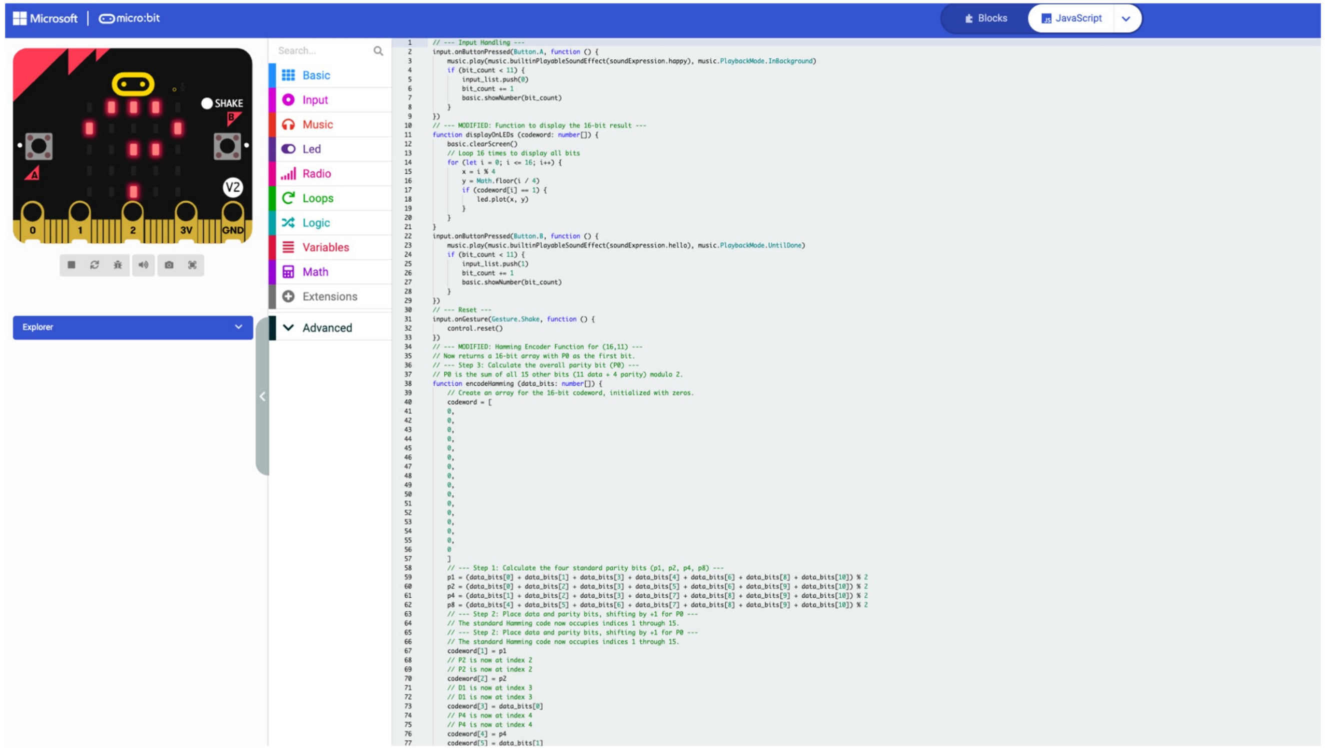
ENR-325/325L Principles of Digital Electronics and Laboratory

Xiang Li

Fall 2025

Coe College

Hamming codes can be done in the CS way



COE COLLEGE

Hamming codes can be done in the EE way

- Before that, we need to acquire some basic skill sets.

Pre-step: Data forms

Step 1: Data manipulation

Step 2: Information storage

Step 3: Interface

Pre-step: Data forms

- Say bye-bye to base 10:

Base 10 (0,1,2,3,4,5,6,7,8,9):

$$(4321)_{10}= 4\times 10^3 + 3\times 10^2 + 2\times 10^1 + 1\times 10^0$$

Base 2 (0,1):

$$(1011)_{2}= 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0$$

Base 16 (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F):

$$(FF12)_{16}= 15\times 16^3 + 15\times 16^2 + 1\times 16^1 + 2\times 16^0$$

Looking up how we do base conversions manually and in python.

The calculation of base 2 are pretty boring compared to base 10

Base 10	Base 2	Base 10	Base 2
$\begin{array}{r} 324 \\ \underline{+123} \end{array}$	$\begin{array}{r} 110 \\ \underline{+101} \end{array}$	$\begin{array}{r} 324 \\ \underline{\times 123} \end{array}$	$\begin{array}{r} 110 \\ \underline{\times\ 101} \end{array}$
$\begin{array}{r} 324 \\ \underline{-123} \end{array}$	$\begin{array}{r} 110 \\ \underline{-101} \end{array}$	$\begin{array}{r} 324 \\ \underline{\div\ 6} \end{array}$	$\begin{array}{r} 110 \\ \underline{\div 10} \end{array}$

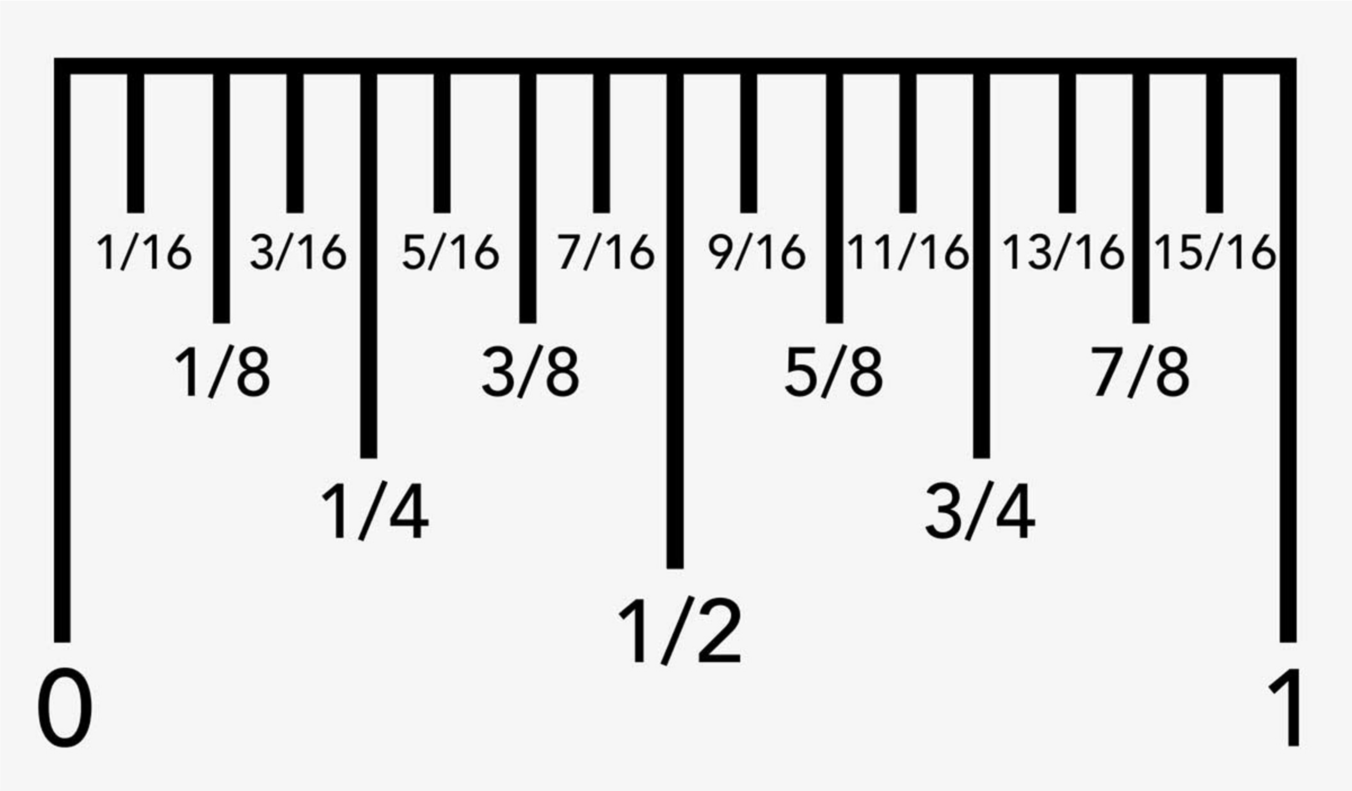
- Your base 10 arithmetic skills can be translated to base 2 ones.
- We will revisit more binary arithmetic operation later, after the logic gates!

