

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

Xiang Li
Fall 2025

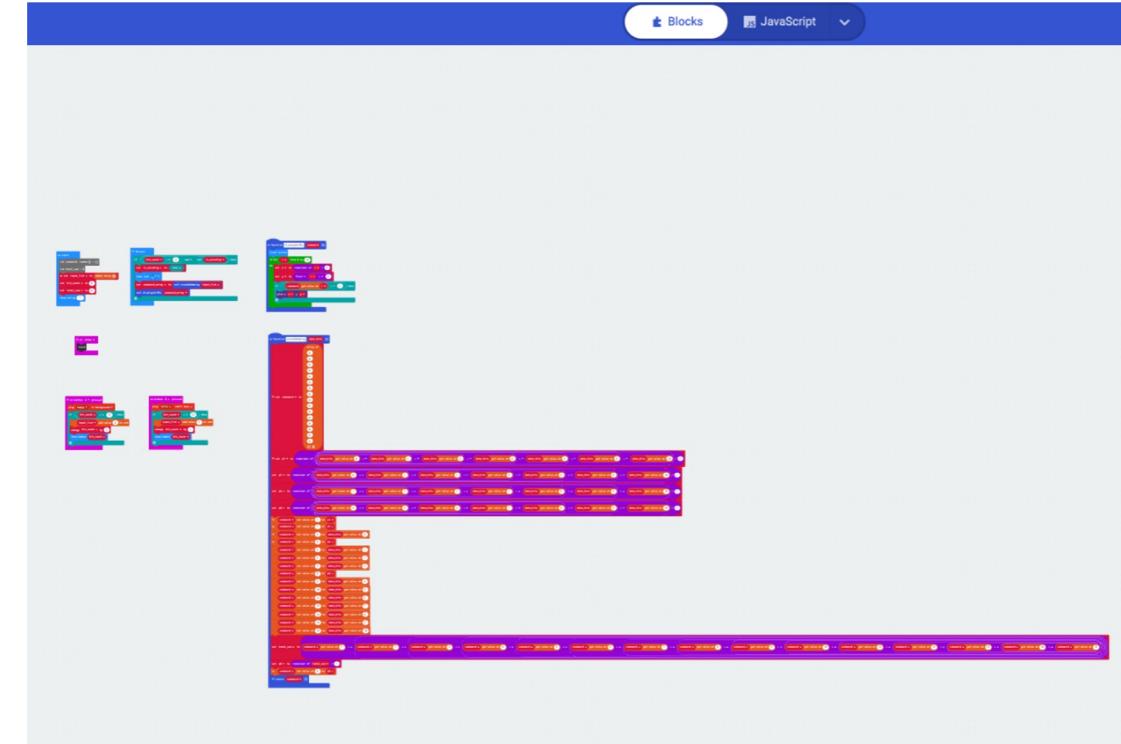


COE COLLEGE®

Hamming codes can be done in the CS way

The screenshot shows the Microsoft MakeCode IDE interface. On the left, there's a preview of a micro:bit board with red LEDs lit up in a pattern. The main area contains two tabs: 'Blocks' (selected) and 'JavaScript'. Below the tabs is a search bar and a sidebar with categories like Basic, Input, Music, Led, Radio, Loops, Logic, Variables, Math, and Extensions. The 'Advanced' category is expanded. The code itself is a JavaScript file with numerous comments explaining the logic of Hamming code generation and decoding.

```
1 // --- Input Handling ---
2 input.onButtonPressed(button.A, function () {
3   music.play(music.builtinPlayableSoundEffect(soundExpression.happy), music.PlaybackMode.untilDone)
4   if (bitCount < 1) {
5     bitCount = 1
6     basic.showNumber(bitCount)
7   }
8 })
9 // --- MODIFIED: Function to display the 16-bit result ---
10 function displayOnLEDs(codeword: number[]) {
11   basic.clearScreen()
12   // Now we need to display all bits
13   for (let i = 0; i < 16; i++) {
14     x = i % 4
15     y = Math.floor(i / 4)
16     if (codeword[i] == 1) {
17       led.plot(x, y)
18     }
19   }
20 }
21 input.onButtonPressed(button.B, function () {
22   music.play(music.builtinPlayableSoundEffect(soundExpression.hello), music.PlaybackMode.untilDone)
23   if (bitCount < 1) {
24     bitCount = 1
25     basic.showNumber(bitCount)
26   }
27 })
28 // --- Reset ---
29 input.onGesture(Gesture.Shake, function () {
30   control.reset()
31 })
32 // --- MODIFIED: Hamming Encoder Function for (16,11) ---
33 // Now returns a 16-bit code with PB as the first bit.
34 // PB is the parity bit, which is the sum of all other bits (PB = sum of all 15 other bits (11 data + 4 parity) modulo 2.
35 // Function encodinghamming(dataBits: number[]): number[] {
36   // Create an array for the 16-bit codeword, initialized with zeros.
37   codeword = [0,
38   0,
39   0,
40   0,
41   0,
42   0,
43   0,
44   0,
45   0,
46   0,
47   0,
48   0,
49   0,
50   0,
51   0,
52   0,
53   0,
54   0,
55   0,
56   0]
57 // ... Step 1: Calculate the four standard parity bits (d0, d1, p0, p1) ...
58 p0 = (dataBits[0] + dataBits[1] + dataBits[2] + dataBits[3] + dataBits[4] + dataBits[5] + dataBits[6] + dataBits[7] + dataBits[8]) % 2
59 p1 = (dataBits[0] + dataBits[1] + dataBits[2] + dataBits[3] + dataBits[4] + dataBits[5] + dataBits[6] + dataBits[7] + dataBits[8] + dataBits[9]) % 2
60 p2 = (dataBits[0] + dataBits[1] + dataBits[2] + dataBits[3] + dataBits[4] + dataBits[5] + dataBits[6] + dataBits[7] + dataBits[8] + dataBits[9] + dataBits[10]) % 2
61 p3 = (dataBits[0] + dataBits[1] + dataBits[2] + dataBits[3] + dataBits[4] + dataBits[5] + dataBits[6] + dataBits[7] + dataBits[8] + dataBits[9] + dataBits[10] + dataBits[11]) % 2
62 // ... Step 2: Place data and parity bits, shifting by 1 for PB ...
63 // The standard Hamming code now occupies indices 1 through 15.
64 // The standard Hamming code now occupies indices 1 through 15.
65 codeword[0] = p0
66 codeword[1] = dataBits[0]
67 codeword[2] = p1
68 codeword[3] = dataBits[1]
69 codeword[4] = p2
70 codeword[5] = dataBits[2]
71 codeword[6] = p3
72 codeword[7] = dataBits[3]
73 codeword[8] = dataBits[4]
74 codeword[9] = dataBits[5]
75 codeword[10] = dataBits[6]
76 codeword[11] = dataBits[7]
77 codeword[12] = dataBits[8]
78 codeword[13] = dataBits[9]
79 codeword[14] = dataBits[10]
80 codeword[15] = dataBits[11]
```



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



COE COLLEGE®

Pre-step: Data forms

- Say bye-bye to base 10:

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

$$(4321)_{10} =$$
$$\begin{array}{r} 4 \times \\ + 10^3 \end{array} \quad \begin{array}{r} 3 \times \\ + 10^2 \end{array} \quad \begin{array}{r} 2 \times \\ + 10^1 \end{array} \quad \begin{array}{r} 1 \times \\ + 10^0 \end{array}$$

Base 2 (0,1):

$$(1011)_2 =$$
$$\begin{array}{r} 1 \times \\ + 2^3 \end{array} \quad \begin{array}{r} 0 \times \\ + 2^2 \end{array} \quad \begin{array}{r} 1 \times \\ + 2^1 \end{array} \quad \begin{array}{r} 1 \times \\ + 2^0 \end{array}$$

