CPE 462
VHDL: Simulation and Synthesis

Topic #02 - a) Review of Combinational Logic

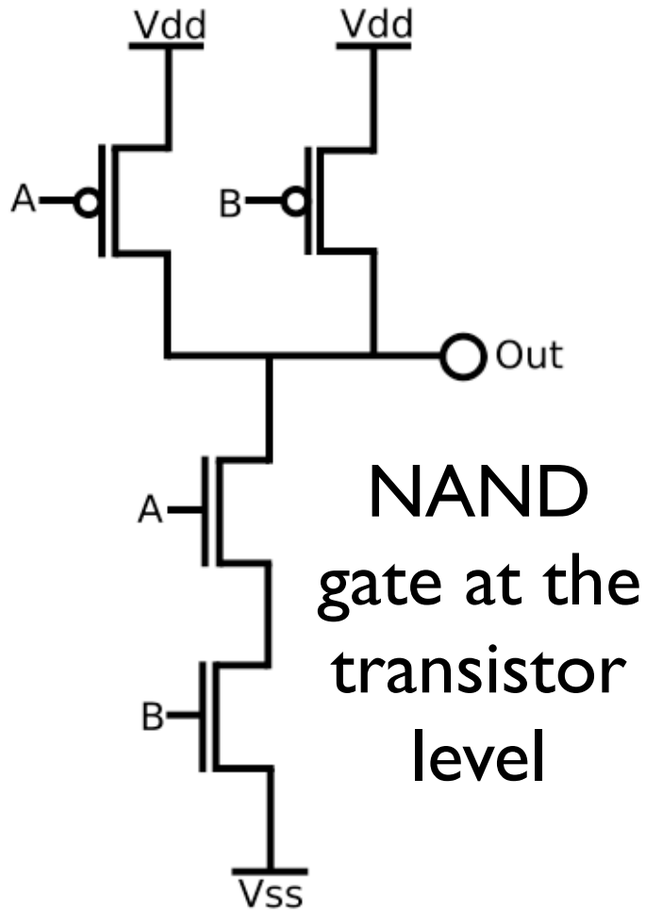
Some information on the following slides was taken from:
http://hyperphysics.phy-astr.gsu.edu

Basic Gates

1) These are the basic gates we will be dealing with:

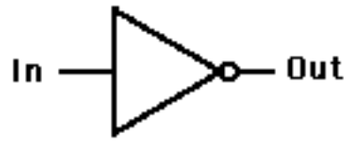
- AND, NAND
- BUF, NOT
- OR, NOR
- XOR, XNOR

2) While you should know how each logic gate is implemented at the transistor level, in this course we are mainly working at the register-transfer level (RTL).



NAND gate at the transistor level

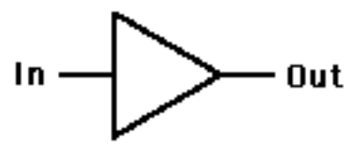
AND, OR, NOT and BUF



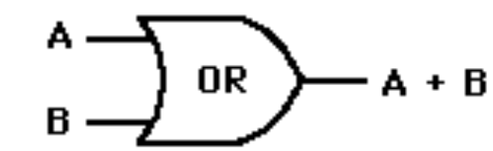
In	Out
0	1
1	0

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

The AND operation will be signified by AB or A•B. Other common mathematical notations for it are $A \wedge B$ and $A \cap B$, called the intersection of A and B.



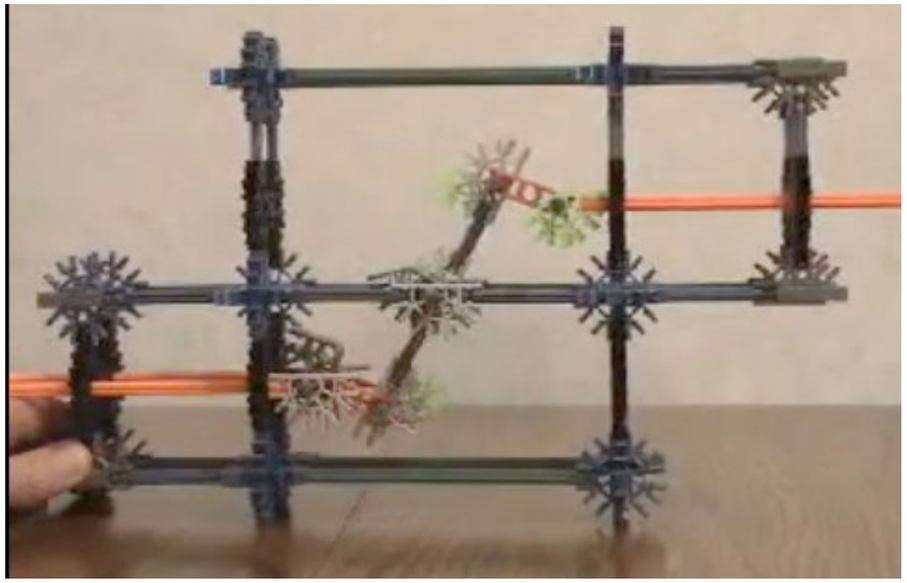
In	Out
0	0
1	1

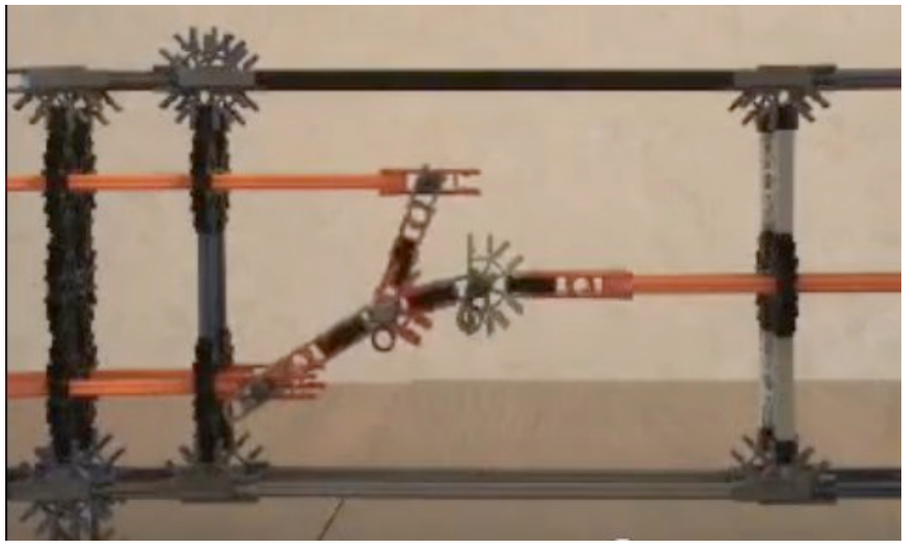


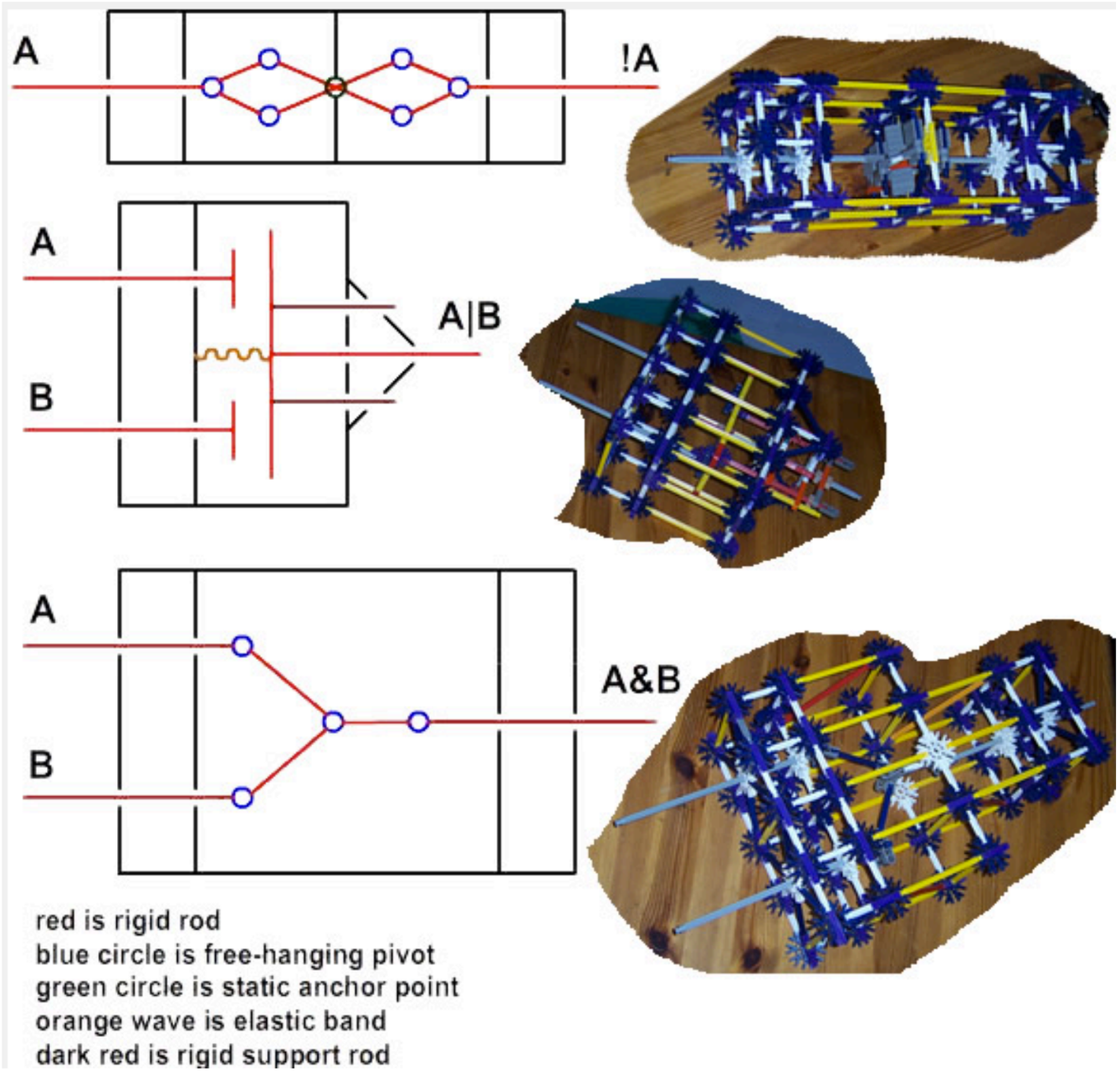
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

The OR operation will be signified by A + B. Other common mathematical notations for it are $A \vee B$ and $A \cup B$, called the union of A and B.