Discuss: the origin of base 16?

Discuss: the origin of base 16?

My theory:
An easy and fair way to compute with a weightless balance scale.

https://commons.wikimedia.org/w/index.php?curid=79229218



https://www.inchcalculator.com/how-to-read-a-ruler/

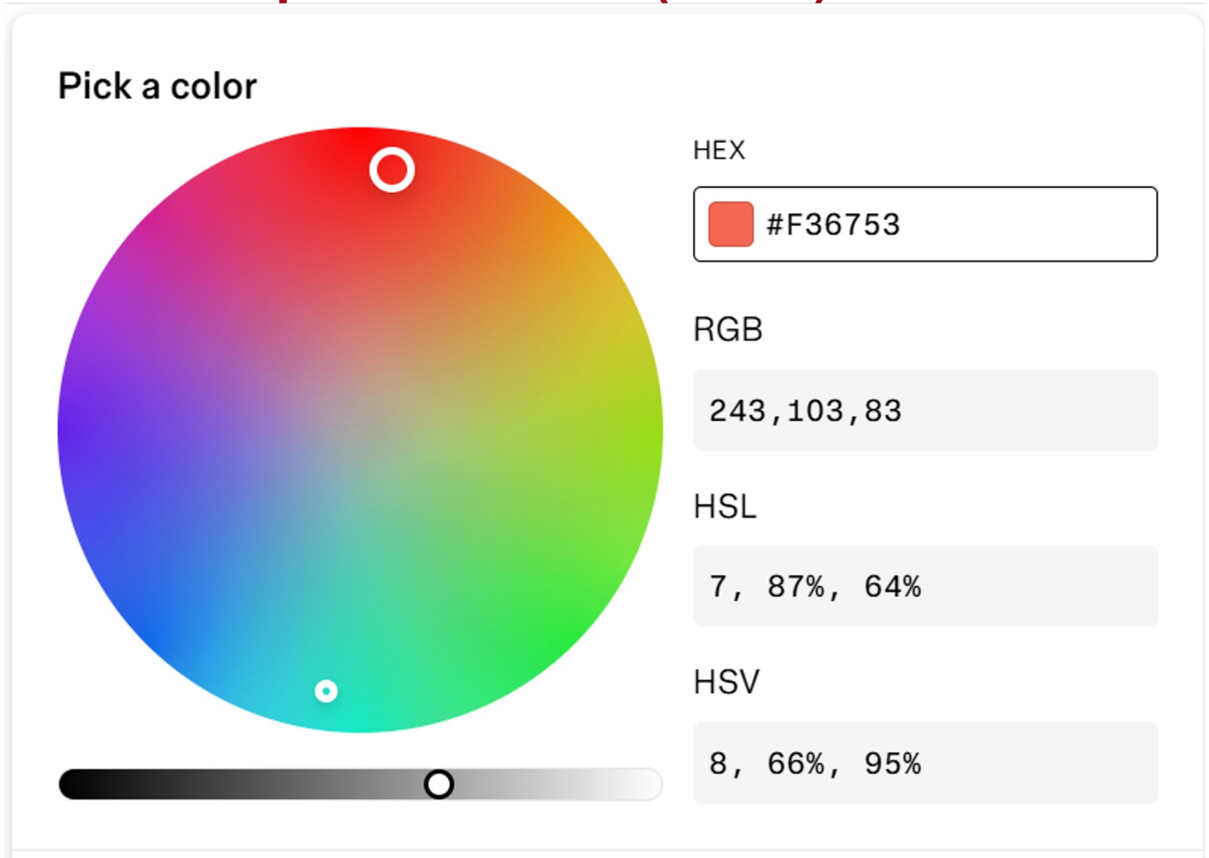
Discuss: why CS loves Hex(decimal) coding

`0b`:0011100100101111111010

`0x`:392FA

Example: why CS loves Hex(decimal) coding

- Example: RGB (8bit) color code or Hex code



https://www.figma.com/color-wheel/

#F36753

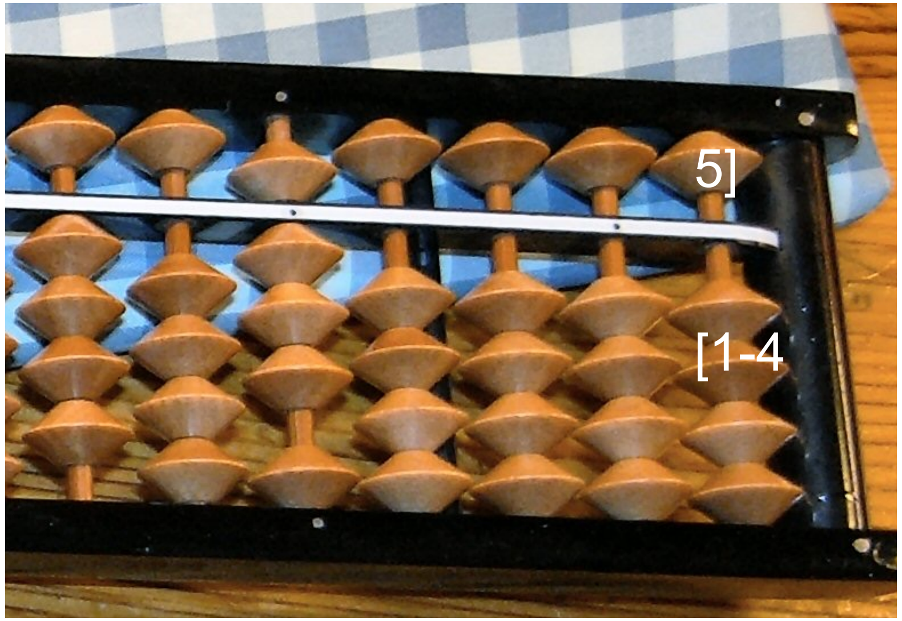
R:
Bin(243)=11110011

G:
Bin(103)=01100111

B: Bin(83)=01010011

COE COLLEGE

Before logic gates: why abacus, again?



https://upload.wikimedia.org/wikipedia/commons/9/98/Soroban_%28Abacus%29.JPG

Or presented in this way [0,1]

These bits are for counting "not 5".

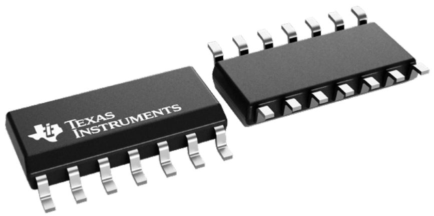
These bits are for counting "5".

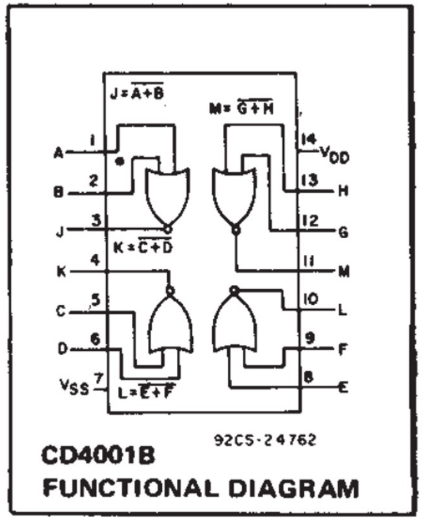
- This is forcing more states in a bit.
- Or due to the polarity of [0,1], it is a state vector.
- State vector is a useful tool for cutting-edge computing.

