Simple As Possible — Computers

Chandradeep Dey

G Namratha Reddy

Part 1

Boolean Algebras

Part 1

A Boolean Algebra $\mathcal{B} = (B, \sqcup, \sqcap, \neg, 0, 1)$ is a distributive lattice B with top and bot...

Part 1

A Boolean Algebra $\mathcal{B} = (B, \sqcup, \sqcap, \neg, 0, 1)$ is a distributive lattice B with top and bot...

Boolean Algebra — Ooga booga monkey version

TRUE

Boolean Algebra — Ooga booga monkey version

TRUE

FALSE

Boolean Algebra — Ooga booga monkey version

TRUE

FALSE

AND ($\wedge$)

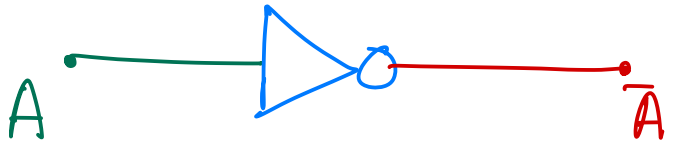
OR ($\vee$)

NOT ($\neg$)

Digital Logic

Digital Logic

NOT GATE



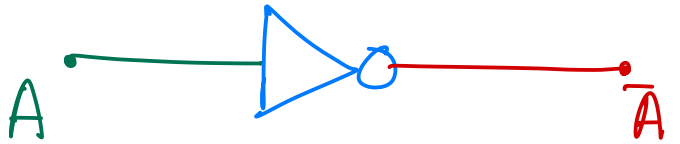
A	$\bar{A}$
0	1
1	0

Truth - table

7404

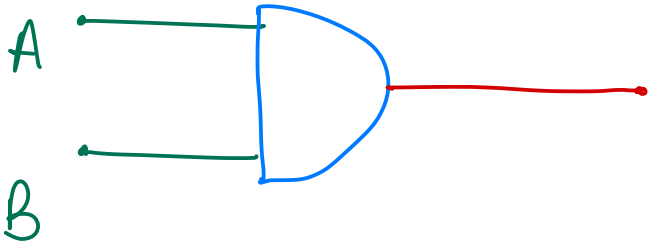
Digital Logic

AND GATE



A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Truth-table

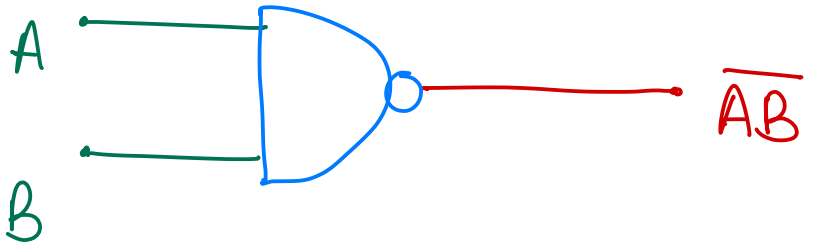


$A \wedge B$ / AB

7408

Universal Gate

NAND GATE



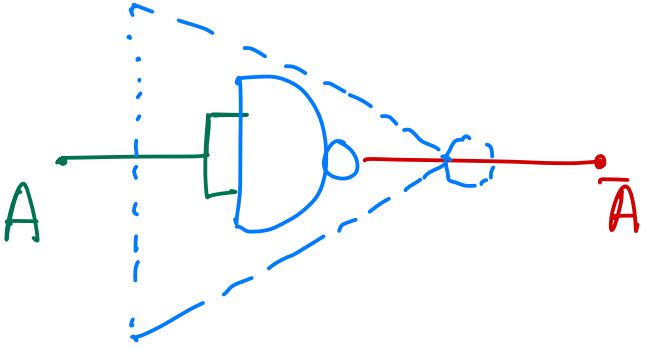
A	B	$A \wedge B$
0	0	1
0	1	1
1	0	1
1	1	0

Truth - table

7400

Universal Gate

NOT GATE



A	$\bar{A}$
0	1
1	0

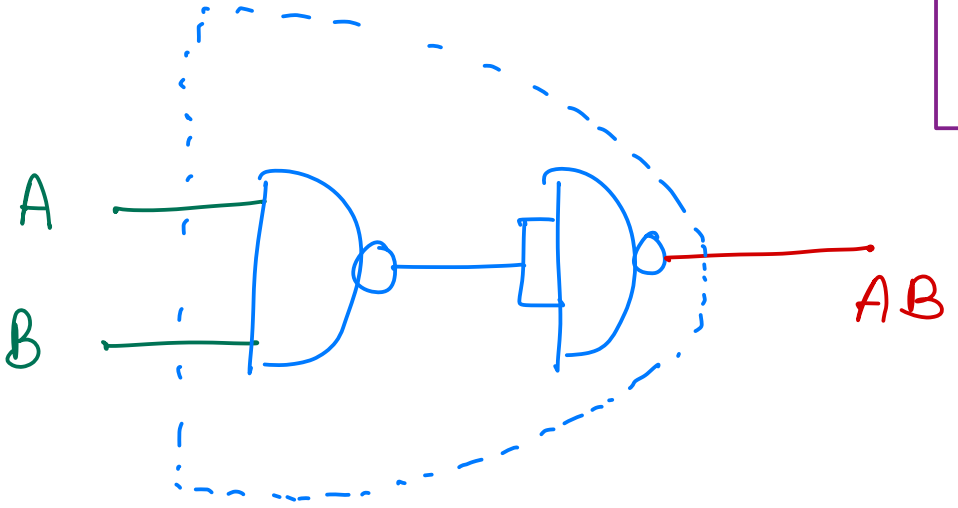
Truth - table

Universal Gate

AND GATE

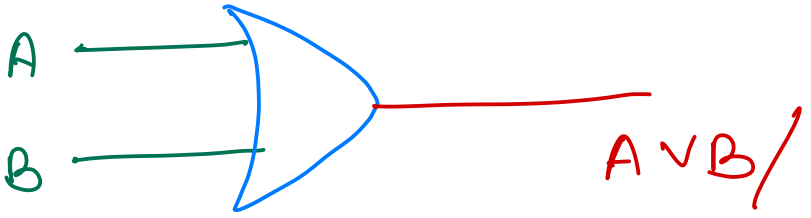
A	B	A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

Truth - table



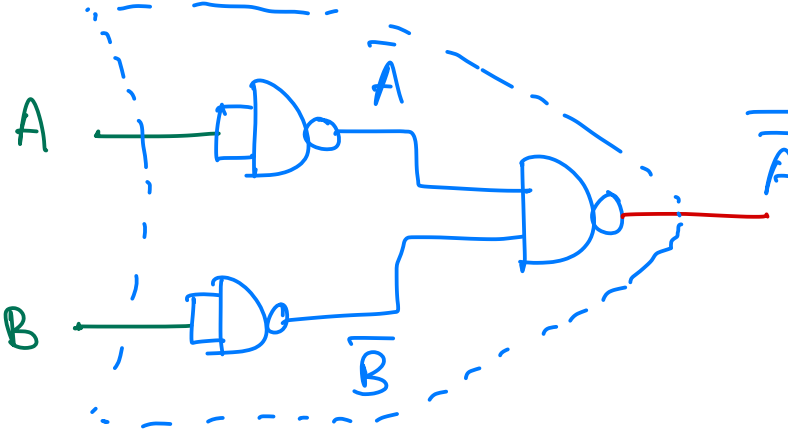
Universal Gate

OR GATE



A	B	A ∨ B
0	0	0
0	1	1
1	0	1
1	1	1

Truth - table

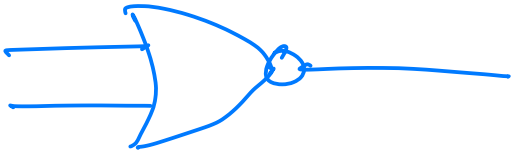


$\overline{\overline{A}\,\overline{B}} = A + B$ (De Morgan's Law)

7432

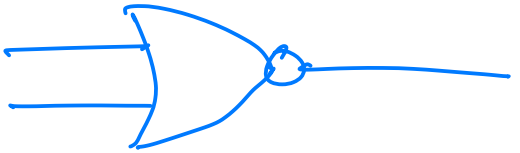
More stuff

NOR gate –



More stuff

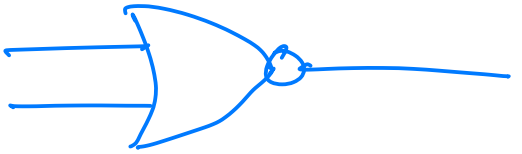
NOR gate*



* Also universal.

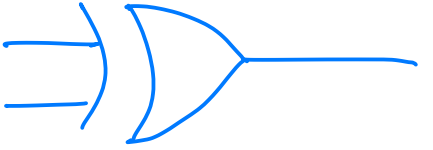
More stuff

NOR gate* –



* Also universal.

Exclusive-OR (XOR) gate –



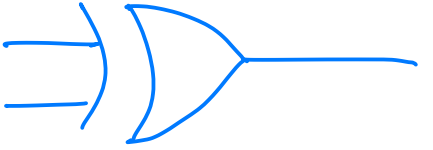
$$A \oplus B = \bar{A}B + A\bar{B}$$

More stuff

NOR gate* –

* Also universal.

Sum of products expression, a canonical way to describe behaviour



Exclusive-OR (XOR) gate –

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

7402

7486

Combinational Circuits

Full Adder

A	B	Cin	S	Cout
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

Full Adder

A	B	Cin	S	Cout
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

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

Full Adder

A	B	Cin	S	Cout
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

sum of products expression in canonical form:

$S =$

$Cout =$

Full Adder

A	B	Cin	S	Cout
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

sum of products expression in canonical form:

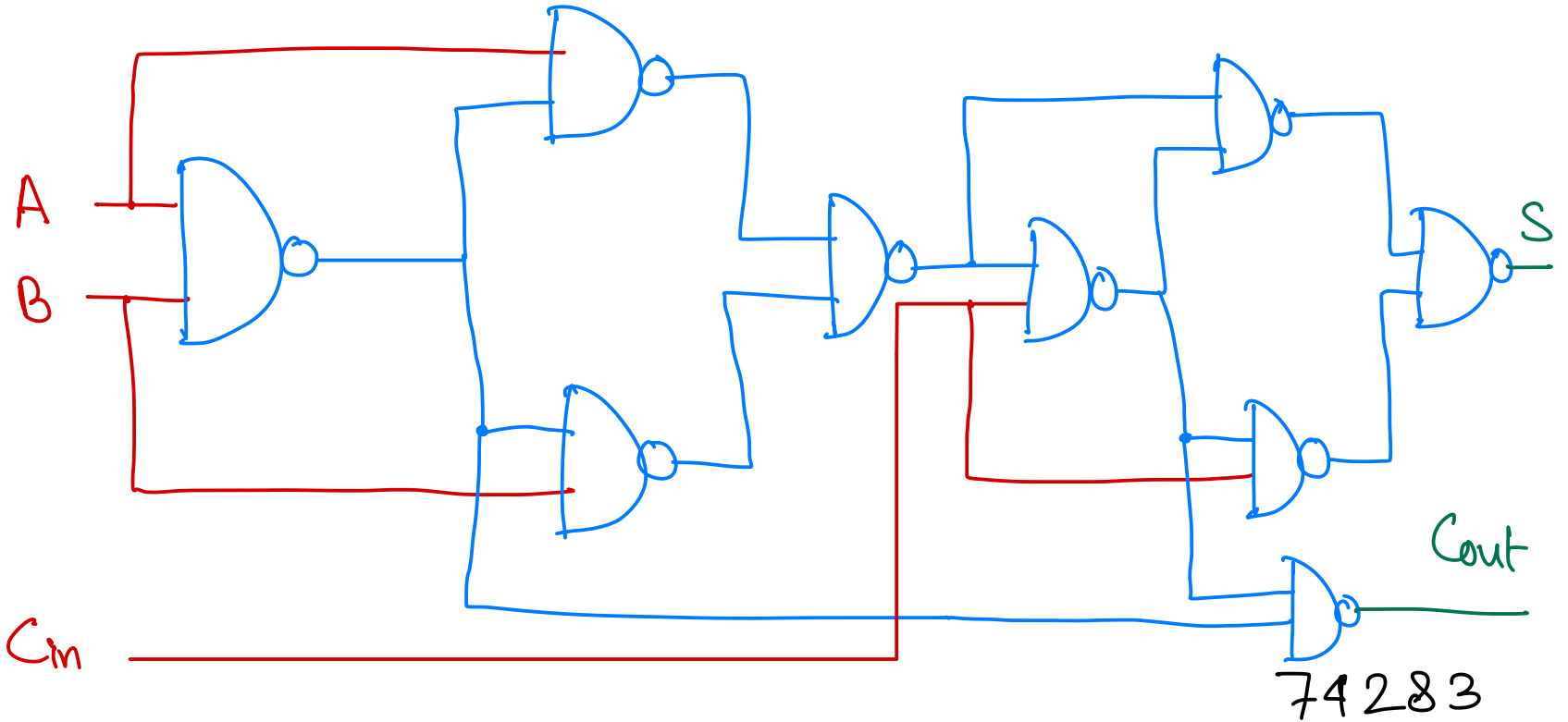
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$C_{out} = AB\bar{C}_{in} + A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in}$$

Full Adder

$$S = A\bar{B}\bar{C}_{in} + \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + ABC_{in}$$

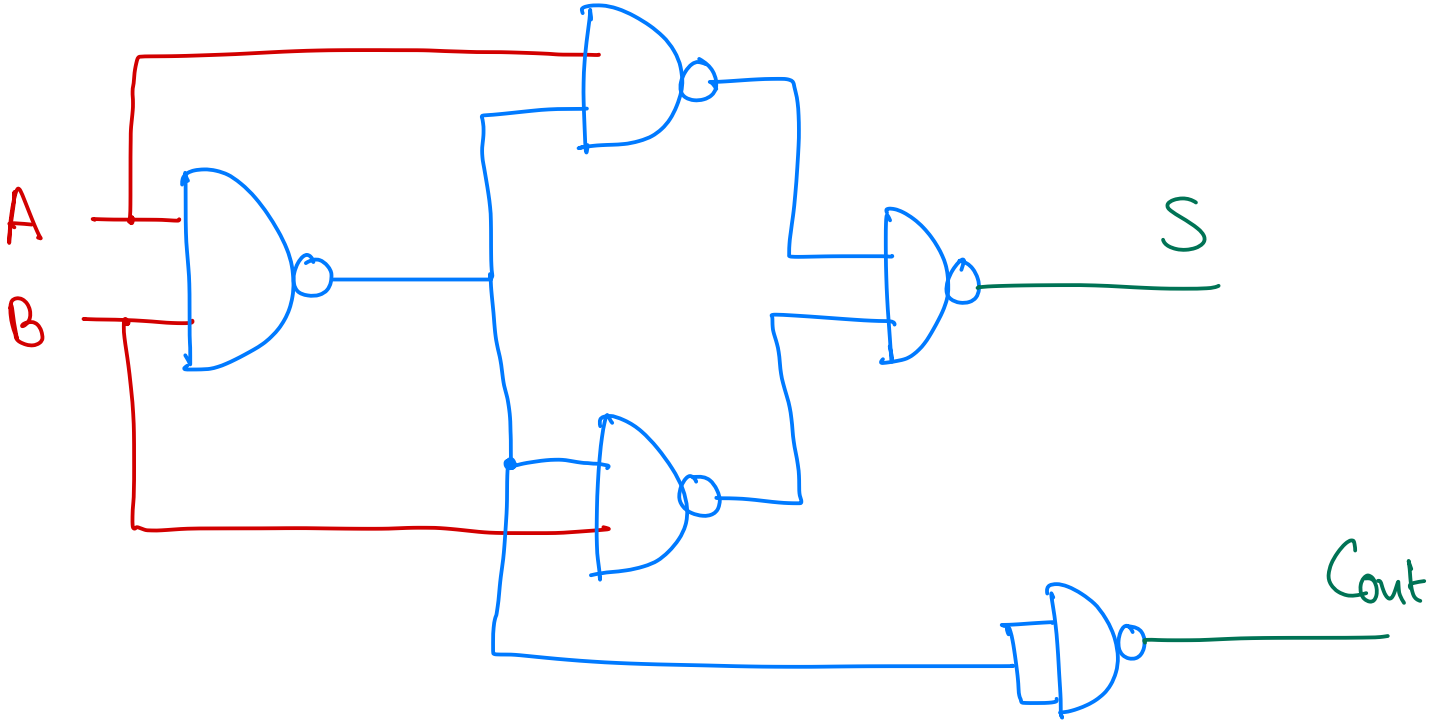
$$C_{out} = AB\bar{C}_{in} + A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in}$$



Half Adder

$S = A\bar{B} + \bar{A}B$

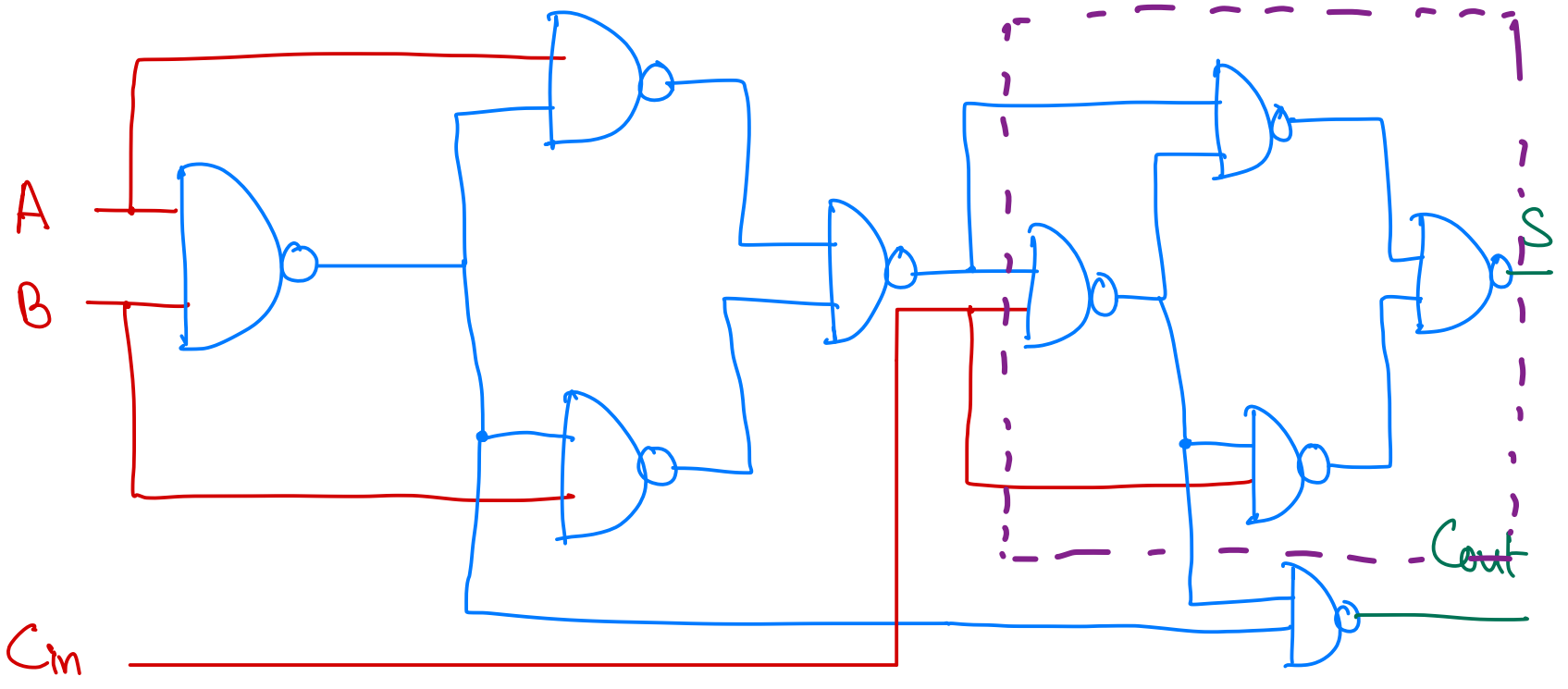
$C_{out} = AB$



Full Adder

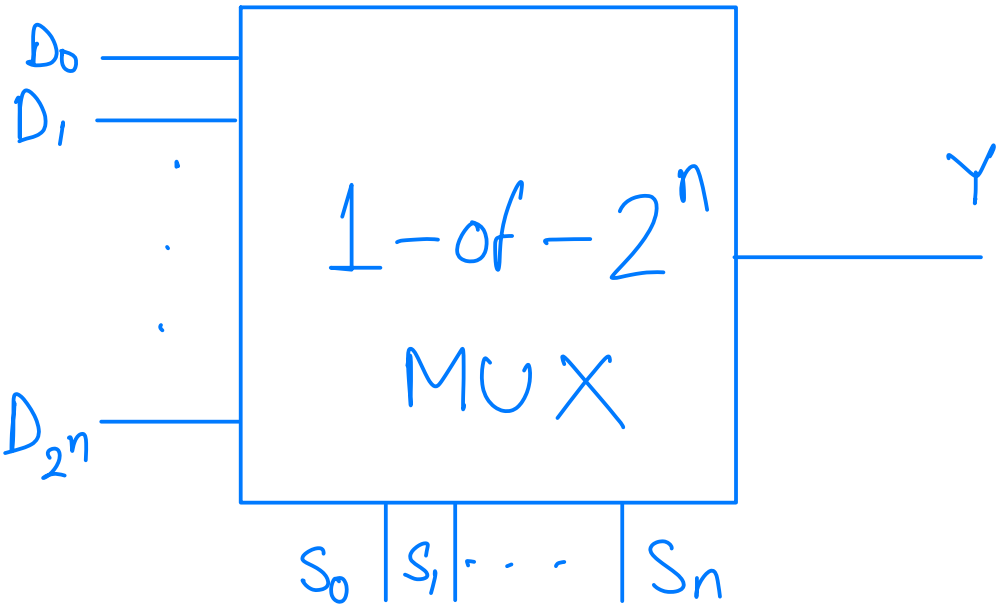
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$C_{out} = AB\bar{C}_{in} + A\bar{B}C_{in} + \bar{A}BC_{in} + ABC_{in}$$