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

$$(FF12)_{16} =$$
$$\begin{array}{r} 15 \times \\ + 16^3 \end{array} \quad \begin{array}{r} 15 \times \\ + 16^2 \end{array} \quad \begin{array}{r} 1 \times \\ + 16^1 \end{array} \quad \begin{array}{r} 2 \times \\ + 16^0 \end{array}$$

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



COE COLLEGE®

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

Base 10

$$\begin{array}{r} 324 \\ +123 \\ \hline \end{array}$$

Base 2

$$\begin{array}{r} 110 \\ +101 \\ \hline \end{array}$$

Base 10

$$\begin{array}{r} 324 \\ \times 123 \\ \hline \end{array}$$

Base 2

$$\begin{array}{r} 110 \\ \times 101 \\ \hline \end{array}$$

$$\begin{array}{r} 324 \\ -123 \\ \hline \end{array}$$

$$\begin{array}{r} 110 \\ -101 \\ \hline \end{array}$$

$$\begin{array}{r} 324 \\ \div 6 \\ \hline \end{array}$$

$$\begin{array}{r} 110 \\ \div 10 \\ \hline \end{array}$$

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



COE COLLEGE®

Discuss: the origin of base 16?



COE COLLEGE®

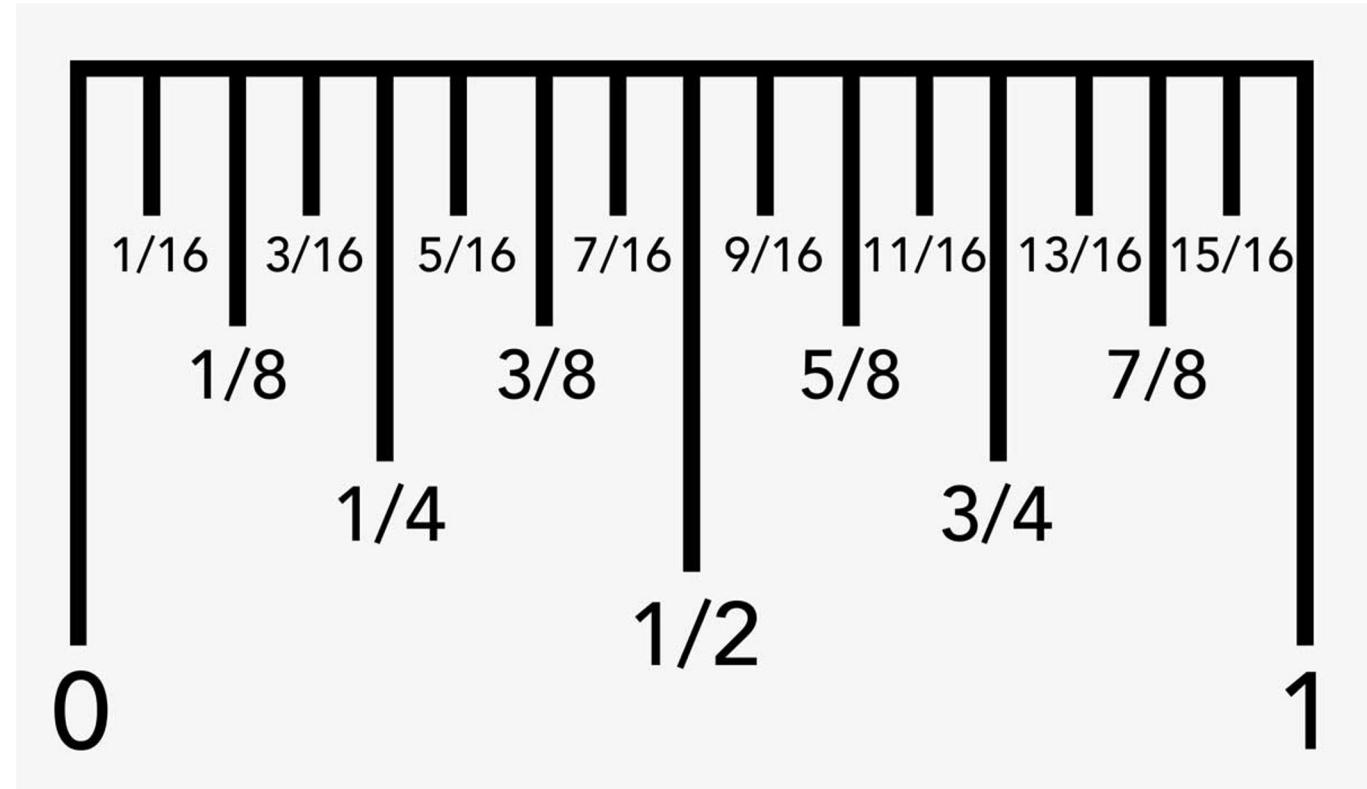
Discuss: the origin of base 16?

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/>



COE COLLEGE®

Discuss: why CS loves Hex(decimal) coding

0b : 00111001001011111010

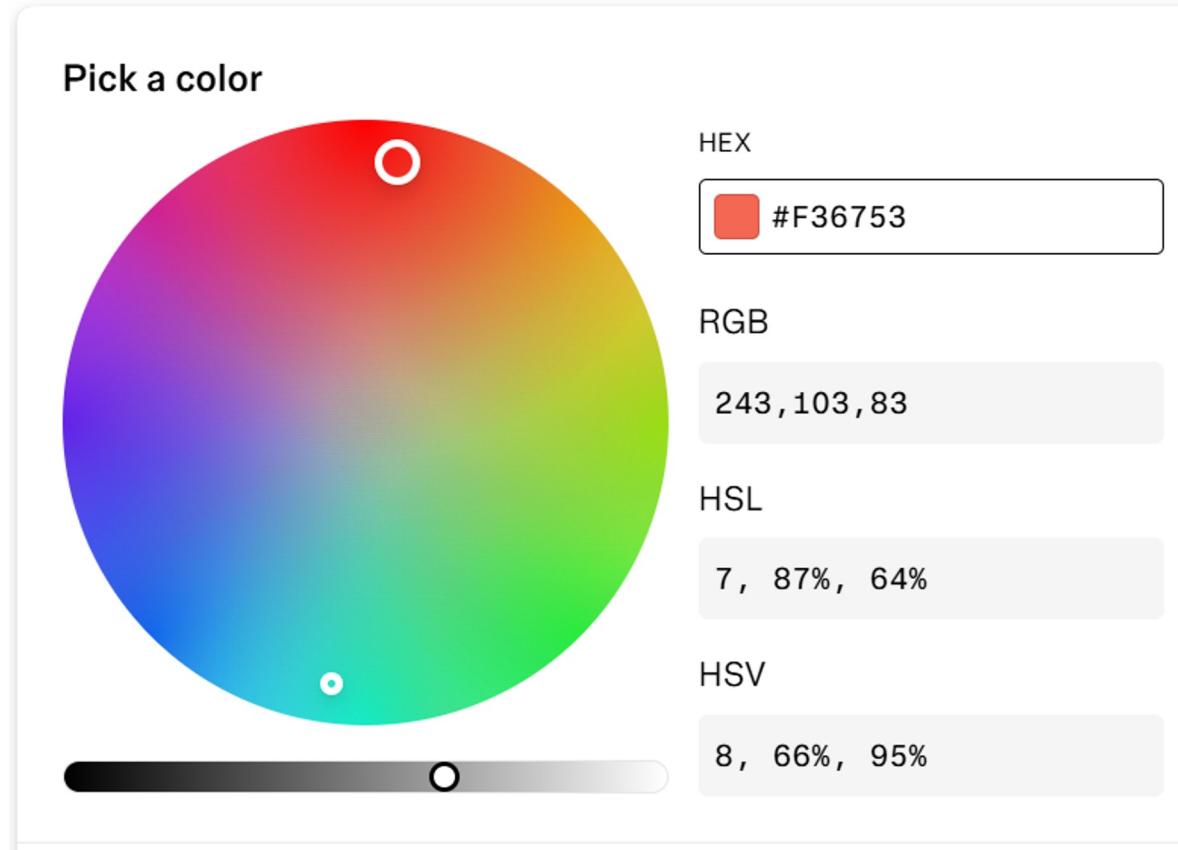
0x : 392FA



COE COLLEGE®

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:

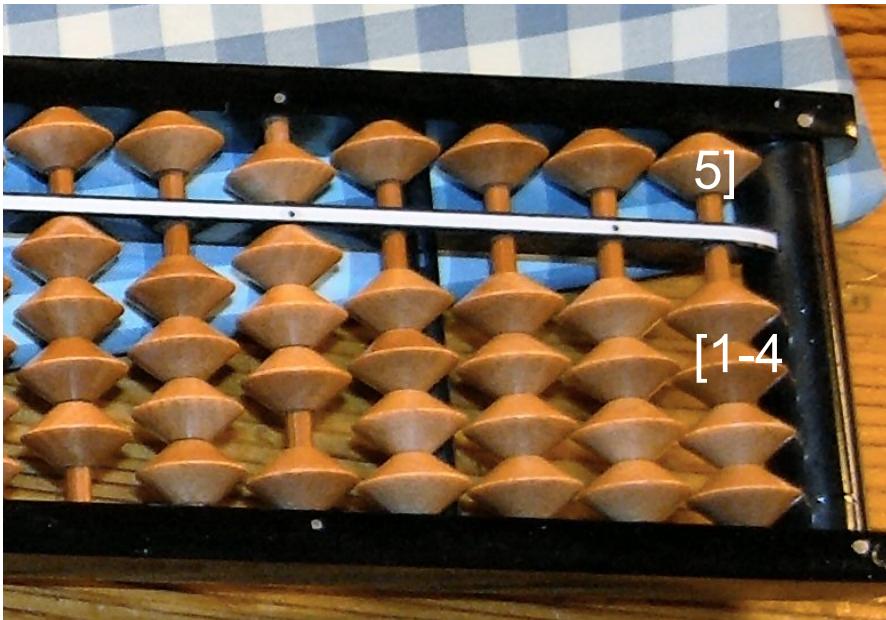
Bin(103)=01100111

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

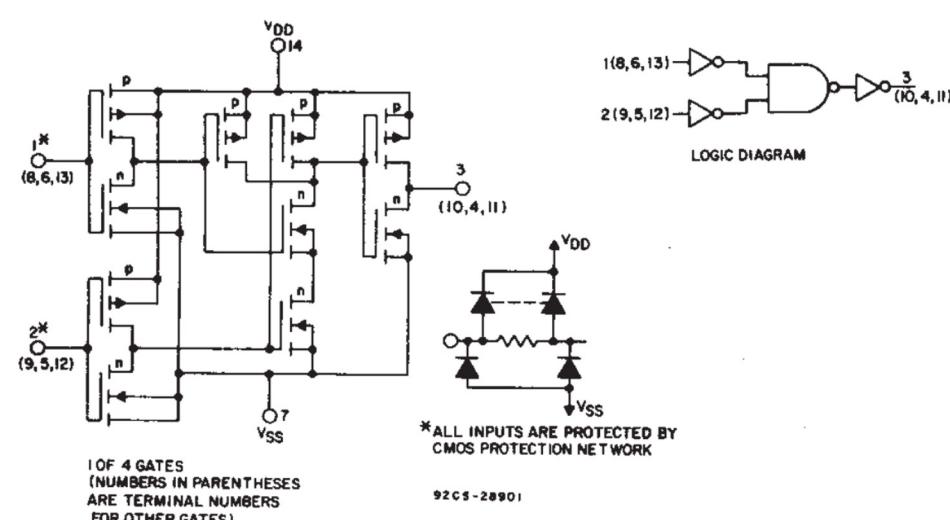
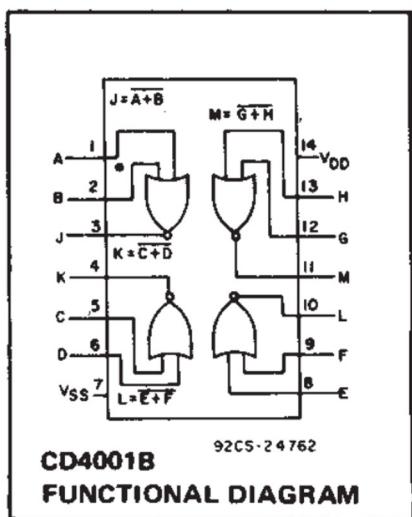
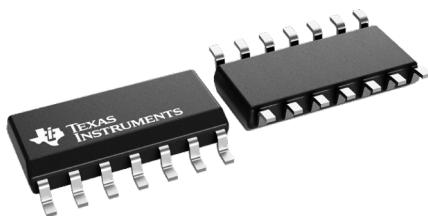
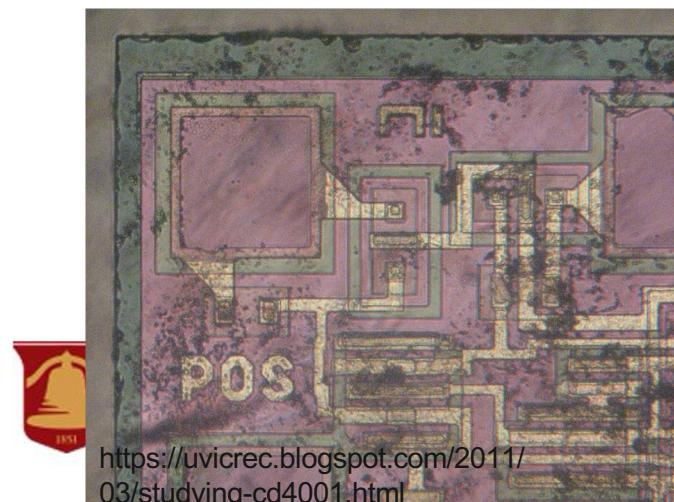
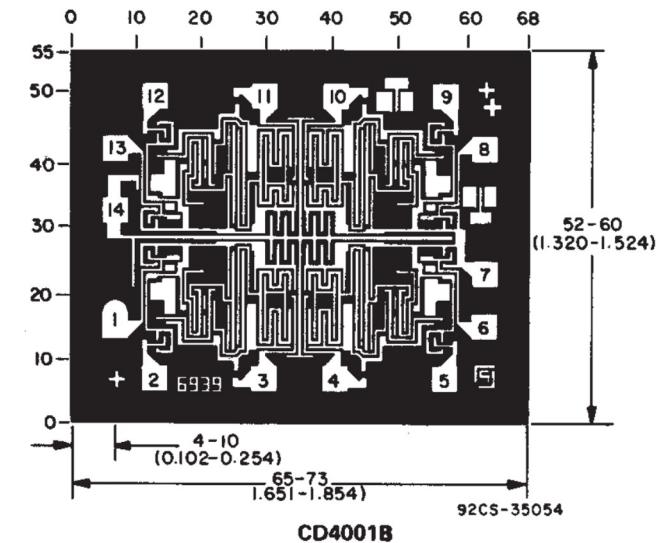


Fig.5 – Schematic and logic diagrams for CD4001B.