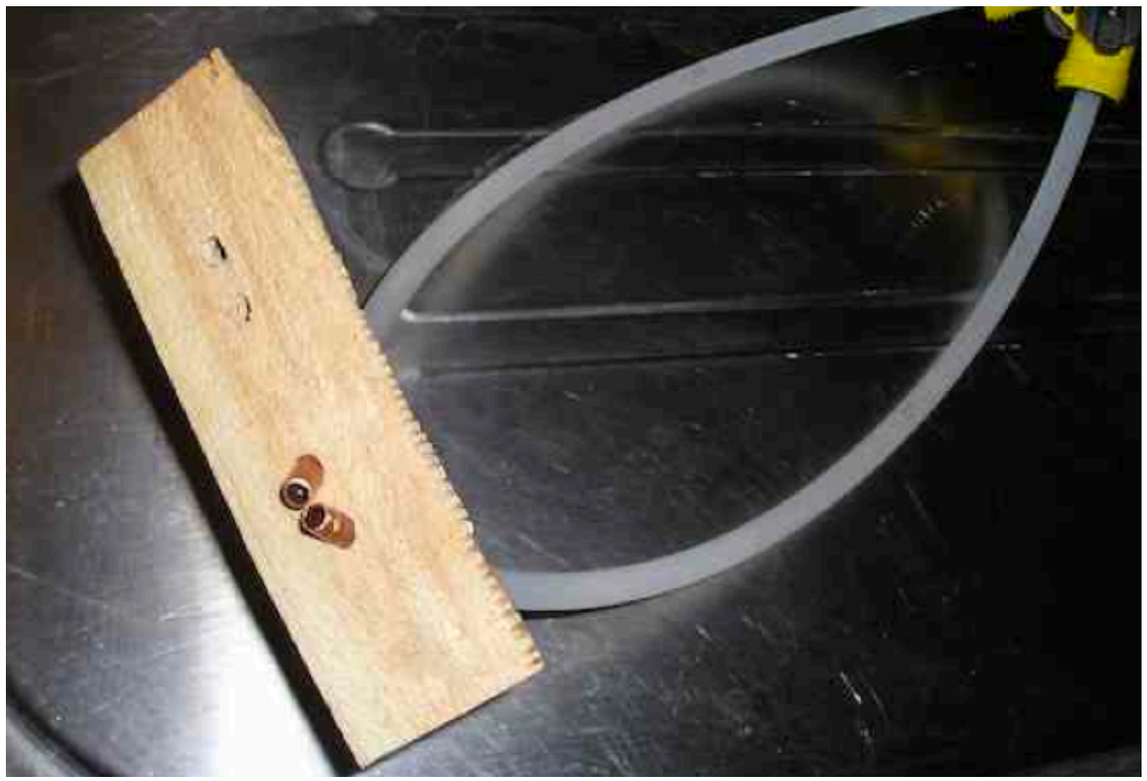
K'nex implementations of logic gates







Water gates

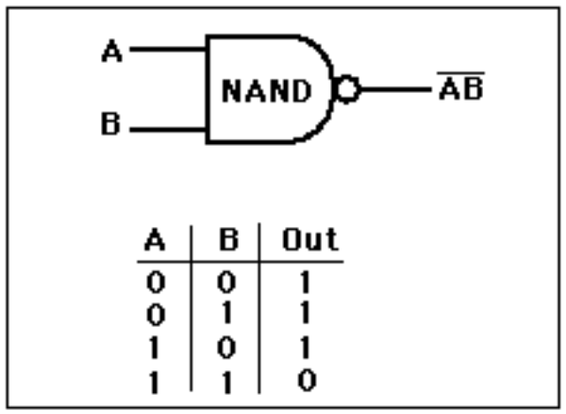


AND gate

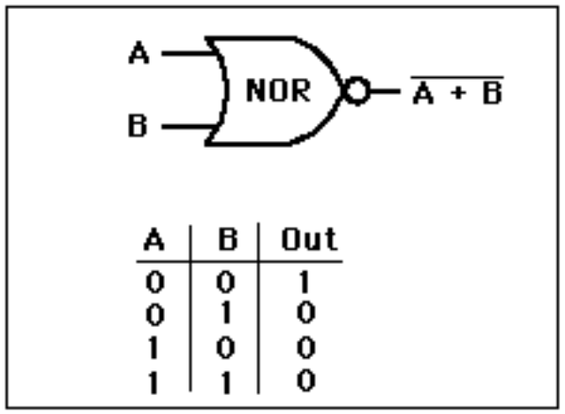
NAND and NOR

The NAND gate and the NOR gate are universal.

This means any combinations of them can be used to accomplish any of the basic logic operations.



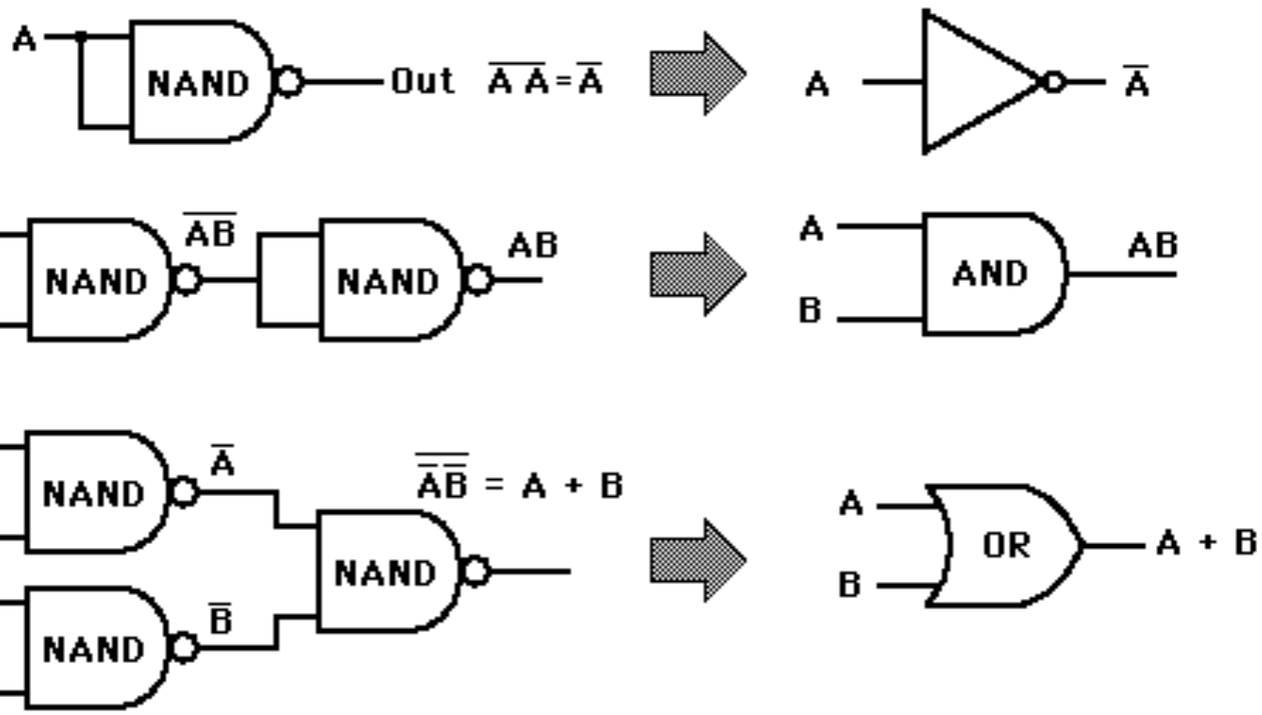
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

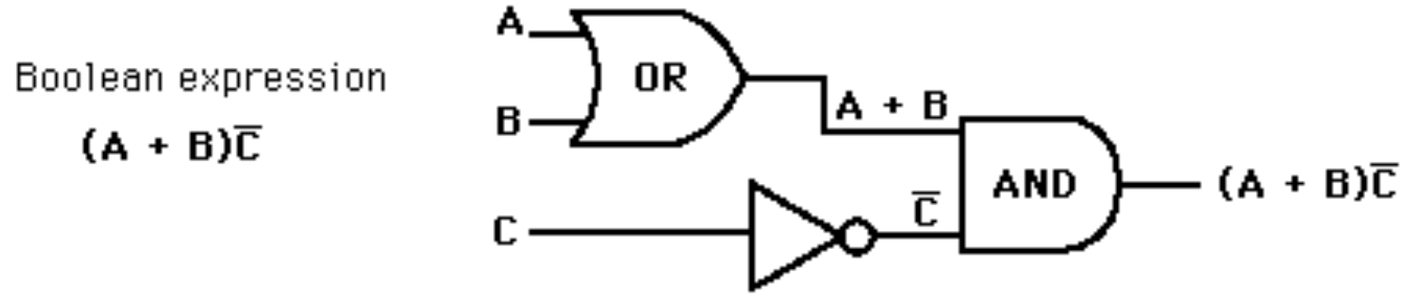
NAND gate operations

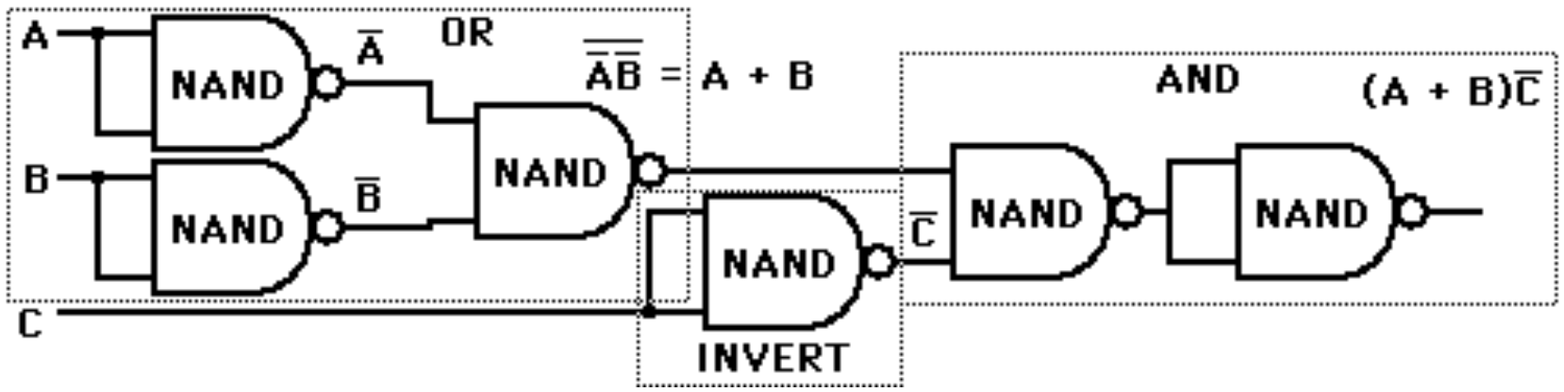
With just NAND gates we can come up with all other basic gates...



A more complex circuit with just NAND gates

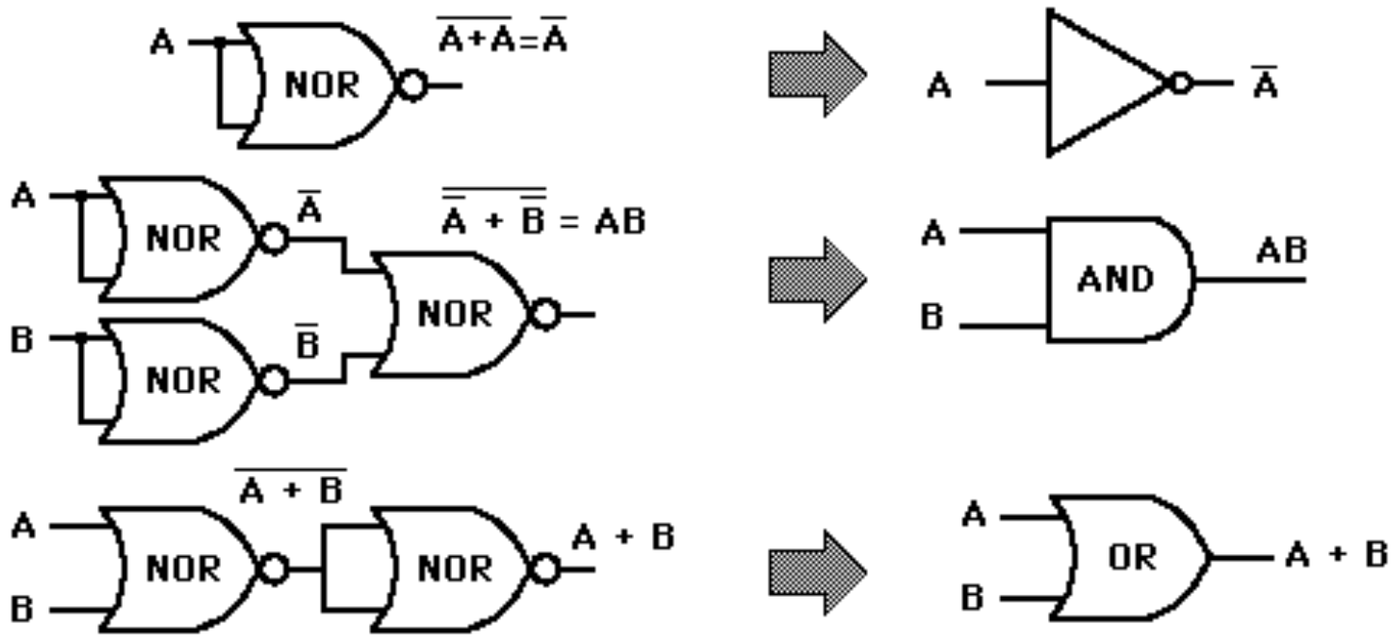
Suppose you want a high output when either A or B is high but C is low.