Boolean functions - Multiplexers / Data Selectors

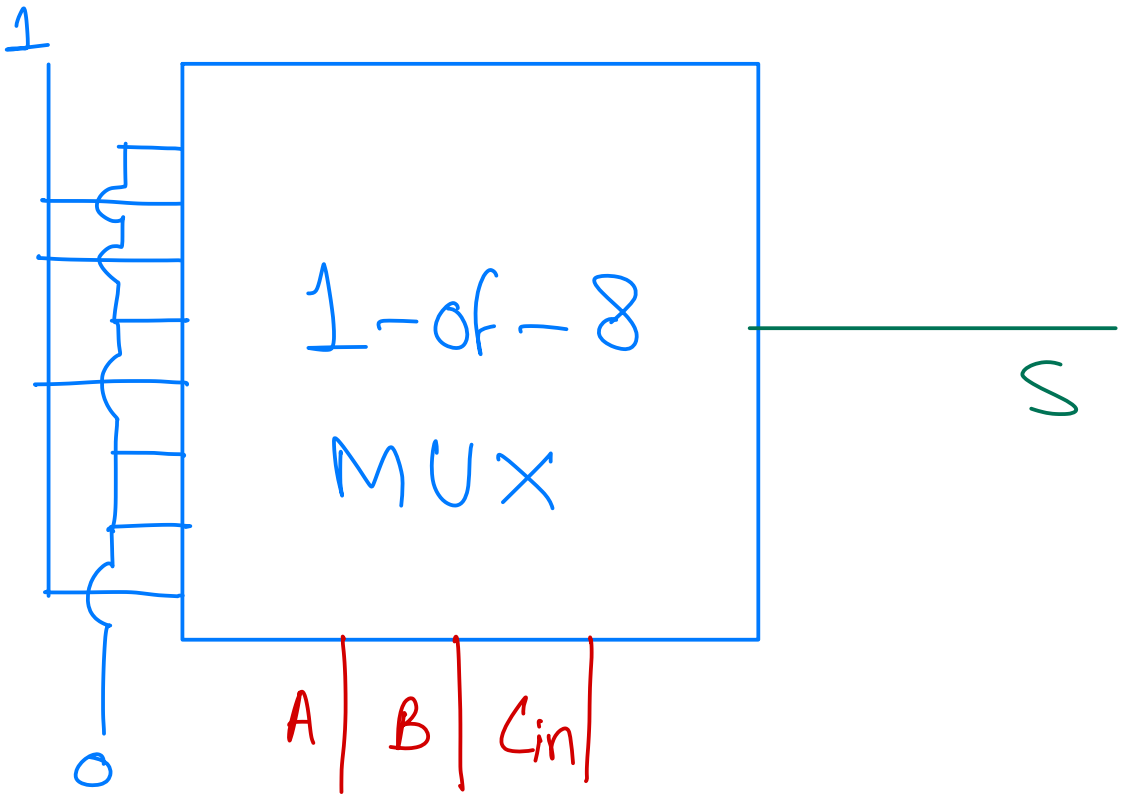
$f : \mathbb{B}^n \longrightarrow \mathbb{B}$



Full Adder

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

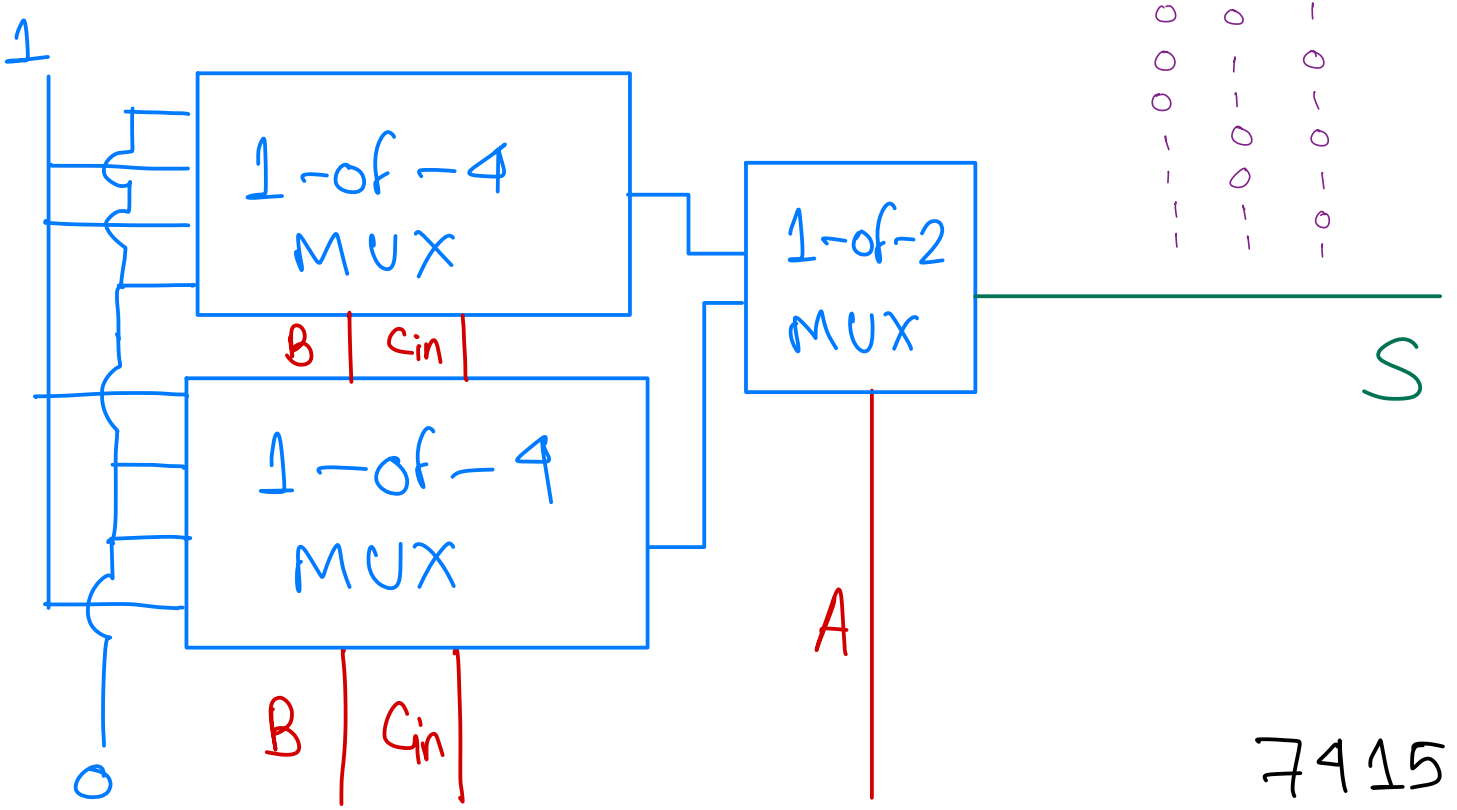
A	B	Cin	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Full Adder

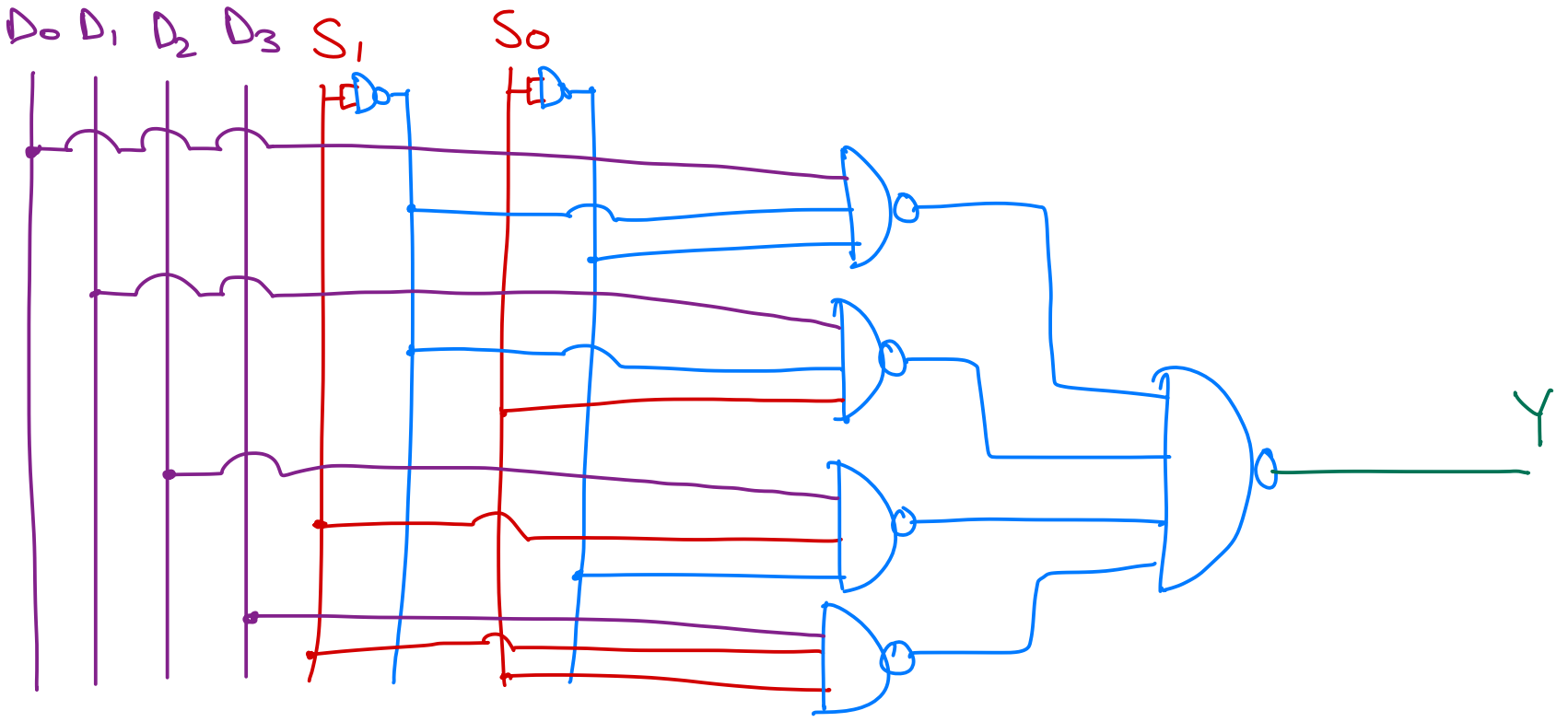
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