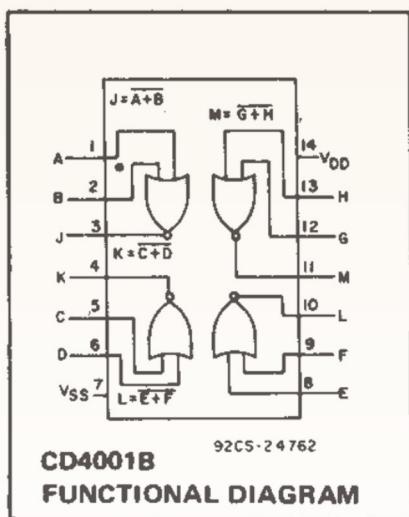
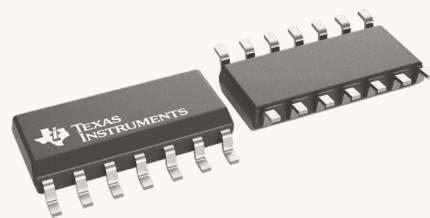


<https://uvicrec.blogspot.com/2011/03/studying-cd4001.html>

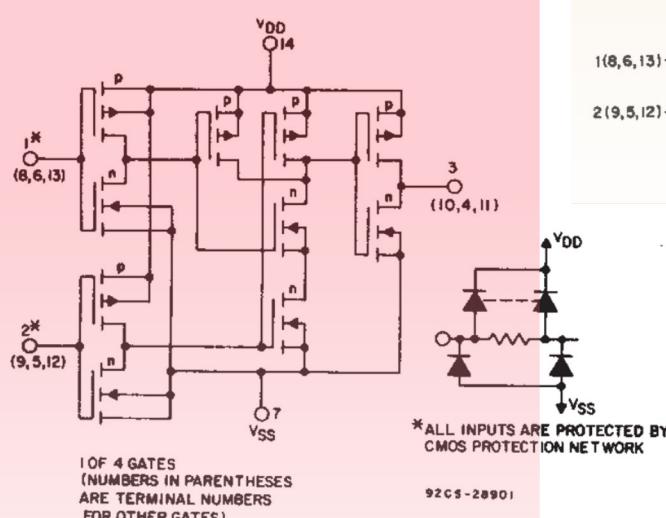
GE®

BTW: the multi-staged abstraction:

Application



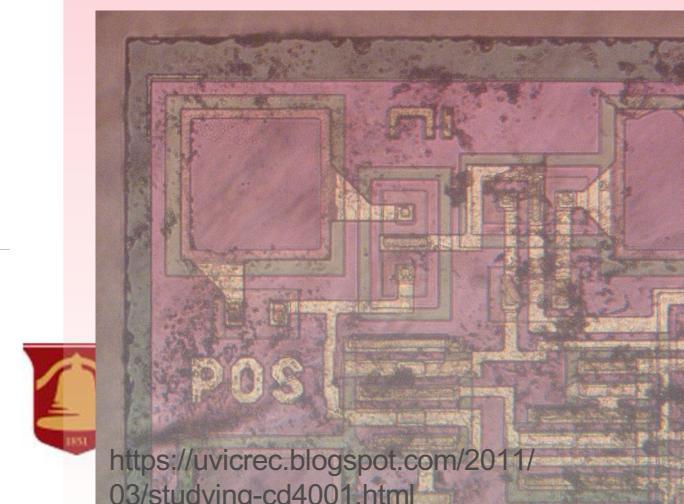
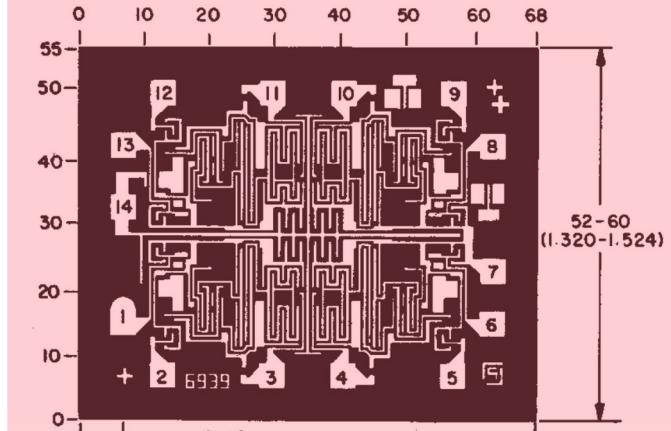
EE



CE



MicroE



Step 1: data manipulation

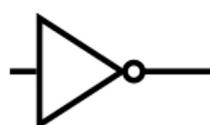
- Boolean operations 1st gen: basic True or False algebra

AND



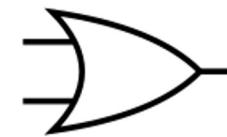
AND gate truth table	
Input	Output
A	A AND B
0	0
0	1
1	0
1	1

NOT



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

OR



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

Boolean arithmetic
symbols are messy

Viable symbols

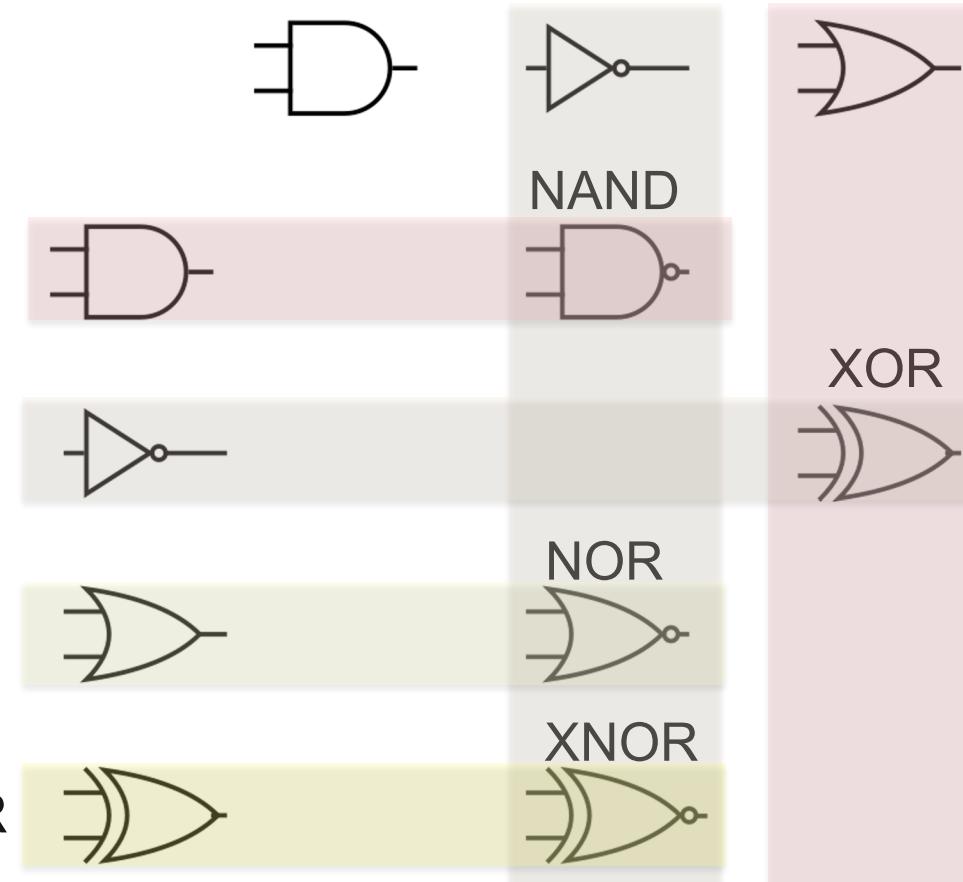
AND	• * × & \wedge
NOT	! - _ ' \neg
OR	+ \vee