COE COLLEGE

Step 1-2: store data and move data around

- The basic functional unit for digital electronics: gate





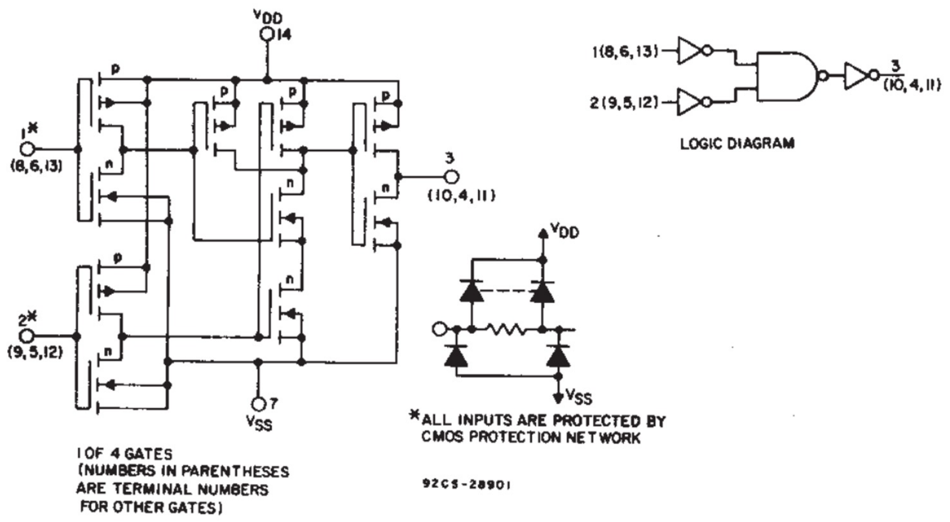
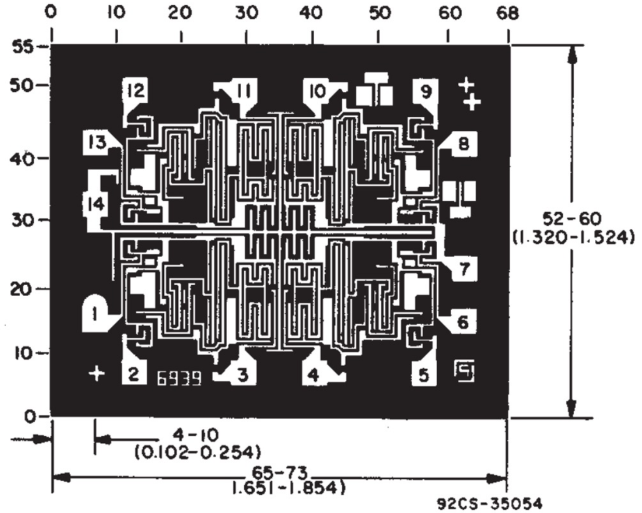
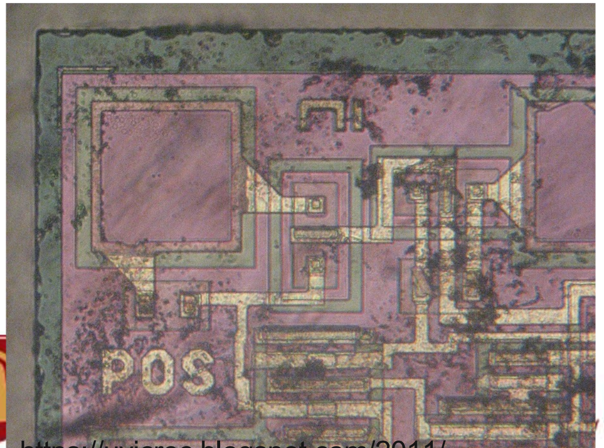


Fig.5 — Schematic and logic diagrams for CD4001B.



CD4001B

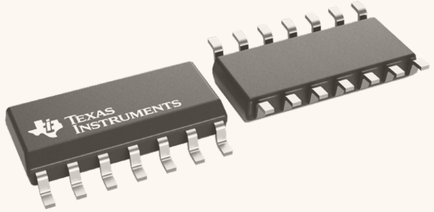


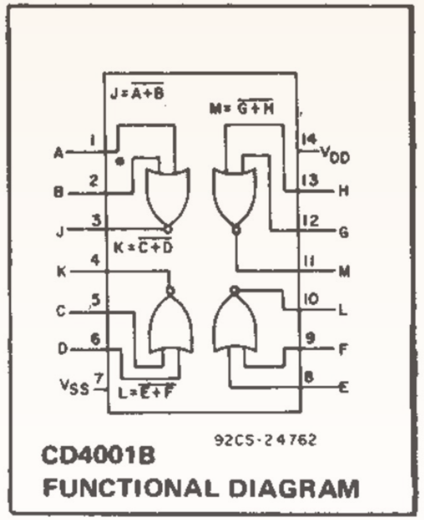
https://www.ti.com/product/CD4001B/part-details/CD4001BM

https://uvicrec.blogspot.com/2011/03/studying-cd4001.html

BTW: the multi-staged abstraction:

Application	EE	CE	MicroE





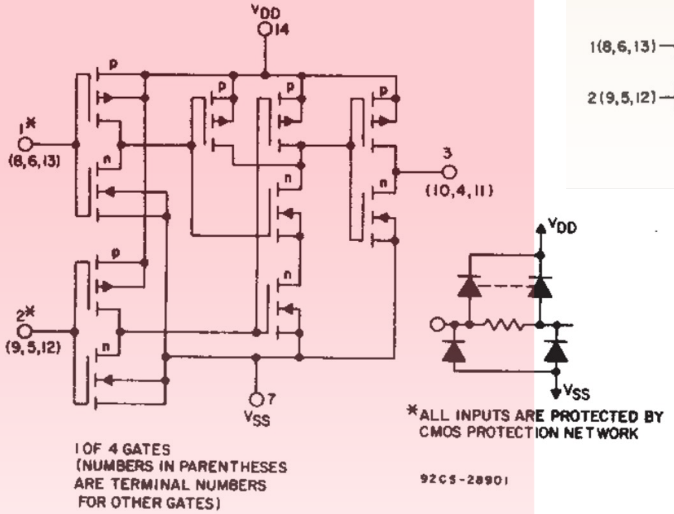
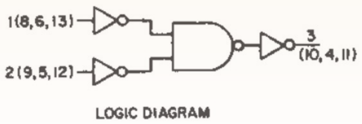
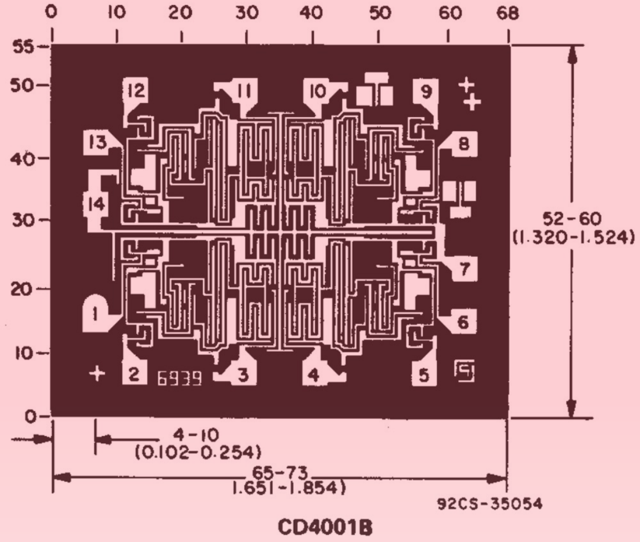
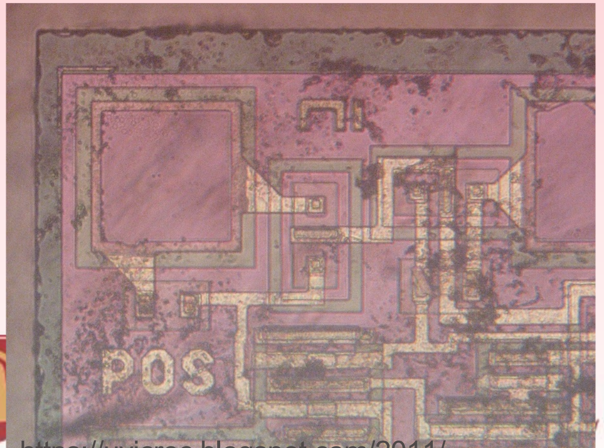


Fig.5 — Schematic and logic diagrams for CD4001B.