A	B	Cin	S
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	1

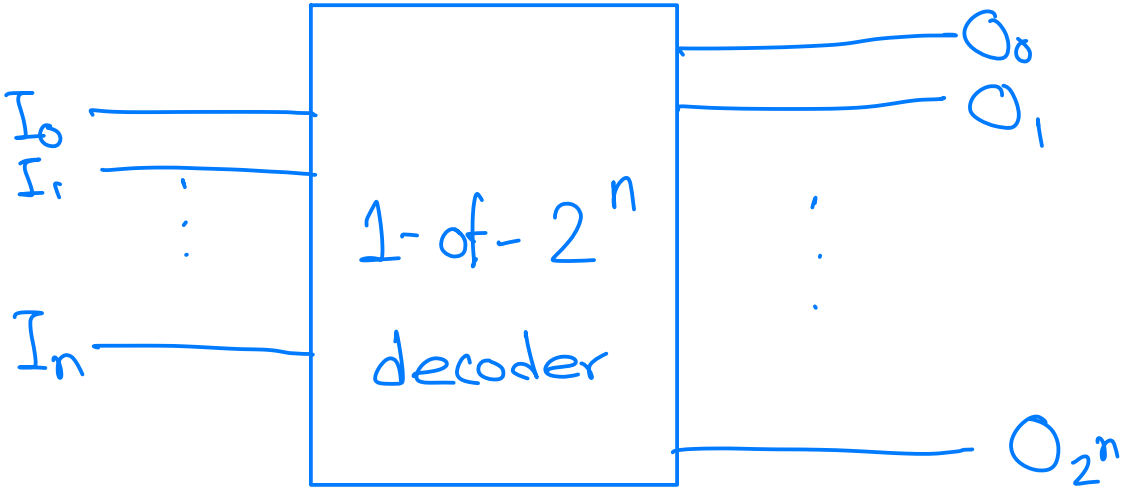


74153

Multiplexer

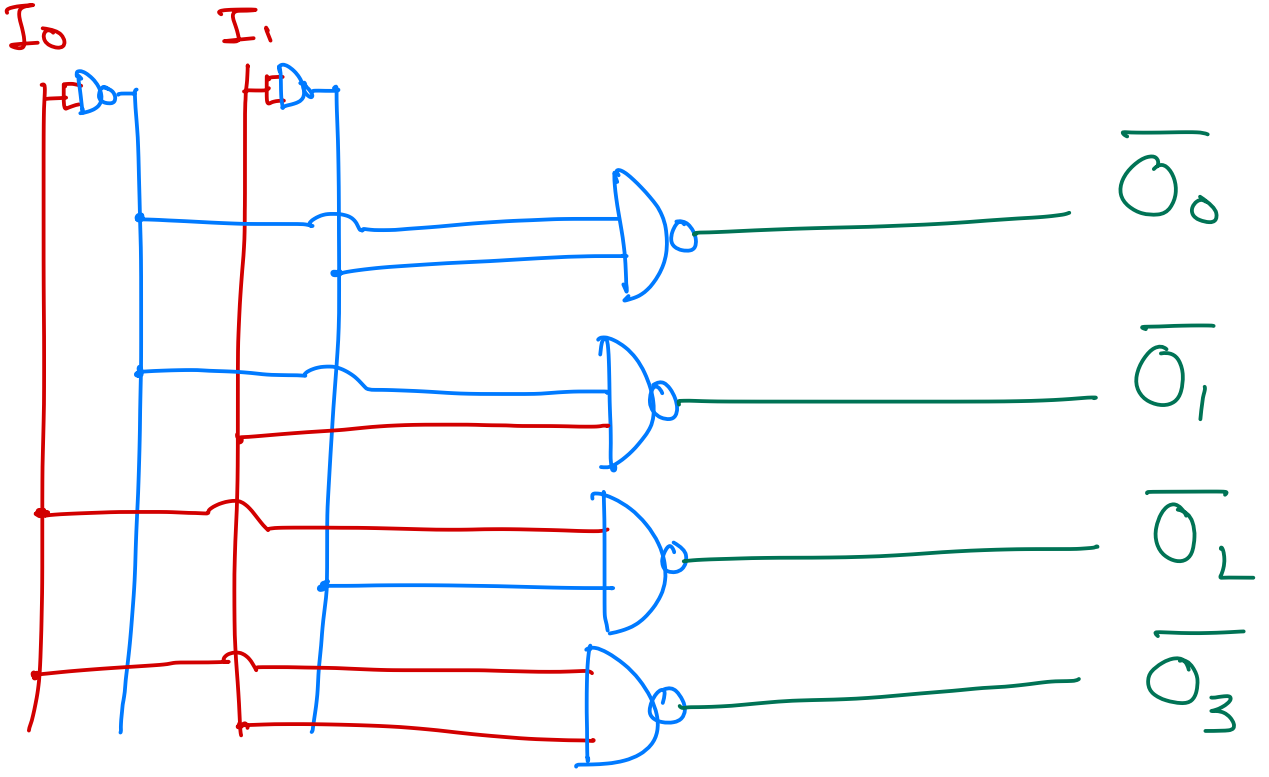


Decoders



Puts a 1 at the output corresponding to the input combination and 0 everywhere else.

Decoders



Tangent

Gray Code

Gray Code

B_2	B_1	B_0	Decimal	G_2	G_1	G_0
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	2	0	1	1
0	1	1	3	0	1	0
1	0	0	4	1	1	0
1	0	1	5	1	1	1
1	1	0	6	1	0	1
1	1	1	7	1	0	0

Gray Code

- One bit changes on every step

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- One bit changes on every step
- Usage in —
  - error correction

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- One bit changes on every step
- Usage in —
  - error correction
  - circuit minimisation

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- One bit changes on every step
- Usage in —
  - error correction
  - circuit minimisation
- arithmetic ? coding theory ? information theory ?

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- Binary-to-Gray

$$G_n = B_n$$

$$\forall 1 \le i \le n.\ G_{i-1} = B_i \oplus B_{i-1}$$

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- Gray-to-Binary

$$B_n = G_n$$

$$\forall 1 \le i \le n.\ B_{i-1} = B_i \oplus G_{i-1}$$

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Gray Code

- Gray-to-Binary

$$B_n = G_n$$

$$\forall 1 \le i \le n.\ B_{i-1} = B_i \oplus G_{i-1}$$

G_2	G_1	G_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

More stuff

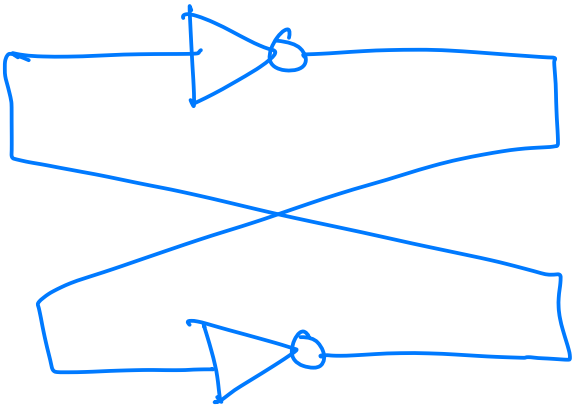
- Subtractors
  - 2's complement code.
    Subtraction using addition circuitry.
- Comparators
- Multipliers
- Demultiplexers
- Encoders