COE COLLEGE®

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®

Logic operation in the early days

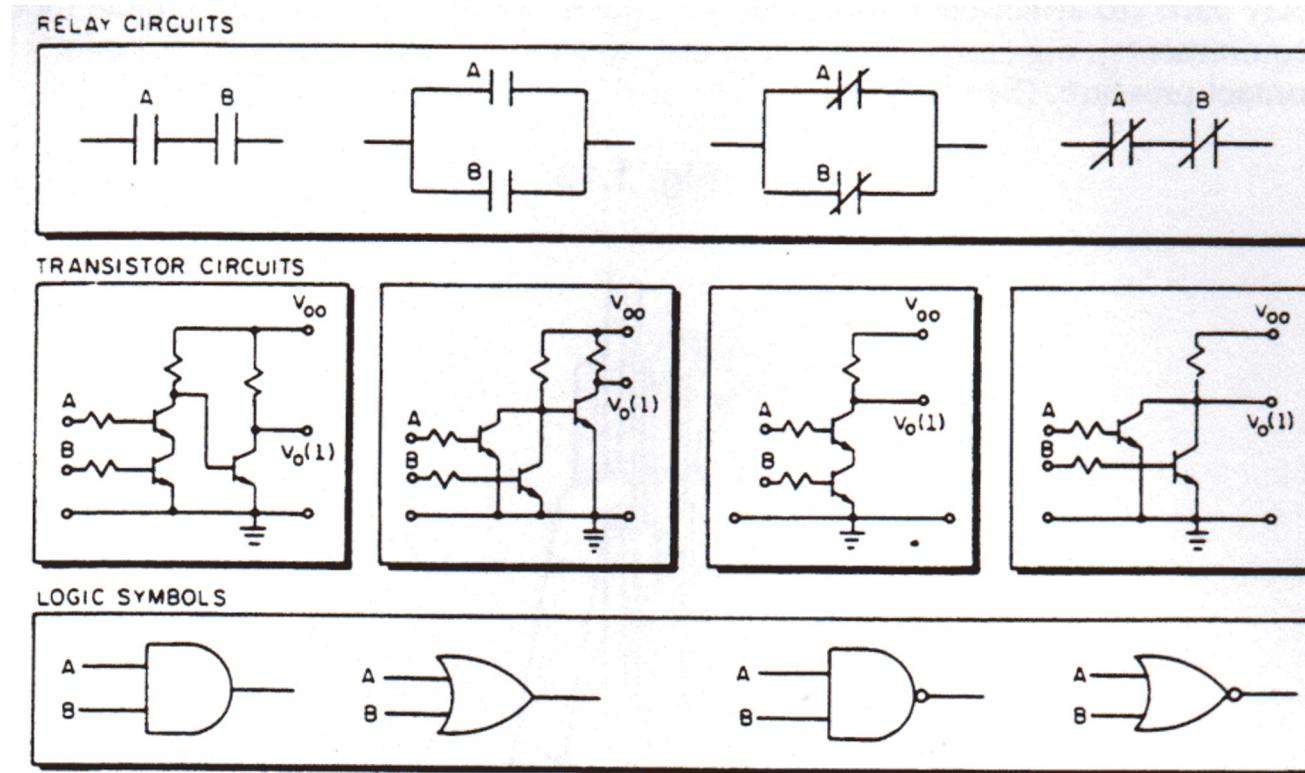


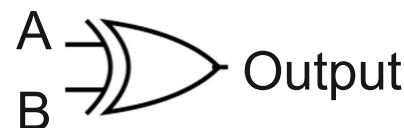
Fig. 1.12 Symbols used in cam-operated timer control.



COE COLLEGE®

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

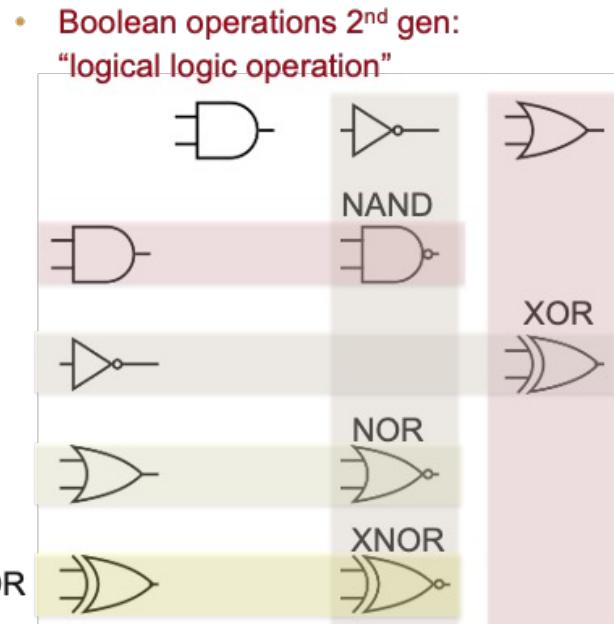
Parity check?



COE COLLEGE®

“Universal Gate”

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



- We can build any gates with NAND gate:

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

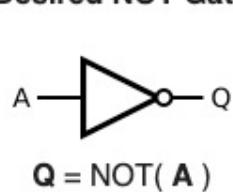


COE COLLEGE®

“Universal Gate”

- We can build any gates with NAND gate:

Desired NOT Gate



NAND
Construction

$$= A \text{ NAND } A$$

$$Q = \text{NOT}(A)$$

Truth Table

Input A	Output Q
0	1
1	0

Desired AND Gate



$$Q = A \text{ AND } B$$

NAND Construction



$$= (A \text{ NAND } B) \text{ NAND } (A \text{ NAND } B)$$

Truth Table

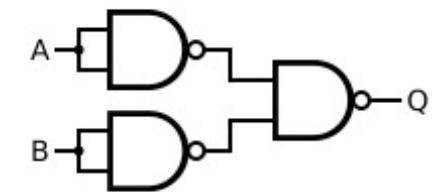
Input A	Input B	Output Q
0	0	0
0	1	0
1	0	0
1	1	1

Desired OR Gate



$$Q = A \text{ OR } B$$

NAND Construction



$$= (A \text{ NAND } A) \text{ NAND } (B \text{ NAND } B)$$

Truth Table

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



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	$A \cdot B$
Inhibition	A NOT B	A/B
Transfer	A	A
Inhibition	B NOT A	B/A
Transfer	B	B
Exclusive-OR	XOR	$A \oplus B$
OR	OR	$A+B$
NOR	NOR	$A \downarrow B$
XNOR	XNOR	$A \ominus B$
Complement	NOT B	B'
Implication	A OR NOT B	$A+B'$
Complement	NOT A	A'
Implication	NOT A OR B	$A'+B$
NAND	NAND	$A \uparrow B$
Identity	TRUE (1)	1

Fundamental Theorems & Postulates of Boolean Algebra

Identities:	(1) $X + 0 = X$	(1D) $X \cdot 1 = X$
Null Elements:	(2) $X + 1 = 1$	(2D) $X \cdot 0 = 0$
Indempotency:	(3) $X + X = X$	(3D) $X \cdot X = X$
Involution (Double Negation):	(4) $(X')' = X$	
Complements:	(5) $X + X' = 1$	(5D) $X \cdot X' = 0$
Commutativity:	(6) $X + Y = Y + X$	(6D) $X \cdot Y = Y \cdot X$
Associativity:	(7) $(X+Y)+Z = X+(Y+Z)$	(7D) $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$
Distributivity:	(8) $X \cdot Y + X \cdot Z = X \cdot (Y+Z)$	(8D) $(X+Y) \cdot (X+Z) = X+Y \cdot Z$
Combining:	(9) $X \cdot Y + X \cdot Y' = X$	(9D) $(X+Y) \cdot (X+Y') = X$
Covering:	(10) $X + X \cdot Y = X$	(10D) $X \cdot (X+Y) = X$
DeMorgan's Laws:	(12) $(X \cdot Y \cdot Z)' = X' + Y' + Z'$	(12D) $(X+Y+Z)' = X' \cdot Y' \cdot Z'$
Consensus:	(17) $X \cdot Y + X \cdot Z + Y \cdot Z = X \cdot Y + X \cdot Z$	(17D) $(X+Y) \cdot (X+Z) \cdot (Y+Z) = (X+Y) \cdot (X+Z)$
Shannon Expansion:	(18) $F(X,Y,Z) = X \cdot F(1,Y,Z) + X' \cdot F(0,Y,Z)$	
	(18D) $F(X,Y,Z) = (X+F(0,Y,Z)) \cdot (X'+F(1,Y,Z))$	

