Breadboard logic

Tobias Eckert, Simon Pirkelmann

imaginärraum

January 13th, 2020

Outline

Boolean logic and basic gates

Binary addition

Memory

Basic logic operation

- Two states **TRUE** and **FALSE** (also written as **1** and **0**)
- Boolean logic describes logical operations
- **NOT**

A	Y
0	1
1	0

- **AND**

A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

- **OR**

A	B	Y
0	0	0
1	0	1
0	1	1
1	1	1

- **XOR**

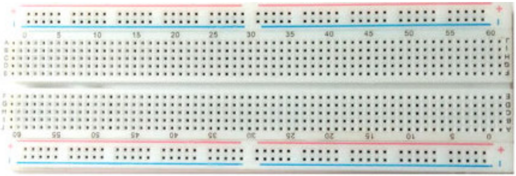
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

- **NAND**

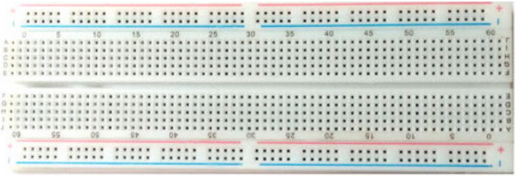
A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

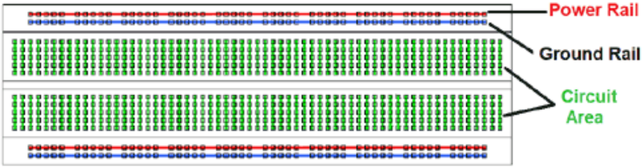
- In electronics boolean states are represented by different voltage levels, e.g. **FALSE** = 0 V, **TRUE** = 5 V

Breadboard

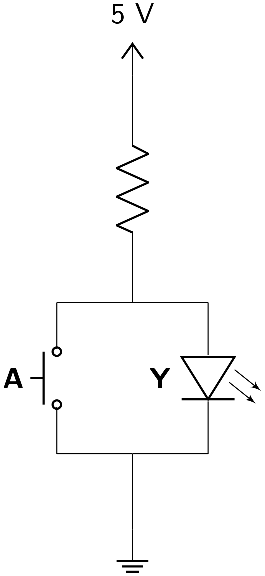


Breadboard



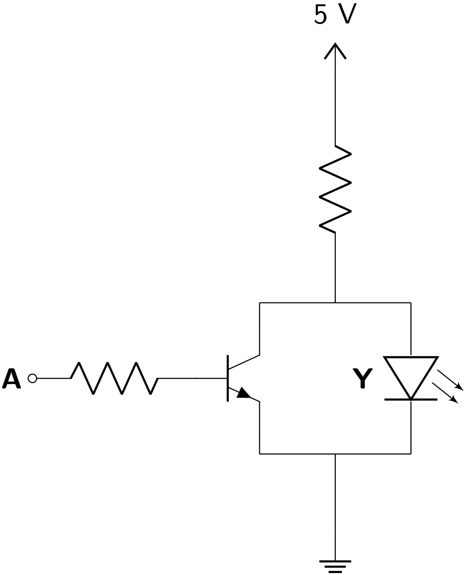


NOT gate (inverter)



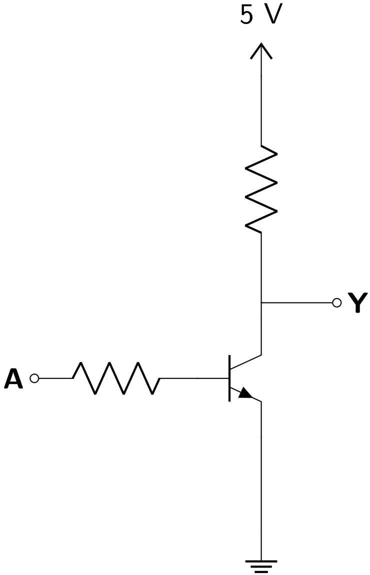
A	Y
0	1
1	0

NOT gate (inverter)



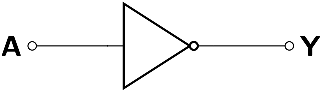
A	Y
0	1
1	0

NOT gate (inverter)