Sequential Circuits

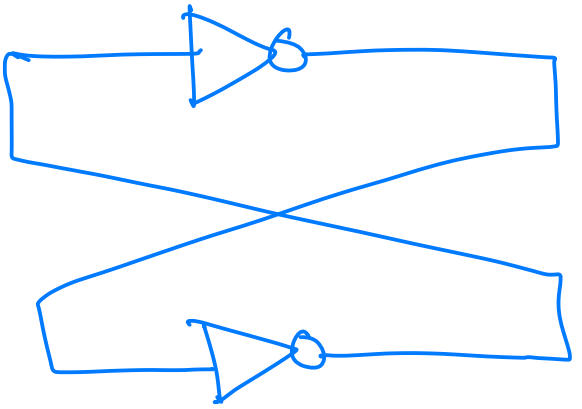
Blasphemy

TRIGGER WARNING FOR MATH PEOPLE

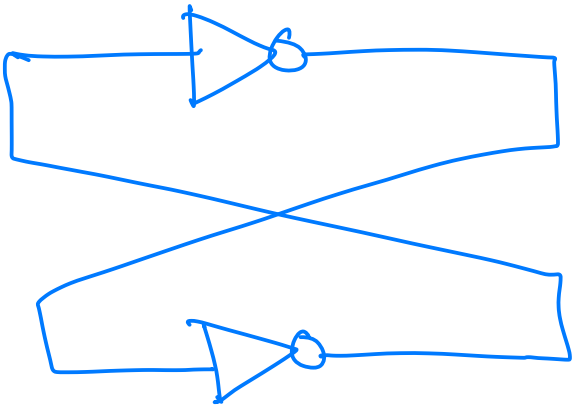
Blasphemy



Latch / bistable multivibrator

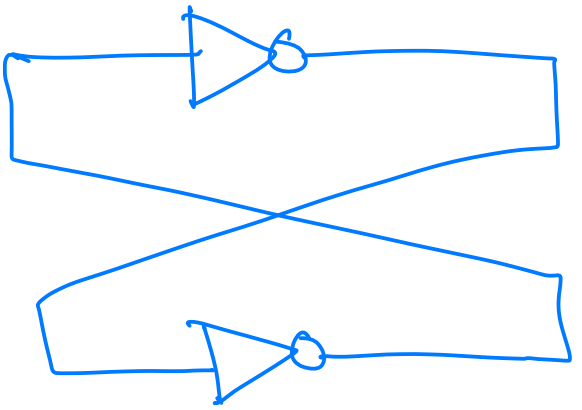


Latch / bistable multivibrator



— latches onto a value

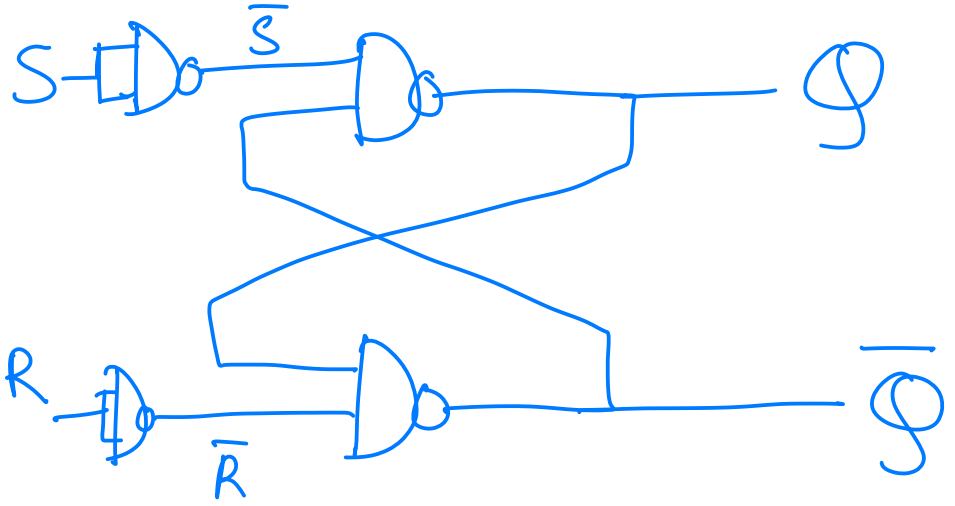
Latch / bistable multivibrator



— latches onto a value

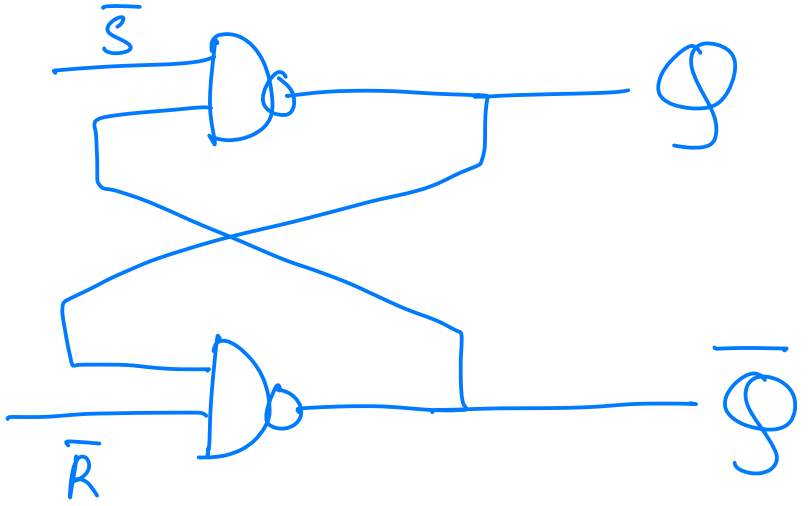
— has two stable states

Set - Reset (SR) latch



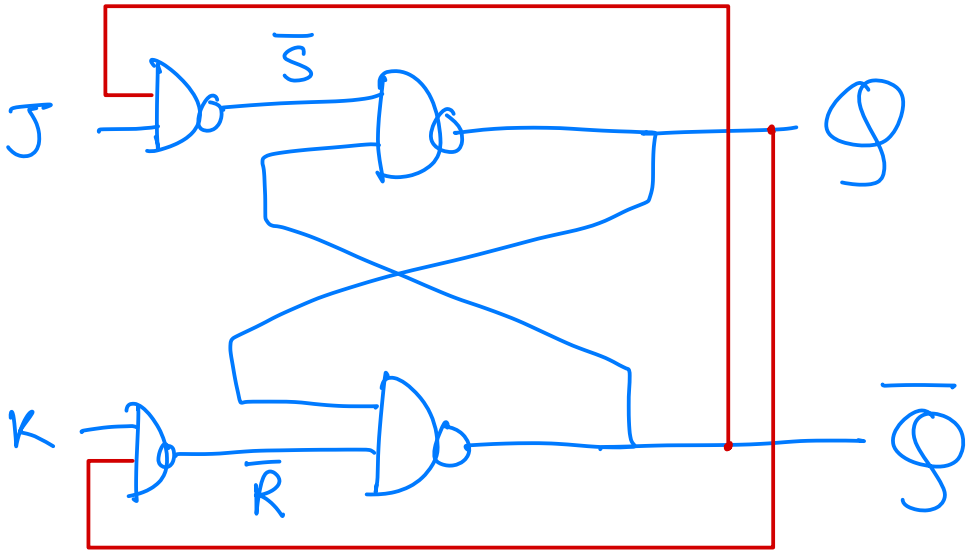
S	R	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	undefined

Set - Reset (SR) latch



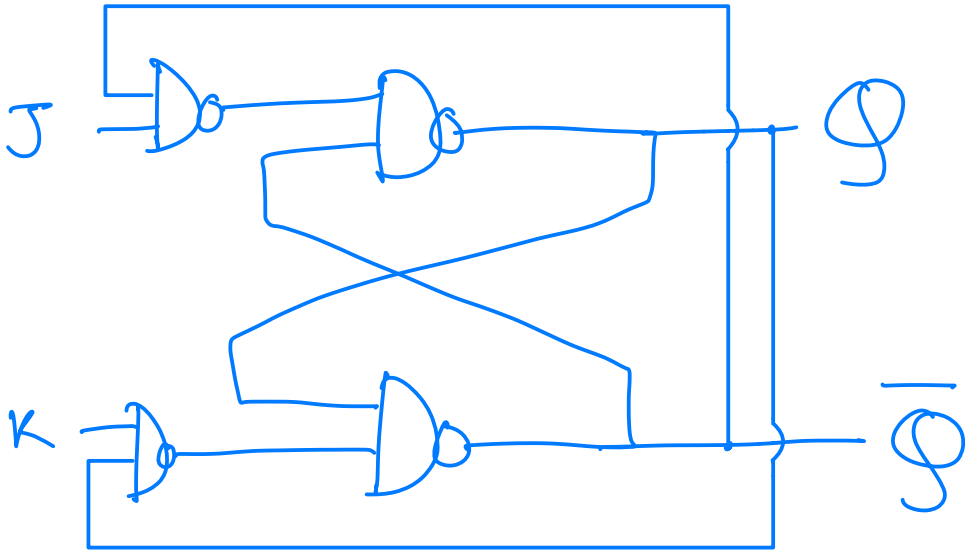
$\bar{S}$	$\bar{R}$	Q_{next}
1	1	Q
1	0	0
0	1	1
0	0	undefined

JK latch — let's define the undefined



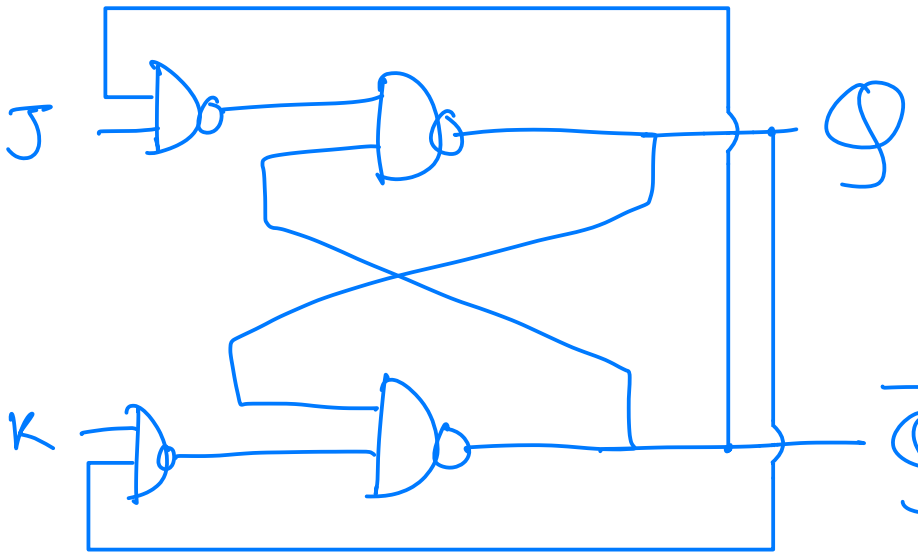
J	K	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

JK latch — how many flips?



J	K	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

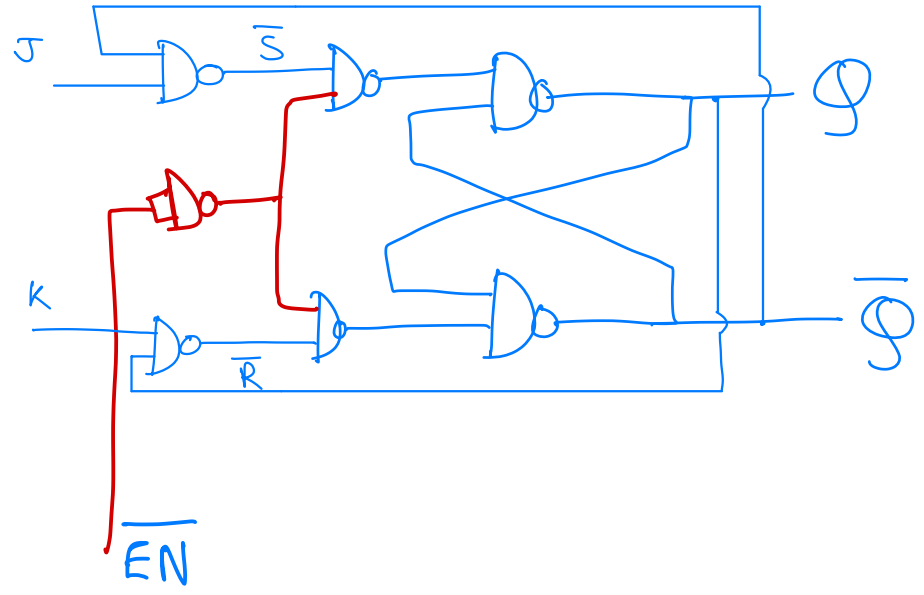
JK latch — how many flips?



J	K	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	$\overline{Q}$

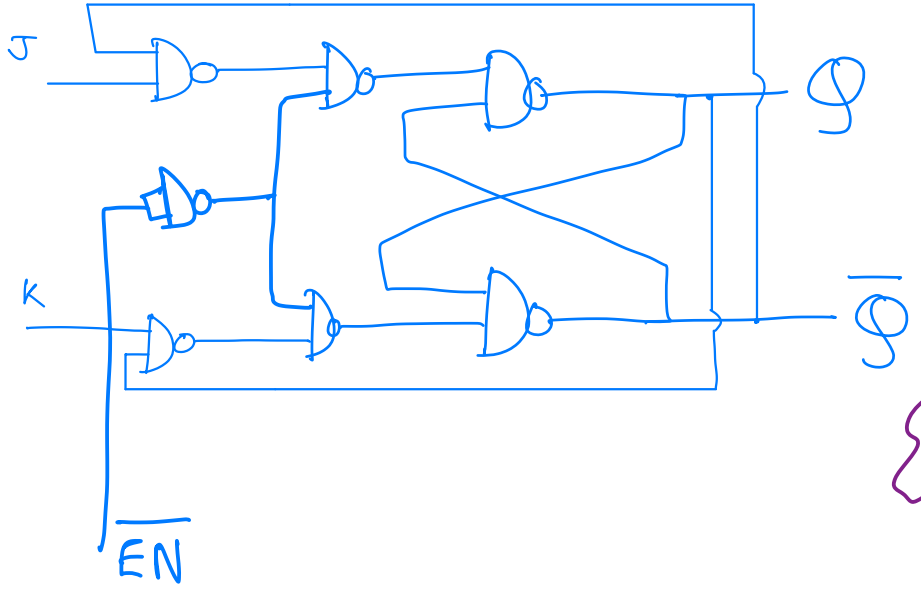
race-around condition

Level-triggering — Enable Input



$\overline{EN}$	J	K	Q_{next}
0	0	0	Q
0	0	1	0
0	1	0	1
0	1	1	$\overline{Q}$
1	X	X	Q