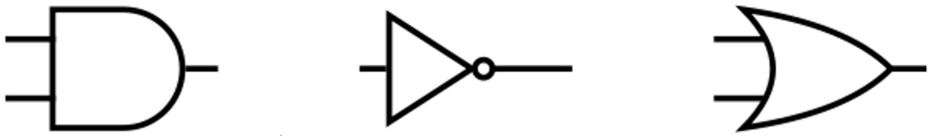
https://uvicrec.blogspot.com/2011/03/studying-cd4001.html

https://www.ti.com/product/CD4001B/part-details/CD4001BM

Step 1: data manipulation

- Boolean operations 1st gen: basic True or False algebra

AND NOT OR



AND gate truth table		
Input		Output
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Inverter truth table	
Input	Output
A	NOT A
0	1
1	0

OR gate truth table		
Input		Output
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Boolean arithmetic symbols are messy

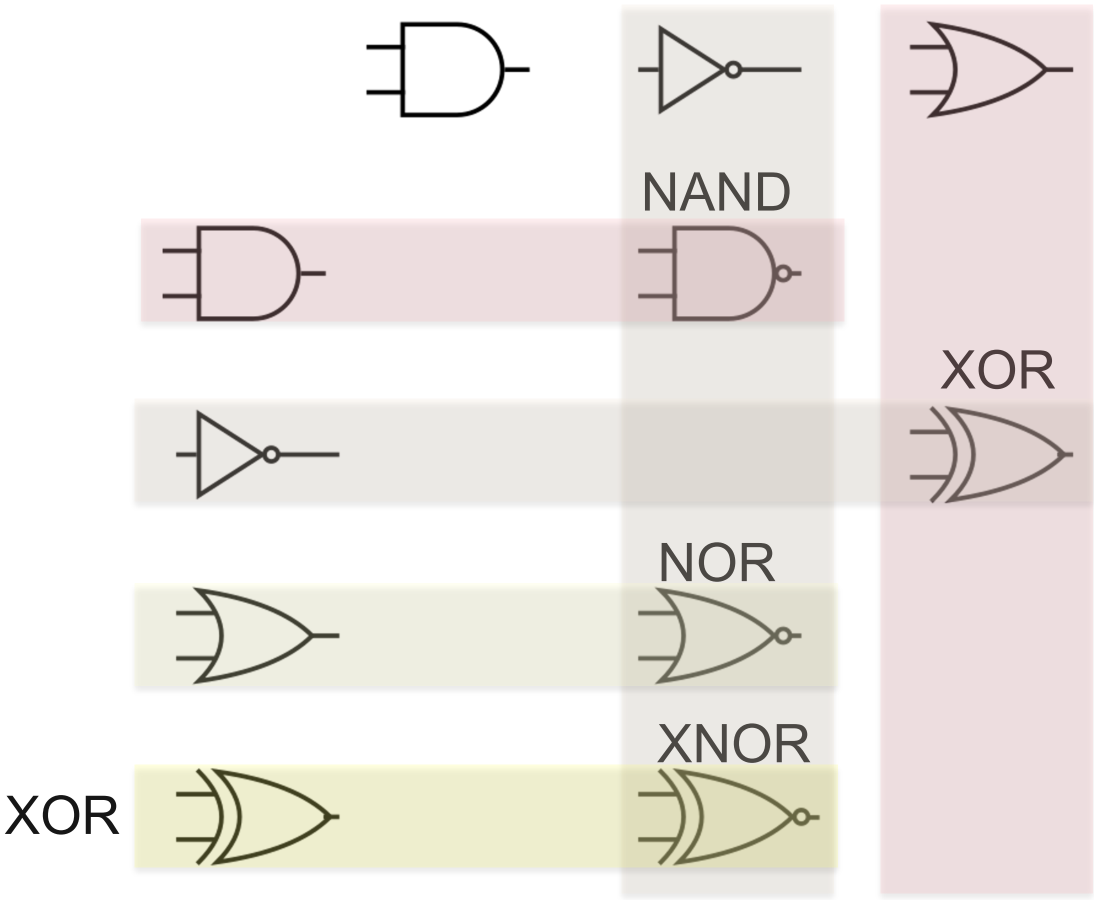
	Viable symbols
AND	● * × & ∧
NOT	! - _ ‘ ¬
OR	+ | ∨

COE COLLEGE

Truth table from: https://en.wikipedia.org/wiki/~_gate

Step 1: data manipulation

- Boolean operations 2nd gen:
  “logical logic operation”



“reverse AND”

NAND gate truth table

Input		Output
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

“reverse OR”

NOR gate truth table

Input		Output
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

“exclusive OR”

XOR gate truth table

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

“reverse XOR”

