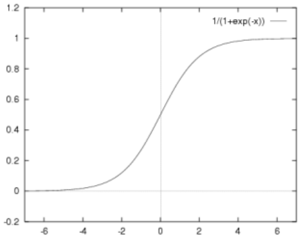
Behind the scenes

- Instantaneous on IC?



V+

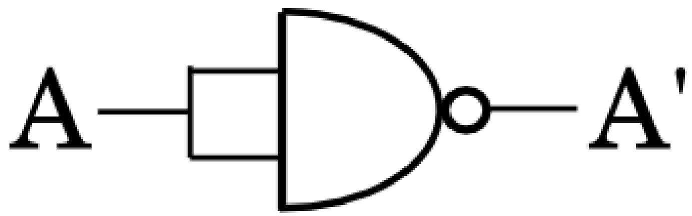
?

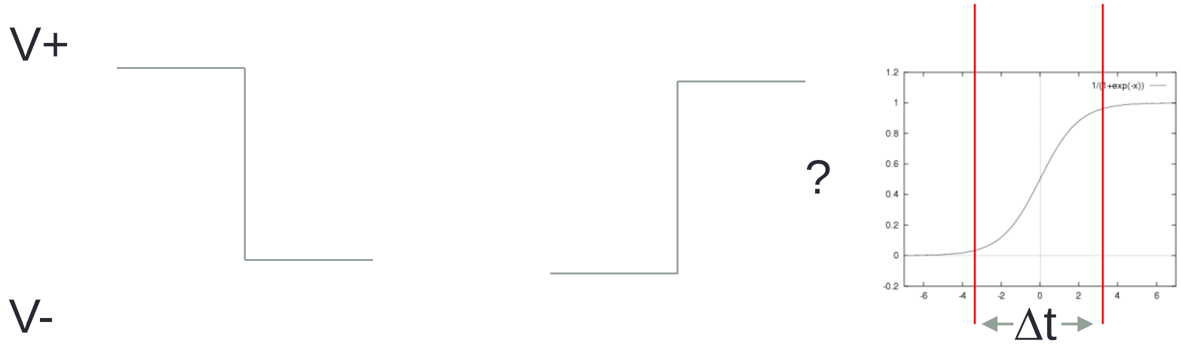


V-

Behind the scenes

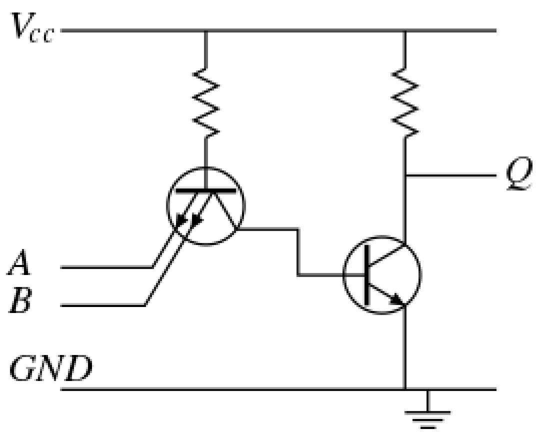
- Time delay and saturation limit state ‘switching’ speed of real in-silico circuits