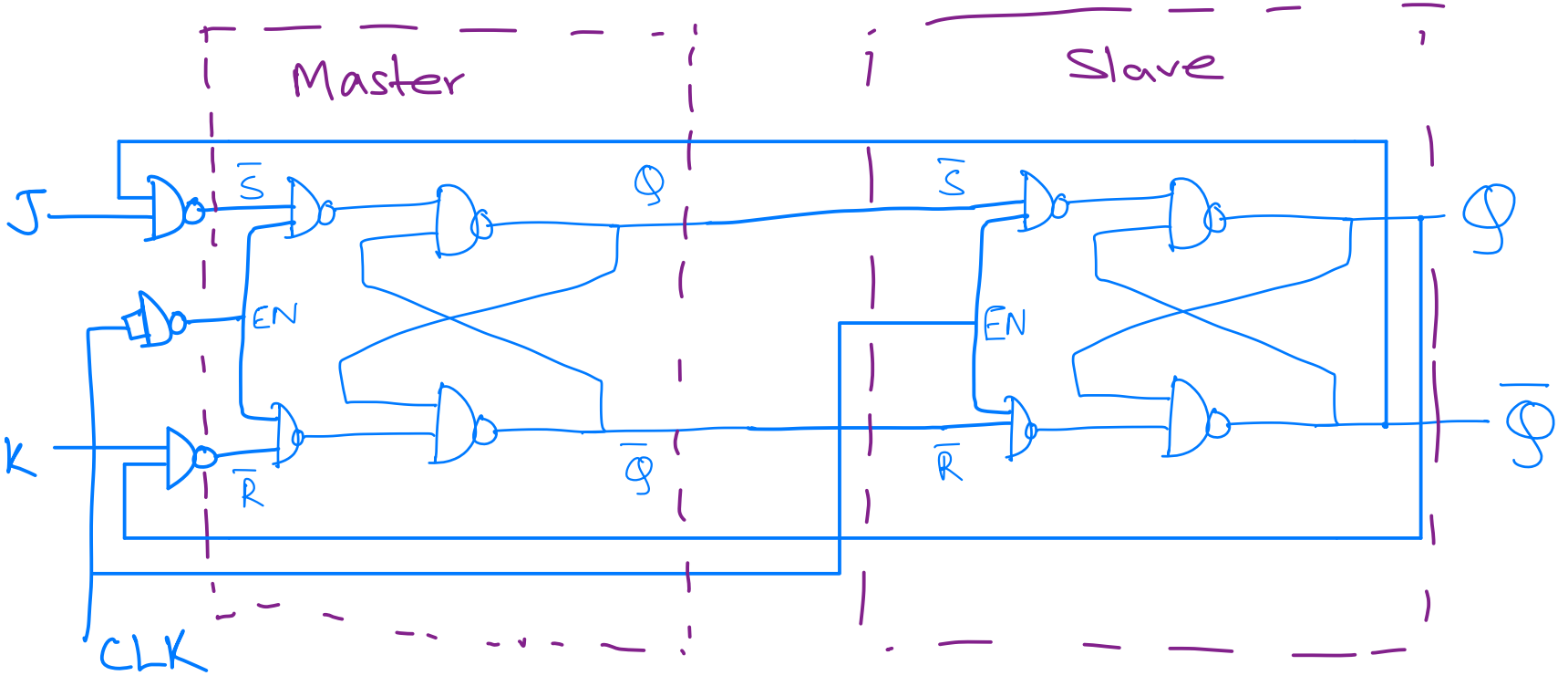
Level-triggering — Enable Input



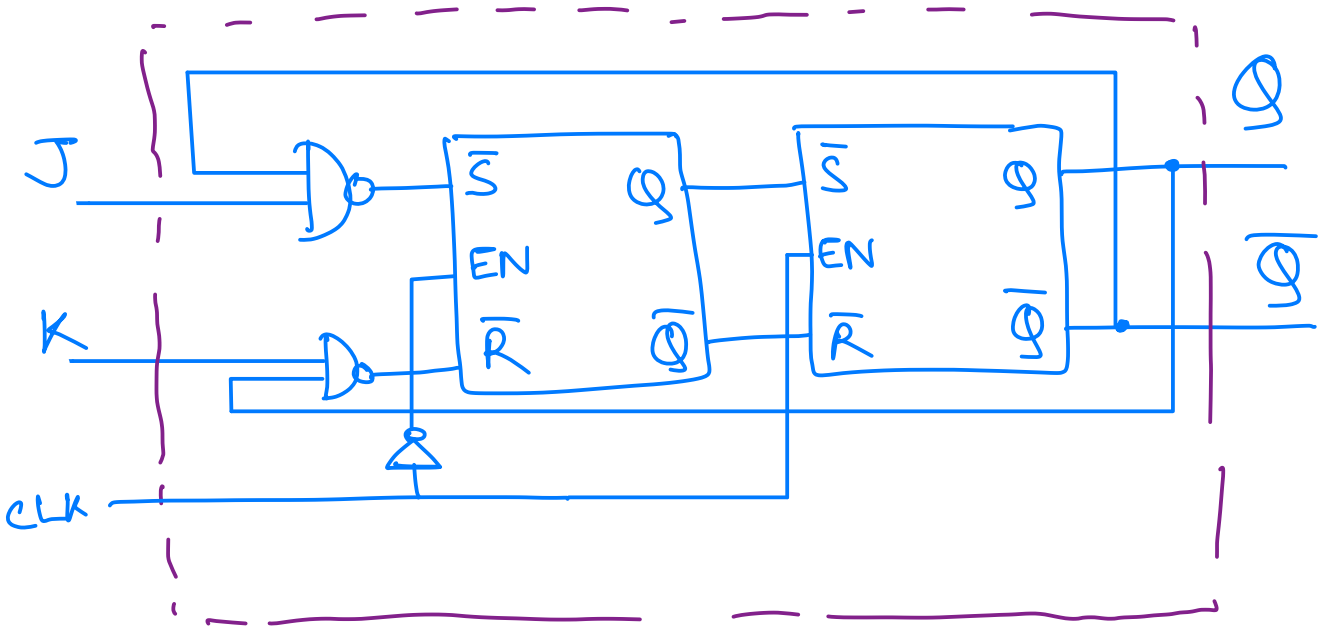
race?!

$\overline{EN}$	J	K	Q_{next}
0	0	0	Q
0	0	1	0
0	1	0	1
0	1	1	$\overline{Q}$
1	X	X	Q

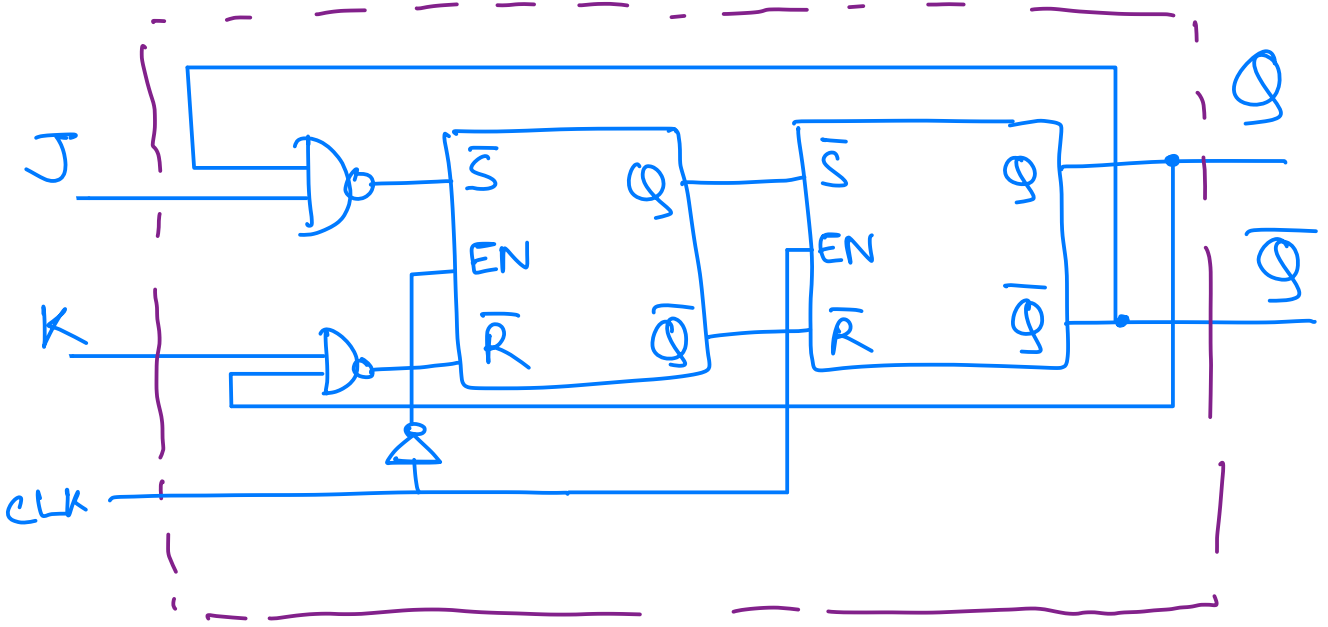
Edge-triggering — Master-slave circuit



JK Flip-Flop



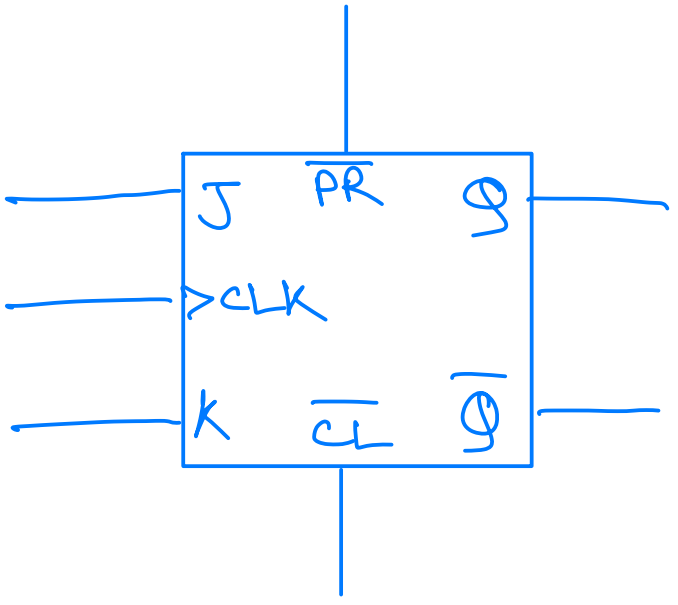
JK Flip-Flop



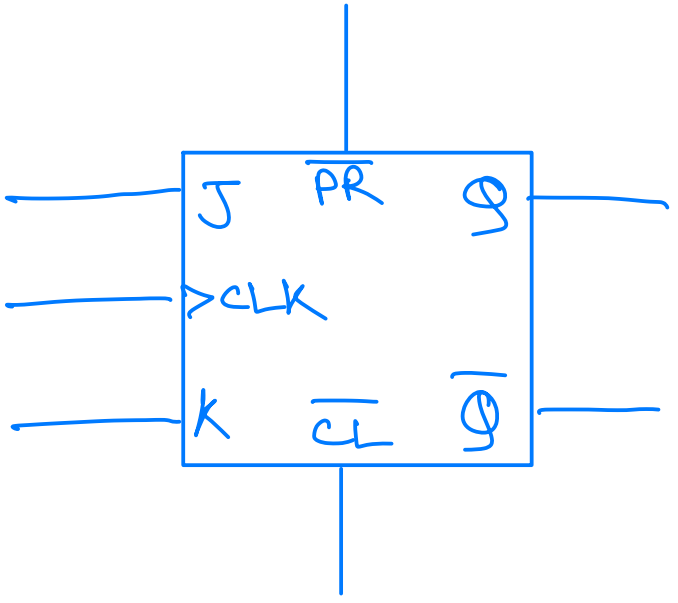
Note: Not the only way to achieve edge-triggering