XNOR gate truth table

Input		Output
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

COE COLLEGE

Truth table from: https://en.wikipedia.org/wiki/~_gate

Logic operation in the early days

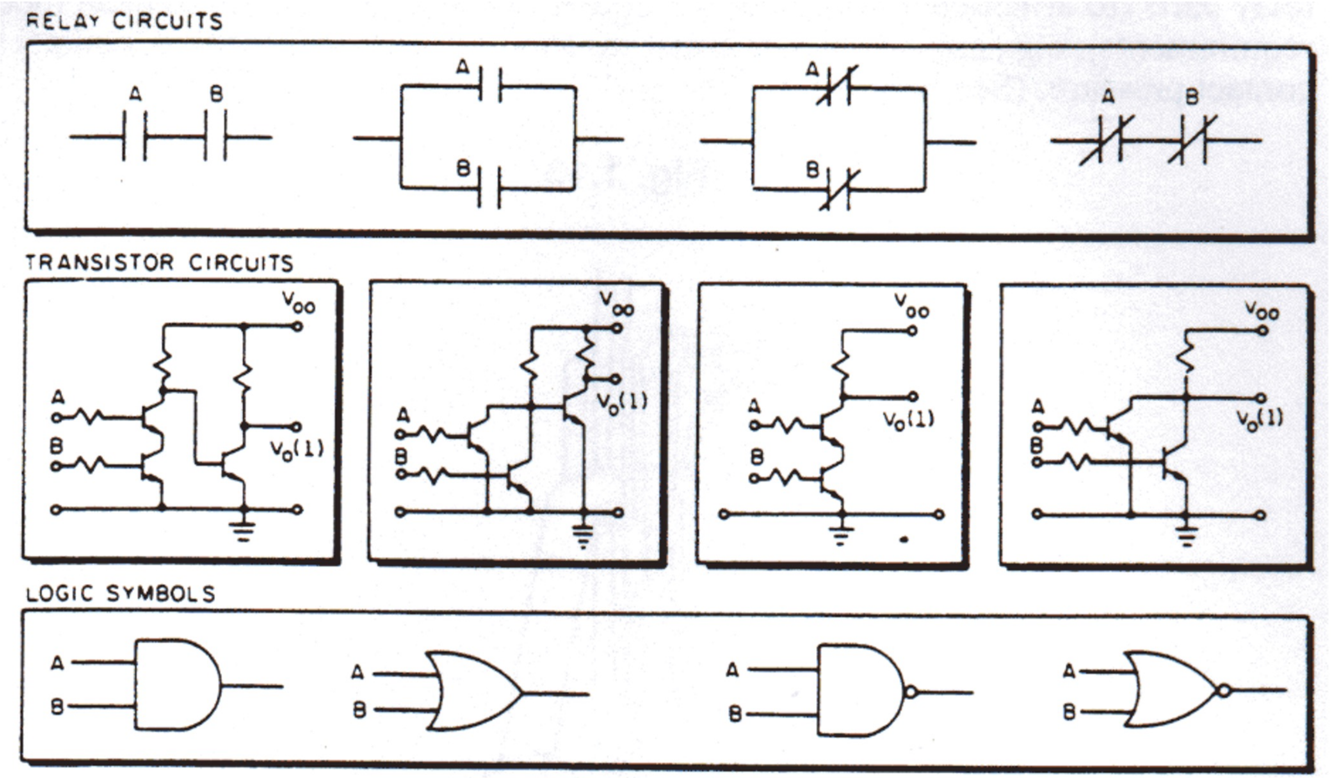


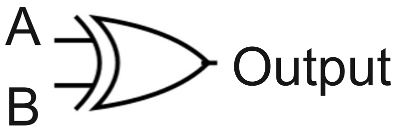
Fig. 1.12 Symbols used in cam-operated timer control.

Transistor–transistor logic (TTL) built with BJT

Engineer's Relay Handbook, 5th edition, Relay and Switch Industry Association (RSIA), 1966

Step 1: data manipulation

- A great example of logic gate functions:



XOR gate truth table

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Parity check?

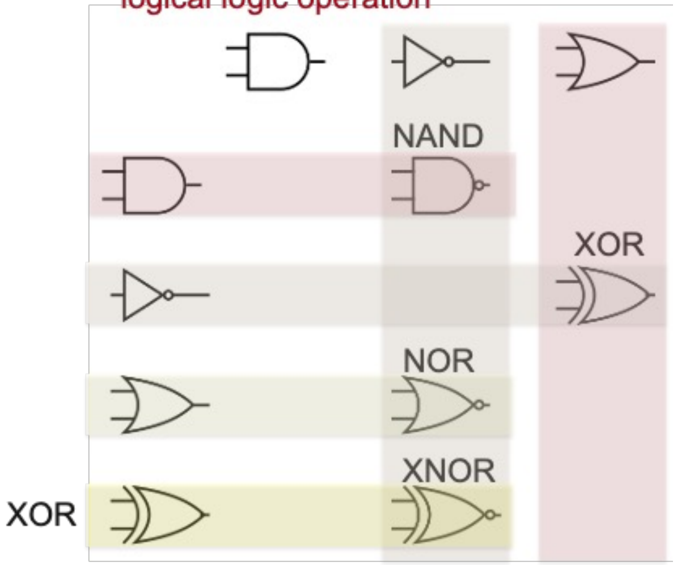
Coe College

Truth table from: https://en.wikipedia.org/wiki/~_gate

“Universal Gate”

- We can use AND, OR, and NOT to build any gates:

  - Boolean operations 2nd gen: “logical logic operation”



- We can build any gates with NAND gate:

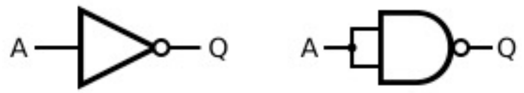
https://en.wikipedia.org/wiki/NAND_logic

COE COLLEGE

"Universal Gate"

- We can build any gates with NAND gate:

Desired NOT Gate | **NAND Construction**

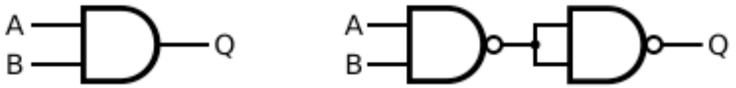


Q = NOT(**A**) = **A** NAND **A**

Truth Table

Input A	Output Q
0	1
1	0

Desired AND Gate | **NAND Construction**

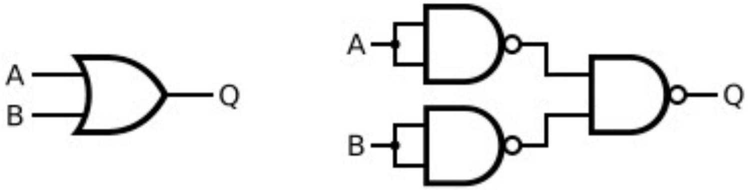


Q = **A** AND **B** = (**A** NAND **B**) NAND (**A** NAND **B**)

Truth Table

Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

Desired OR Gate | **NAND Construction**



Q = **A** OR **B** = (**A** NAND **A**) NAND (**B** NAND **B**)

Truth Table

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

https://en.wikipedia.org/wiki/NAND_logic

Step 1: data manipulation

- One can dig deeper into the Boolean operations:

Boolean expressions of the 16 functions between two variables.

Function Name	Function description	Boolean Expression
Null	FALSE (0)	0
AND	AND	A·B
Inhibition	A NOT B	A/B
Transfer	A	A
Inhibition	B NOT A	B/A
Transfer	B	B
Exclusive-OR	XOR	A⊕B
OR	OR	A+B
NOR	NOR	A↓B
XNOR	XNOR	A⊙B
Complement	NOT B	B'
Implication	A OR NOT B	A+B'
Complement	NOT A	A'
Implication	NOT A OR B	A'+B
NAND	NAND	A↑B
Identity	TRUE (1)	1

Fundamental Theorems & Postulates of Boolean Algebra

Identities:	(1)	X + 0 = X	(1D)	X•1 = X
Null Elements:	(2)	X + 1 = 1	(2D)	X•0 = 0
Indempotency:	(3)	X + X = X	(3D)	X•X = X
Involution (Double Negation):	(4)			(X')' = X
Complements:	(5)	X + X' = 1	(5D)	X•X' = 0
Commutativity:	(6)	X + Y = Y + X	(6D)	X•Y = Y•X
Associativity:	(7)	(X+Y)+Z = X+(Y+Z)	(7D)	(X•Y)•Z = X•(Y•Z)
Distributivity:	(8)	X•Y+X•Z = X•(Y+Z)	(8D)	(X+Y)•(X+Z) = X+Y•Z
Combining:	(9)	X•Y + X•Y' = X	(9D)	(X+Y)•(X+Y') = X
Covering:	(10)	X + X•Y = X	(10D)	X•(X+Y) = X
DeMorgan's Laws:	(12)	(X•Y•Z)' = X'+Y'+Z'	(12D)	(X+Y+Z)' = X'•Y'•Z'
Consensus:	(17)	X•Y+X'•Z+Y•Z = X•Y+X'•Z	(17D)	(X+Y)•(X'+Z)•(Y+Z) = (X+Y)•(X'+Z)

Shannon Expansion: (18) F(X,Y,Z) = X•F(1,Y,Z) + X'•F(0,Y,Z)
(18D) F(X,Y,Z) = (X+F(0,Y,Z))•(X'+F(1,Y,Z))