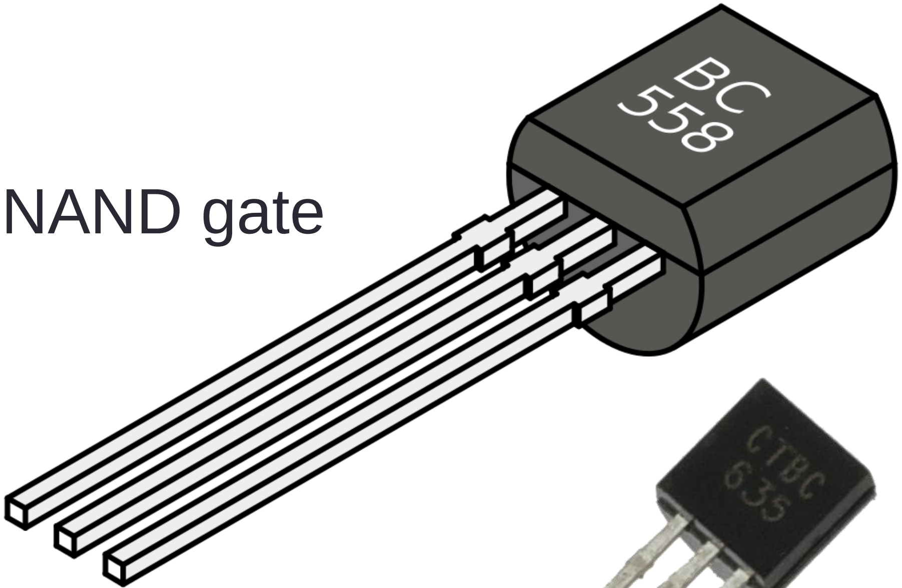


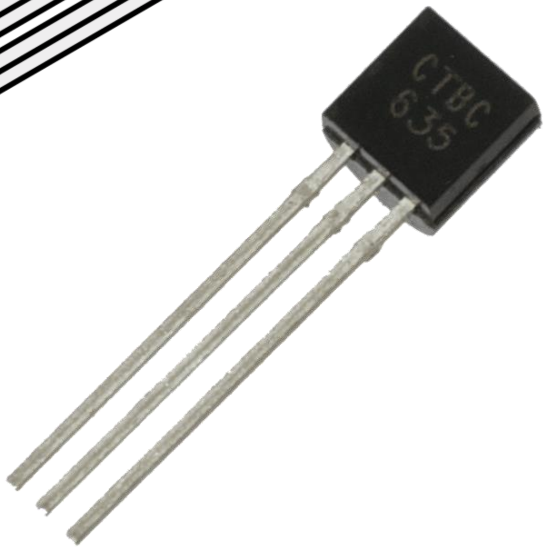


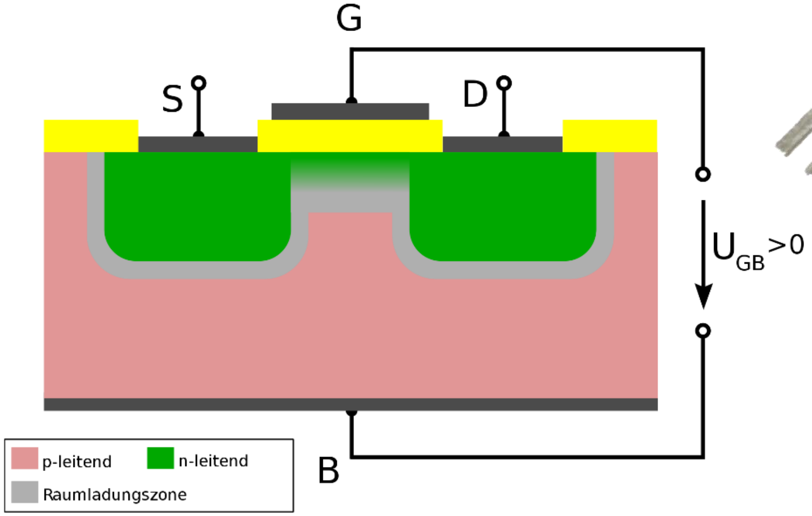
Behind the scenes

- Transistor-transistor logic TTL NAND gate



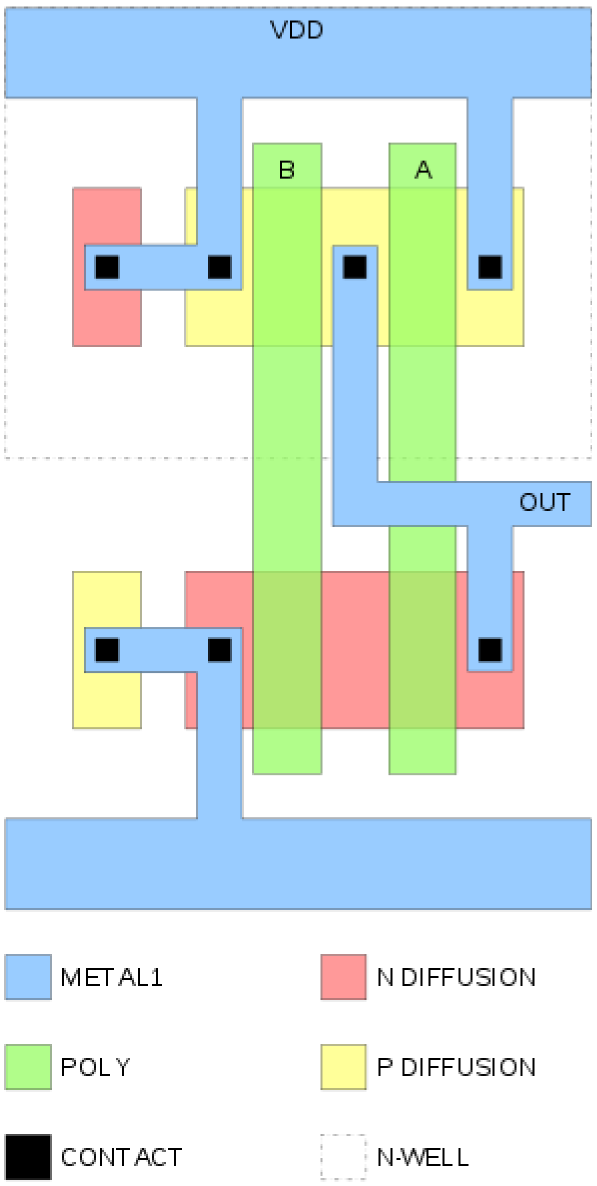






Behind the scenes

- CMOS NAND in silico



Behind the scenes

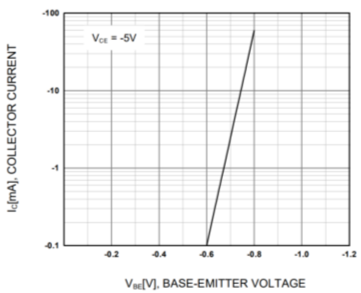


Figure 4. Base-Emitter On Voltage