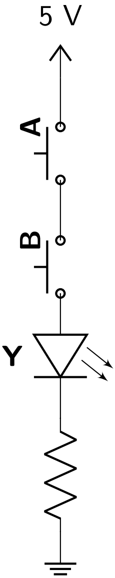


A	Y
0	1
1	0

Symbol:

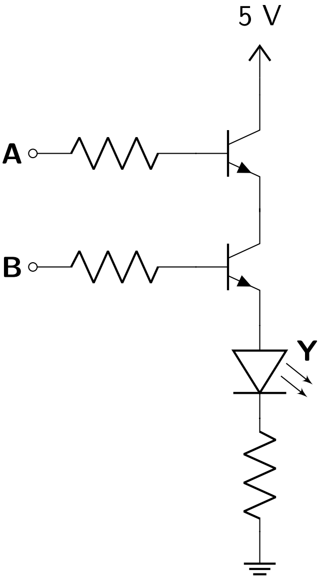


AND gate



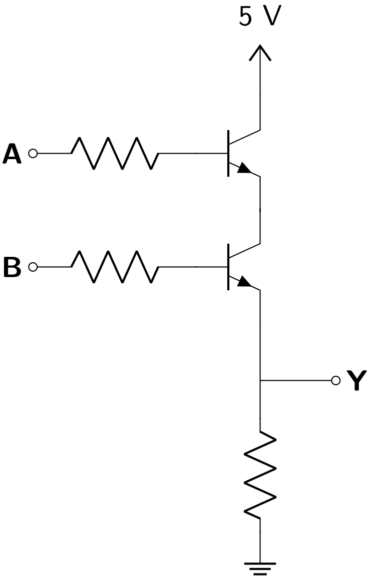
A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

AND gate



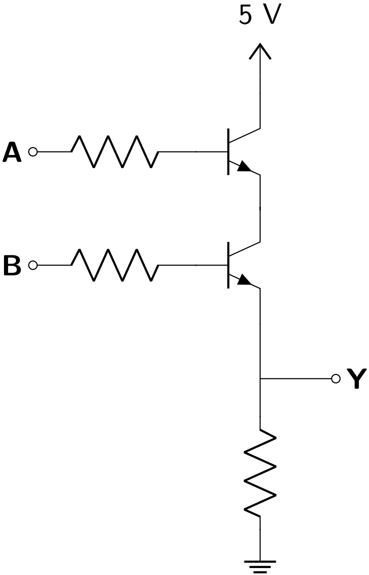
A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

AND gate



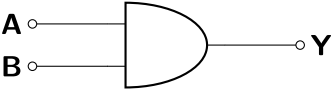
A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

AND gate

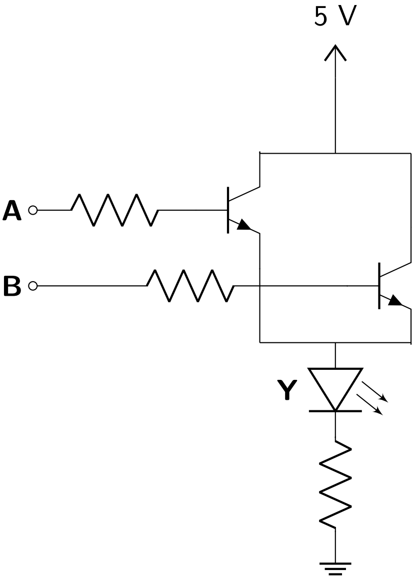


A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

Symbol:

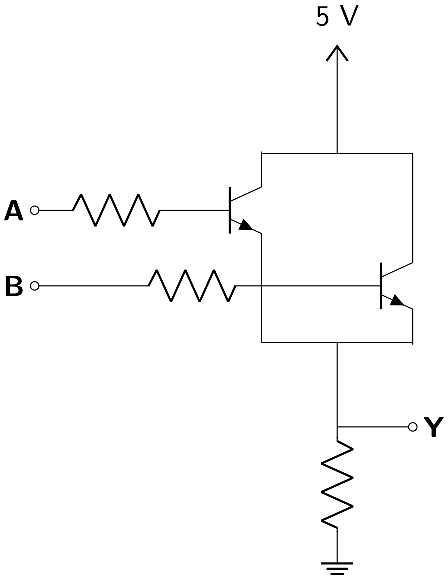


OR gate



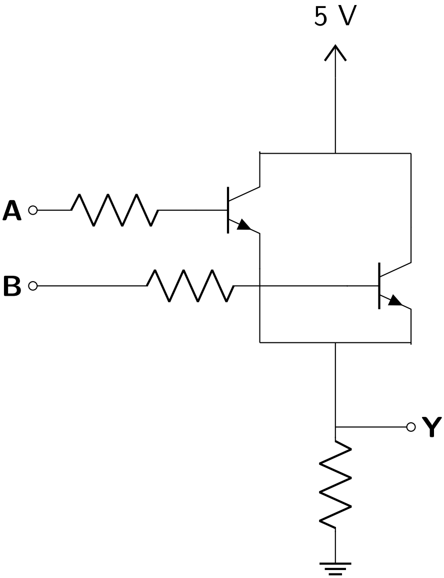
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	1

OR gate



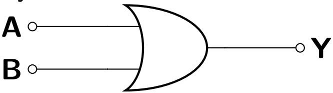
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	1

OR gate

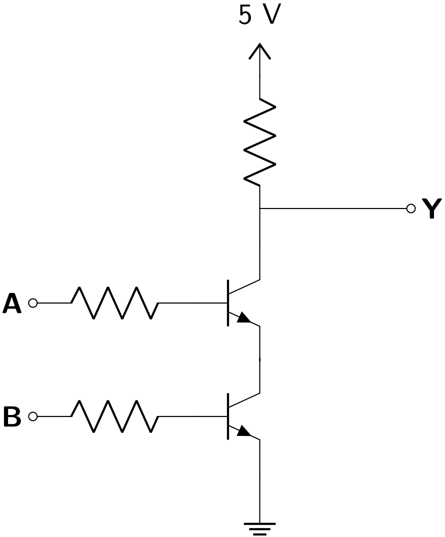


A	B	Y
0	0	0
1	0	1
0	1	1
1	1	1

Symbol:

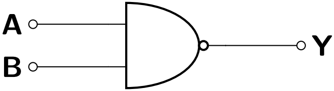


NAND gate



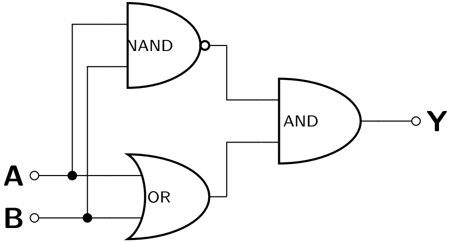
A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

Symbol:



Building some gates with other gates

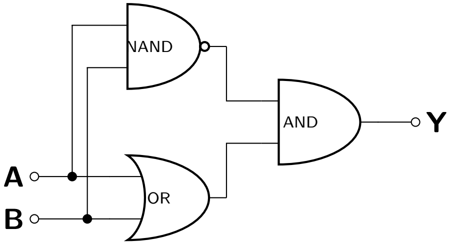
XOR gate



A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

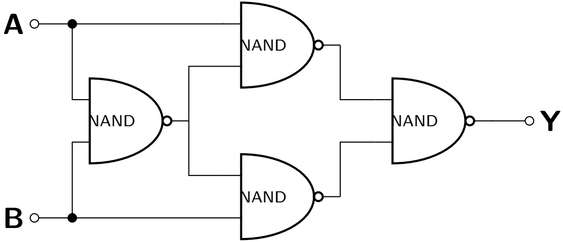
Building some gates with other gates

XOR gate



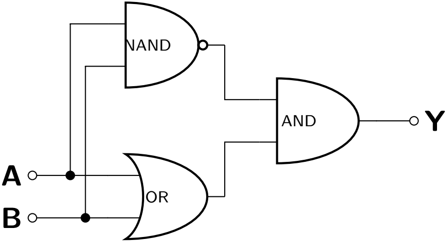
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

Alternatively using only **NAND** gates:



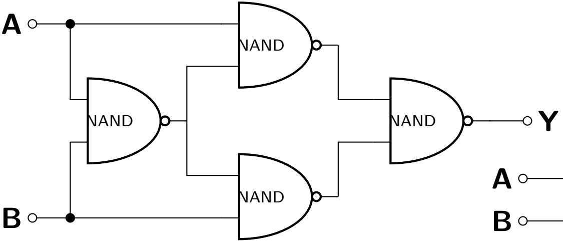
Building some gates with other gates

XOR gate

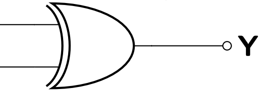


A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

Alternatively using only **NAND** gates:



Symbol:



Binary addition

- Adding two 1-bit numbers:

A	B	Y
0	0	0
1	0	1
0	1	1
1	1	10

Binary addition

- Adding two 1-bit numbers:

A	B	Y
0	0	0
1	0	1
0	1	1
1	1	10

← 2 bit output: **CARRY** and **SUM**

Binary addition

- Adding two 1-bit numbers:

A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

← 2 bit output: **CARRY** and **SUM**

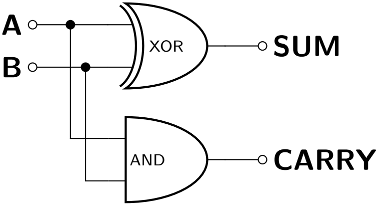
Binary addition

- Adding two 1-bit numbers:

A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

← 2 bit output: **CARRY** and **SUM**

- Can be implemented with one AND gate and one XOR gate:



Full Adder

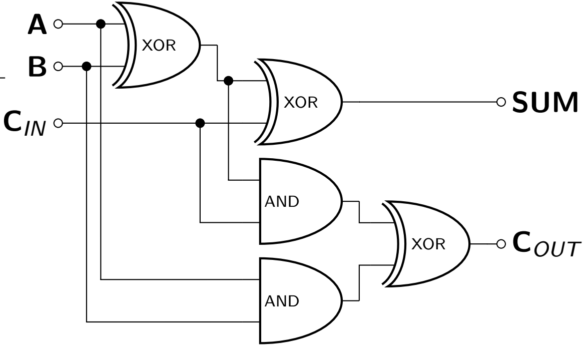
If we want to add N-bit numbers we have to account for the carry bit of lower-valued digits

A	**B**	**C**$_{in}$	**C**$_{out}$	**S**
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
1	1	0	1	0
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Full Adder

If we want to add N-bit numbers we have to account for the carry bit of lower-valued digits

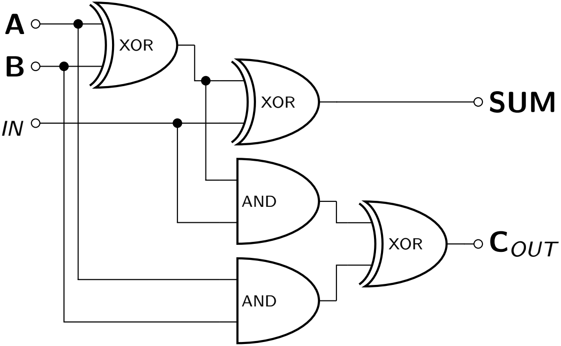
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
1	1	0	1	0
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



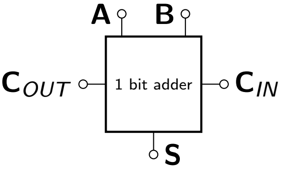
Full Adder

If we want to add N-bit numbers we have to account for the carry bit of lower-valued digits

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



Symbol for full adder:



Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:				0
SUM:				

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:				0
SUM:				1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:			0	0
SUM:				1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:			0	0
SUM:			0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:		1	0	0
SUM:			0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:		1	0	0
SUM:		0	0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:	1	1	0	0
SUM:		0	0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:	1	0	1	1
B:	1	1	1	0
CARRY:	1	1	0	0
SUM:	1	0	0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:		1	0	1	1
B:		1	1	1	0
CARRY:	1	1	1	0	0
SUM:		1	0	0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

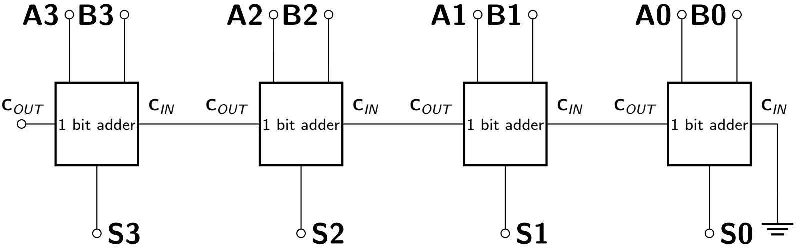
A:		1	0	1	1
B:		1	1	1	0
CARRY:	1	1	1	0	0
SUM:	1	1	0	0	1

Building a N-bit adder

- Adding two N-bit binary numbers:

A:		1	0	1	1
B:		1	1	1	0
CARRY:	1	1	1	0	0
SUM:	1	1	0	0	1

- nibble **ripple carry adder**:

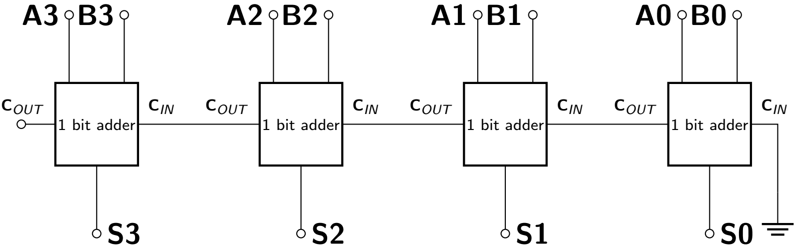


Building a N-bit adder

- Adding two N-bit binary numbers:

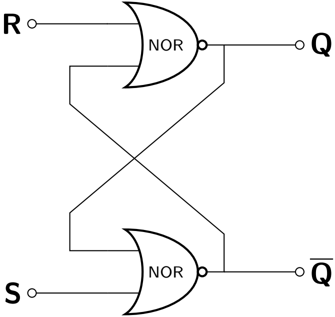
A:		1	0	1	1
B:		1	1	1	0
CARRY:	1	1	1	0	0
SUM:	1	1	0	0	1

- nibble **ripple carry adder**:



- Note: propagation delay of full adders

S-R latch

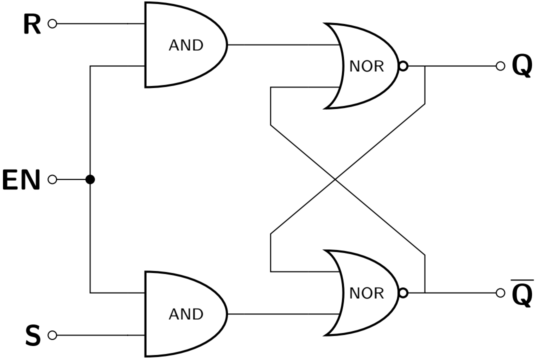


S	R	Q	$\overline{Q}$	
0	0			LATCHED
1	0	1	0	
0	1	0	1	
1	1			UNDEFINED

- Can be used to store 1 bit of information

Gated S-R latch

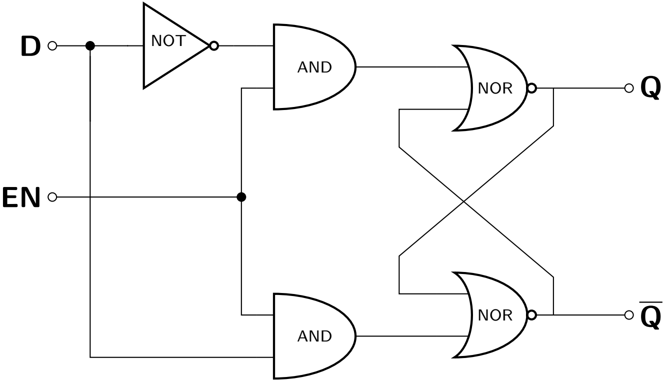
- S-R latch should only change state when certain conditions are met (e.g. on clock pulse) → gated S-R latch



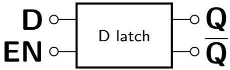
- output **Q** can only change when **EN** is HIGH

(Transparent) D latch

- Problem of "forbidden" state **R** = **S** = 1 for S-R latch
  → D latch prevents this from happening

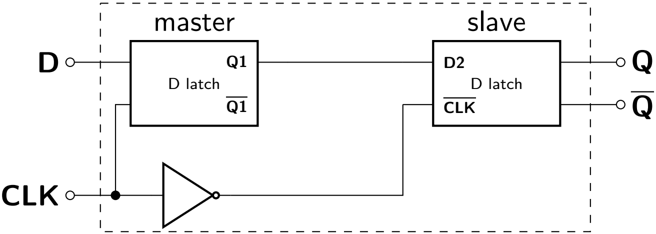


Symbol:



D flip-flop

- Only want the state **Q** to change at specific point in time
- Implemented using two latches in master-slave configuration:



- **Q** changes only on falling edge of the **CLK** signal