CS211, Rutgers 2013 notes

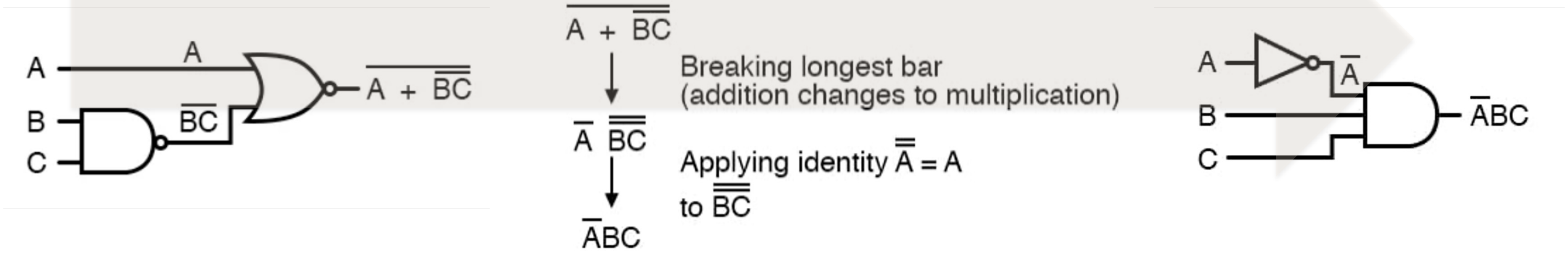
Step 1: data manipulation

But we will only touch base on two:

- De Morgan's laws (1/2)

not (A OR B) = (not A) AND (not B)
not (A AND B) = (not A) OR (not B)

$$\overline{(A \cdot B)} \equiv (\overline{A} + \overline{B})$$
$$\overline{(A + B)} \equiv (\overline{A} \cdot \overline{B})$$



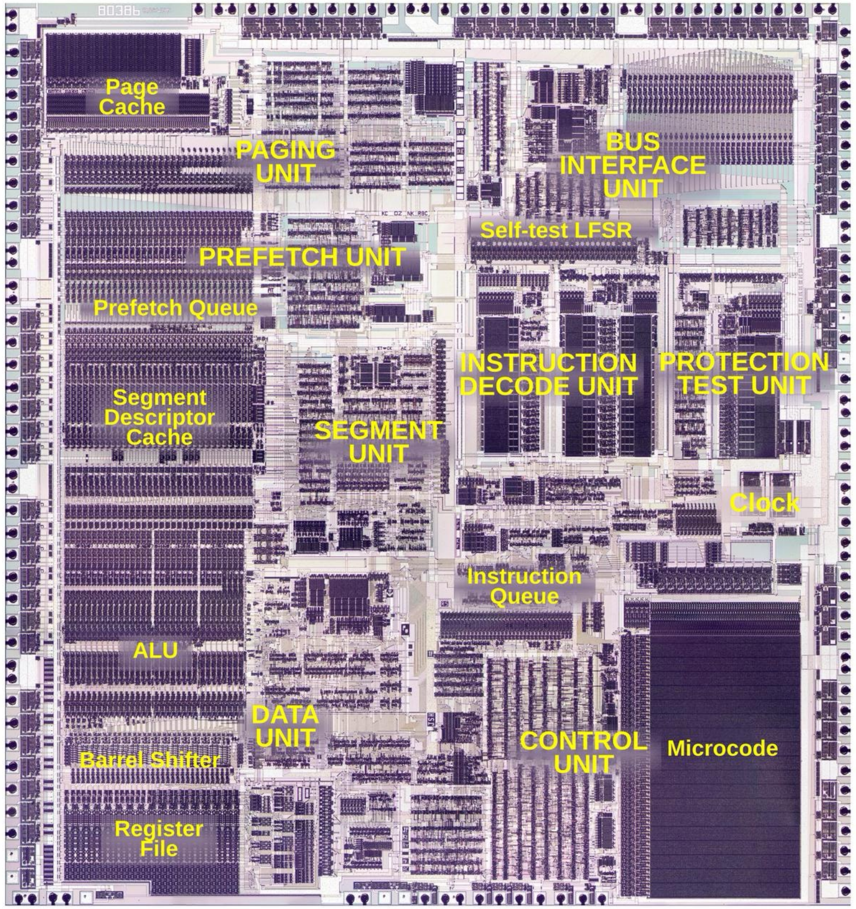
Example from: https://www.allaboutcircuits.com/textbook/digital/chpt-7/demorgans-theorems/

De Morgan's laws is a way to **mathematically** simplified a circuit, but not always realistic for IC.

COE COLLEGE

This is the true logic simplification for IC

Remember XOR?



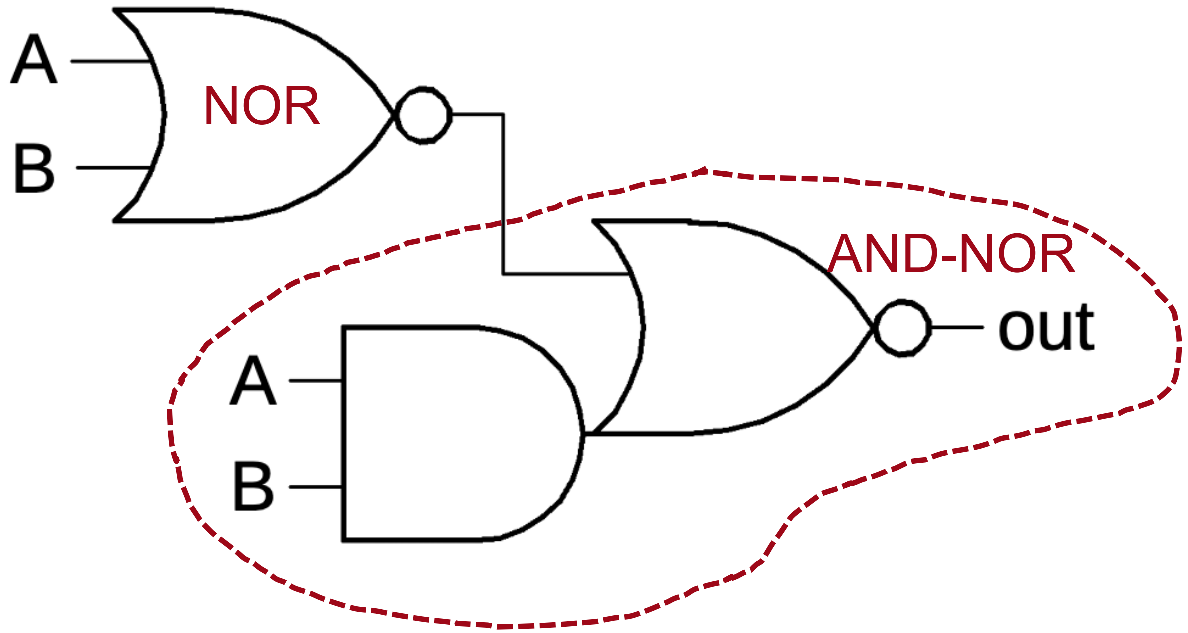
https://www.righto.com/2023/12/386-xor-circuits.html

This Intel's 386 processor (1985) sure has a lot of it.