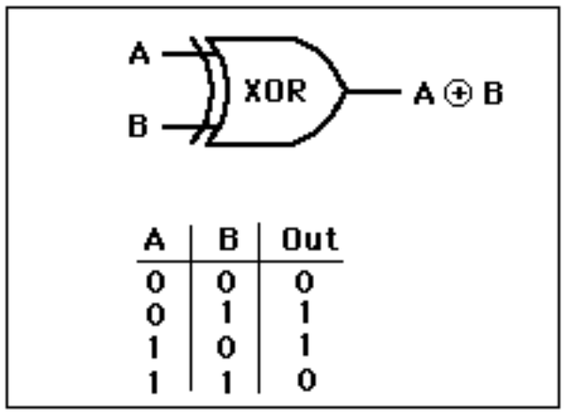


NOR gate operations

Since NOR gates are also universal...



XOR gates

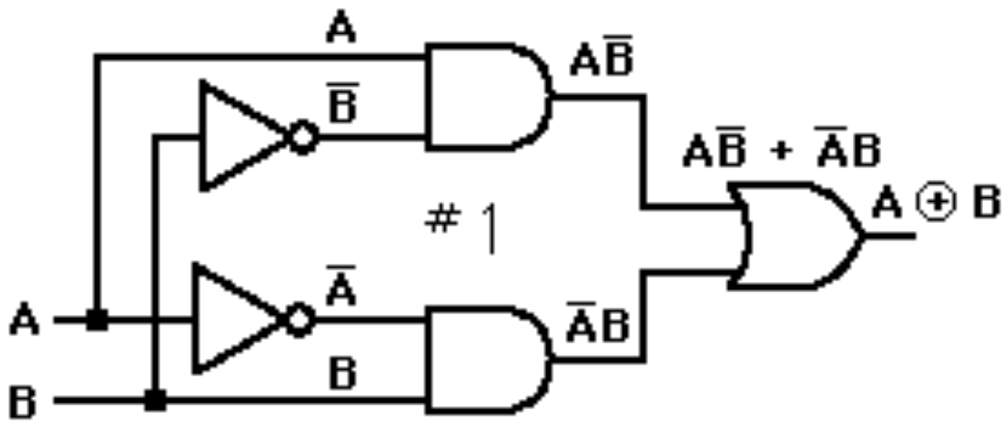


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

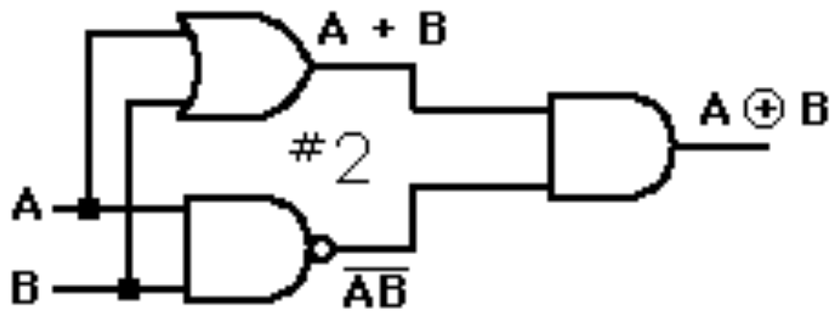
The output is high when either of inputs A or B is high, but not if both A and B are high.

Logically, the XOR operation can be seen as either of the following operations:

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

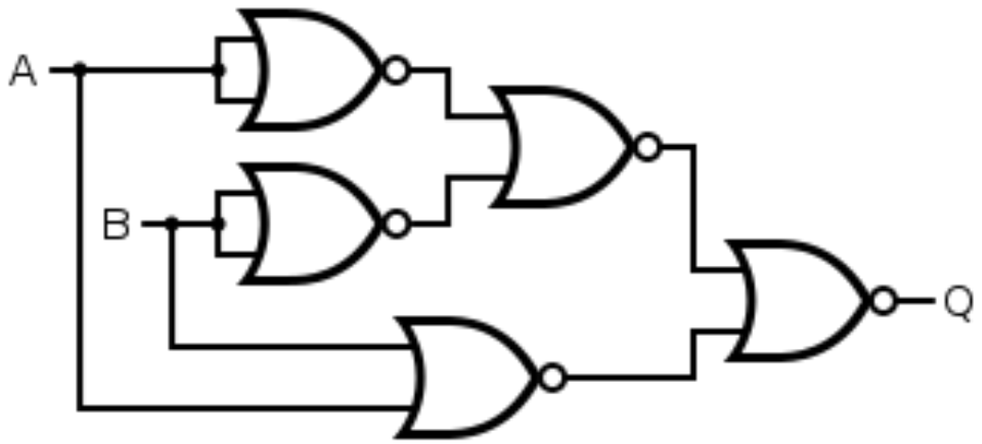


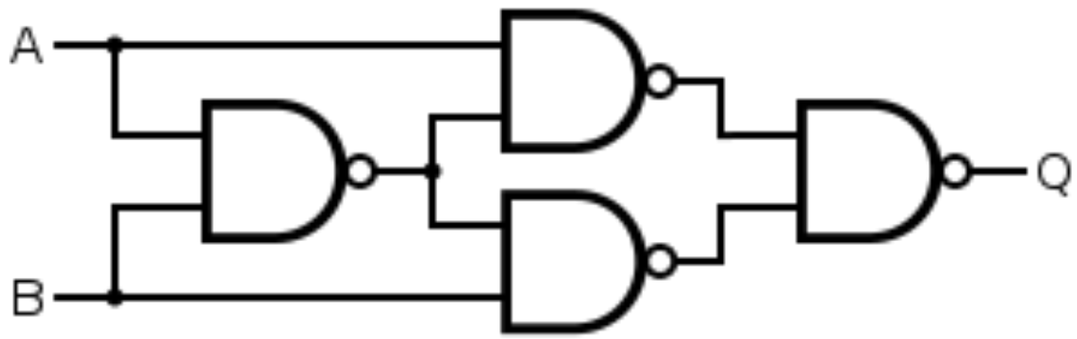
$A \oplus B = (A + B)(\overline{AB})$



XOR gate implementations

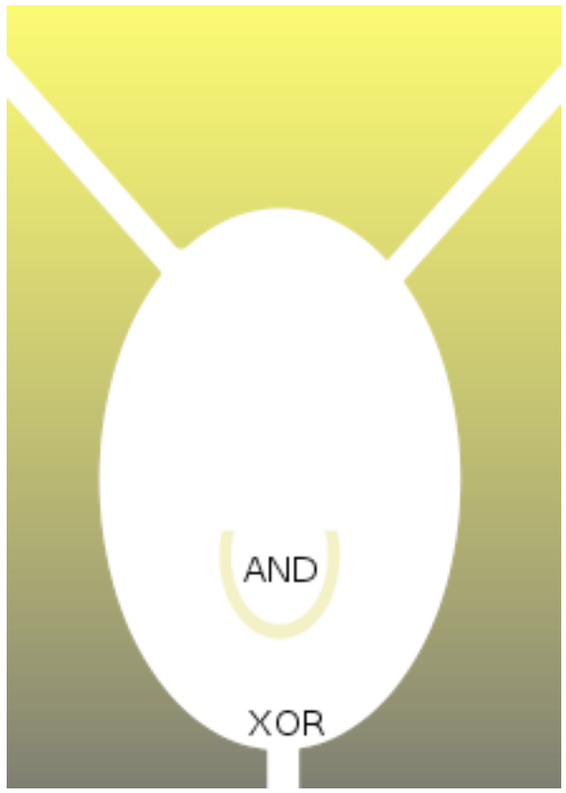
Implementing XOR gates with NOR and NAND universal gates...





Fluidics

Fluidics is the use of a fluid to perform analog or digital operations similar to those performed with electronics.



The two gates AND and XOR in one module. The bucket in the center collects the AND output, and the output at the bottom is A XOR B.

http://www.blikstein.com/paulo/projects/project_water.html

Review of Boolean Logic Rules

NOT Operations (')

```
0' = 1        A''  = A
       1' = 0
```

AND Operations (.)

```
0 . 0 = 0     A . 0  = 0
1 . 0 = 0     A . 1  = A
0 . 1 = 0     A . A  = A
1 . 1 = 1     A . A' = 0
```

OR Operations (+)

```
0 + 0 = 0     A + 0  = A
1 + 0 = 1     A + 1  = 1
0 + 1 = 1     A + A  = A
1 + 1 = 1     A + A' = 1
```

Associative Law

```
(A.B) . C = A . (B.C) = A.B.C
(A+B) + C = A + (B+C) = A+B+C
```

Distributive Law

```
A . (B+C) = (A.B) + (A.C)
A + (B.C) = (A+B) . (A+C)
```

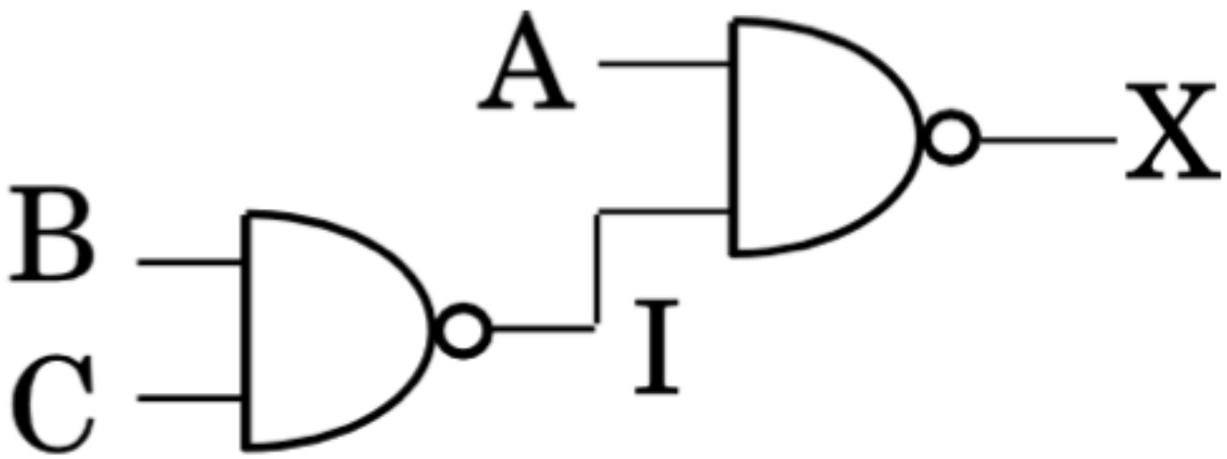
Commutative Law

```
A.B = B.A
A+B = B+A
```

Precedence

```
AB = A.B
A.B+C = (A.B) + C
A+B.C = A + (B.C)
```

What is this circuit output?