JK Flip-Flop

7476

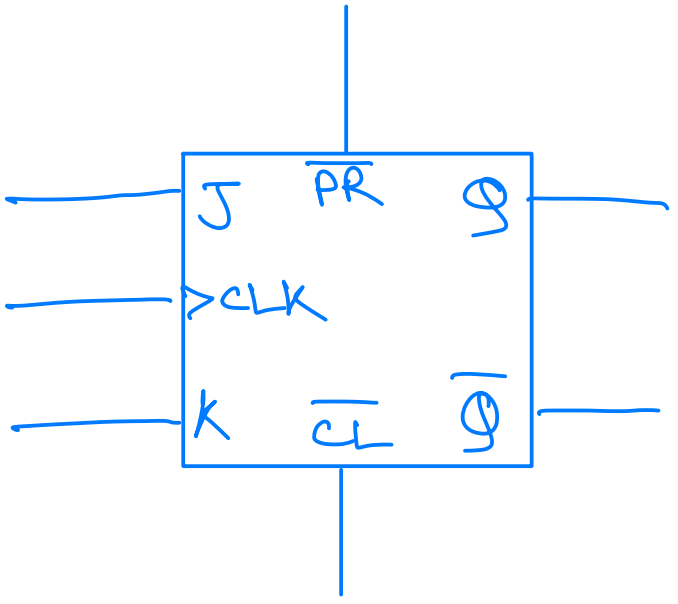


JK Flip-Flop



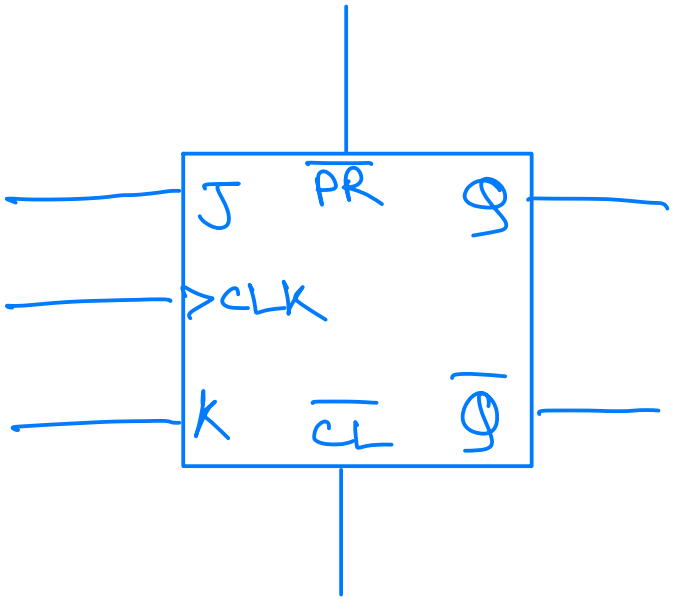
CLK	J	K	Q_{next}
0	X	X	Q
1	X	X	Q
↓	X	X	Q
↑	0	0	Q
↑	0	-	0
↑	-	0	1
↑	-	-	$\bar{Q}$

JK Flip-Flop



Preset and clear

JK Flip-Flop

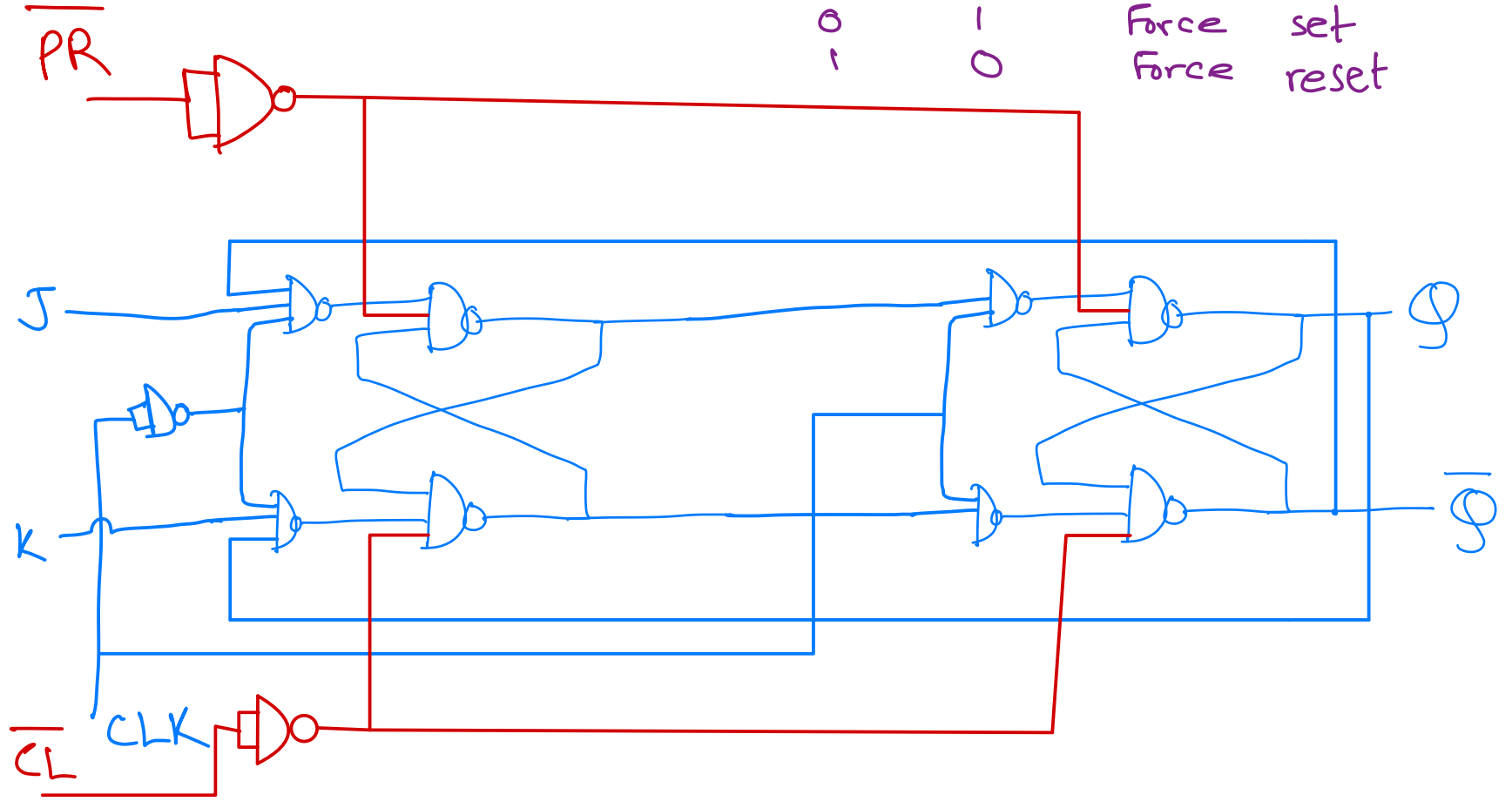


Preset and clear

Directly connects to the inner NAND gates to asynchronously set or reset the flip-flop.

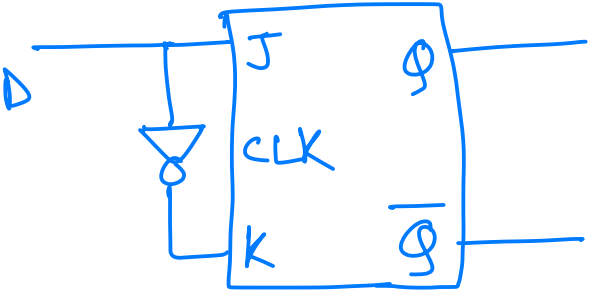
Preset and clear

$\overline{PR}$	$\overline{CL}$	
1	1	Normal operation
0	1	Force set
1	0	Force reset