COE COLLEGE

This is the true logic simplification for IC

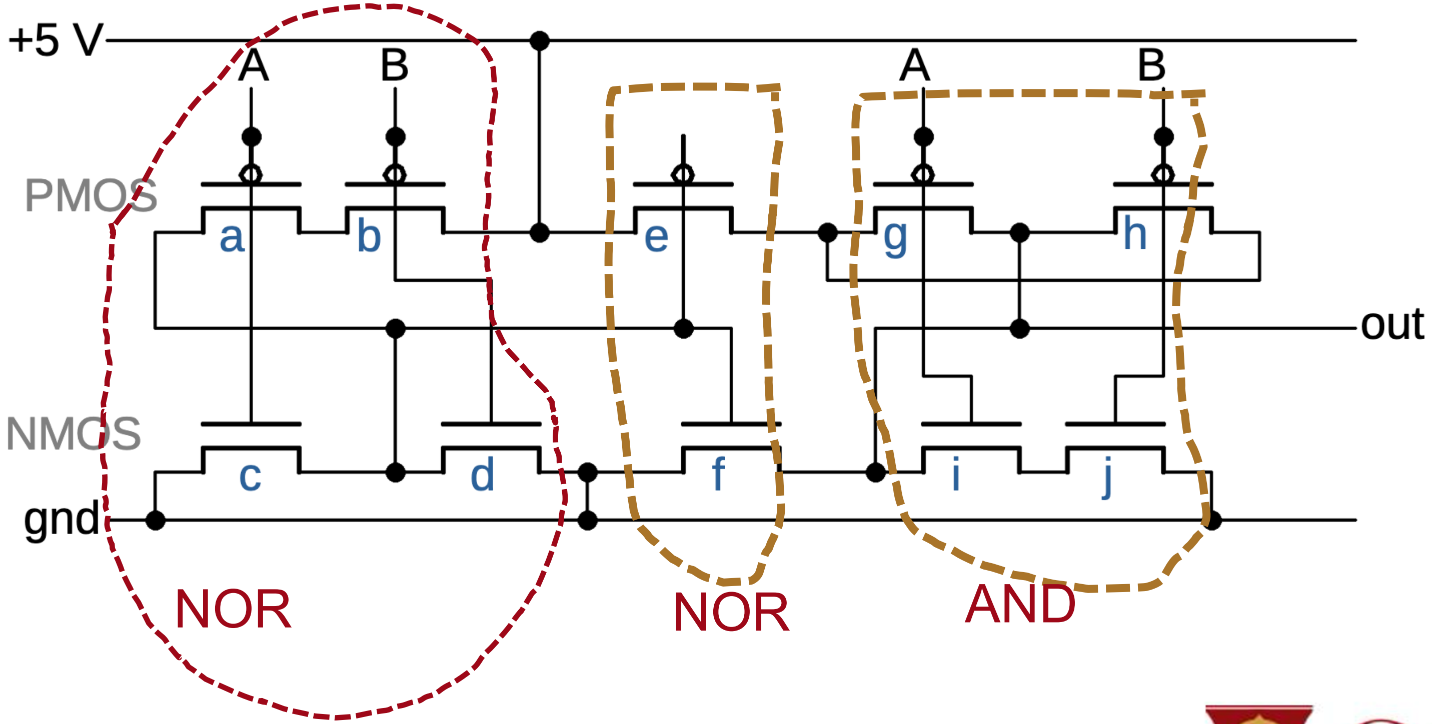
This is the logic gate it used to generate XOR:



https://www.righto.com/2023/12/386-xor-circuits.html

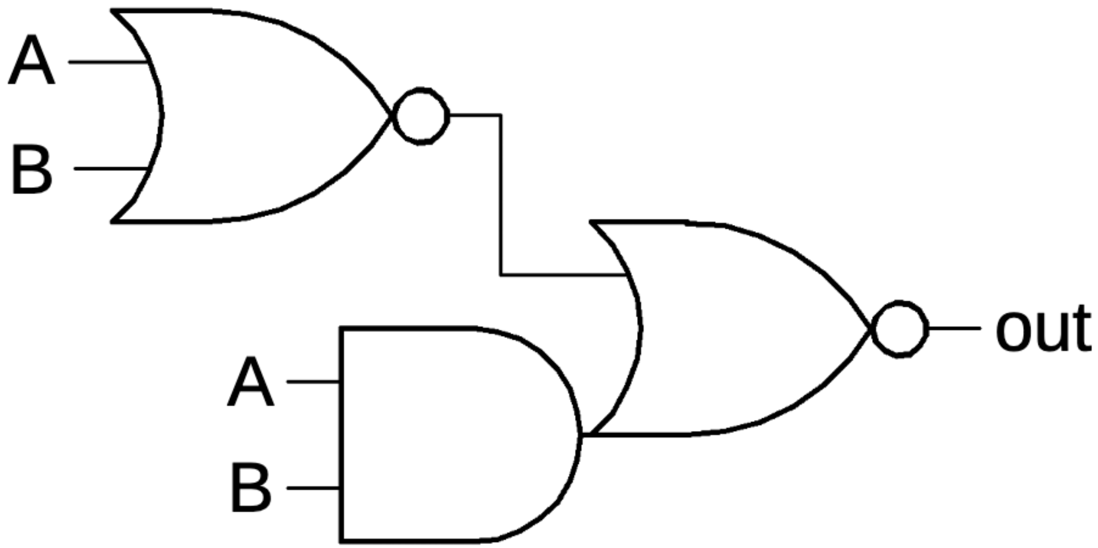
This is the true logic simplification for IC

This is the transistor layout it used to generate XOR:

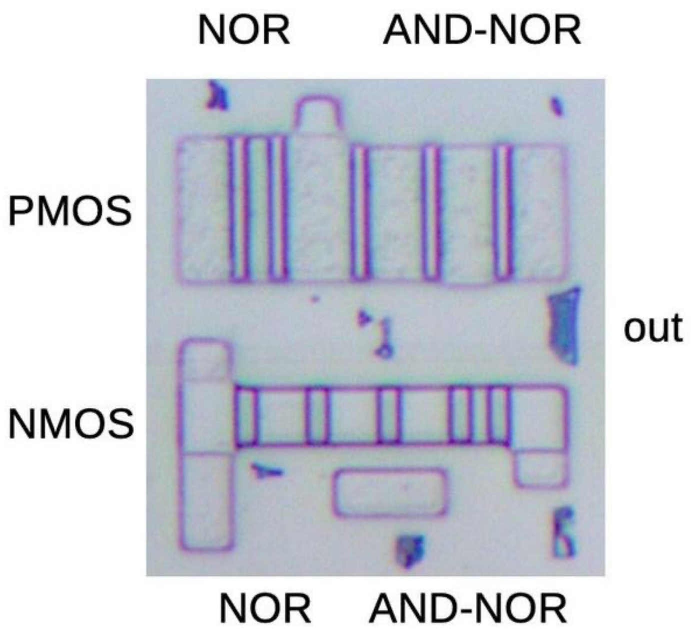


https://www.righto.com/2023/12/386-xor-circuits.html

This is the true logic simplification for IC



This is the actual XOR gate on IC



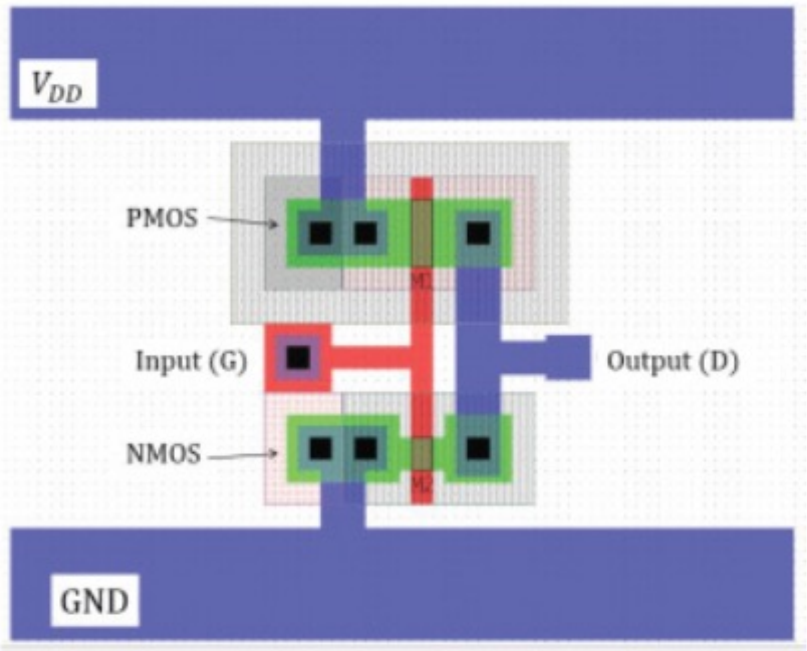
COE COLLEGE

https://www.righto.com/2023/12/386-xor-circuits.html

This is the true logic simplification for IC

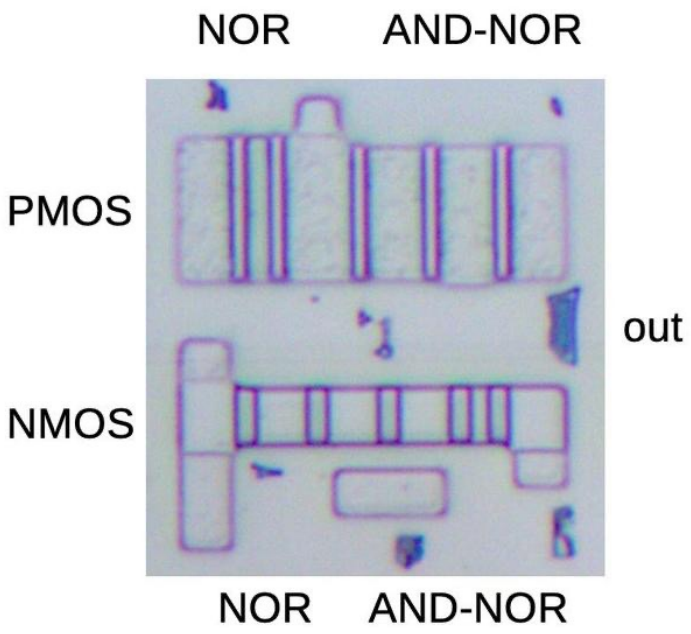
P well and N well design wise, not much differ from this invertor design:

CMOS invertor including electrodes



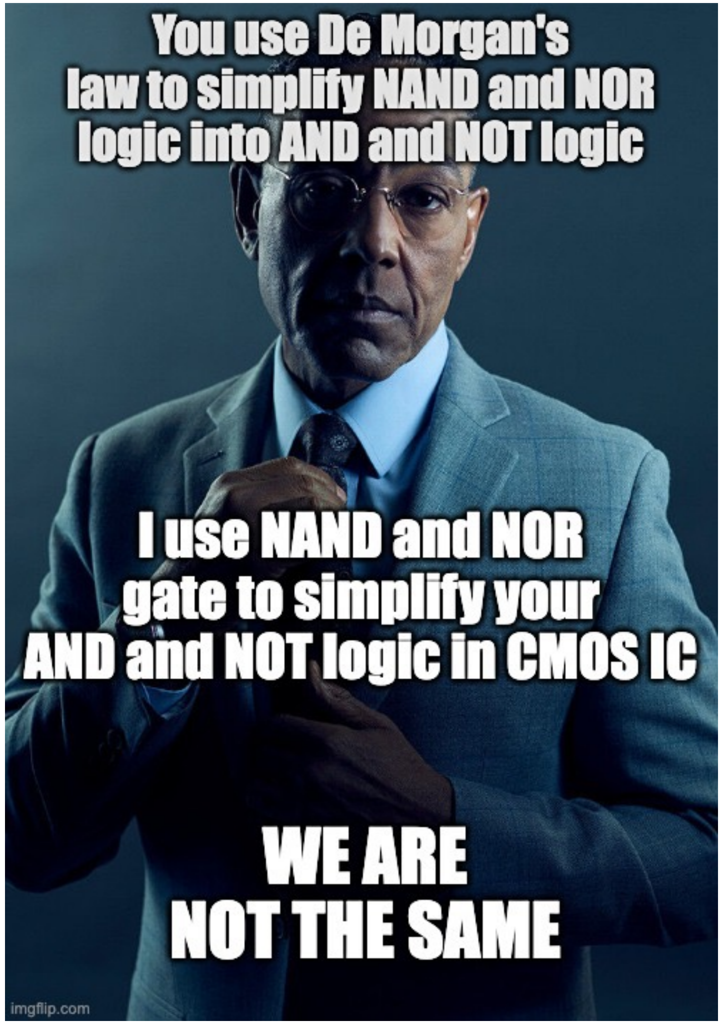
DOI: 10.1145/2755563

This is the actual XOR gate on IC

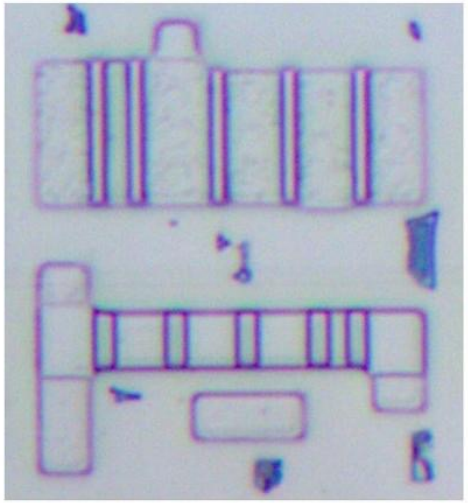


https://www.righto.com/2023/12/386-xor-circuits.html

This is the true logic simplification for IC



This is the actual XOR gate on IC



https://www.righto.com/2023/12/386-xor-circuits.html

COE COLLEGE

Step 1: data manipulation

But we will only pouch base on two:

- Karnaugh map (K-map) (2/2)

NAND gate truth table		
Input		Output
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

K-map of NAND gate

B \ A	0	1
0	**1**	**1**
1	**1**	**0**

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

Grouping rules: https://physics.umd.edu/~drew/spr07/Karnaugh%20Maps%20-%20Rules%20of%20Simplification.htm

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), continued

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups possible.

(A **AND** B) **OR** (**NOT**C **AND** D)

(A×B)+(!C ×D)

	AB / CD	00	01	11	10
!C×!D	00	**0**	**0**	**1**	**0**
!C×D	01	**1**	**1**	**1**	**1**
C×D	11	**0**	**0**	**1**	**0**
C×!D	10	**0**	**0**	**1**	**0**

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), in class practice

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. **Only power of 2 number of cells in each group.**
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. **Wrap around allowed.**
8. Fewest number of groups possible.

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

Coe College

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), in class practice

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. **Only power of 2 number of cells in each group.**
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. Overlapping allowed.
7. **Wrap around allowed.**
8. Fewest number of groups possible.

(!A × !C × D)+(A × !D)

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

COE COLLEGE

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), 1 more in class practice

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. **Only power of 2 number of cells in each group.**
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. **Overlapping allowed.**
7. Wrap around allowed.
8. Fewest number of groups possible.

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

Step 1: data manipulation

- Karnaugh map (K-map) (2/2), 1 more in class practice

K-map grouping rules

1. No zeros allowed.
2. No diagonals.
3. **Only power of 2 number of cells in each group.**
4. Groups should be as large as possible.
5. Every one must be in at least one group.
6. **Overlapping allowed.**
7. Wrap around allowed.
8. Fewest number of groups possible.

(A × !B)+ (!A × B) + C

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

COE COLLEGE

Step 1: data manipulation

- Other than K-map, the more mechanical way to turn truth map to Boolean equations:

Step 1

A	B	C	
0	0	**1**	C_0=!A ×!B
0	1	**1**	C_1=!A ×B
1	0	**1**	C_2=A ×!B
1	1	**0**	

Step 2

C = C_0 + C_1 + C_2

Step 3

C = !A ×!B + !A ×B + A ×!B
= !A(!B + B)+A ×!B   Complements: (5) X + X' = 1
= !A +A ×!B
= !A × (1+ ! B) +A × !B   Null Elements: (2) X + 1 = 1
= !A + !A ! B +A !B
= !A + (!A +A) × !B
=!A + !B

- This expression is also called **Sum of Products (SOP)**.
- There are also **Product of Sums**, for sure.

COE COLLEGE