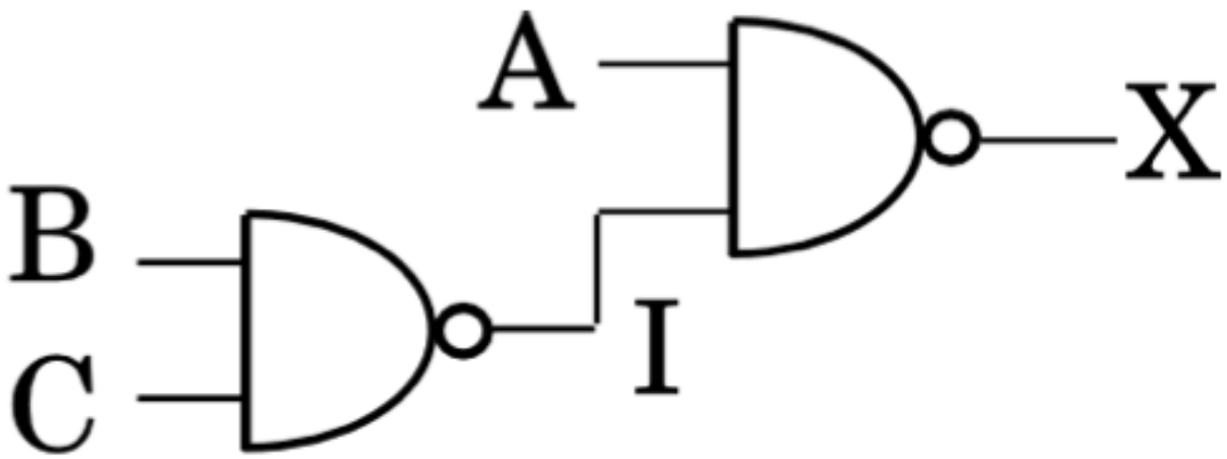


De'Morgan Rules:

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$

$$\overline{a + b} = \overline{a} \cdot \overline{b}$$

What is this circuit output?



De'Morgan Rules:

$$\overline{a \cdot b} = \overline{a} + \overline{b}$$

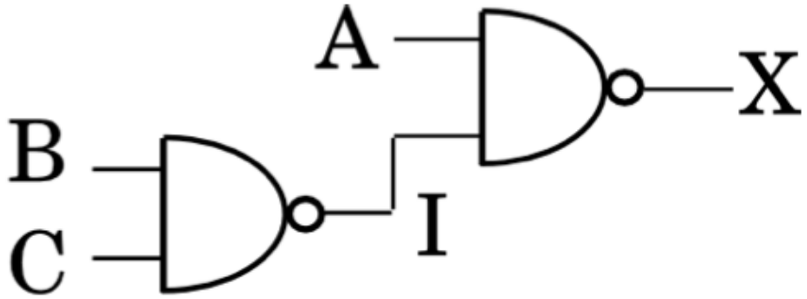
$$\overline{a + b} = \overline{a} \cdot \overline{b}$$

$$i = \overline{b \cdot c}$$

$$x = (\overline{a \cdot i})$$

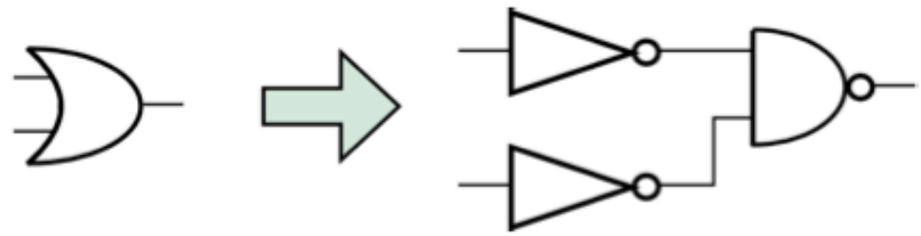
$$x = \overline{a \cdot (\overline{b \cdot c})} = \overline{a} + (\overline{\overline{b \cdot c}}) = \overline{a} + (b \cdot c)$$

Alternative is to create a truth table

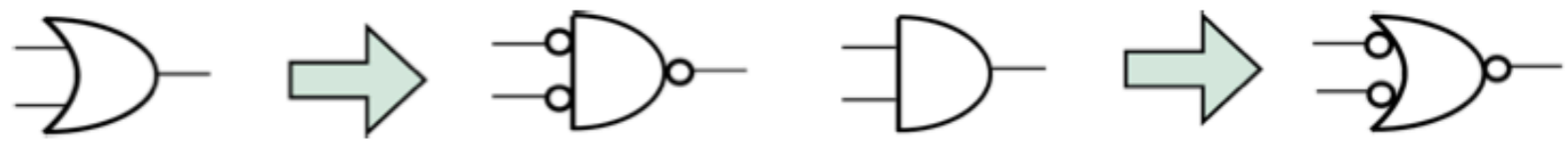


A	B	C	I = $(B \cdot C)'$	X = $(A \cdot I)'$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

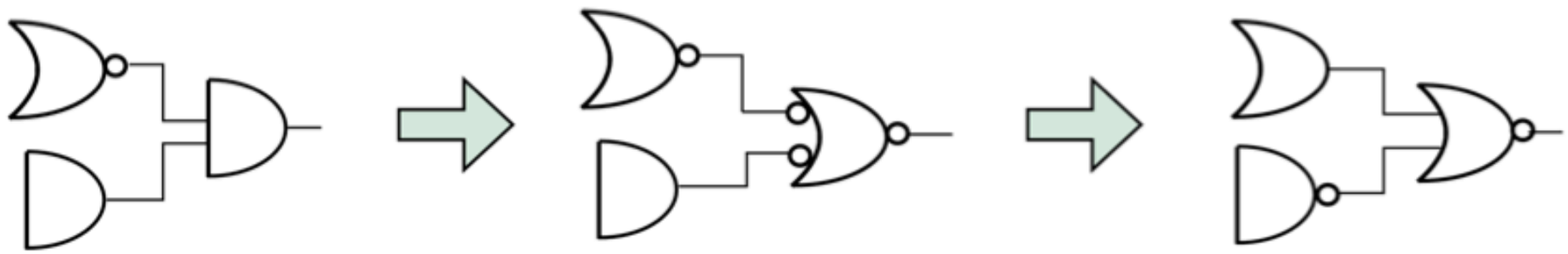
Graphical Representation of De'Morgan Rules



Often it is convenient to use just the circle for inversion rather than the whole inverter gate symbol. Doing this we represent de Morgan's theorem graphically as follows:

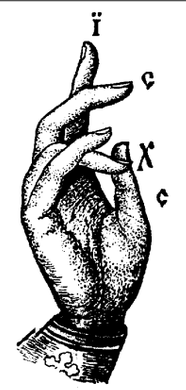


And we can now perform the same manipulations that we did using Boolean algebra directly to circuits as we design them:

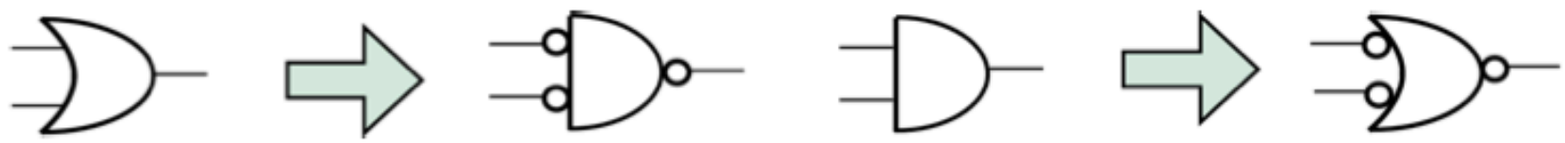


Apply de Morgan's theorem

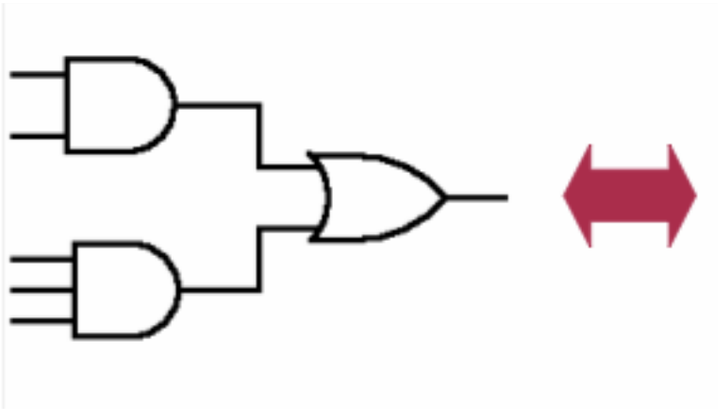
Re-group and cancel the inverters



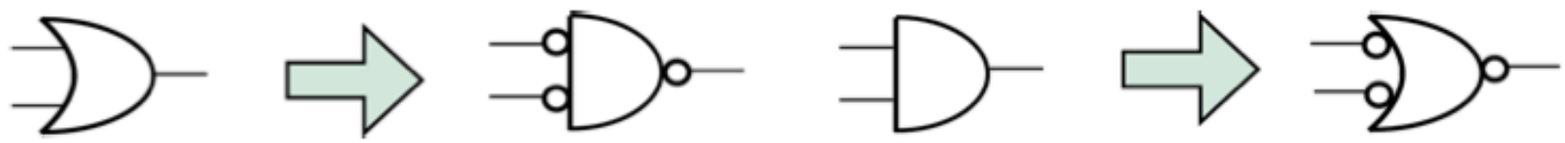
Circuit with just NAND gates



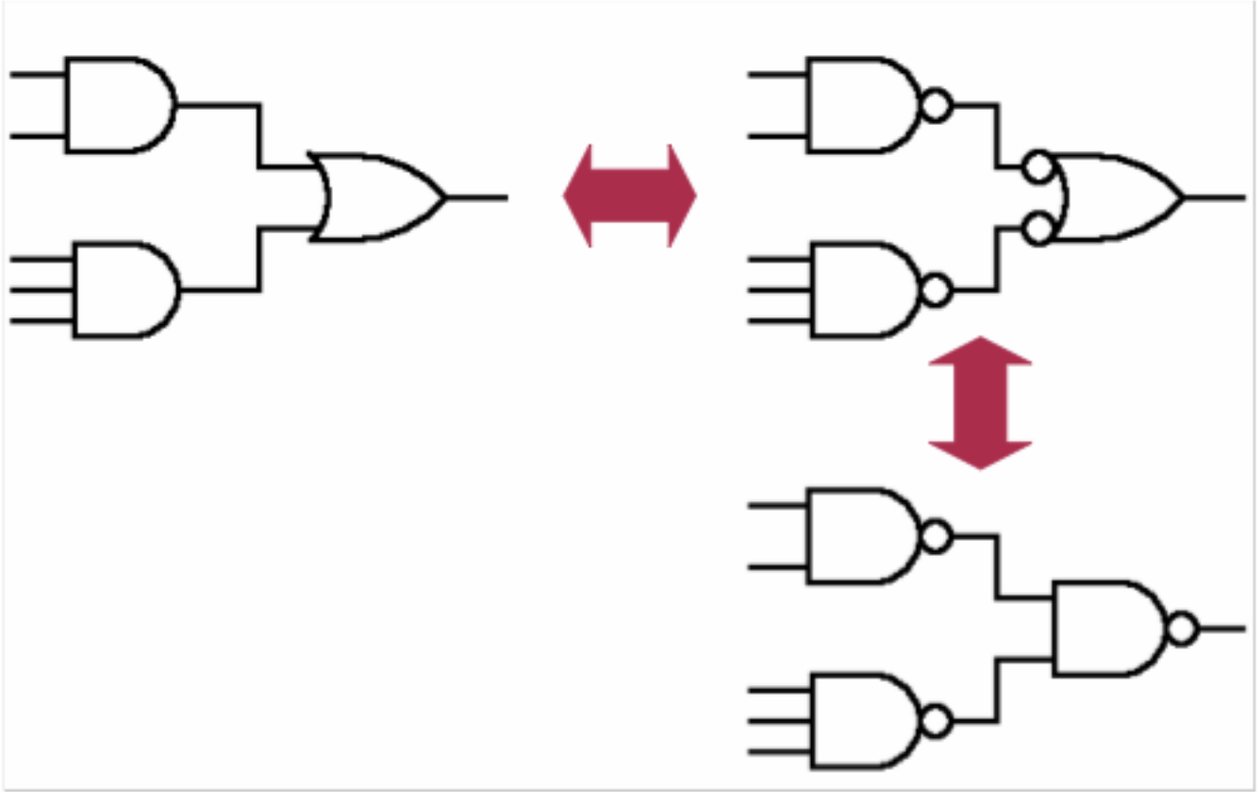
Using graphical representations of De'Morgan rules we can convert circuits so they are implemented with just NAND gates.



Circuit with just NAND gates



Using graphical representations of De'Morgan rules we can convert circuits so they are implemented with just NAND gates.



Review of binary addition: half-adder

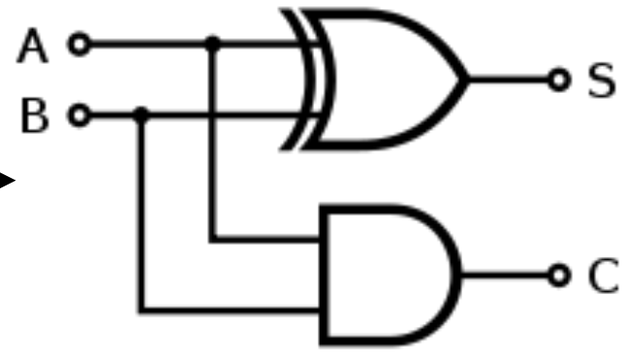
0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

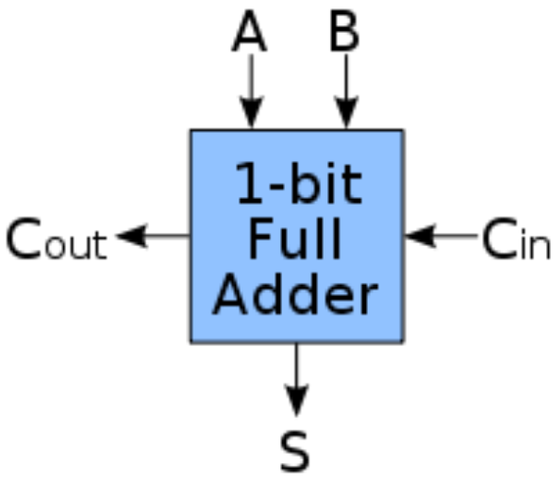
1 + 1 = 10

INPUTS		OUTPUTS	
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

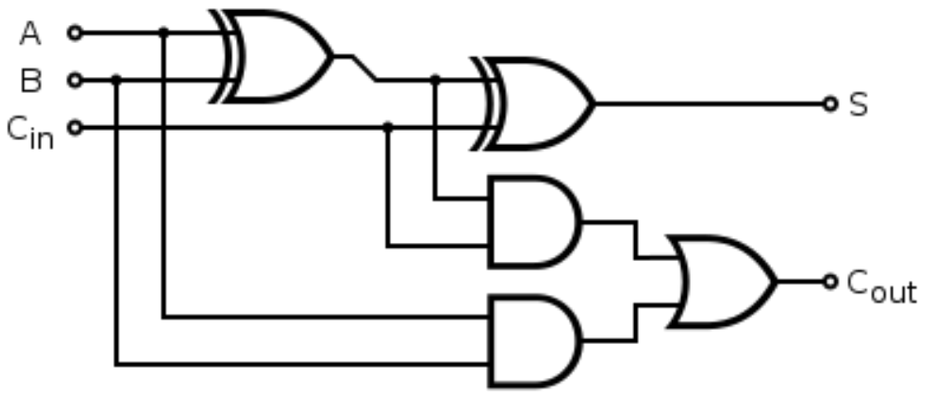


The Carry output is an AND function, and the Sum is an XOR.