More stuff

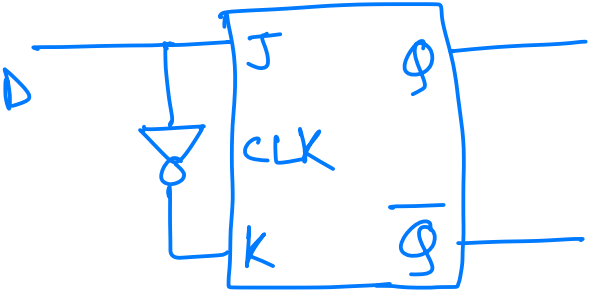
D Flip-flop



D	Q_{next}
0	0
1	1

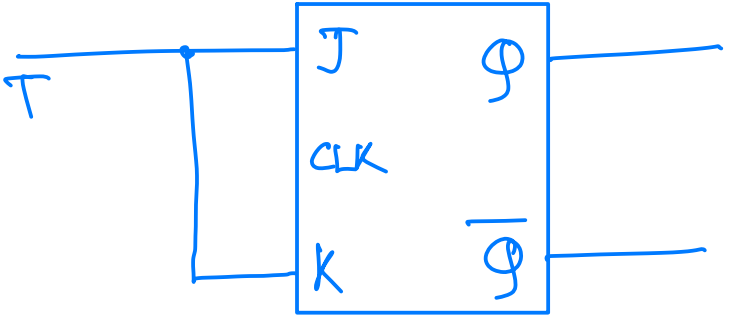
More stuff

D Flip-flop



D	Q_{next}
0	0
1	1

T Flip-flop



T	Q_{next}
0	Q
1	$\overline{Q}$

Registers

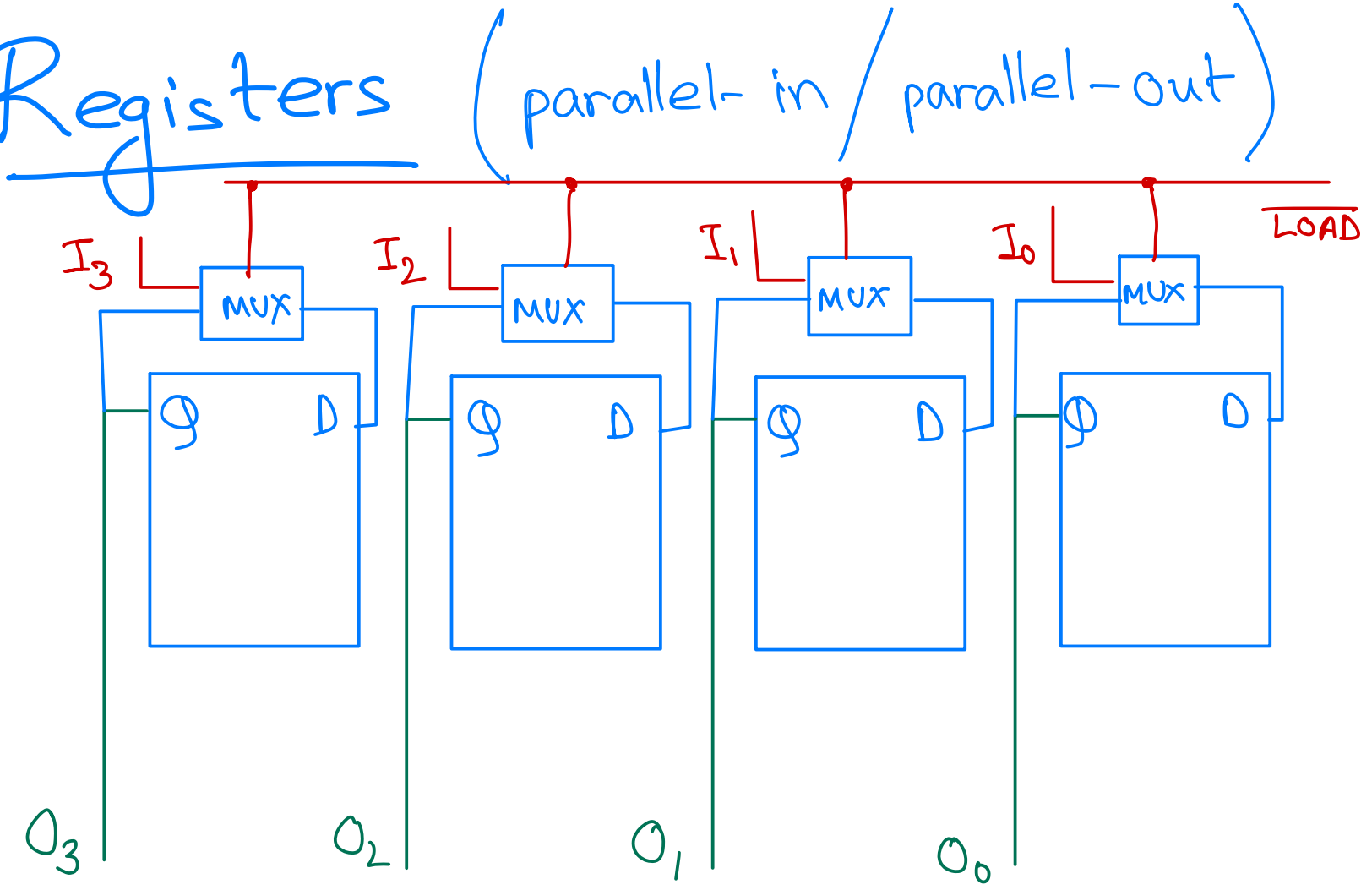
Registers

- use flip-flops to store data

Registers

- use flip-flops to store data
- inputs/outputs can be serial/parallel

Registers (parallel-in / parallel-out)



Counters

Counters

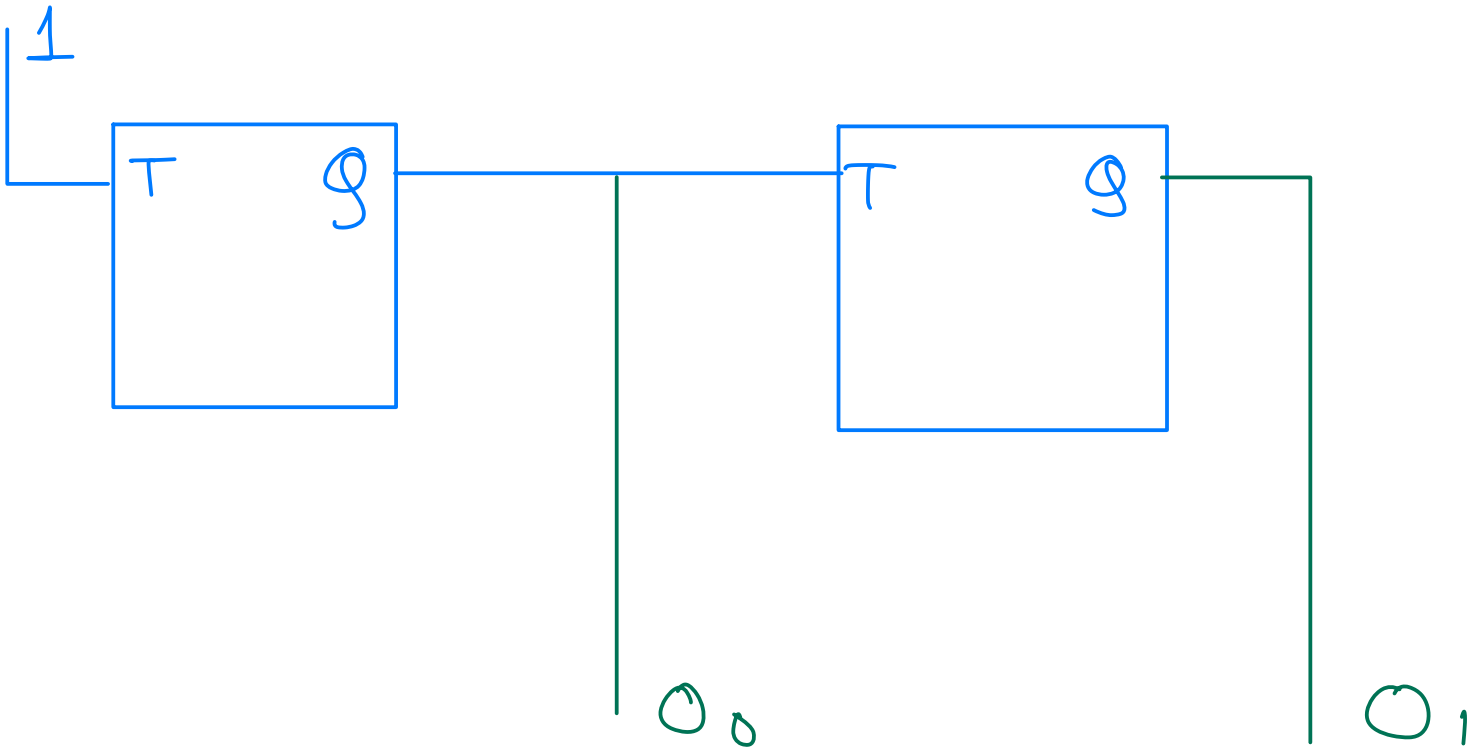
— they count, duh.

Counters

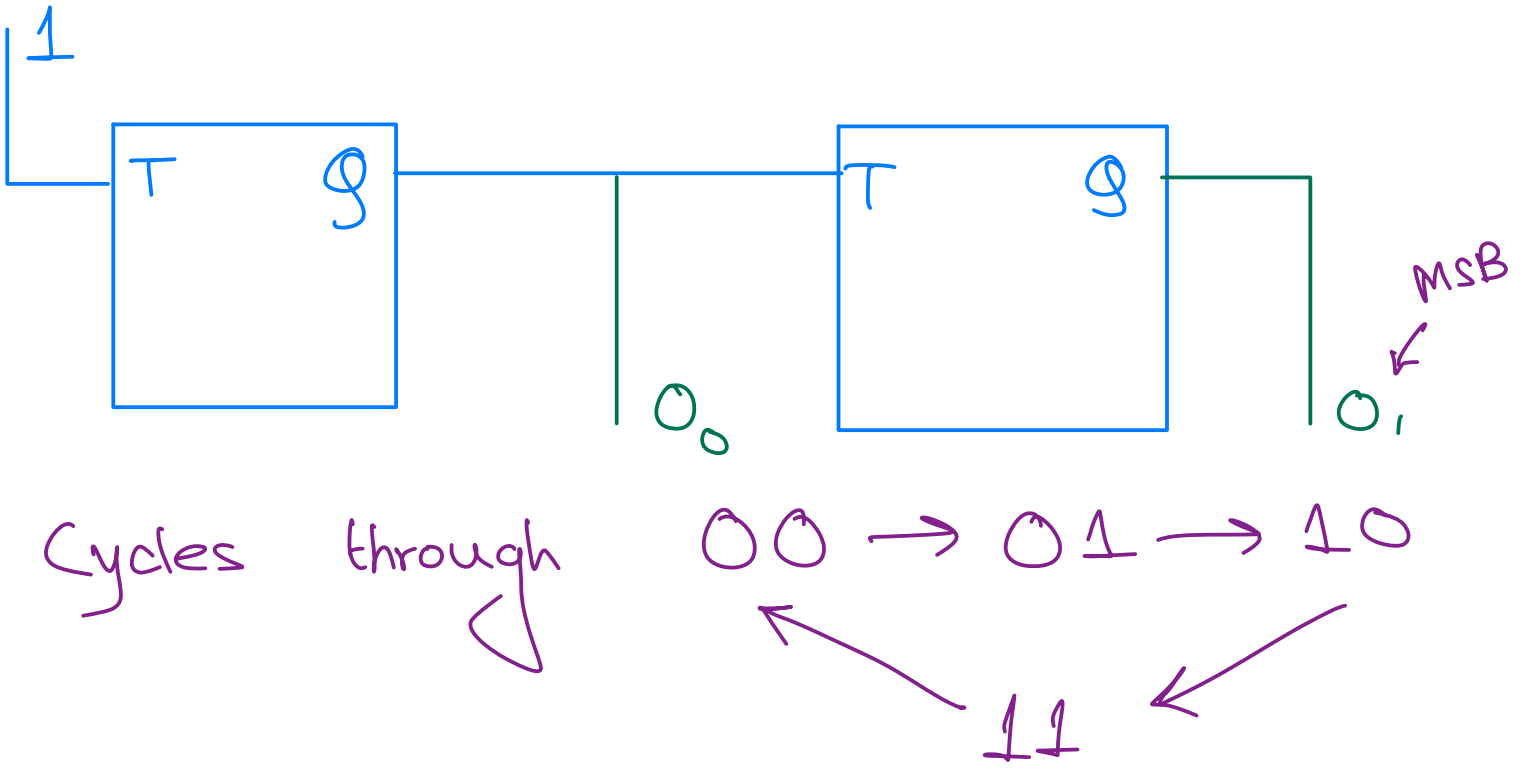
— they count, duh.

— rather, they cycle through a sequence of states.

Counters (mod-4 counter)



Counters (mod-4 counter)



Counters

— can stop before $2^n - 1$ and start after 0.

- use the outputs to create a term that denotes the number you want to stop at.
- feed the output of that term into appropriate presets and clears.

RAM

— bunch of memory cells arranged in a matrix.

— address decoders find out which row and column to read from / write to.