CS211, Rutgers 2013 notes



COE COLLEGE[®]

Step 1: data manipulation

But we will only touch base on two:

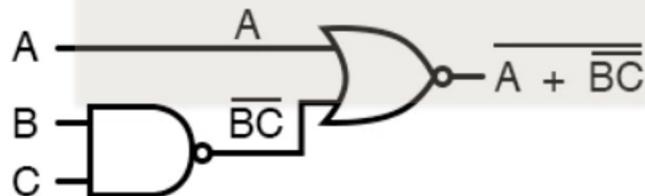
- De Morgan's laws (1/2)

$$\text{not } (A \text{ OR } B) = (\text{not } A) \text{ AND } (\text{not } B)$$

$$\text{not } (A \text{ AND } B) = (\text{not } A) \text{ OR } (\text{not } B)$$

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

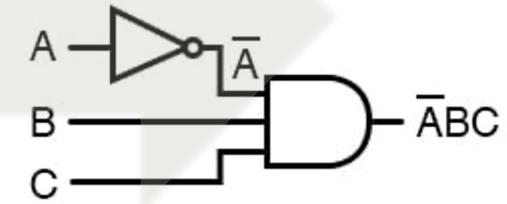
$$\overline{(A + B)} \equiv (\overline{A} \cdot \overline{B})$$



$$\begin{array}{c} \overline{A + \overline{B}C} \\ \downarrow \\ \overline{A} \overline{\overline{B}C} \\ \downarrow \\ \overline{ABC} \end{array}$$

Breaking longest bar
(addition changes to multiplication)

Applying identity $\overline{\overline{A}} = A$
to $\overline{\overline{B}C}$



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

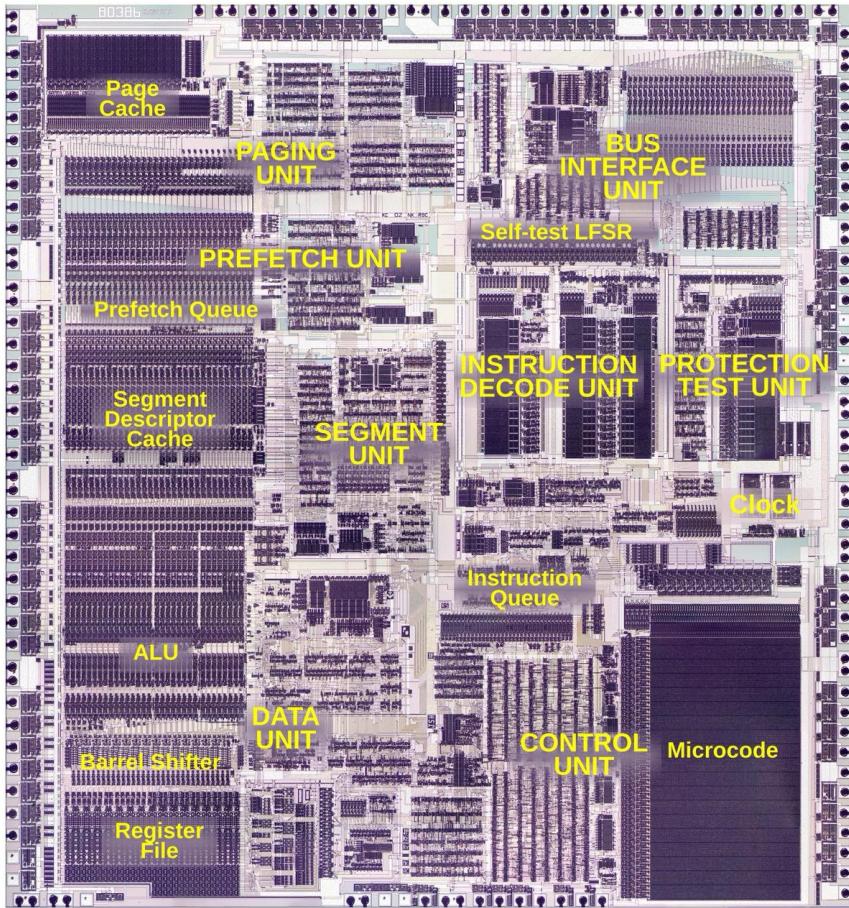
De Morgan's laws is a way to **mathematically**
simplified a circuit, but not always realistic for IC.



COE COLLEGE[®]

This is the true logic simplification for IC

Remember XOR?

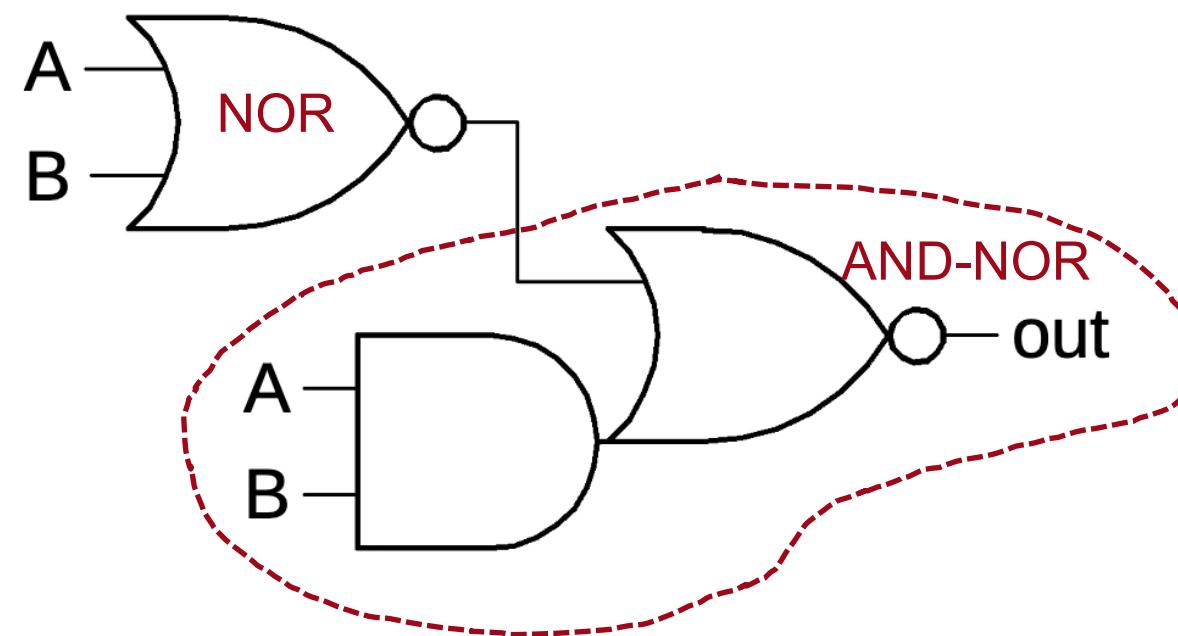


This Intel's 386 processor (1985) sure has a lot of it.