Review of binary addition: full-adder



Inputs			Outputs	
A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

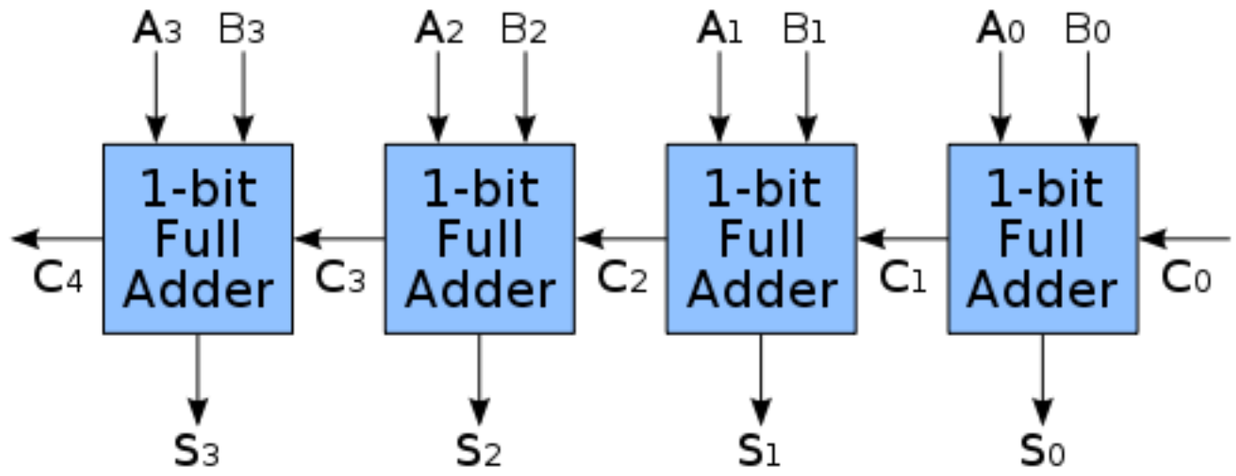


$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$$

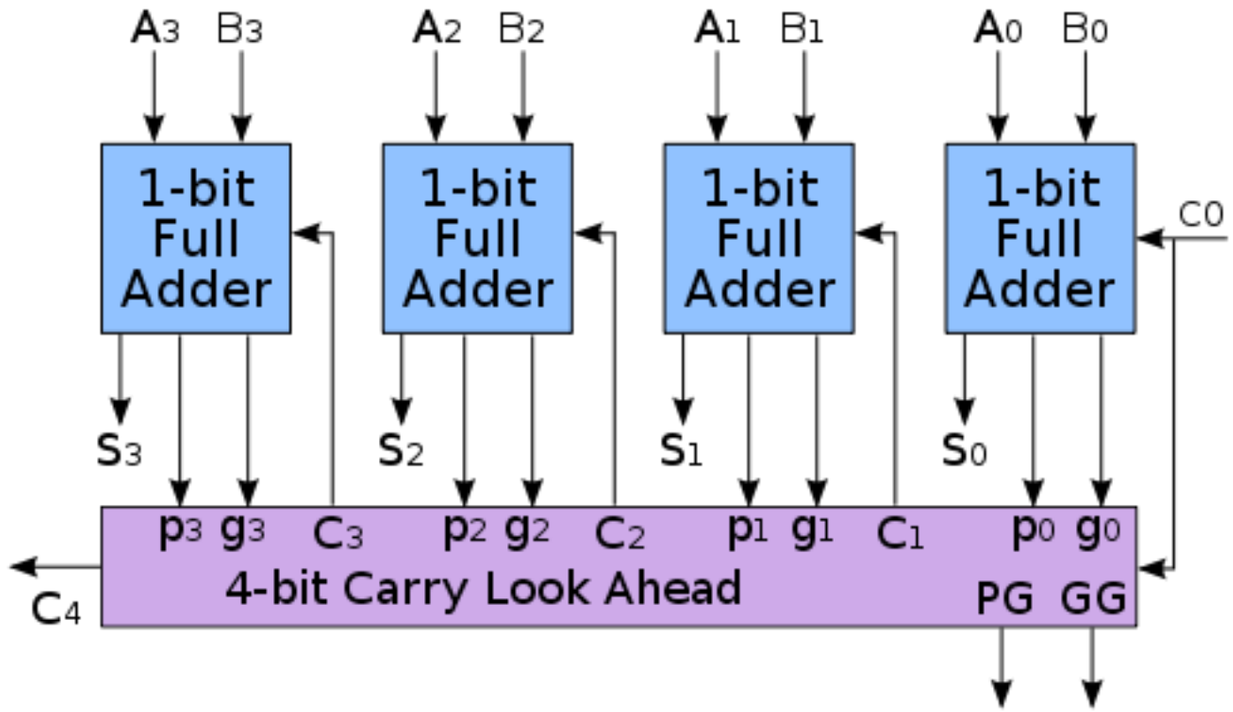
$$S = A \oplus B \oplus C_{in}$$

4-Bit adders

4-Bit Adder

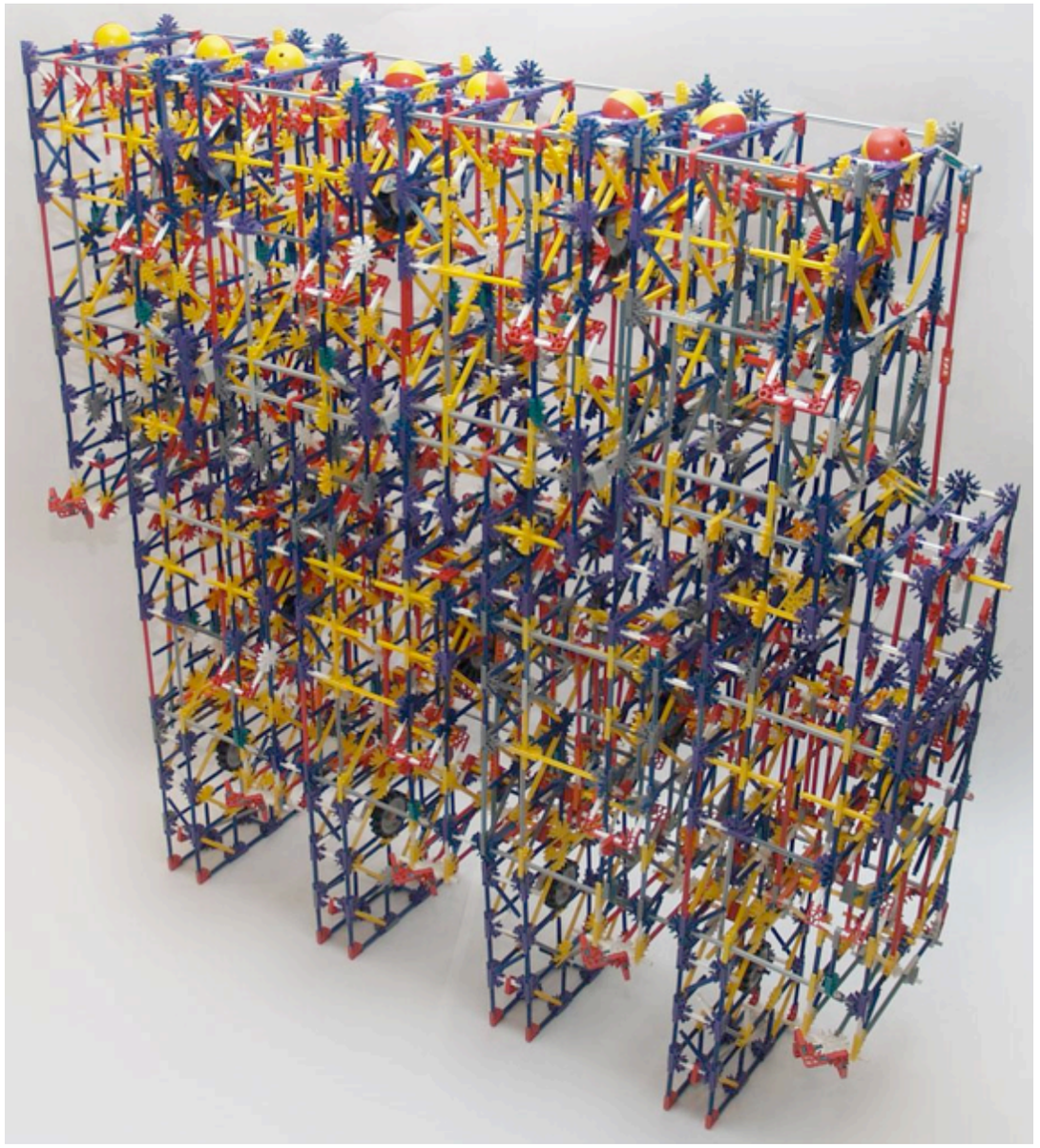


4-Bit adder with carry lookahead

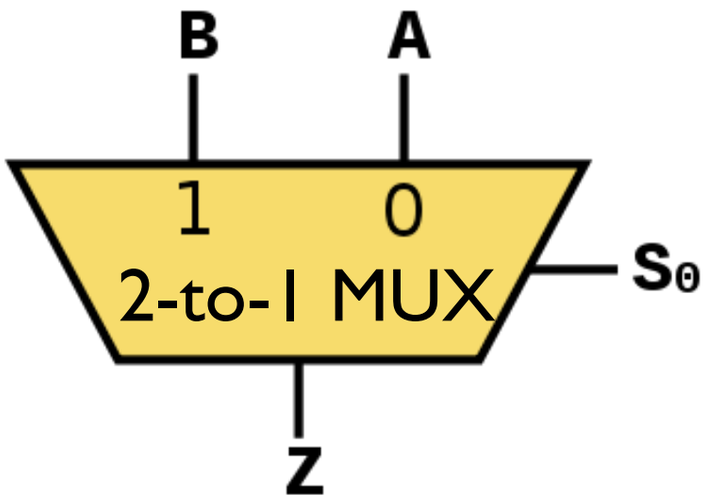


Physical implementation of a 4-Bit adder

http://knexcomputer.blogspot.com/2006_12_01_archive.html



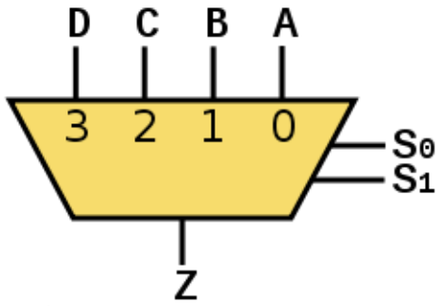
Multiplexer (MUX)



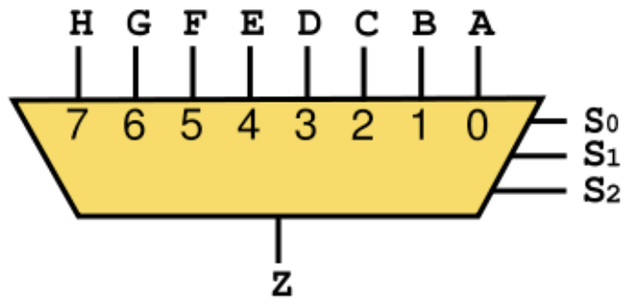
$$Z = (A \cdot \overline{S}) + (B \cdot S)$$

S	A	B	Z
0	1	1	1
	1	0	1
	0	1	0
	0	0	0
1	1	1	1
	1	0	0
	0	1	1
	0	0	0

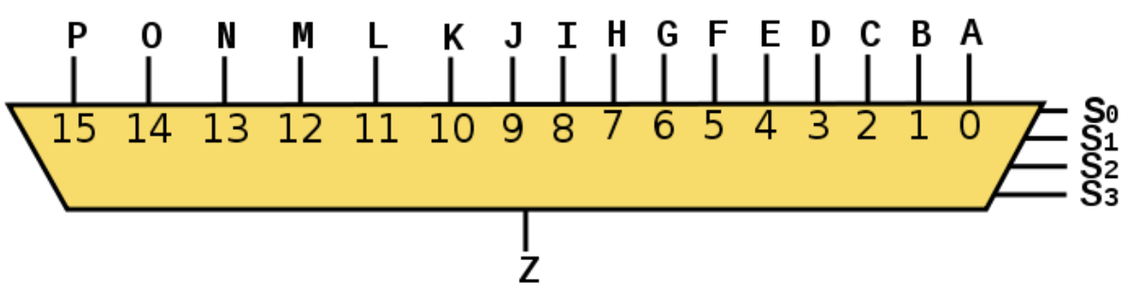
You have n input signals (A and B in this example). With the select bit(s), you determine which input will propagate to the single output.



4-to-1 MUX

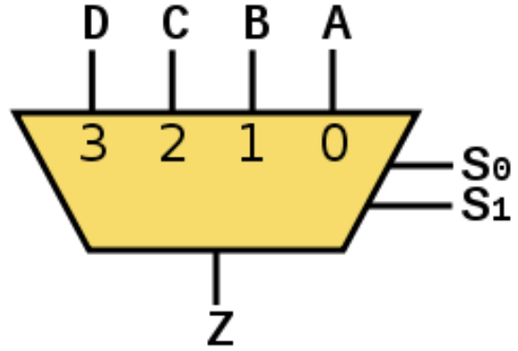


8-to-1 MUX

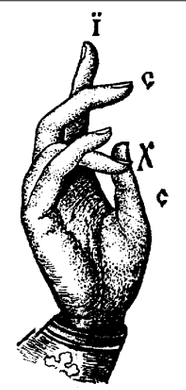


16-to-1 MUX

4-to-1 MUX

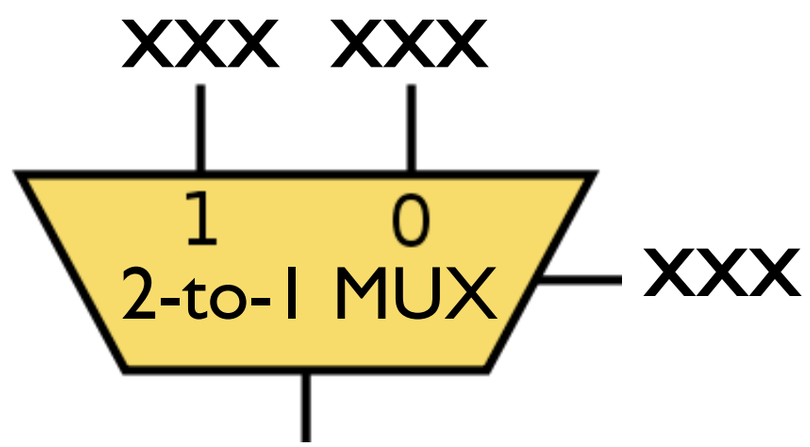


$$F = (A \cdot \overline{S_0} \cdot \overline{S_1}) + (B \cdot S_0 \cdot \overline{S_1}) + (C \cdot \overline{S_0} \cdot S_1) + (D \cdot S_0 \cdot S_1)$$



Lets play some MUX games

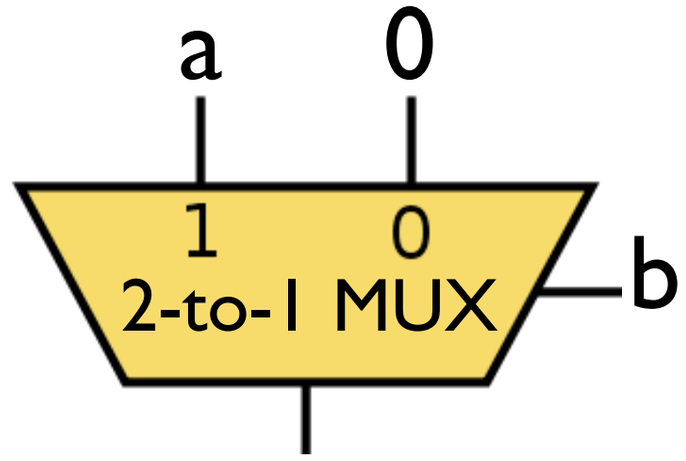
With any number of 2-to-1 MUX gates, implement an AND gate with the function OUT = A . B



OUT = xxx

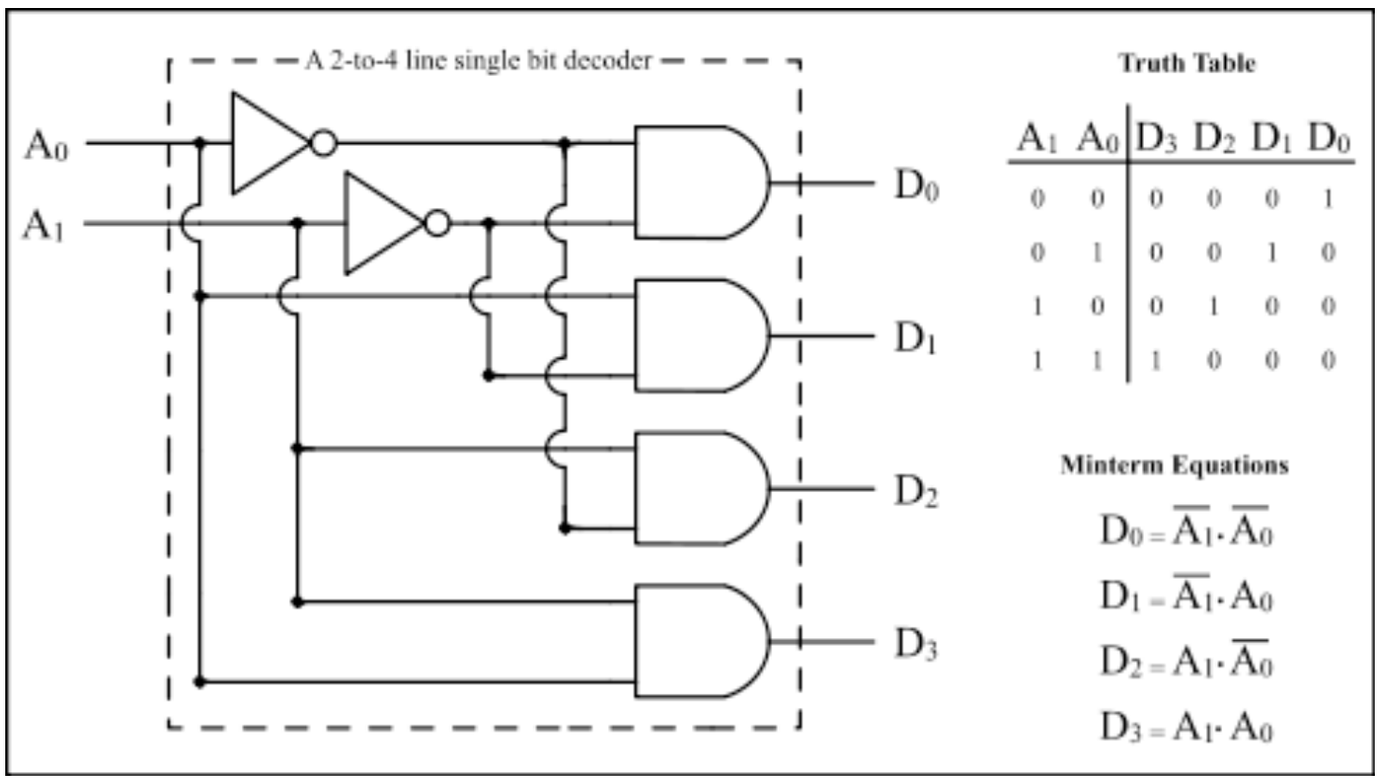
Lets play some MUX games

With any number of 2-to-1 MUX gates, implement an AND gate with the function OUT = A . B



$$OUT = a.b + 0.\overline{b}$$

Single-bit decoder



Idea: In the simplest case, you are trying to enable a **single** output, with a combination of input signals.

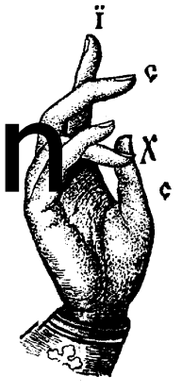
2 inputs can enable 4 different outputs

...

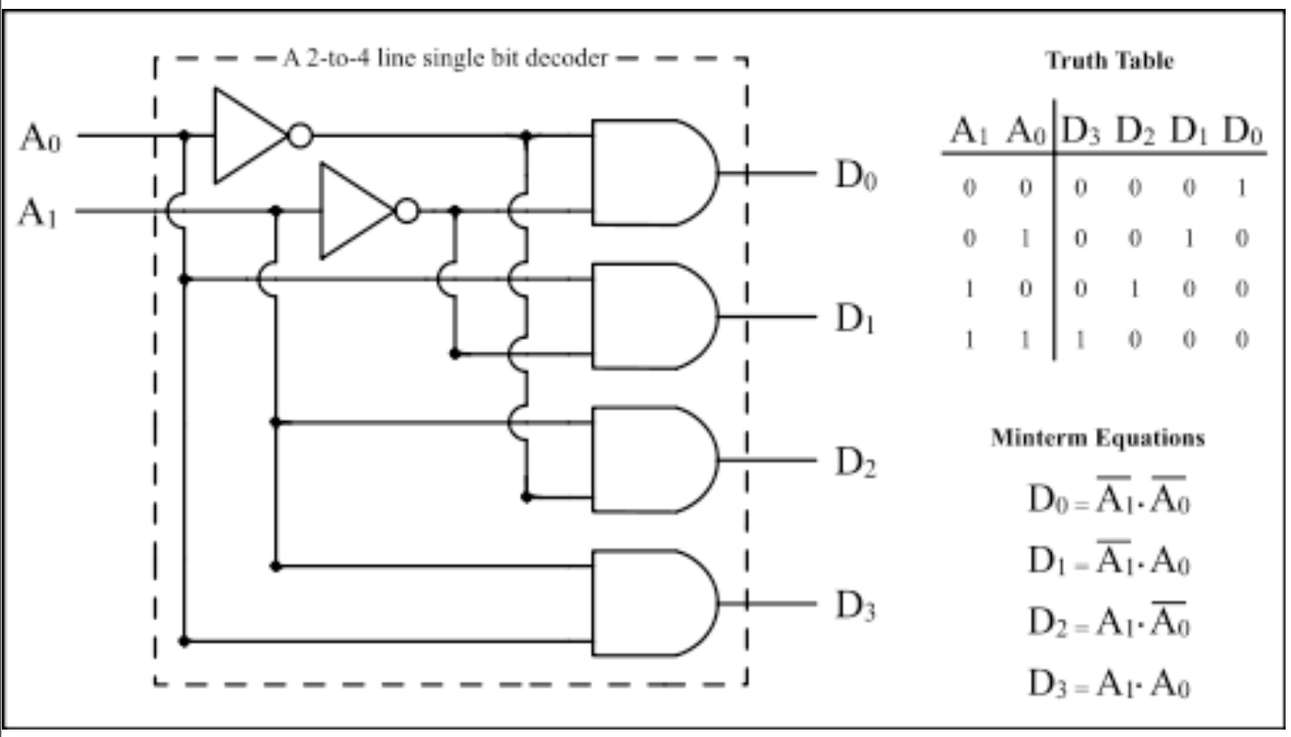
n inputs can enable 2^n different outputs

Why is this useful? Well, imagine you have an 64 LEDs and you would like to control them individually...

What if I want to turn ON more than one single output?



For example, how can I set D0 and D3 to HIGH?



Truth Table

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Minterm Equations

$$D_0 = \overline{A_1} \cdot \overline{A_0}$$

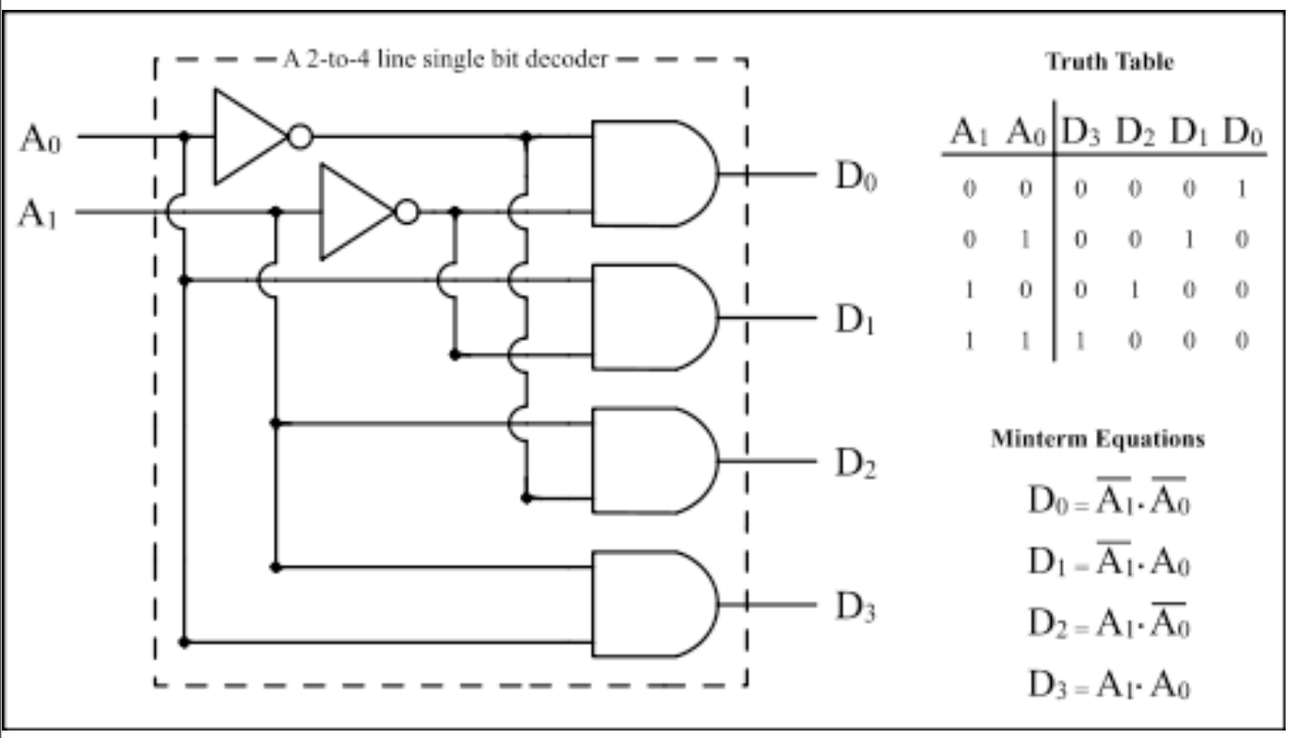
$$D_1 = \overline{A_1} \cdot A_0$$

$$D_2 = A_1 \cdot \overline{A_0}$$

$$D_3 = A_1 \cdot A_0$$

What if I want to turn ON more than one single output?

For example, how can I set D0 and D3 to HIGH?

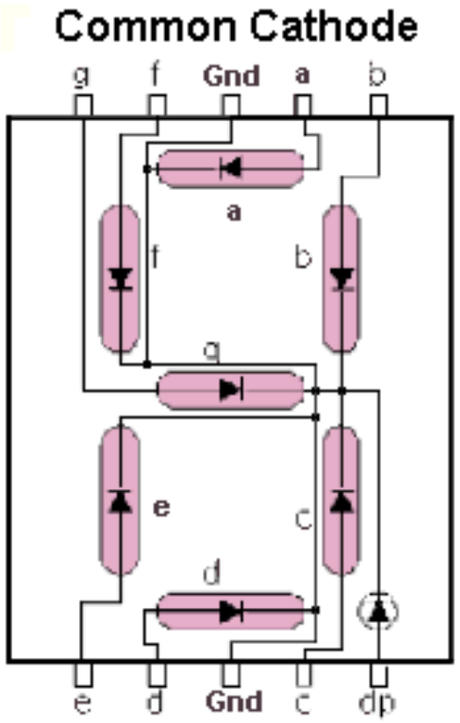


Use time division multiplexing!

1. Set both A0 and A1 to LOW,
2. Set both A0 and A1 to HIGH
3. Cycle through 1 and 2 very very fast!

If each output was a connected to LED, you wouldn't be able to see the difference!