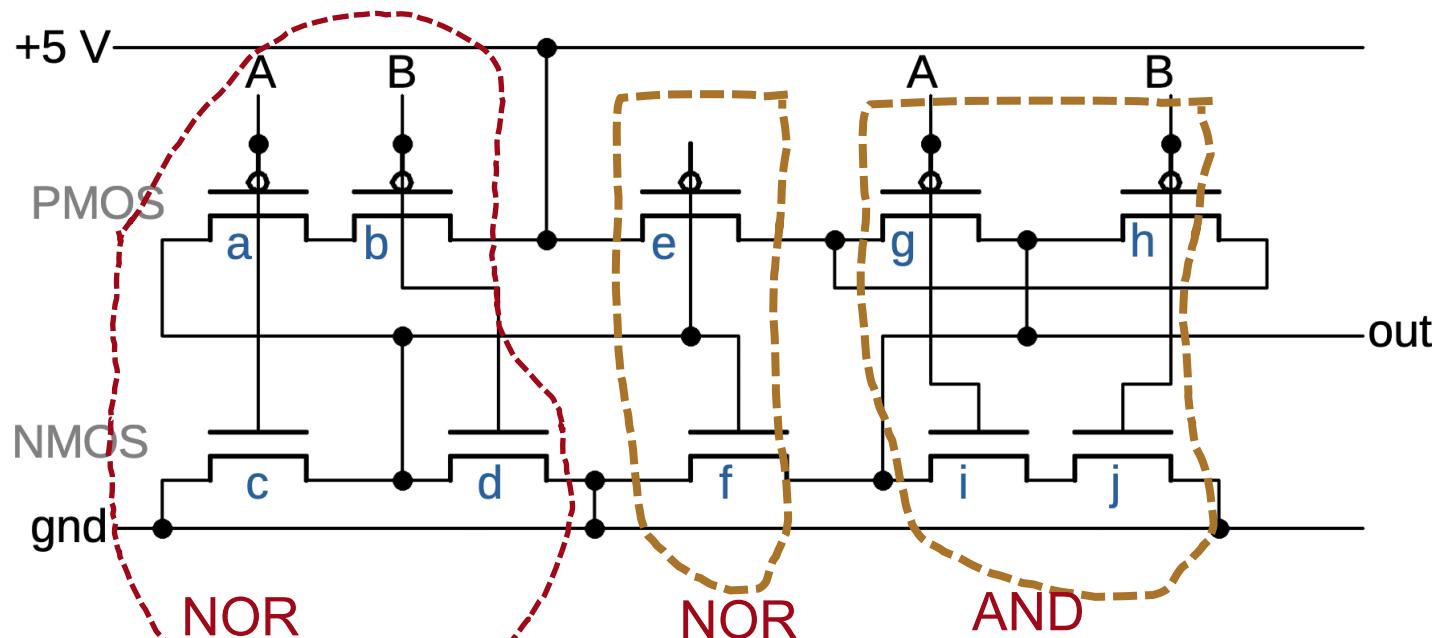
This is the true logic simplification for IC

This is the logic gate it used to generate XOR:



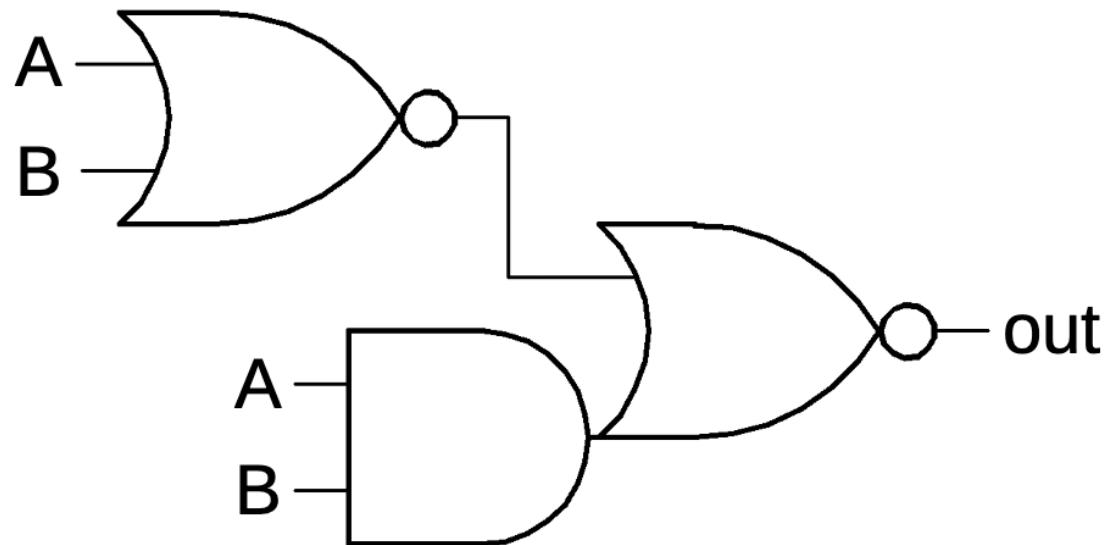
This is the true logic simplification for IC

This is the transistor layout it used to generate XOR:

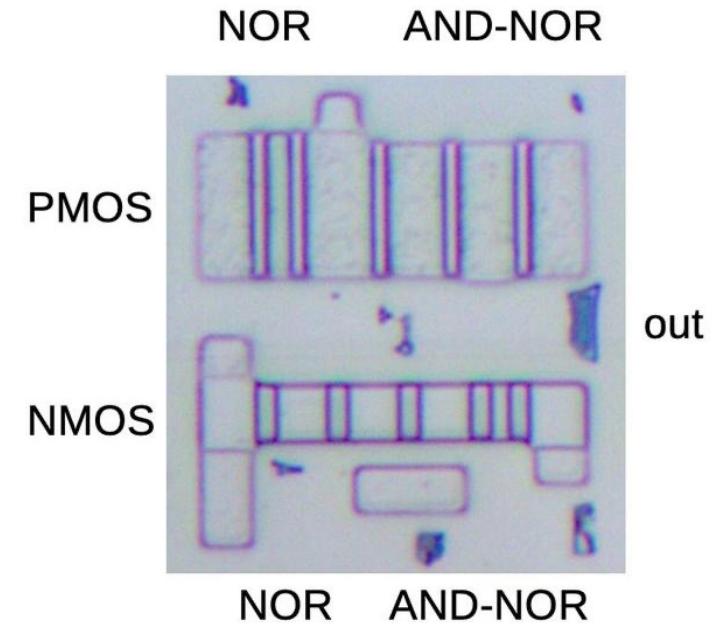


COE COLLEGE.