Slightly more complex decoder BCD-to-7-segment (multi-output)



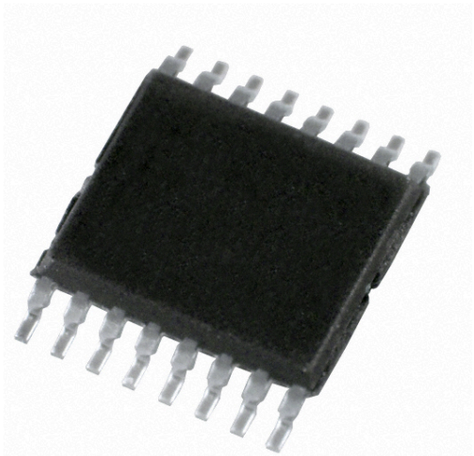
$$a = A + C + B \bullet D + \overline{B} \bullet \overline{D}$$

$$b = \overline{B} + C \bullet D + \overline{C} \bullet \overline{D}$$

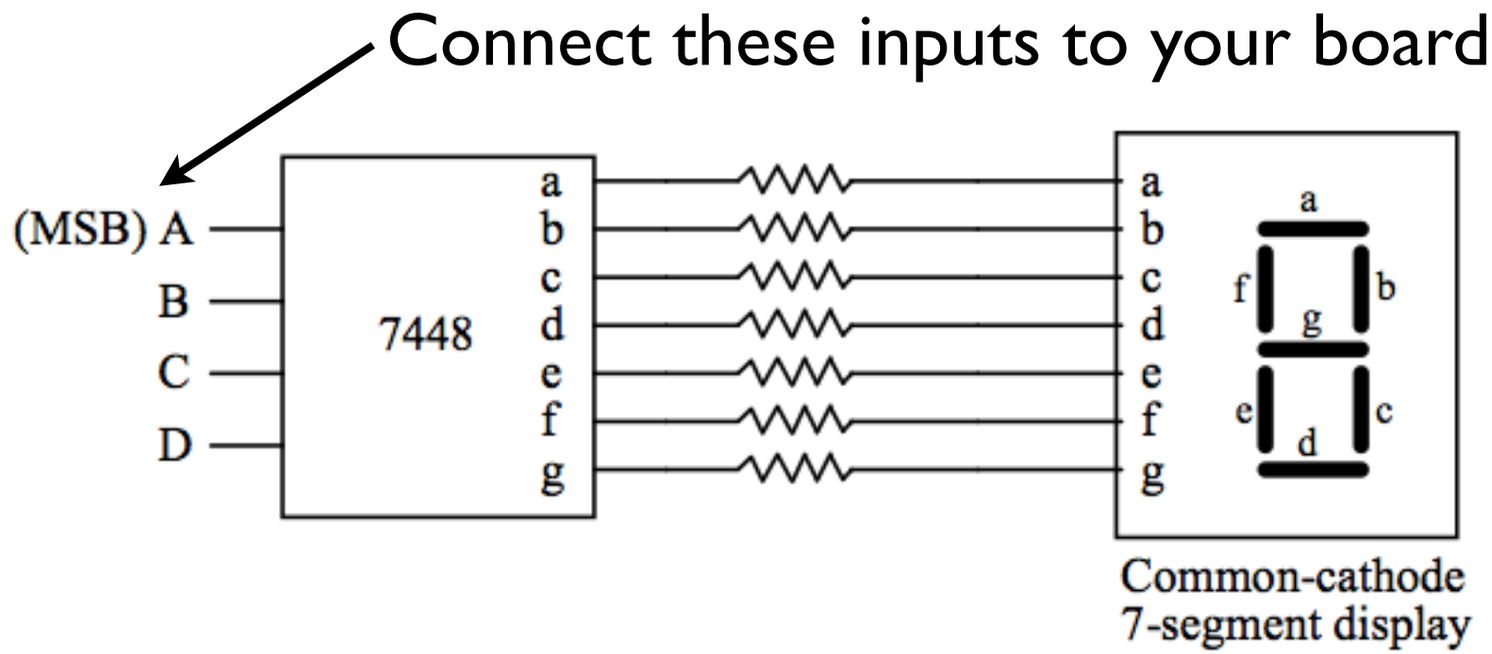
We need to get a boolean logic expression for all outputs... above we find a and b.

	Inputs				Outputs						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
	1	0	1	0	x	x	x	x	x	x	x
	1	0	1	1	x	x	x	x	x	x	x
	1	1	0	0	x	x	x	x	x	x	x
	1	1	0	1	x	x	x	x	x	x	x
	1	1	1	0	x	x	x	x	x	x	x
	1	1	1	1	x	x	x	x	x	x	x

Decoders are so useful that many have been chipped...



IC Decoder
#7448
($0.15)

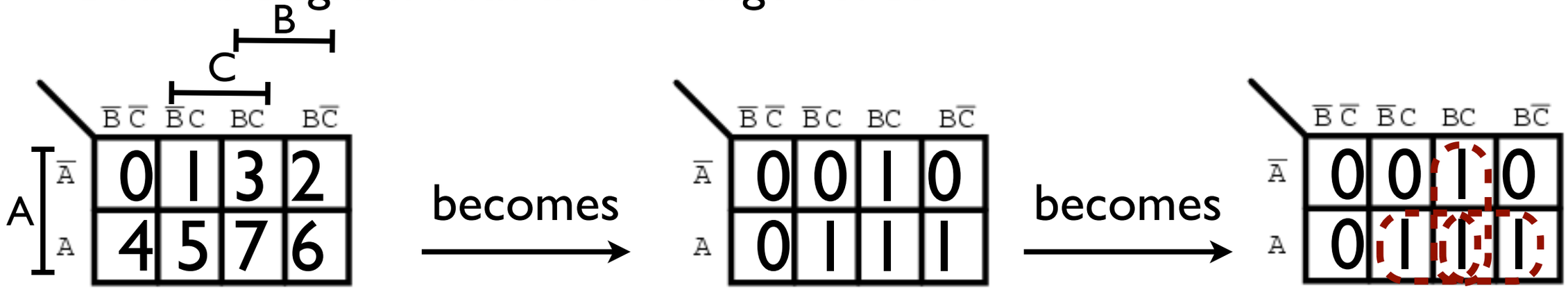


Karnaugh map: method to simplify boolean logic expressions

- A Karnaugh Map is a grid-like representation of a truth table
- Here is a sample truth table with 3 inputs(a,b,c) and one output (v)
- f(a,b,c) = Sum(3,5,6,7)

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

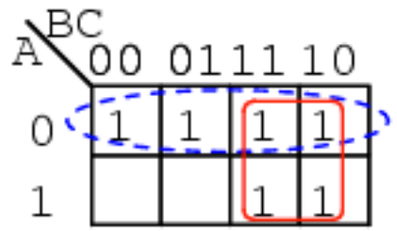
- Draw a diagram with the "strange" order



v = BC + AB + AC

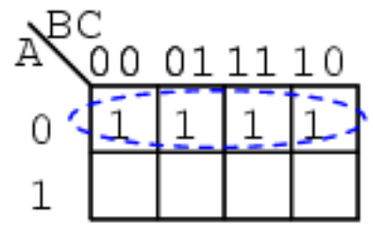
Karnaugh map review

Out= $\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C+\overline{A}BC+\overline{A}B\,\overline{C}+ABC+AB\,\overline{C}$



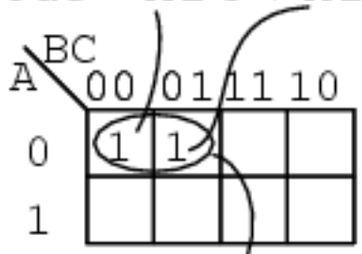
Out= $\overline{A} + B$

Out= $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}\,B\,\overline{C}$



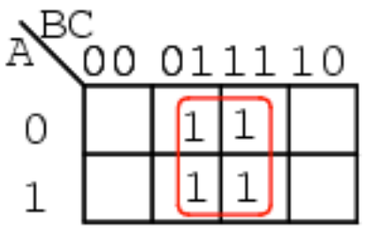
Out= $\overline{A}$

Out= $\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C$



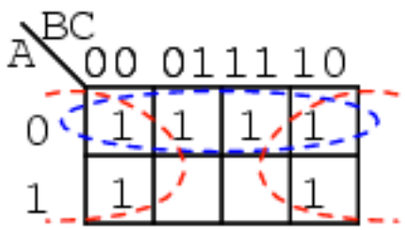
Out= $\overline{A}\,\overline{B}$

Out= $\overline{A}\,\overline{B}C + \overline{A}\,B\,C + A\,\overline{B}C + ABC$



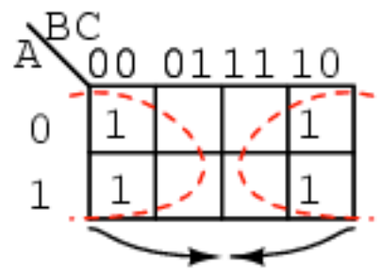
Out= C

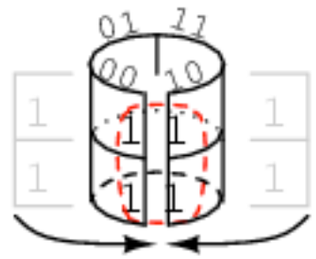
Out= $\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}C+\overline{A}BC+\overline{A}B\,\overline{C}+A\,\overline{B}\,\overline{C}+AB\,\overline{C}$



Out= $\overline{A} + \overline{C}$

Out= $\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}B\,\overline{C} + AB\,\overline{C}$

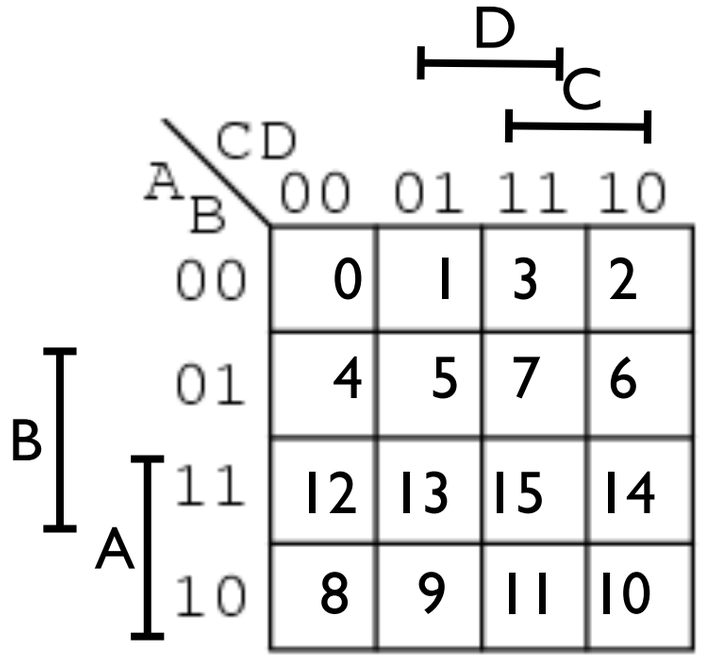




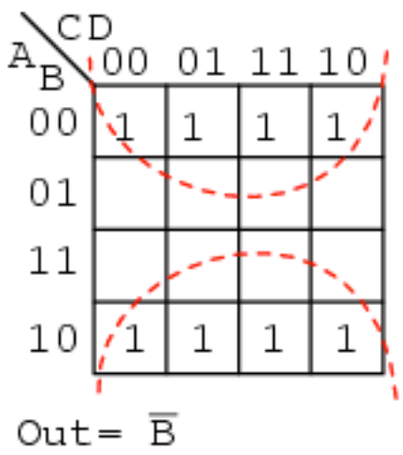
Out= $\overline{C}$

Karnaugh map with 4 variables

Same thing as before... but with more columns and rows.

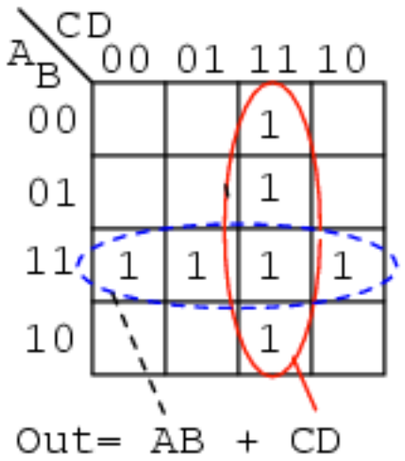


Out= $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$



Out= $\overline{B}$

Out= $\overline{A}\overline{B}CD + \overline{A}BCD + ABCD + A\overline{B}CD + AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D}$



Out= AB + CD