This is the true logic simplification for IC



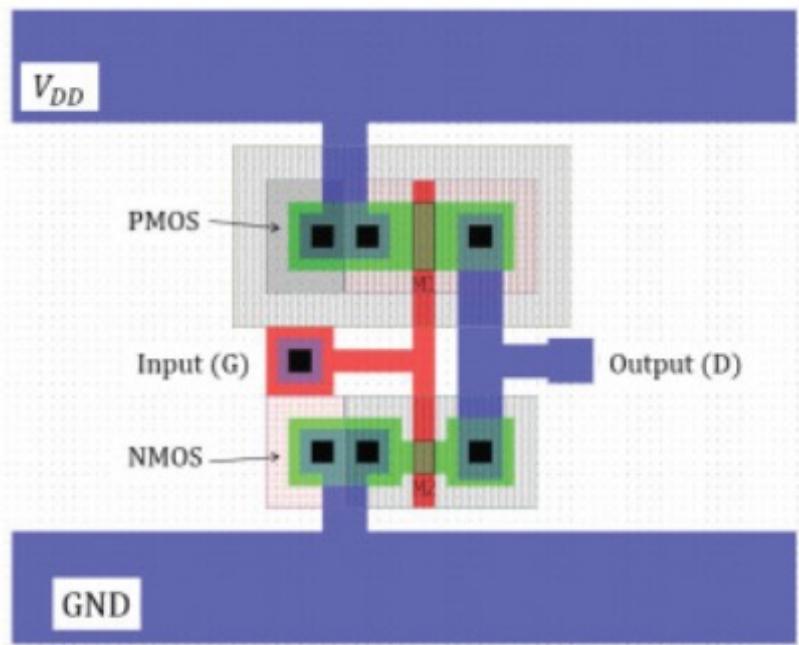
This is the actual XOR gate on IC



This is the true logic simplification for IC

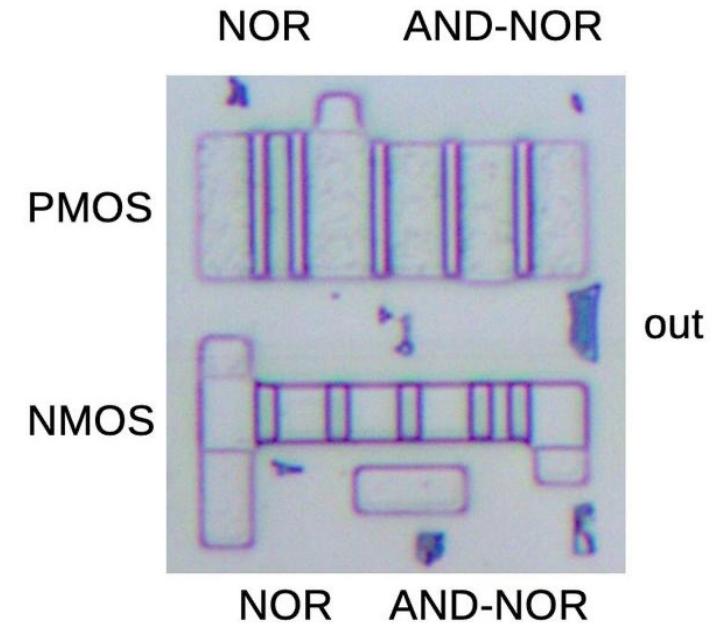
P well and N well design wise, not much differ from this invertor design:

CMOS invertor including electrodes



DOI: 10.1145/2755563

This is the actual XOR gate on IC

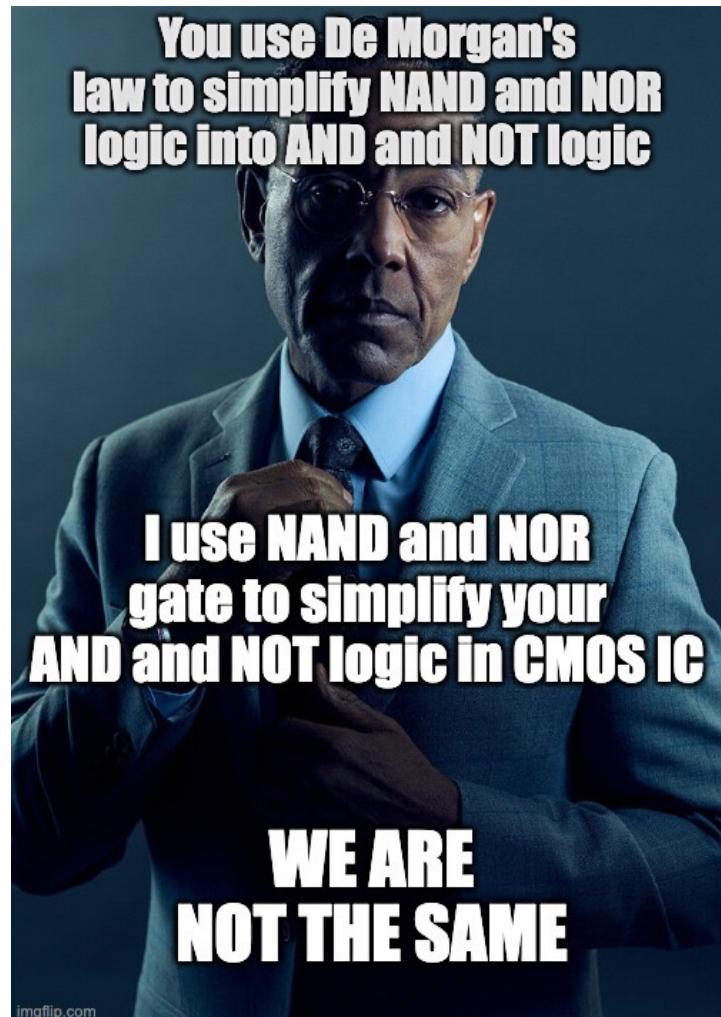


<https://www.righto.com/2023/12/386-xor-circuits.html>

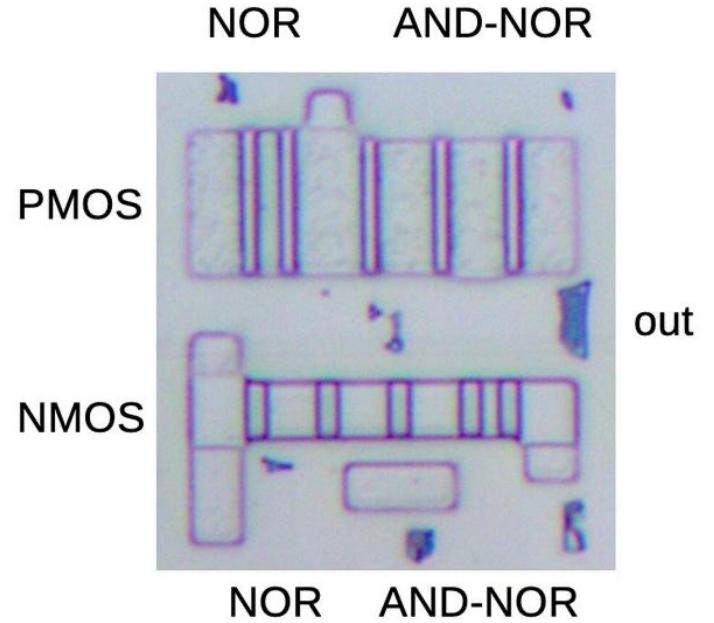


COE COLLEGE®

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 focus base on two:

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

Input		Output
A	B	$A \text{ NAND } B$
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.



COE COLLEGE®

Step 1: data manipulation

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

K-map grouping rules

- No zeros allowed.
- No diagonals.
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping allowed.
- Wrap around allowed.
- Fewest number of groups possible.

$$\begin{aligned} & (\text{A AND B}) \text{ OR } (\text{NOT C AND D}) \\ & (\text{A} \times \text{B}) + (\text{!C} \times \text{D}) \end{aligned}$$

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

The Karnaugh map shows the function $(\text{A} \times \text{B}) + (\text{!C} \times \text{D})$. The terms are represented by 1s in the cells (00, 01), (01, 01), (11, 00), and (10, 01). A red oval highlights the group for $\text{!C} \times \text{D}$, which covers cells (01, 01), (01, 11), (11, 01), and (10, 01). Another red oval highlights the group for $\text{A} \times \text{B}$, which covers cells (00, 01) and (01, 01).



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.

AB CD	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.

$$(!A \times !C \times D) + (A \times !D)$$

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



COE COLLEGE®

Step 1: data manipulation

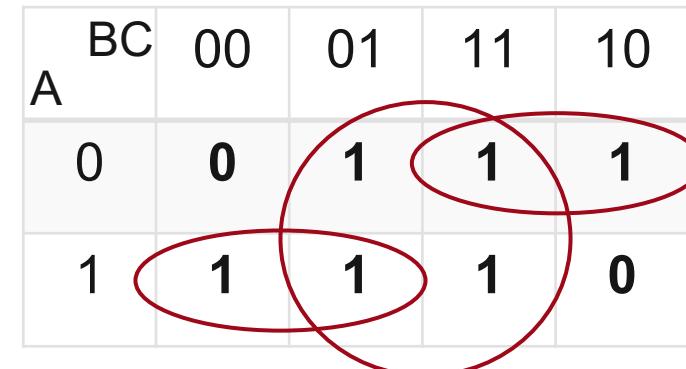
- Karnaugh map (K-map) (2/2), 1 more in class practice

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.

$$(A \times !B)+ (!A \times B) + C$$

		BC	00	01	11	10
		A	0	1	1	1
0	0	0	1	1	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:

